Walmart

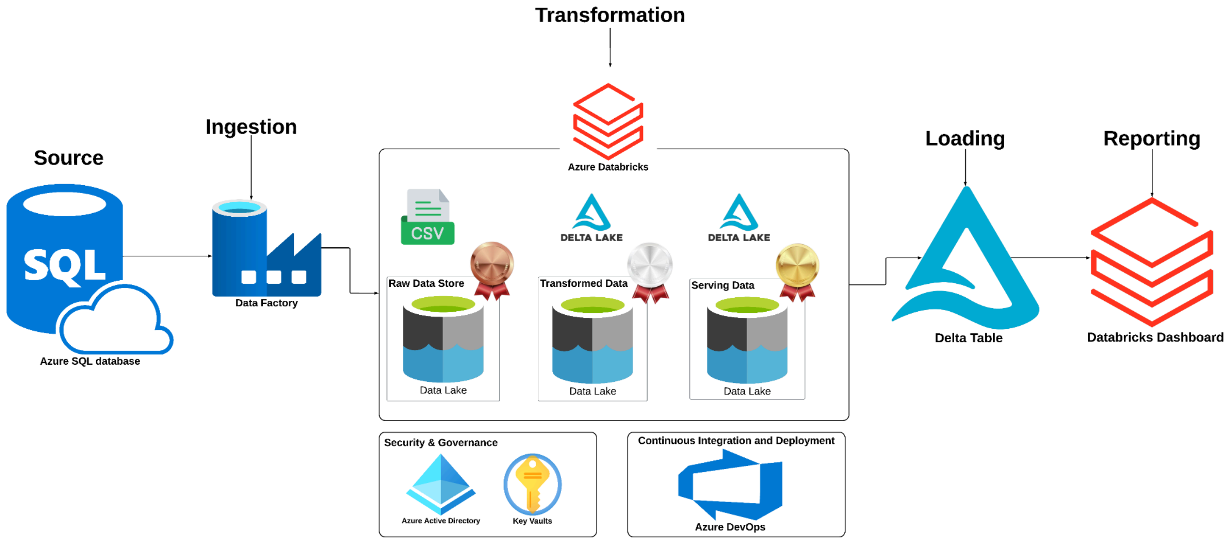
Problem Statement

Walmart, as a major retail corporation, faces the challenge of accurately predicting future sales. This is crucial for various reasons:

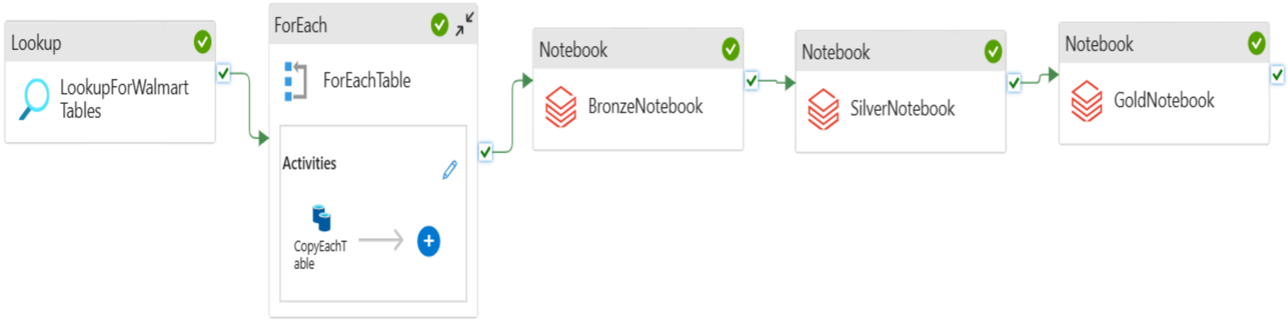
- **Inventory Management:** Optimal stock levels are essential to meet demand without incurring excessive holding costs of stockouts.
- **Revenue Forecasting:** Accurate sales forecasts enable the company to plan its revenue and budget accordingly.
- **Investment Decisions:** Informed investment decisions can be made based on predicted future sales.
- **Stock Price and Investor Perception:** Meeting or exceeding sales targets can positively impact stock prices and investor confidence, while underperforming can have negative consequences.

A significant factor influencing sales is the presence of seasonal trends. Identifying and understanding these seasonal patterns is vital for making informed business decisions. By accurately predicting future sales, Walmart can optimize its operations, mitigate risks, and capitalize on opportunities.

Solution Architecture



Pipeline Architecture



Overview of Datasets

Overview of Dataset

This project leverages a rich dataset encompassing **historical sales data** from 45 Walmart stores across various regions. The dataset provides insights into department-level sales patterns and includes critical external factors such as promotional markdowns, holidays, and regional economic conditions. Below is a detailed breakdown of the dataset:

---

1. Dataset Files

File Name	Description
stores.csv	Contains anonymized details about the 45 Walmart stores, including their type and size.(**45 rows**)
train.csv	Historical sales data (2010-02-05 to 2012-11-01) for department-level sales predictions.(**421570 rows**)
test.csv	Similar to `train.csv`, but with weekly sales values withheld. Used for testing predictions.(**115064 rows**)
features.csv	Additional data about stores, departments, and regional activity, including markdowns and holidays.(**8190 rows**)

---

2. Key Features

- **Store and Department Information**
  - *Store*: Unique store identifier.
  - *Dept*: Unique department identifier within each store.
- **Sales and Temporal Data**
  - *Date*: Weekly sales data is recorded.
  - *Weekly_Sales*: Total sales for the specific department in a given store.

    - *IsHoliday*: Indicates if the week includes a prominent holiday (e.g., Super Bowl, Christmas).
- **Promotional and Regional Factors**
    - *MarkDown1-5*: Data related to promotional markdown events (available after Nov 2011).
    - *Temperature*: Average regional temperature during the week.
    - *Fuel_Price*: Regional fuel price during the week.
    - *CPI*: Consumer Price Index, indicating economic conditions.
    - *Unemployment*: Regional unemployment rate.

---

4. Summary of Dataset Dimensions

File	Records	Fields
stores.csv	**45 stores**	2 (Store Type, Store Size)
train.csv	**~421,570 entries**	5 (Store, Dept, Date, Weekly_Sales, IsHoliday)
test.csv	**~115,064 entries**	4 (Store, Dept, Date, IsHoliday)
features.csv	**~819,780 entries**	8 (Store, Date, Temperature, Fuel_Price, MarkDown1-5, CPI, Unemployment, IsHoliday)

This comprehensive dataset provides the foundation for building a robust sales forecasting model that captures the nuances of seasonal sales patterns and external factors affecting Walmart’s operations.

Data Ingestion

To efficiently ingest and store the data for this project, We utilized **Azure Data Factory (ADF)** to create a dynamic and scalable pipeline. The process is as follows:

1. **Source Database**
    - The data resided in an **Azure SQL Database**, containing multiple tables with relevant historical sales and store information.
2. **Dynamic Pipeline in ADF**
    - A **dynamic pipeline** was designed in Azure Data Factory to automate the data ingestion process.
    - The pipeline dynamically iterates over all the tables in the Azure SQL Database.
3. **Data Storage in ADLS Gen 2**
    - The pipeline extracts data from each table and stores it in **Azure Data Lake Storage (ADLS) Gen 2**.
    - The extracted data is saved in **CSV format**, ensuring compatibility and ease of use for downstream processes such as data transformation and analysis.

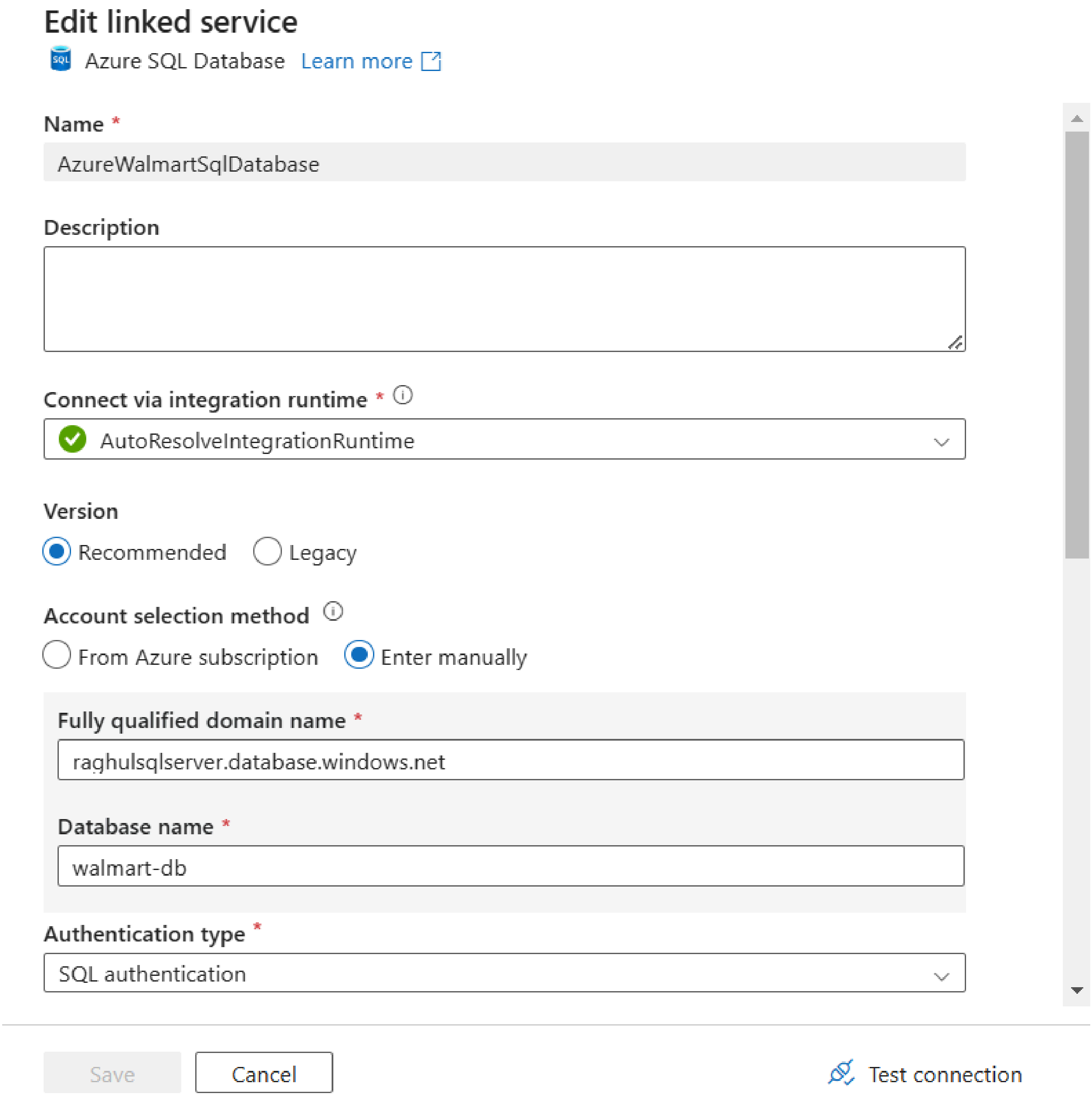
This automated data ingestion approach ensures scalability, reliability, and reusability, enabling efficient handling of large datasets and seamless integration into the data pipeline.

Steps for Data Ingestion

The following steps were implemented to extract data from **Azure SQL Database** and store it in **Azure Data Lake Storage (ADLS) Gen 2** using **Azure Data Factory (ADF):**

1. **Set Up Azure Data Factory**
    - Created an **Azure Data Factory instance** in the Azure portal.
    - Configured **Linked Services** for:

- **Azure SQL Database** (source).



Edit linked service

Azure SQL Database Learn more

Name *

AzureWalmartSqlDatabase

Description

Connect via integration runtime *

✓ AutoResolveIntegrationRuntime

Version

● Recommended ○ Legacy

Account selection method

○ From Azure subscription ● Enter manually

Fully qualified domain name *

raghulsqlserver.database.windows.net

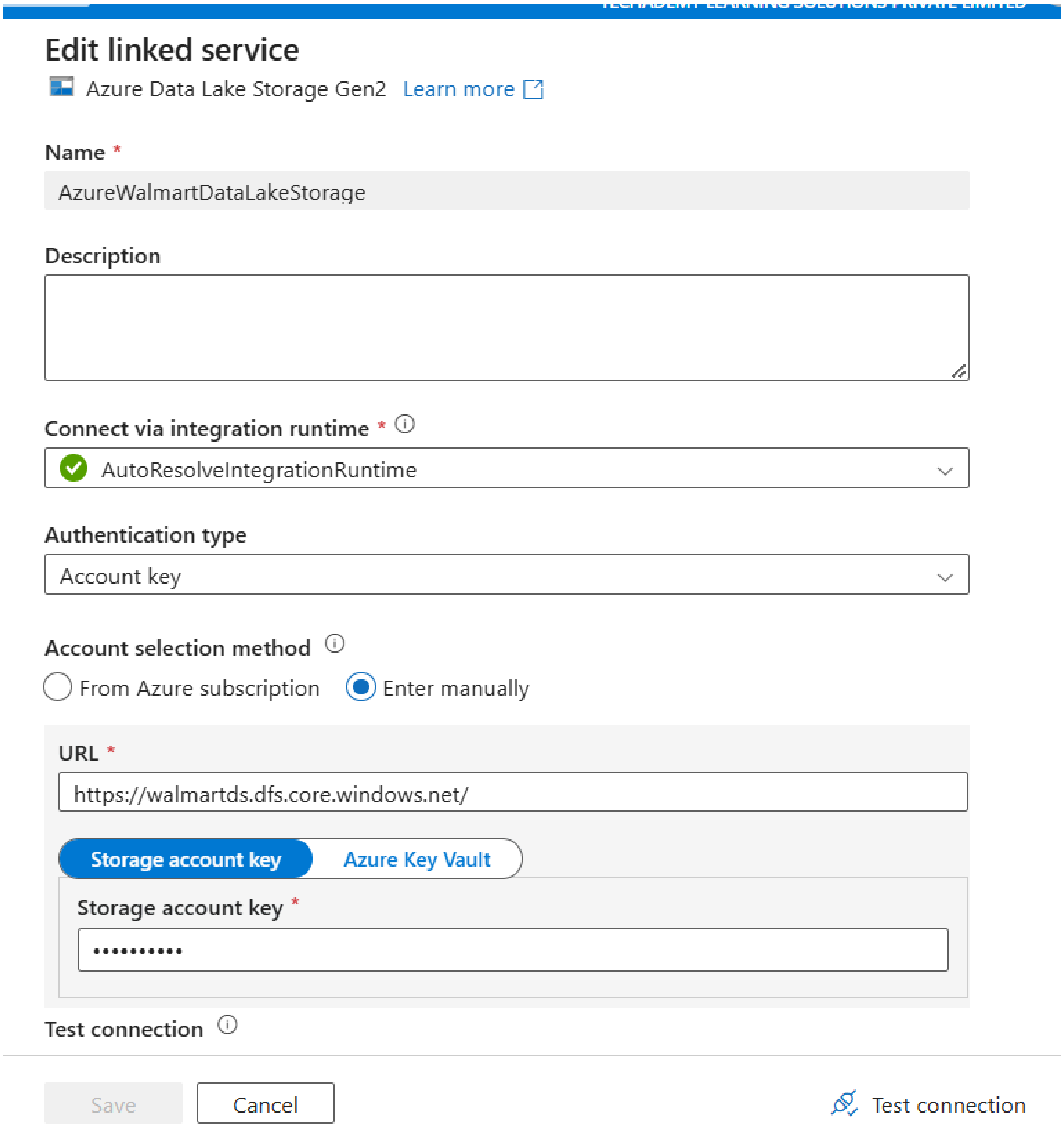
Database name *

walmart-db

Authentication type *

SQL authentication

- **ADLS Gen 2** (destination).



TECHADEMY LEARNING SOLUTIONS PRIVATE LIMITED

Edit linked service

Azure Data Lake Storage Gen2 Learn more

Name *

AzureWalmartDataLakeStorage

Description

Connect via integration runtime *

✓ AutoResolveIntegrationRuntime

Authentication type

Account key

Account selection method

○ From Azure subscription ● Enter manually

URL *

https://walmartds.dfs.core.windows.net/

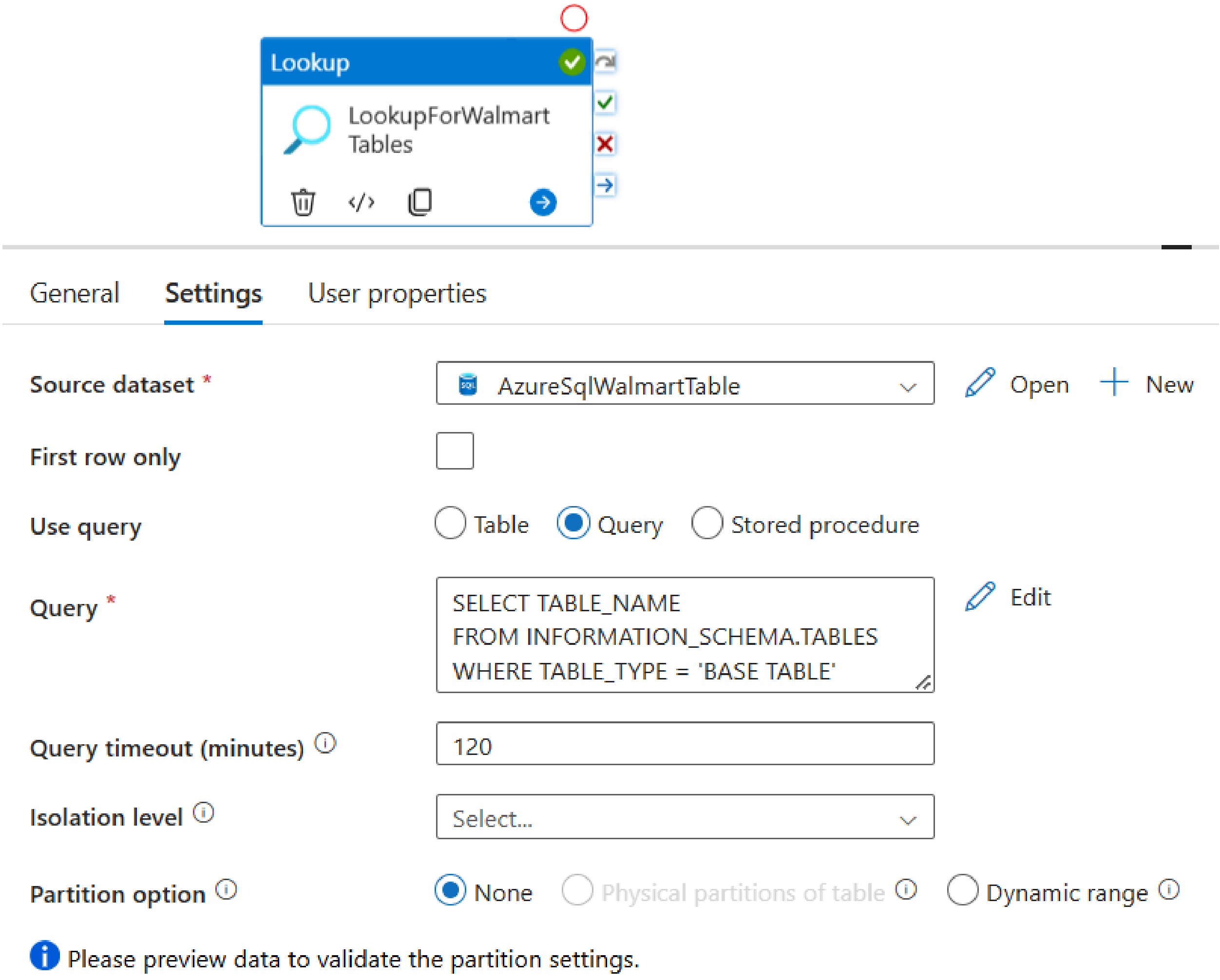
Storage account key | Azure Key Vault

Storage account key *

••••••••••

Test connection

2. **Dynamic Table Listing**
   - Used the **Lookup activity** in ADF to query and retrieve the list of table names from the Azure SQL Database dynamically.



