Retail Inventory Management and Forecasting

october 2024

Table of Content

01 Introduction

02 Project Objectives

03 Database and Data Warehouse

04 Technology Stack

05 Data Ingestion

06 Data Transformation & Analysis

07 Building the Forecasting Model

08 Automation and Monitoring

Introduction

In our daily lives, we constantly face choices that depend on the availability of products and services. Whether shopping for groceries, electronics, or clothing, having access to what we need at the right time is essential for our convenience and satisfaction. Effective inventory management is crucial in meeting consumer demands and ensuring a seamless shopping experience.

When businesses manage their inventory well, they fulfill consumer needs and enhance quality of life. However, poor inventory management can lead to stockouts and frustrated customers. As expectations for prompt service and product availability rise, accurate demand forecasting becomes vital. By analyzing historical sales data and market trends, retailers can anticipate needs and optimize stock levels, allowing them to adapt to changing consumer preferences and unforeseen circumstances.

Project Objectives

The primary objective is to create an data pipeline on Azure that facilitates the extraction, transformation, and analysis of customer and sales data from an on-premises SQL Server database. By implementing machine learning models, we aim to accurately forecast future inventory demand, enabling stakeholders to make data-driven decisions that enhance stock management practices.

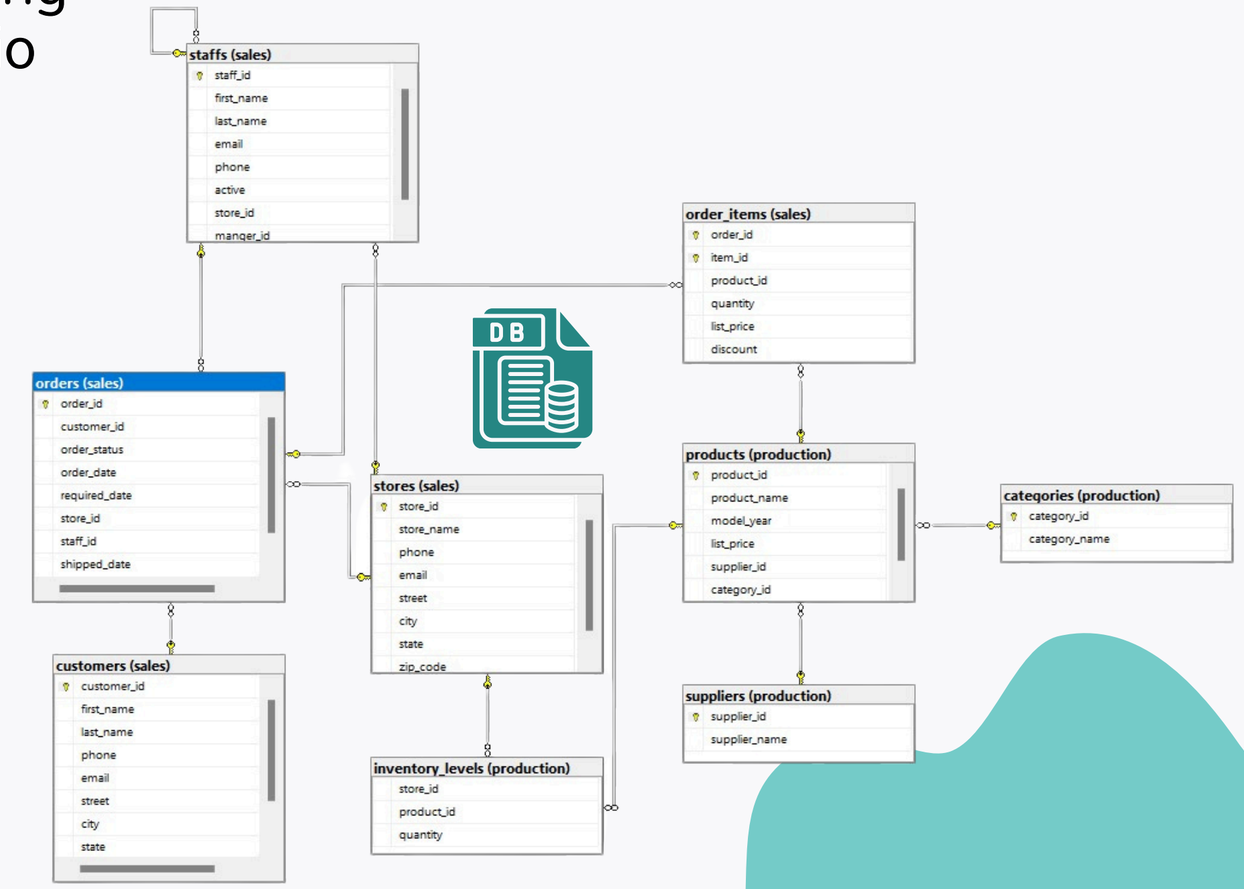
As we progress through this presentation, we will explore the various components of the project, including data ingestion, transformation processes, model building, and result visualization. This approach not only provides valuable insights into inventory dynamics but also empowers retailers to anticipate demand changes and maintain optimal stock levels.

Database and Data Warehouse

- SQL Database Design and Implementation using Microsoft SQL Server, SQL Management Studio

SQL database schema for managing retail inventory :

two schemas including tables for products, categories, suppliers, sales, and inventory levels, orders, order_items, sotres



Database and Data Warehouse

- SQL queries for inventory tracking and sales reporting

inventory_item

This stored procedure manages inventory for a specific product at a designated store.

```
-- take product_id, quantity_sold, store_id
-- update the inventory level and return the rest of the quantity
inventory_item 1,0,1
```

	store...	product...	quant...
1	1	1	26

```
--- inventory tracking

CREATE PROC inventory_item @product INT, @quantity_sold SMALLINT, @store SMALLINT
AS
BEGIN
    DECLARE @product_level INT;
    BEGIN TRY
        BEGIN TRANSACTION
            SELECT @product_level = quantity
            FROM production.inventory_levels
            WHERE product_id = @product AND store_id = @store;

            IF @product_level >= @quantity_sold
                BEGIN
                    UPDATE production.inventory_levels
                    SET quantity = quantity - @quantity_sold
                   WHERE product_id = @product AND store_id = @store;
                COMMIT TRANSACTION;
                END
            ELSE
                BEGIN
                ROLLBACK TRANSACTION;
                RAISERROR('Not enough stock to fulfill the sale', 16, 1);
                END
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
            DECLARE @ErrorMessage NVARCHAR(4000);
        SELECT @ErrorMessage = ERROR_MESSAGE();
        RAISERROR(@ErrorMessage, 16, 1);
    END CATCH;
    SELECT *
    FROM production.inventory_levels
    WHERE product_id = @product AND store_id = @store;

END;
```

Database and Data Warehouse

dailyReport

This stored procedure generates a daily report based on sales for a specified date.

```
-- it show the product quantity sold and the sales for each one
dailyReport '1/4/2016'
```

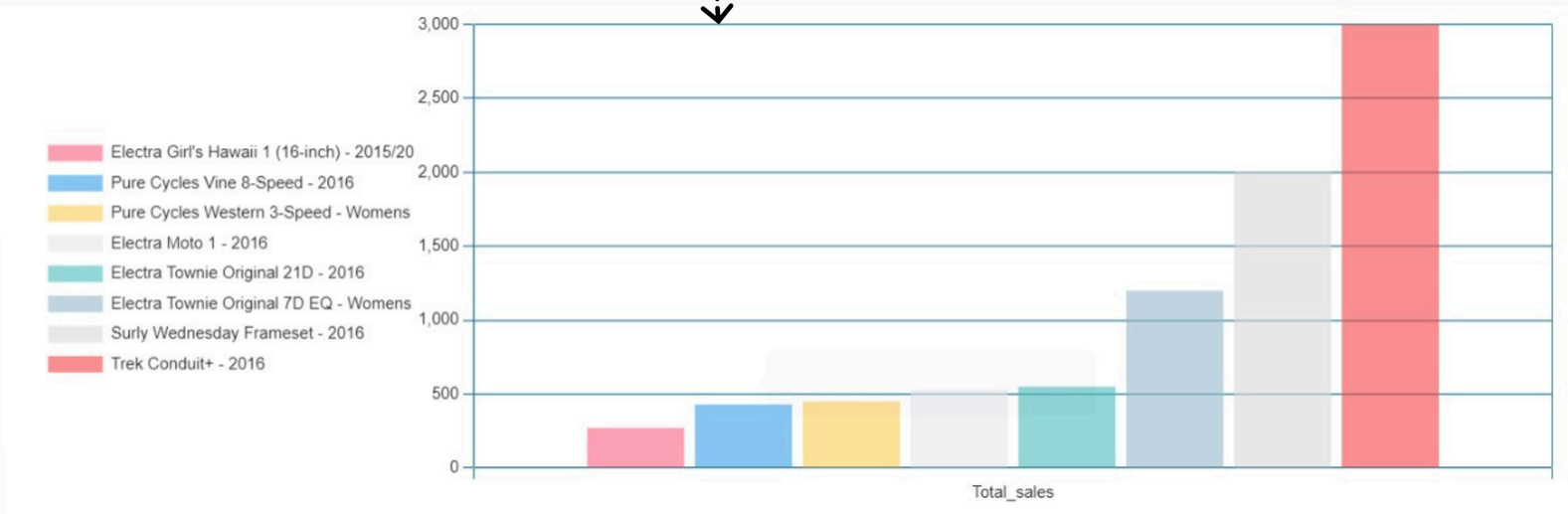
	product_name	Total_s…
1	Electra Girl's Hawaii 1 (16-inch)…	269.99
2	Pure Cycles Vine 8-Speed - 2016	429.00
3	Pure Cycles Western 3-Speed - Wom…	449.00
4	Electra Moto 1 - 2016	529.99
5	Electra Townie Original 21D - 201…	549.99
6	Electra Townie Original 7D EQ - W…	1199.98
7	Surly Wednesday Frameset - 2016	1999.98
8	Trek Conduit+ - 2016	2999.99

	product_name	Total_quan…
1	Electra Girl's Hawaii 1 (16-inch)…	1
2	Electra Moto 1 - 2016	1
3	Pure Cycles Western 3-Speed - Wom…	1
4	Pure Cycles Vine 8-Speed - 2016	2
5	Trek Conduit+ - 2016	2
6	Electra Townie Original 21D - 201…	2
7	Electra Townie Original 7D EQ - W…	3
8	Surly Wednesday Frameset - 2016	3

```
--- daily report

CREATE PROC dailyReport (@order_date date)
AS
BEGIN
    SELECT product_name, SUM(quantity) AS Total_quantity
    FROM sales.orders AS O
    JOIN sales.order_items AS OI
    ON o.order_id = oi.order_id
    JOIN production.products AS P
    ON OI.product_id = P.product_id
    WHERE order_date = @order_date
    GROUP BY product_name
    ORDER BY Total_quantity;


    SELECT product_name, SUM(OI.list_price) AS Total_sales
    FROM sales.orders AS O
    JOIN sales.order_items AS OI
    ON o.order_id = oi.order_id
    JOIN production.products AS P
    ON OI.product_id = P.product_id
    WHERE order_date = @order_date
    GROUP BY product_name
    ORDER BY Total_sales;

END;
```

