USE CASE - SALES DATA PROCESSING FOR DATA OPTIMISATION

Submitted by:

Thangavignesh T 👦🏻

Data cleaning

Pipeline run ID *: 56643fce-1cfb-411d-aedb-1409b9410f4f*

Data Transformation

Pipeline run ID *: 52bf75c2-fd1b-4bf9-b9d0-4b8aeaa064cb*

Introduction

To address the challenges faced by XYZ Retail Inc. in managing and deriving insights from their sales data, a comprehensive solution was implemented. This solution involved building a robust, end-to-end data pipeline that efficiently handles data ingestion, cleaning, transformation, and visualization. The primary objective was to ensure data quality, enable meaningful insights, and support data-driven decision-making.

Problem Statement:

XYZ Retail Inc., a prominent player in the retail industry, faces the challenge of efficiently managing and analyzing its sales data. The company operates through multiple channels like online platforms, physical stores, and mobile apps. To enhance decision-making, XYZ Retail aims to create an end-to-end data pipeline that ingests, processes, and visualizes sales data.

Solution Overview

1. Proposed Solution
2. Choice of Data Storage
3. Architectural Diagram

Proposed Solution

To address the challenges faced by XYZ Retail Inc., a scalable and efficient end-to-end data pipeline was designed and implemented. The solution comprises the following steps:

1. Data Ingestion

- Raw sales data from multiple sources is ingested into **Azure Data Lake Storage (ADLS)** using **Azure Data Factory (ADF)** pipelines.
- Ensures seamless integration and scalability for managing diverse, high-volume data.

2. Data Cleaning

- Performs quality checks to handle missing values, duplicates, and outliers.
- Ensures data accuracy and reliability for analysis.

3. Data Transformation

- Cleaned data is transformed to compute key metrics, including:
    - **Total Sales**
    - **Average Order Value (AOV)**
    - **Sales by Product and Location**
- **PII data** (e.g., credit card numbers) is securely masked to safeguard sensitive information.

4. Data Storage

- Transformed data is stored in a structured format within **ADLS**, ensuring accessibility for downstream processes.

5. Visualization

- A **Power BI Dashboard** presents insights such as:
    - **Sales Trends**
    - **Regional Performance**
    - **Product Contributions**
- Enables informed, data-driven decision-making.

Choice of Data Storage

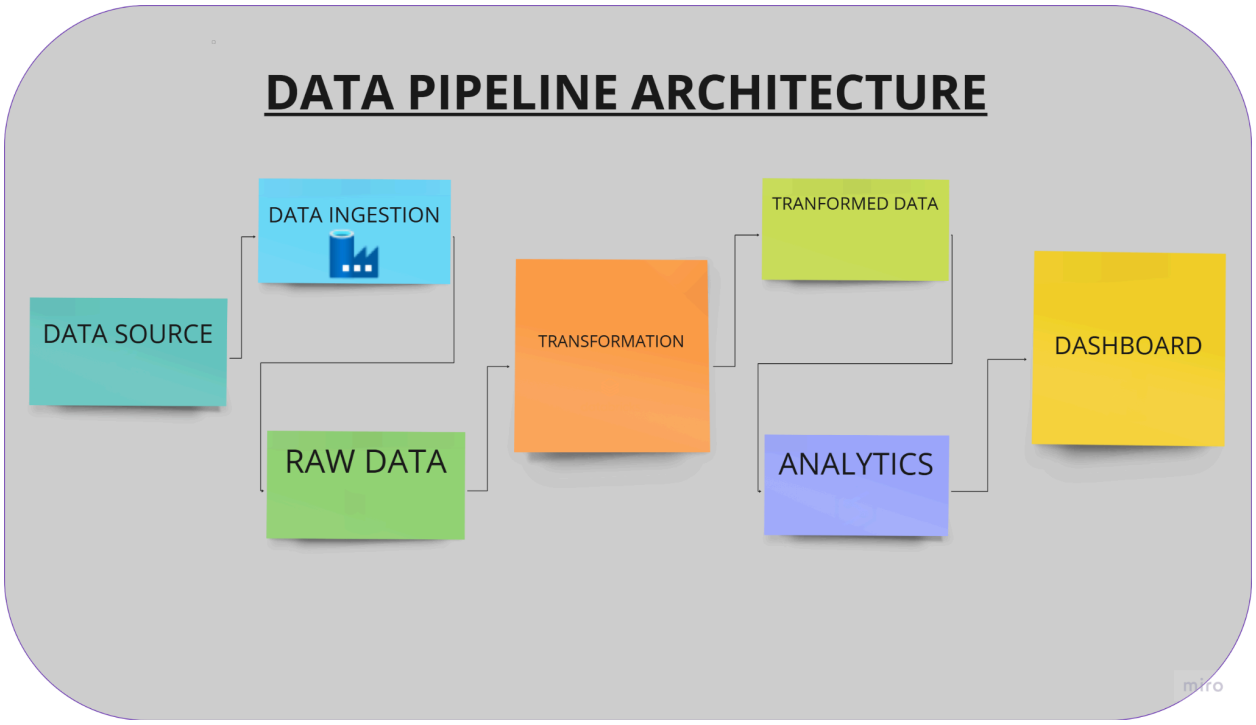
Comparison: Azure Data Lake Storage (ADLS) vs. Azure SQL Database

Feature	Azure Data Lake Storage (ADLS)	Azure SQL Database
Scalability	Designed for large-scale data storage	Limited scalability for high-volume data
Cost	Pay-as-you-go, cost-effective for large datasets	Higher costs for storing large datasets
Flexibility	Supports diverse data types (structured/unstructured)	Primarily for structured data
Integration	Excellent integration with data transformation tools like PySpark	Limited integration with big data tools
Analytics	Built for analytical workloads	Focused on transactional operations

Justification for Selecting ADLS

- **Scalability**: Designed for managing high-volume datasets like sales data.
- **Cost-Effectiveness**: Economical with a pay-as-you-go pricing model, ideal for large datasets.
- **Flexibility**: Supports multiple data types and integrates seamlessly with tools like PySpark for transformations.
- **Analytics**: Tailored for analytical workloads, aligning with the objective of deriving insights from sales data.