Source dataset * AzureSqlWalmartTable

First row only □

Use query ○ Table ● Query ○ Stored procedure

Query *

```
SELECT TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'BASE TABLE'
```

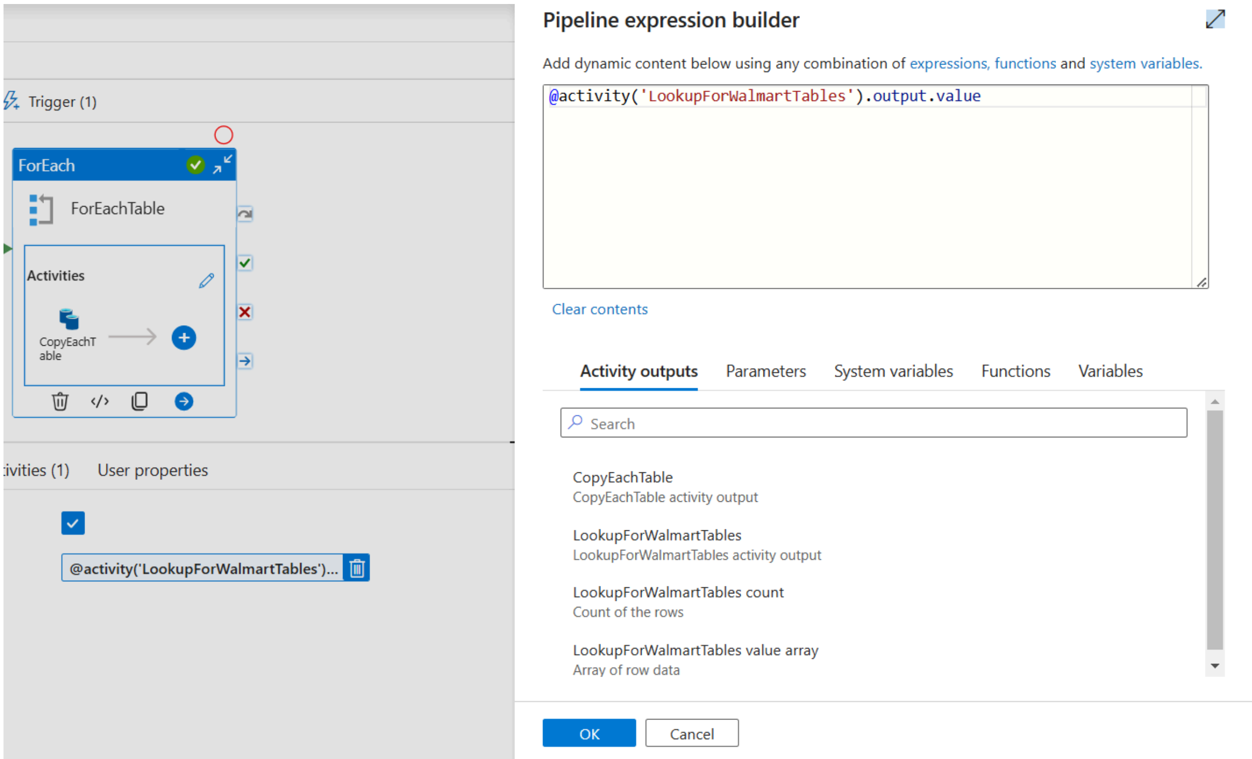
Query timeout (minutes) 120

Isolation level Select...

Partition option ● None ○ Physical partitions of table ○ Dynamic range

Please preview data to validate the partition settings.

3. **ForEach Activity**
    - Implemented a **ForEach loop** to iterate over the list of table names obtained from the Lookup activity.



Pipeline expression builder

Add dynamic content below using any combination of expressions, functions and system variables.

```
@activity('LookupForWalmartTables').output.value
```

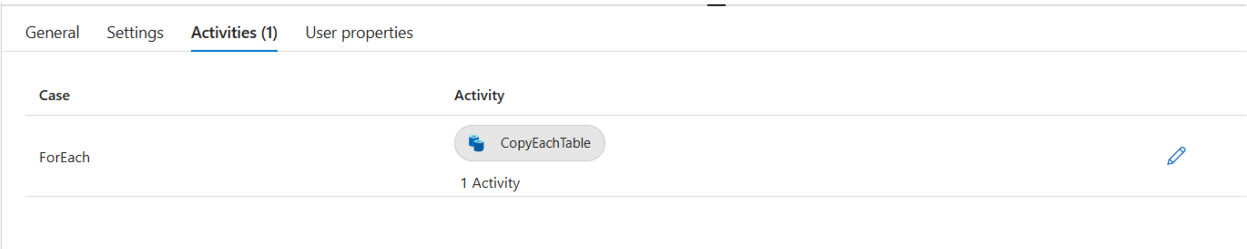
Clear contents

CopyEachTable
CopyEachTable activity output

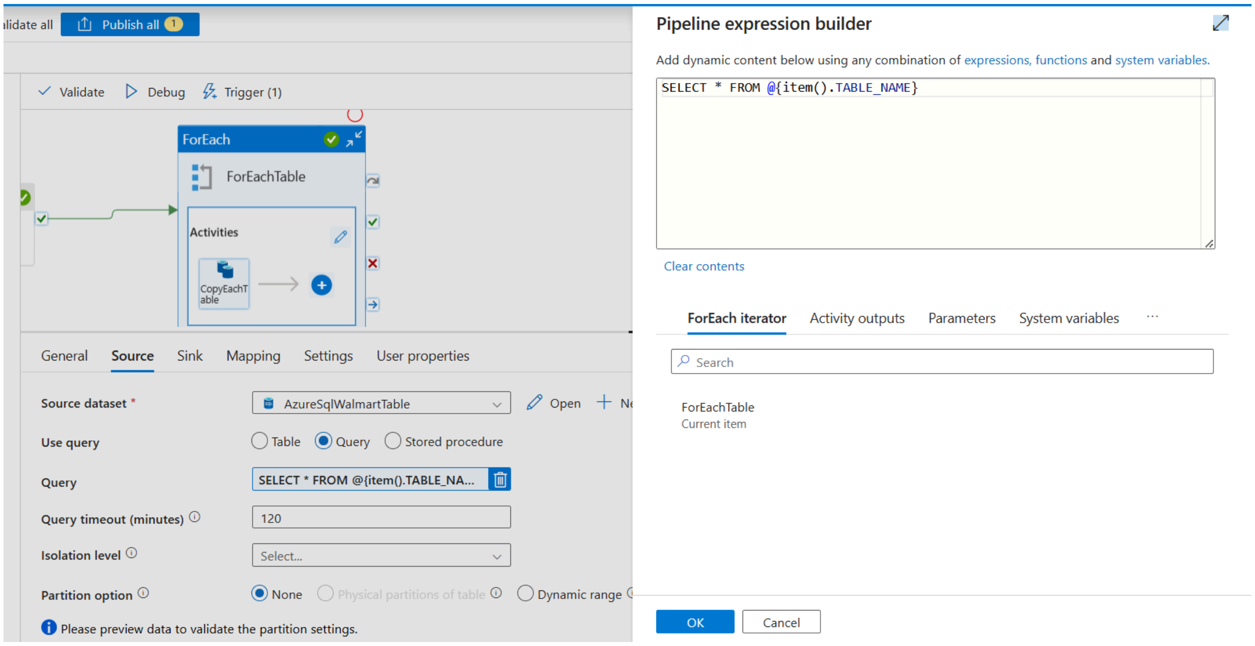
LookupForWalmartTables
LookupForWalmartTables activity output

LookupForWalmartTables count
Count of the rows

LookupForWalmartTables value array
Array of row data



Case	Activity
ForEach	CopyEachTable 1 Activity



Source dataset * AzureSqlWalmartTable

Use query ○ Table ● Query ○ Stored procedure

Query SELECT * FROM @{item().TABLE_NA...

Query timeout (minutes) 120

Isolation level Select...

Partition option ● None ○ Physical partitions of table ○ Dynamic range

Please preview data to validate the partition settings.

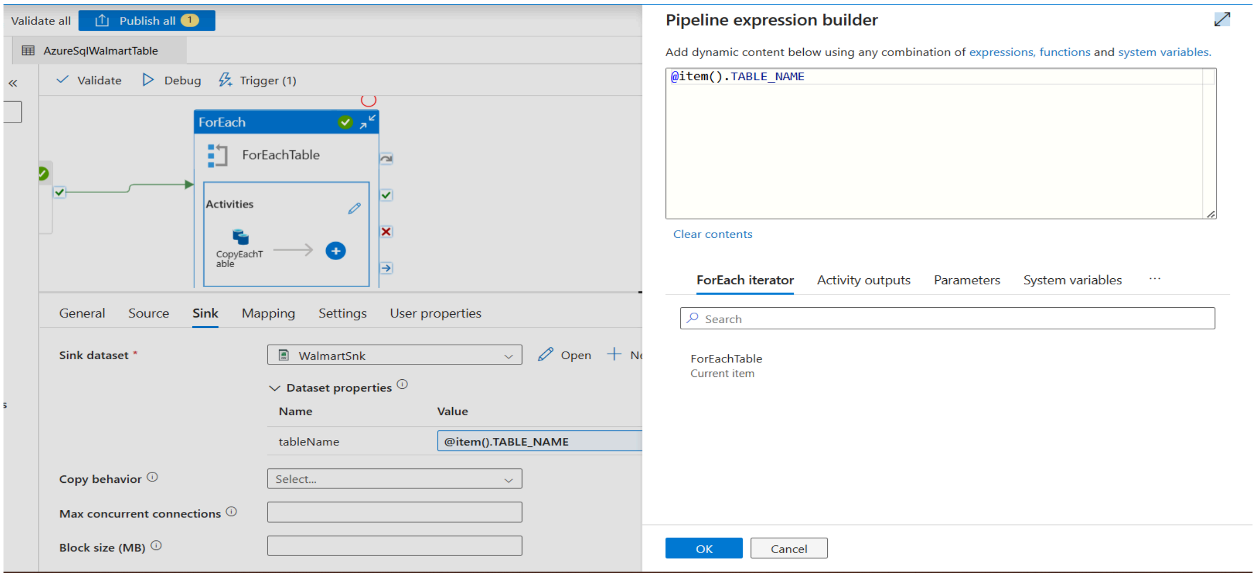
Pipeline expression builder

Add dynamic content below using any combination of expressions, functions and system variables.

```
SELECT * FROM @{item().TABLE_NAME}
```

ForEachTable
Current item

4. **Dynamic Folder Structure and File Naming**
   - Inside the ForEach loop, used a **Copy Data activity** to:
     - Dynamically create folder structures in ADLS Gen 2 for each table.
     - Generate CSV files with file names corresponding to the table names being processed.
   - Dynamic parameters were configured for folder paths and file names using expressions to fetch the table name at runtime.



Sink dataset * WalmartSnk

Dataset properties

Name	Value
tableName	@item().TABLE_NAME

Copy behavior Select...

Max concurrent connections

Block size (MB)

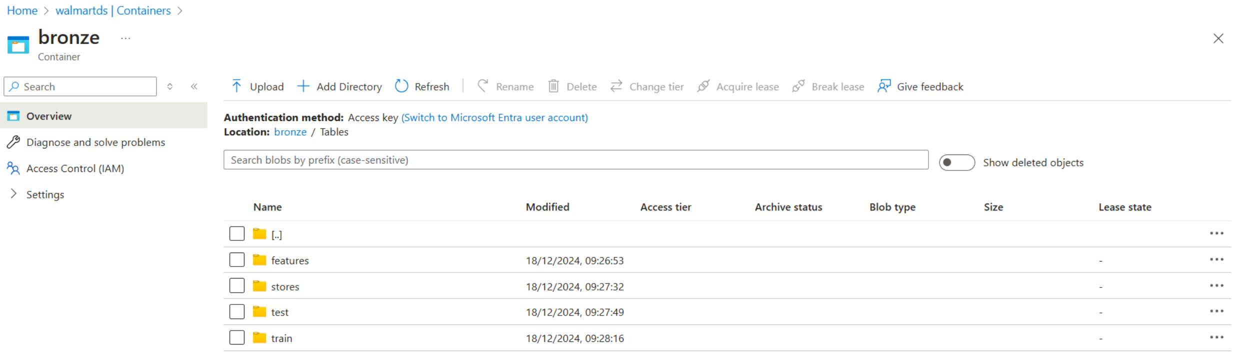
Pipeline expression builder

Add dynamic content below using any combination of expressions, functions and system variables.

```
@item().TABLE_NAME
```

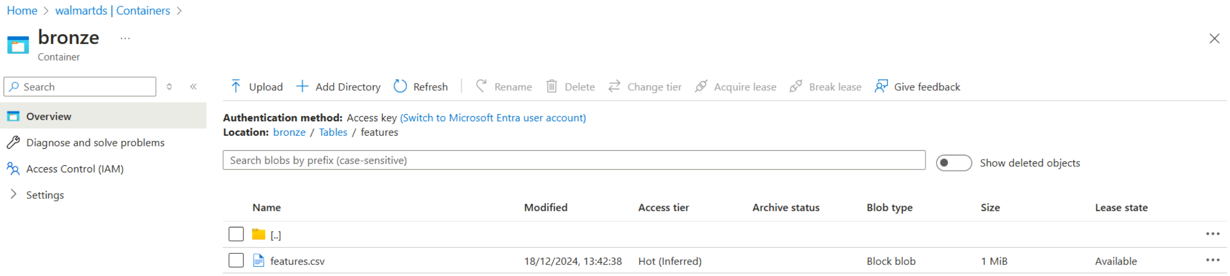
ForEachTable
Current item

5. **Copy Data Activity Configuration**
   - Configured the Copy Data activity to:
     - Extract data from each table in the Azure SQL Database.
     - Store it as **CSV files** in the dynamically created folders within ADLS Gen 2.



Authentication method: Access key (Switch to Microsoft Entra user account)
Location: bronze / Tables

	Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
□	[..]						
□	features	18/12/2024, 09:26:53					-
□	stores	18/12/2024, 09:27:32					-
□	test	18/12/2024, 09:27:49					-
□	train	18/12/2024, 09:28:16					-



Authentication method: Access key (Switch to Microsoft Entra user account)
Location: bronze / Tables / features

	Name	Modified	Access tier	Archive status	Blob type	Size	Lease state
□	[..]						
□	features.csv	18/12/2024, 13:42:38	Hot (Inferred)		Block blob	1 MiB	Available

Key Features of the Pipeline

- **Dynamic Folder Structure**: Automatically creates separate folders for each table in ADLS Gen 2.
- **Dynamic File Naming**: Ensures each CSV file is named based on the corresponding table name, simplifying data identification.
- **Reusability**: Designed to adapt to changes in the database schema or addition of new tables without manual reconfiguration.

This approach streamlines the ingestion process, ensuring scalability and efficient organization of the data for downstream analytics and processing tasks.

Data Transformation

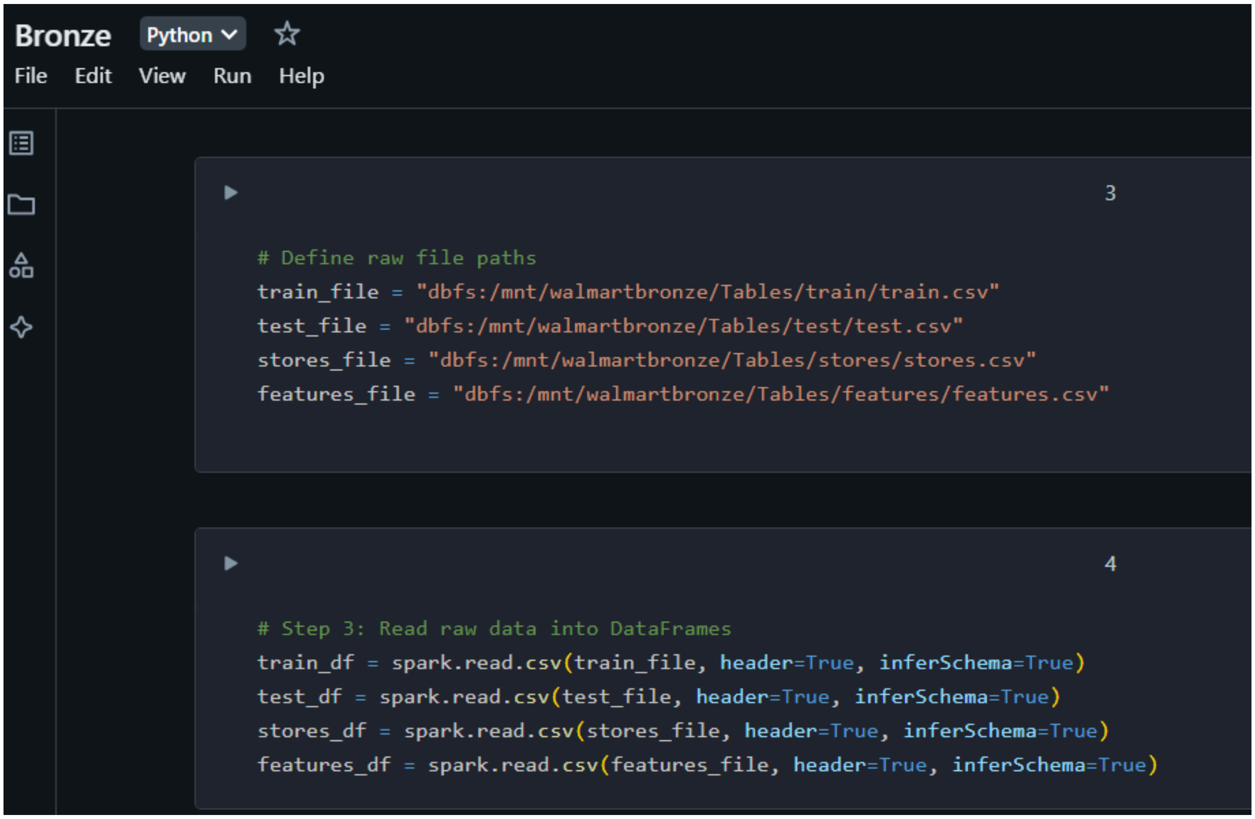
To enable data transformation and analytics, I utilized Azure Databricks (ADB) for the ETL pipeline. The process was divided into three distinct layers: **Bronze**, **Silver**, and **Gold**, each designed to manage specific stages of data transformation.

1. **Mounting Containers**:
   I began by creating a notebook named `Mount`, where I mounted three ADLS Gen2 containers: **Bronze**, **Silver**, and **Gold**. This setup ensured seamless access to data stored in Azure Data Lake.
2. **Bronze Layer**:
   - In the `Bronze` notebook, I loaded raw data from the Bronze container into PySpark DataFrames.
   - Performed data cleaning and validation steps such as removing duplicates, handling missing values, standardizing formats, and addressing outliers.
   - After cleaning, I stored the transformed data into the **Silver container** in Delta format for further processing.
3. **Silver Layer**:
   - In the `Silver` notebook, I focused on refining and enriching the data.
   - This involved joining multiple DataFrames (e.g., `Train`, `Test`, `Stores`, and `Features`) based on shared keys such as `Store`, `Date`, and `IsHoliday`.
   - The enriched data was saved back to the **Silver container** in a structured Delta format.
4. **Gold Layer**:
   - In the `Gold` notebook, I performed business-level calculations and prepared the data for analytics and reporting.
   - This included aggregations, feature engineering, and preparing datasets for predictive modeling using PySpark MLlib.
   - The final outputs were stored in the **Gold container** in Delta format for downstream consumption.

