

**NEOSTATS**

Placement Assessment

**Role: Cloud Data Engineer**

**Sales Data Processing for Retail Optimization**

by

**Justin Ziv Patil (2347122)**

Submission Date: 19/11/2024

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **SL .No** | **Title** | **Page Number** |
| 1 | **Design Document**   * High-Level Design document * The choice of data storage(ADLS) * Architectural diagram illustrating the data flow | 3 - 5 |
| 2 | **Code Implementation**:   * Code/Scripts to implement the data pipeline * Code Explanation * Handle PII data | 4 |
| 3 | **Dataset Documentation**:   * Describe the structure of the sales dataset (columns, data types). * Explain any data pre processing steps performed. | 5 |
| 4 | **Azure Work Flow Screen Shots** | 6 |
| 5 | **Power BI Dashboard** | 14 |

**High-Level Design Document**

**Solution Overview**

The solution addresses the need for **sales data processing and analysis** for XYZ Retail Inc. by creating a robust and scalable data pipeline. The system ingests raw sales data from various sources, processes it for insights, and visualizes the results to optimize retail operations.

**System Architecture**

1. **Data Ingestion**
   * **Source**: Sales data in CSV format from online platforms, physical stores, and mobile apps.
   * **Tool**: **Azure Data Factory (ADF)** is used for extracting and loading raw sales data into the storage layer.
   * **Target**: Data is stored in **Azure Data Lake Storage Gen2 (ADLS)** for scalability and cost efficiency.
2. **Data Storage**
   * **Raw Storage**: Azure Data Lake stores raw data for staging and further processing.
   * **Processed Storage**: After transformation, enriched and aggregated data is stored either in ADLS or Azure SQL Database, depending on use cases.
3. **Data Transformation**
   * **Tool**: **Azure Databricks** processes and transforms raw data to calculate metrics like total sales, average order value, and enrich customer information.
   * **Key Outputs**:
     + Total Sales
     + Average Order Value
     + Customer Enrichment
4. **Data Visualization**
   * **Tool**: **Power BI** connects to processed data in ADLS to create dashboards.
   * **Key Outputs**:
     + Real-time sales dashboards.
     + Insights into channel-specific performance and customer behaviour.

**Proposed Workflow**

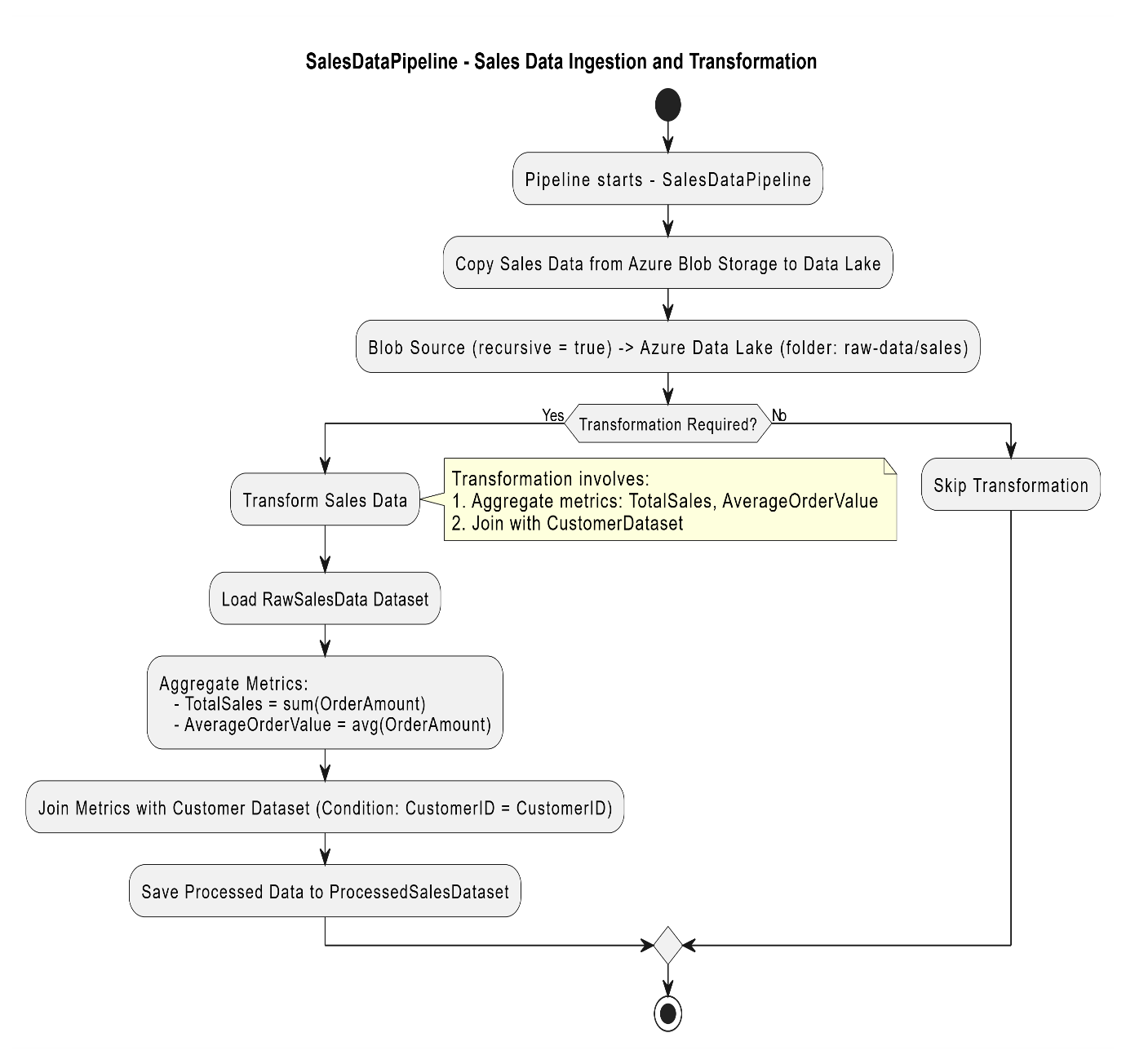
1. **Ingestion**:
   * Azure Data Factory ingests sales data from CSV into ADLS.
2. **Storage**:
   * Raw data is stored in ADLS.
3. **Transformation**:
   * Azure Databricks processes data to calculate metrics and enrich datasets.
4. **Visualization**: Power BI dashboards display insights for decision-making.

**Choice of Data Storage: Azure Data Lake Storage (ADLS)**

**Why Azure Data Lake Storage (ADLS) was chosen?**

1. **Scalability**
   * ADLS provides virtually unlimited storage, making it ideal for handling large and diverse datasets such as raw sales data from multiple sources (e.g., online platforms, physical stores, and mobile apps).
   * It scales seamlessly as the data grows, ensuring that storage needs are met without impacting performance.
2. **Cost Efficiency**
   * ADLS is designed to be cost-effective for storing raw and semi-structured data.
   * The pay-as-you-go model reduces operational expenses for storing large volumes of sales data.
3. **Support for Big Data Analytics**
   * ADLS integrates well with big data processing tools like **Azure Databricks**, enabling efficient data transformation and advanced analytics.
   * It supports parallel processing, which is essential for handling large datasets and running complex transformations.
4. **Schema Flexibility**
   * Unlike Azure SQL Database, ADLS does not enforce a rigid schema, making it more flexible for storing raw data in various formats (CSV, JSON, Parquet, etc.).
   * This flexibility is beneficial for staging data before processing.
5. **Integration with Azure Services**
   * ADLS integrates seamlessly with Azure Data Factory, Azure Databricks, and Azure Synapse Analytics, enabling an end-to-end data pipeline.
   * These integrations streamline the ingestion, transformation, and analysis workflows.
6. **Data Lake Features**
   * ADLS supports hierarchical namespaces, enabling the organization of data in a directory-like structure, which is useful for categorizing sales data (e.g., by date or channel).
   * Native security features, such as Azure Active Directory (AAD) and Role-Based Access Control (RBAC), ensure secure access to data.

**Architectural Diagram illustrating the data flow**



**Code Implementation:**

**Code/Scripts to implement the data pipeline:**

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, sum as \_sum, avg, when, lit, udf, sha2, regexp\_replace

# Step 1: Initialize Spark Session

spark = SparkSession.builder.appName("DataPipeline").getOrCreate()

# Step 2: Load Raw Data

raw\_data\_path = "/content/Sample\_Data\_For\_Data\_Engineering\_UseCase (1).csv"  # Update with actual path

raw\_df = spark.read.csv(raw\_data\_path, header=True, inferSchema=True)

# Step 3: Data Validation

# Check for missing critical columns

required\_columns = ["OrderID", "Quantity", "Price", "CreditCardNumber"]

missing\_columns = [col for col in required\_columns if col not in raw\_df.columns]

if missing\_columns:

    raise ValueError(f"Missing required columns: {missing\_columns}")

# Step 4: Data Transformation

# Add Total\_Sales column

transformed\_df = raw\_df.withColumn("Total\_Sales", col("Quantity") \* col("Price"))

# Handle missing values

transformed\_df = transformed\_df.fillna({

    "CustomerName": "Unknown",

    "PhoneNumber": "000-000-0000",

    "Location": "Unknown",

    "Country": "Unknown"

}).na.drop(subset=["OrderID", "Price", "Quantity"])

# Group by Location and calculate metrics

aggregated\_df = (

    transformed\_df.groupBy("Location")

    .agg(

        \_sum("Total\_Sales").alias("Total\_Sales"),

        avg("Total\_Sales").alias("Average\_Order\_Value"),

        \_sum("Quantity").alias("Total\_Quantity")

    )

)

# Step 5: Handle PII Data

# Anonymize CreditCardNumber

pii\_anonymized\_df = transformed\_df.withColumn(

    "CreditCard\_Anonymized", sha2(col("CreditCardNumber"), 256)  # SHA-256 hashing

).drop("CreditCardNumber")  # Drop original column for security

# Mask other sensitive data (e.g., phone numbers)

pii\_anonymized\_df = pii\_anonymized\_df.withColumn(

    "Masked\_PhoneNumber", regexp\_replace(col("PhoneNumber"), r".(?=.{4}$)", "\*")

)

# Step 6: Save Outputs

# Save transformed data

transformed\_output\_path = "transformed\_data"

transformed\_df.write.mode("overwrite").parquet(transformed\_output\_path)

# Save aggregated data

# Changed output path to a local directory. Update with your desired path.

aggregated\_output\_path = "aggregated\_data"

aggregated\_df.write.mode("overwrite").parquet(aggregated\_output\_path)

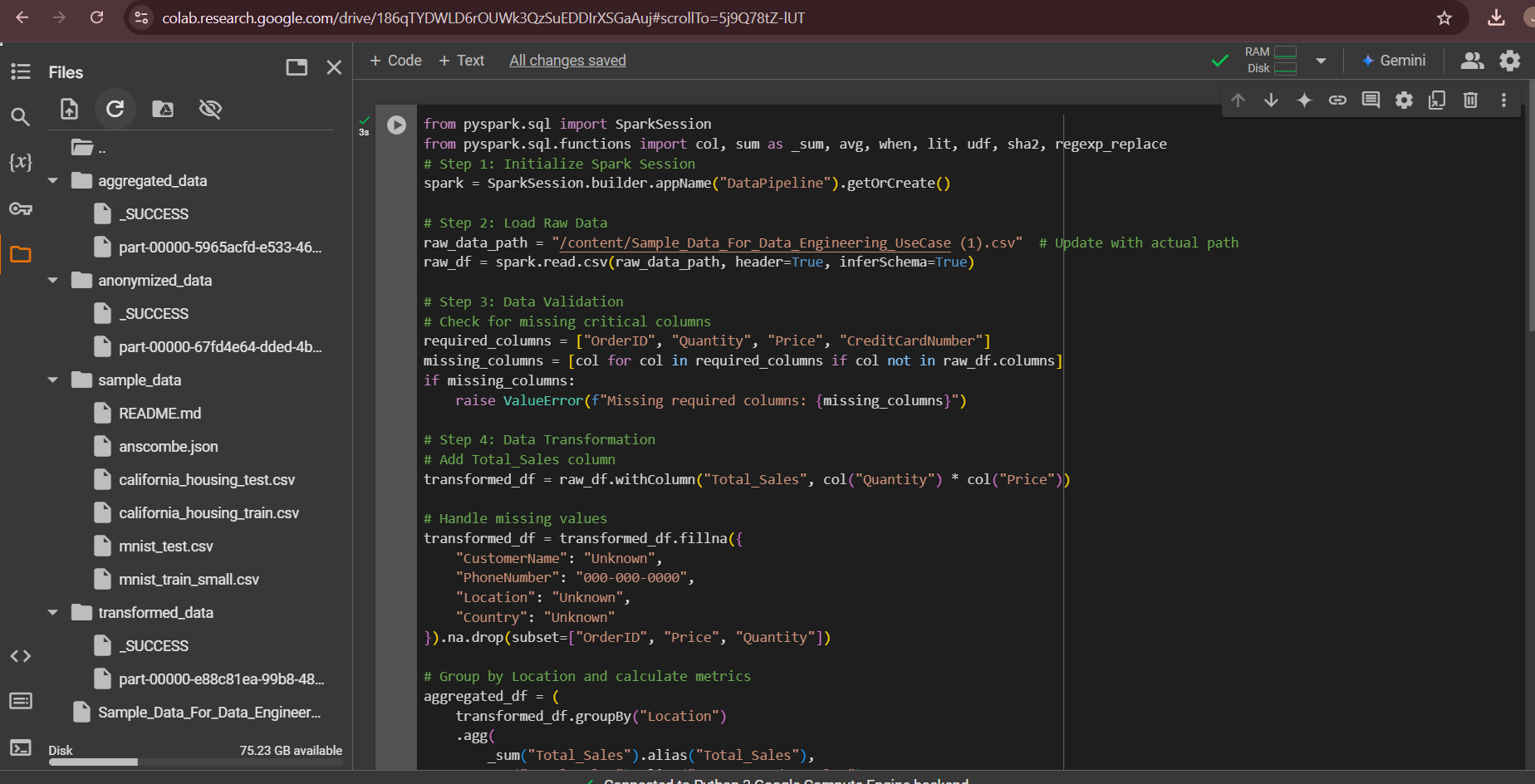
# Save anonymized PII data

# Changed output path to a local directory. Update with your desired path.

pii\_output\_path = "anonymized\_data"

pii\_anonymized\_df.write.mode("overwrite").parquet(pii\_output\_path)

print("Pipeline completed successfully.")



from pyspark.sql import SparkSession

from pyspark.sql.functions import udf, col

from pyspark.sql.types import StringType

from Crypto.Cipher import AES

import base64

# Initialize Spark Session

spark = SparkSession.builder.appName("EncryptCreditCard").getOrCreate()

# Sample data (replace with your actual DataFrame)

data = [("1234567812345678",), ("8765432187654321",)]

columns = ["CreditCardNumber"]

sales\_df = spark.createDataFrame(data, columns)

# Encryption setup

def create\_cipher():

    key = b'sixteen byte key'  # Ensure 16-byte key

    return AES.new(key, AES.MODE\_ECB)

# Encrypt function

def encrypt\_credit\_card(card\_number):

    if card\_number is None:

        return None

    padded\_number = card\_number.ljust(32)  # Pad to make it 32 bytes

    cipher = create\_cipher()  # Create cipher locally in the function

    encrypted = cipher.encrypt(padded\_number.encode())

    return base64.b64encode(encrypted).decode()

# Register UDF for encryption

encrypt\_udf = udf(encrypt\_credit\_card, StringType())

# Apply encryption

sales\_df = sales\_df.withColumn(

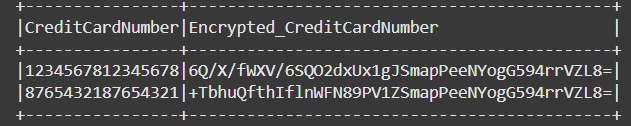
    "Encrypted\_CreditCardNumber",

    encrypt\_udf(col("CreditCardNumber"))

)

# Show the results

sales\_df.show(truncate=False)



### **Handling PII Data**

* Masked or encrypted credit card numbers to ensure privacy.
* Dropped fields like CreditCardNumber and ExpiryDate after securing them.
* Example: sales\_df = sales\_df.drop('CreditCardNumber', 'ExpiryDate')

**Dataset Documentation**:

## **Structure of the Sales Dataset**

The sales dataset consists of the following columns with their respective data types and descriptions:

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| OrderID | String | Unique identifier for each sales order. |
| CustomerName | String | Name of the customer who placed the order. |
| PhoneNumber | String | Contact number of the customer. |
| Location | String | Geographic location of the sale. |
| Country | String | Country where the sale occurred. |
| StoreCode | String | Identifier for the store or channel (e.g., online, app). |
| Product | String | Name or description of the product sold. |
| Quantity | Integer | Number of units sold in the transaction. |
| Price | Float | Price per unit of the product. |
| Date | Date | Date of the sales transaction. |
| CreditCardNumber | String | Credit card number used for the transaction. |
| ExpiryDate | Date | Expiry date of the credit card. |

## **Structure of the Sales Dataset**

The sales dataset consists of the following columns with their respective data types and descriptions:

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Description** |
| OrderID | String | Unique identifier for each sales order. |
| CustomerName | String | Name of the customer who placed the order. |
| PhoneNumber | String | Contact number of the customer. |
| Location | String | Geographic location of the sale. |
| Country | String | Country where the sale occurred. |
| StoreCode | String | Identifier for the store or channel (e.g., online, app). |
| Product | String | Name or description of the product sold. |
| Quantity | Integer | Number of units sold in the transaction. |
| Price | Float | Price per unit of the product. |
| Date | Date | Date of the sales transaction. |
| CreditCardNumber | String | Credit card number used for the transaction. |
| ExpiryDate | Date | Expiry date of the credit card. |

## **Data Preprocessing Steps**

### **Schema Validation**

* Verified column names and data types to ensure consistency with the expected structure.

### **Handling Missing Values**

* Imputed missing values for non-critical fields (e.g., CustomerName, PhoneNumber) with placeholder values like N/A.
* Dropped rows with missing critical fields (OrderID, Product, Price, Quantity).
* Example: sales\_df = sales\_df.dropna(subset=['OrderID', 'Product', 'Price', 'Quantity'])

### **Data Transformation**

* Added calculated fields such as:
* Total\_Sales = Quantity × Price.
* Avg\_Order\_Value = Total sales divided by the number of orders.
* Example: sales\_df = sales\_df.withColumn('Total\_Sales', col('Quantity') \* col('Price'))

### **Handling PII Data**

* Masked or encrypted credit card numbers to ensure privacy.
* Dropped fields like CreditCardNumber and ExpiryDate after securing them.
* Example: sales\_df = sales\_df.drop('CreditCardNumber', 'ExpiryDate')

### **Data Enrichment**

* Added external fields, such as customer demographic data, using a lookup table or external dataset.
* Example: enriched\_df = sales\_df.join(demographic\_df, on='CustomerName', how='left')

### **Date Parsing**

* Converted date columns into a standard Date format for easy aggregation.
* Example: sales\_df = sales\_df.withColumn('Date', to\_date(col('Date'), 'yyyy-MM-dd'))

### **Data Validation**

* Ensured that all calculated and transformed columns adhere to expected ranges and formats.
* Example: sales\_df = sales\_df.filter((col('Price') > 0) & (col('Quantity') > 0))

## **Summary**

* Columns Removed: CreditCardNumber, ExpiryDate.
* New Columns Added: Total\_Sales

**Azure Work Flow Screen Shots**

**1. Setting Up Azure Resources**

**Step 1: Create a Resource Group**

1. In the Azure Portal, click on **"Resource Groups"** from the left-hand menu.
2. Click **"Create"** and fill in:
   * **Resource Group Name**: e.g., RetailSalesRG
   * **Region**: Select your closest region (e.g., East US).
3. Click **"Review + Create"** and then **"Create"**.

**Step 2: Create Azure Data Lake Storage (Gen2)**

1. In the Azure Portal, search for **"Storage Accounts"** in the search bar.
2. Click **"Create"** and fill in:
   * **Subscription**: Select your active subscription.
   * **Resource Group**: Choose the one you created (e.g., RetailSalesRG).
   * **Storage Account Name**: e.g., xyzretailstorage.
   * **Performance**: Standard (recommended for most cases).
   * **Redundancy**: Locally Redundant Storage (LRS).
3. Under **Advanced**, enable **Hierarchical Namespace** to use Data Lake features.
4. Click **"Review + Create"** and then **"Create"**.