Flow Diagram Description



The solution's data flow is organized through the following components:

1. **Azure Data Factory (ADF)**
    - Automates workflows for data ingestion, cleaning, and transformation.
2. **Azure Data Lake Storage (ADLS)**
    - Serves as the central repository for raw, cleaned, and transformed datasets.
3. **Azure Databricks**
    - Executes data cleaning, transformation, and enrichment processes to derive meaningful insights.
4. **Azure Synapse Analytics**
    - Facilitates advanced querying and analytics on the transformed data.
5. **Power BI**
    - Provides interactive dashboards to visualize sales trends and key business metrics, enabling data-driven decision-making.
    -

Code Implementation

Data Ingestion

09:45 AM (26s) 1 Python

```
%python
# Unmount the existing mount point if it exists
if any(mount.mountPoint == '/mnt/tokyoolymic' for mount in dbutils.fs.mounts()):
    dbutils.fs.unmount('/mnt/tokyoolymic')

# Define the configurations
configs = {
    "fs.azure.account.auth.type": "OAuth",
    "fs.azure.account.oauth.provider.type": "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
    "fs.azure.account.oauth2.client.id": "655257ed-7dfd-4bd2-a4f8-a4bff35ef054",
    "fs.azure.account.oauth2.client.secret": 'Oyj8Q~t9u_MzDNAYLU~7j0QRx4uz7BWpF-LZYahl',
    "fs.azure.account.oauth2.client.endpoint": "https://login.microsoftonline.com/e0d9ab7b-9acb-4ee6-9105-7538a44aa880/oauth2/token"
}

# Mount the storage
dbutils.fs.mount(
    source="abfss://slaesdatacontainer@salesdatapipelinestorage.dfs.core.windows.net",
    mount_point="/mnt/tokyoolymic",
    extra_configs=configs
)
```

```
/mnt/tokyoolymic has been unmounted.
True
```

10:23 AM (12s) 8 Python

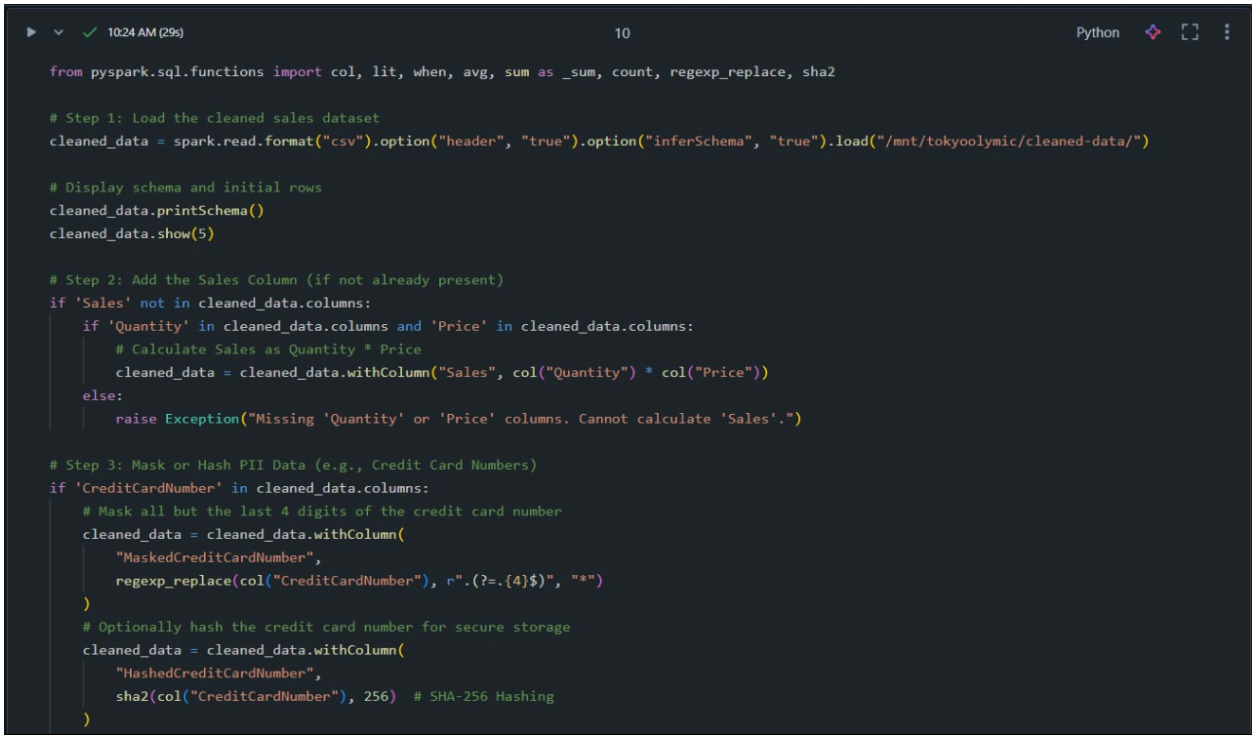
```
from pyspark.sql.functions import col, lit, when, to_date

# Step 1: Load the sales dataset
sales = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("/mnt/tokyoolymic/raw-data/sales.csv")

# Print schema to verify column data types
sales.printSchema()

# Display initial data
sales.show(5)

# Step 2: Add or Create the Sales Column
# Check if 'Quantity' and 'Price' columns exist
if 'Quantity' in sales.columns and 'Price' in sales.columns:
    # Calculate Sales as Quantity * Price
    sales = sales.withColumn("Sales", col("Quantity").cast("double") * col("Price").cast("double"))
else:
    print("Columns 'Quantity' and 'Price' are missing, cannot calculate 'Sales'.")

# Step 3: Handle Missing Data
# Drop rows with missing critical fields
critical_fields = ["OrderID", "Sales", "Date"]
sales = sales.na.drop(subset=critical_fields)

# Fill missing non-critical fields with default values
sales = sales.fillna({
    "CustomerName": "Unknown",
    "PhoneNumber": "000-000-0000",
    "Location": "Unknown",
```

```
    "Country": "Unknown"
})

# Step 4: Remove Duplicates
# Remove duplicate rows based on OrderID
sales = sales.dropDuplicates(["OrderID"])

# Step 5: Validate and Correct Data Types
# Convert Date column to date type and drop rows with invalid dates
sales = sales.withColumn("Date", to_date(col("Date"), "yyyy-MM-dd")).na.drop(subset=["Date"])

# Ensure Sales column is numeric
sales = sales.withColumn("Sales", col("Sales").cast("double"))

# Step 6: Handle Outliers
# Replace negative or unrealistic Sales values with 0
sales = sales.withColumn("Sales", when(col("Sales") < 0, 0).otherwise(col("Sales")))

# Optional: Remove rows with extreme sales values (e.g., greater than 1,000,000)
sales = sales.filter(col("Sales") <= 1000000)

# Step 7: Save the Cleaned Dataset in the Cleaned Data Container
# Coalesce to 1 partition to ensure a single output file
sales.coalesce(1).write.format("csv").mode("overwrite").option("header", "true").save("dbfs:/mnt/tokyoolymic/cleaned-data/")

print("Cleaned data with 'Sales' column saved successfully as a single CSV file in dbfs:/mnt/tokyoolymic/cleaned-data/")

▸ (5) Spark Jobs
```