Notebook Overview

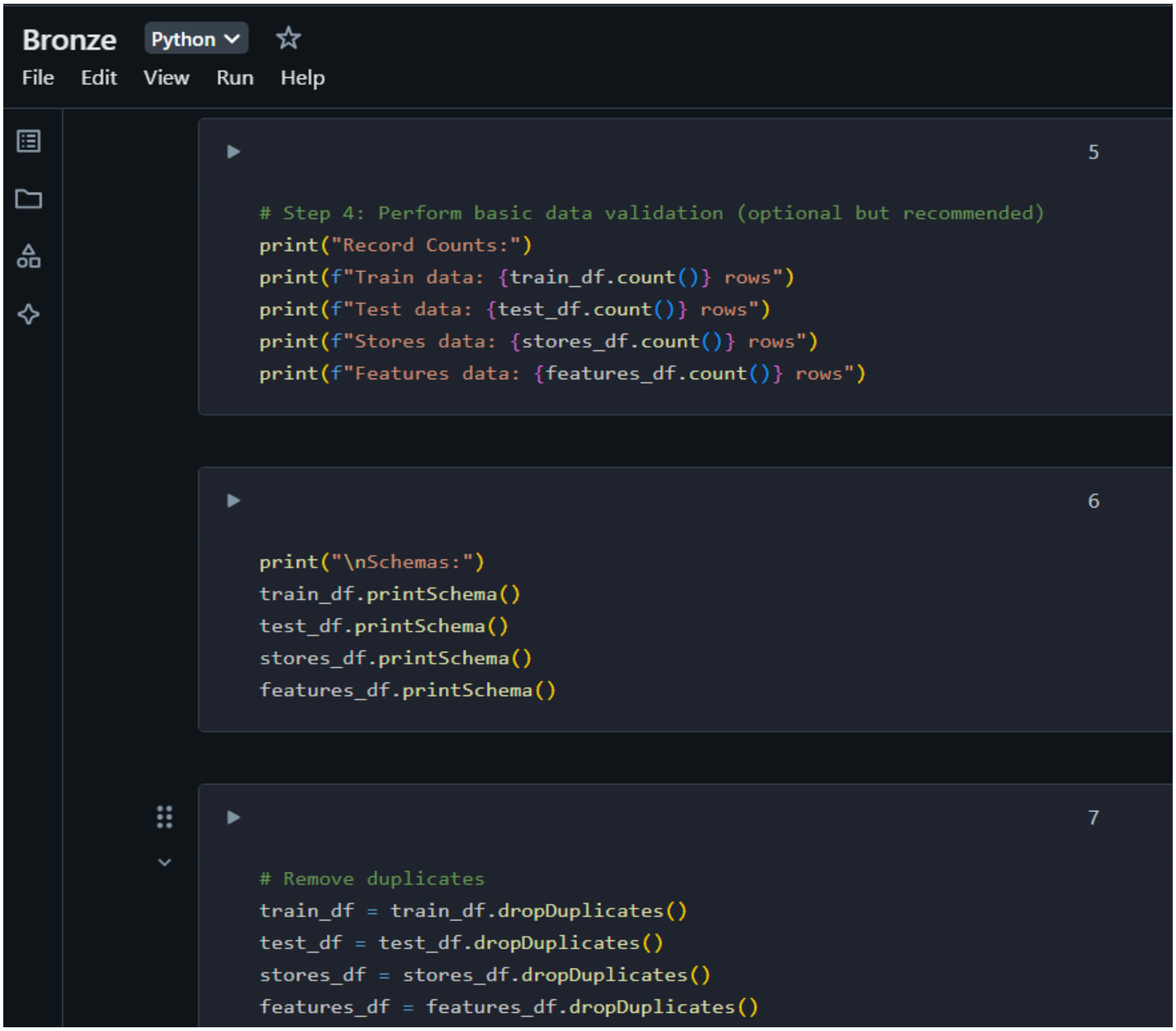
Bronze Notebook:

- Loaded raw data into PySpark DataFrames from ADLS Gen2.
- Applied cleaning steps such as duplicate removal, handling missing values, and addressing outliers.
- Transformed data was written to the **Silver container** in Delta format.



```
# Define raw file paths
train_file = "dbfs:/mnt/walmartbronze/Tables/train/train.csv"
test_file = "dbfs:/mnt/walmartbronze/Tables/test/test.csv"
stores_file = "dbfs:/mnt/walmartbronze/Tables/stores/stores.csv"
features_file = "dbfs:/mnt/walmartbronze/Tables/features/features.csv"
```

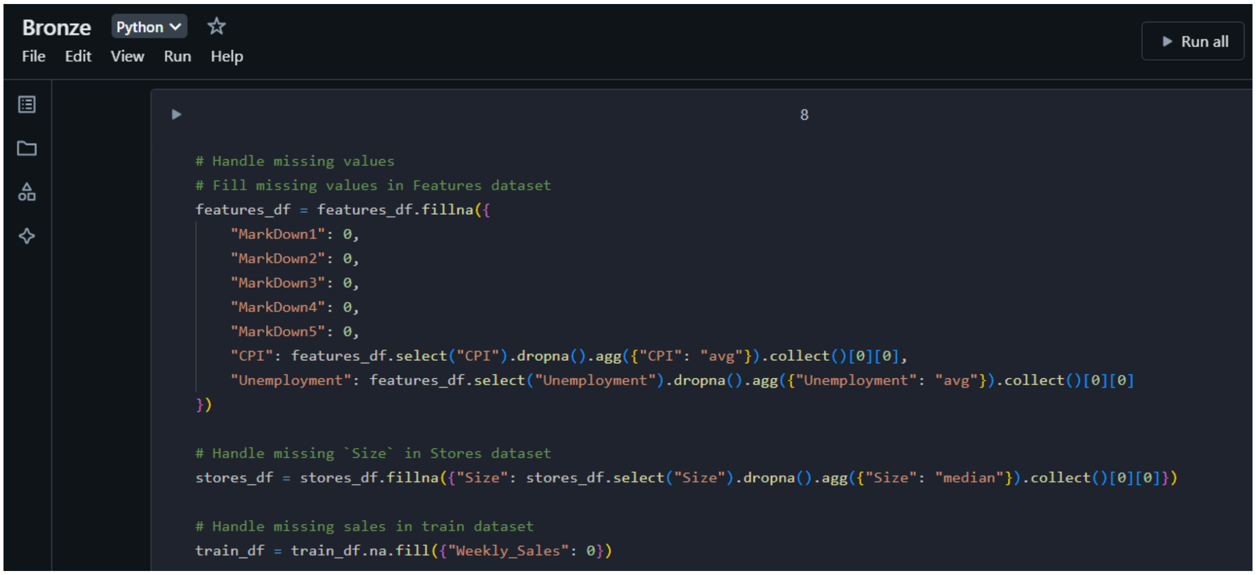
```
# Step 3: Read raw data into DataFrames
train_df = spark.read.csv(train_file, header=True, inferSchema=True)
test_df = spark.read.csv(test_file, header=True, inferSchema=True)
stores_df = spark.read.csv(stores_file, header=True, inferSchema=True)
features_df = spark.read.csv(features_file, header=True, inferSchema=True)
```



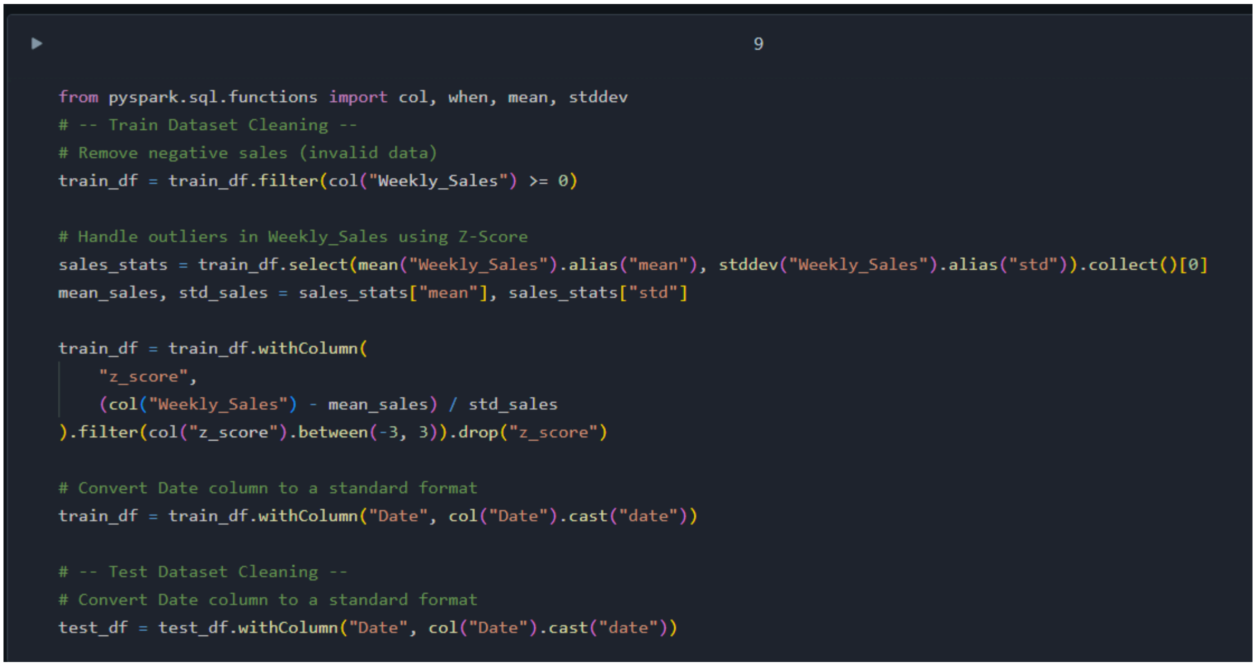
```python
# Step 4: Perform basic data validation (optional but recommended)
print("Record Counts:")
print(f"Train data: {train_df.count()} rows")
print(f"Test data: {test_df.count()} rows")
print(f"Stores data: {stores_df.count()} rows")
print(f"Features data: {features_df.count()} rows")
```

```python
print("\nSchemas:")
train_df.printSchema()
test_df.printSchema()
stores_df.printSchema()
features_df.printSchema()
```

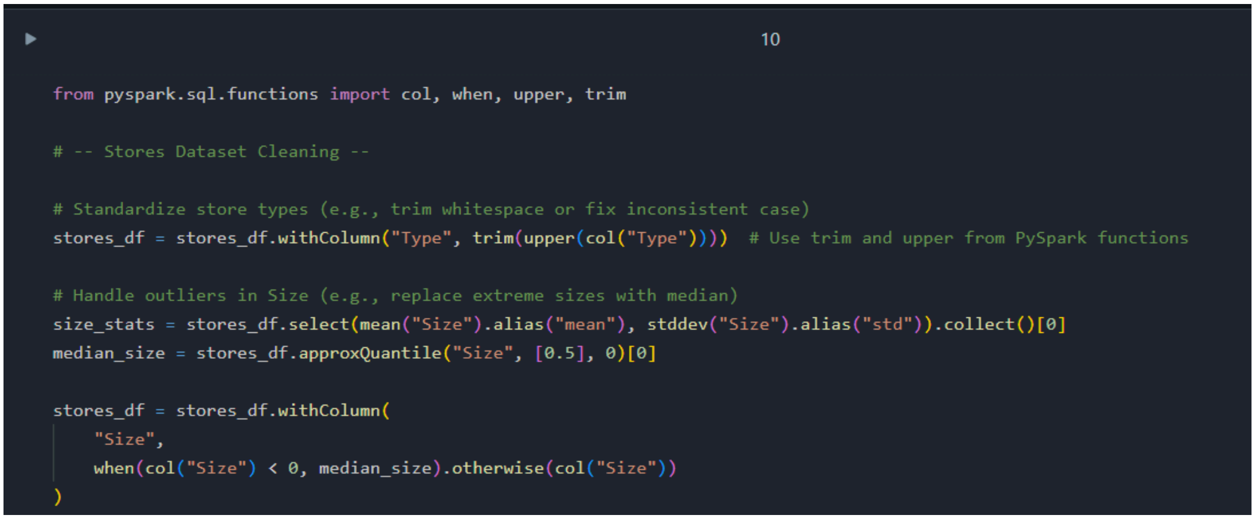
```python
# Remove duplicates
train_df = train_df.dropDuplicates()
test_df = test_df.dropDuplicates()
stores_df = stores_df.dropDuplicates()
features_df = features_df.dropDuplicates()
```



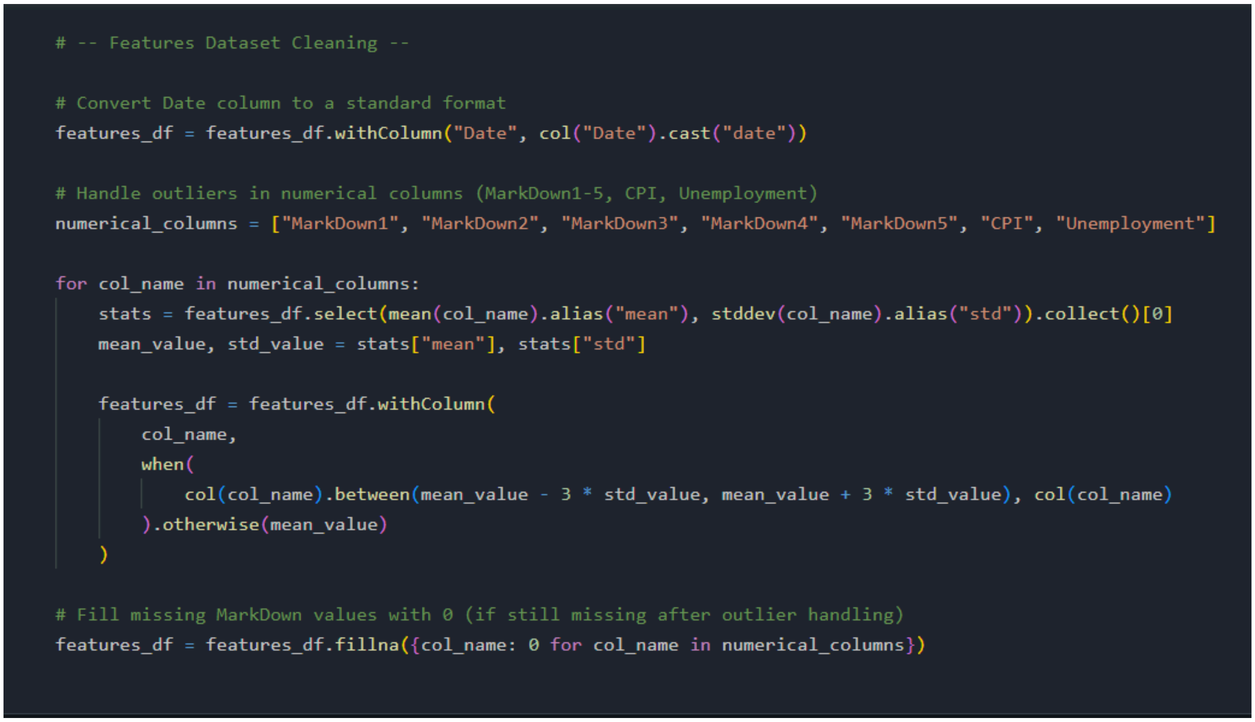
```python
# Handle missing values
# Fill missing values in Features dataset
features_df = features_df.fillna({
    "MarkDown1": 0,
    "MarkDown2": 0,
    "MarkDown3": 0,
    "MarkDown4": 0,
    "MarkDown5": 0,
    "CPI": features_df.select("CPI").dropna().agg({"CPI": "avg"}).collect()[0][0],
    "Unemployment": features_df.select("Unemployment").dropna().agg({"Unemployment": "avg"}).collect()[0][0]
})

# Handle missing `Size` in Stores dataset
stores_df = stores_df.fillna({"Size": stores_df.select("Size").dropna().agg({"Size": "median"}).collect()[0][0]})

# Handle missing sales in train dataset
train_df = train_df.na.fill({"Weekly_Sales": 0})
```



```python
from pyspark.sql.functions import col, when, mean, stddev
# -- Train Dataset Cleaning --
# Remove negative sales (invalid data)
train_df = train_df.filter(col("Weekly_Sales") >= 0)

# Handle outliers in Weekly_Sales using Z-Score
sales_stats = train_df.select(mean("Weekly_Sales").alias("mean"), stddev("Weekly_Sales").alias("std")).collect()[0]
mean_sales, std_sales = sales_stats["mean"], sales_stats["std"]

train_df = train_df.withColumn(
    "z_score",
    (col("Weekly_Sales") - mean_sales) / std_sales
).filter(col("z_score").between(-3, 3)).drop("z_score")

# Convert Date column to a standard format
train_df = train_df.withColumn("Date", col("Date").cast("date"))

# -- Test Dataset Cleaning --
# Convert Date column to a standard format
test_df = test_df.withColumn("Date", col("Date").cast("date"))
```

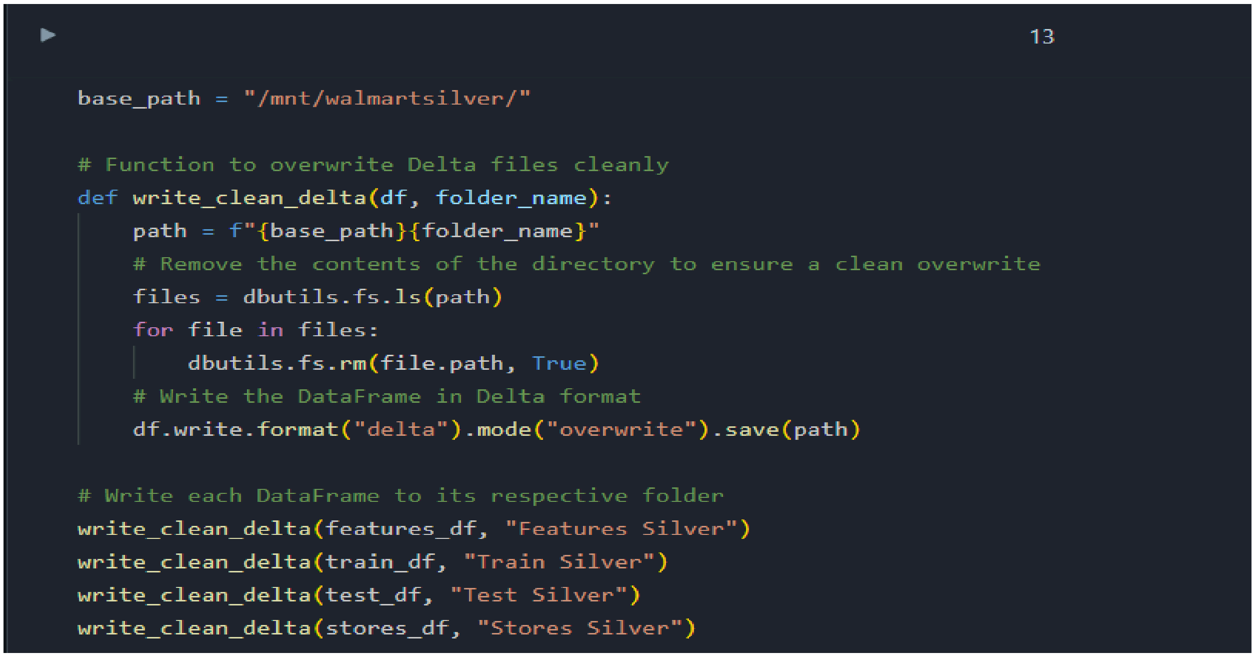


```python
from pyspark.sql.functions import col, when, upper, trim

# -- Stores Dataset Cleaning --

# Standardize store types (e.g., trim whitespace or fix inconsistent case)
stores_df = stores_df.withColumn("Type", trim(upper(col("Type"))))  # Use trim and upper from PySpark functions

# Handle outliers in Size (e.g., replace extreme sizes with median)
size_stats = stores_df.select(mean("Size").alias("mean"), stddev("Size").alias("std")).collect()[0]
median_size = stores_df.approxQuantile("Size", [0.5], 0)[0]

stores_df = stores_df.withColumn(
    "Size",
    when(col("Size") < 0, median_size).otherwise(col("Size"))
)
```



```python
# -- Features Dataset Cleaning --

# Convert Date column to a standard format
features_df = features_df.withColumn("Date", col("Date").cast("date"))

# Handle outliers in numerical columns (MarkDown1-5, CPI, Unemployment)
numerical_columns = ["MarkDown1", "MarkDown2", "MarkDown3", "MarkDown4", "MarkDown5", "CPI", "Unemployment"]

for col_name in numerical_columns:
    stats = features_df.select(mean(col_name).alias("mean"), stddev(col_name).alias("std")).collect()[0]
    mean_value, std_value = stats["mean"], stats["std"]

    features_df = features_df.withColumn(
        col_name,
        when(
            col(col_name).between(mean_value - 3 * std_value, mean_value + 3 * std_value), col(col_name)
        ).otherwise(mean_value)
    )

# Fill missing MarkDown values with 0 (if still missing after outlier handling)
features_df = features_df.fillna({col_name: 0 for col_name in numerical_columns})
```

