Futurense Technologies |
13/06/2023

E-COMMERCE: CUSTOMER ANALYSIS

AZURE--PYSPARK-SPARK SQL-- SYNAPSE AND MUCH MORE....

Agenda

1. Team Introduction & Project Overview
2. Data Description & Preprocessing
3. Azure Pipelines and Analysis
4. Visualization and Results
5. Final Thoughts

Team Introduction

Nahush Dhavalikar

Venkat Sir DE Expert

Shriya Gandreti

Shivam Kumar

Jyoti Biswas

Sejal Singh

Deepak Mehta

Project Goals

- Analyze an e-commerce dataset to understand customer behavior
- Present the analysis results in a clear and concise manner, facilitating easy comprehension and decision-making.
- Enable the client to make informed decisions and take necessary actions to drive positive business outcomes.

Project Objectives

- Gain a comprehensive understanding of the provided e-commerce dataset.
- Identify any data quality issues, missing values, or anomalies that may impact the analysis.
- Utilize Azure platform to perform in-depth analysis of the preprocessed dataset.
- Leverage SQL queries and aggregations to derive valuable insights into client problems and potential solutions.
- Foster collaboration among team members by leveraging Azure Databricks and ADF to facilitate seamless data processing and sharing.
- Document the analysis process, methodologies, and key findings for future reference and knowledge sharing within the organization.

Data Description

The e-commerce dataset consists of transactional data related to customer purchases.

Columns:

- **InvoiceNo***:* A unique identifier for each invoice or transaction. It can include alphanumeric characters.
- **StockCode** : Unique identifier for the stock of the product.
- **Description**: A text description of the item or product purchased in the transaction.
- **Quantity**: The quantity or number of units purchased for a particular item in the transaction.
- **InvoiceDate**: The date and time when the invoice was generated or the transaction occurred. It includes information about both the date and time.
- **UnitPrice**: The price of a single unit of the product or item purchased. It is represented in the currency of the dataset.
- **CustomerID**: A unique identifier for each customer. It may contain alphanumeric characters or numeric IDs.
- **Country**: The country or region associated with the customer or transaction.

Data Size: 43.7 MB

Number Of Rows: 541909

Number of Columns: 8

Data Pre-processing

- Converted Date column from string to required data type, i.e. timestamp

```
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])
```

- Dropped the null values and duplicated rows

```
data=data.dropna(how='any')
data.drop_duplicates(inplace=True)
```

- Filtered the dataset to remove rows where Quantity is negative AND UnitPrice = 0.0

- Reduced memory usage by using data types that take less memory

```
data['Quantity'] = data['Quantity'].astype('int32')
data['UnitPrice'] = data['UnitPrice'].astype('float32')
data['CustomerID'] = data['CustomerID'].astype('int64')
```

Data

BEFORE

```
df.info()
```

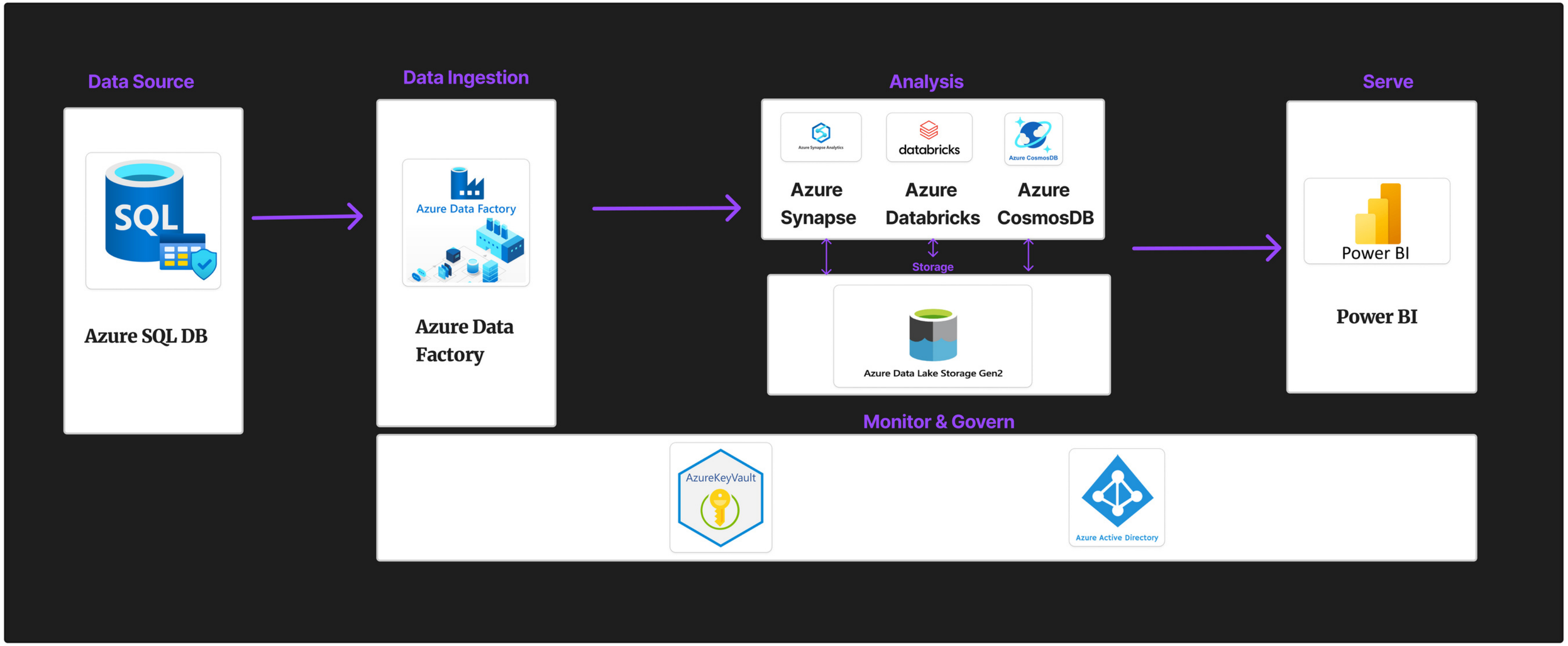
```
<class 'pandas.core.frame.DataFrame'>
Index: 397884 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    397884 non-null  object
 1   StockCode    397884 non-null  object
 2   Description  397884 non-null  object
 3   Quantity     397884 non-null  float64
 4   InvoiceDate  397884 non-null  datetime64[ns]
 5   UnitPrice    397884 non-null  float64
 6   CustomerID   397884 non-null  float64
 7   Country      397884 non-null  object
dtypes: datetime64[ns](1), float64(3), object(4)
memory usage: 44+ MB
```