Database and Data Warehouse

- daily report using azure data studio



Database and Data Warehouse

inventory_check

This function returns a table containing all products with inventory levels below a specified threshold (less than 5 units).

```
---- inventory check
CREATE FUNCTION inventory_check()
RETURNS TABLE
AS
RETURN(
    SELECT *
    FROM production.inventory_levels
    WHERE quantity < 5
);
```

	store...	product...	quant...
1	1	6	0
2	1	8	0
3	1	15	3
4	1	16	4
5	1	17	2
6	1	19	4
7	1	31	2
8	1	32	0
9	1	34	2
10	1	39	2
11	1	42	0
12	1	43	2
13	1	44	1
14	1	51	2
15	1	57	1
16	1	58	1
17	1	63	4
18	1	69	4
19	1	70	3
20	1	82	1

```
-- check for values less than or equal 5 in inventory
select * from inventory_check()
```

Visualization of Results

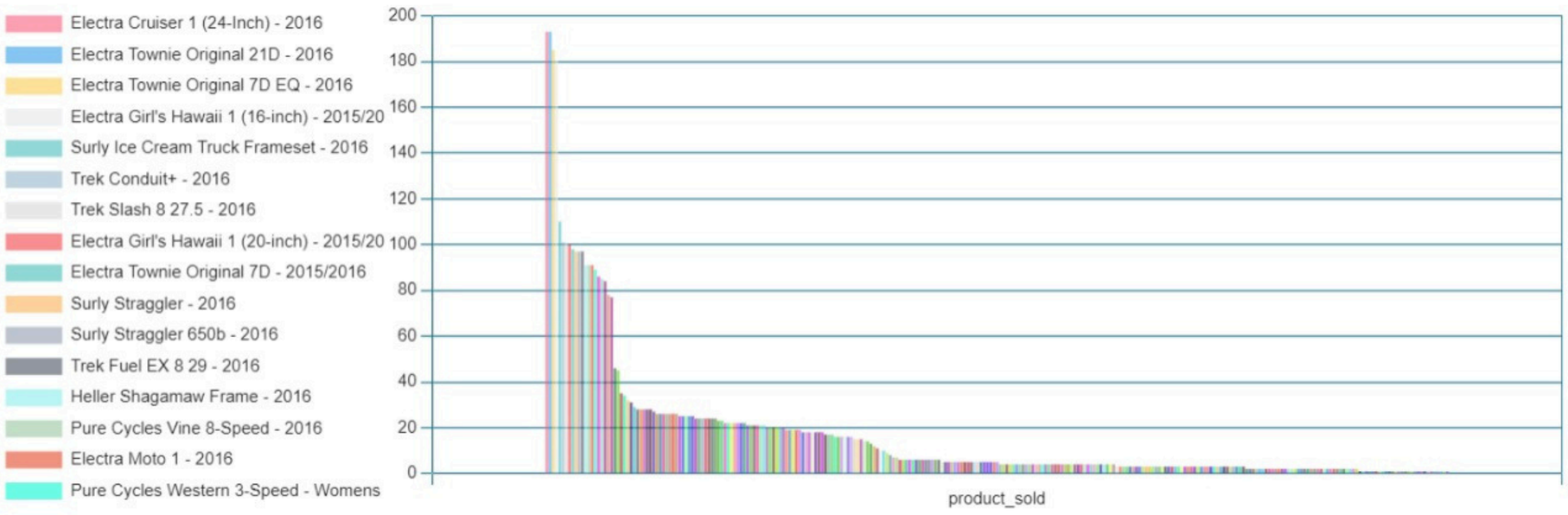
- Sample SQL queries for inventory and sales analysis using azure data studio

most product sold

```sql
SELECT product_name,count(*) AS product_sold
FROM sales.order_items AS O
JOIN production.products AS P
ON O.product_id = P.product_id
GROUP BY product_name
ORDER BY product_sold DESC;
```

rows affected)

execution time: 00:00:00.021



Visualization of Results

most customer city have order

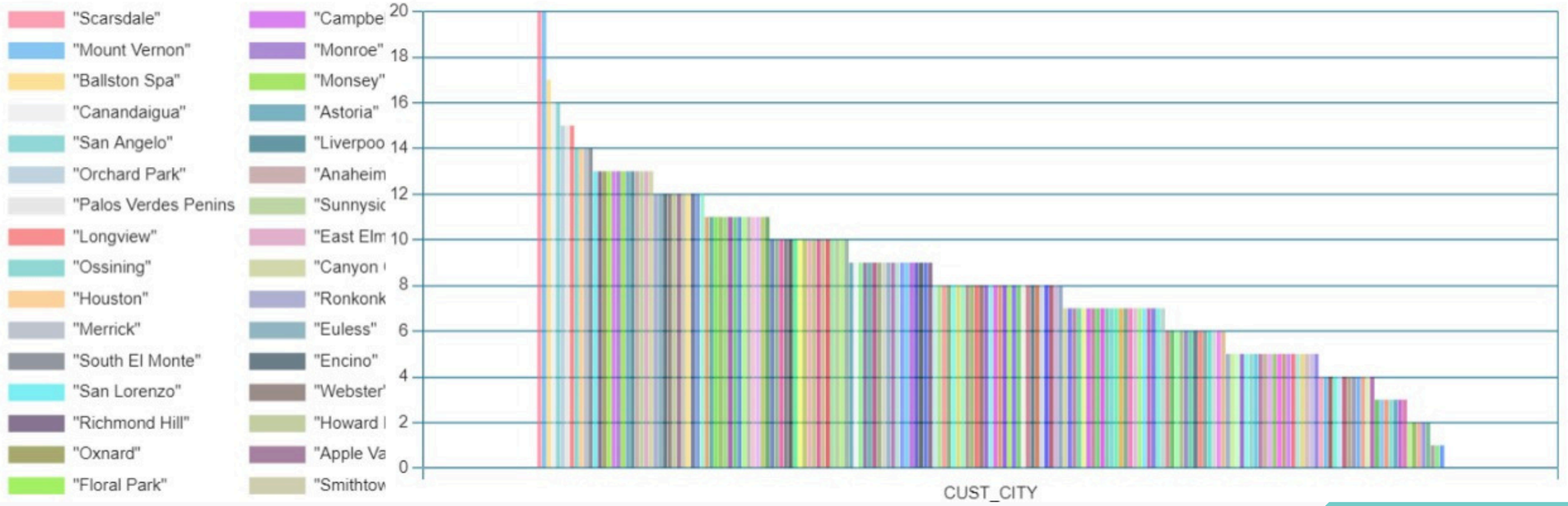
```sql
SELECT city, COUNT(*) AS CUST_CITY
FROM sales.customers AS C
JOIN sales.orders AS O
ON C.customer_id = O.customer_id
GROUP BY city
ORDER BY CUST_CITY DESC;
```

SQL

5 rows affected)

al execution time: 00:00:00.027

Copy as image Save as image Configure Chart



Visualization of Results

order type

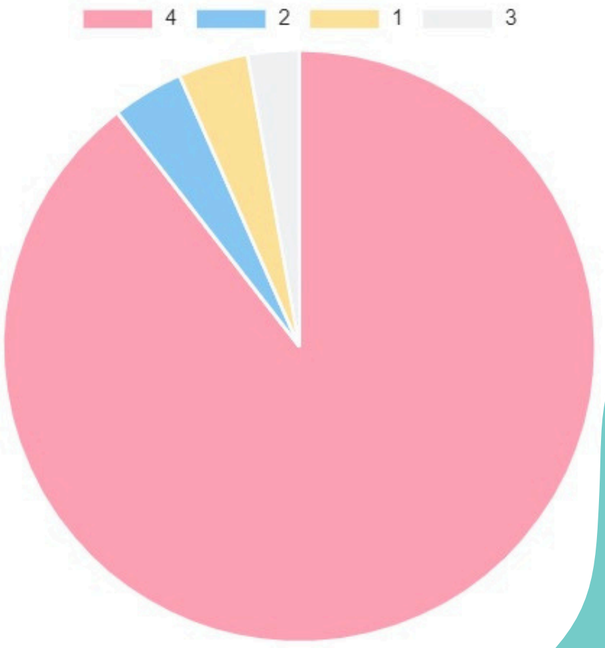
[52]

```sql
SELECT order_status, count(*) AS orders
FROM sales.orders
GROUP BY order_status
ORDER BY orders DESC;
```

(4 rows affected)

Total execution time: 00:00:00.004

Copy as image Save as image Configure Chart



Visualization of Results

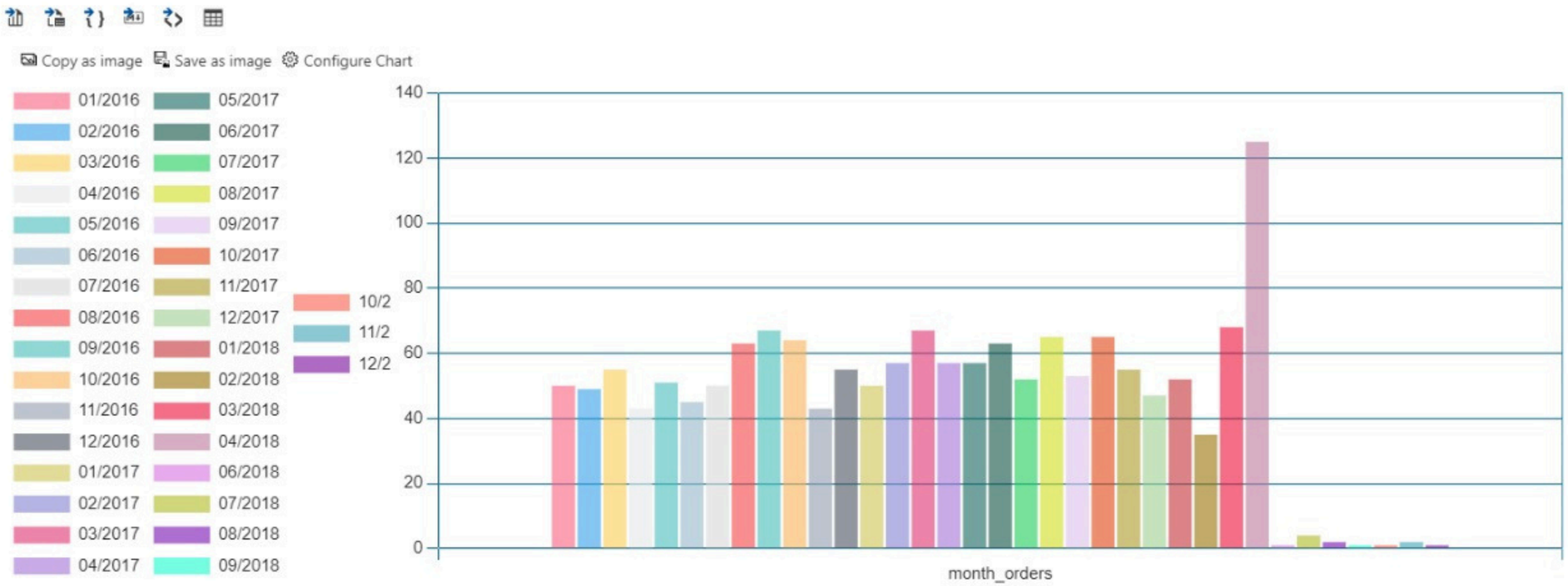
order per month

[88]

```sql
SELECT FORMAT(order_date, 'MM/yyyy'), COUNT(*) AS month_orders
FROM sales.orders
GROUP BY FORMAT(order_date, 'MM/yyyy')
ORDER BY MIN(Order_Date);
```

(35 rows affected)

Total execution time: 00:00:00.020



Database and Data Warehouse

- ## Creation of Dimension Tables :

dim.products: Contains product-related information (e.g., product_name, model_year, category_id, supplier_id), with foreign keys linking to production.categories and production.suppliers.