OUTPUT

```
▸ sales: pyspark.sql.dataframe.DataFrame = [OrderID: string, CustomerName: string ... 11 more fields]
 |-- Country: string (nullable = true)
 |-- StoreCode: string (nullable = true)
 |-- Product: string (nullable = true)
 |-- Quantity: integer (nullable = true)
 |-- Price: double (nullable = true)
 |-- Date: date (nullable = true)
 |-- CreditCardNumber: string (nullable = true)
 |-- ExpiryDate: string (nullable = true)

+-------+------------+---------------+-----------+-------+---------+----------+--------+------+----------+------------------+----------+
|OrderID|CustomerName|    PhoneNumber|   Location|Country|StoreCode|   Product|Quantity| Price|      Date|  CreditCardNumber|ExpiryDate|
+-------+------------+---------------+-----------+-------+---------+----------+--------+------+----------+------------------+----------+
| HEXHEV|    John Doe|+1 990-186-7268|   New York|    USA|    ST026|     Phone|       5|979.97|2021-07-18|5676 8888 7887 7263|    Jul-29|
| JCKRFW|    John Doe|+1 659-832-6831|   New York|    USA|    ST154|Headphones|       1|218.94|2020-02-01|5676 8888 7887 8526|    Jan-23|
| PZXZUL|    John Doe|+1 564-127-5258|    Houston|    USA|    ST013|Headphones|       4|823.08|2020-10-11|5676 8888 7887 1823|    Mar-29|
| QELSGN|    John Doe|+1 571-789-2219|Los Angeles|    USA|    ST029|    Tablet|      10| 572.4|2021-08-17|5676 8888 7887 1242|    Nov-23|
| KCFBLY| Alice Smith|+1 631-561-3575|    Chicago|     UK|    ST140|    Laptop|       6|226.93|2021-03-21|5676 8888 7887 9764|    Feb-27|
+-------+------------+---------------+-----------+-------+---------+----------+--------+------+----------+------------------+----------+
only showing top 5 rows

Cleaned data with 'Sales' column saved successfully as a single CSV file in dbfs:/mnt/tokyoolymic/cleaned-data/
```

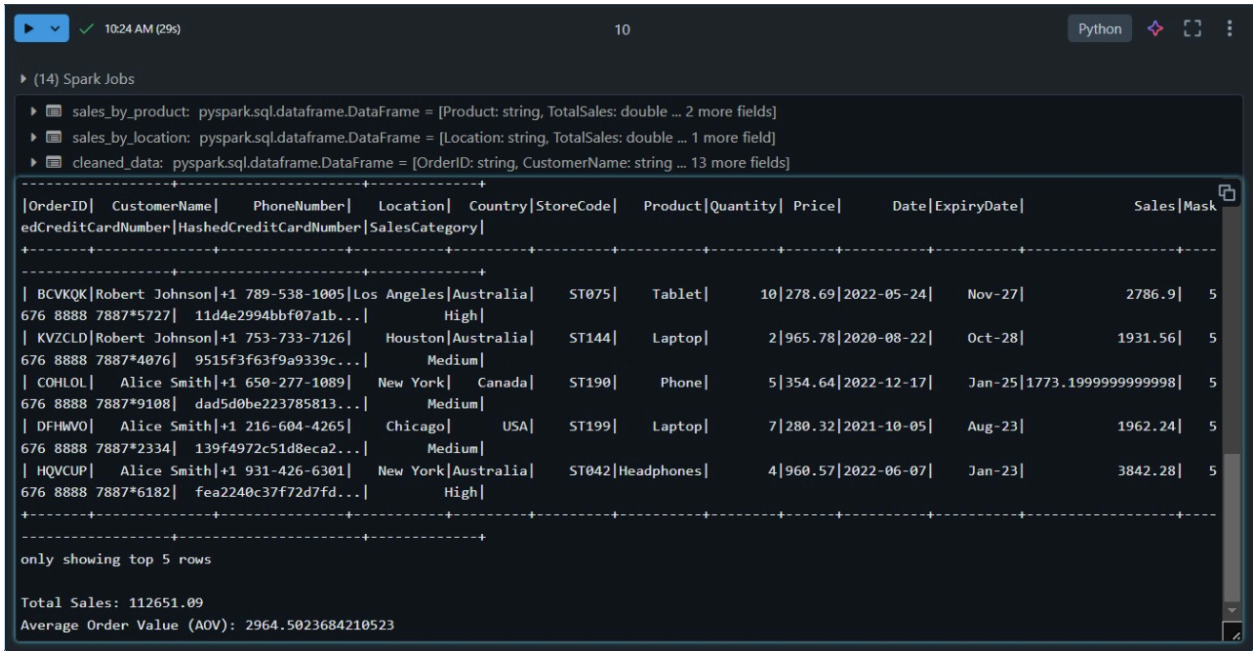
DATA TRANSFORMATION AND ENRICHMENT



```
10:24 AM (29s)                                    10                                    Python

from pyspark.sql.functions import col, lit, when, avg, sum as _sum, count, regexp_replace, sha2

# Step 1: Load the cleaned sales dataset
cleaned_data = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("/mnt/tokyoolymic/cleaned-data/")

# Display schema and initial rows
cleaned_data.printSchema()
cleaned_data.show(5)

# Step 2: Add the Sales Column (if not already present)
if 'Sales' not in cleaned_data.columns:
    if 'Quantity' in cleaned_data.columns and 'Price' in cleaned_data.columns:
        # Calculate Sales as Quantity * Price
        cleaned_data = cleaned_data.withColumn("Sales", col("Quantity") * col("Price"))
    else:
        raise Exception("Missing 'Quantity' or 'Price' columns. Cannot calculate 'Sales'.")

# Step 3: Mask or Hash PII Data (e.g., Credit Card Numbers)
if 'CreditCardNumber' in cleaned_data.columns:
    # Mask all but the last 4 digits of the credit card number
    cleaned_data = cleaned_data.withColumn(
        "MaskedCreditCardNumber",
        regexp_replace(col("CreditCardNumber"), r".(?=.{4}$)", "*")
    )
    # Optionally hash the credit card number for secure storage
    cleaned_data = cleaned_data.withColumn(
        "HashedCreditCardNumber",
        sha2(col("CreditCardNumber"), 256)  # SHA-256 Hashing
    )
```

```
10:24 AM (29s)                                    10                                    Python

# Step 5: Add Sales Category Based on Sales Value
cleaned_data = cleaned_data.withColumn(
    "SalesCategory",
    when(col("Sales") < 500, "Low")
    .when((col("Sales") >= 500) & (col("Sales") < 2000), "Medium")
    .otherwise("High")
)

# Step 6: Save the Transformed Dataset in the Transformed Data Container
# Coalesce to 1 partition to ensure a single output file
cleaned_data.coalesce(1).write.format("csv").mode("overwrite").option("header", "true").save("dbfs:/mnt/tokyoolymic/transformed-data/")

# Save grouped metrics for further analysis
sales_by_product.write.format("csv").mode("overwrite").option("header", "true").save("dbfs:/mnt/tokyoolymic/metrics/sales_by_product/")
sales_by_location.write.format("csv").mode("overwrite").option("header", "true").save("dbfs:/mnt/tokyoolymic/metrics/sales_by_location/")

# Display final transformed dataset
cleaned_data.show(5)

# Print metrics
print(f"Total Sales: {total_sales}")
print(f"Average Order Value (AOV): {average_order_value}")
```