**Step 3: Create a Container in the Storage Account**

1. Go to your newly created **Storage Account**.
2. In the left-hand menu, select **"Containers"** under the **Data Lake Storage** section.
3. Click **"+ Container"** and:
   * **Name**: e.g., salesdata.
   * **Public Access Level**: Private.
4. Click **"Create"**.

**2. Uploading CSV Data to Azure Data Lake**

**Step 1: Open the Storage Explorer in the Portal**

1. In your **Storage Account**, click on **"Storage Browser"** in the left-hand menu.
2. Navigate to your container (salesdata).

**Step 2: Upload Your CSV File**

1. Click **"Upload"** and choose the CSV file from your computer.
2. Click **"Upload"** again to confirm.

**3. Ensuring Data Quality During Ingestion**

**Step 1: Inspect the Uploaded File**

1. In your container (salesdata), click on the uploaded file.
2. Use the **"Open"** or **"Download"** option to verify its contents.

**Step 2: Set Up Azure Data Factory (Optional for Automation)**

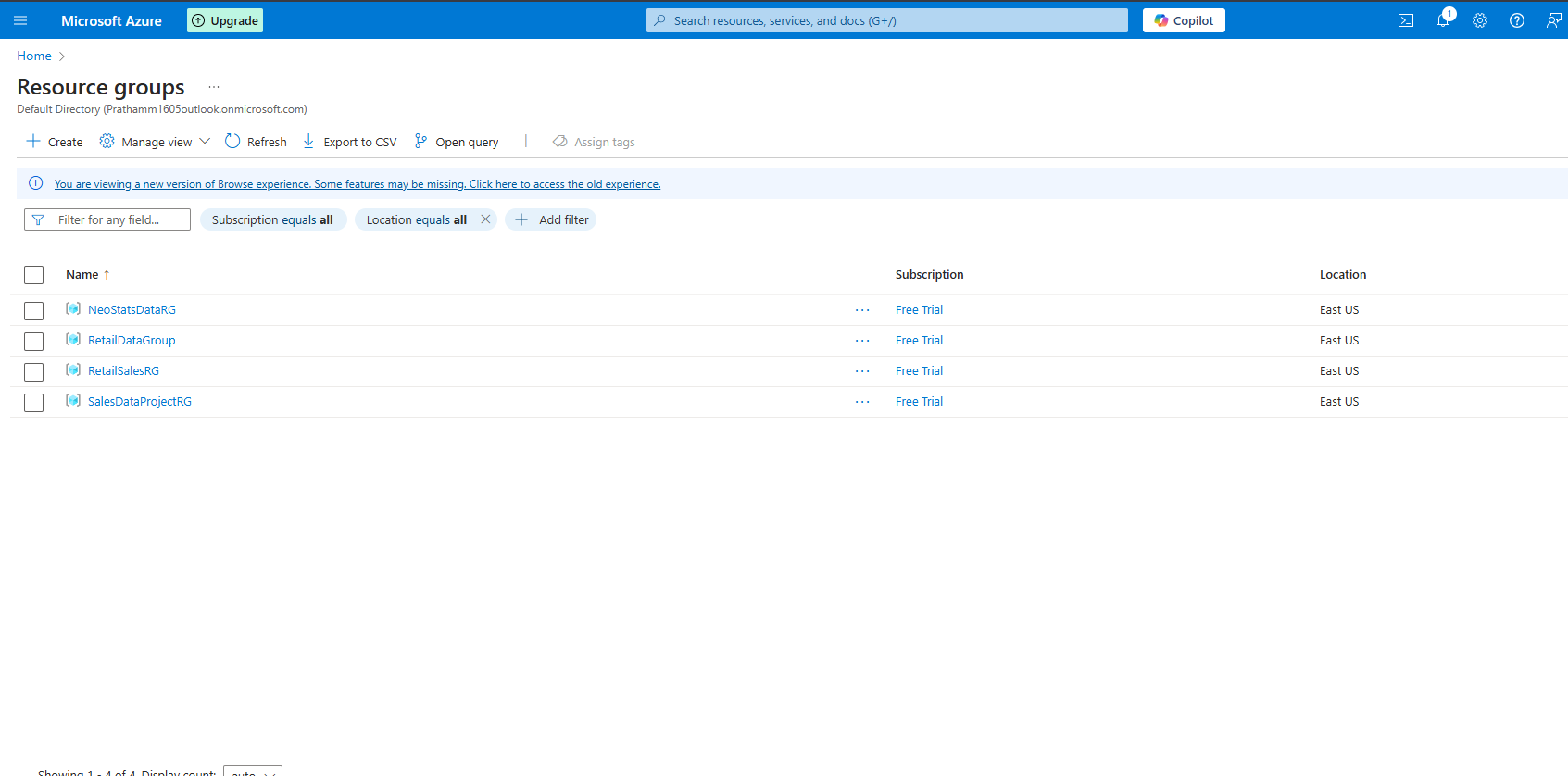
1. In the Azure Portal, search for **"Data Factory"** in the search bar.
2. Click **"Create"** and fill in:
   * **Subscription**: Select your active subscription.
   * **Resource Group**: Choose the one you created earlier.
   * **Data Factory Name**: e.g., SalesDataPipeline.
   * **Region**: Same as your storage account.
3. Click **"Review + Create"** and then **"Create"**.

**4. Handling Anomalies During Ingestion**

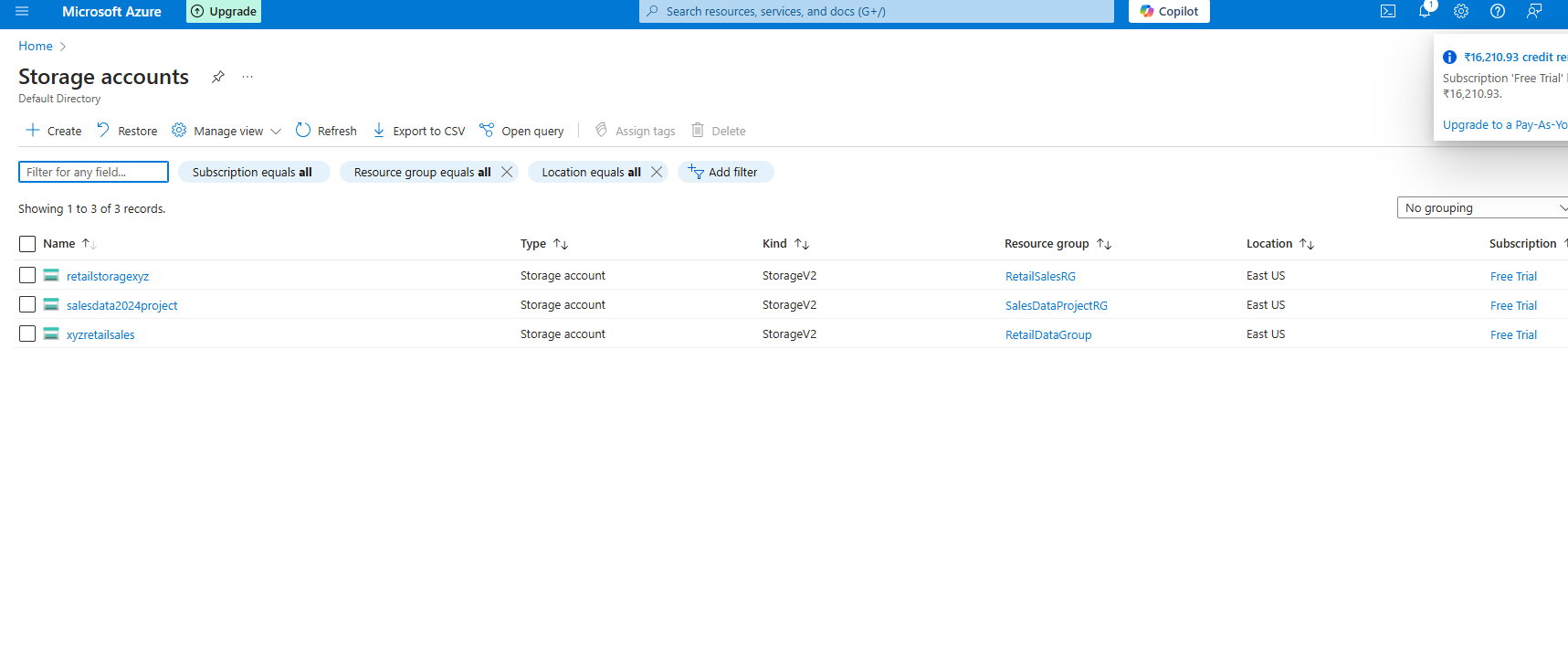
**Step 1: Use Azure Data Factory for Validation**

1. Open your newly created **Data Factory**.
2. Click **"Author & Monitor"** to open the Data Factory Studio.
3. Create a new **Pipeline**:
   * Add a **"Copy Data"** activity.
   * Source: Select your uploaded CSV file in Data Lake.
   * Sink: Specify another container or a new table in Azure SQL Database.
4. Add a **Mapping Data Flow**:
   * Add transformations like removing null rows, trimming spaces, or formatting fields.
   * Save the transformed data into a clean **sink**.

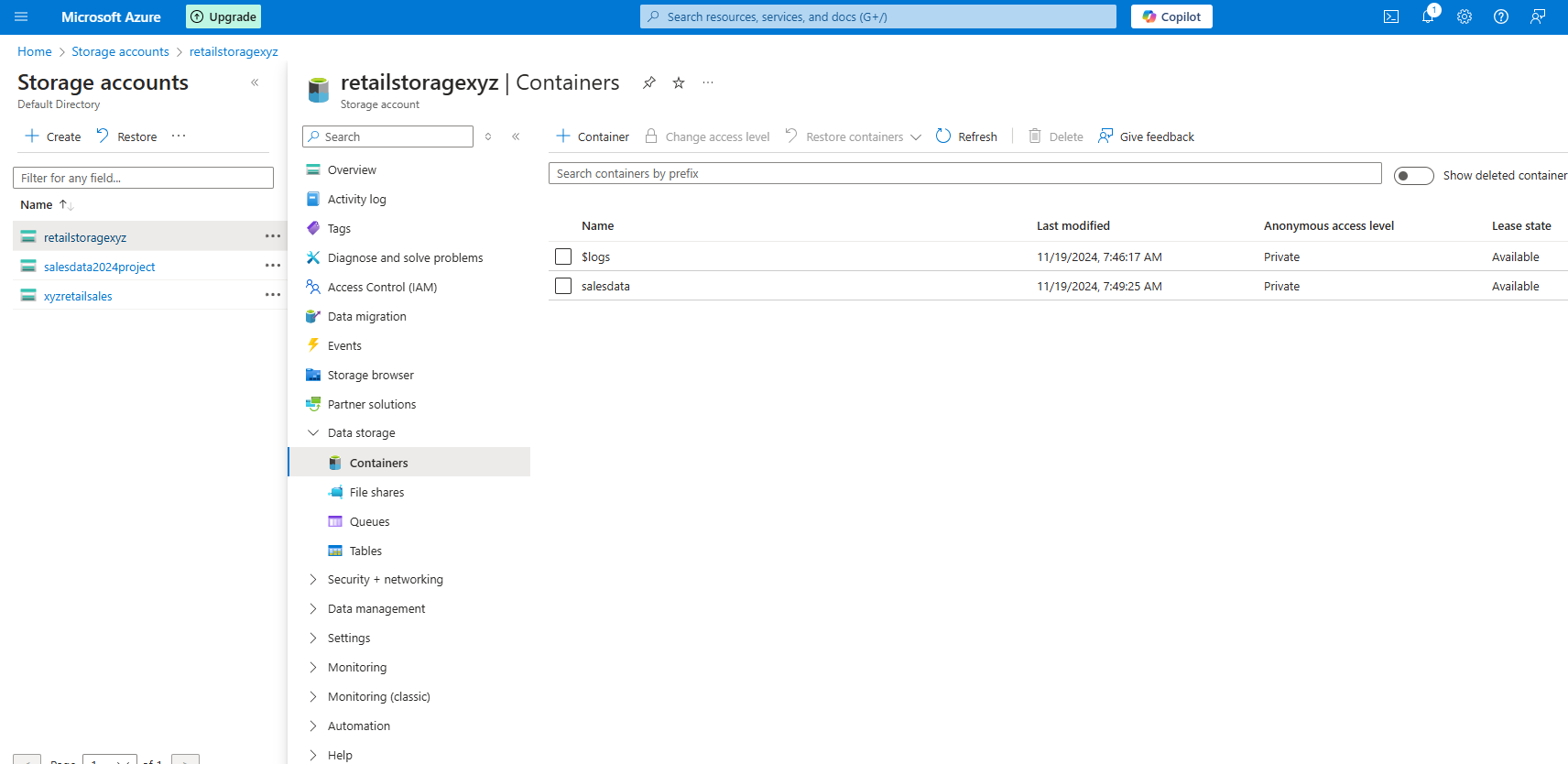
**In the resource created a file called RetailSalesRG**



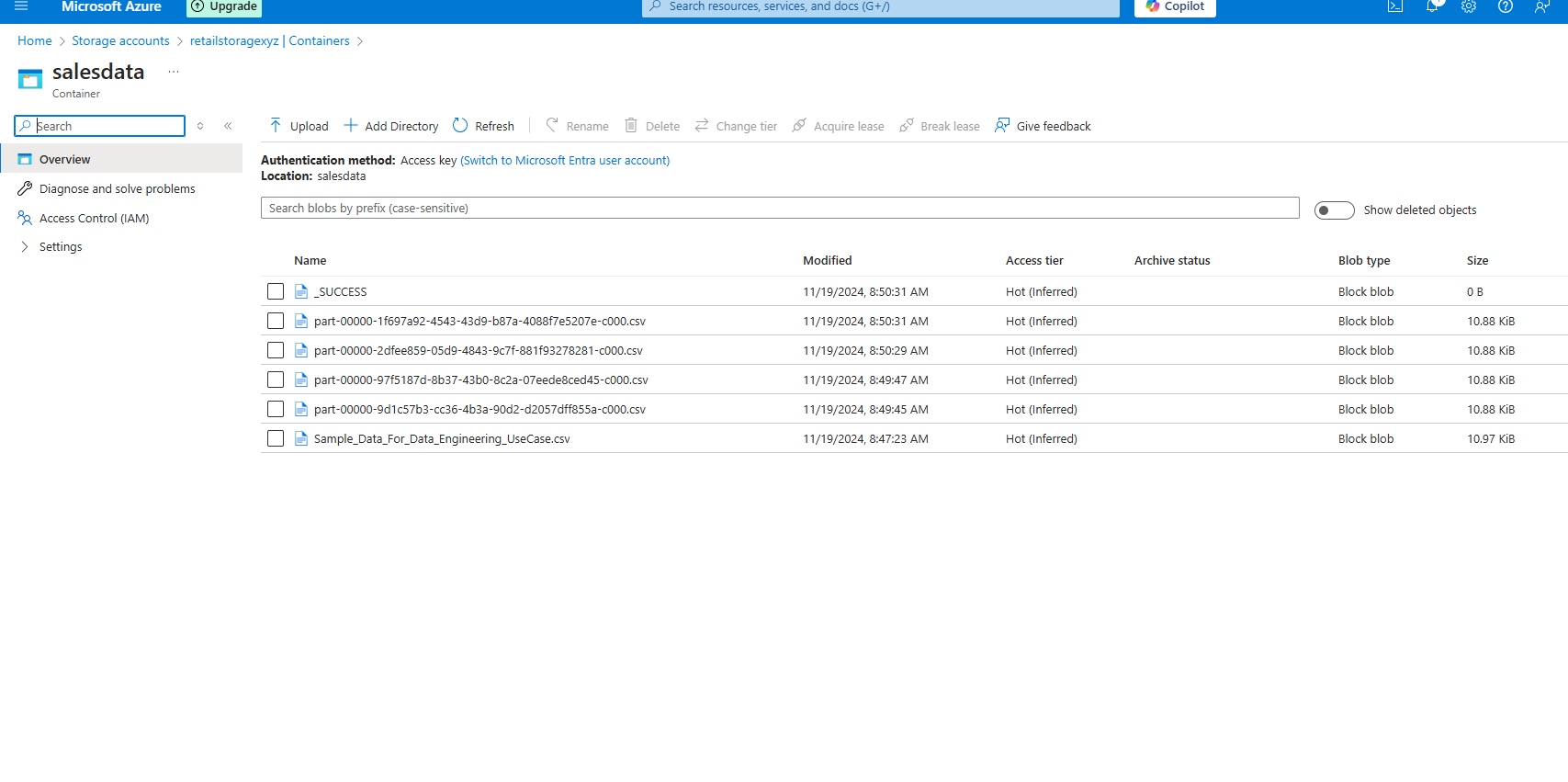
**Go to storage accounts and create a storage account  
Here I Have created with a name retailstoragexyz**



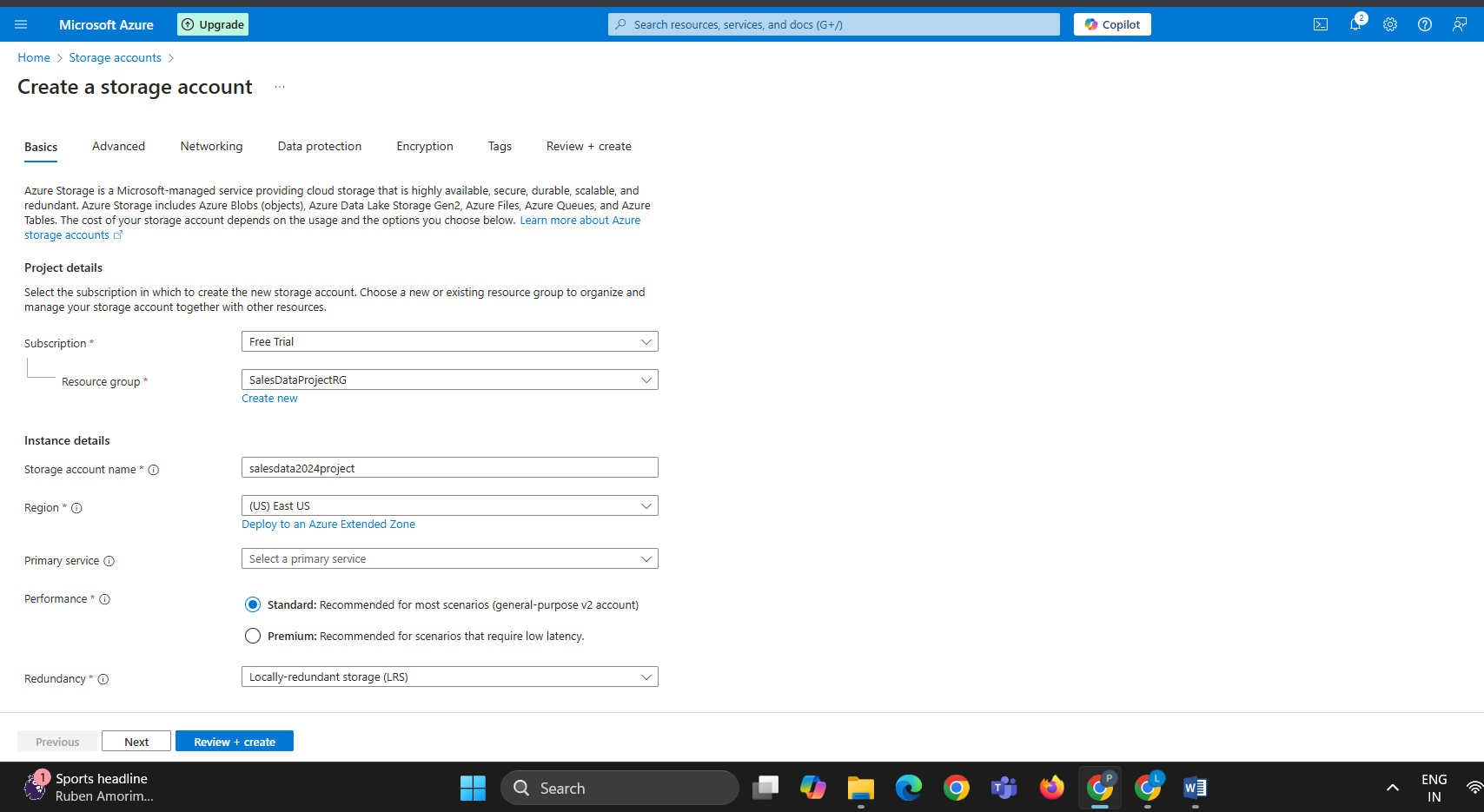
**Click on retailstoragexyz and go to Containers in Data Storage**