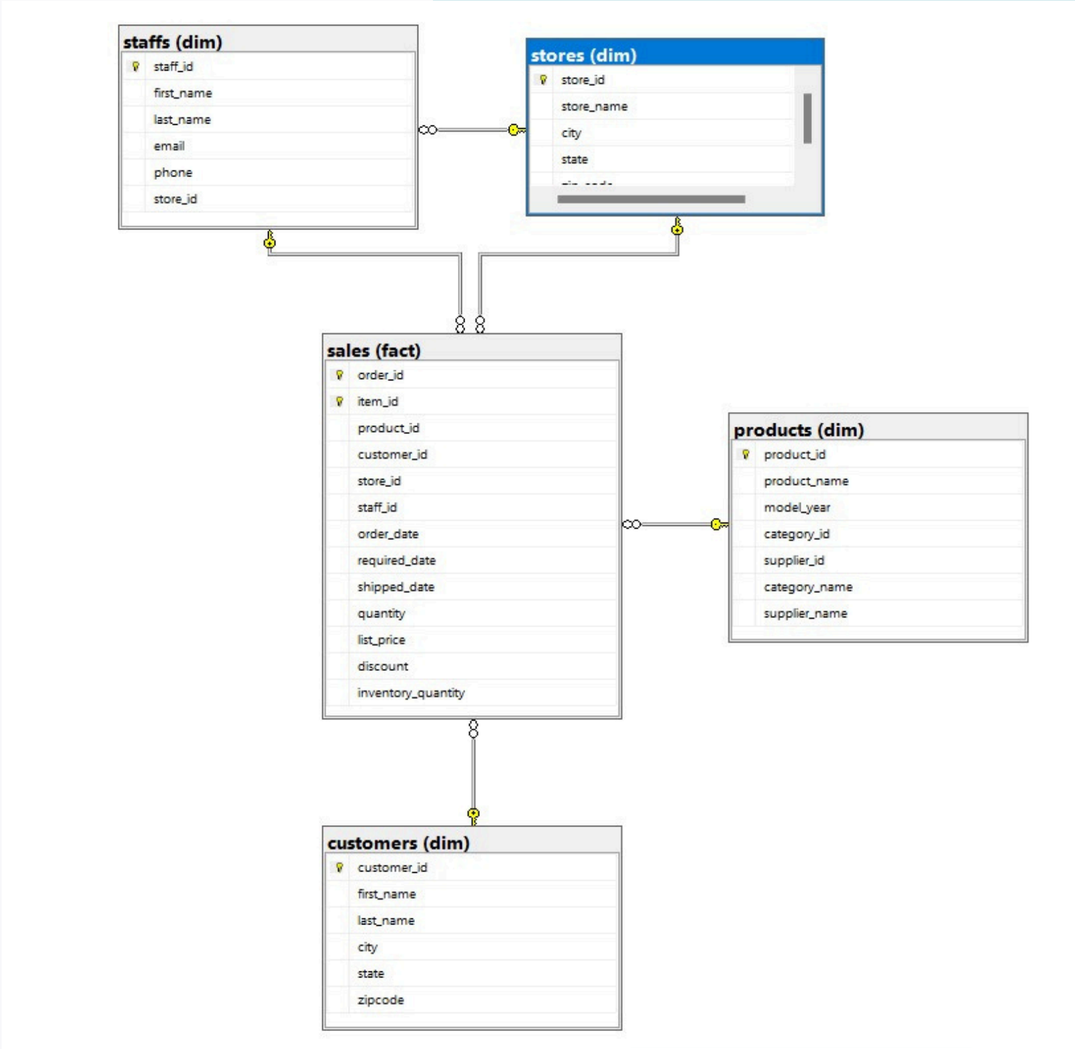
dim.stores: Stores information such as store_name, city, state, and zip_code.

dim.customers: Holds customer data, including first_name, last_name, city, and zipcode.

dim.staffs: Staff-related data, such as first_name, last_name, email, phone_number, and store_id, with a foreign key linking to dim.stores.

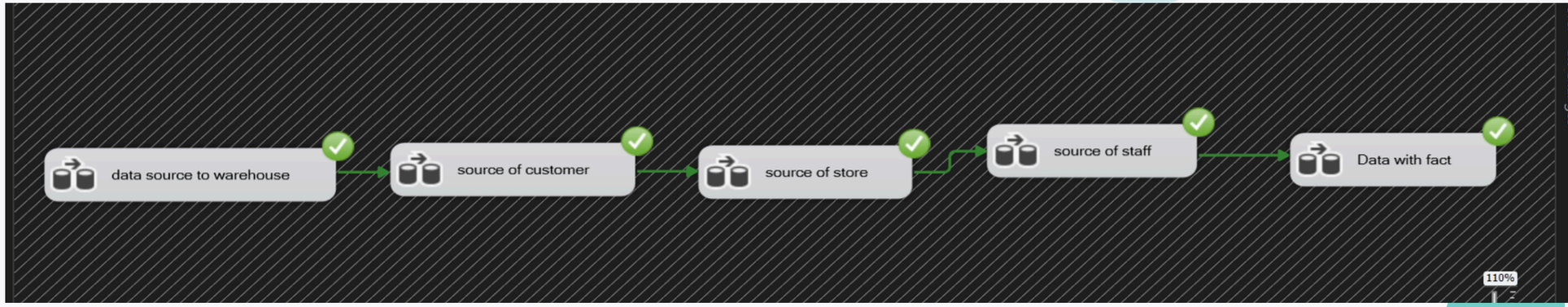
- ## Creation of Fact Tables :

fact.sales: A sales-related table with details like order_id, item_id, product_id, customer_id, store_id, staff_id, dates (e.g., order_date, shipped_date), quantity, list_price, discount, and inventory_quantity. Several foreign keys reference the dim tables for products, customers, stores, and staffs.



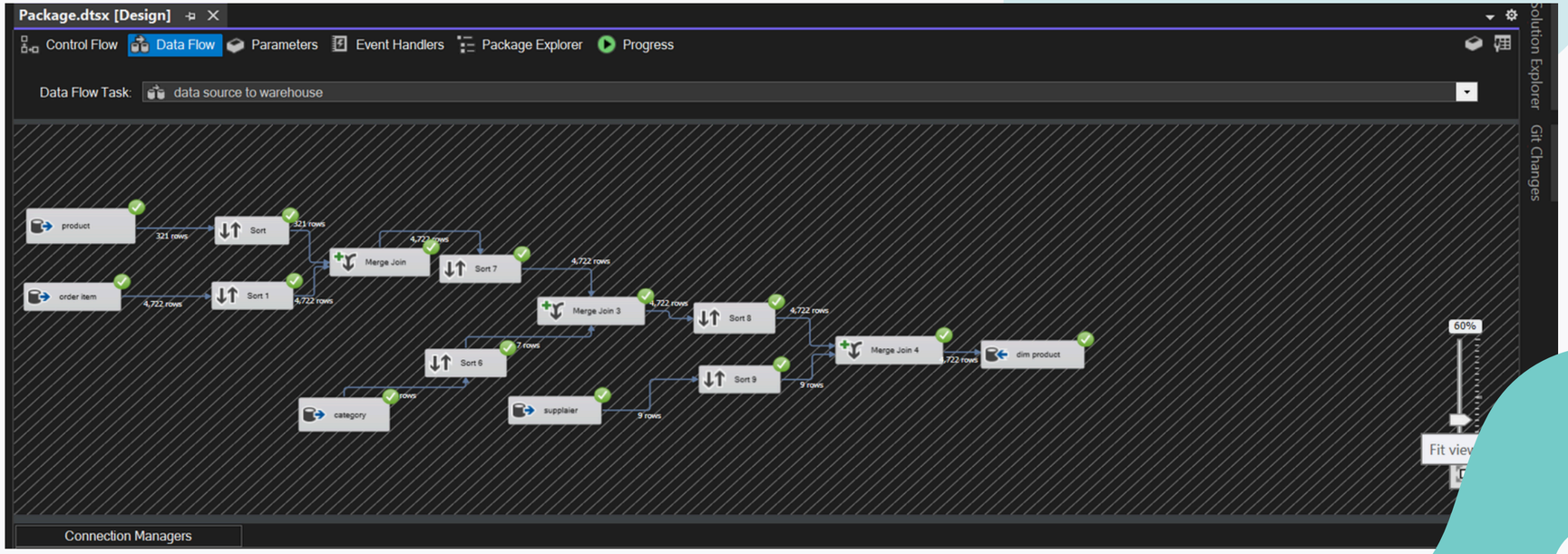
Database and Data Warehouse

- Data source to Data Warehouse



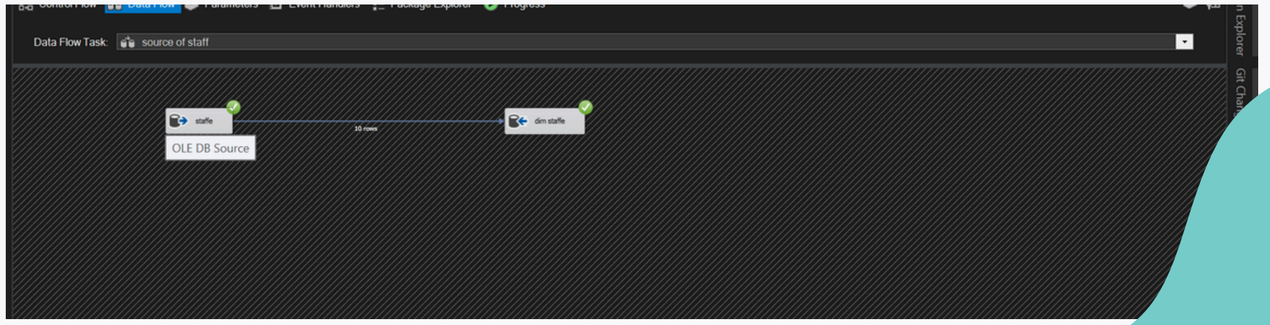
Database and Data Warehouse

- Transfer data from Data source of product to Dim product



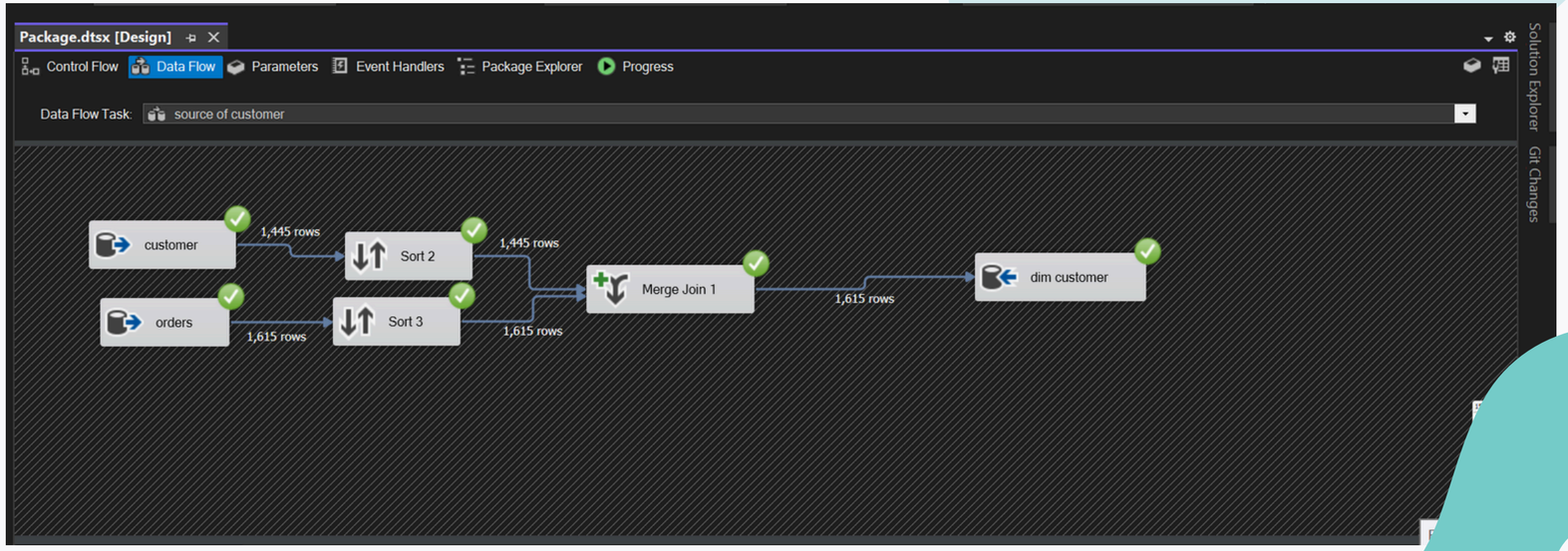
Database and Data Warehouse

- Transfer data from Data source of store and staff to Dim store and Dim staff
- Note : source staff must run before store