AFTER

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 392692 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    392692 non-null  object
 1   StockCode    392692 non-null  object
 2   Description  392692 non-null  object
 3   Quantity     392692 non-null  int32
 4   InvoiceDate  392692 non-null  datetime64[ns]
 5   UnitPrice    392692 non-null  float32
 6   CustomerID   392692 non-null  int64
 7   Country      392692 non-null  object
dtypes: datetime64[ns](1), float32(1), int32(1), int64(1), object(4)
memory usage: 24.0+ MB
```

ARCHITECTURE



Azure Databricks

databricks

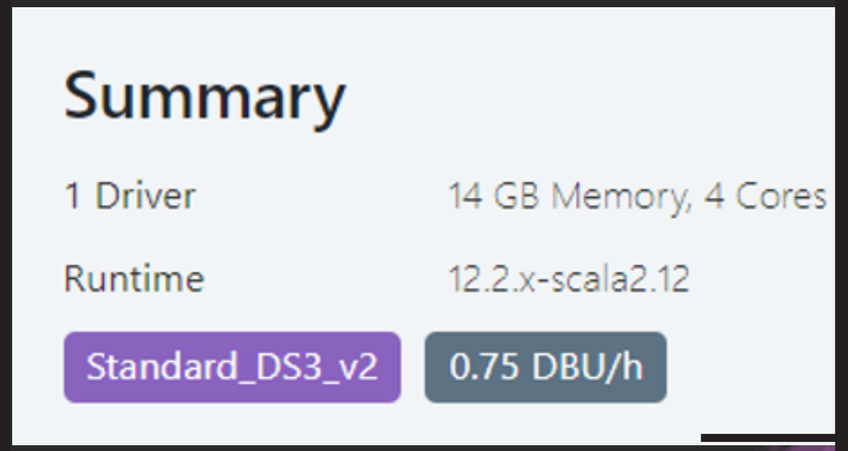
Analysis using Azure Databricks

Pipeline Used :

Azure SQL DB ----------> Azure Databricks ----------> ADLS Gen 2

SOURCE : Azure SQL DB
ANALYSIS PLATFORM : Azure Databricks
DESTINATION : ADLS Gen 2

Databricks Cluster:

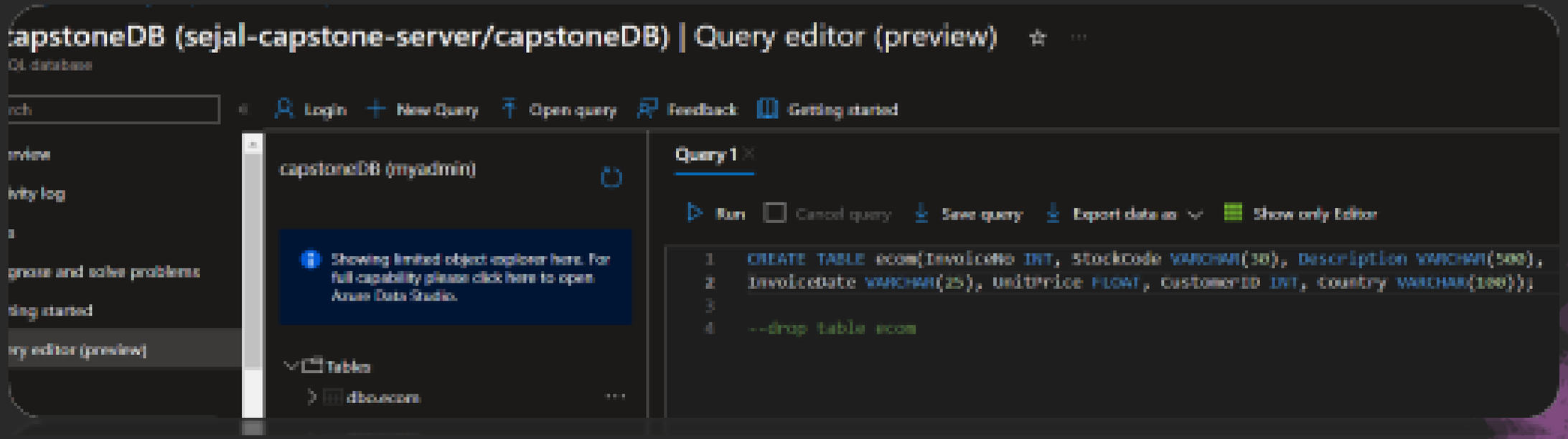


STEPS

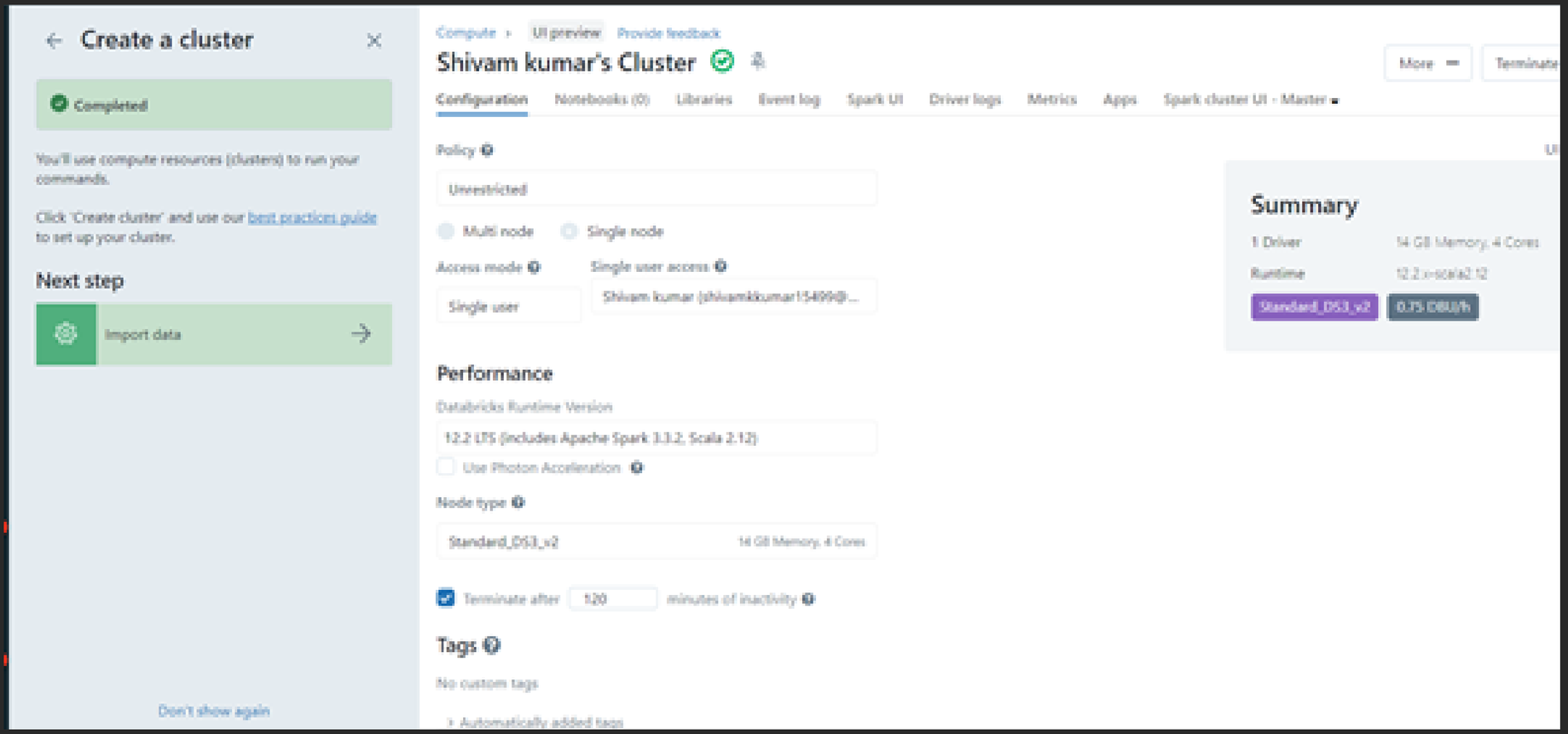
1. Created ADLS Storage account

2. Created Azure SQL DB Server

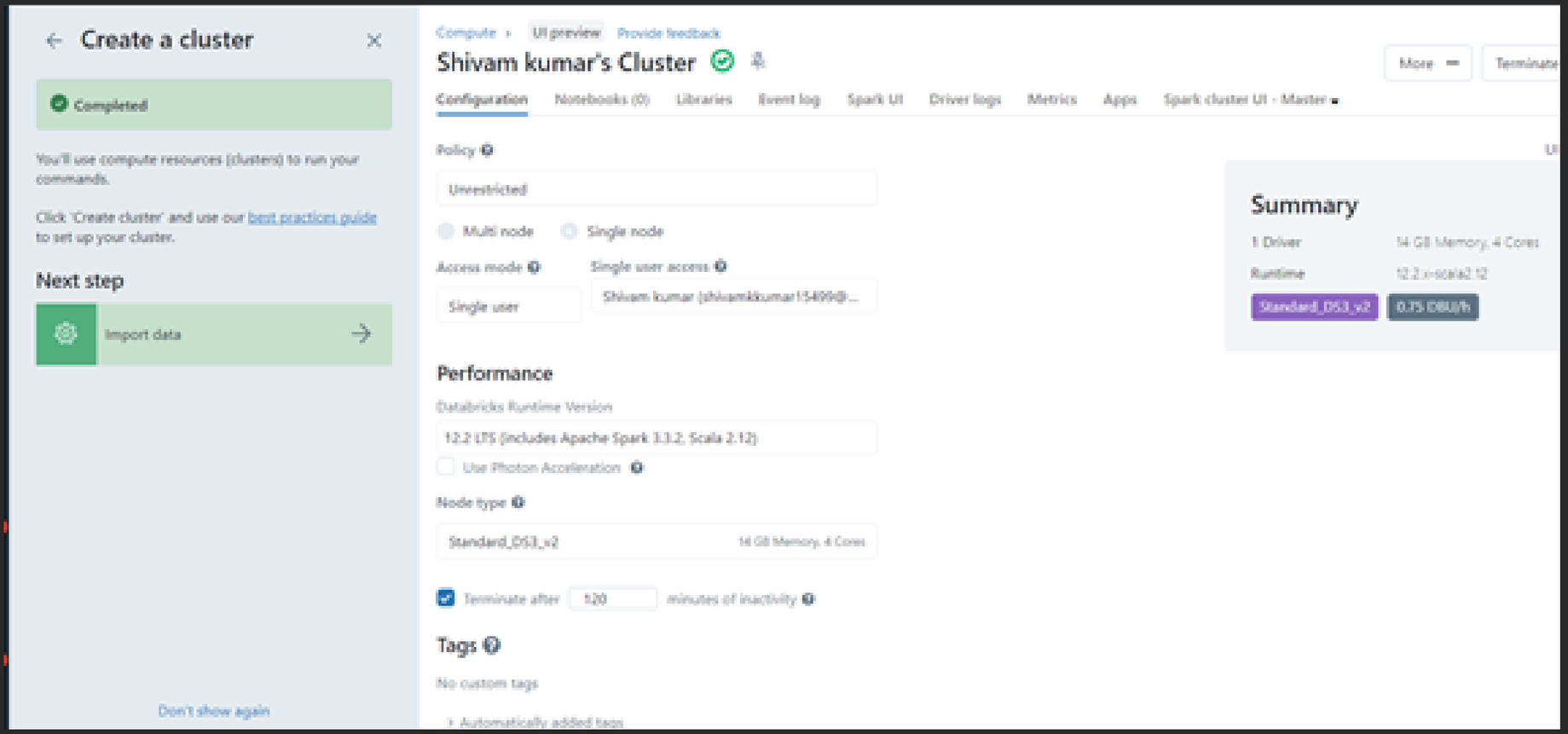
3. Created table in Azure SQL to store my dataset



4. Created Azure Databricks Cluster inside Azure Databricks workspace



4. Created Azure Databricks Cluster inside Azure Databricks workspace

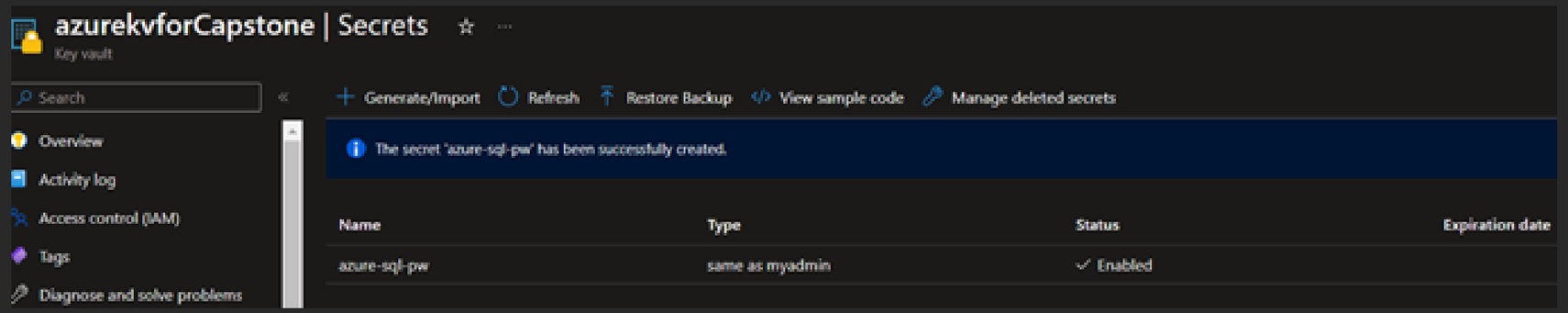


5. Used Azure Key Vault  to access Azure SQL 

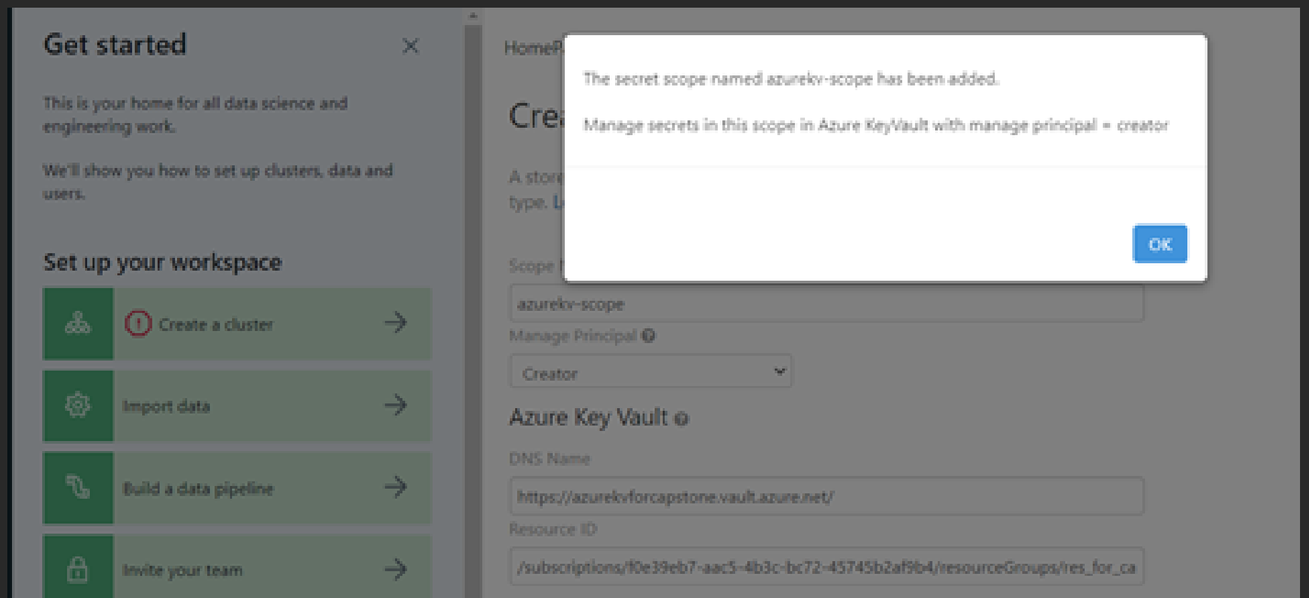
5. Used Azure Key Vault to access Azure SQL

   a. Created Azure Key Vault

   b. Created Secrets in Key Vault to contain Azure SQL Password, user name, host name etc.



   c. Created Databricks scoped credential in Databricks workspace



STEP 6:

Analysis in Databricks Notebook

a. Connecting Azure SQL with Databricks