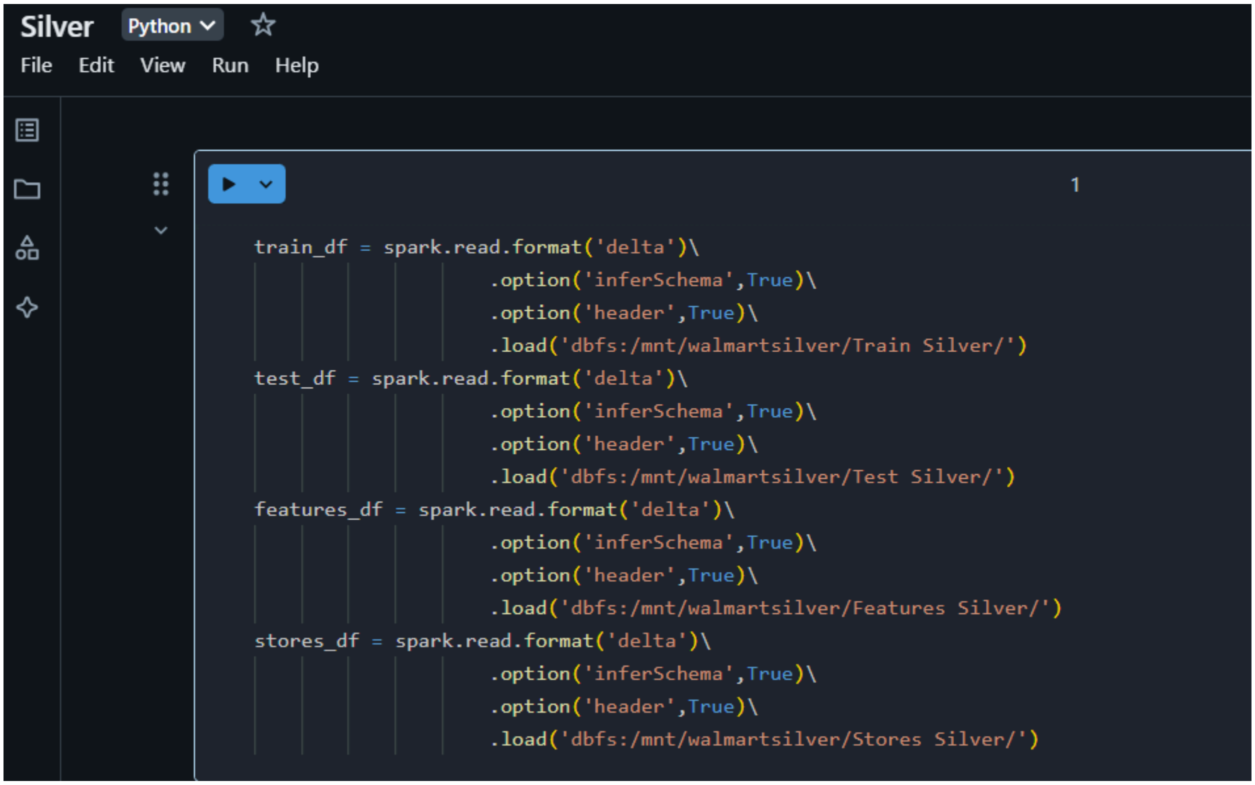
Writing All DF to Silver Container



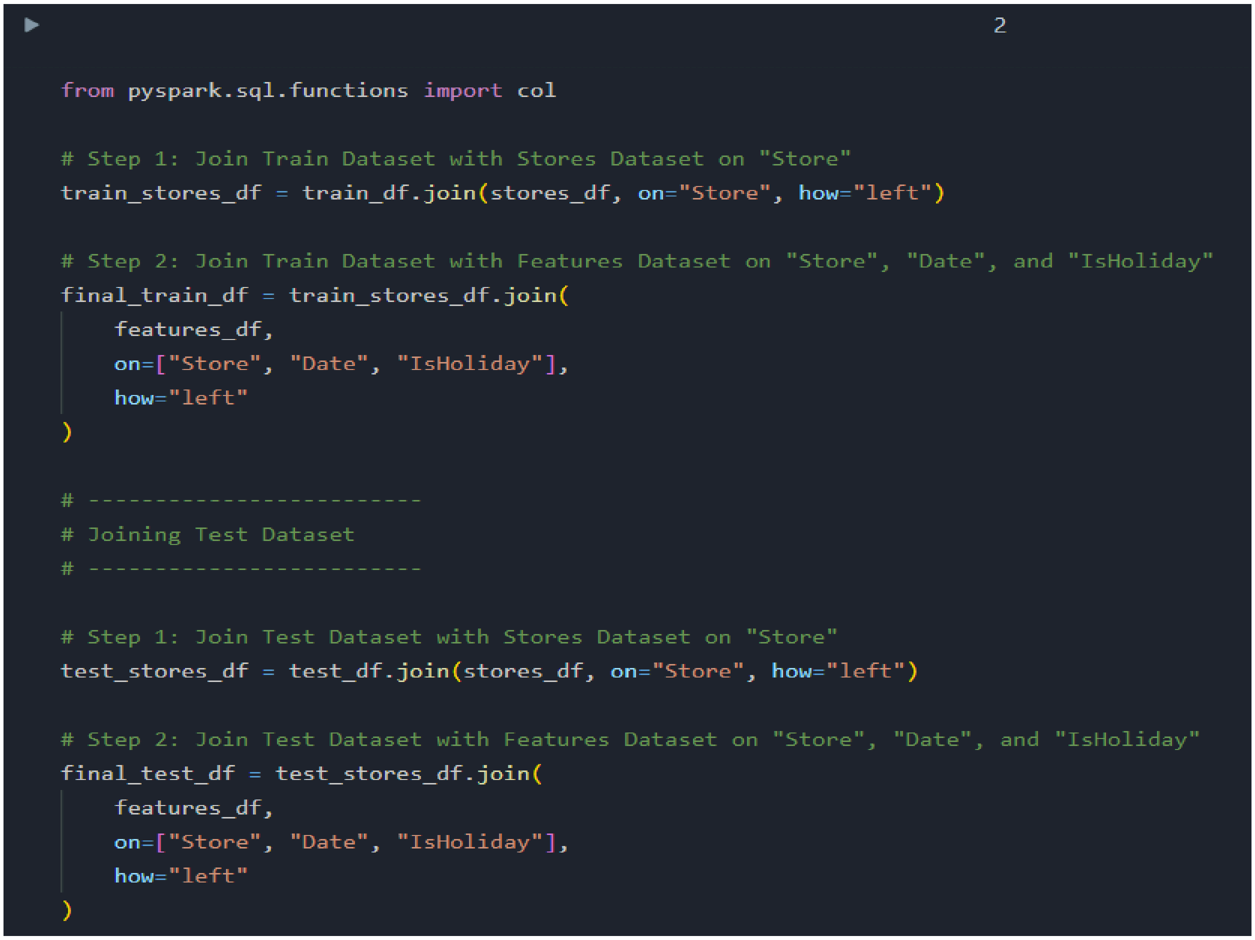
```python
base_path = "/mnt/walmartsilver/"

# Function to overwrite Delta files cleanly
def write_clean_delta(df, folder_name):
    path = f"{base_path}{folder_name}"
    # Remove the contents of the directory to ensure a clean overwrite
    files = dbutils.fs.ls(path)
    for file in files:
        dbutils.fs.rm(file.path, True)
    # Write the DataFrame in Delta format
    df.write.format("delta").mode("overwrite").save(path)

# Write each DataFrame to its respective folder
write_clean_delta(features_df, "Features Silver")
write_clean_delta(train_df, "Train Silver")
write_clean_delta(test_df, "Test Silver")
write_clean_delta(stores_df, "Stores Silver")
```

Silver Notebook:

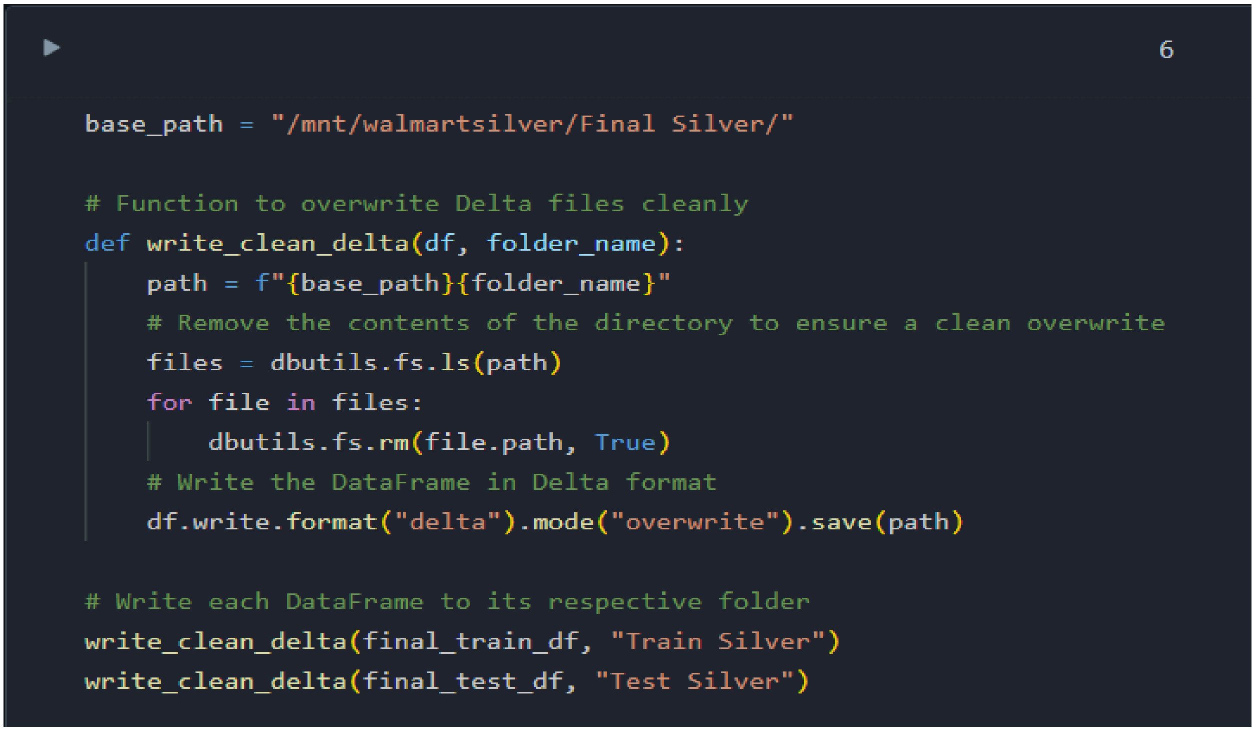
- Read data from the **Silver container**.
- Enriched the data by performing joins and standardizations.
- Saved the enriched data back to the **Silver container** in Delta format.



```
train_df = spark.read.format('delta')\
                    .option('inferSchema',True)\
                    .option('header',True)\
                    .load('dbfs:/mnt/walmartsilver/Train Silver/')
test_df = spark.read.format('delta')\
                    .option('inferSchema',True)\
                    .option('header',True)\
                    .load('dbfs:/mnt/walmartsilver/Test Silver/')
features_df = spark.read.format('delta')\
                    .option('inferSchema',True)\
                    .option('header',True)\
                    .load('dbfs:/mnt/walmartsilver/Features Silver/')
stores_df = spark.read.format('delta')\
                    .option('inferSchema',True)\
                    .option('header',True)\
                    .load('dbfs:/mnt/walmartsilver/Stores Silver/')
```



```python
from pyspark.sql.functions import col

# Step 1: Join Train Dataset with Stores Dataset on "Store"
train_stores_df = train_df.join(stores_df, on="Store", how="left")

# Step 2: Join Train Dataset with Features Dataset on "Store", "Date", and "IsHoliday"
final_train_df = train_stores_df.join(
    features_df,
    on=["Store", "Date", "IsHoliday"],
    how="left"
)

# -------------------------
# Joining Test Dataset
# -------------------------

# Step 1: Join Test Dataset with Stores Dataset on "Store"
test_stores_df = test_df.join(stores_df, on="Store", how="left")

# Step 2: Join Test Dataset with Features Dataset on "Store", "Date", and "IsHoliday"
final_test_df = test_stores_df.join(
    features_df,
    on=["Store", "Date", "IsHoliday"],
    how="left"
)
```



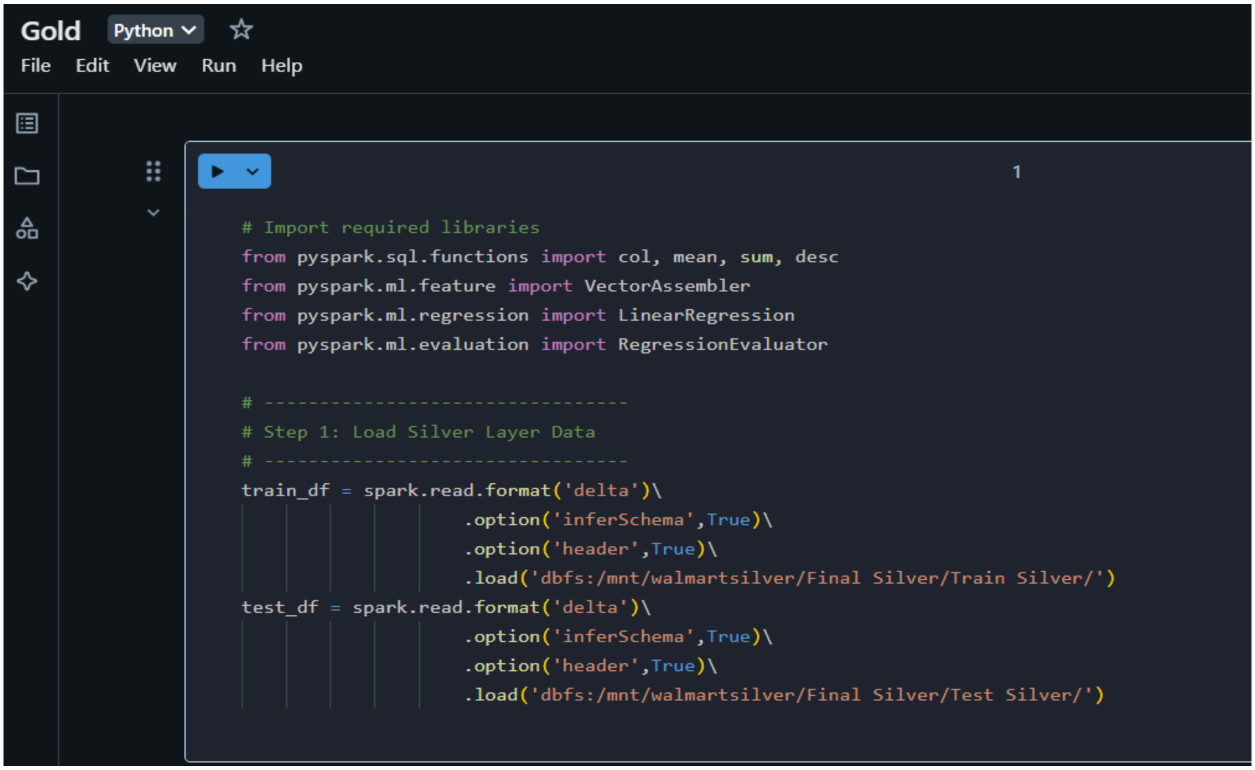
```python
base_path = "/mnt/walmartsilver/Final Silver/"

# Function to overwrite Delta files cleanly
def write_clean_delta(df, folder_name):
    path = f"{base_path}{folder_name}"
    # Remove the contents of the directory to ensure a clean overwrite
    files = dbutils.fs.ls(path)
    for file in files:
        dbutils.fs.rm(file.path, True)
    # Write the DataFrame in Delta format
    df.write.format("delta").mode("overwrite").save(path)

# Write each DataFrame to its respective folder
write_clean_delta(final_train_df, "Train Silver")
write_clean_delta(final_test_df, "Test Silver")
```

Gold Notebook:

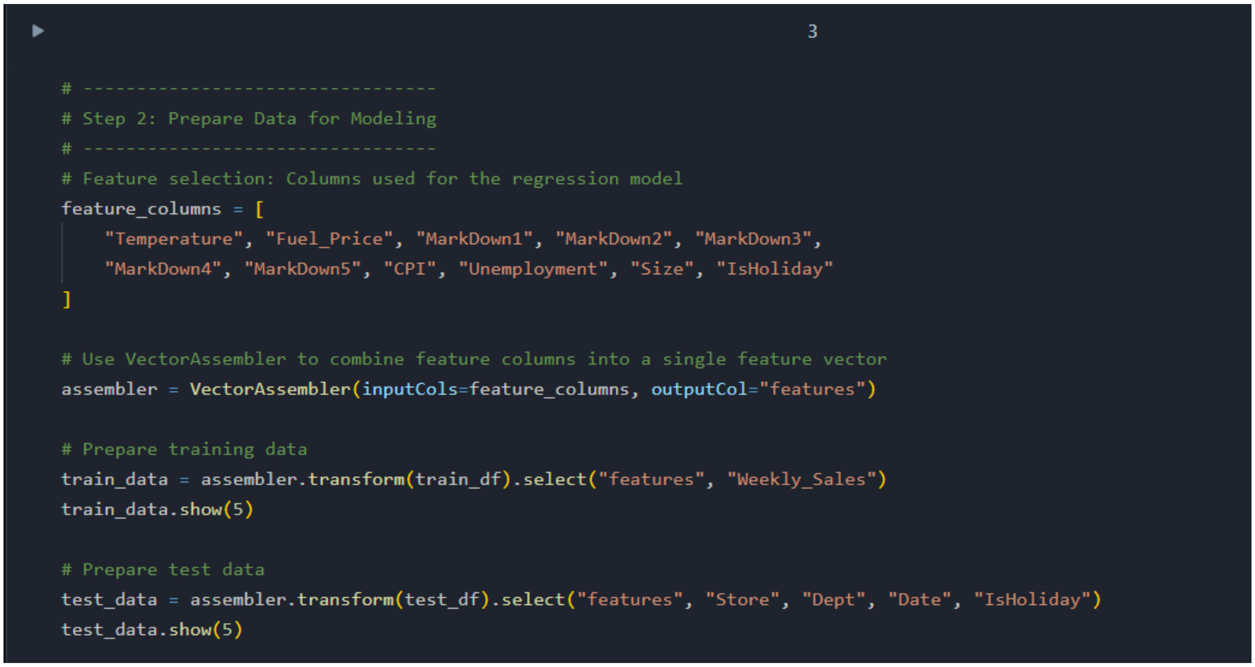
In the **Gold Layer** notebook, we performed business-level calculations and model training to derive valuable insights from the data. Here's a breakdown of the steps involved:

1. **Loading Silver Layer Data:**
   - We loaded the cleaned and transformed data from the Silver layer, which had undergone multiple transformations and joins in the previous notebooks.
2. **Preparing Data for Modeling:**
   - We selected the relevant features for the regression model, which included columns like `Temperature`, `Fuel_Price`, `MarkDown1`, `MarkDown2`, `CPI`, and others.
   - We used **VectorAssembler** to combine these features into a single feature vector (`features`) that could be used in the regression model.
3. **Training the Regression Model:**
   - We used **Linear Regression** to predict the `Weekly_Sales` based on the selected features. This involved:
     - Initializing the regression model with `features` as the input and `Weekly_Sales` as the label.
     - Fitting the model to the training data and evaluating the model's performance.
4. **Model Evaluation:**
   - After training the model, we evaluated its performance using metrics such as **Root Mean Squared Error (RMSE)** and **R²** (coefficient of determination), which indicate the model's accuracy and how well it explains the variance in the data.

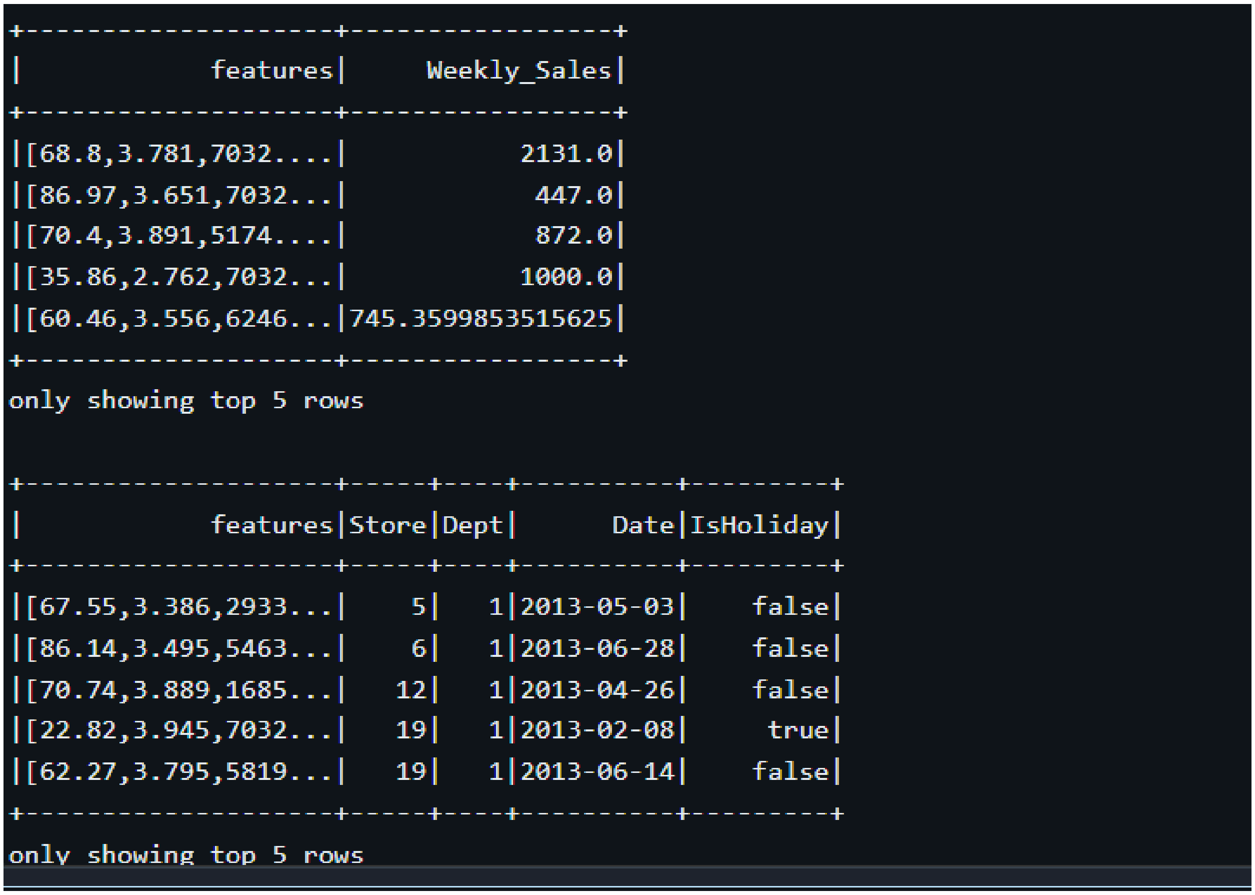
In the Gold notebook, the goal was to perform high-level business analytics by training predictive models (such as Linear Regression) to forecast `Weekly_Sales` and evaluate the performance of the model to ensure accurate and reliable predictions.