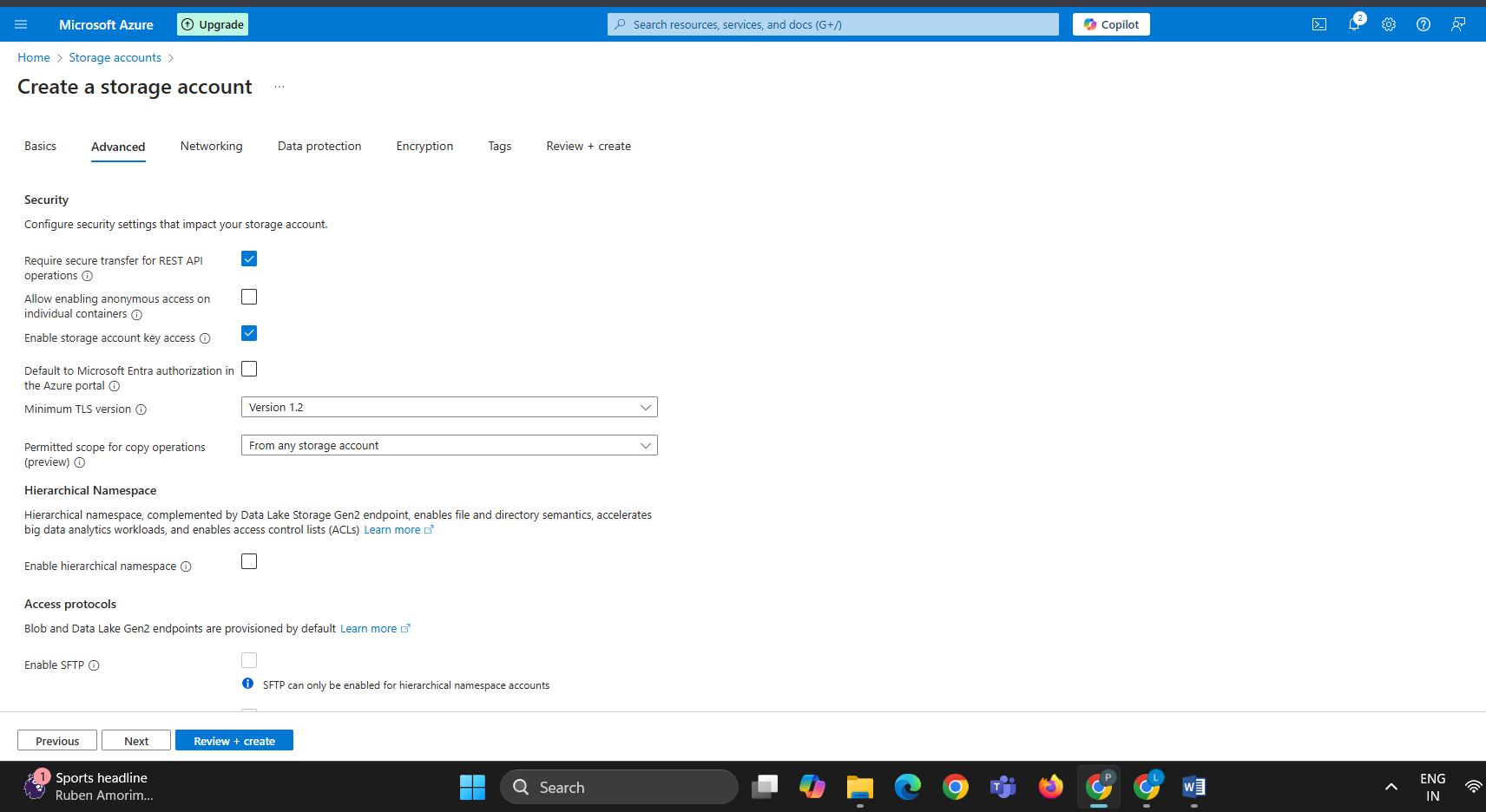


**Create a raw folder named salesdata and upload the csv file which is given**

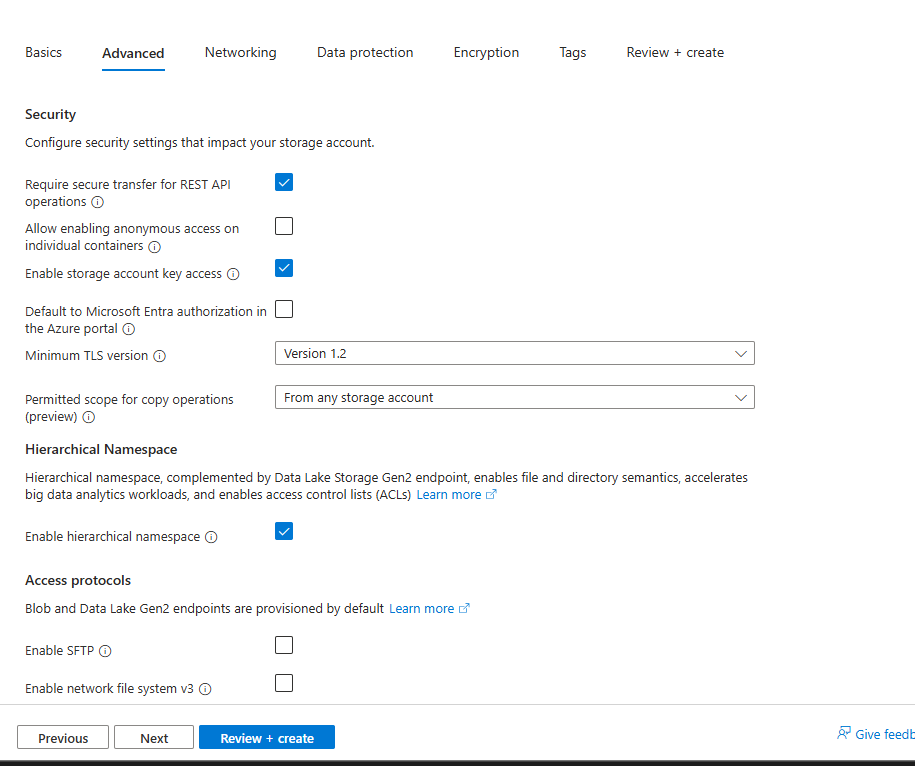


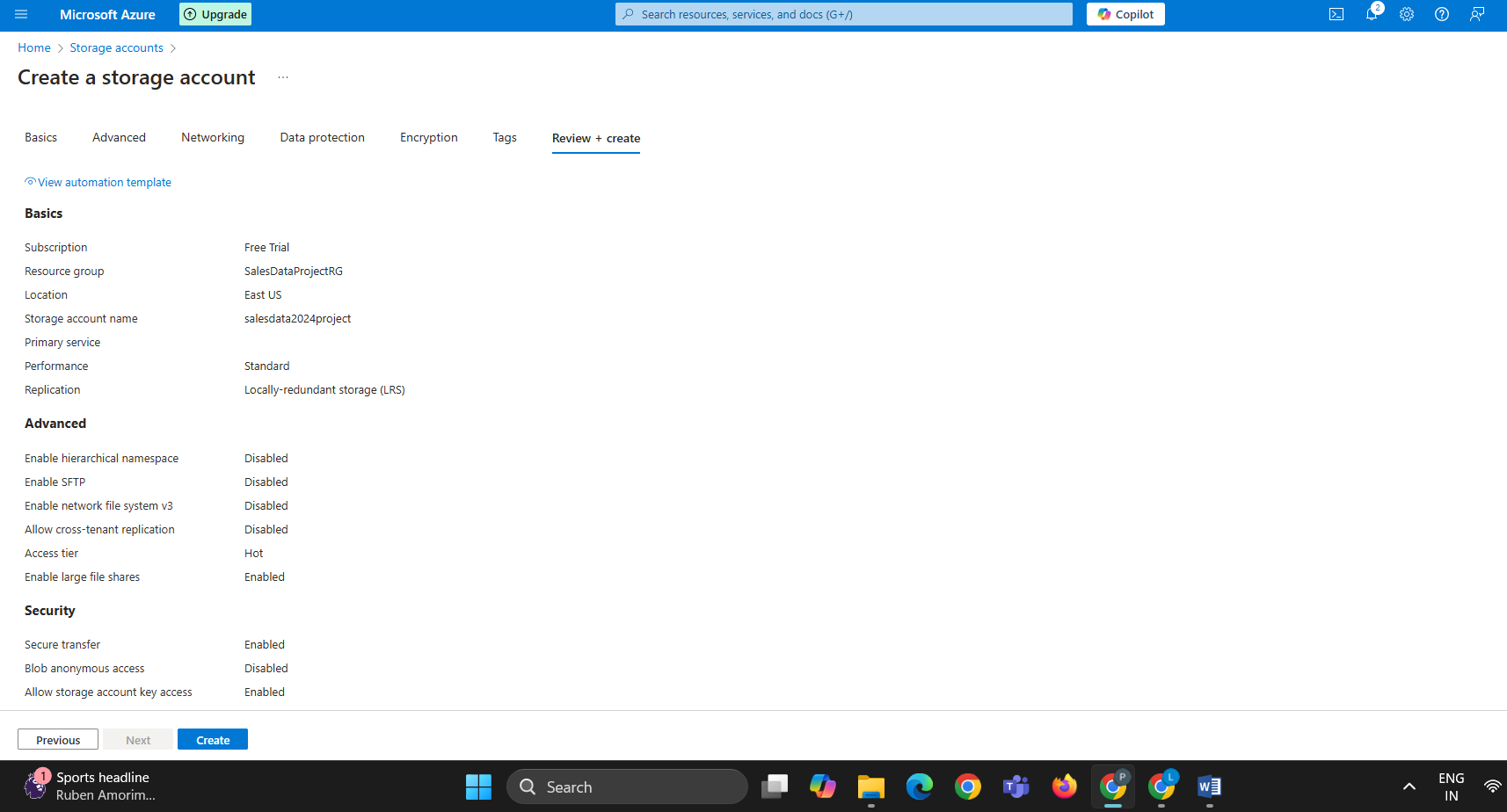
**Creating Storage Account**



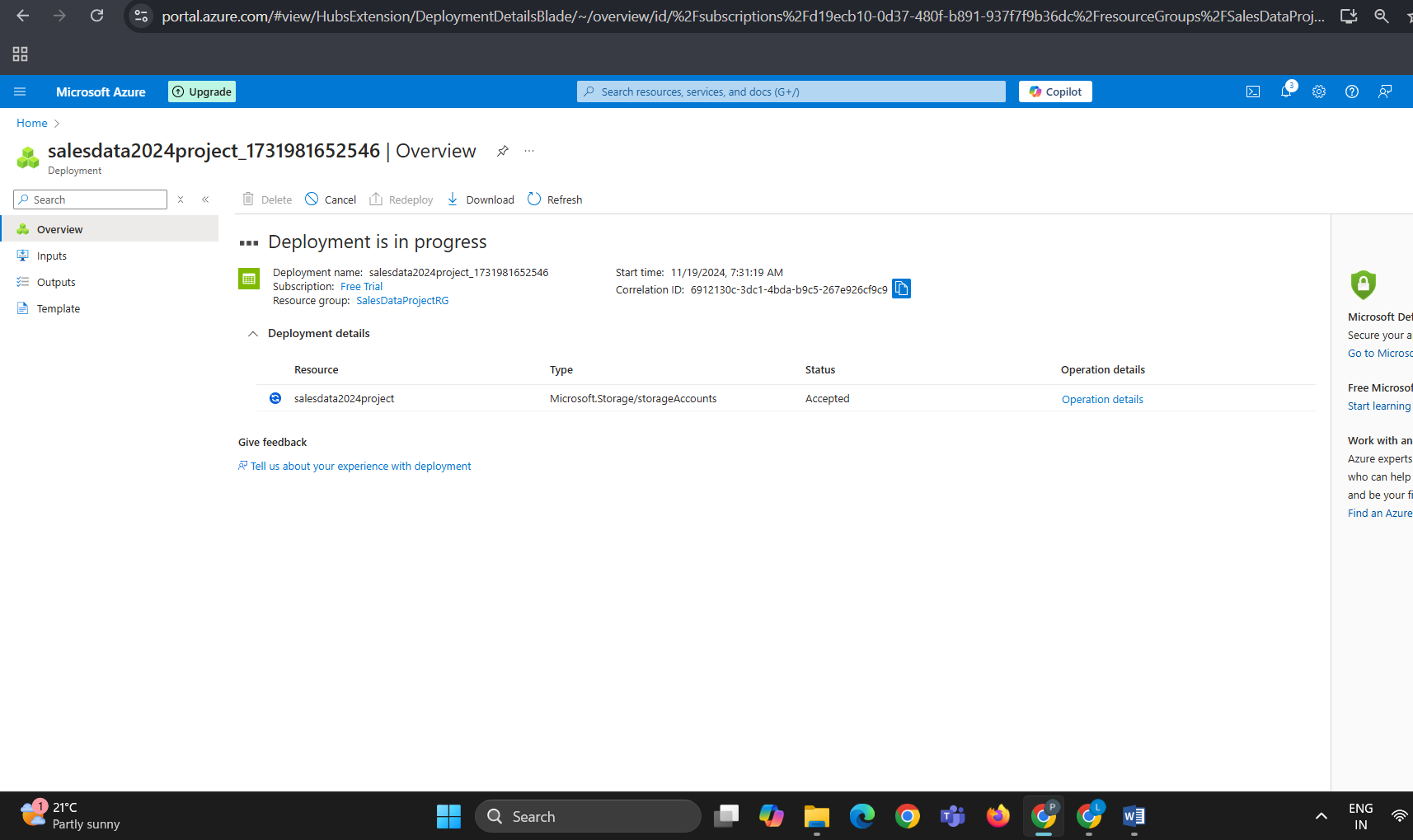


**Enabeling Hierarchical Namespace**

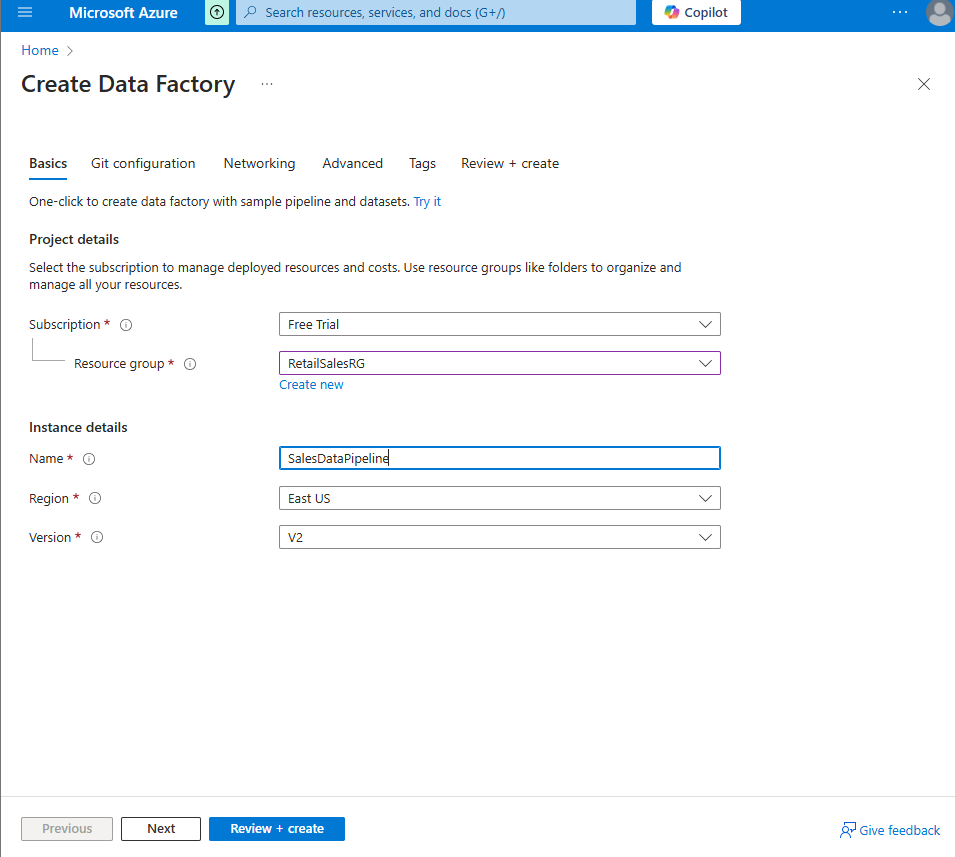


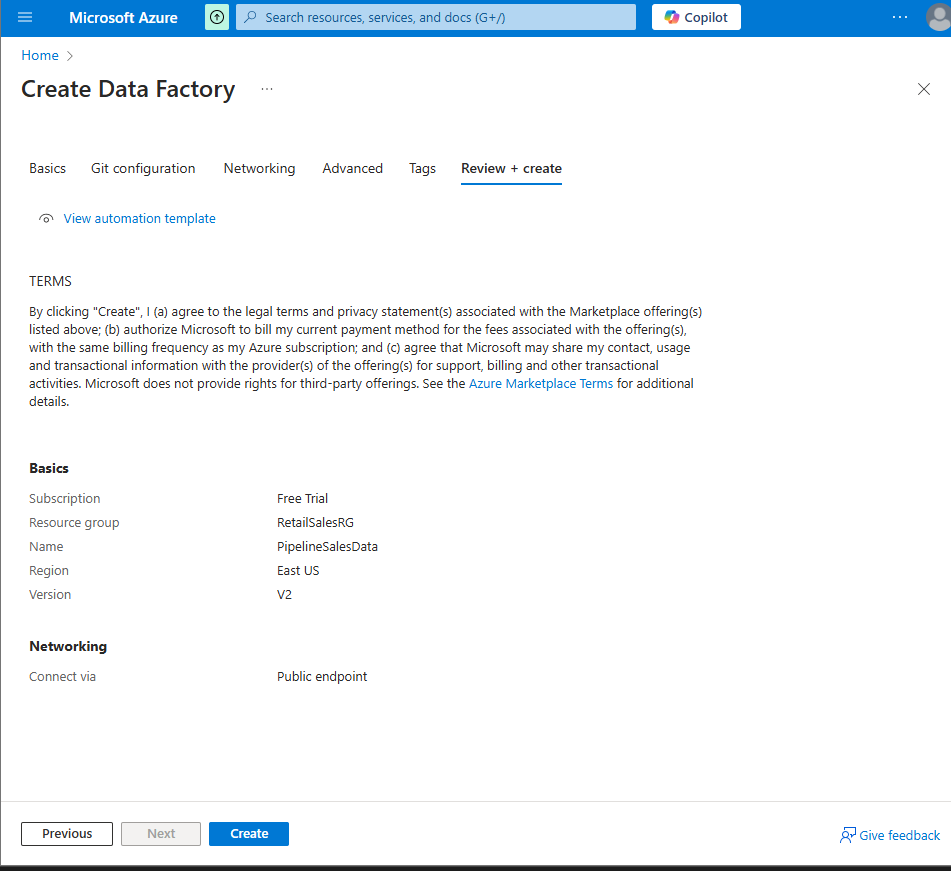