```
jdbcHost = "sejal-capstone-server.database.windows.net"
jdbcPort = 1433
jbbcDB = "capstoneDB"
jdbcUser = "myadmin"
jdbcPW = dbutils.secrets.get("azurekv-scope", "azure-sql-pw")
jdbcDriver = "com.microsoft.sqlserver.jdbc.SQLServerDriver"

jdbcUrl = f"jdbc:sqlserver://{jdbcHost}:{jdbcPort};databaseName={jbbcDB};user={jdbcUser};password={jdbcPW}"
```

b. Loading data

```
ecom = spark.read.format("jdbc").option("url",jdbcUrl).option("dbtable", tableName).load()
```

STEP 6: (c)

Analysis in Databricks Notebook

PROBLEM STATEMENTS:

1. What is the total purchase bill for each customer?
2. Who are the customers from United Kingdom who have made purchases in July?
3. How many among customers are from France?
4. What products have Unit price of more than 10?

Analysis in Databricks Notebook

User story 1 solution:

```
# 1.Find the total price of the customer based on the customer id given. (price = quantity*Unit price )

job1DF = spark.sql('SELECT CustomerID, ROUND(sum(Quantity * UnitPrice),3) as Total_Price FROM ecom \
                    GROUP BY CustomerID ORDER BY  ROUND(sum(Quantity * UnitPrice),3) DESC ')
```

CustomerID	Total_Price
14646	280206.02
18102	259657.3
17450	194390.79
16446	168472.5
14911	143711.17
12415	124914.53
14156	117210.08
17511	91062.38
16029	80850.84
12346	77183.6
16684	66653.56
14096	65164.79
13694	65039.62
15311	60632.75
13089	58762.08
17949	58510.48
15769	56252.72
15061	54534.14
14298	51527.3
14088	50491.81

only showing top 20 rows

User story 2 solution:

```
# 2.Find the customer ID who are from country United Kingdom and has Invoice date at July month
job2DF = spark.sql('SELECT DISTINCT CustomerID, Country, SUBSTRING(InvoiceDate,1,4) AS Year,\
                    SUBSTRING(InvoiceDate,6,2) AS Month \
                    FROM ecom WHERE Country = "United Kingdom" AND SUBSTRING(InvoiceDate,6,2) = 7\
                    ORDER BY CustomerID')
```

CustomerID	Country	Year	Month
12748	United Kingdom	2011	7
12830	United Kingdom	2011	7
12833	United Kingdom	2011	7
12836	United Kingdom	2011	7
12839	United Kingdom	2011	7
12840	United Kingdom	2011	7
12841	United Kingdom	2011	7
12843	United Kingdom	2011	7
12853	United Kingdom	2011	7
12864	United Kingdom	2011	7
12875	United Kingdom	2011	7
12901	United Kingdom	2011	7
12910	United Kingdom	2011	7
12915	United Kingdom	2011	7
12916	United Kingdom	2011	7
12919	United Kingdom	2011	7
12921	United Kingdom	2011	7
12922	United Kingdom	2011	7
12928	United Kingdom	2011	7
12935	United Kingdom	2011	7

only showing top 20 rows

Analysis in Databricks Notebook

User story 3 solution:

```
# 3.Find the total number of people from France.
job3DF = spark.sql("SELECT COUNT(CustomerID) AS Total_People_from_France FROM ecom \
                    WHERE Country = 'France' ")
```

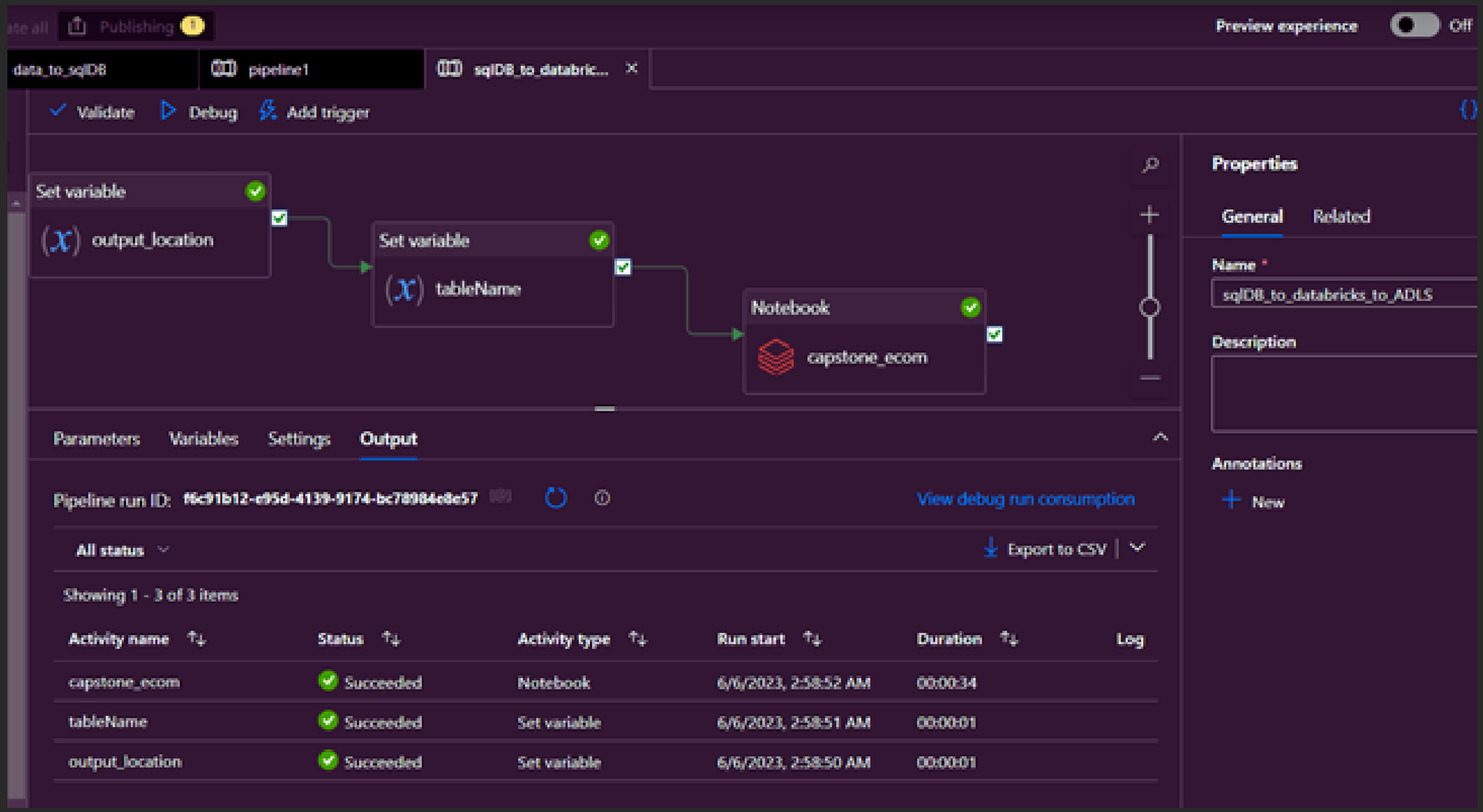
```
+------------------------+
|Total_People_from_France|
+------------------------+
|                      87|
+------------------------+
```

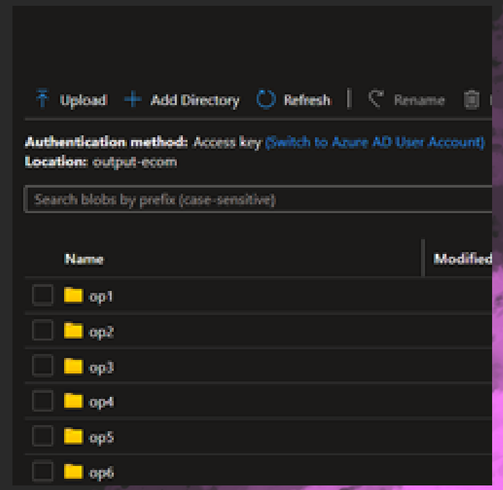
```
+---------+---------+
|StockCode|UnitPrice|
+---------+---------+
|    22120|    10.13|
|    23314|    10.39|
|    23143|     10.4|
|    23104|     10.4|
|    23085|     10.4|
|    23142|     10.4|
|    23427|     10.4|
|    23111|     10.4|
|   47570B|    10.65|
|    85141|    10.75|
|   90035A|    10.75|
|   90035C|    10.75|
|    22832|    10.75|
|    22165|    10.75|
|    21216|    10.79|
|    85174|    10.79|
|    23245|    10.79|
|    47566|    10.79|
|    22476|    10.79|
|    22139|    10.79|
+---------+---------+
only showing top 20 rows
```

User story 4 solution:

```
# 4. Find out the stock code which has unit price more than 10
job4DF = spark.sql("SELECT DISTINCT StockCode, UnitPrice FROM ecom \
                    WHERE UnitPrice > 10 ORDER BY UnitPrice")
```

STEP 7: ADF Pipeline





Standalone Analysis

PySpark RDD

Importing the CSV into RDD

```
# 1. Creating RDD for ecommerce dataset

ecomRDD = sc.textFile('data_processed.csv')
type(ecomRDD)
ecomRDD.count()
```

```
# 2. Removing headers

header = ecomRDD.first()
ecomRDD = ecomRDD.filter(lambda row: row!=header)
ecomRDD.take(2)
```

```
# 3. Splitting records using comma -> " , " as delimiter

ecomRDD_final = ecomRDD.map(lambda x : x.split(','))
ecomRDD_final.take(1)
```

```
# 5. Converting Data types for required columns using function

def change_dataType(x):
    return [x[0], x[1], x[2], int(x[3]), x[4], float(x[5]), x[6], x[7]]


ecomRDD_final = ecomRDD_final.map(change_dataType)
ecomRDD_final.take(2)
```

Final RDD

```
ecom_final.take(2)
```

```
[['536365',
  '85123A',
  'WHITE HANGING HEART T-LIGHT HOLDER',
  6,
  '01-12-2010 08:26',
  2.55,
  '17850',
  'United Kingdom'],
 ['536365',
  '71053',
  'WHITE METAL LANTERN',
  6,
  '01-12-2010 08:26',
  3.39,
  '17850',
  'United Kingdom']]
```

User Story : How many customers from France?

```
def is_from_France(row):
    return row[7] == 'France'
```

```
ecom_fromFrance = ecomRDD_final.filter(is_from_France)
```

```
custCount_fromFrance = ecom_fromFrance.map(lambda r: r[6]).distinct().count()
```

```
Total No. of Customers from France : ------>  87
```

User Story : Find out the stock code which has unit price more than 10 ?

```
# Defining a function to get rows only where Unit price > 10

def unitPrice_moreThan10(row):
    return row[5] > 10
```

```
ecom_price10 = ecomRDD_final.filter(unitPrice_moreThan10)
ecom_price10.take(2)
```

```
Stock Codes where Unit Price is greater than 10:
```

```
[('22120', 10.13),
 ('23314', 10.39),
 ('23427', 10.4),
 ('23142', 10.4),
 ('23143', 10.4),
 ('23111', 10.4),
 ('23085', 10.4),
 ('23104', 10.4),
 ('47570B', 10.65),
 ('22832', 10.75),
 ('90035A', 10.75),
 ('22165', 10.75),
 ('85141', 10.75),
```

Total No. of products with unit price more than 10