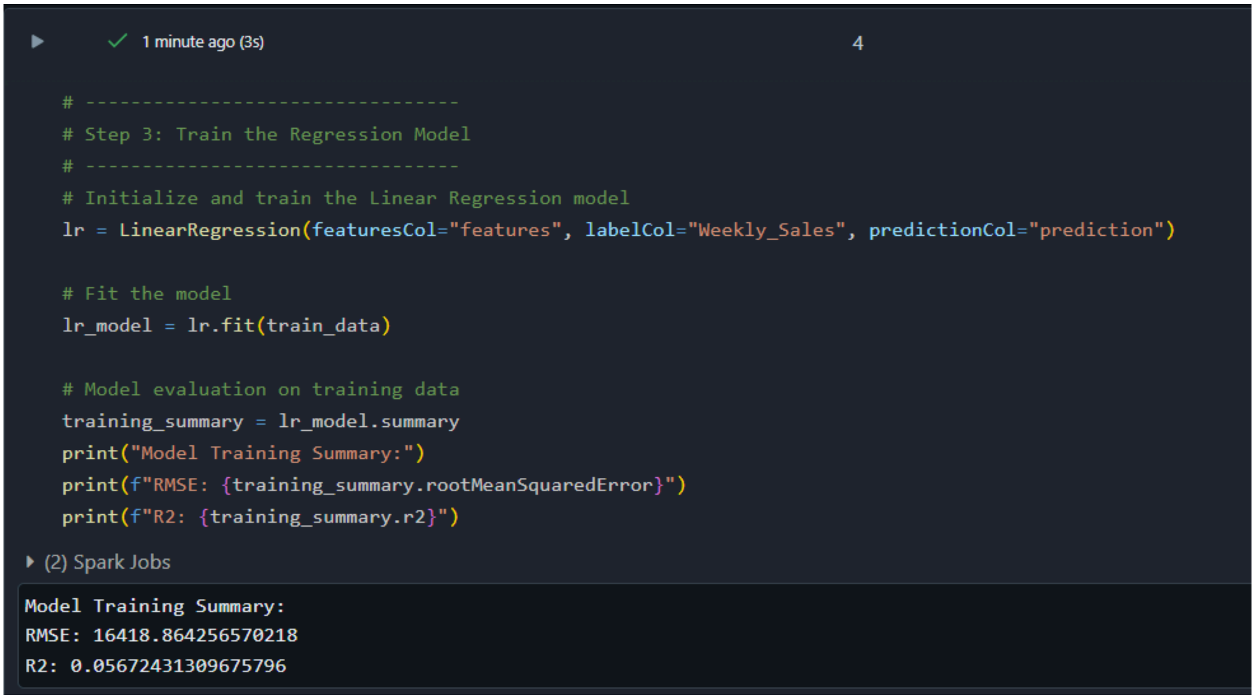
```
# Import required libraries
from pyspark.sql.functions import col, mean, sum, desc
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator

# --------------------------------
# Step 1: Load Silver Layer Data
# --------------------------------
train_df = spark.read.format('delta')\
                    .option('inferSchema',True)\
                    .option('header',True)\
                    .load('dbfs:/mnt/walmartsilver/Final Silver/Train Silver/')
test_df = spark.read.format('delta')\
                    .option('inferSchema',True)\
                    .option('header',True)\
                    .load('dbfs:/mnt/walmartsilver/Final Silver/Test Silver/')
```



```
# --------------------------------
# Step 2: Prepare Data for Modeling
# --------------------------------
# Feature selection: Columns used for the regression model
feature_columns = [
    "Temperature", "Fuel_Price", "MarkDown1", "MarkDown2", "MarkDown3",
    "MarkDown4", "MarkDown5", "CPI", "Unemployment", "Size", "IsHoliday"
]

# Use VectorAssembler to combine feature columns into a single feature vector
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")

# Prepare training data
train_data = assembler.transform(train_df).select("features", "Weekly_Sales")
train_data.show(5)

# Prepare test data
test_data = assembler.transform(test_df).select("features", "Store", "Dept", "Date", "IsHoliday")
test_data.show(5)
```

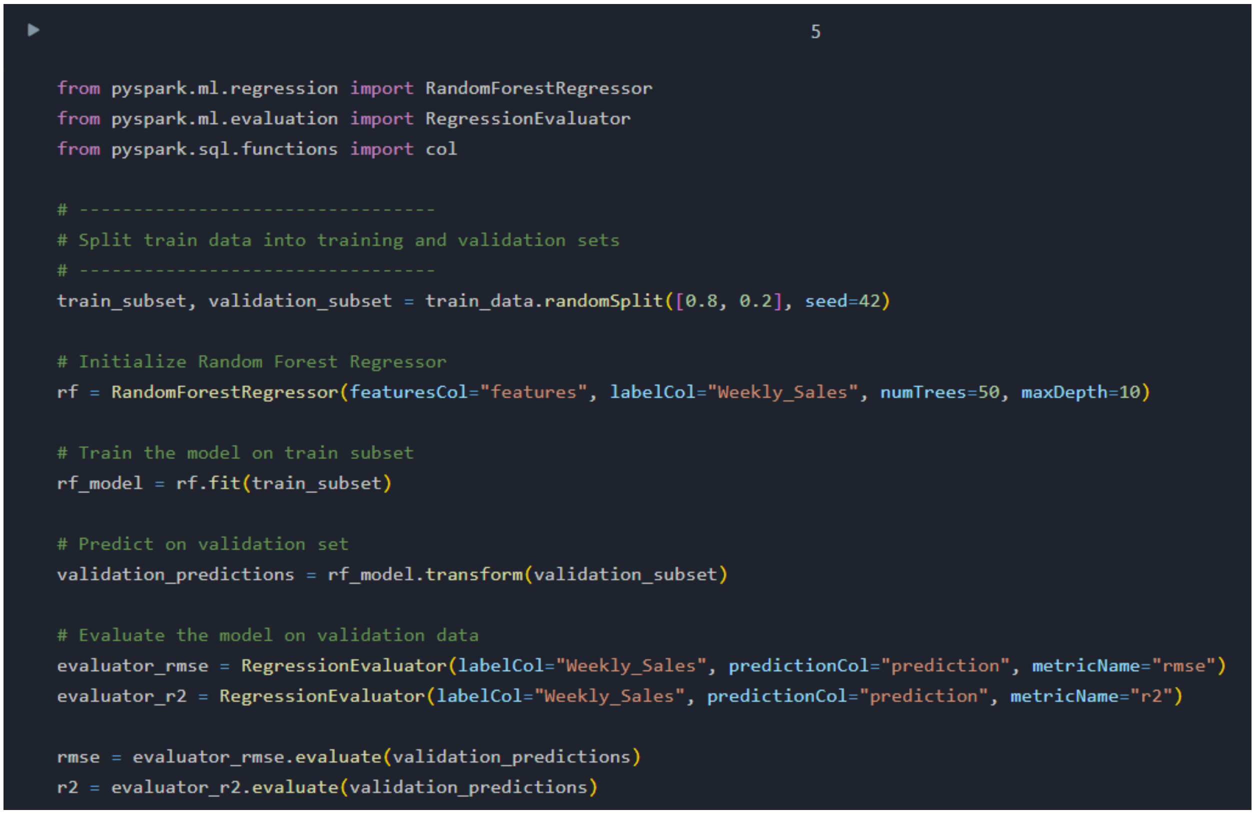


```
+--------------------+-----------------+
|            features|     Weekly_Sales|
+--------------------+-----------------+
|[68.8,3.781,7032....|           2131.0|
|[86.97,3.651,7032...|            447.0|
|[70.4,3.891,5174....|            872.0|
|[35.86,2.762,7032...|           1000.0|
|[60.46,3.556,6246...|745.3599853515625|
+--------------------+-----------------+
only showing top 5 rows

+--------------------+-----+----+----------+---------+
|            features|Store|Dept|      Date|IsHoliday|
+--------------------+-----+----+----------+---------+
|[67.55,3.386,2933...|    5|   1|2013-05-03|    false|
|[86.14,3.495,5463...|    6|   1|2013-06-28|    false|
|[70.74,3.889,1685...|   12|   1|2013-04-26|    false|
|[22.82,3.945,7032...|   19|   1|2013-02-08|     true|
|[62.27,3.795,5819...|   19|   1|2013-06-14|    false|
+--------------------+-----+----+----------+---------+
only showing top 5 rows
```

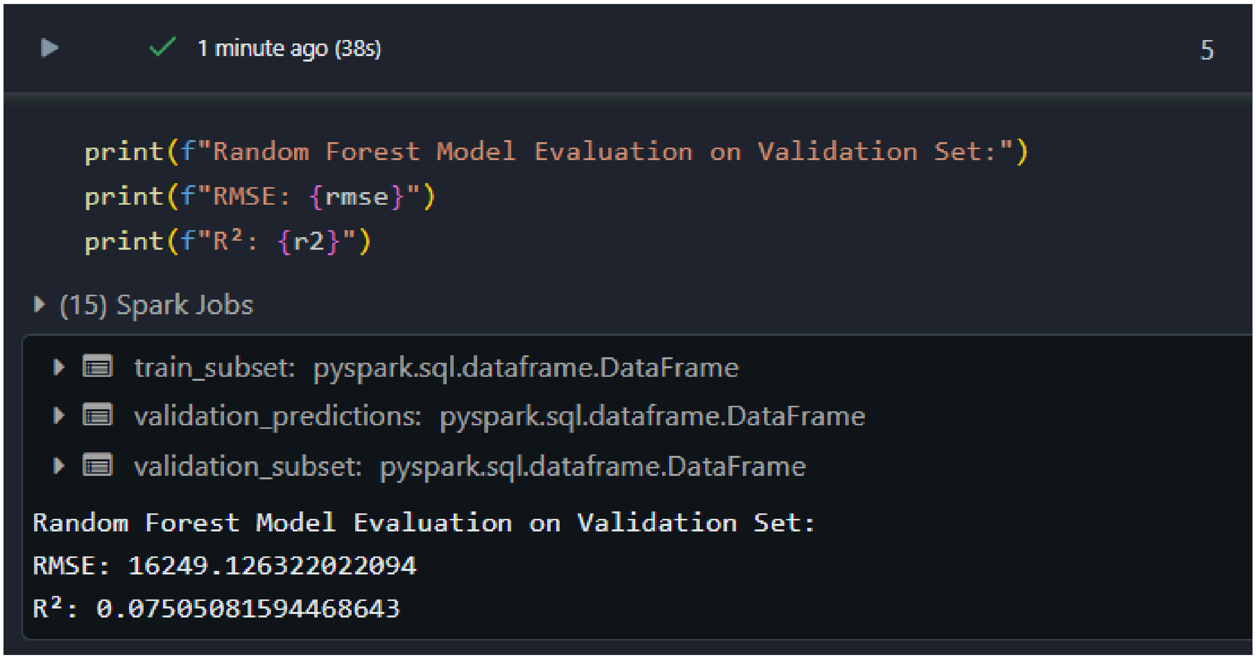


```python
# --------------------------------
# Step 3: Train the Regression Model
# --------------------------------
# Initialize and train the Linear Regression model
lr = LinearRegression(featuresCol="features", labelCol="Weekly_Sales", predictionCol="prediction")

# Fit the model
lr_model = lr.fit(train_data)

# Model evaluation on training data
training_summary = lr_model.summary
print("Model Training Summary:")
print(f"RMSE: {training_summary.rootMeanSquaredError}")
print(f"R2: {training_summary.r2}")
```

```
Model Training Summary:
RMSE: 16418.864256570218
R2: 0.05672431309675796
```

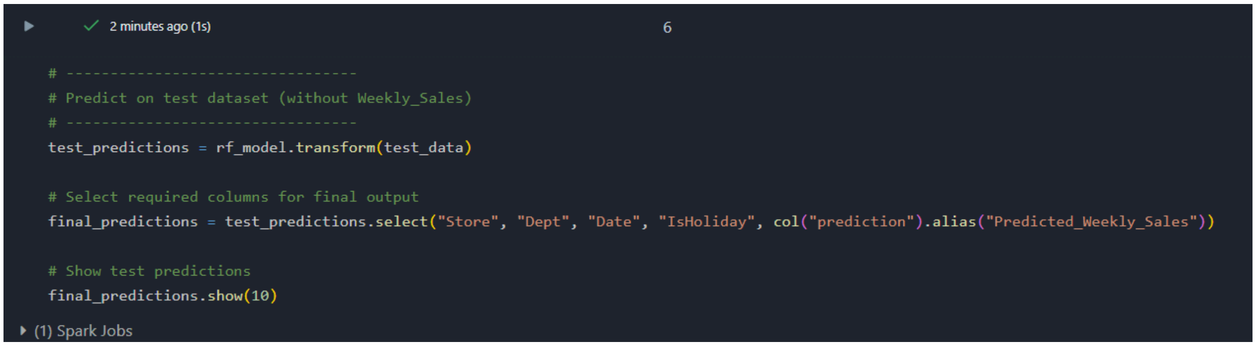


```python
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql.functions import col

# --------------------------------
# Split train data into training and validation sets
# --------------------------------
train_subset, validation_subset = train_data.randomSplit([0.8, 0.2], seed=42)

# Initialize Random Forest Regressor
rf = RandomForestRegressor(featuresCol="features", labelCol="Weekly_Sales", numTrees=50, maxDepth=10)

# Train the model on train subset
rf_model = rf.fit(train_subset)

# Predict on validation set
validation_predictions = rf_model.transform(validation_subset)

# Evaluate the model on validation data
evaluator_rmse = RegressionEvaluator(labelCol="Weekly_Sales", predictionCol="prediction", metricName="rmse")
evaluator_r2 = RegressionEvaluator(labelCol="Weekly_Sales", predictionCol="prediction", metricName="r2")

rmse = evaluator_rmse.evaluate(validation_predictions)
r2 = evaluator_r2.evaluate(validation_predictions)
```

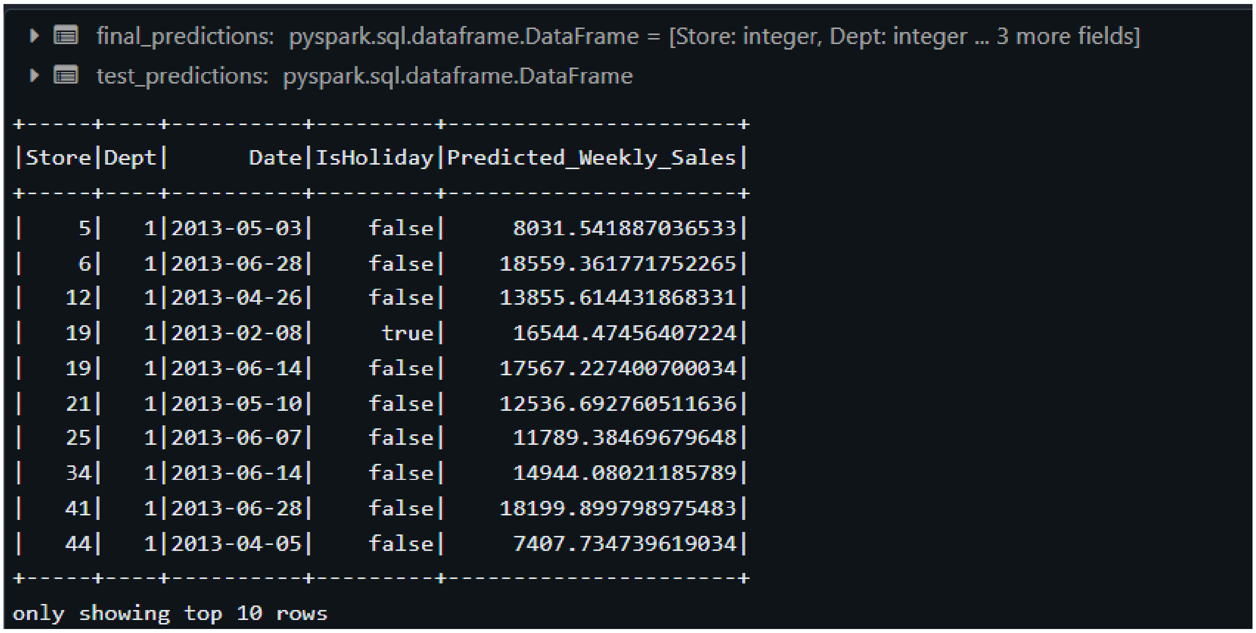


```python
print(f"Random Forest Model Evaluation on Validation Set:")
print(f"RMSE: {rmse}")
print(f"R²: {r2}")
```

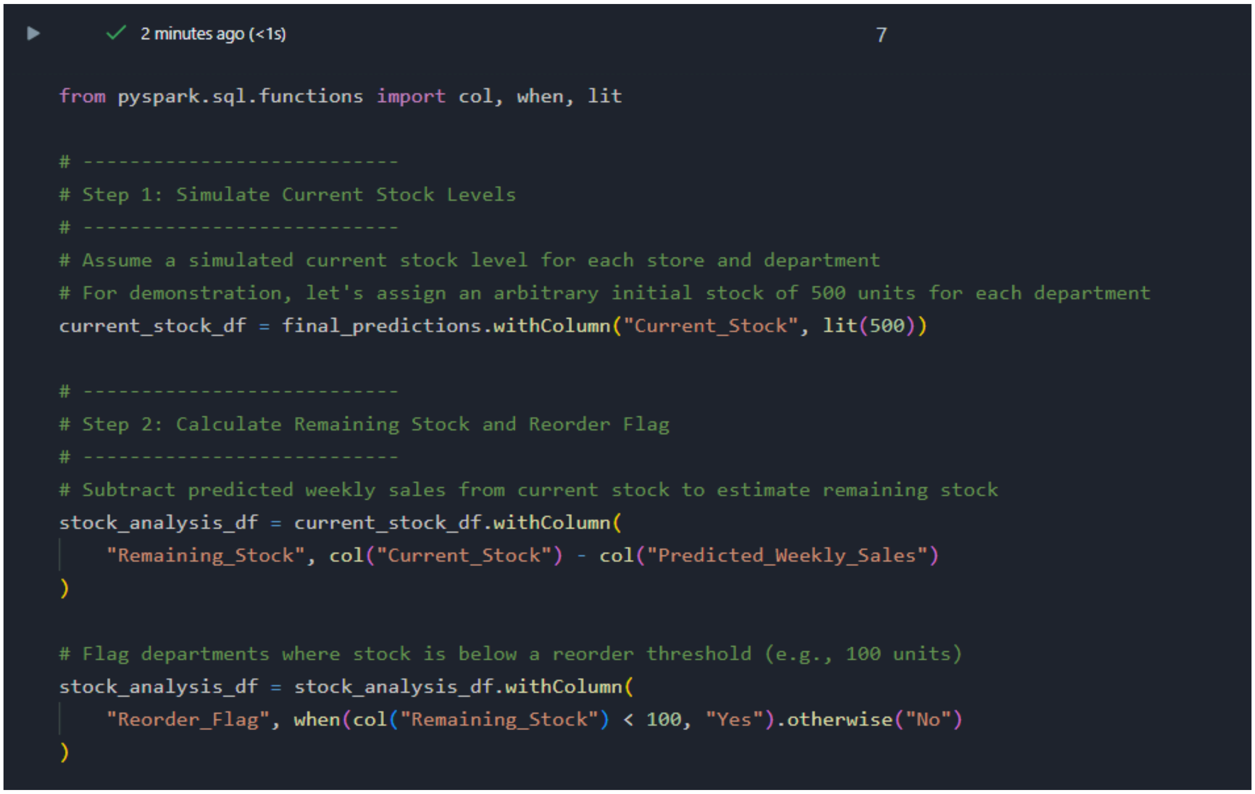
```
(15) Spark Jobs
train_subset:  pyspark.sql.dataframe.DataFrame
validation_predictions:  pyspark.sql.dataframe.DataFrame
validation_subset:  pyspark.sql.dataframe.DataFrame
Random Forest Model Evaluation on Validation Set:
RMSE: 16249.126322022094
R²: 0.07505081594468643
```



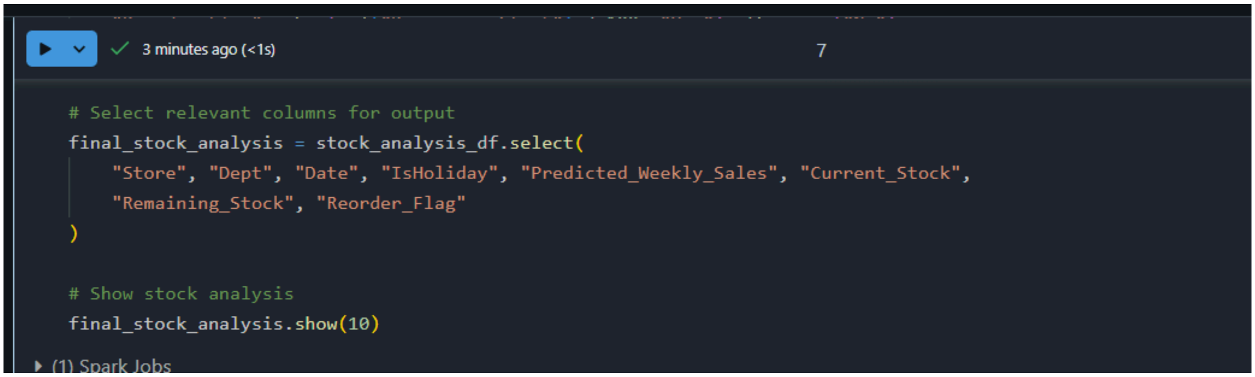
```
# --------------------------------
# Predict on test dataset (without Weekly_Sales)
# --------------------------------
test_predictions = rf_model.transform(test_data)

# Select required columns for final output
final_predictions = test_predictions.select("Store", "Dept", "Date", "IsHoliday", col("prediction").alias("Predicted_Weekly_Sales"))

# Show test predictions
final_predictions.show(10)
```