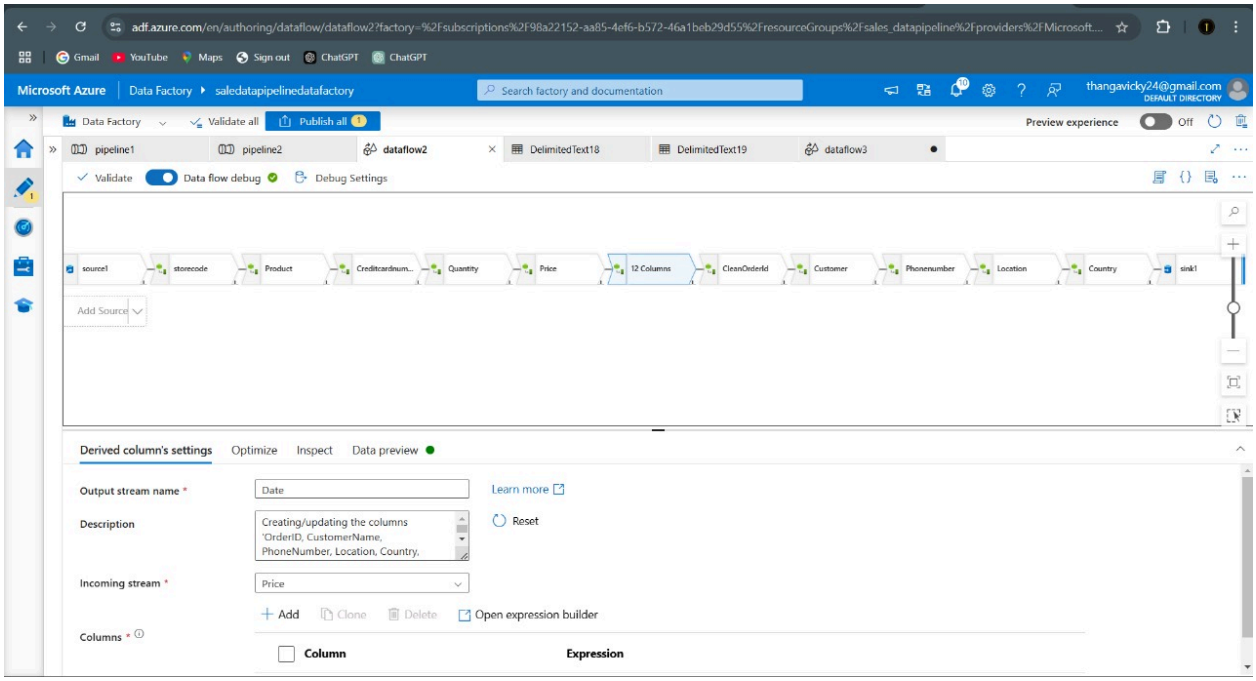
OUTPUT



```
10:24 AM (29s)                                  10                                  Python

(14) Spark Jobs
  sales_by_product: pyspark.sql.dataframe.DataFrame = [Product: string, TotalSales: double ... 2 more fields]
  sales_by_location: pyspark.sql.dataframe.DataFrame = [Location: string, TotalSales: double ... 1 more field]
  cleaned_data: pyspark.sql.dataframe.DataFrame = [OrderID: string, CustomerName: string ... 13 more fields]

+-------+--------------+---------------+-----------+---------+---------+----------+--------+------+----------+----------+------------------+----
|OrderID|  CustomerName|    PhoneNumber|   Location|  Country|StoreCode|   Product|Quantity| Price|      Date|ExpiryDate|             Sales|Mask
edCreditCardNumber|HashedCreditCardNumber|SalesCategory|
+-------+--------------+---------------+-----------+---------+---------+----------+--------+------+----------+----------+------------------+----
------------------+----------------------+-------------+
| BCVKQK|Robert Johnson|+1 789-538-1005|Los Angeles|Australia|    ST075|    Tablet|      10|278.69|2022-05-24|    Nov-27|            2786.9|   5
676 8888 7887*5727|  11d4e2994bbf07a1b...|         High|
| KVZCLD|Robert Johnson|+1 753-733-7126|    Houston|Australia|    ST144|    Laptop|       2|965.78|2020-08-22|    Oct-28|           1931.56|   5
676 8888 7887*4076|  9515f3f63f9a9339c...|       Medium|
| COHLOL|   Alice Smith|+1 650-277-1089|   New York|   Canada|    ST190|     Phone|       5|354.64|2022-12-17|    Jan-25|1773.1999999999998|   5
676 8888 7887*9108|  dad5d0be223785813...|       Medium|
| DFHWVO|   Alice Smith|+1 216-604-4265|    Chicago|      USA|    ST199|    Laptop|       7|280.32|2021-10-05|    Aug-23|           1962.24|   5
676 8888 7887*2334|  139f4972c51d8eca2...|       Medium|
| HQVCUP|   Alice Smith|+1 931-426-6301|   New York|Australia|    ST042|Headphones|       4|960.57|2022-06-07|    Jan-23|           3842.28|   5
676 8888 7887*6182|  fea2240c37f72d7fd...|         High|
+-------+--------------+---------------+-----------+---------+---------+----------+--------+------+----------+----------+------------------+----
------------------+----------------------+-------------+
only showing top 5 rows

Total Sales: 112651.09
Average Order Value (AOV): 2964.5023684210523
```