**Deploying the project in progress**

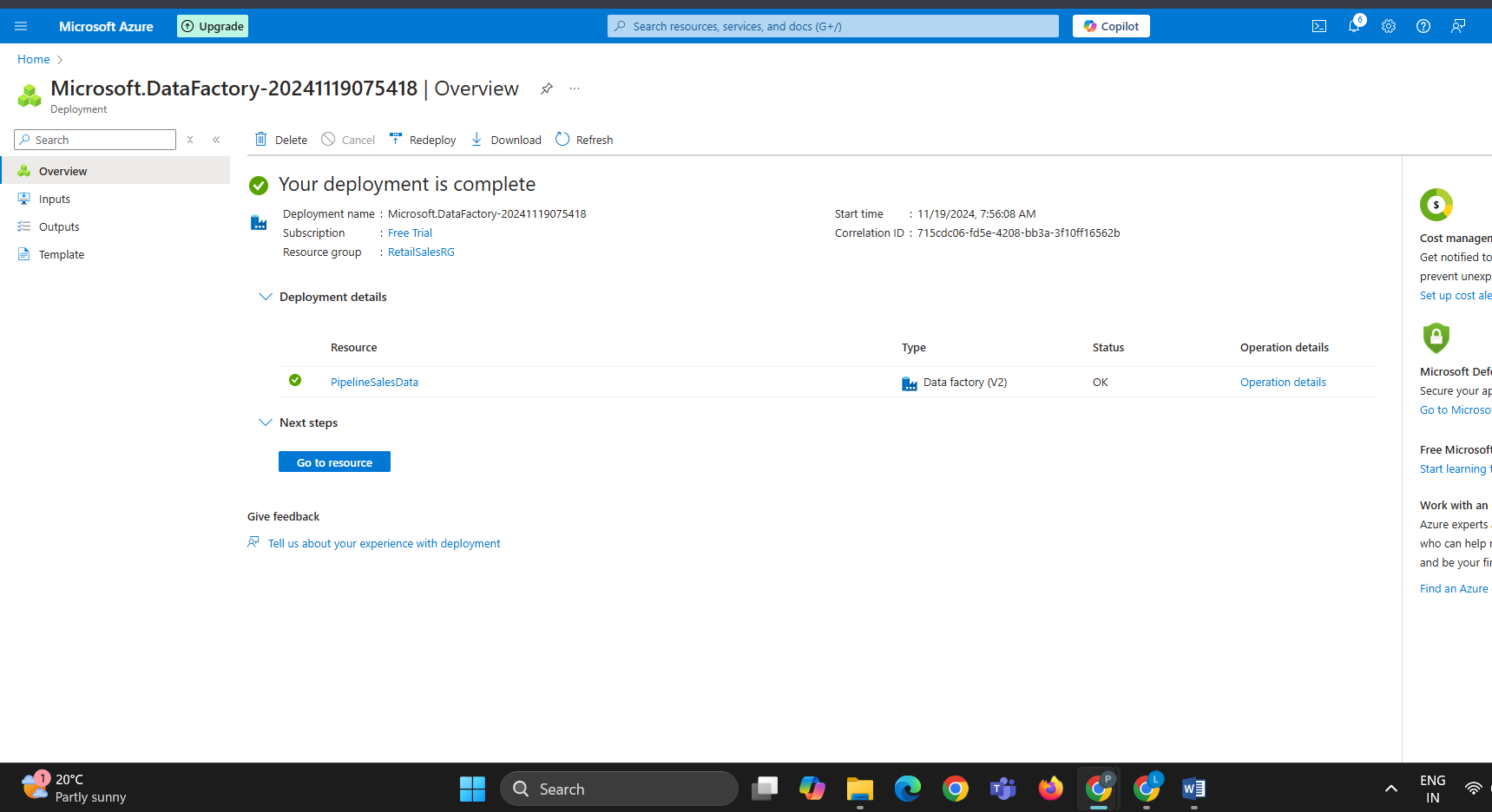


**Creating the Data Factory**

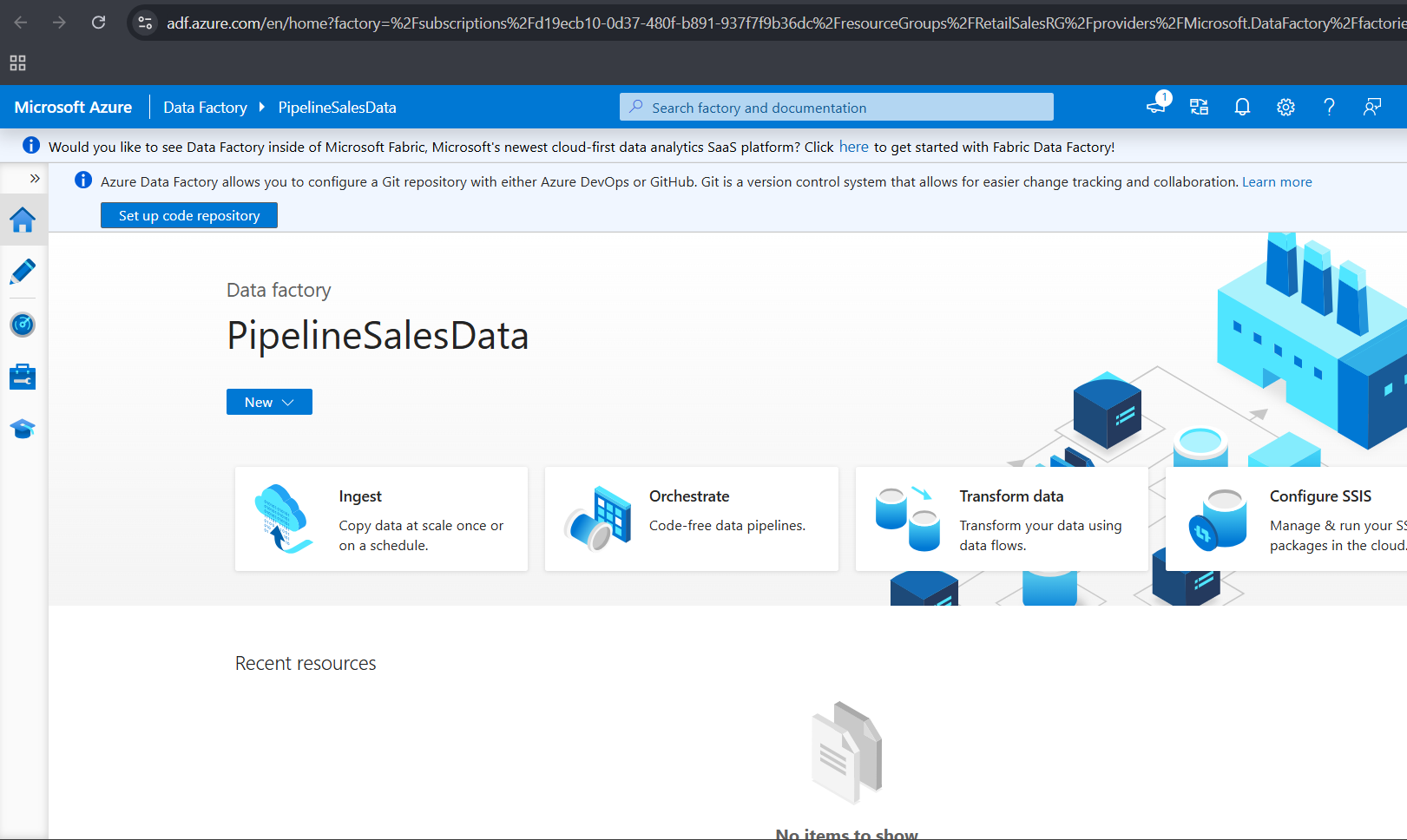


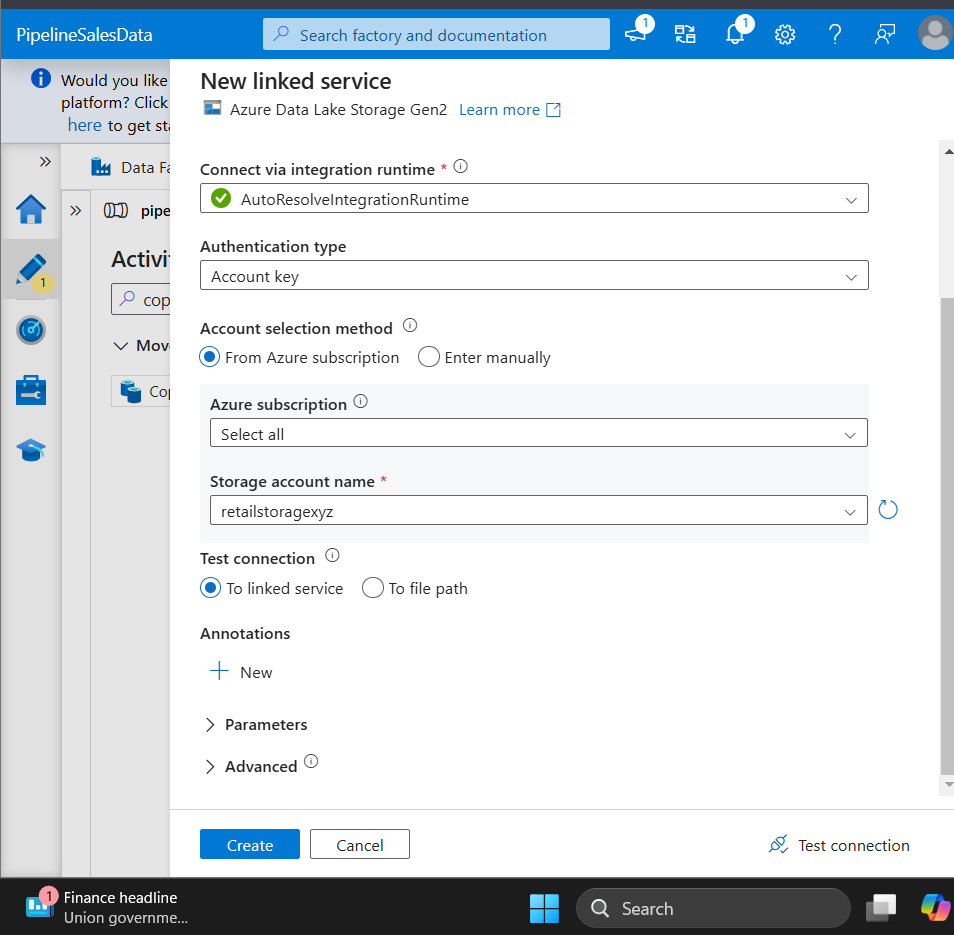


**Deployed Successfully**

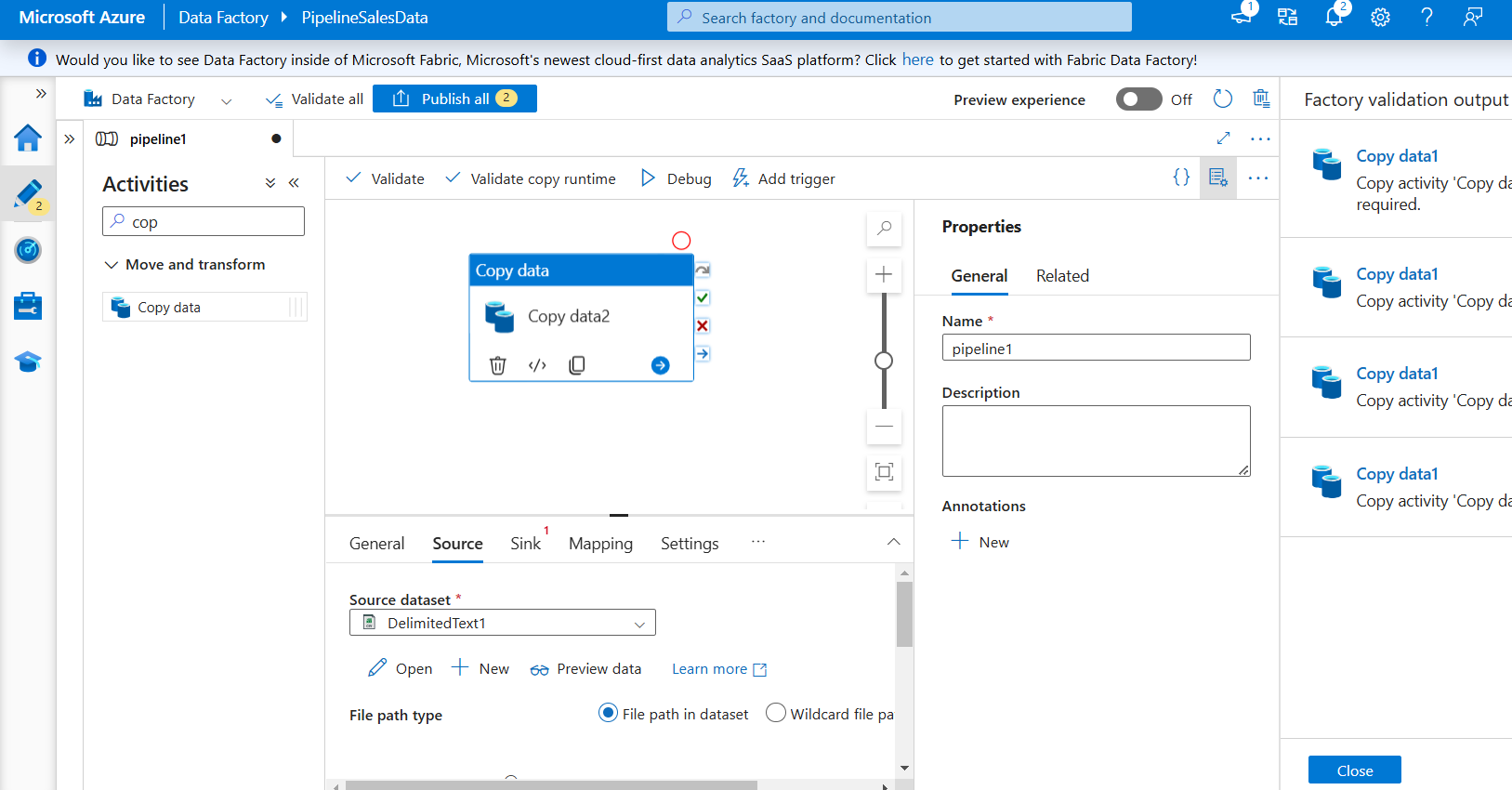


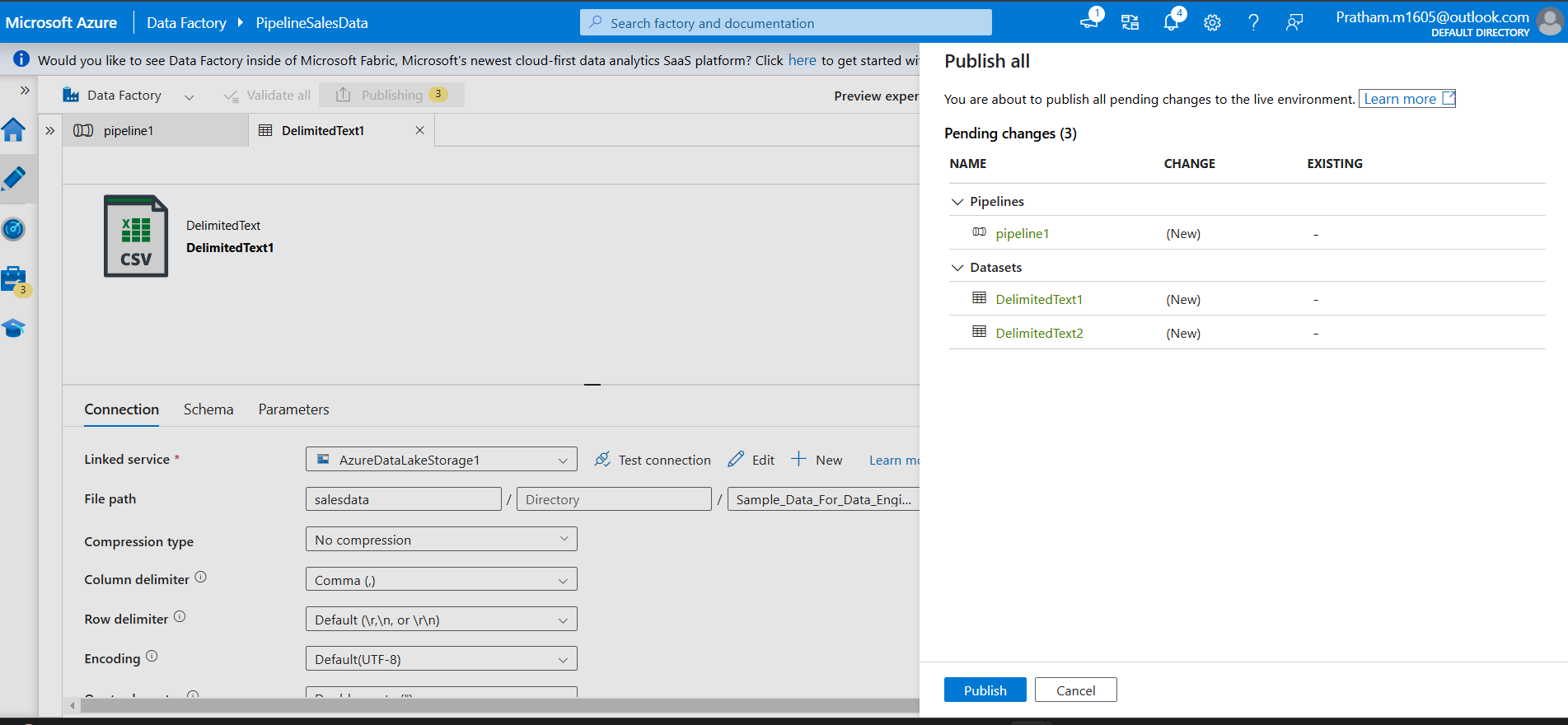
**Starting up with pipeline**

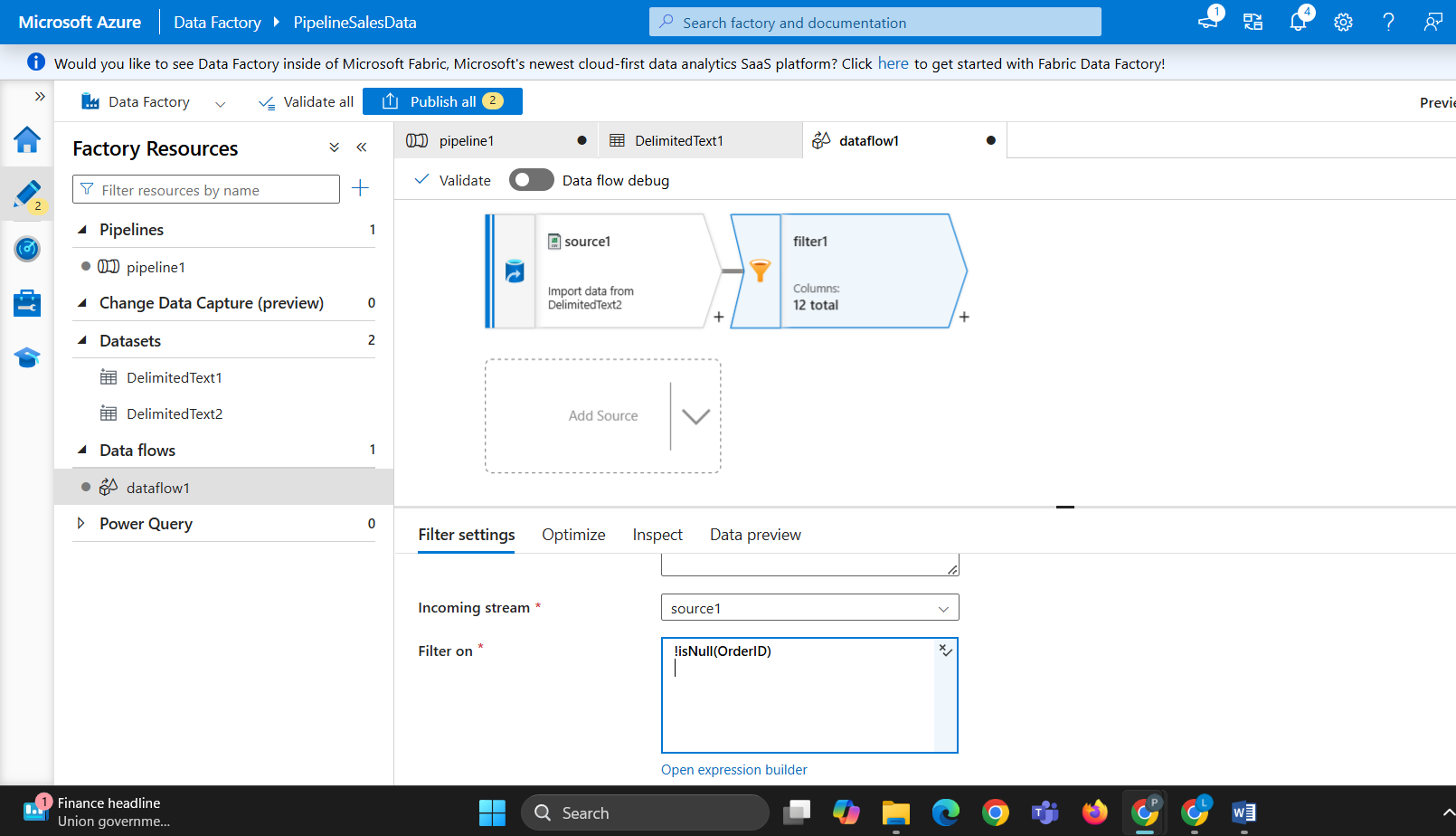


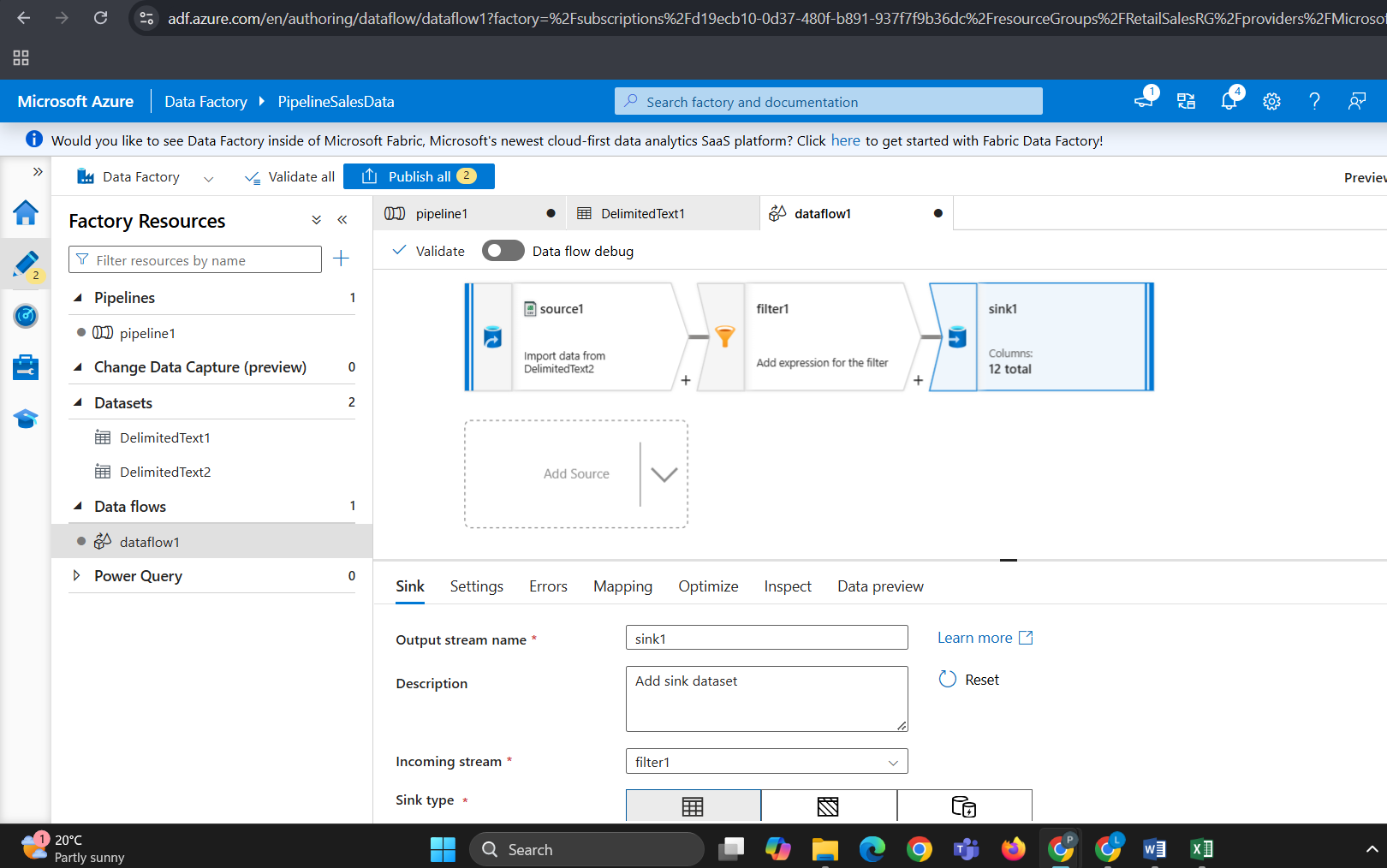