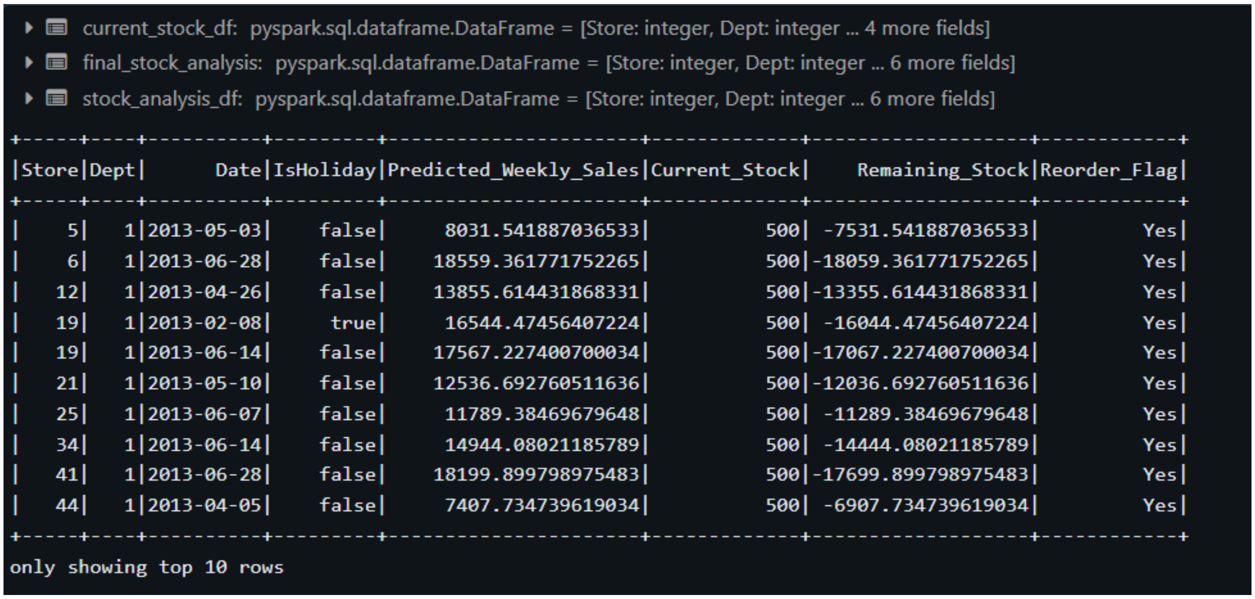
```
final_predictions:  pyspark.sql.dataframe.DataFrame = [Store: integer, Dept: integer ... 3 more fields]
test_predictions:  pyspark.sql.dataframe.DataFrame

+-----+----+----------+---------+----------------------+
|Store|Dept|      Date|IsHoliday|Predicted_Weekly_Sales|
+-----+----+----------+---------+----------------------+
|    5|   1|2013-05-03|    false|     8031.541887036533|
|    6|   1|2013-06-28|    false|    18559.361771752265|
|   12|   1|2013-04-26|    false|    13855.614431868331|
|   19|   1|2013-02-08|     true|     16544.47456407224|
|   19|   1|2013-06-14|    false|    17567.227400700034|
|   21|   1|2013-05-10|    false|    12536.692760511636|
|   25|   1|2013-06-07|    false|     11789.38469679648|
|   34|   1|2013-06-14|    false|     14944.08021185789|
|   41|   1|2013-06-28|    false|    18199.899798975483|
|   44|   1|2013-04-05|    false|     7407.734739619034|
+-----+----+----------+---------+----------------------+
only showing top 10 rows
```



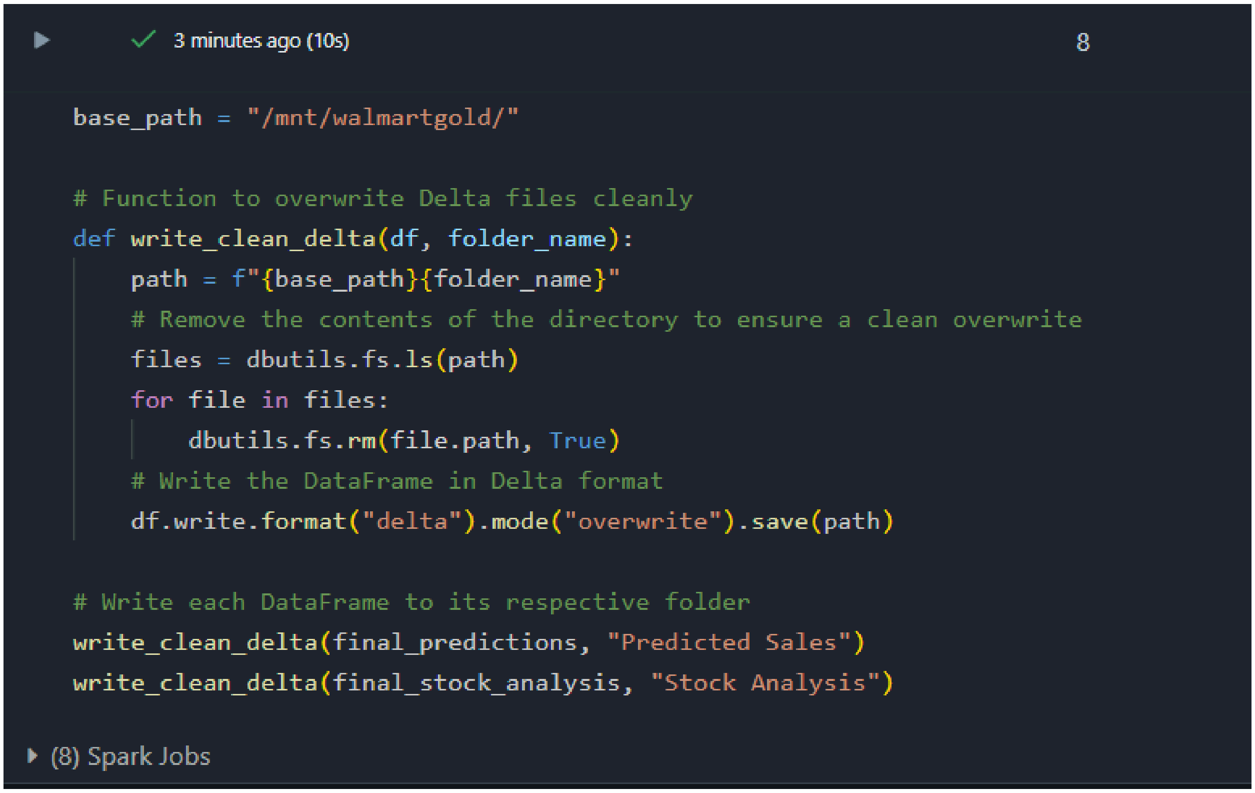
```
from pyspark.sql.functions import col, when, lit

# ---------------------------
# Step 1: Simulate Current Stock Levels
# ---------------------------
# Assume a simulated current stock level for each store and department
# For demonstration, let's assign an arbitrary initial stock of 500 units for each department
current_stock_df = final_predictions.withColumn("Current_Stock", lit(500))

# ---------------------------
# Step 2: Calculate Remaining Stock and Reorder Flag
# ---------------------------
# Subtract predicted weekly sales from current stock to estimate remaining stock
stock_analysis_df = current_stock_df.withColumn(
    "Remaining_Stock", col("Current_Stock") - col("Predicted_Weekly_Sales")
)

# Flag departments where stock is below a reorder threshold (e.g., 100 units)
stock_analysis_df = stock_analysis_df.withColumn(
    "Reorder_Flag", when(col("Remaining_Stock") < 100, "Yes").otherwise("No")
)
```



```
# Select relevant columns for output
final_stock_analysis = stock_analysis_df.select(
    "Store", "Dept", "Date", "IsHoliday", "Predicted_Weekly_Sales", "Current_Stock",
    "Remaining_Stock", "Reorder_Flag"
)

# Show stock analysis
final_stock_analysis.show(10)
```



```
current_stock_df:  pyspark.sql.dataframe.DataFrame = [Store: integer, Dept: integer ... 4 more fields]
final_stock_analysis:  pyspark.sql.dataframe.DataFrame = [Store: integer, Dept: integer ... 6 more fields]
stock_analysis_df:  pyspark.sql.dataframe.DataFrame = [Store: integer, Dept: integer ... 6 more fields]

+-----+----+----------+---------+----------------------+-------------+-------------------+------------+
|Store|Dept|      Date|IsHoliday|Predicted_Weekly_Sales|Current_Stock|    Remaining_Stock|Reorder_Flag|
+-----+----+----------+---------+----------------------+-------------+-------------------+------------+
|    5|   1|2013-05-03|    false|     8031.541887036533|          500| -7531.541887036533|         Yes|
|    6|   1|2013-06-28|    false|    18559.361771752265|          500|-18059.361771752265|         Yes|
|   12|   1|2013-04-26|    false|    13855.614431868331|          500|-13355.614431868331|         Yes|
|   19|   1|2013-02-08|     true|     16544.47456407224|          500| -16044.47456407224|         Yes|
|   19|   1|2013-06-14|    false|    17567.227400700034|          500|-17067.227400700034|         Yes|
|   21|   1|2013-05-10|    false|    12536.692760511636|          500|-12036.692760511636|         Yes|
|   25|   1|2013-06-07|    false|     11789.38469679648|          500| -11289.38469679648|         Yes|
|   34|   1|2013-06-14|    false|     14944.08021185789|          500| -14444.08021185789|         Yes|
|   41|   1|2013-06-28|    false|    18199.899798975483|          500|-17699.899798975483|         Yes|
|   44|   1|2013-04-05|    false|     7407.734739619034|          500| -6907.734739619034|         Yes|
+-----+----+----------+---------+----------------------+-------------+-------------------+------------+
only showing top 10 rows
```



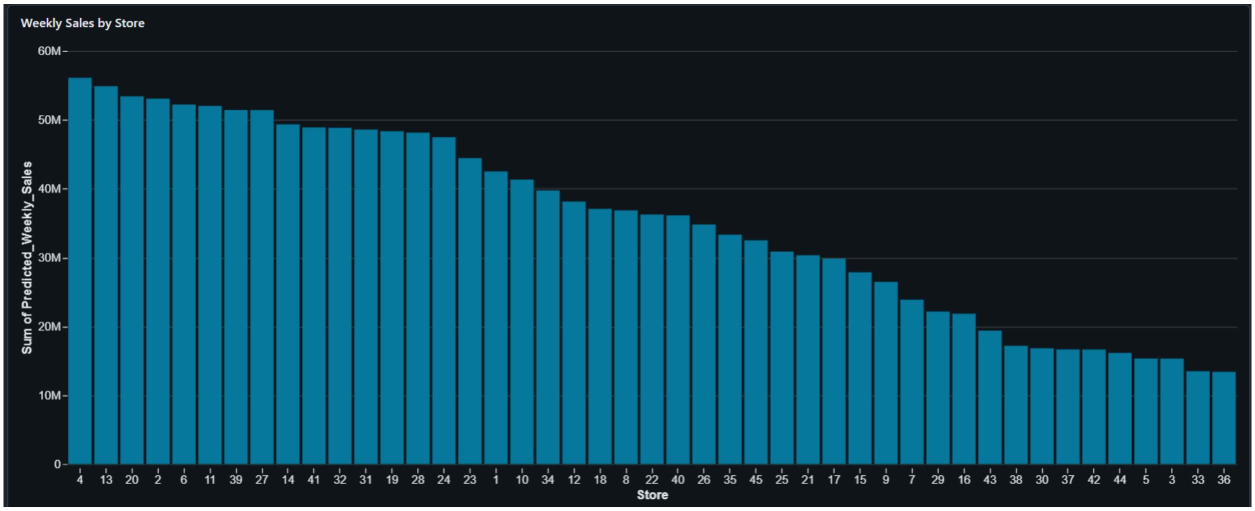
```python
base_path = "/mnt/walmartgold/"

# Function to overwrite Delta files cleanly
def write_clean_delta(df, folder_name):
    path = f"{base_path}{folder_name}"
    # Remove the contents of the directory to ensure a clean overwrite
    files = dbutils.fs.ls(path)
    for file in files:
        dbutils.fs.rm(file.path, True)
    # Write the DataFrame in Delta format
    df.write.format("delta").mode("overwrite").save(path)

# Write each DataFrame to its respective folder
write_clean_delta(final_predictions, "Predicted Sales")
write_clean_delta(final_stock_analysis, "Stock Analysis")
```

Reporting

In the project, for data visualization, I saved my final `final_stock_analysis` DataFrame as a Delta table and utilized Databricks Dashboards to build and present various visualizations. These visualizations provide insights into key business metrics related to stock management, sales performance, and trends. Below are the visualizations I created:

1. **Weekly Sales by Stores**:

   This visualization shows the weekly sales performance across different stores, helping identify trends and sales patterns.



Visual Representation:

- A bar chart visualization effectively illustrates the distribution of weekly sales across different stores.
- The x-axis represents individual stores, while the y-axis indicates the total predicted weekly sales for each store.

Key Insights:

- **Store Performance:** The visualization highlights the top-performing stores with the highest predicted weekly sales.

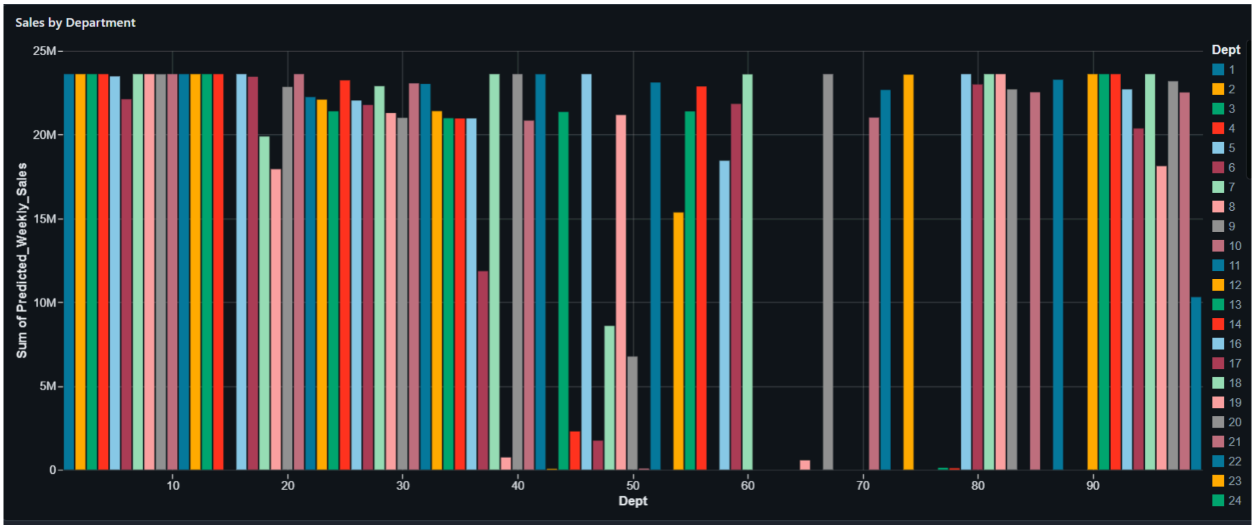
- **Sales Variation:** It reveals significant variation in sales across different stores, indicating potential opportunities for optimization and targeted strategies.
- **Store Ranking:** The stores are ranked based on their predicted weekly sales, providing a clear visual comparison.

Potential Use Cases:

- **Resource Allocation:** Identifying high-performing stores to allocate additional resources or promotional activities.
- **Inventory Management:** Optimizing inventory levels based on store-specific sales trends.
- **Marketing Strategies:** Tailoring marketing campaigns to specific stores to maximize impact.
- **Business Decision-Making:** Providing data-driven insights to inform strategic decisions.

2. **Sales by Department**:

   It highlights the sales performance by department, allowing for an understanding of how different product categories are performing.



Visual Representation:

- A grouped bar chart provides a clear visual comparison of the total predicted weekly sales for each department.
- The x-axis represents the department number, while the y-axis indicates the sum of predicted weekly sales.
- Different colors are used to differentiate between departments, making it easy to identify patterns and trends.

Key Insights:

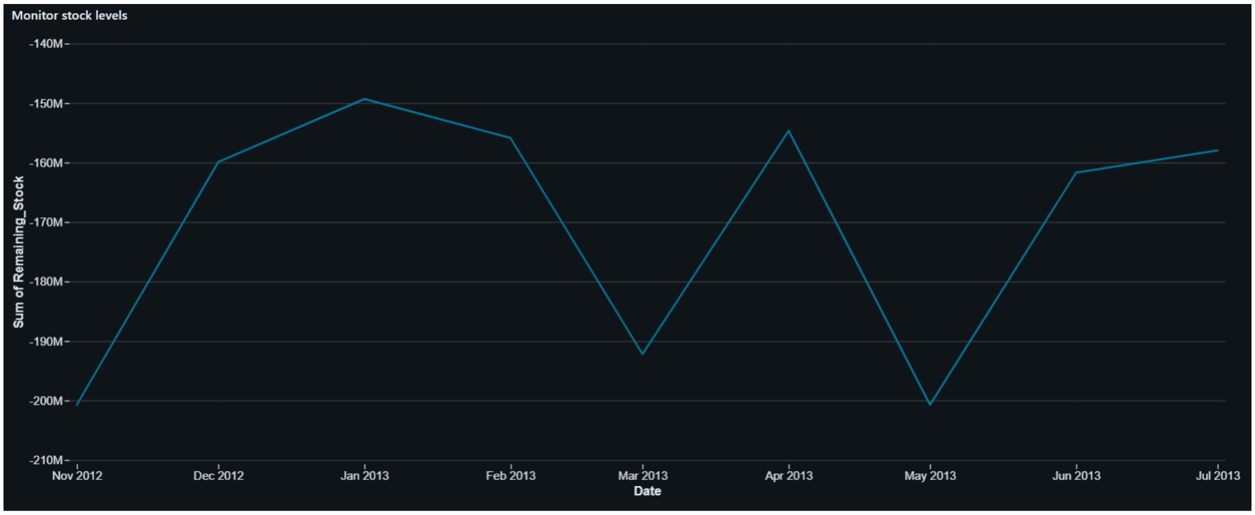
- **Departmental Performance:** The chart reveals significant variation in sales performance across different departments.
- **Top-Performing Departments:** Certain departments consistently generate higher sales compared to others.
- **Departmental Trends:** It's possible to identify departments with increasing or decreasing sales trends over time.

Potential Use Cases:

- **Resource Allocation:** Allocating resources to high-performing departments to further boost sales.
- **Product Assortment:** Analyzing the performance of different product categories within departments to optimize inventory and assortment.
- **Marketing Strategies:** Developing targeted marketing campaigns for specific departments.
- **Business Decision-Making:** Providing data-driven insights to inform strategic decisions about product mix, pricing, and promotions.

3. **Monitor Stock Levels**:

   This visualization tracks stock levels over time to ensure that the business is aware of inventory trends and can make data-driven decisions.



Visual Representation:

- A line chart provides a clear visual representation of the trend in stock levels over time.
- The x-axis represents the date, while the y-axis represents the sum of remaining stock.

Key Insights:

- **Stock Fluctuations:** The chart reveals significant fluctuations in stock levels throughout the period.
- **Seasonal Patterns:** There appear to be seasonal trends, with stock levels peaking in certain months and declining in others.
- **Potential Stockouts:** The sharp drops in stock levels suggest potential risks of stockouts during certain periods.

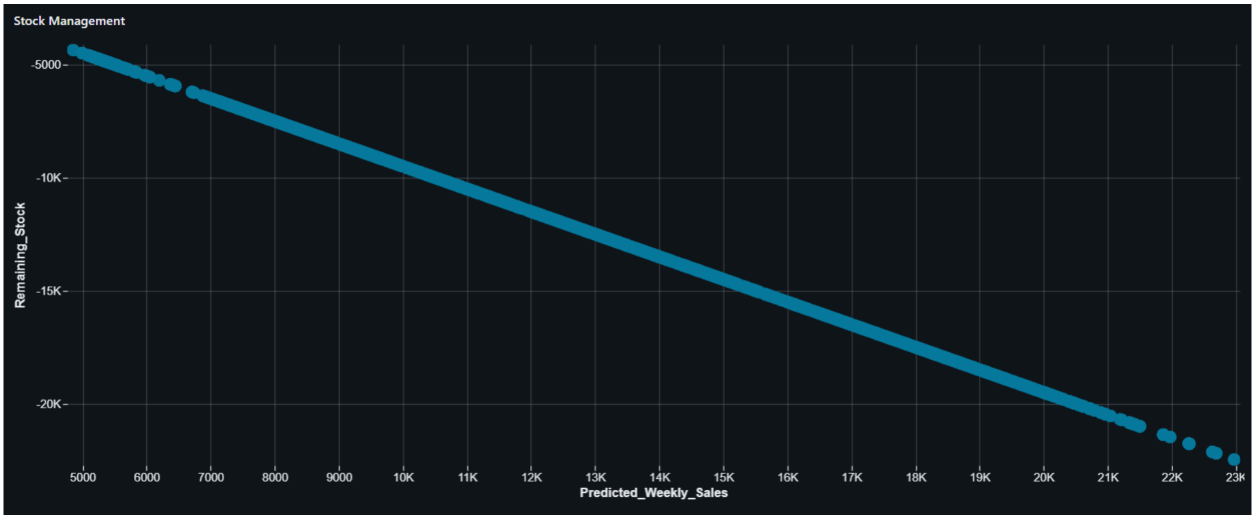
Potential Use Cases:

- **Inventory Management:** Identifying optimal stock levels to avoid both excess inventory and stockouts.
- **Demand Forecasting:** Predicting future demand to optimize inventory planning and replenishment.