Data Cleaning

Data Flow



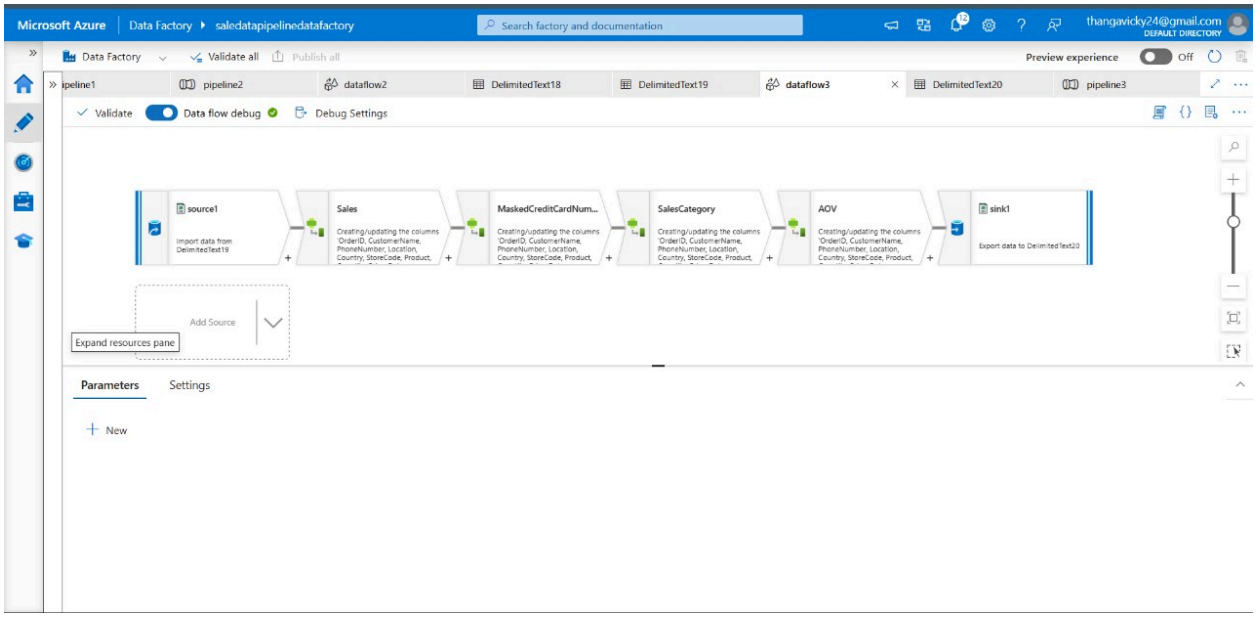
Script

```
source(output(
		OrderID as string,
		CustomerName as string,
		PhoneNumber as string,
		Location as string,
		Country as string,
		StoreCode as string,
		Product as string,
		Quantity as string,
		Price as string,
		Date as string,
		CreditCardNumber as string,
		ExpiryDate as string
	),
	allowSchemaDrift: true,
	validateSchema: false,
	ignoreNoFilesFound: false) ~> source1
Date derive(OrderID = iif(isNull(OrderID) || trim(OrderID) == '', 'Unknown',
OrderID)) ~> CleanOrderId
CleanOrderId derive(CustomerName = iif(isNull(CustomerName) ||
trim(CustomerName) == '', 'Unknown', CustomerName)) ~> Customer
Customer derive(PhoneNumber = iif(isNull(PhoneNumber) || trim(PhoneNumber)
== '', 'Unknown', PhoneNumber)) ~> Phonenumber
Phonenumber derive(Location = iif(isNull(Location) || trim(Location) == '',
'Unknown', Location)) ~> Location
Location derive(Country = iif(isNull(Country) || trim(Country) == '', 'Unknown',
Country)) ~> Country
source1 derive(StoreCode = iif(isNull(StoreCode) || trim(StoreCode) == '',
'Unknown', StoreCode)) ~> storecode
storecode derive(Product = iif(isNull(Product) || trim(Product) == '', 'Unknown',
Product)) ~> Product
```

```
Product derive(CreditCardNumber = iif(isNull(CreditCardNumber) ||
trim(CreditCardNumber) == '', 'Unknown', CreditCardNumber)) ~>
Creditcardnumber
Creditcardnumber derive(Quantity = iif(isNull(trim(Quantity)) || trim(Quantity) ==
'', '10', Quantity)) ~> Quantity
Quantity derive(Price = iif(isNull(trim(Price)) || trim(Price) == '', '100', Price)) ~>
Price
Price derive(Date = iif(isNull(toDate(Date, 'yyyy-MM-dd')), toDate('1970-01-01',
'yyyy-MM-dd'), toDate(Date, 'yyyy-MM-dd'))) ~> Date
Country sink(allowSchemaDrift: true,
        validateSchema: false,
        input(
                OrderID as string,
                CustomerName as string,
                PhoneNumber as string,
                Location as string,
                Country as string,
                StoreCode as string,
                Product as string,
                Quantity as string,
                Price as string,
                Date as string,
                CreditCardNumber as string,
                ExpiryDate as string
        ),
        partitionFileNames:['cleaned_dataset.csv'],
        umask: 0022,
        preCommands: [],
        postCommands: [],
        skipDuplicateMapInputs: true,
        skipDuplicateMapOutputs: true,
        partitionBy('hash', 1)) ~> sink1
```