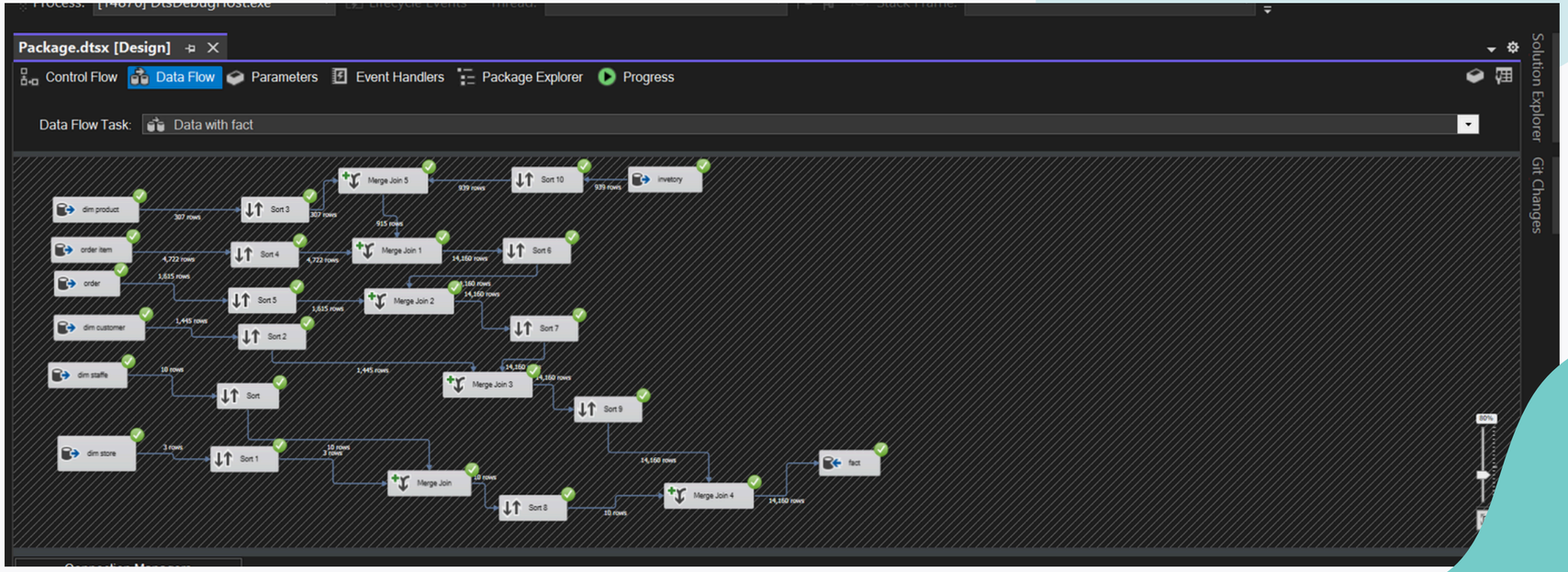
Database and Data Warehouse

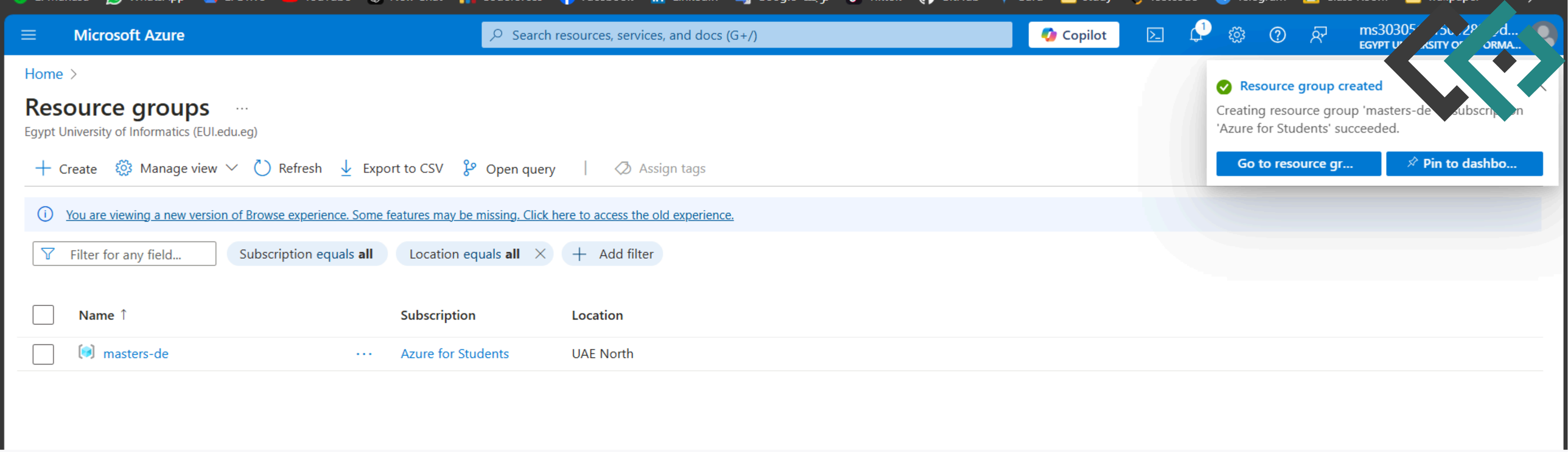
- Transfer data from Data source of Customer to Dim Customer



Database and Data Warehouse

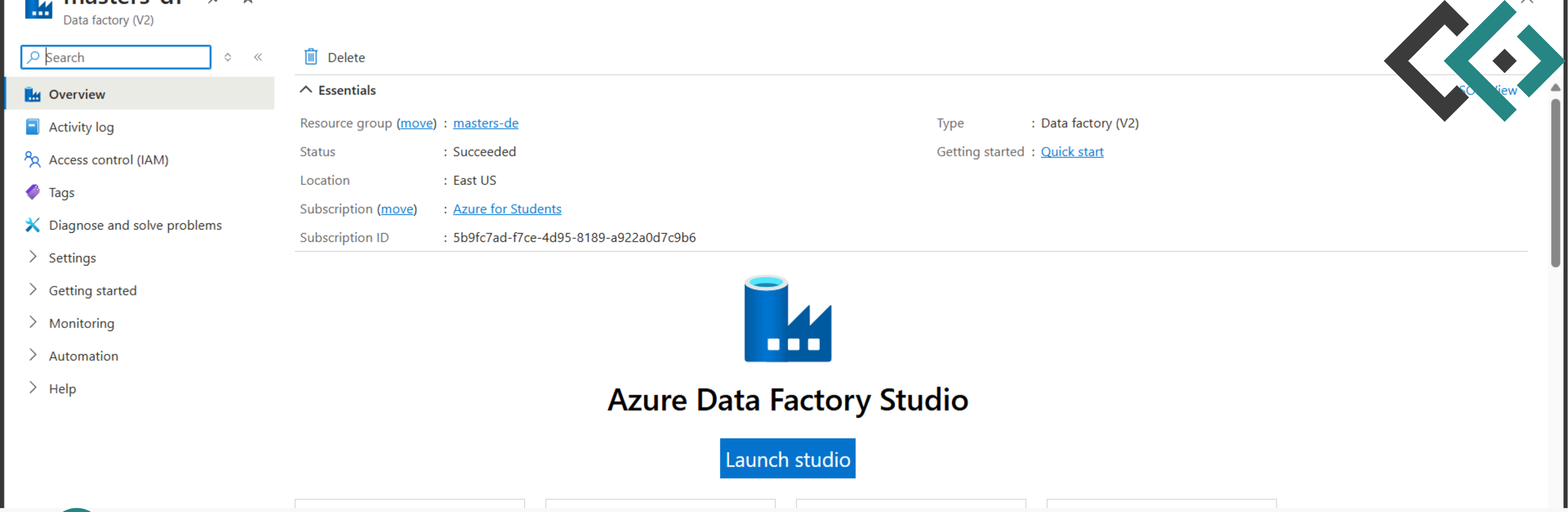
- Transfer data from Data Dimension to Fact





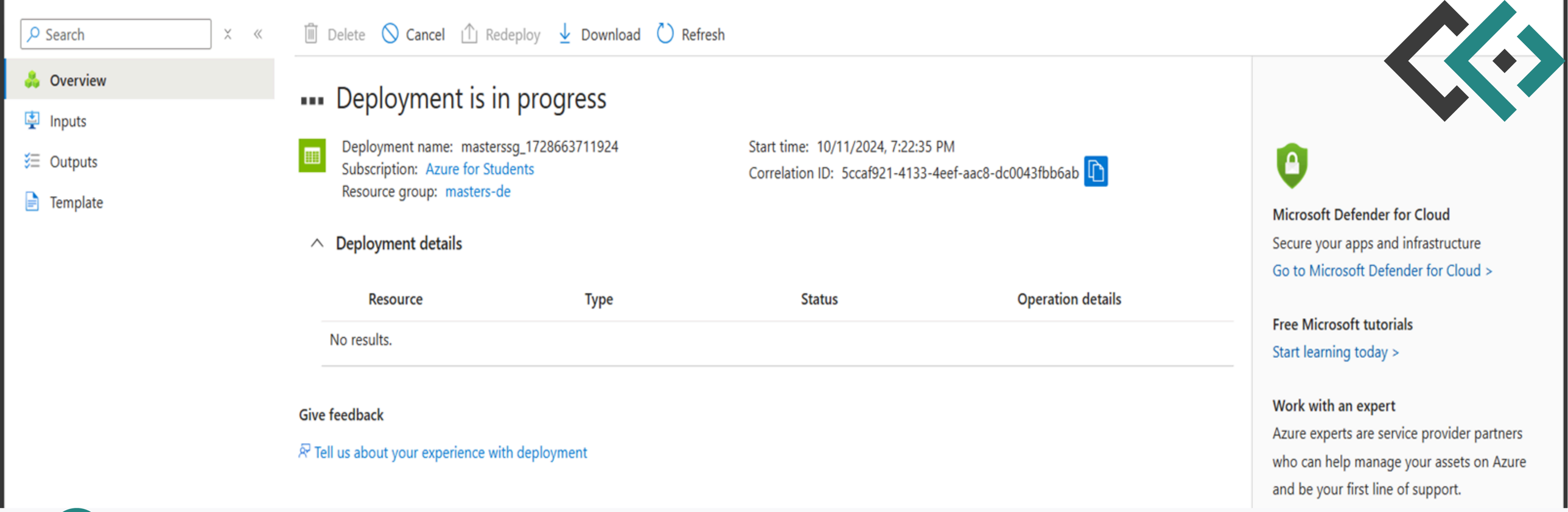
Technology Stack

As part of our Azure environment setup for the inventory retail project, we successfully created a resource group named masters-de. This resource group serves as a foundational container for organizing and managing all related Azure resources throughout the project lifecycle. By consolidating our services within this resource group, we ensure efficient resource management, streamlined monitoring, and improved collaboration among team members. The "masters-de" resource group will house essential components such as Azure Data Factory, Azure Data Lake Storage, Azure Databricks, and Azure Synapse Analytics, forming the backbone of our comprehensive data pipeline for inventory management and demand forecasting.