```
StockCode_price10.count()
```

```
504
```

Do You Understand ?

Venkat Sir DE Expert

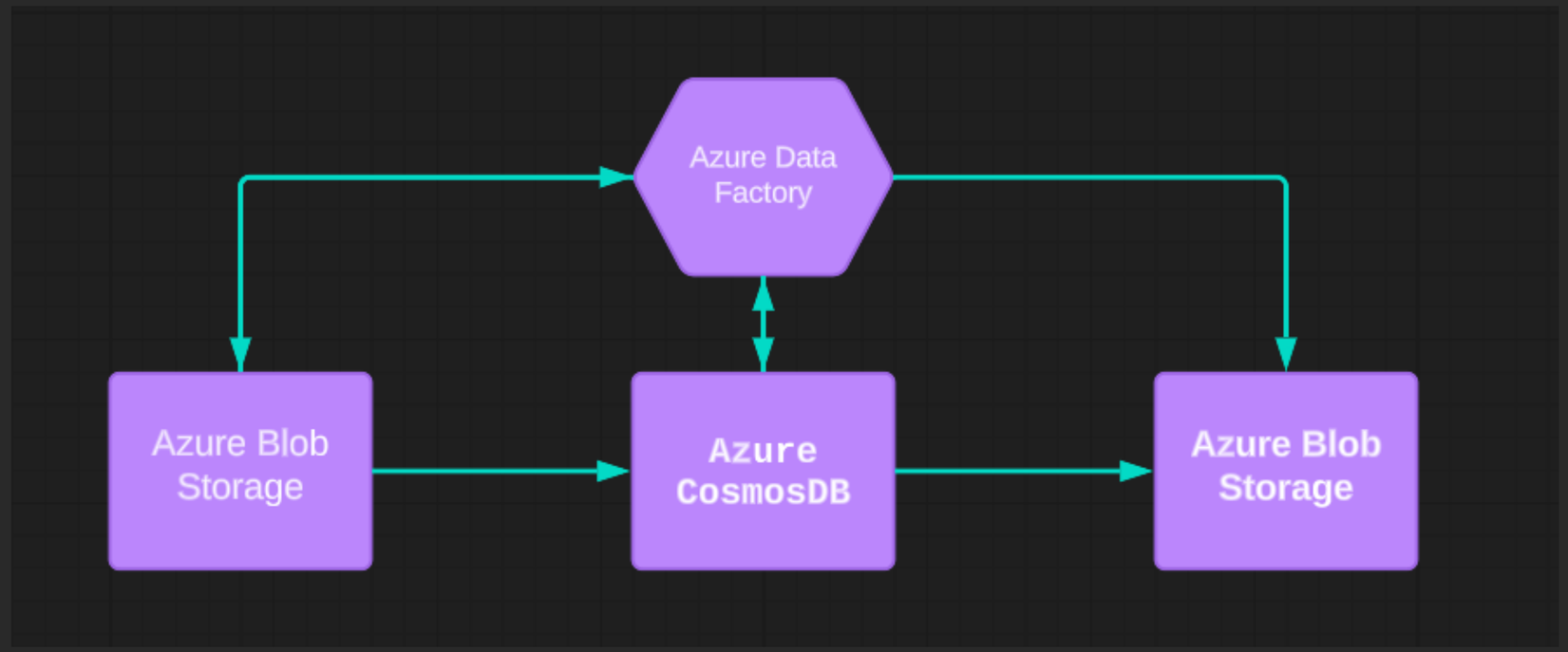
Azure

Pipeline 2 - CosmosDB

Why CosmosDB?

1. Global Distribution:
   a. Cosmos DB provides global distribution capabilities, allowing you to replicate your data across multiple regions worldwide. This enables low-latency access to data for users located in different regions, ensuring a better user experience.
2. Scalability:
   a. Cosmos DB is designed to scale seamlessly and handle massive amounts of data. It can handle both read and write-heavy workloads, and it automatically scales throughput and storage based on demand. This elasticity makes it suitable for applications with unpredictable or rapidly changing workloads.
3. Multi-Model Database:
   a. Cosmos DB supports multiple data models, including key-value, column-family, document, graph, and time-series. This flexibility allows you to choose the most appropriate data model for your application without the need for separate databases, reducing complexity and cost.