Data Transformation

Data Flow



Script

```
source(output(
                OrderID as string,
                CustomerName as string,
                PhoneNumber as string,
                Location as string,
                Country as string,
                StoreCode as string,
                Product as string,
                Quantity as string,
                Price as string,
                Date as string,
                CreditCardNumber as string,
                ExpiryDate as string
        ),
        allowSchemaDrift: true,
        validateSchema: false,
        ignoreNoFilesFound: false) ~> source1
source1 derive(Sales = toDouble(Quantity) * toDouble(Price)) ~> Sales
```

```
Sales derive(MaskedCreditCardNumber = iif(isNull(CreditCardNumber), '',
concat('xxxx-xxxx-xxxx-', substring(CreditCardNumber, length(CreditCardNumber) - 4, 4)))) ~>
MaskedCreditCardNumber
MaskedCreditCardNumber derive(SalesCategory = iif(Sales < 500, "Low", iif(Sales >= 500 &&
Sales < 2000, "Medium", "High"))) ~> SalesCategory
SalesCategory derive(AverageOrderValue = Sales / toFloat(Quantity)) ~> AOV
AOV sink(allowSchemaDrift: true,
        validateSchema: false,
        input(
                OrderID as string,
                CustomerName as string,
                PhoneNumber as string,
                Location as string,
                Country as string,
                StoreCode as string,
                Product as string,
                Quantity as string,
                Price as string,
                Date as string,
                CreditCardNumber as string,
                ExpiryDate as string
        ),
        partitionFileNames:['transformdataset.csv'],
        umask: 0022,
        preCommands: [],
        postCommands: [],
        skipDuplicateMapInputs: true,
        skipDuplicateMapOutputs: true,
        partitionBy('hash', 1)) ~> sink1
```

Dataset Documentation

Dataset Structure

The sales dataset serves as the foundation for all data processing and analysis tasks. Below is the detailed structure of the dataset:

Column Name	Data Type	Description
OrderID	String	Unique identifier for each order.
CustomerName	String	Name of the customer who placed the order.
PhoneNumber	String	Contact number of the customer.
Location	String	Geographic location of the order (city or region).
Country	String	Country where the order was placed.
StoreCode	String	Unique identifier for the store.
Product	String	Name of the product ordered.
Quantity	Integer	Quantity of the product ordered.
Price	Double	Unit price of the product.
Date	Date	Date when the order was placed.
Sales	Double	Total sales for the order (Quantity * Price).
SalesCategory	String	Categorization of sales (Low, Medium, High).
MaskedCreditCardNumber	String	Encrypted or masked credit card numbers for security.

Data Preprocessing Steps

The preprocessing of the sales dataset ensures data accuracy, quality, and readiness for downstream analysis. The steps include:

1. **Data Ingestion**
    - **Raw Data**: Sales data ingested into Azure Data Lake Storage (ADLS) using Azure Data Factory pipelines.
    - **Storage**: Initial dataset stored in the `raw-data` container within ADLS for cleaning and transformation.
2. **Handling Missing Values**
    - **Critical Fields**: Rows with missing values in fields such as `OrderID`, `Sales`, and `Date` were dropped to maintain integrity.
3. **Removing Duplicates**
    - Duplicate records were removed based on the unique `OrderID` column to prevent redundancy.
4. **Correcting Data Types**
    - **Date Conversion**: `Date` column converted to the `Date` type using the format `yyyy-MM-dd`.

Sales Calculation: New `Sales` column created as:
makefile
Copy code
```
Sales = Quantity * Price
```

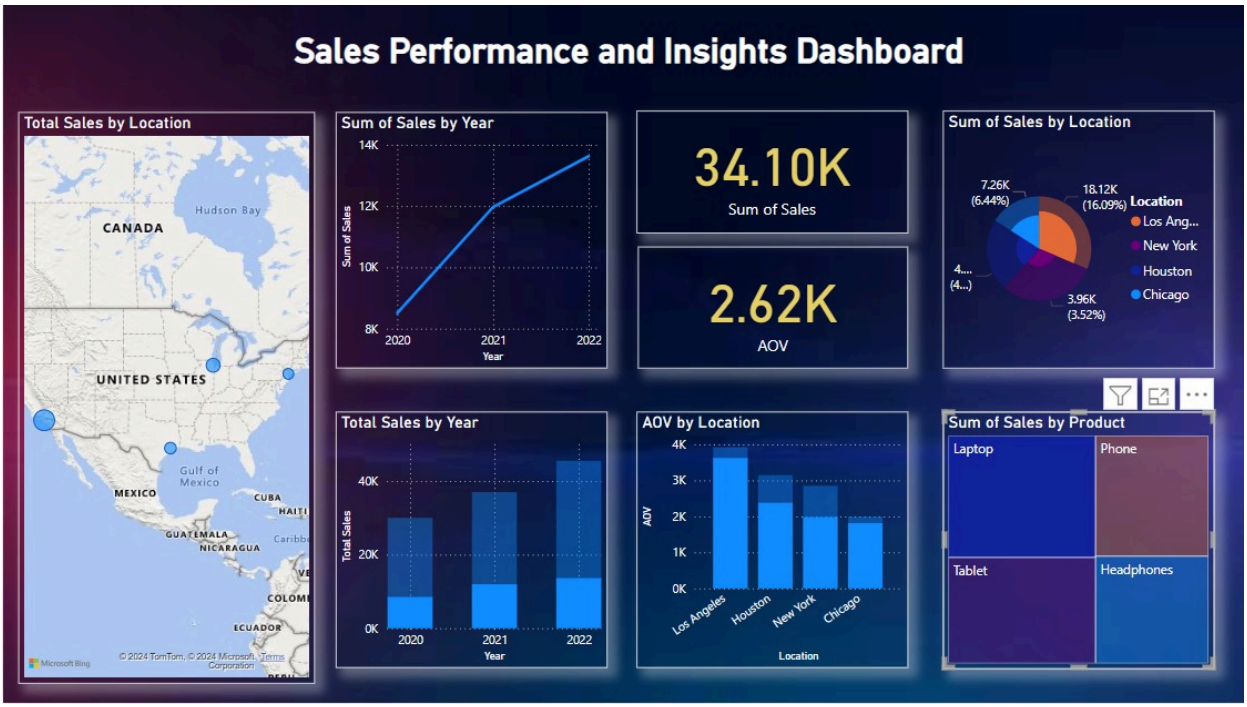
5. **Data Transformation**
    - Computed key business metrics, including:
        - **Total Sales**
        - **Average Order Value (AOV)**
        - Grouped metrics by **Product** and **Location**.
    - Stored transformed data for analytical use.
6. **Masking PII Data**
    - **Credit Card Numbers**: Masked sensitive information using **SHA-256 hashing** to ensure secure handling of Personally Identifiable Information (PII).
    - **Output**: Stored masked values in the `MaskedCreditCardNumber` column while excluding original data.