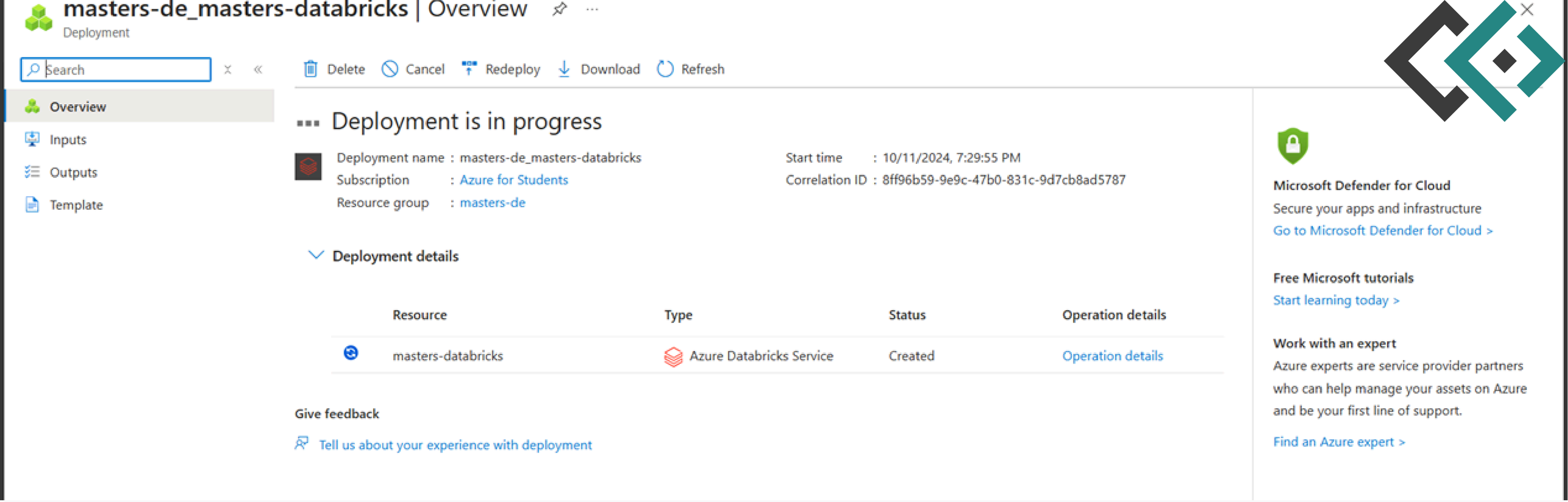
Technology Stack

To enhance our data integration capabilities for the inventory retail project, we successfully provisioned an Azure Data Factory named masters-df. This cloud-based data integration service allows us to orchestrate and automate data movement and transformation efficiently. With masters-df, we can create data-driven workflows that extract, transform, and load (ETL) data from our on-premises SQL database into Azure Data Lake Storage. This foundational component will enable us to build a robust data pipeline, facilitating better inventory management and demand forecasting through effective data processing.



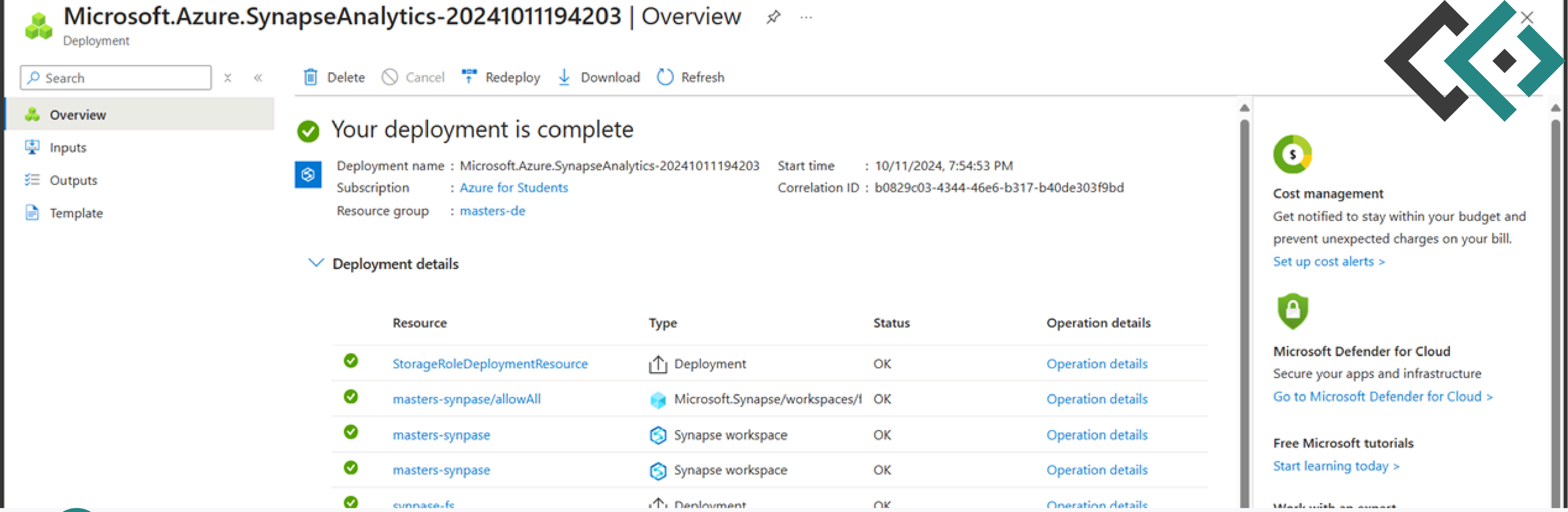
Technology Stack

As part of our Azure environment setup , we created a storage account named masterssg. This storage account is essential for hosting our data lake, enabling us to securely store and manage raw and processed data. With masterssg, we can leverage Azure Data Lake Storage capabilities to efficiently organize our data into different layers, including Bronze, Silver, and Gold, supporting our data transformation and analytics processes. This foundational storage solution plays a crucial role in facilitating data accessibility and scalability as we build our comprehensive data pipeline for inventory management and demand forecasting.



Technology Stack

To support our data transformation and analytics efforts in the inventory retail project, we successfully set up an Azure Databricks workspace named masters-databricks. This collaborative environment will host our data processing tasks across the Bronze, Silver, and Gold stages. The Bronze layer will contain raw data ingested from various sources, while the Silver layer will consist of cleaned and transformed data, ready for analysis. Finally, the Gold layer will include aggregated, analysis-ready data that enables us to build and validate our demand forecasting models. The masters-databricks workspace is crucial for leveraging Apache Spark's powerful analytics capabilities, facilitating efficient data manipulation and visualization throughout the project.



Technology Stack

In line with our goal to develop a comprehensive data pipeline for the inventory retail project, we have successfully provisioned master-synapse, an Azure Synapse Analytics workspace. This powerful analytics service will allow us to perform large-scale data analysis and reporting. With master-synapse, we can integrate data from the Gold layer of our Databricks pipeline, enabling SQL-based analytics and further refining the insights we generate from the forecasting models. Azure Synapse also supports advanced querying, reporting, and real-time analysis, ensuring that our data is readily available for decision-makers to make informed, data-driven choices.

Data Ingestion

Structuring the Data Lake: Bronze, Silver, and Gold Files

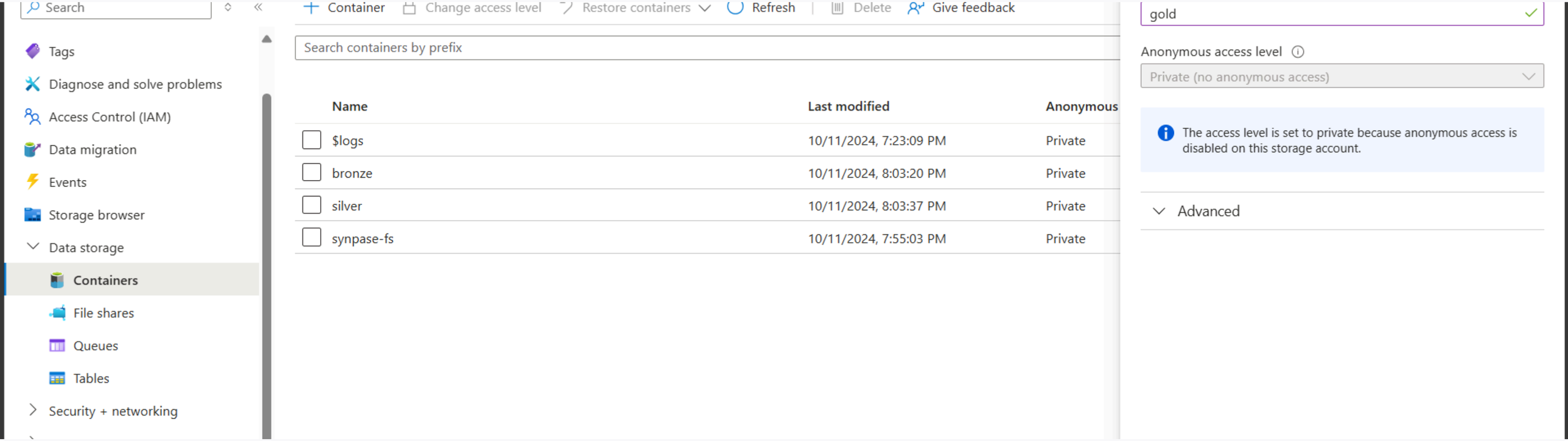
In the masterssg storage account, we have organized the data lake by creating three distinct directories: Bronze, Silver, and Gold. These stages form the backbone of our data transformation pipeline:

Bronze: Contains raw, ingested data from various sources, including the on-premises SQL database. This is the initial landing area for unprocessed data.

Silver: Hosts cleaned and transformed data, making it suitable for further analysis. Here, data quality improvements are made, ensuring it is ready for downstream processes.