ADF Pipeline Architecture -



Analysis in CosmosDB

PROBLEM STATEMENTS:

1. Filter out the column where quantity between 4 and 9?
> SELECT * FROM ecom WHERE Quantity BETWEEN 4 AND 9 ORDER BY InvoiceNo;

2. Filter out the column where customer ID 15862?
> SELECT DISTINCT InvoiceNo, InvoiceDate, StockCode, Description, Quantity, UnitPrice FROM ecom WHERE CustomerID = 15862;

3. Filter out the unit Price which has more than 5?
> SELECT * FROM ecom WHERE UnitPrice > 5 ORDER BY InvoiceNo;

Azure

Pipeline 3 - Synapse

1. **Data Integration:** Azure Synapse brings together data from different sources for seamless analysis.
2. **Data Warehousing:** It provides a high-performance, scalable data warehousing solution for structured data.
3. **Big Data Analytics:** Synapse supports Apache Spark and Hadoop for analyzing large volumes of unstructured data.
4. **Advanced Analytics:** Integrated with Azure Machine Learning, it enables the development and deployment of machine learning models.
5. **Real-Time Data Processing:** Synapse supports real-time data ingestion, processing, and monitoring.
6. **Security and Governance:** It offers robust security features and integrates with Azure Purview for data governance.
7. **Scalability and Flexibility:** Synapse leverages Azure's cloud infrastructure for unlimited scalability and cost efficiency.

Architecture of Synapse

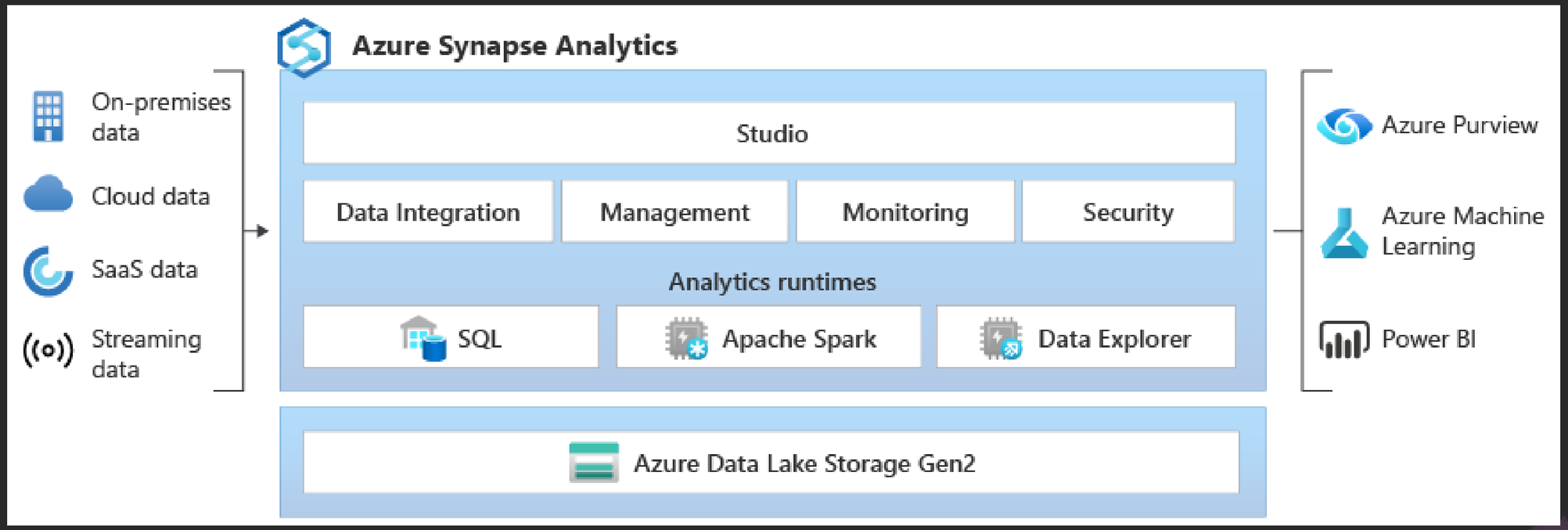
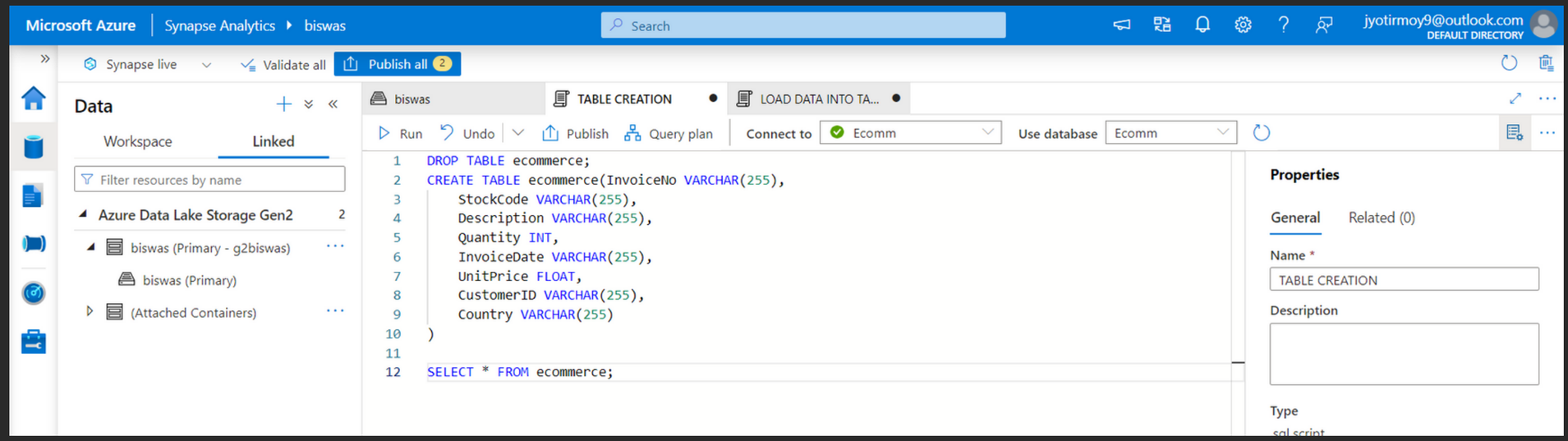
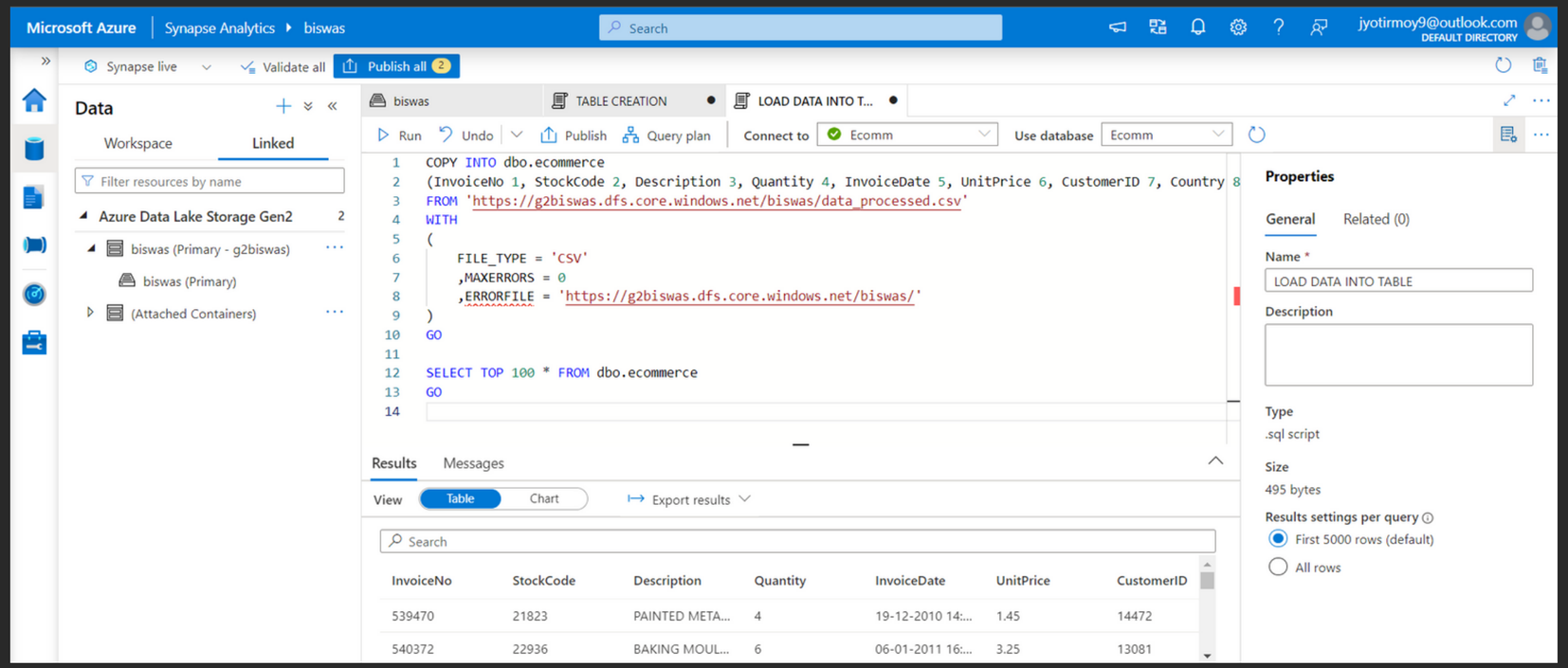
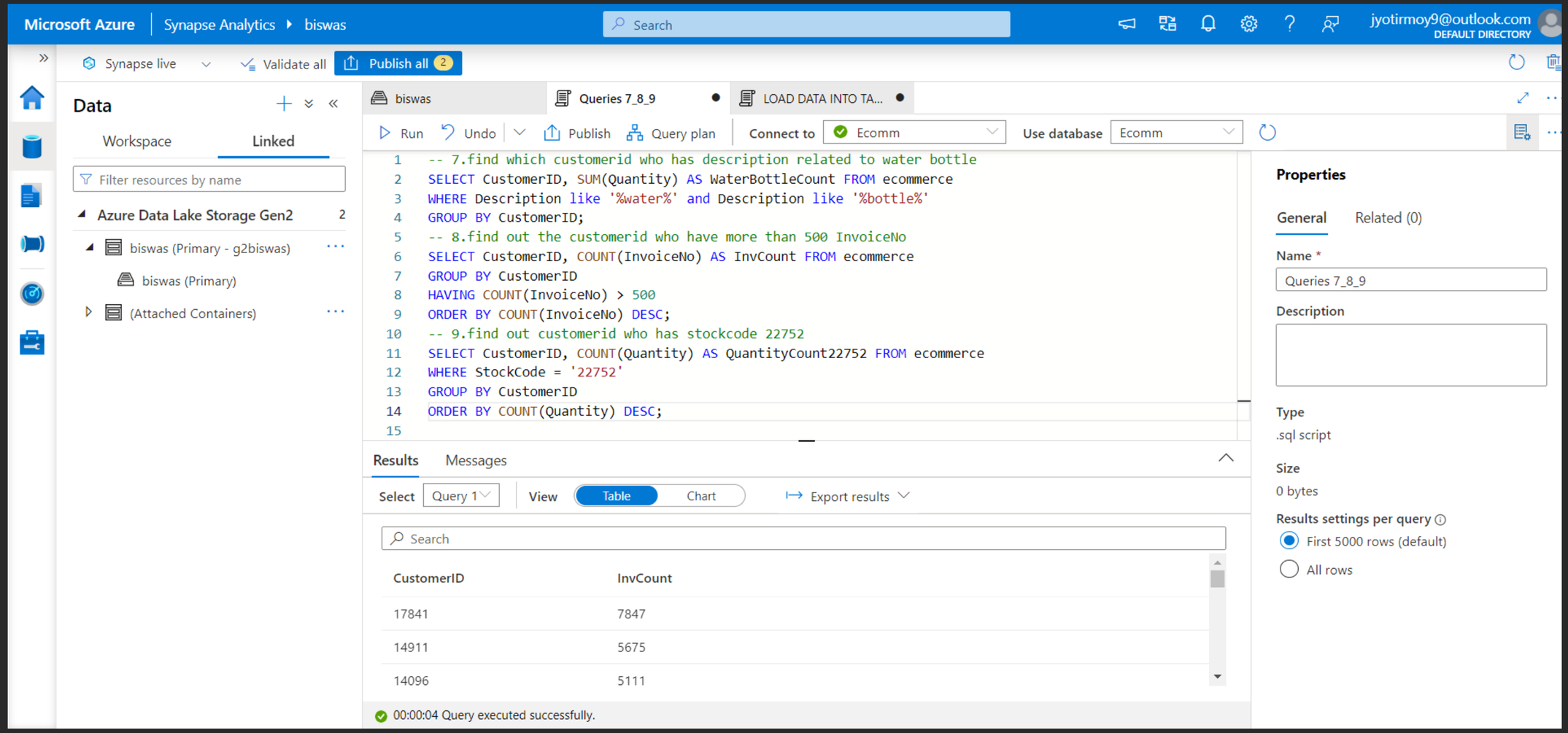


Table Creation and Data Insertion



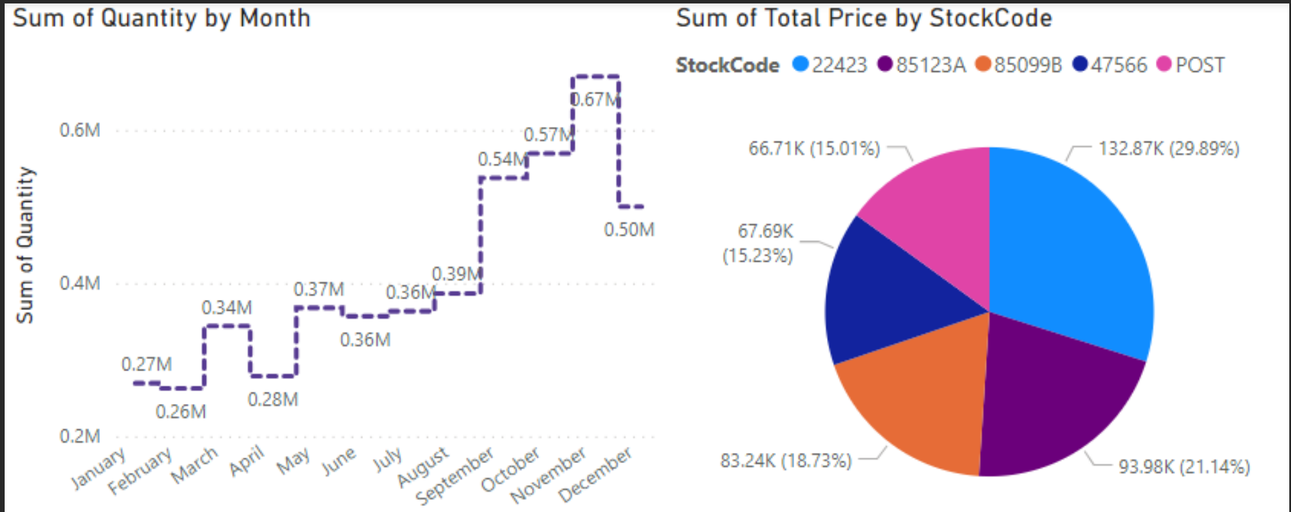


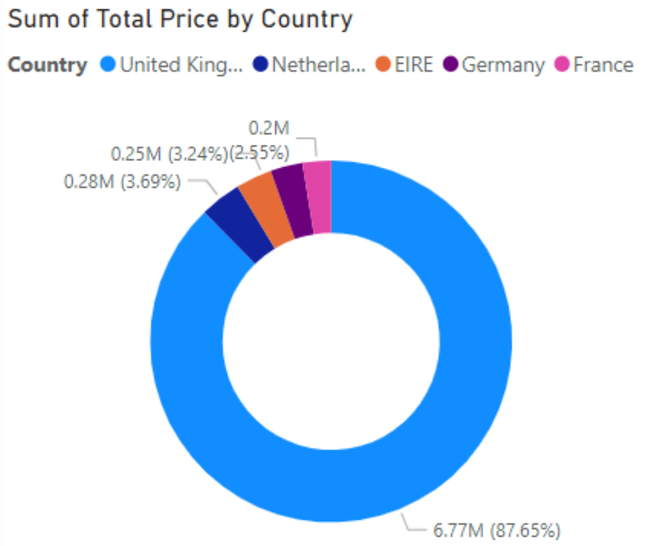
Synapse Queries

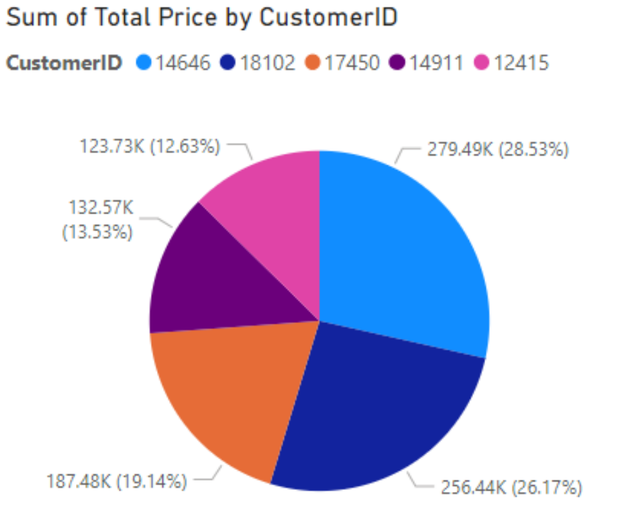


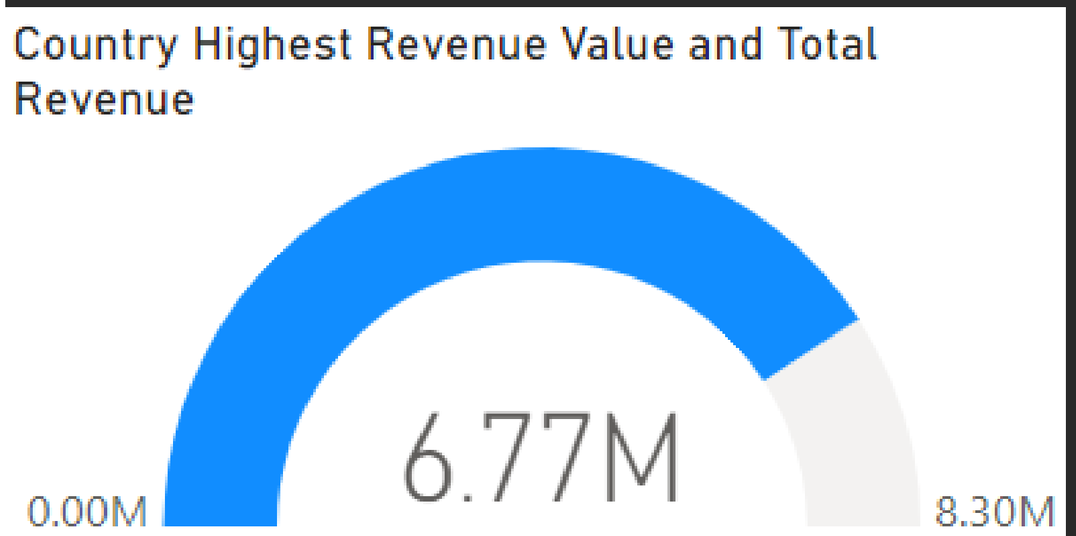
Visualization

Visualization









Visualization

Singapore
Country Highest Avg Unit Price