Explanation of the Sales Performance and Insights Dashboard

The **Sales Performance and Insights Dashboard** provides a visual representation of key sales metrics, trends, and regional performance to support data-driven decision-making.

Key Metrics:

- **Total Sales**: The dashboard highlights a total sales value .
- **Average Order Value (AOV)**: The average revenue per order.



Visuals Overview:

1. Total Sales by Location (Map)

- **Insight**: The map visual illustrates sales distribution geographically.
    - Larger circles represent regions with higher sales volumes.
    - **Key Locations**:**Los Angeles** and **New York** contribute significantly to sales, identifying them as primary markets for marketing efforts or inventory optimization.

2. Sum of Sales by Year (Line Chart)

- **Insight**: The consistent year-over-year growth in sales indicates a positive trajectory.
    - This suggests successful strategies in areas like product offerings, customer engagement, or market expansion.

- Further analysis is recommended to pinpoint specific growth drivers.

3. Total Sales and AOV (Card Visuals)

- **Insight**:
  - **Total Sales**: **34.10K**, reflecting the overall revenue during the observed period.
  - **Average Order Value (AOV)**: **2.62K**, indicating strong customer spending per transaction.
  - This highlights efficient sales strategies and a potentially loyal customer base.

4. Sum of Sales by Location (Pie Chart)

- **Insight**: The pie chart breaks down sales contributions by location.
  - **Los Angeles** leads with **18.12K (16.09%)**, followed by New York and Houston.
  - Cities like **Chicago**, with smaller contributions, may represent opportunities for targeted growth initiatives or promotional campaigns.

5. Total Sales by Year (Bar Chart)

- **Insight**: The bar chart emphasizes the growth pattern more explicitly.
  - Each year's total sales progression reinforces the scalability of the current business strategies.

6. AOV by Location (Bar Chart)

- **Insight**: The AOV varies significantly across locations.
  - **Los Angeles** and **Houston** have the highest AOVs, showing higher spending per transaction in these areas.
  - **Chicago**, with a lower AOV, may benefit from strategies like product bundling or discounts to increase transaction values.

7. Sum of Sales by Product (Treemap)

- **Insight**: The treemap highlights product contributions to total sales.
  - **Phones** and **Laptops** dominate sales, likely due to their popularity or higher price points.
  - **Headphones** and **Tablets** contribute less, signaling a need for promotions, repositioning, or reassessment of these product lines.

Challenges and Solutions

During the implementation of the data pipeline and dashboard, several challenges were encountered. Below is a summary of the key challenges and their corresponding solutions:

1. **Handling Missing Data**
    - **Challenge**: Missing values in critical fields (e.g., `CustomerName`, `PhoneNumber`, `Location`) compromised data accuracy.
    - **Solution**:
        - Dropped rows with missing critical fields (`OrderID`, `Sales`, `Date`).
        - Filled non-critical fields with default values (e.g., `"Unknown"` for `Location` and `"000-000-0000"` for `PhoneNumber`).
2. **Duplicate Records**
    - **Challenge**: Duplicate `OrderID` records caused inaccuracies in metrics like Total Sales and AOV.
    - **Solution**: Removed duplicate records based on `OrderID` in the data transformation pipeline.
3. **Outliers in Sales Data**
    - **Challenge**: Unrealistic Sales values (e.g., negatives or values >1,000,000)metrics.
    - **Solution**:
        - Replaced negative values with `0` using conditional transformations.
        - Excluded records with Sales values above 1,000,000.
4. **Protecting Personally Identifiable Information (PII)**
    - **Challenge**: Sensitive data, such as credit card numbers, posed security risks.
    - **Solution**:
        - Masked credit card numbers to display only the last four digits.
        - Applied SHA-256 hashing for secure storage of sensitive information.
5. **Visualization Challenges**
    - **Challenge**: Designing an intuitive Power BI dashboard while maintaining simplicity.
    - **Solution**:
        - Grouped visuals into categories(e.g., Sales Trends, Regional Performance)
        - Added slicers and filters for interactivity and customization.
6. **Performance Optimization**
    - **Challenge**: Slow performance with large datasets during transformations and visualizations.
    - **Solution**:
        - Optimized PySpark scripts for in-memory processing.

Conclusion

This project successfully demonstrated the development and implementation of a scalable, efficient, and secure data pipeline utilizing Azure services to transform raw sales data into meaningful insights. By leveraging Azure Data Factory for seamless data ingestion, Azure Data Lake Storage for centralized storage, and PySpark for advanced data transformation, the pipeline effectively handled large datasets while maintaining data integrity and accuracy.

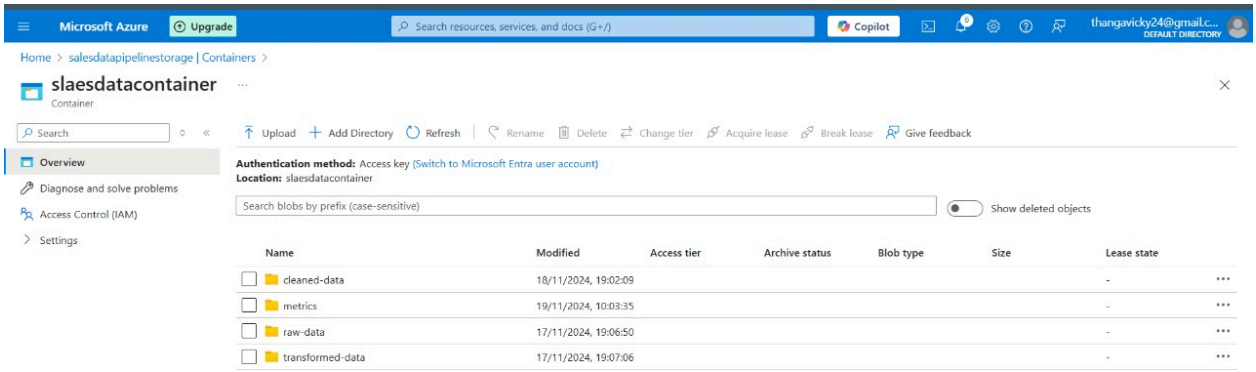
Key achievements of the project include:

1. **Improved Data Quality:**
    - The project enhanced data reliability through rigorous cleaning, deduplication, and outlier handling, ensuring accurate and consistent inputs for analysis.
2. **Actionable Insights through Visualizations:**
    - The Power BI dashboard provided detailed insights into sales trends, regional performance, and product contributions, enabling data-driven decision-making.
3. **Data Security and Privacy:**
    - Sensitive Personally Identifiable Information (PII), such as credit card details, was masked and hashed to meet stringent data security and compliance requirements.
4. **Scalability and Future Readiness:**
    - The adoption of Azure cloud-based solutions ensured that the pipeline is highly scalable, ready to accommodate increasing data volumes and complexity in future use cases.