**Dragging data to canvas for performing filter, transformation, aggregation ,derived column and sinking it to get the desired OUTPUT in CSV format.**

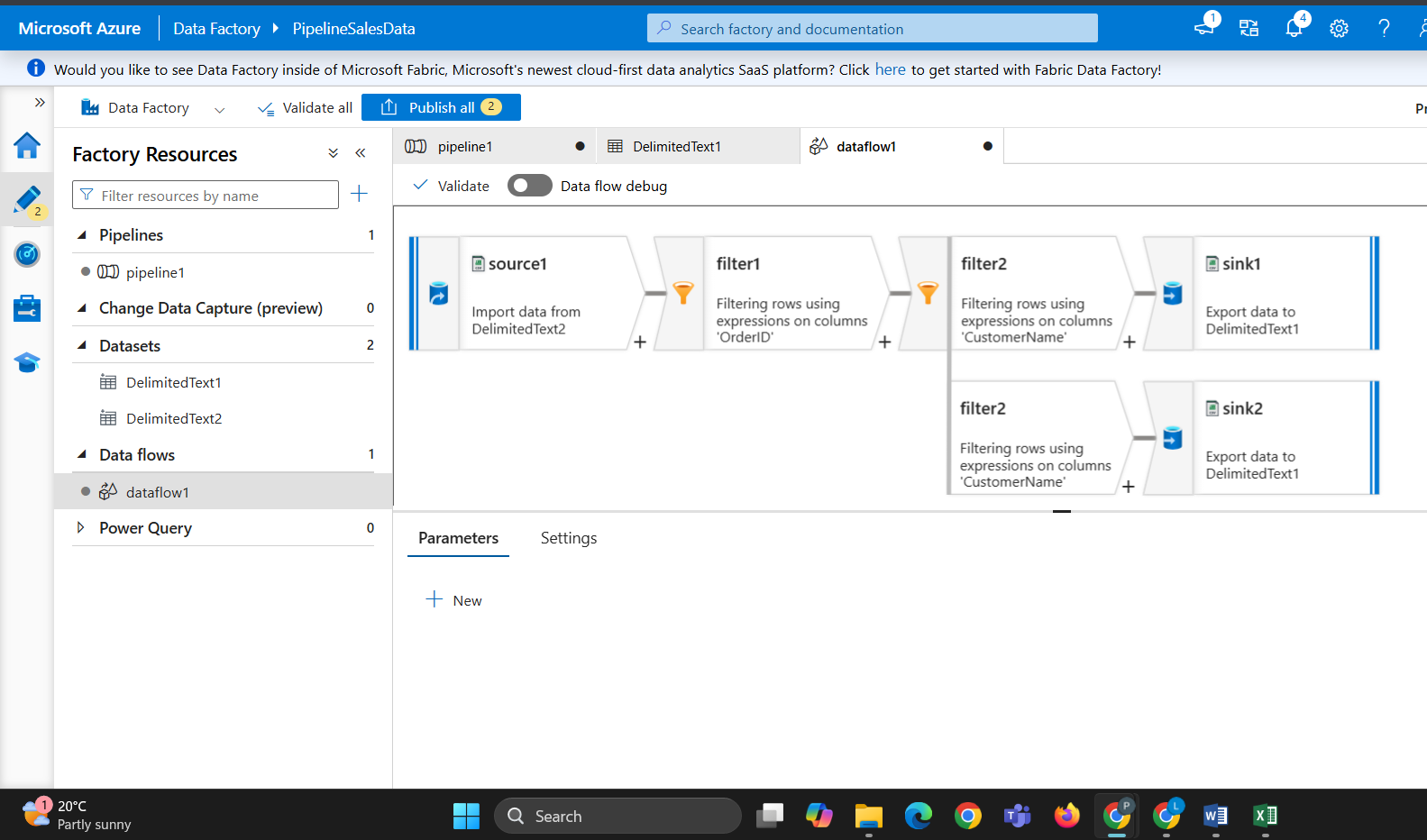


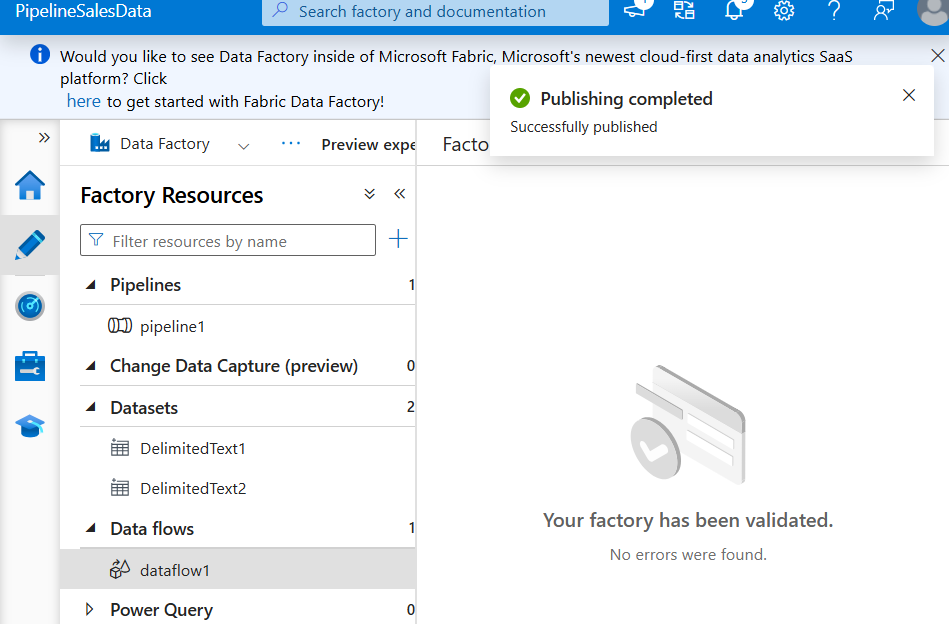


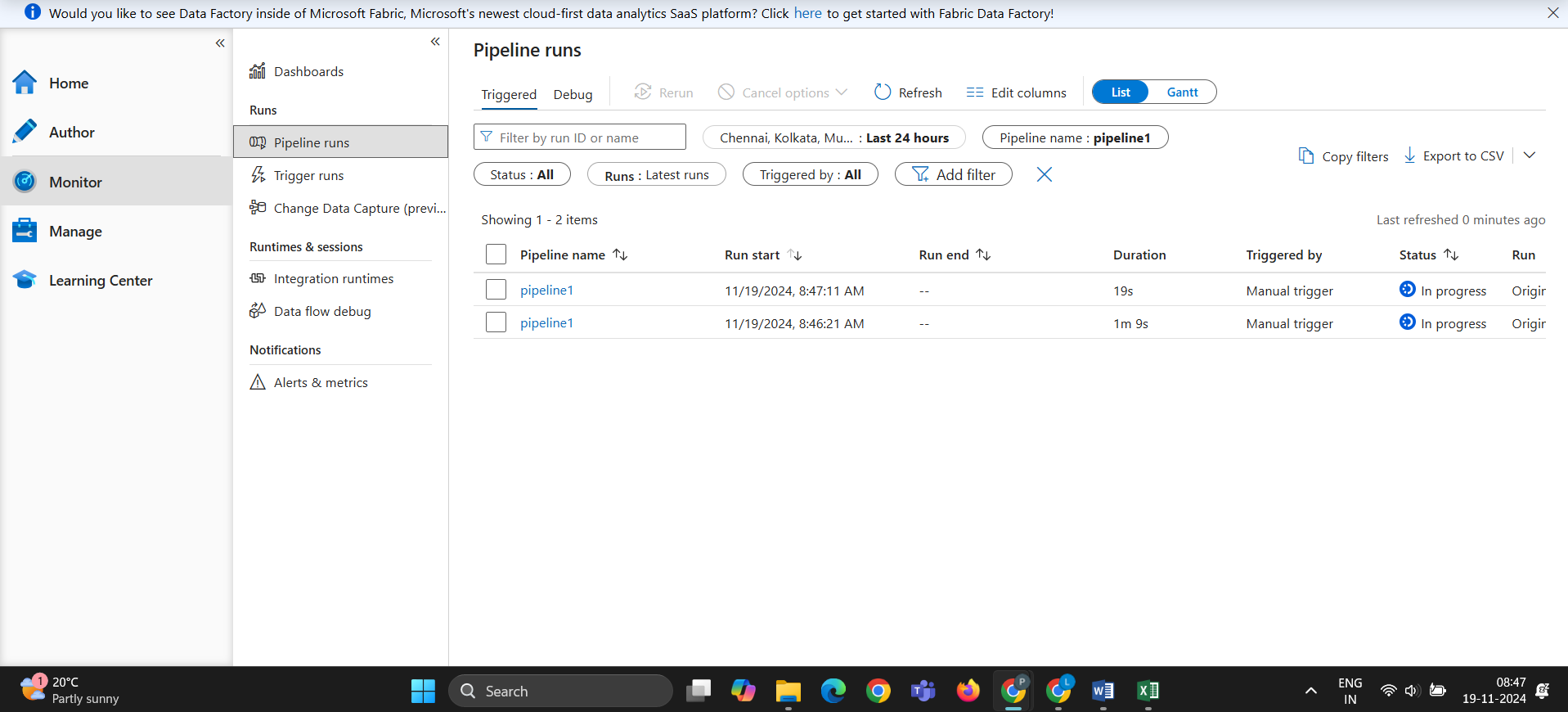




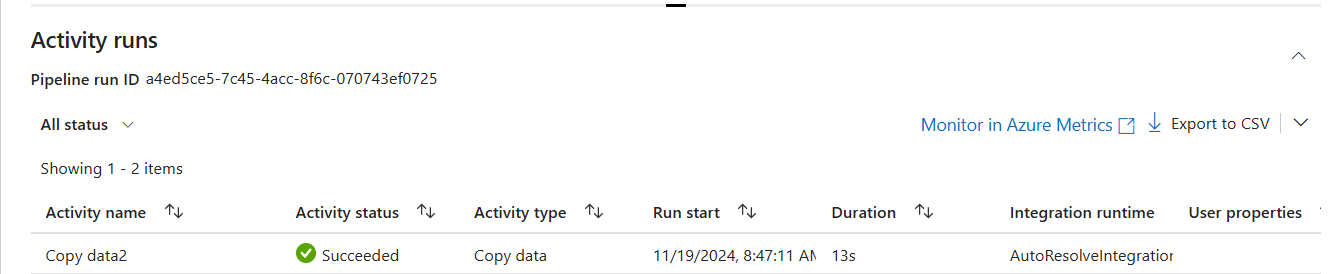
**If 2 filters are used it might render some error so in next step we will be reducing it and we will be using a single filter.**