- **Supply Chain Optimization:** Identifying bottlenecks in the supply chain that may contribute to stock level fluctuations.
- **Business Decision-Making:** Providing data-driven insights to inform strategic decisions about inventory management and procurement.

4. **Stock Management**:

This chart visualizes overall stock management, focusing on inventory turnover and the need for replenishment.



Visual Representation:

- A scatter plot illustrates the relationship between predicted weekly sales and remaining stock.
- Each data point represents a specific product or department.
- The x-axis represents the predicted weekly sales, while the y-axis represents the remaining stock.

Key Insights:

- **Negative Correlation:** The plot shows a clear negative correlation between predicted weekly sales and remaining stock. This suggests that as predicted sales increase, the remaining stock tends to decrease.

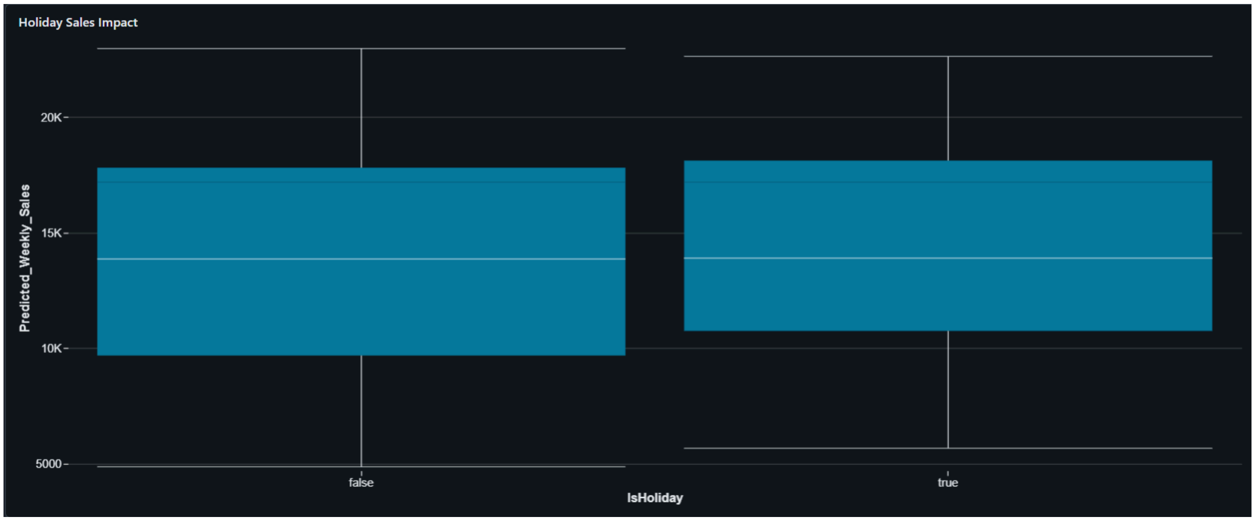
- **Inventory Management:** The visualization highlights the importance of effective inventory management to ensure that sufficient stock is available to meet demand without excessive holding costs.
- **Demand Forecasting:** Accurate demand forecasting is crucial for maintaining optimal stock levels.

Potential Use Cases:

- **Stock Replenishment:** Identifying products or departments that require replenishment based on their predicted sales and current stock levels.
- **Inventory Optimization:** Adjusting inventory levels to minimize stockouts and excess inventory.
- **Demand Planning:** Forecasting future demand to optimize production and procurement.
- **Business Decision-Making:** Providing data-driven insights to inform strategic decisions about inventory management and supply chain operations.

5. **Holiday Sales Impact**:

   It analyzes the impact of holidays on sales, helping the business prepare for seasonal demand fluctuations.



Visual Representation:

- A box plot illustrates the distribution of predicted weekly sales for both holiday and non-holiday weeks.
- The x-axis represents the holiday status (True or False), while the y-axis represents the predicted weekly sales.
- The box plot displays the median, quartiles, and outliers for each group.

Key Insights:

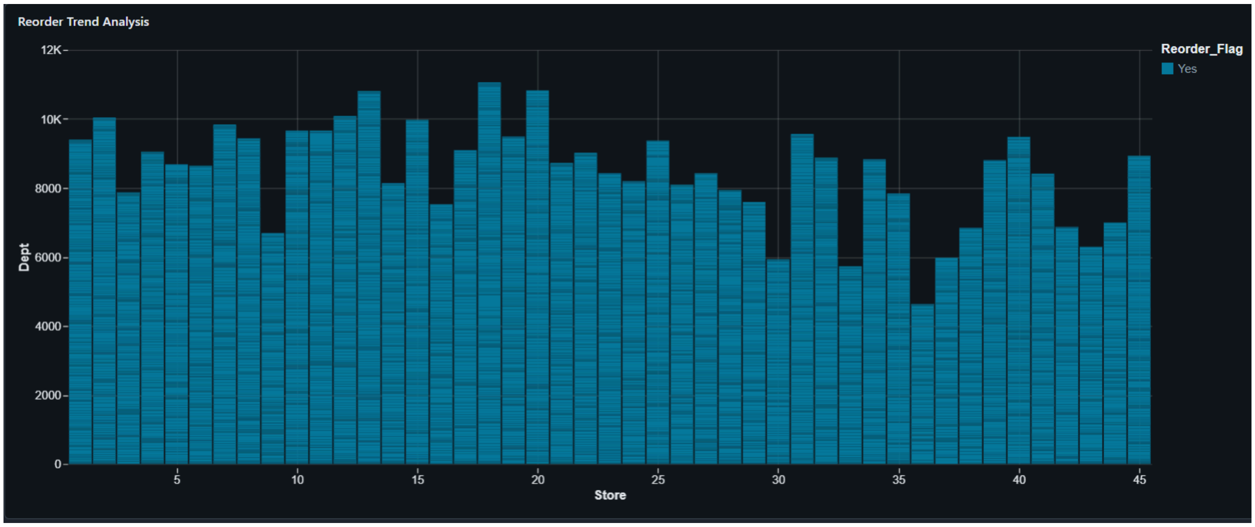
- **Holiday Impact:** The visualization clearly shows that holiday weeks tend to have higher predicted sales compared to non-holiday weeks.
- **Sales Variability:** The box plots reveal the variability in sales within each group, with holiday weeks exhibiting a wider range.
- **Outliers:** The presence of outliers suggests that some holiday weeks have significantly higher or lower sales compared to the average.

Potential Use Cases:

- **Sales Forecasting:** Incorporating holiday effects into sales forecasting models to improve accuracy.
- **Inventory Management:** Adjusting inventory levels to accommodate increased demand during holiday periods.
- **Marketing Strategies:** Developing targeted marketing campaigns for holiday seasons.
- **Business Decision-Making:** Providing data-driven insights to inform strategic decisions about resource allocation and promotions.

6. **Reorder Trend Analysis**:

   This visualization helps identify trends in stock reordering, indicating when and how often products need to be reordered.



Visual Representation:

- A bar chart effectively illustrates the distribution of reorder frequency across different departments and stores.
- The x-axis represents the store number.
- The y-axis indicates the count of reordered items within each department of a store.
- The color of the bars represents the reorder flag, indicating whether an item has been reordered.

Key Insights:

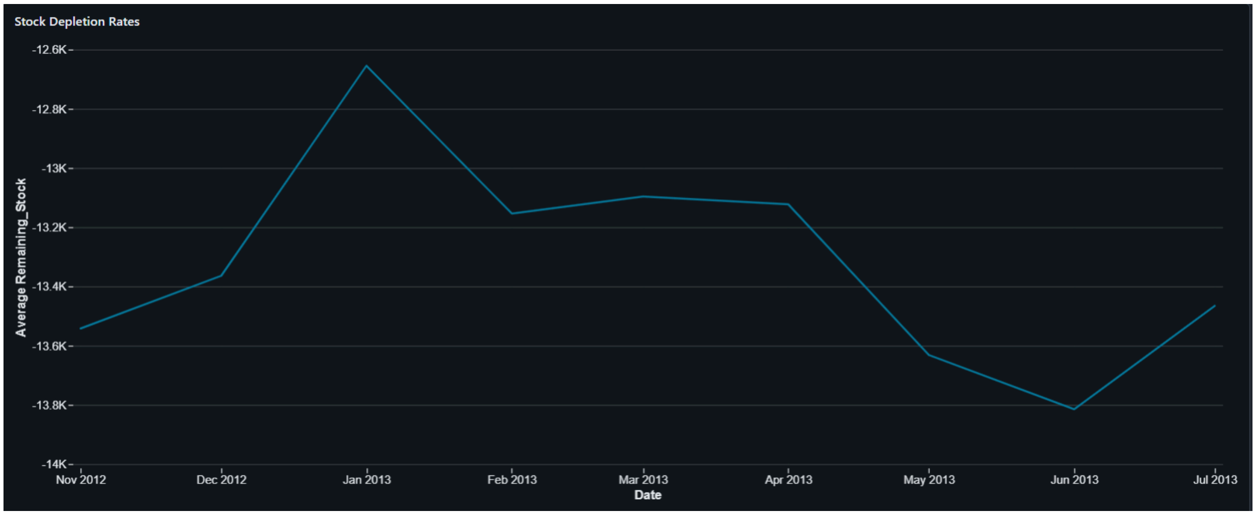
- **Reorder Patterns:** The chart highlights departments and stores with higher reorder frequencies, indicating potential popular or essential items.
- **Store-Department Variation:** It reveals variation in reorder patterns across different stores and departments, suggesting potential differences in customer preferences or inventory management strategies.
- **Reorder Opportunities:** Identifying departments with lower reorder frequencies can help identify potential opportunities for increased sales or inventory optimization.

Potential Use Cases:

- **Inventory Management:** Optimizing inventory levels based on reorder frequency to avoid stockouts and excess inventory.
- **Demand Forecasting:** Predicting future demand for products based on historical reorder patterns.
- **Marketing Strategies:** Targeting promotions or marketing campaigns to departments with high reorder frequencies.
- **Business Decision-Making:** Providing data-driven insights to inform strategic decisions.

7. **Stock Depletion Rates**:

It shows the rate at which stock is being depleted over time, aiding in stock management and inventory planning.



Visual Representation:

- A line chart illustrates the trend in average remaining stock over time.
- The x-axis represents the date, while the y-axis represents the average remaining stock.

Key Insights:

- **Stock Depletion:** The chart shows a general trend of decreasing stock levels over the period.

- **Seasonal Fluctuations:** There are noticeable fluctuations in stock levels, possibly due to seasonal demand patterns.
- **Potential Stockouts:** The sharp decline in stock levels towards the end of the period raises concerns about potential stockouts.

Potential Use Cases:

- **Inventory Management:** Identifying optimal stock levels to avoid both excess inventory and stockouts.
- **Demand Forecasting:** Predicting future demand to optimize inventory planning and replenishment.
- **Supply Chain Optimization:** Identifying bottlenecks in the supply chain that may contribute to stock level fluctuations.
- **Business Decision-Making:** Providing data-driven insights to inform strategic decisions about inventory management and procurement.

8. **Stock Depletion by Department**:

This chart provides insights into stock depletion rates for each department, allowing for targeted replenishment efforts.

Visual Representation:

- A bar chart provides a visual representation of the sum of remaining stock for each department, with a focus on departments flagged for reordering.
- The x-axis represents the department number, while the y-axis indicates the sum of remaining stock.
- The color of the bars differentiates between departments that are flagged for reordering and those that are not.

Key Insights:

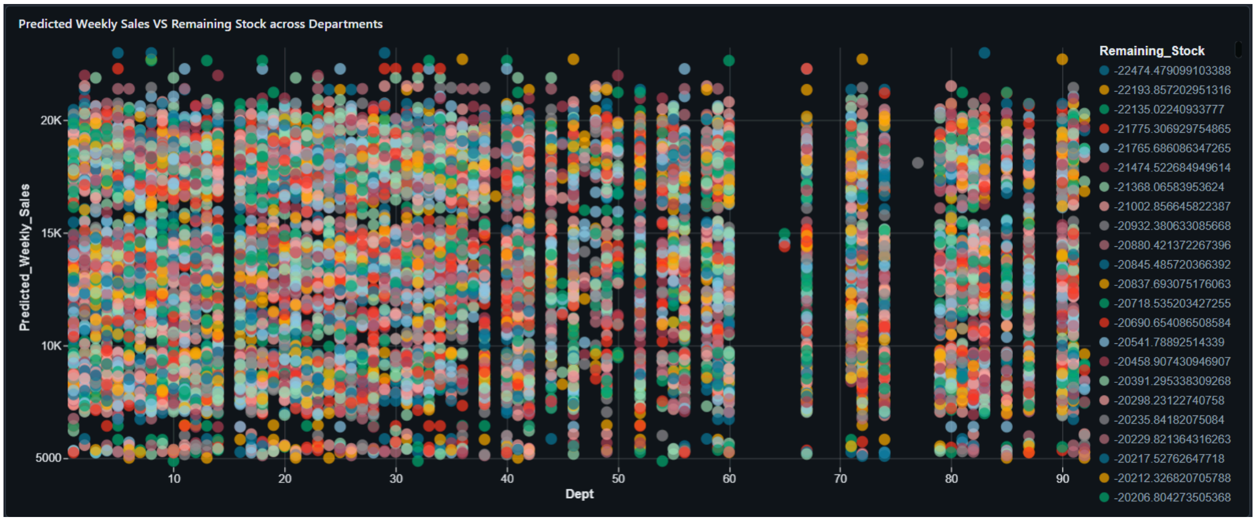
- **Stock Depletion:** The chart highlights departments that are low on stock and require reordering.
- **Departmental Variation:** There is significant variation in stock levels across different departments.
- **Reordering Prioritization:** The visualization can help prioritize reordering efforts for departments with the lowest stock levels.

Potential Use Cases:

- **Inventory Management:** Identifying departments that need urgent replenishment to avoid stockouts.
- **Supply Chain Optimization:** Streamlining the supply chain to ensure timely delivery of goods to low-stock departments.
- **Purchasing Decisions:** Optimizing purchasing decisions to minimize costs and maximize inventory turnover.
- **Business Decision-Making:** Providing data-driven insights to inform strategic decisions about inventory management and procurement.

9. **Predicted Weekly Sales Vs Remaining Stocks Across Departments**:

   It compares the predicted weekly sales with the remaining stock for each department, helping to avoid stockouts and plan for future stock needs.



Visual Representation:

- A scatter plot illustrates the relationship between predicted weekly sales and remaining stock across different departments.
- Each data point represents a specific product or department.
- The x-axis represents the department number, while the y-axis represents the predicted weekly sales.

- The color of each point represents the remaining stock level, providing additional information about inventory status.

Key Insights:

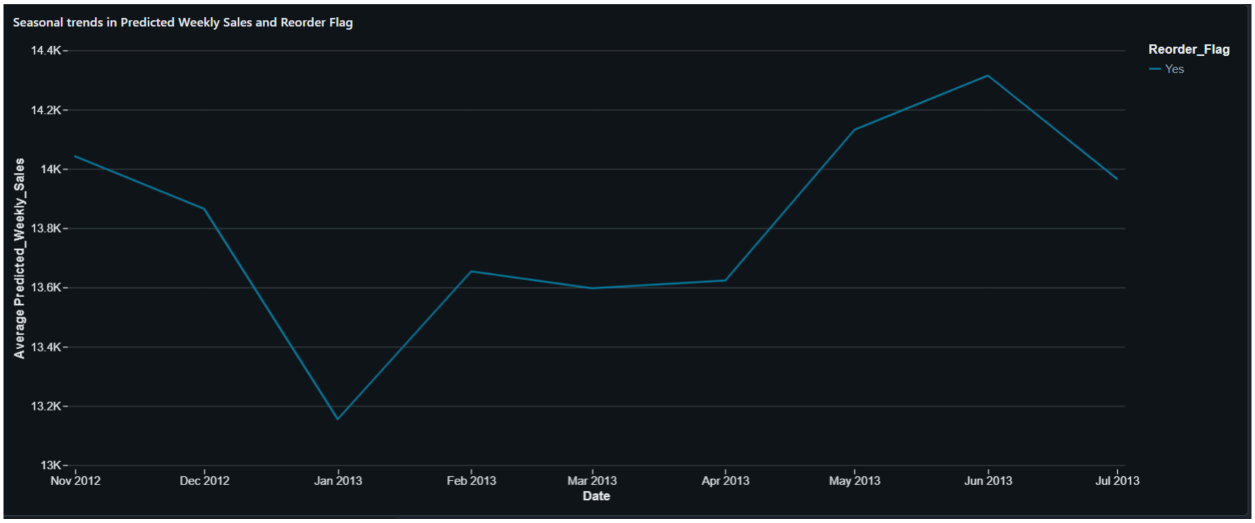
- **Departmental Variation:** The visualization reveals significant variation in the relationship between predicted sales and stock levels across different departments.
- **Stock Management Challenges:** Some departments may face challenges in managing stock levels, as evidenced by low remaining stock levels despite high predicted sales.
- **Demand Forecasting Accuracy:** The accuracy of demand forecasting can impact the effectiveness of inventory management.

Potential Use Cases:

- **Inventory Optimization:** Identifying departments that require more precise demand forecasting and inventory management strategies.
- **Supply Chain Improvement:** Addressing potential supply chain bottlenecks that may contribute to stockouts.
- **Product Assortment:** Analyzing the relationship between product assortment and stock levels to optimize product offerings.
- **Business Decision-Making:** Providing data-driven insights to inform strategic decisions about inventory management, procurement, and sales planning.

10. **Seasonal Trends in Predicted Weekly Sales and Reorder Flag**:

This visualization analyzes seasonal trends in predicted sales, along with reorder flags, to forecast future demand.



Visual Representation:

This line chart effectively illustrates the seasonal trends in average predicted weekly sales and reorder frequency over time. The x-axis represents the date range from November 2012 to July 2013, while the y-axis indicates the average predicted weekly sales.

Key Insights:

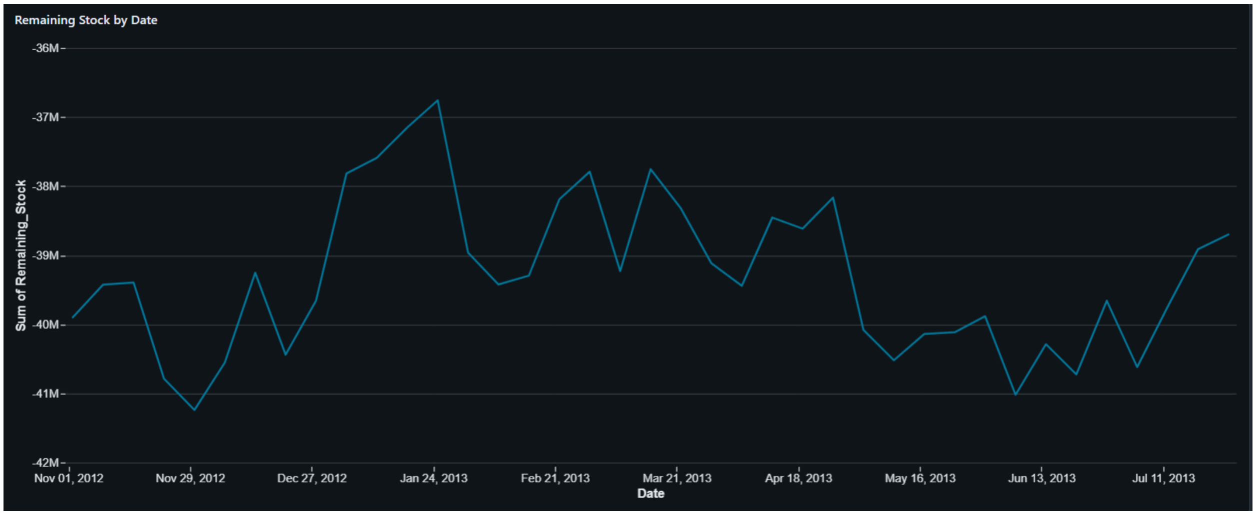
- **Seasonal Patterns:** The chart reveals distinct seasonal patterns in both sales and reorder frequency, with fluctuations throughout the year.
- **Peak Sales Periods:** There are periods of increased sales and reorder frequency, potentially corresponding to holiday seasons or promotional events.
- **Seasonal Impact:** Understanding these seasonal patterns can help optimize inventory levels, staffing, and marketing efforts.

Potential Use Cases:

- **Demand Forecasting:** Predicting future demand based on historical seasonal patterns.
- **Inventory Management:** Adjusting inventory levels to align with seasonal fluctuations in demand.
- **Staffing:** Optimizing staffing levels to meet varying customer demand throughout the year.
- **Marketing Strategies:** Planning and executing targeted marketing campaigns during peak sales periods.