This project not only achieved its primary objectives of delivering accurate, actionable insights but also established a robust foundation for future data engineering and analytics efforts. The integration of advanced data processing techniques and intuitive visualization tools reinforces the organization’s ability to leverage data for strategic decision-making and operational excellence.

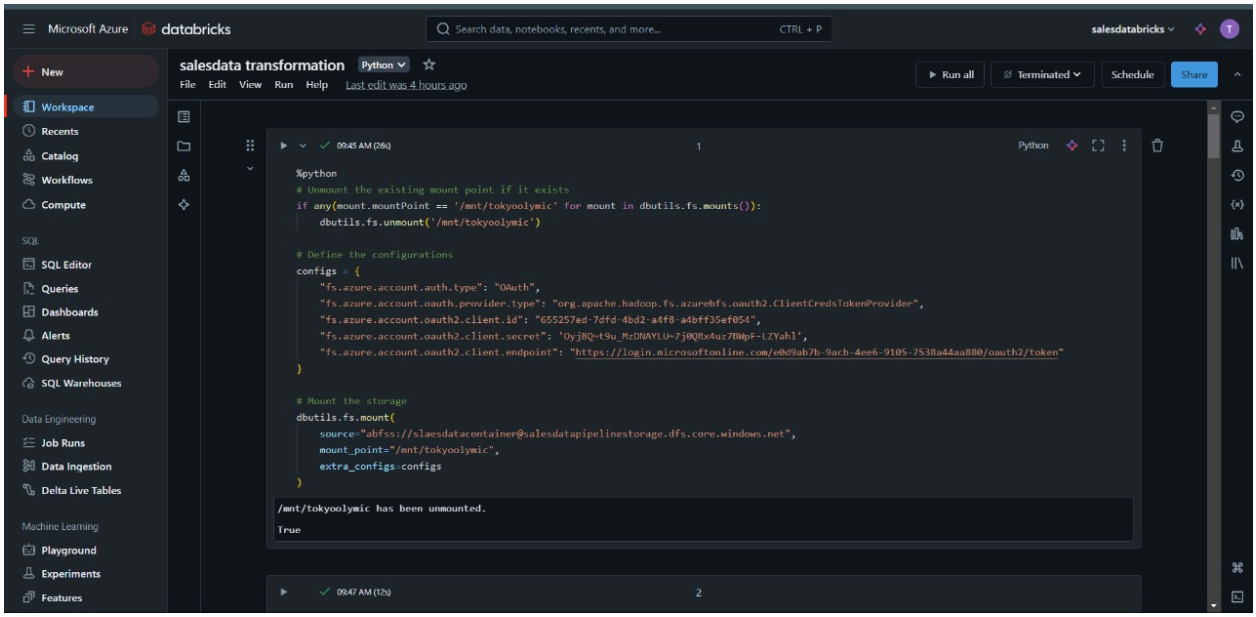
Appendix

Data Container

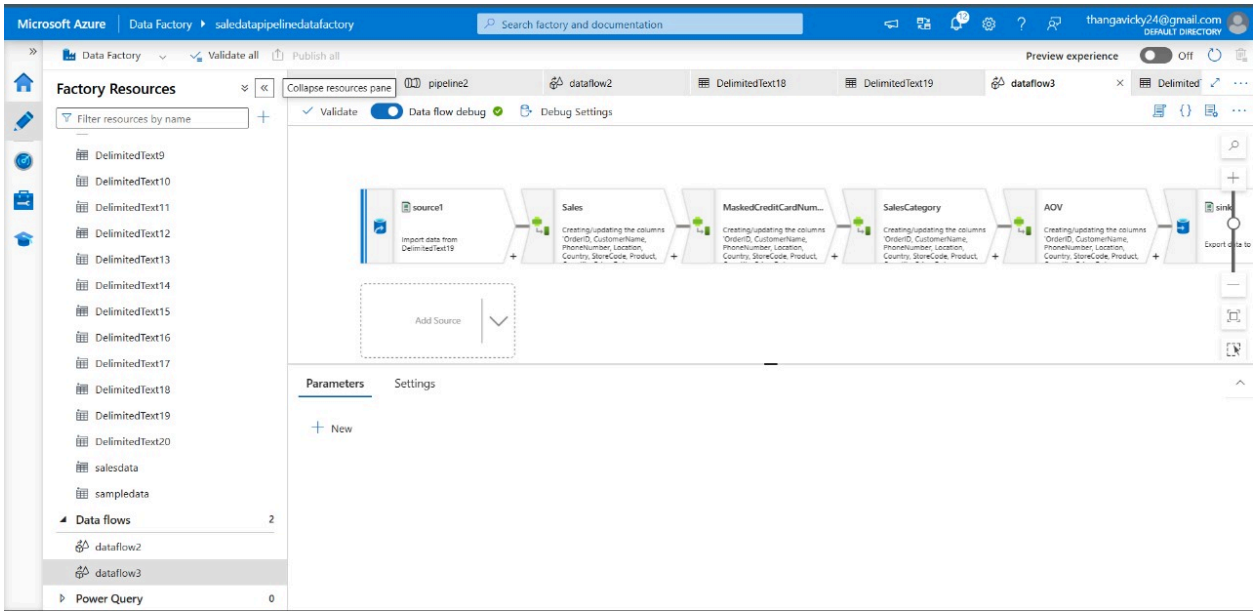


Storage Container

Data Bricks



Data Pipeline



Resources