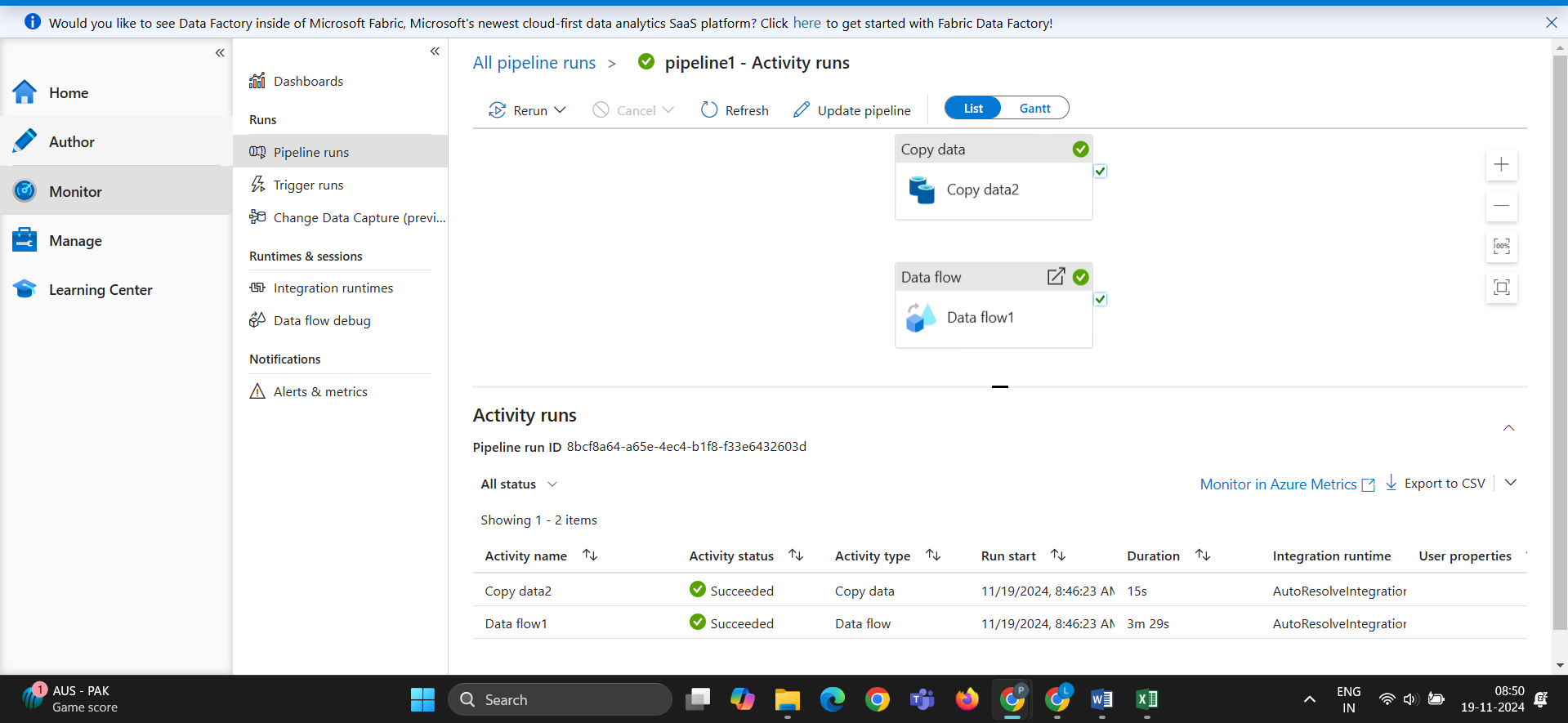


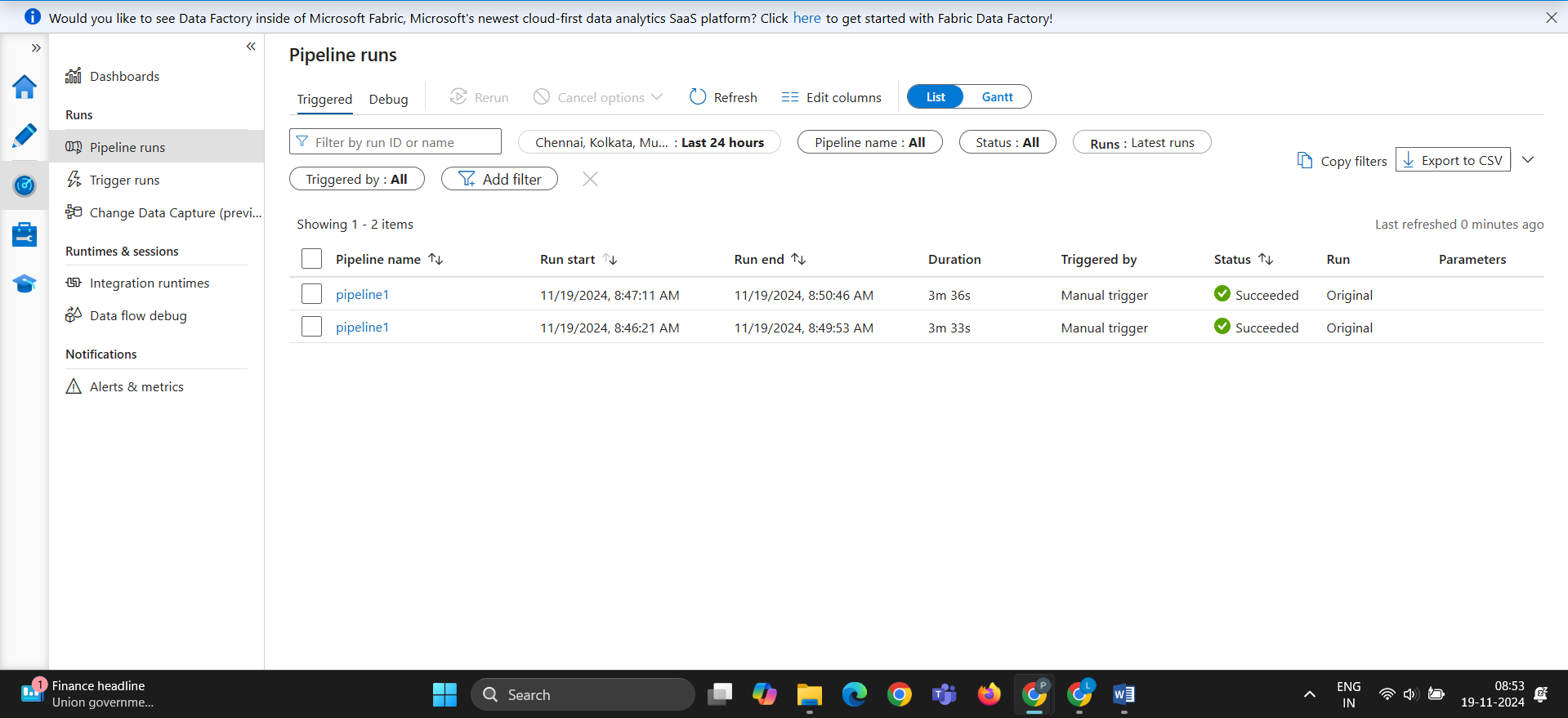


**Activity is running successfully.**

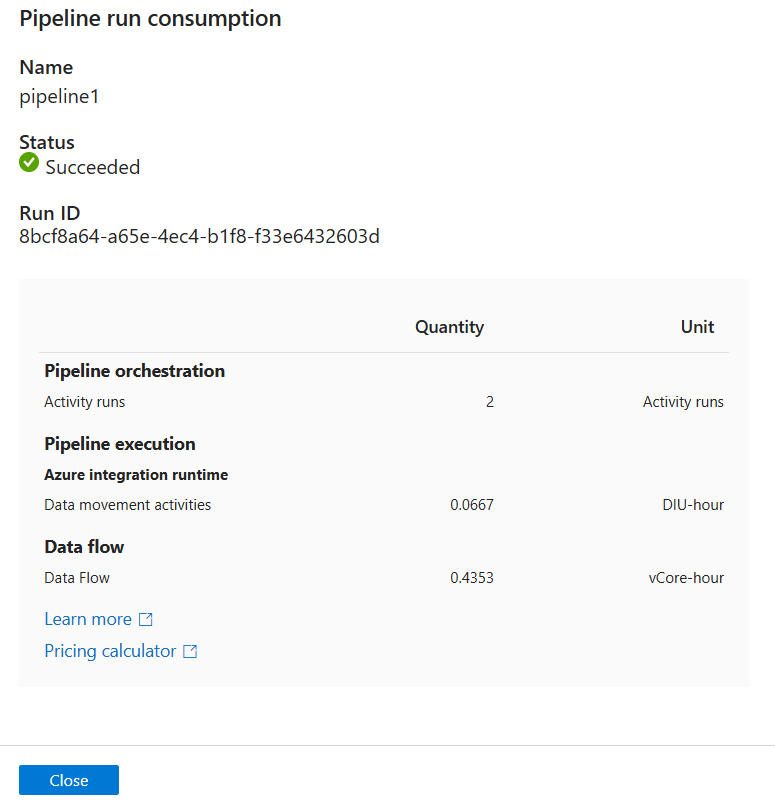


**Data Flow is also running Successfully without Error**

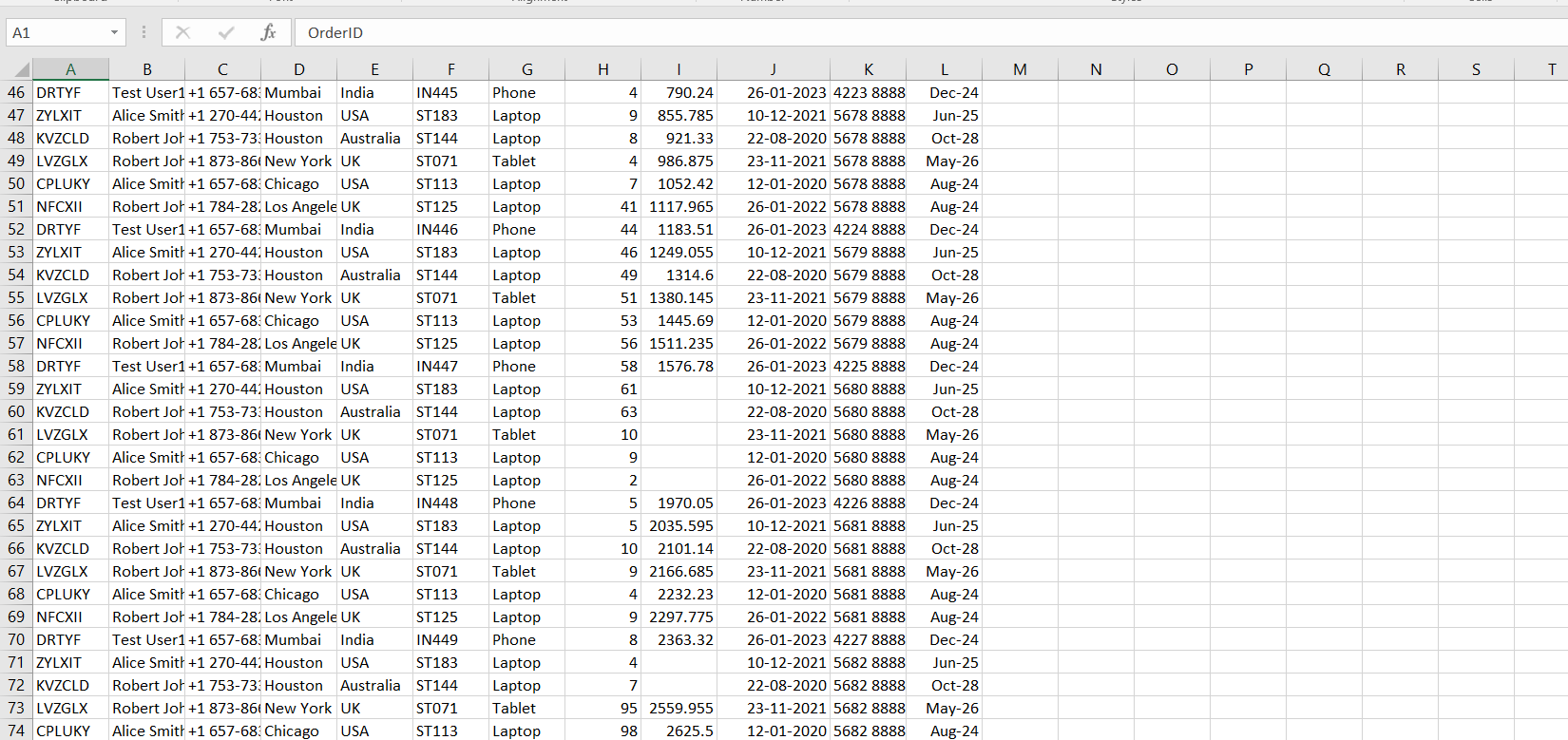




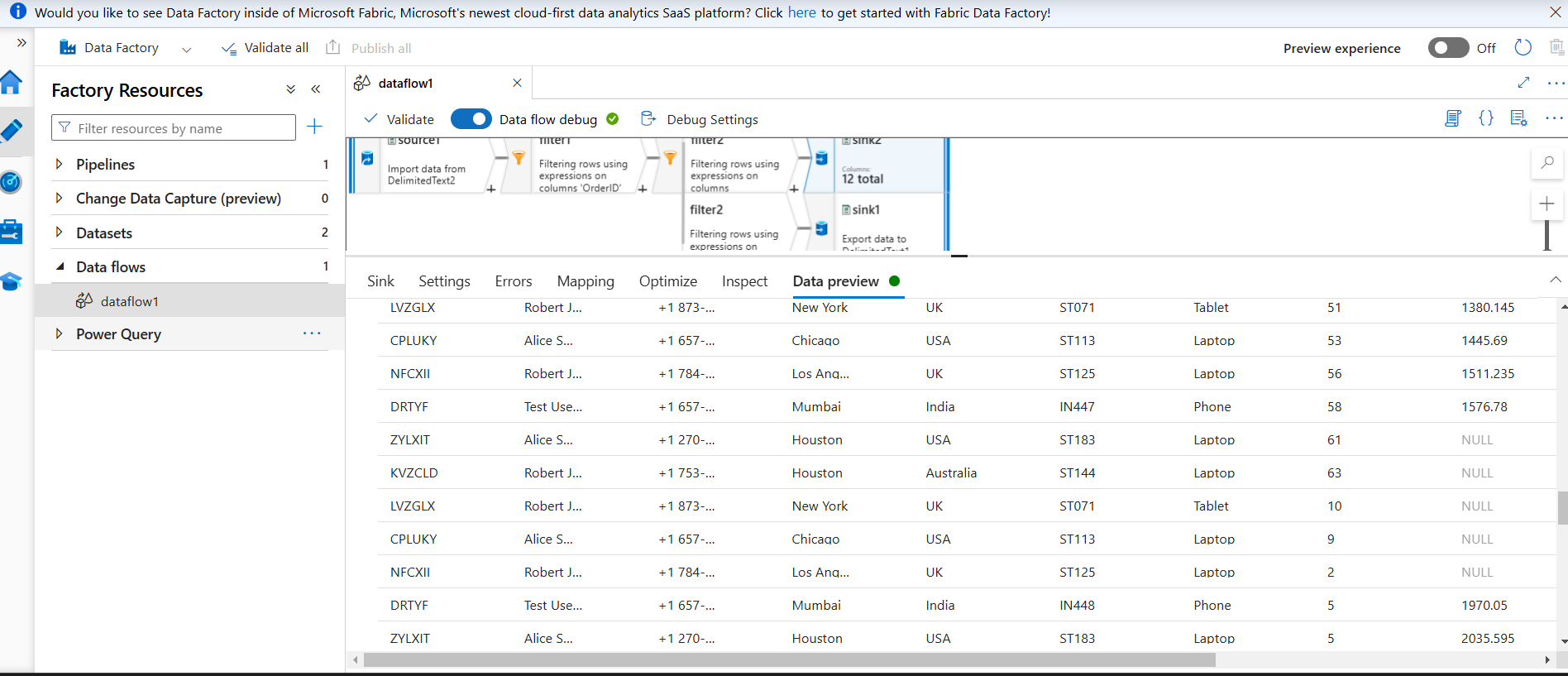
**REPORT OF PIPELINE**



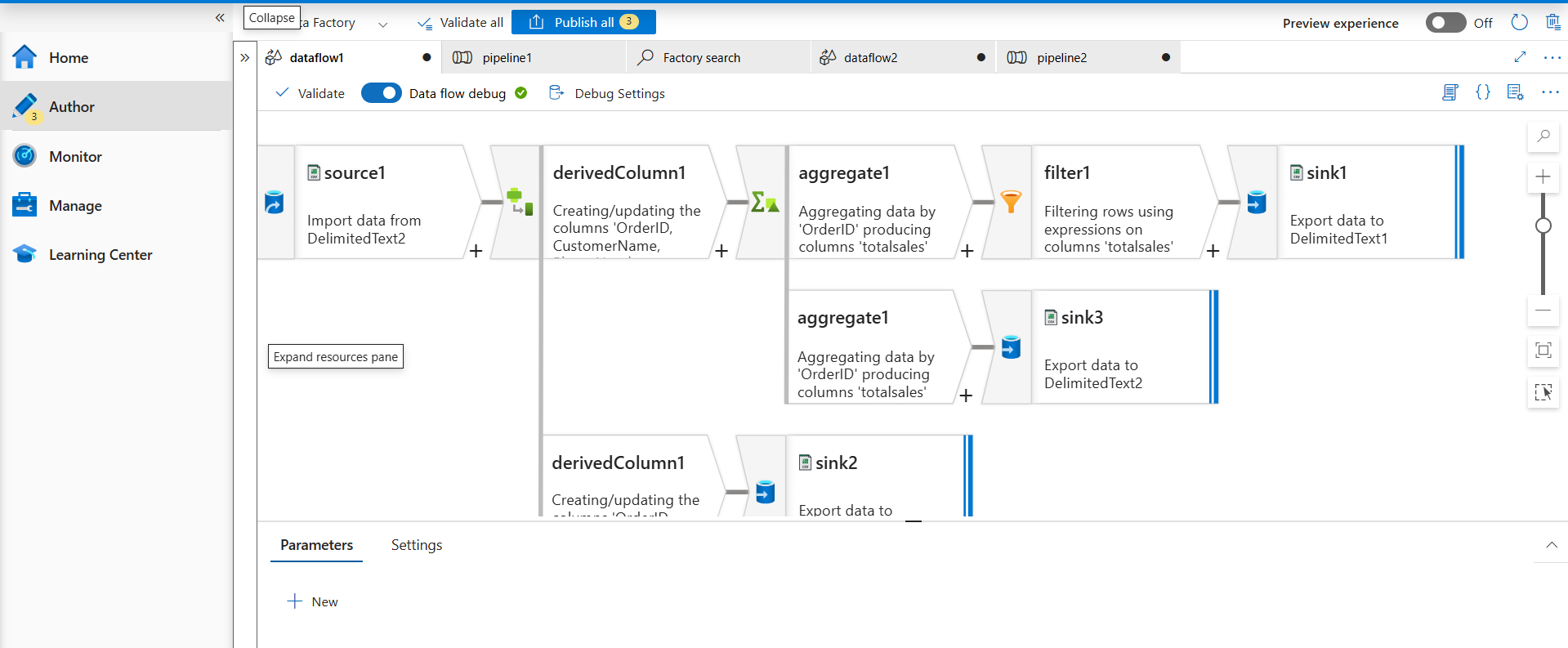
**In given data there is empty spaces**



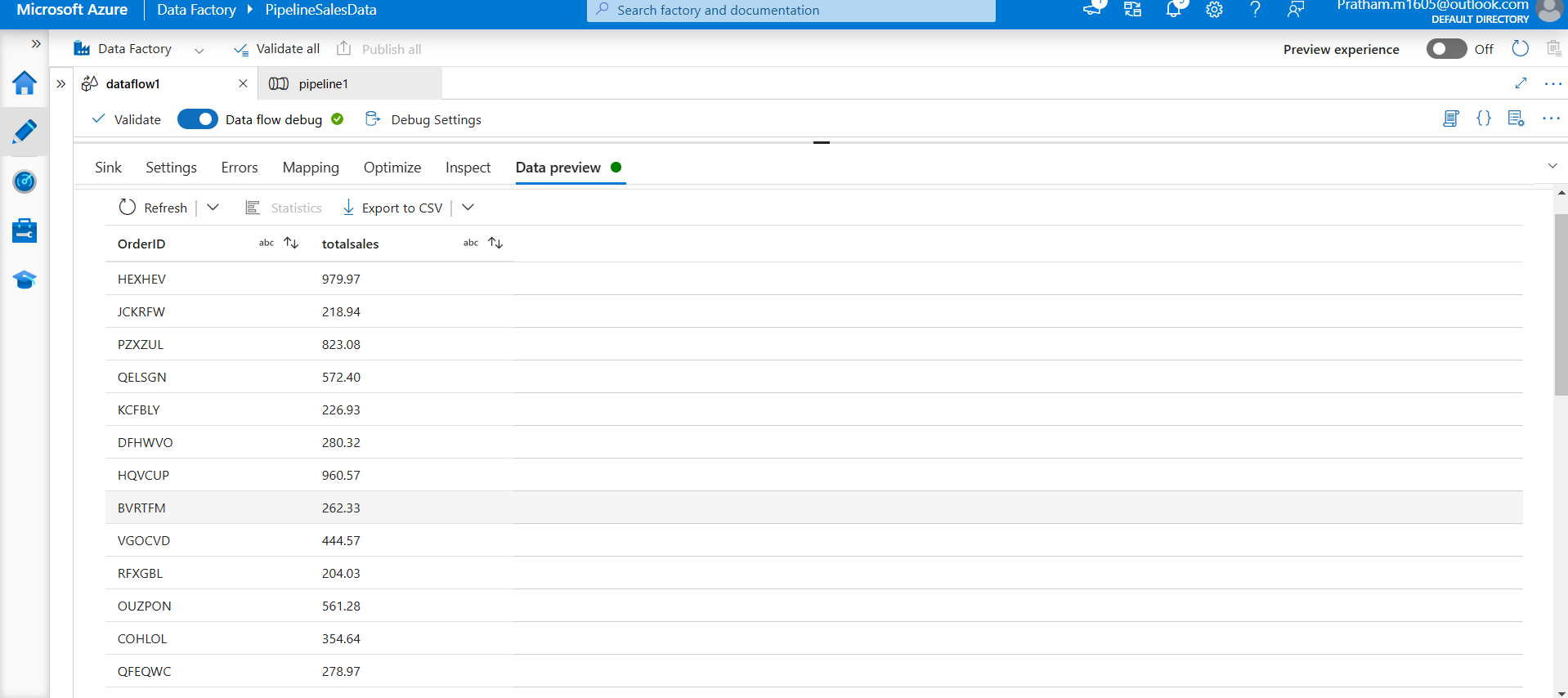
**After Filtered the text null is added to avoid empty spaces. Therefore the Filter which is added is working correctly. And output is displayed on screen**



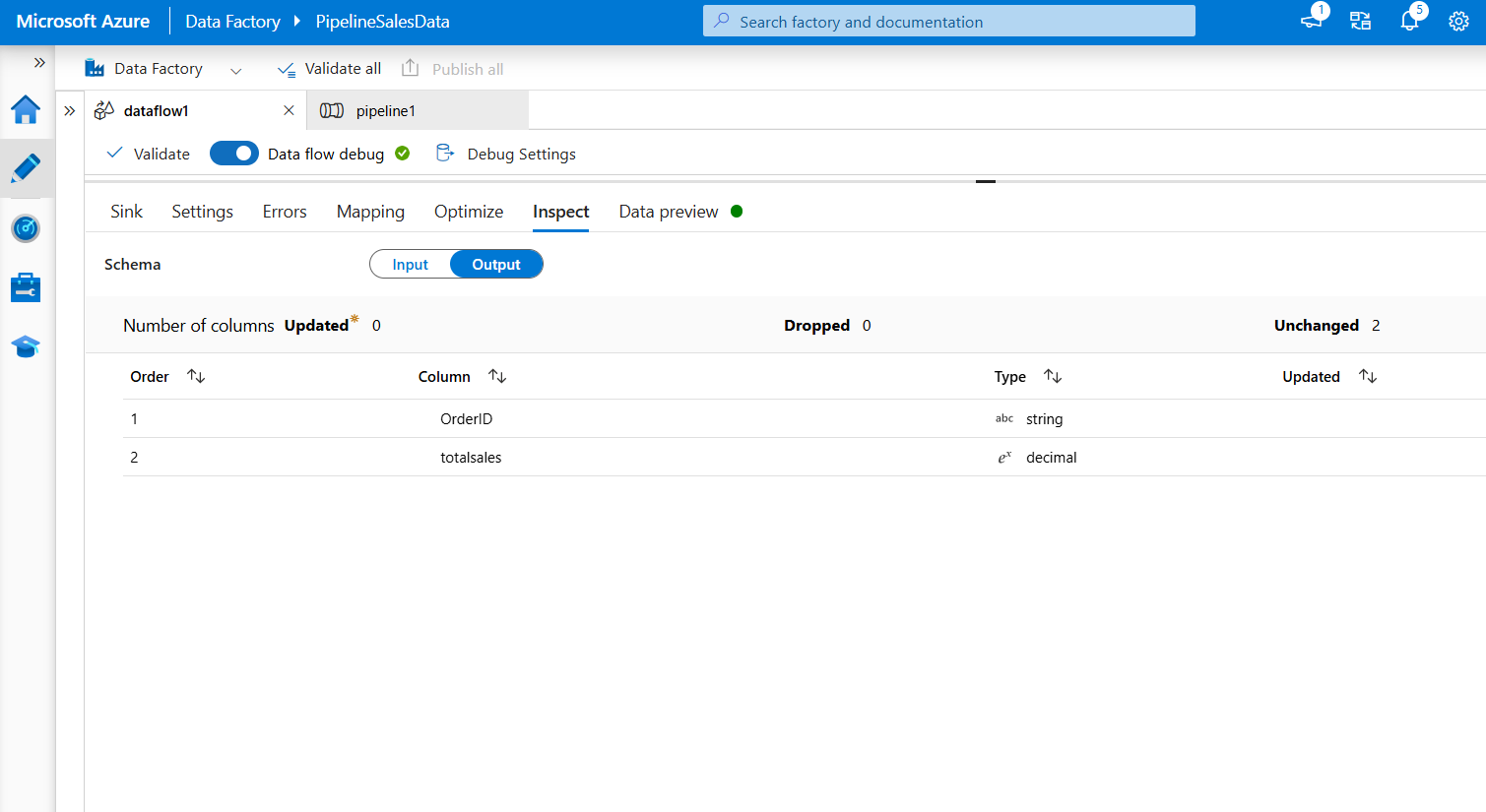
**This final Canvas Connections with DerivedColumn Aggregate and Filter.Everything is connected to sink to get the required output**