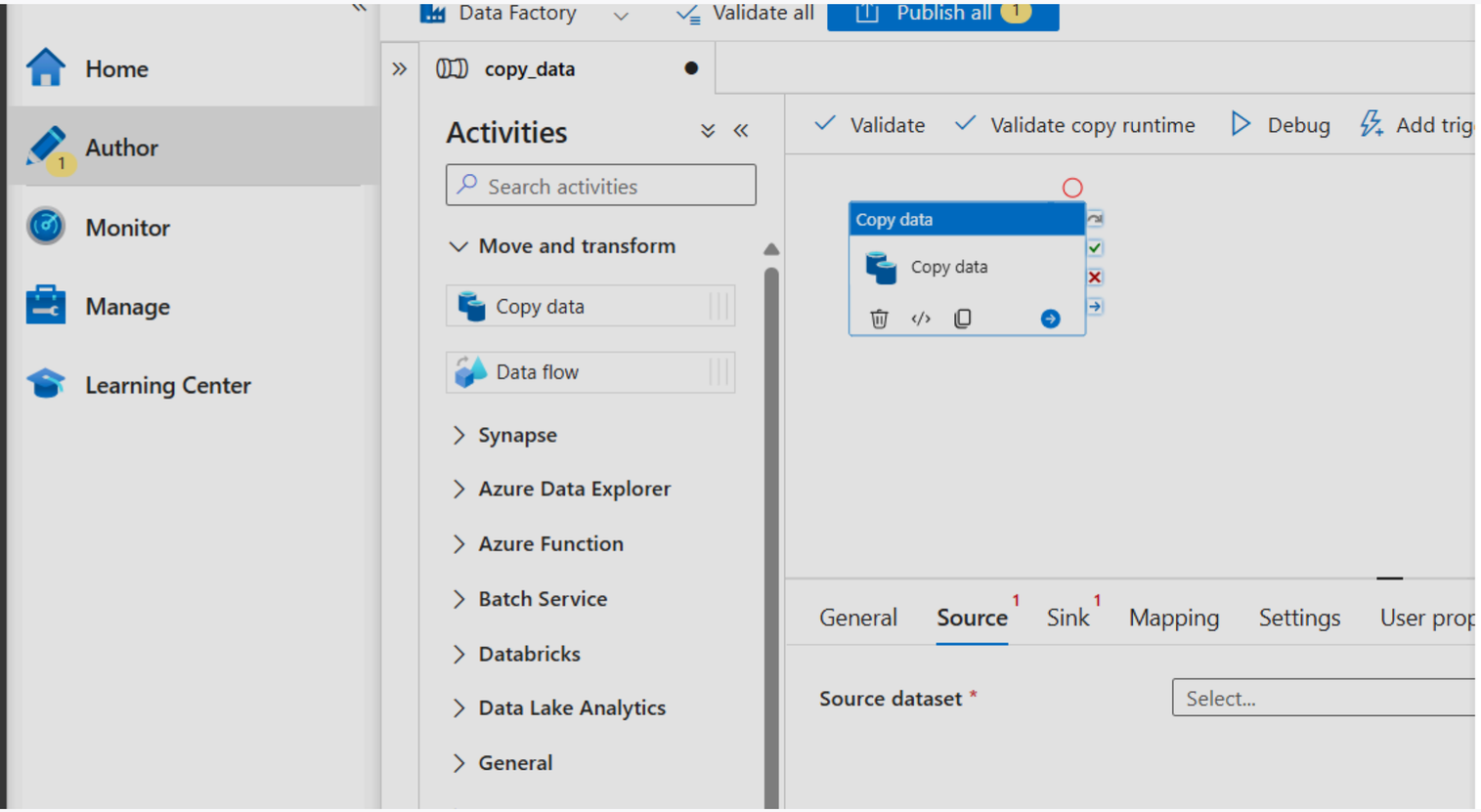
Gold: The final layer, holding aggregated and analysis-ready data. This layer is used for reporting, machine learning models, and insights generation.

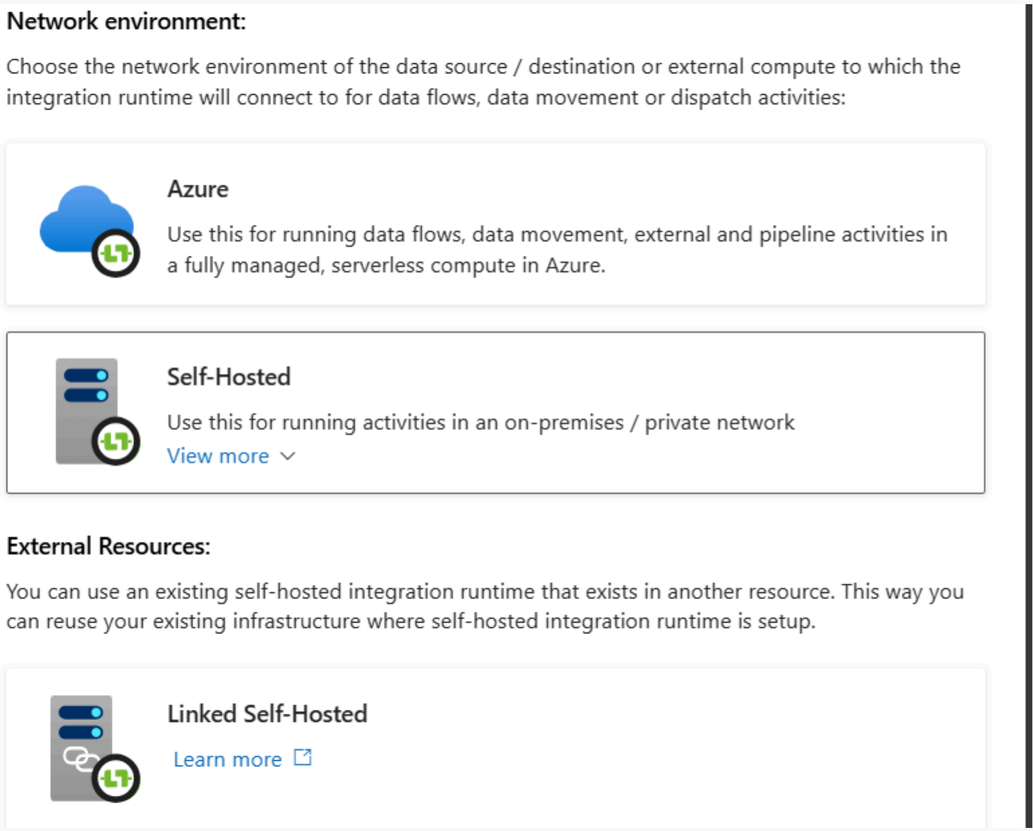
These layers streamline the process of moving data through various transformation stages, ensuring data integrity and consistency as we prepare it for demand forecasting and reporting.



Data Ingestion

setting up a self-hosted integration runtime for data ingestion, you've enabled secure communication between your on-premises SQL Server and Azure Data Factory (ADF). This allows ADF to extract data from your SQL database and load it into Azure Data Lake Storage (ADLS). This step is crucial for moving your raw data to the bronze layer, forming the foundation for your data transformation and analysis processes in Azure.





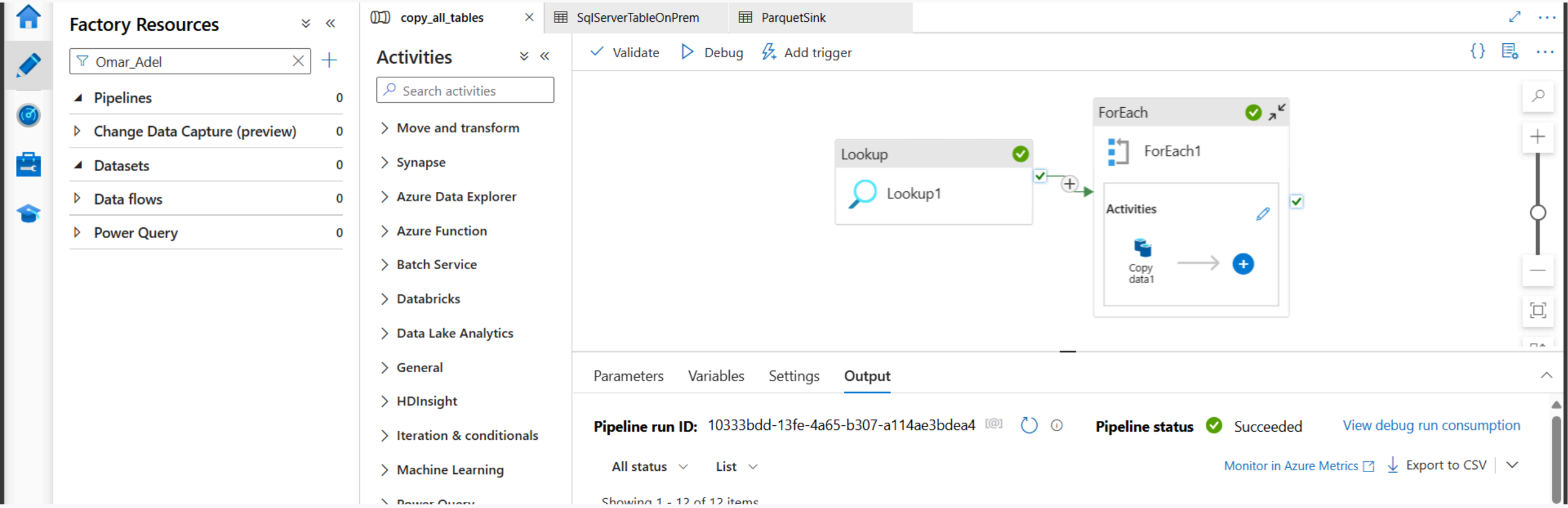
Data Ingestion

By creating a Lookup pipeline and linking it to a ForEach activity in Azure Data Factory (ADF), you've automated the process of ingesting data into the bronze stage.

The Lookup activity retrieves the table metadata (such as table names or paths) from a source like a configuration file or database.

The ForEach activity then iterates over the list of tables, executing the data ingestion process for each one.

This setup ensures that data from multiple tables is efficiently extracted from your SQL Server and loaded into the bronze layer in Azure Data Lake Storage (ADLS), making the raw data available for the next stages of transformation.



Data Transformation

her we mounted our Azure Data Lake Storage (ADLS) Gen2 containers to the Databricks file system, which is a key step in accessing data across the bronze, silver, and gold layers.

Bronze layer: Stores raw data.

Silver layer: Stores cleaned and transformed data.

Gold layer: Stores aggregated, ready-for-analysis data.

Each of these containers is now accessible within Databricks through the respective mount points (/mnt/bronze, /mnt/silver, and /mnt/gold). This will allow you to easily read and write data as you proceed with data transformations and machine learning workflows.

5 days ago (13s)

```
configs = {
    "fs.azure.account.auth.type": "CustomAccessToken",
    "fs.azure.account.custom.token.provider.class": spark.conf.get("spark.databricks.passthrough.adls.gen2.tokenProviderClassName")
}

# Mount ADLS Gen2 to Databricks file system
dbutils.fs.mount(
    source = "abfss://bronze@masterssg.dfs.core.windows.net/",
    mount_point = "/mnt/bronze",
    extra_configs = configs)
```

Data Transformation

This code verifies the transformations of sales and production tables in the silver layer of Azure Data Lake. It defines the table names for both domains, constructs the appropriate paths for each table, and loads the transformed data in Delta Lake format. The script checks the schema and displays a sample of the data for validation. If errors occur (e.g., missing data), they are caught and reported, ensuring the data is correctly transformed and ready for further use.