United Kingdom
Country Highest Revenue

22423
Stock Code Maximum Revenue

512
Average Number of Unique Stock Codes

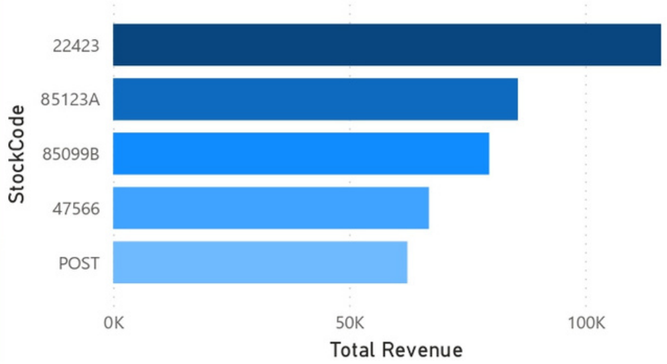
Stock Code with Highest Quantity Sold

84077

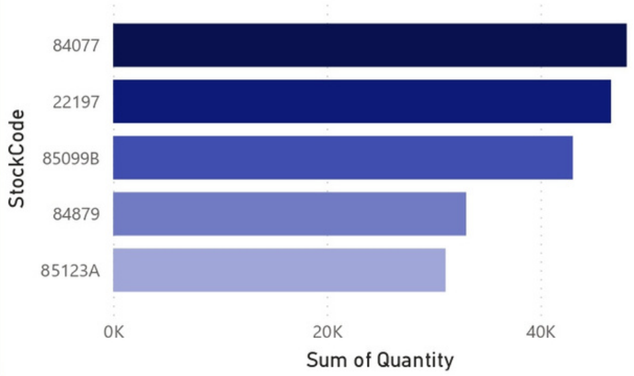
Year

Select all | 2010 | 2011

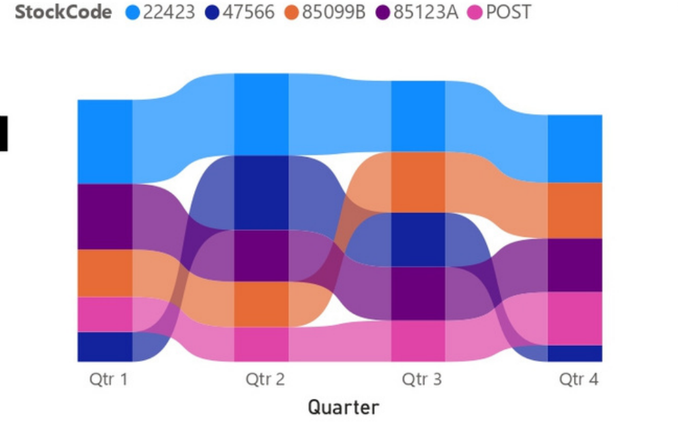
Total Revenue by StockCode



Sum of Quantity by StockCode

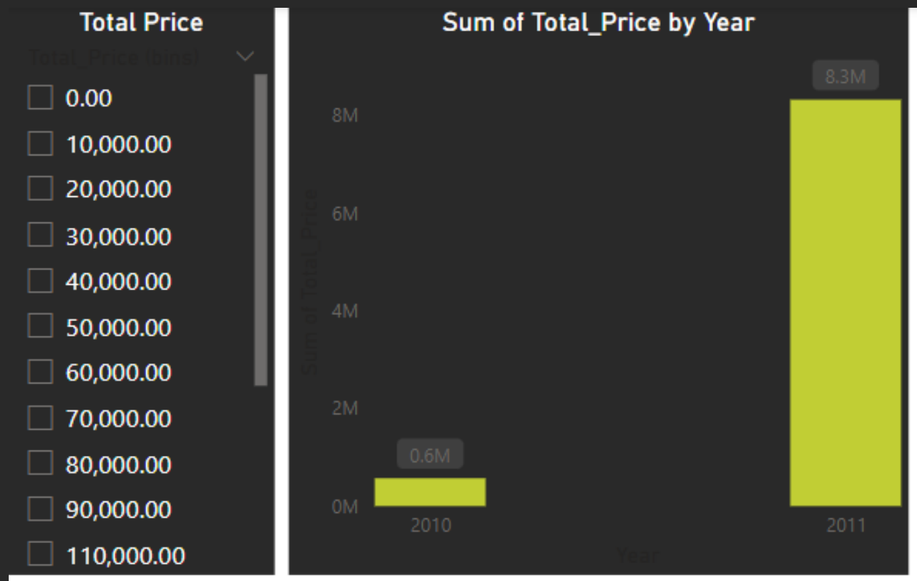


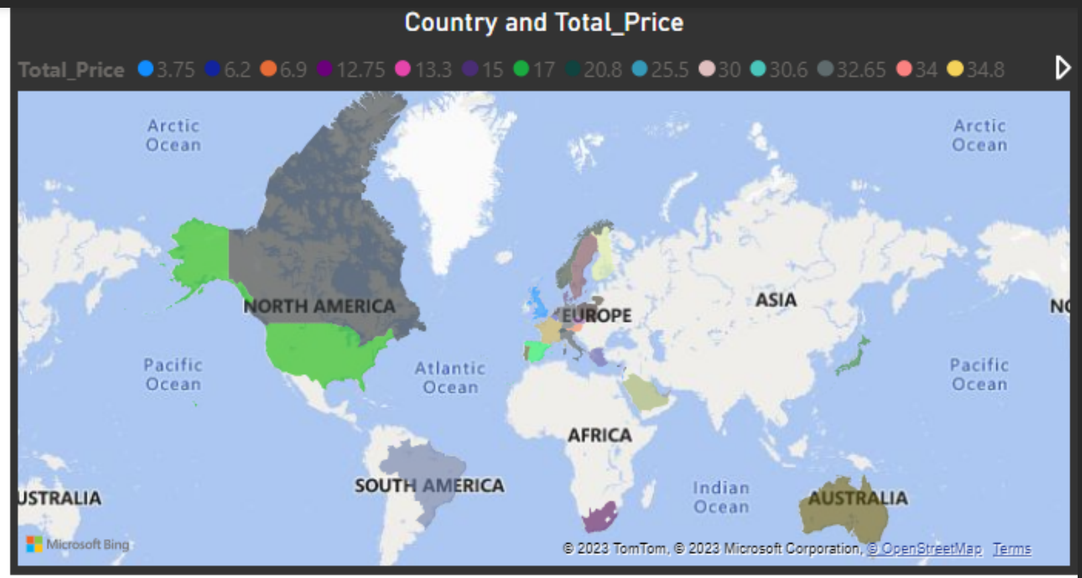
Total Revenue by Quarter and StockCode

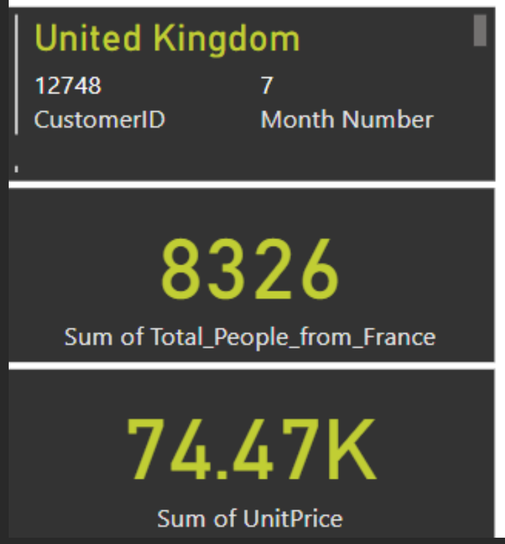


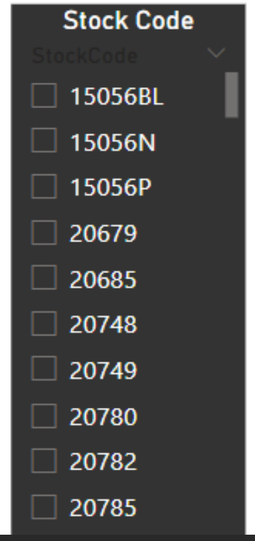
Average Invoice Value by Country

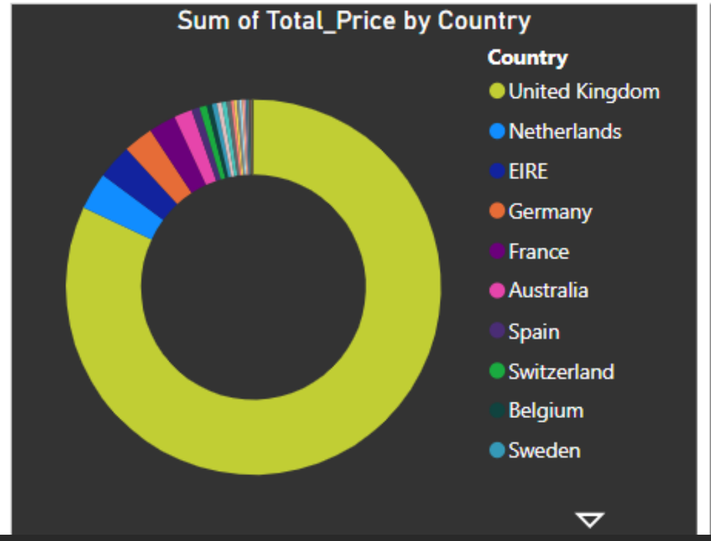
Visualization

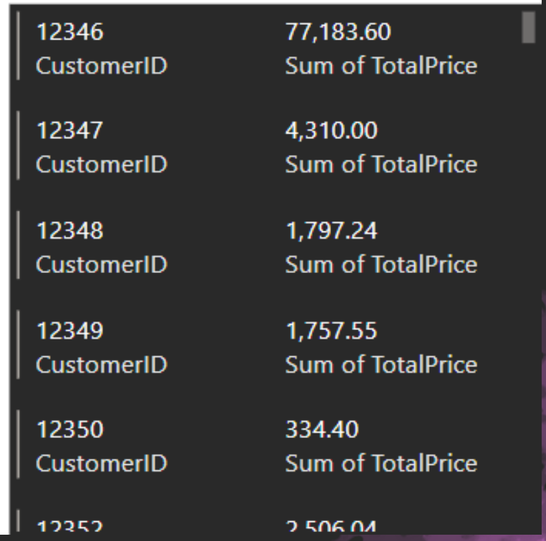






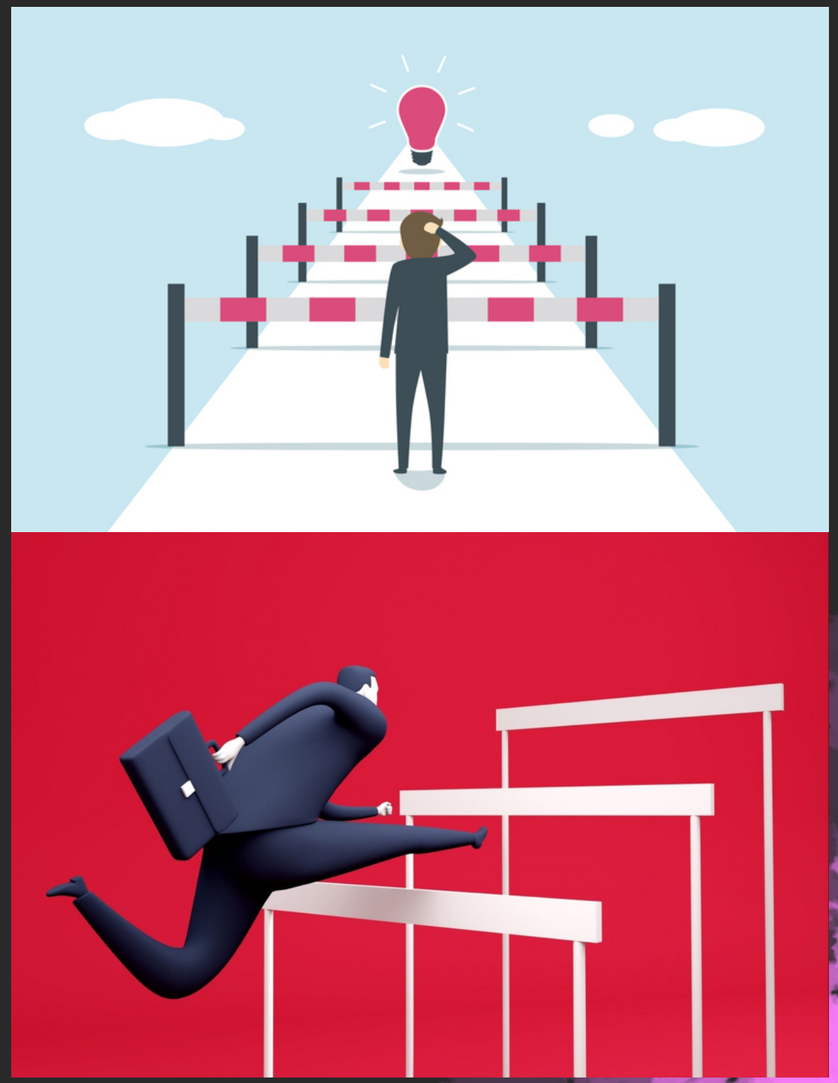






Challenges

- Data Pre-Processing
- Choosing the right tools for Processing and Analyzing
- Building efficient Pipeline
- Methods for Evaluating and Improving Performance



Final Thoughts

Thank you for participating!