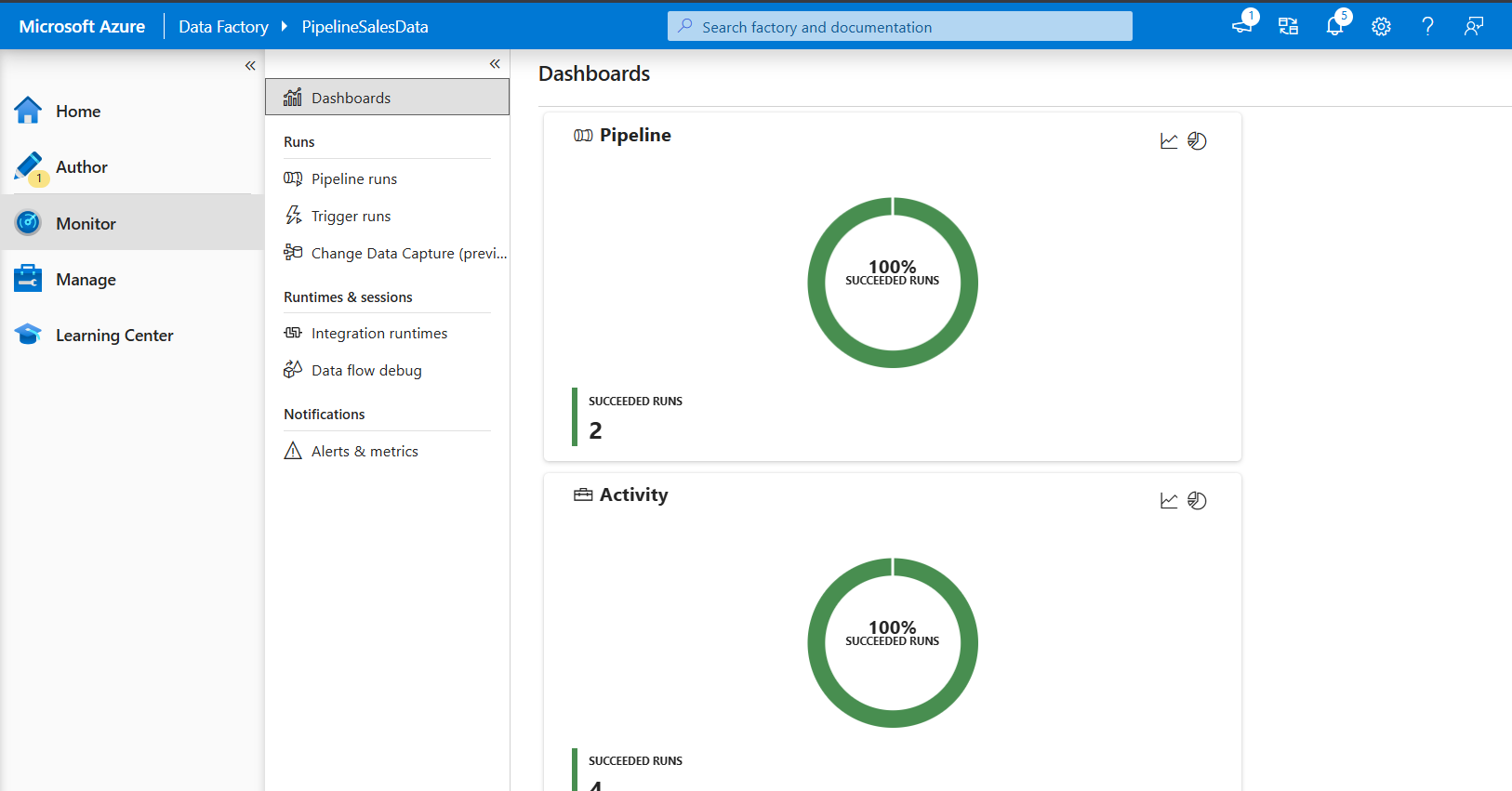
**A new Column is created with totalsales.**



**The created totalsales column is in decimal not in string. So from this we can know that the transformation is done successfully**



**The Report of Monitor is been displayed on screen. The pipeline and Activity is 100% Succeeded.**



**PowerBI Dashboard**

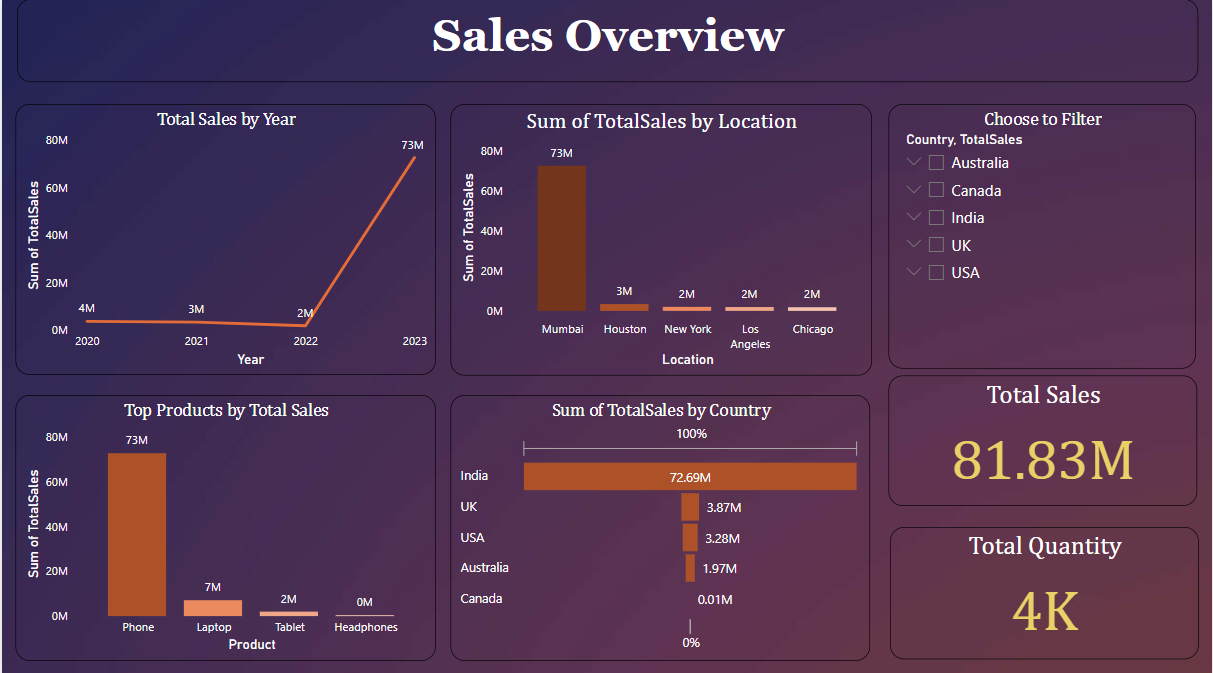
Data Source: Data was uploaded into PowerBI as a CSV file \*(to be changed later)\*

**Dashboard Insights**

Three pages were created for different Insights for:

1. **Sales Overview**
2. **Product Performance**
3. **Customer Insights**

**Sales Overview Insights**



**Graphs used**

1. Total Sales by Year

* Shows Time series analysis Sum of Total Sales year wise using Line chart
* Highest Sales is in year 2023 which is around 73 Million

1. Sales by Location

* Shows Total Sales by Location using Bar chart
* Mumbai shows highest sales of 73 Million amongst other countries

1. Product by Sales

* Shows Total Sales for each product category using Bar chart
* Phone has Highest sales of 72.7 Million precisely
* Laptop has second highest sales of around 7 Million

1. Total Sales by Country

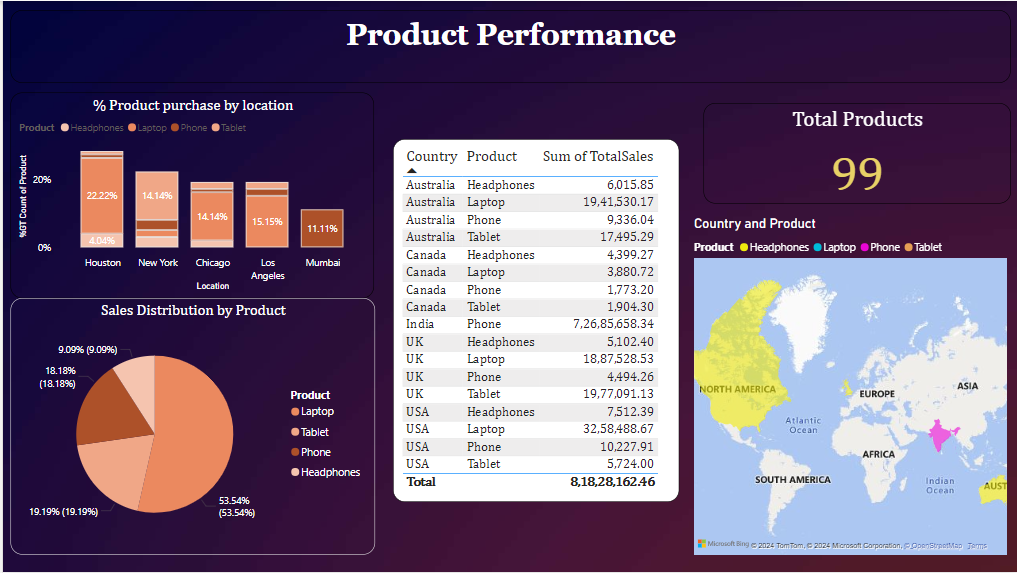
* Shows Country wise total sales using Funnel chart
* India has highest no of sales around 72.6 Million amongst other countries
* UK has second highest no of sales

**Key Performance Indicators (KPIs) used**

1. Total Sales of 82 Million is marked according to data
2. Total Quantity of around 4 K is Sold out

**Filters used :** Slicers are applied country wise to view the trends

**Product Performance Insights**

****

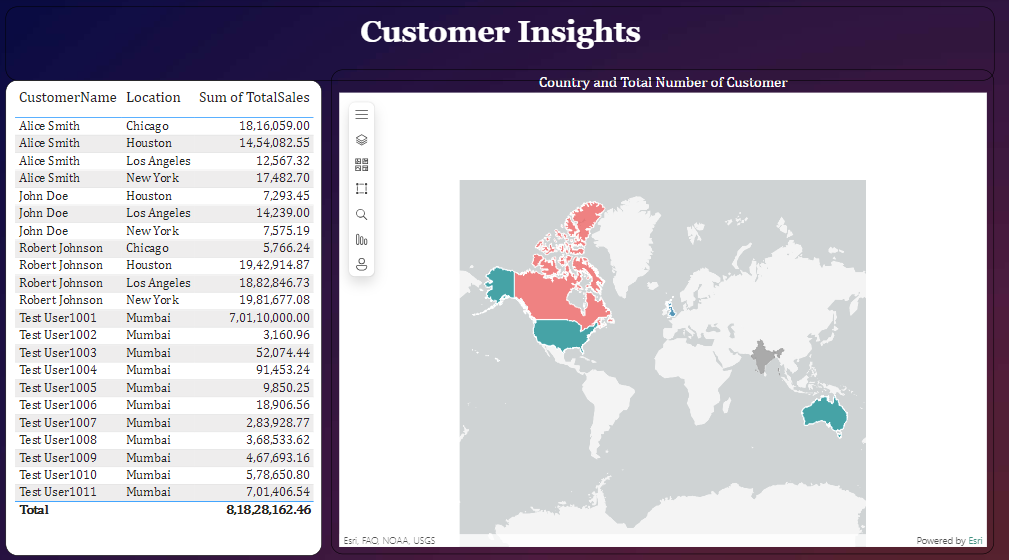
1. Product Purchase % by location

* Shows percentage distribution of different purchased across various locations using Stacked Bar chart.
* **Houston** has the highest percentage of purchases overall, with around **22.22%** of products bought there. It indicates this location has the strongest product sales.
* **New York, Chicago, and Los Angeles** have similar purchase percentages, around **14.14%** to **15.15%.**These locations have moderate and comparable product sales.
* **Mumbai** accounts for the smallest percentage of purchases, around **11.11%.**

1. Sales Distribution of Product

* Shows Total sales distribution for Product using a Pie chart
* Laptop has highest sales

**Customer Insights**

****

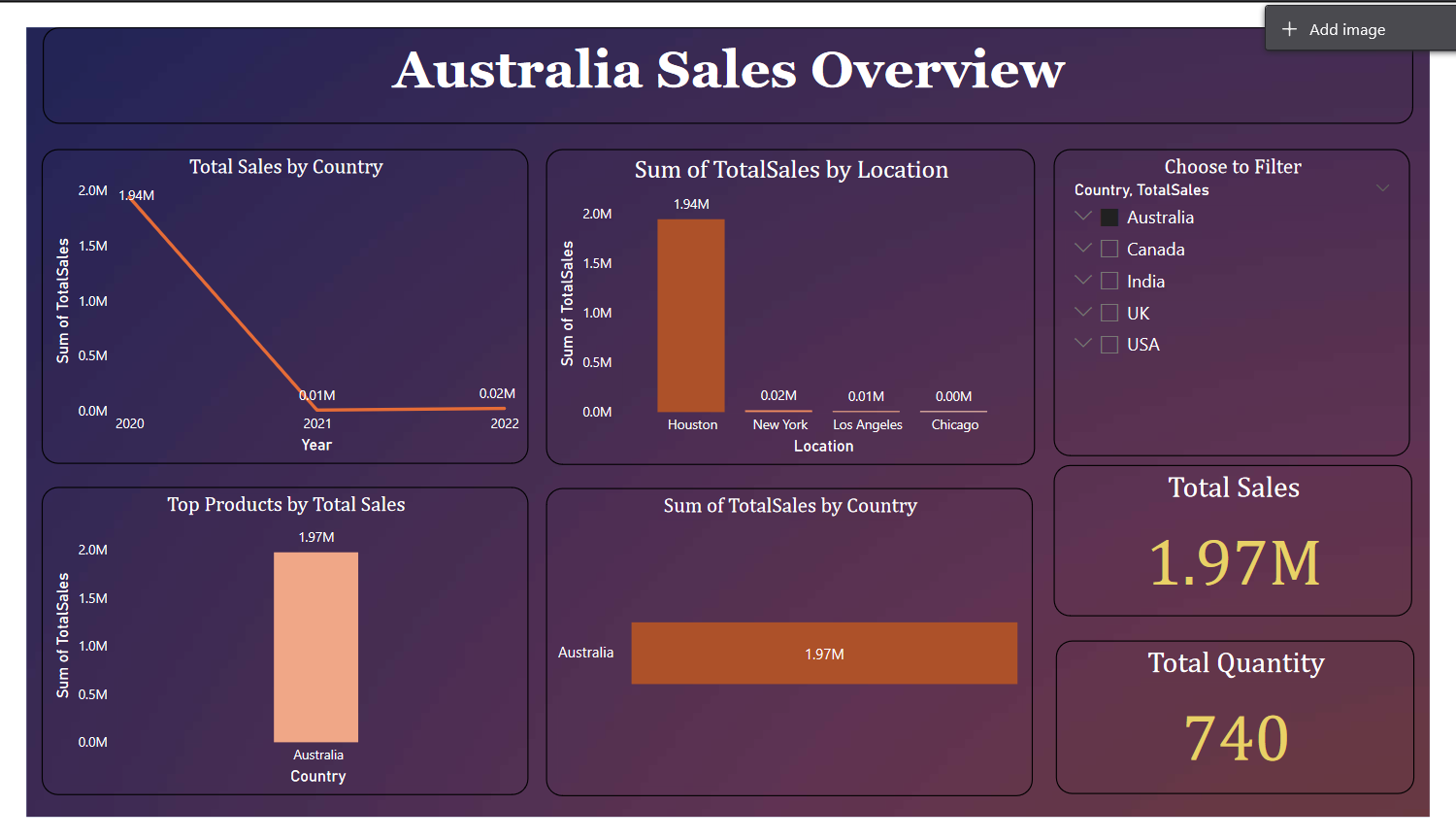
1. Customer Contribution to Sales:

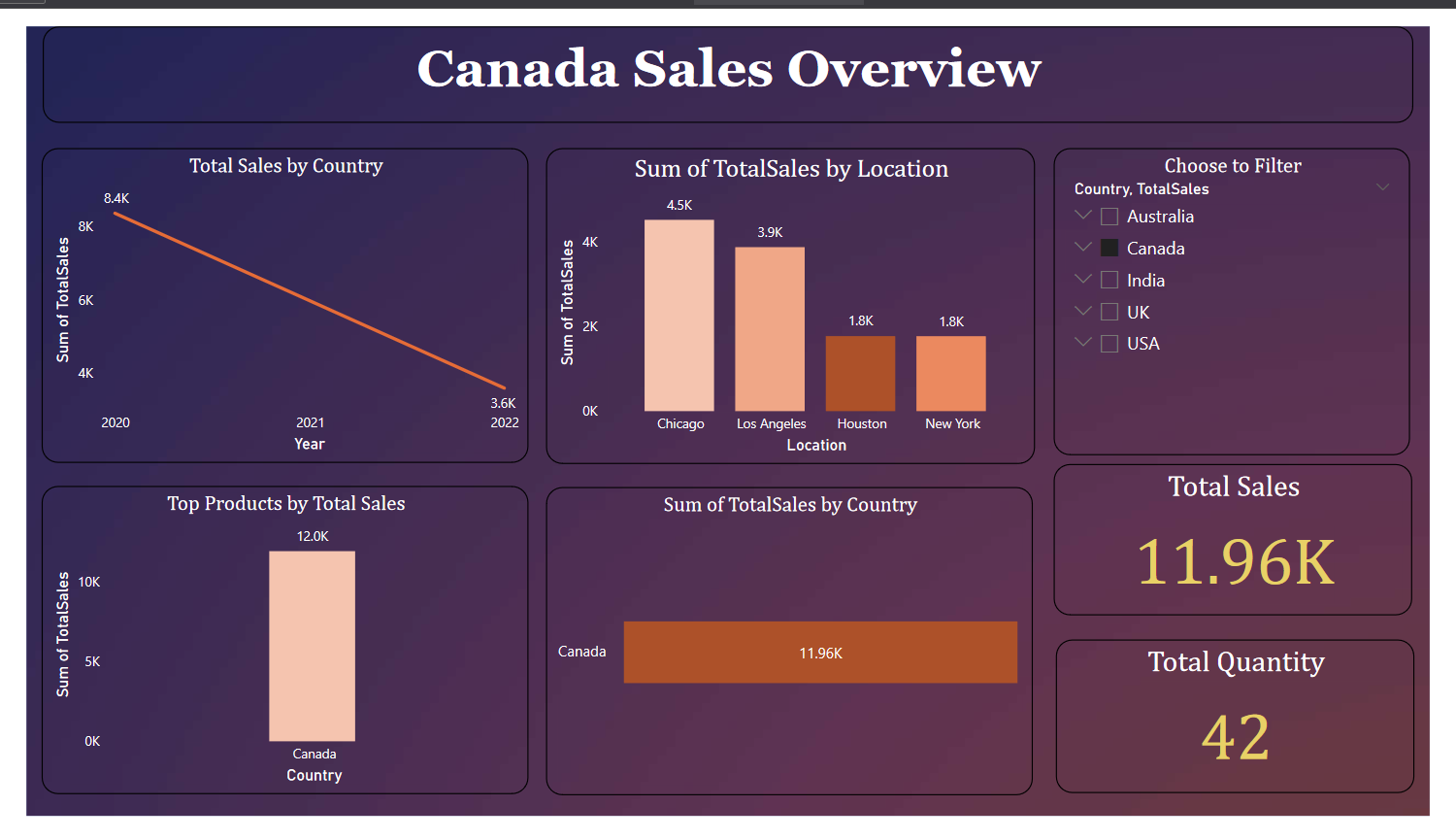
* The table lists customers along with their locations and the Sum of Total Sales they contributed.
* High-value customers include:
  + Alice Smith, with significant sales across multiple locations, including Chicago (₹18,16,059) and Houston (₹14,54,082).
  + Robert Johnson, with notable contributions, especially in Houston (₹19,42,918) and New York (₹19,81,677).