2 days ago (2s)

```python
# Define the table names for sales and production
sales_tables = ['customers', 'order_items', 'orders', 'staffs', 'stores']
production_tables = ['categories', 'inventory_levels', 'products', 'suppliers']

# Combine the table names into one list
all_tables = sales_tables + production_tables

# Loop over each table to verify transformations
for table in all_tables:
    # Determine the schema (sales or production)
    schema = 'sales' if table in sales_tables else 'production'

    # Construct the path to the transformed table in the silver layer
    path = f'/mnt/silver/{schema}/{table}/'
    print(f"Verifying data at: {path}")  # Print the path for debugging

    # Read the transformed DataFrame from the silver layer
    try:
        df = spark.read.format('delta').load(path)

        # Show the schema to verify column types
        print(f"Schema for {table}:")
        df.printSchema()

        # Display a sample of the transformed data
        print(f"Sample data for {table}:")
        df.show(5)  # Show the first 5 records
    except Exception as e:
        print(f"Error reading transformed data at {path}: {e}")
```

Data Transformation

2 days ago (<1s)

```python
table_name = []

for i in dbutils.fs.ls('mnt/silver/sales'):
    table_name.append(i.name.split('/')[0])
```

2 days ago (<1s)

```python
table_name
```

```
['customers', 'order_items', 'orders', 'staffs', 'stores']
```

2 days ago (8s)

```python
for name in table_name:
    path = '/mnt/silver/sales/' + name
    print(path)
    df = spark.read.format('delta').load(path)

    df = rename_columns_to_snake_case(df)

    output_path = '/mnt/gold/ratail_inventory/' + name + '/'
    df.write.format('delta').mode('overwrite').save(output_path)
```

2 days ago (<1s)

```python
table_name = []

for i in dbutils.fs.ls('mnt/silver/production'):
    table_name.append(i.name.split('/')[0])
```

2 days ago (6s)

```python
for name in table_name:
    path = '/mnt/silver/production/' + name
    print(path)
    df = spark.read.format('delta').load(path)

    df = rename_columns_to_snake_case(df)

    output_path = '/mnt/gold/ratail_inventory/' + name + '/'
    df.write.format('delta').mode('overwrite').save(output_path)
```

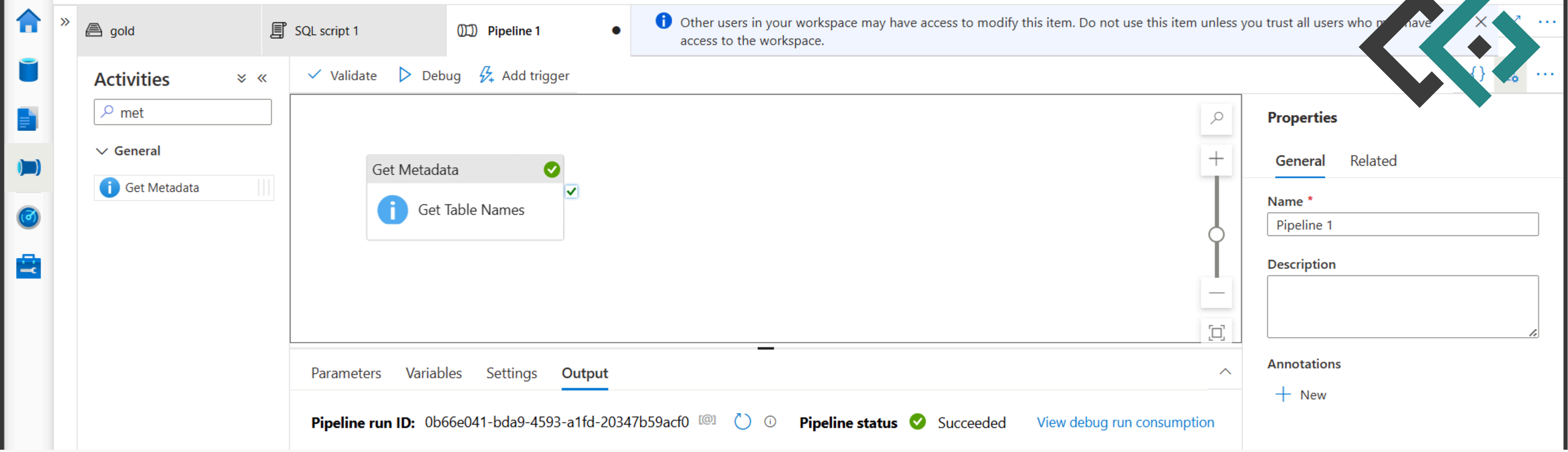
(16) Spark Jobs

Authentication method: Access key (Switch to Microsoft Entra user account)
Location: gold / ratail_inventory

Search blobs by prefix (case-sensitive)

Name	Modified
[..]	
categories	
customers	
inventory_levels	
order_items	
orders	
products	
staffs	
stores	
suppliers	

This script performs the final transformation of sales and production data from the silver layer to the gold layer, grouping both under the inventory_retail structure. By renaming columns to snake case and saving the cleaned data, it prepares the dataset for further analysis and machine learning. This process ensures that all relevant tables are organized and optimized for insights into inventory management.



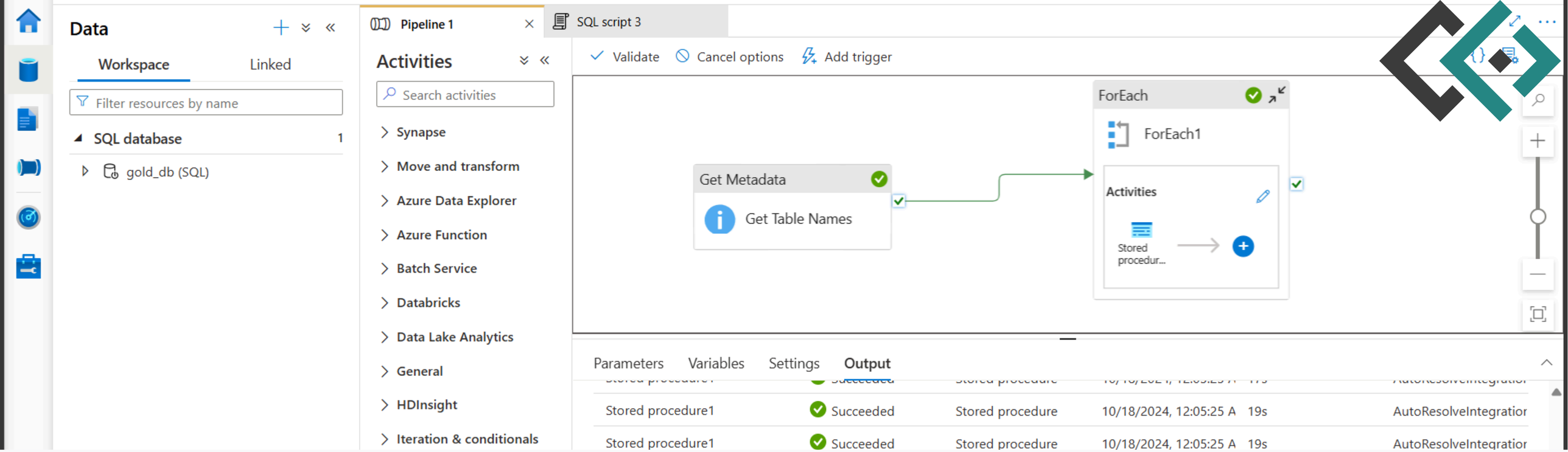
Synpase Analysis

we created a metadata pipeline called Get Table Name with an argument named child item that retrieves data from the inventory_retail structure. This pipeline likely serves the purpose of dynamically fetching the names of tables or files within the inventory_retail dataset, enabling easier management and processing of the data. By using this pipeline, you can streamline the workflow for data ingestion and transformation, ensuring that all relevant tables are consistently accessed for analysis and machine learning tasks.

Synpase Analysis

we developed a stored procedure called CreateSQLServerlessview_gold in the gold_db database that dynamically creates or alters a SQL view based on the provided view name. It utilizes the OPENROWSET function to access Delta data from your Azure Data Lake Storage (ADLS), enabling seamless querying of inventory data. The procedure enhances data accessibility and streamlines the process of integrating your gold layer data into SQL Serverless analytics.
4o mini

```sql

CREATE OR ALTER PROC CreateSQLServerlessview_gold @ViewName NVARCHAR(100)
AS
BEGIN
    DECLARE @statement VARCHAR(MAX)
    SET @statement = N'CREATE OR ALTER VIEW ' + @viewName + 'AS
    SELECT
        *
    FROM
        OPENROWSET(
            BULK = ''https://masterssg.dfs.core.windows.net/gold/sales/' + @viewName + '/'',
            FORMAT = ''DELTA''
        ) AS [result]'

    EXEC (@statement)

END
```



Synpase Analysis

we've implemented a foreach loop that incorporates the stored procedure CreateSQLServerlessview_gold and links it with the metadata pipeline Get Table Name. This setup allows you to iterate through the table names retrieved by the Get Table Name pipeline, dynamically creating or altering SQL views for each table in your inventory_retail dataset. This automation simplifies the process of managing views in your SQL Serverless environment, ensuring that all relevant data is accessible for analysis and reporting.

Building the Forecasting Model

- Imports necessary libraries for data processing and visualization
- Initializes a Spark session named "Forecasting.
- Loads sales data from a Delta Lake table using the specified path

```python
# Load the necessary libraries
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
import matplotlib.pyplot as plt

# Create a Spark session
spark = SparkSession.builder.appName("Forecasting").getOrCreate()

# Read the sales data from the gold layer
sales_df = spark.read.format('delta').load('/mnt/gold/ratail_inventory/orders/')  # Adjust the path based on your data s

# Show the schema and a few rows
sales_df.printSchema()
sales_df.show()
```

```
root
 |-- order_id: integer (nullable = true)
 |-- customer_id: integer (nullable = true)
 |-- order_status: integer (nullable = true)
 |-- order_date: date (nullable = true)
 |-- required_date: date (nullable = true)
 |-- store_id: integer (nullable = true)
 |-- staff_id: integer (nullable = true)
 |-- shipped_date: date (nullable = true)

+--------+-----------+------------+----------+-------------+--------+--------+------------+
|order_id|customer_id|order_status|order_date|required_date|store_id|staff_id|shipped_date|
+--------+-----------+------------+----------+-------------+--------+--------+------------+
|       1|        259|           4|2016-01-01|   2016-01-03|       1|       2|  2016-01-03|
|       2|       1212|           4|2016-01-01|   2016-01-04|       2|       6|  2016-01-03|
|       3|        523|           4|2016-01-02|   2016-01-05|       2|       7|  2016-01-03|
|       4|        175|           4|2016-01-03|   2016-01-04|       1|       3|  2016-01-05|
|       5|       1324|           4|2016-01-03|   2016-01-06|       2|       6|  2016-01-06|
|       6|         94|           4|2016-01-04|   2016-01-07|       2|       6|  2016-01-05|
|       7|        324|           4|2016-01-04|   2016-01-07|       2|       6|  2016-01-05|
|       8|       1204|           4|2016-01-04|   2016-01-05|       2|       7|  2016-01-05|
|       9|         60|           4|2016-01-05|   2016-01-08|       1|       2|  2016-01-08|
|      10|        442|           4|2016-01-05|   2016-01-06|       2|       6|  2016-01-06|
|      11|       1326|           4|2016-01-05|   2016-01-08|       2|       7|  2016-01-07|
|      12|         91|           4|2016-01-06|   2016-01-08|       1|       2|  2016-01-09|
|      13|        873|           4|2016-01-08|   2016-01-11|       2|       6|  2016-01-11|
|      14|        258|           4|2016-01-09|   2016-01-11|       1|       3|  2016-01-12|
|      15|        450|           4|2016-01-09|   2016-01-10|       2|       7|  2016-01-12|
|      16|        552|           4|2016-01-12|   2016-01-15|       1|       3|  2016-01-15|
|      17|       1175|           4|2016-01-12|   2016-01-14|       1|       3|  2016-01-14|
|      18|        541|           4|2016-01-14|   2016-01-17|       1|       3|  2016-01-15|
|      19|        696|           4|2016-01-14|   2016-01-17|       1|       2|  2016-01-16|
|      20|        923|           4|2016-01-14|   2016-01-16|       1|       2|  2016-01-17|
+--------+-----------+------------+----------+-------------+--------+--------+------------+
only showing top 20 rows
```