1. Location Insights:

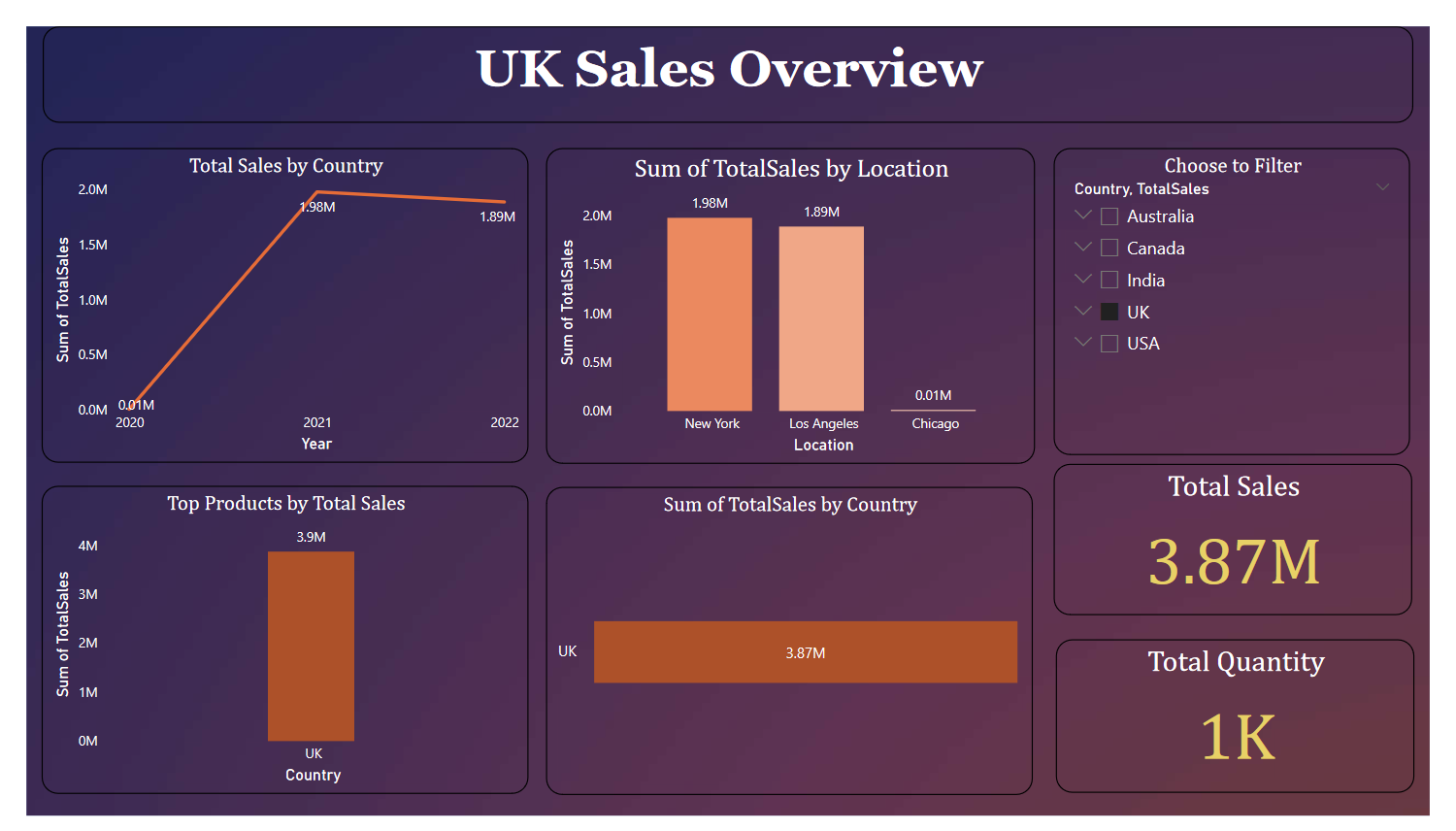
* The map highlights customer distribution by country:
  + USA and India have the highest number of customers.
  + Australia is also represented but with fewer customers
* Locations like Mumbai, Chicago, Houston, and New York show repeated high sales, indicating significant customer activity in these regions and Mumbai, in particular, has several lower-value sales, likely from multiple smaller transactions.

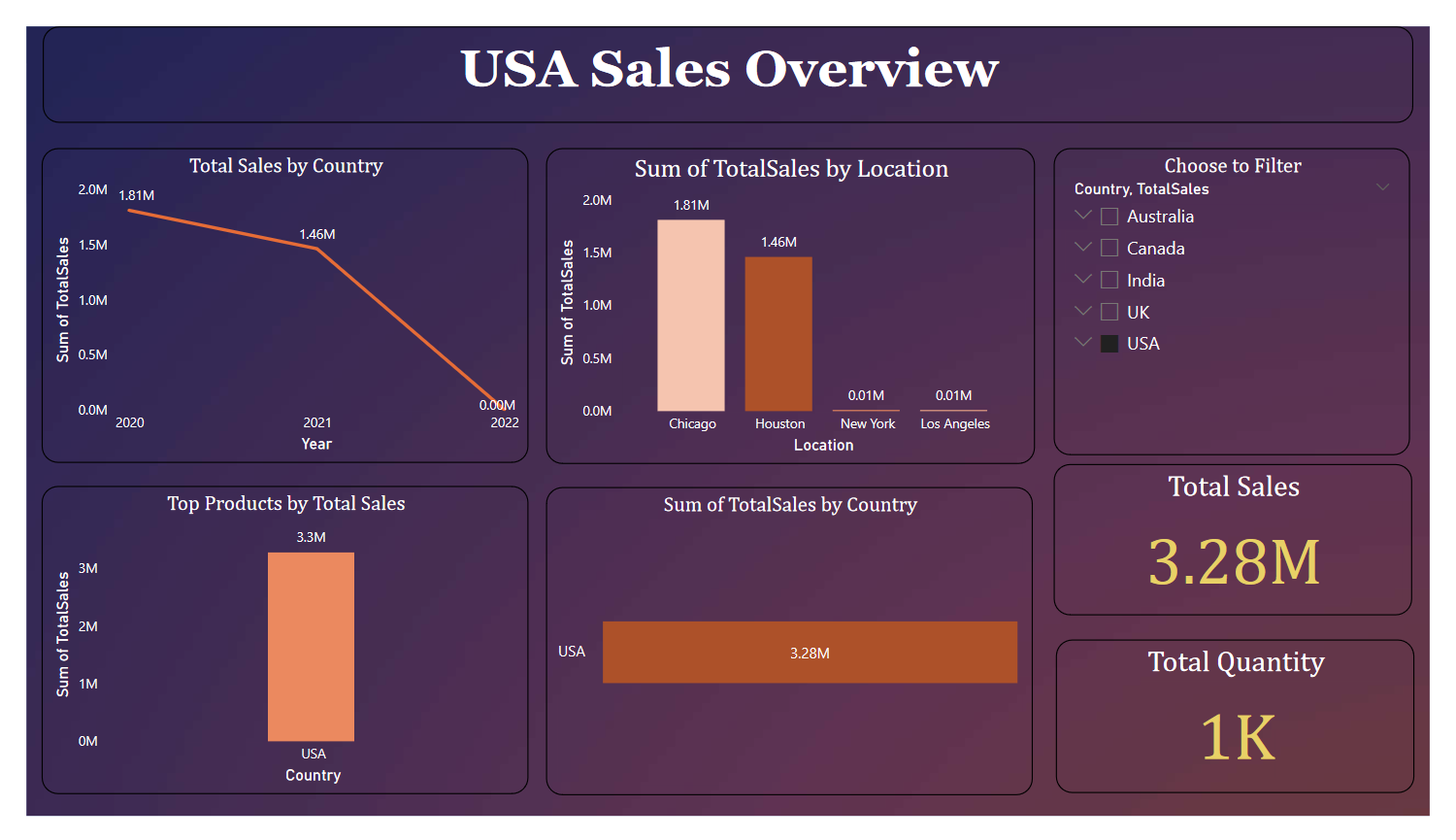
**Full report of PowerBI**

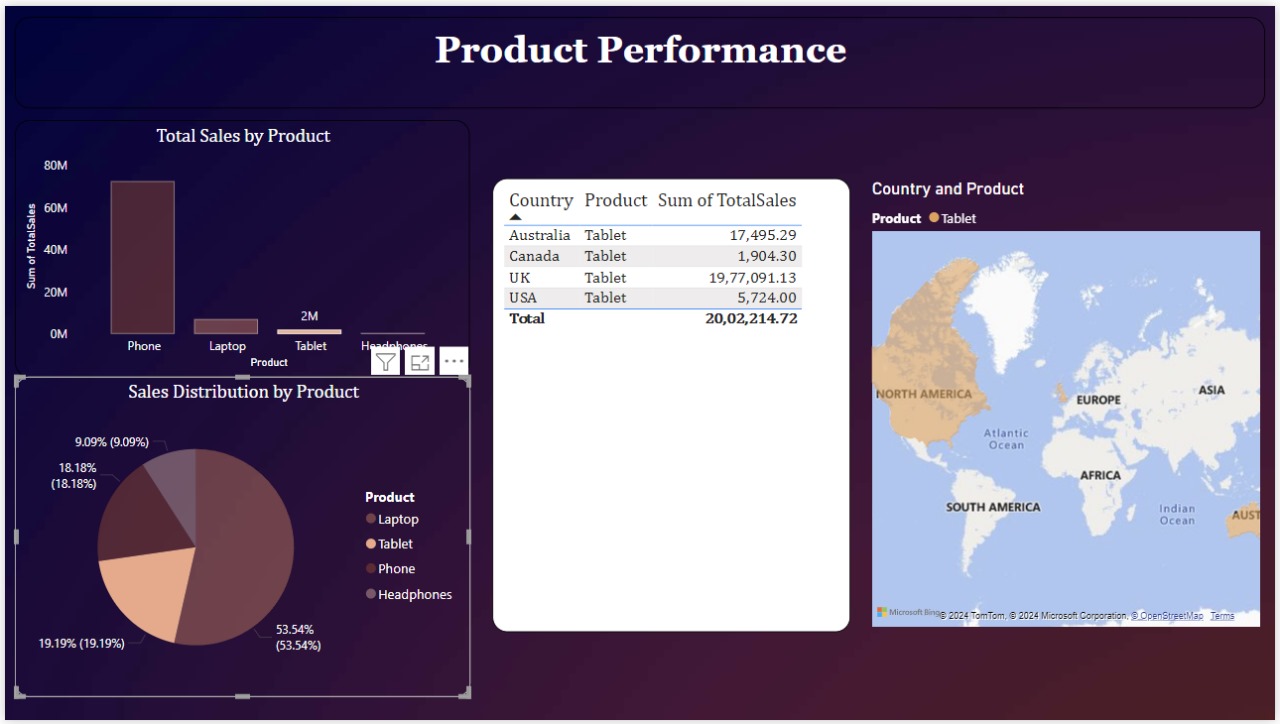
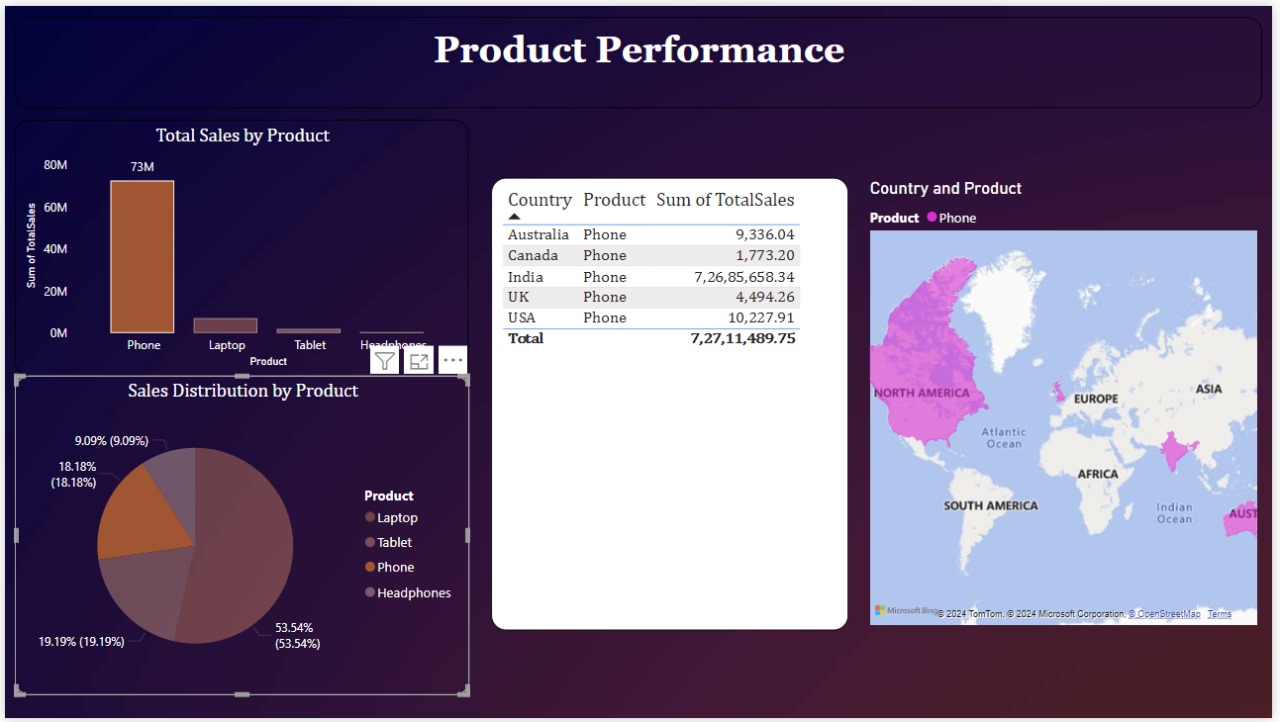
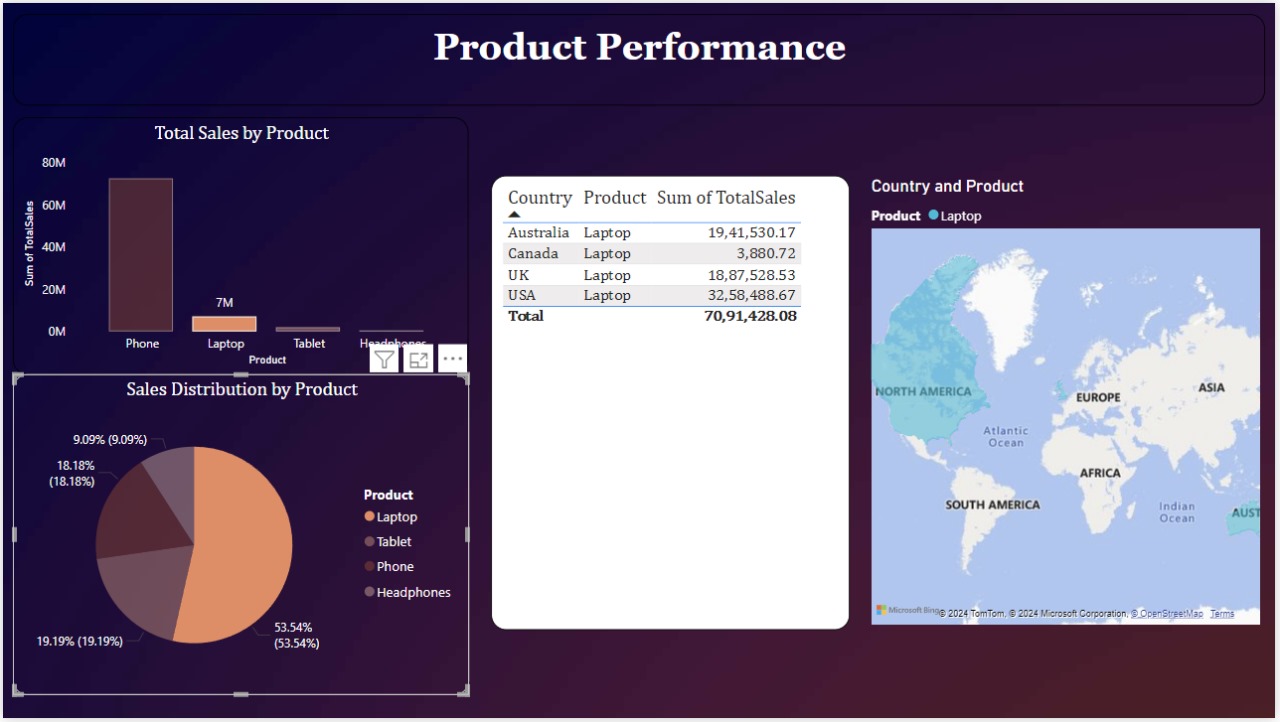


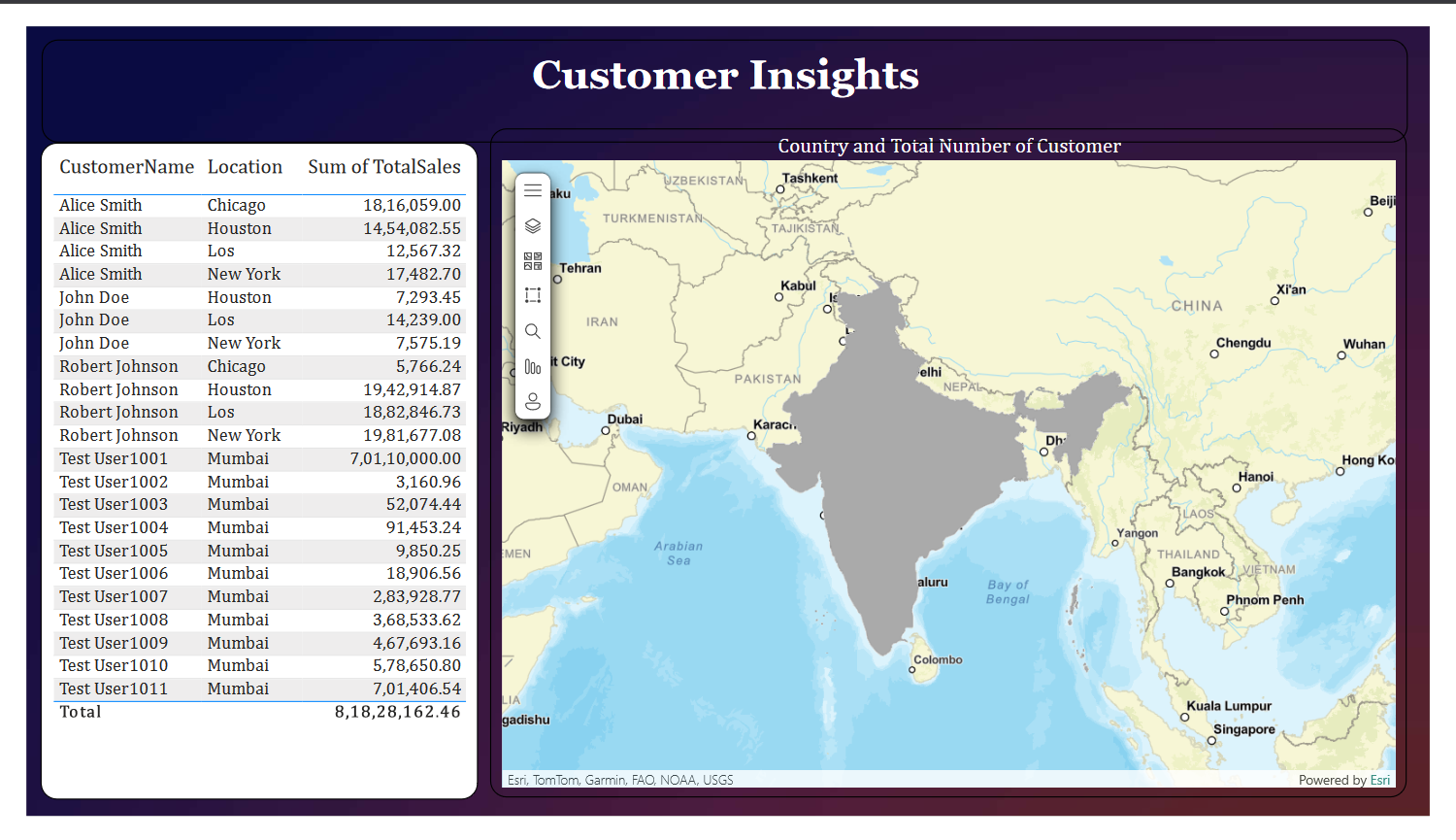










**THANK YOU**