Building the Forecasting Model

- Merging DataFrames & Aggregating Sales Data

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, sum as _sum

# Initialize Spark session
spark = SparkSession.builder.appName("Sales Analysis").getOrCreate()

# Load DataFrames from your inventory
orders = spark.read.format('delta').load('/mnt/gold/ratail_inventory/orders/')
order_items = spark.read.format('delta').load('/mnt/gold/ratail_inventory/order_items/')
products = spark.read.format('delta').load('/mnt/gold/ratail_inventory/products/')

# Ensure the 'order_date' column is of date type
orders = orders.withColumn("order_date", col("order_date").cast("date"))

# Merge DataFrames
orders_items_merged = order_items.join(orders, on='order_id', how='inner')
sales_data = orders_items_merged.join(products, on='product_id', how='inner')

# Aggregate daily sales
daily_sales_df = sales_data.groupBy("order_date").agg(
    _sum(order_items["list_price"]).alias("daily_sales")
)

# Show the aggregated daily sales
display(daily_sales_df)
```

order_date	daily_sales
2016-03-01	7930.94
2017-08-11	1432.98
2017-09-11	2151.98
2017-01-06	9536.91
2018-03-17	4375.94
2016-05-03	5736.95
2016-07-26	6798.97
2016-08-15	5768.96
2016-08-31	15188.90
2016-10-03	6968.96
2016-01-28	8039.95
2017-01-27	599.99
2017-02-26	13556.91
2017-09-28	2101.96
2016-07-17	9216.91
2016-11-08	2119.96
2016-12-19	2899.97
2017-01-24	4301.96
2016-08-23	4940.95

Building the Forecasting Model

- Forecasting Model using Prophet
- Import and Set Up MLflow

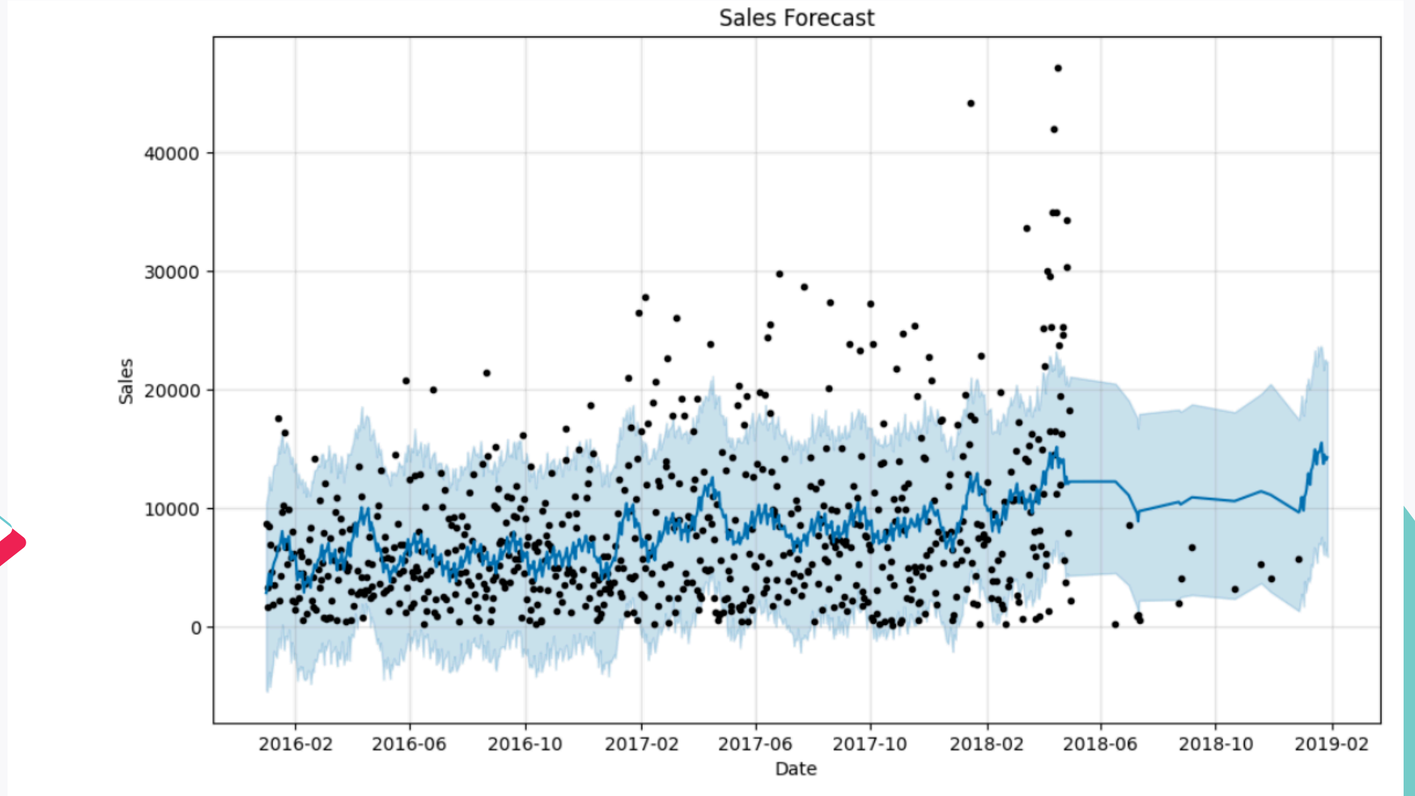
```python
from prophet import Prophet
import pandas as pd
import matplotlib.pyplot as plt

# Convert Spark DataFrame to Pandas DataFrame
daily_sales_pd = daily_sales_df.toPandas()

# Prepare the data for Prophet
daily_sales_pd = daily_sales_pd.rename(columns={'order_date': 'ds', 'daily_sales': 'y'})

# Initialize the Prophet model
model = Prophet()

# Fit the model
model.fit(daily_sales_pd)

# Create a DataFrame for future dates
future_dates = model.make_future_dataframe(periods=30)  # Forecasting for the next 30 days

# Predict future sales
forecast = model.predict(future_dates)
```

```python
import mlflow.pyfunc
import mlflow.prophet

# Start an MLflow run
mlflow.end_run()
mlflow.start_run(run_name="Prophet_Sales_Forecast")

# Cell 11: Log Model and Parameters with MLflow
# Log the model
mlflow.prophet.log_model(model, "prophet_model")

# Log parameters
mlflow.log_param("forecast_periods", 30)

# Ensure the directory exists
dbutils.fs.mkdirs("dbfs:/tmp")

# Log the forecast DataFrame as an artifact
forecast.to_csv("/dbfs/tmp/forecast.csv", index=False)
mlflow.log_artifact("/dbfs/tmp/forecast.csv")

# End the MLflow run
mlflow.end_run()
```

Building the Forecasting Model

- Display the Forecast Plot

```
# Plot the forecast
fig = model.plot(forecast)
plt.title('Sales Forecast')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.show()
```



Building the Forecasting Model

- EVALUATION

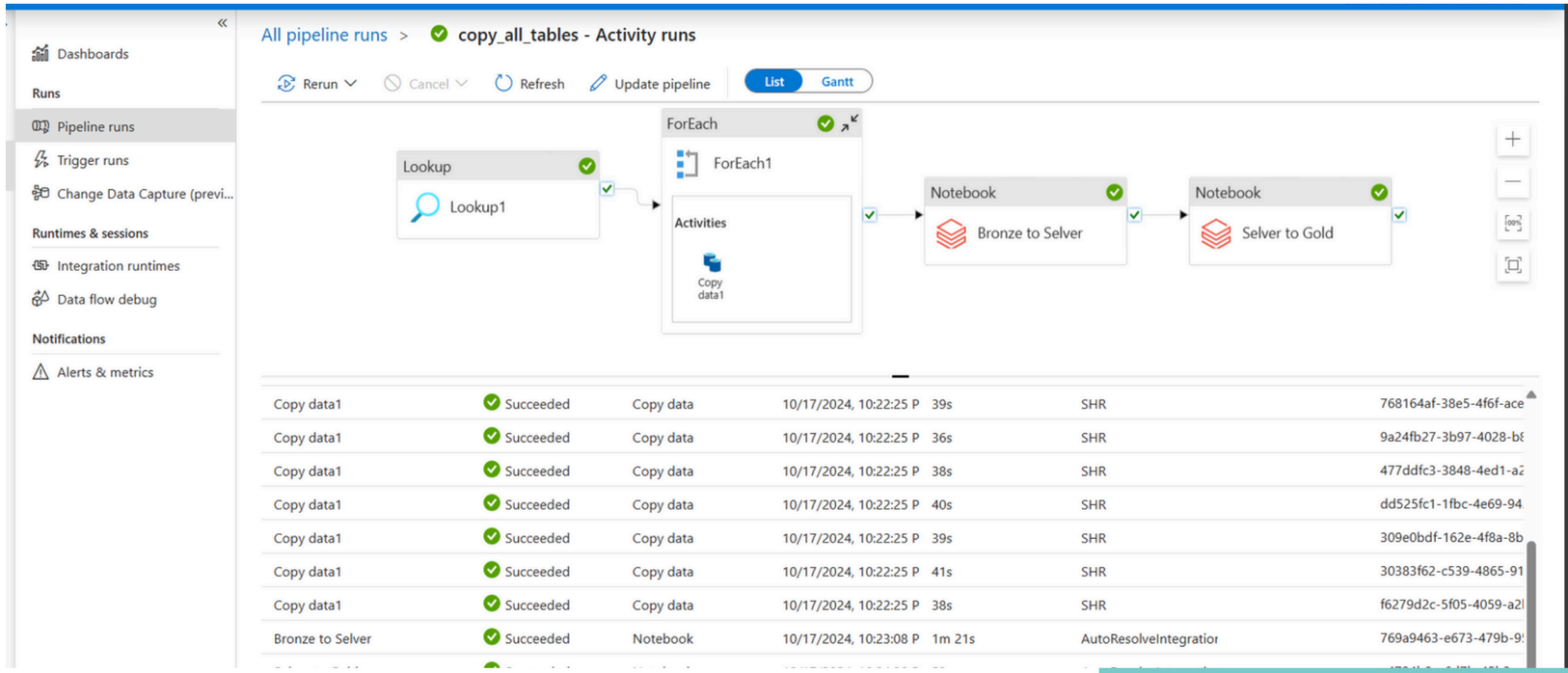
```
# حساب متوسط القيم الحقيقية
mean_actual = np.mean(test)

# إلى نسبة مئوية تحويل MAE و RMSE
mae_percent = (mae / mean_actual) * 100
rmse_percent = (rmse / mean_actual) * 100

print(f'MAE as Percentage: {mae_percent:.2f}%, RMSE as Percentage: {rmse_percent:.2f}%')
```

```
AE as Percentage: 5.25%, RMSE as Percentage: 5.27%
```

Automation and Monitoring

By linking the lookup activity to the foreach loop, you've automated the process of transitioning data from the bronze to silver layers in the Data Lake and then from the silver to gold layers in Databricks. This automation reduces manual intervention, ensuring that data flows seamlessly through the pipeline. As a result, your inventory management project benefits from timely and consistent updates, enabling more accurate analysis and forecasting with minimal effort.

4o mini



Thank You

DEPi

by

Omar Adel El Sheikh
mohamed Mahmoud mowafy
Samia Soliman El seady
Mai Abd El Mohsen Sidek
Mahmoud Ahmed Elsheikh