11. **Remaining Stock by Date**:

It shows the remaining stock for each product by date, helping businesses monitor inventory and avoid shortages.



Visual Representation:

- A line chart illustrates the trend in the sum of remaining stock over time.
- The x-axis represents the date, while the y-axis represents the sum of remaining stock.

Key Insights:

- **Stock Fluctuations:** The chart reveals significant fluctuations in stock levels throughout the period.

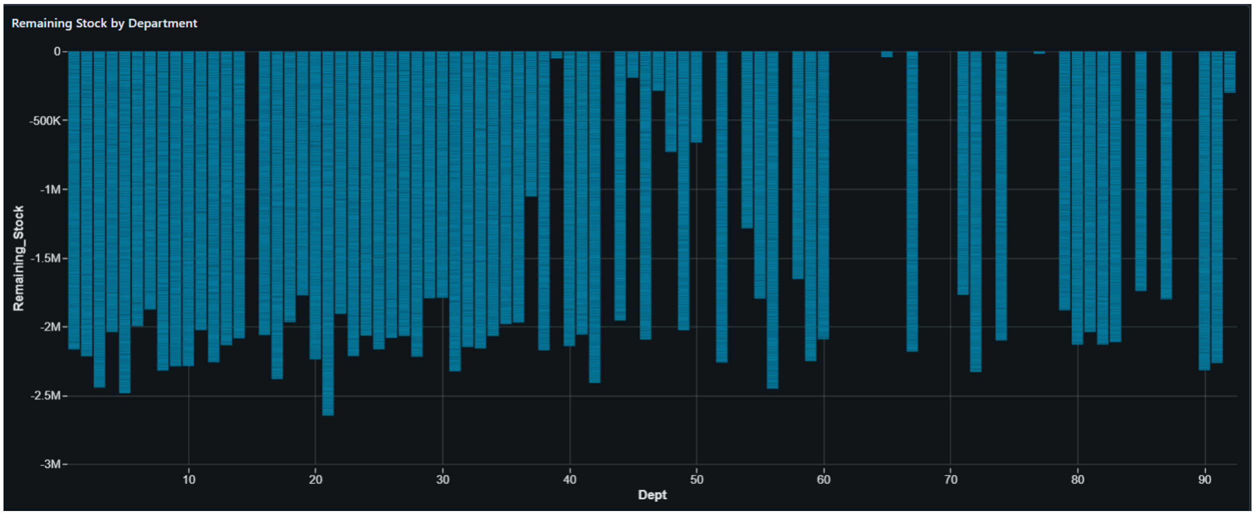
- **Seasonal Patterns:** There appear to be seasonal trends, with stock levels peaking in certain months and declining in others.
- **Potential Stockouts:** The sharp drops in stock levels suggest potential risks of stockouts during certain periods.

Potential Use Cases:

- **Inventory Management:** Identifying optimal stock levels to avoid both excess inventory and stockouts.
- **Demand Forecasting:** Predicting future demand to optimize inventory planning and replenishment.
- **Supply Chain Optimization:** Identifying bottlenecks in the supply chain that may contribute to stock level fluctuations.
- **Business Decision-Making:** Providing data-driven insights to inform strategic decisions about inventory management and procurement.

12. **Remaining Stock by Department**:

This visualization breaks down the remaining stock by department, providing a department-level view of inventory status.



Visual Representation:

- A bar chart provides a visual representation of the remaining stock for each department.
- The x-axis represents the department number, while the y-axis indicates the remaining stock level.

Key Insights:

- **Stock Variation:** The chart reveals significant variation in stock levels across different departments.
- **Low Stock Levels:** Several departments have very low or negative stock levels, indicating potential stockouts or overstocking issues.
- **Departmental Performance:** The visualization can help identify departments with consistently low or high stock levels.

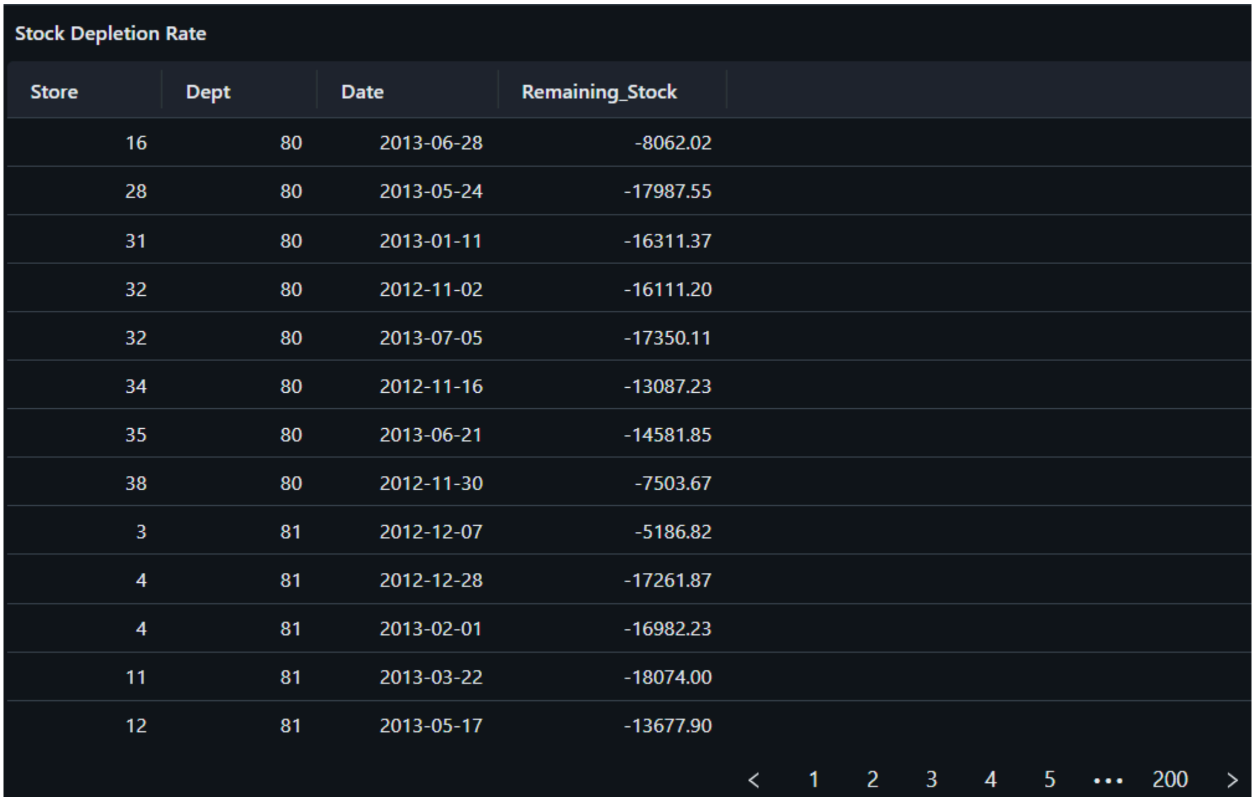
Potential Use Cases:

- **Inventory Management:** Identifying departments that need urgent replenishment or stock reduction.
- **Supply Chain Optimization:** Streamlining the supply chain to ensure timely delivery of goods to low-stock departments.
- **Purchasing Decisions:** Optimizing purchasing decisions to minimize costs and maximize inventory turnover.
- **Business Decision-Making:** Providing data-driven insights to inform strategic decisions about inventory management and procurement.

Additionally, I created several **Tables** in the dashboard to display exact values for key metrics, which include:

1. **Stock Depletion Rate**:

   A table showing the rate at which stock is depleting over time.

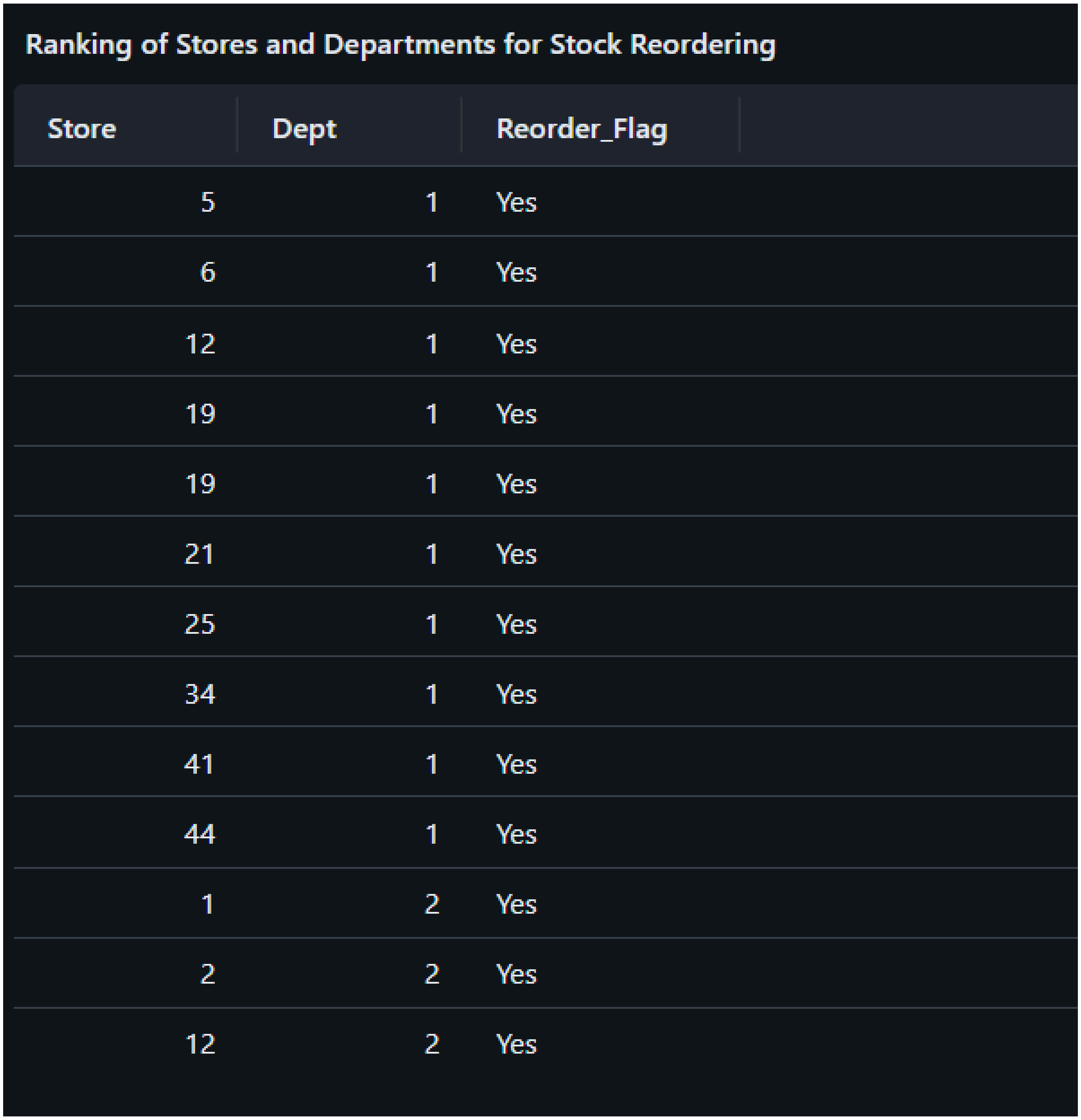


Stock Depletion Rate

Store	Dept	Date	Remaining_Stock
16	80	2013-06-28	-8062.02
28	80	2013-05-24	-17987.55
31	80	2013-01-11	-16311.37
32	80	2012-11-02	-16111.20
32	80	2013-07-05	-17350.11
34	80	2012-11-16	-13087.23
35	80	2013-06-21	-14581.85
38	80	2012-11-30	-7503.67
3	81	2012-12-07	-5186.82
4	81	2012-12-28	-17261.87
4	81	2013-02-01	-16982.23
11	81	2013-03-22	-18074.00
12	81	2013-05-17	-13677.90

2. **Ranking of Stores and Departments for Stock Reordering**:

   A ranking table that identifies stores and departments requiring urgent stock reordering.

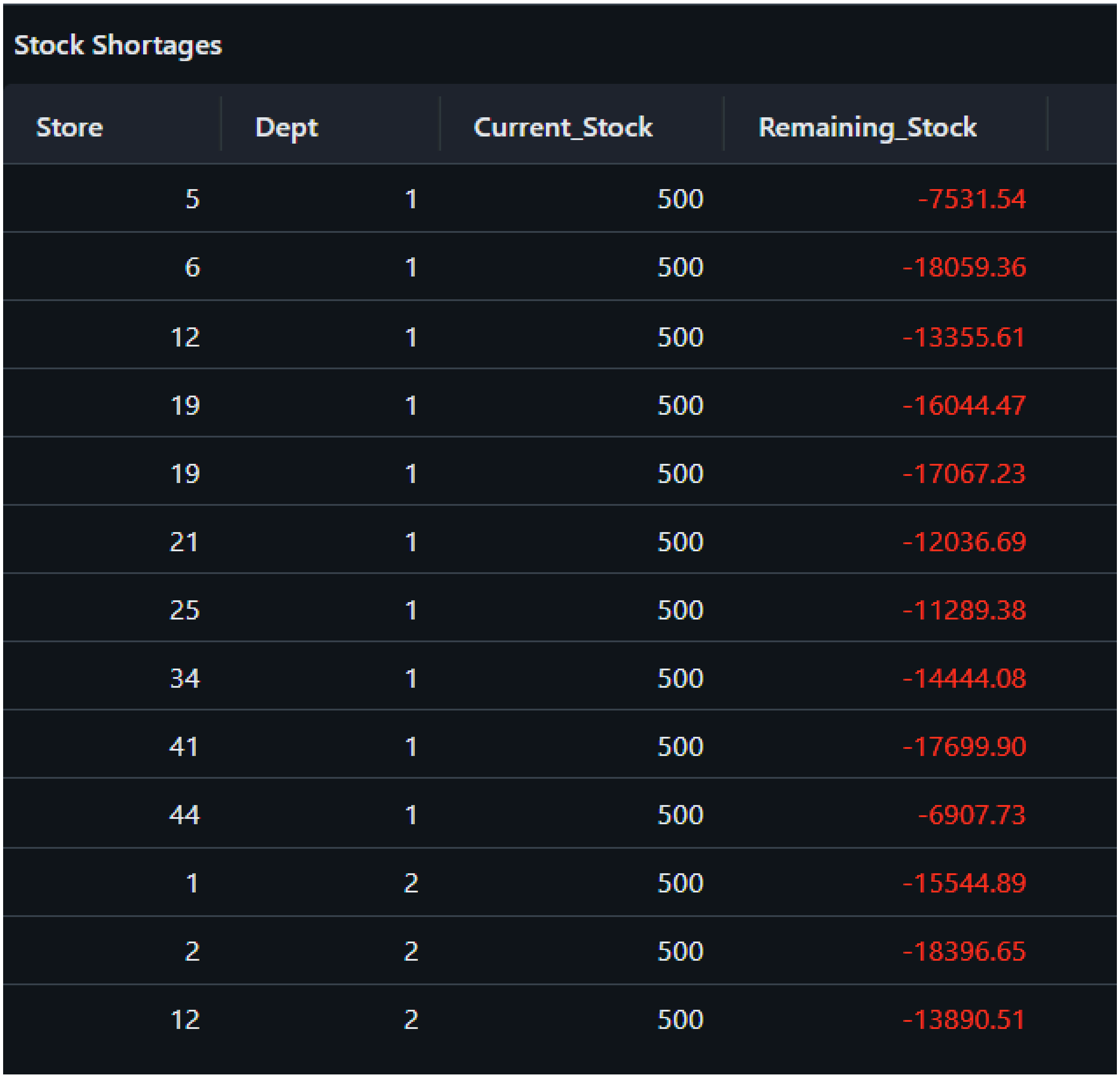


Ranking of Stores and Departments for Stock Reordering

Store	Dept	Reorder_Flag
5	1	Yes
6	1	Yes
12	1	Yes
19	1	Yes
19	1	Yes
21	1	Yes
25	1	Yes
34	1	Yes
41	1	Yes
44	1	Yes
1	2	Yes
2	2	Yes
12	2	Yes

3. **Stock Shortages**:

   A table highlighting any stock shortages, helping prioritize restocking efforts.



Stock Shortages

Store	Dept	Current_Stock	Remaining_Stock
5	1	500	-7531.54
6	1	500	-18059.36
12	1	500	-13355.61
19	1	500	-16044.47
19	1	500	-17067.23
21	1	500	-12036.69
25	1	500	-11289.38
34	1	500	-14444.08
41	1	500	-17699.90
44	1	500	-6907.73
1	2	500	-15544.89
2	2	500	-18396.65
12	2	500	-13890.51

4. **Predicted Weekly Sales**:

 A table displaying the predicted sales for each week, aiding in inventory planning.

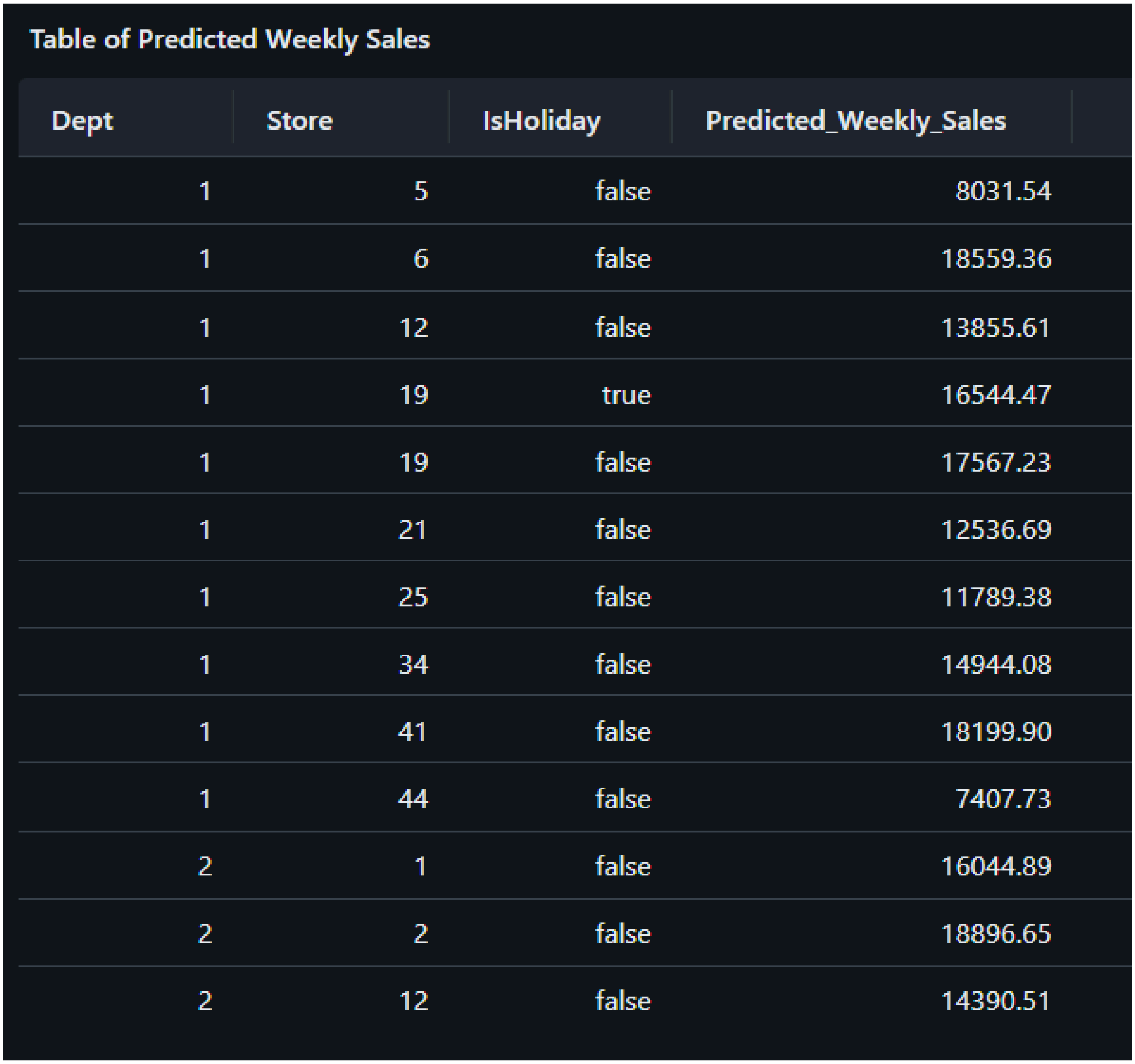


Table of Predicted Weekly Sales

Dept	Store	IsHoliday	Predicted_Weekly_Sales
1	5	false	8031.54
1	6	false	18559.36
1	12	false	13855.61
1	19	true	16544.47
1	19	false	17567.23
1	21	false	12536.69
1	25	false	11789.38
1	34	false	14944.08
1	41	false	18199.90
1	44	false	7407.73
2	1	false	16044.89
2	2	false	18896.65
2	12	false	14390.51

5. **Weekly Sales by Department**:

   A table that presents weekly sales data broken down by department.

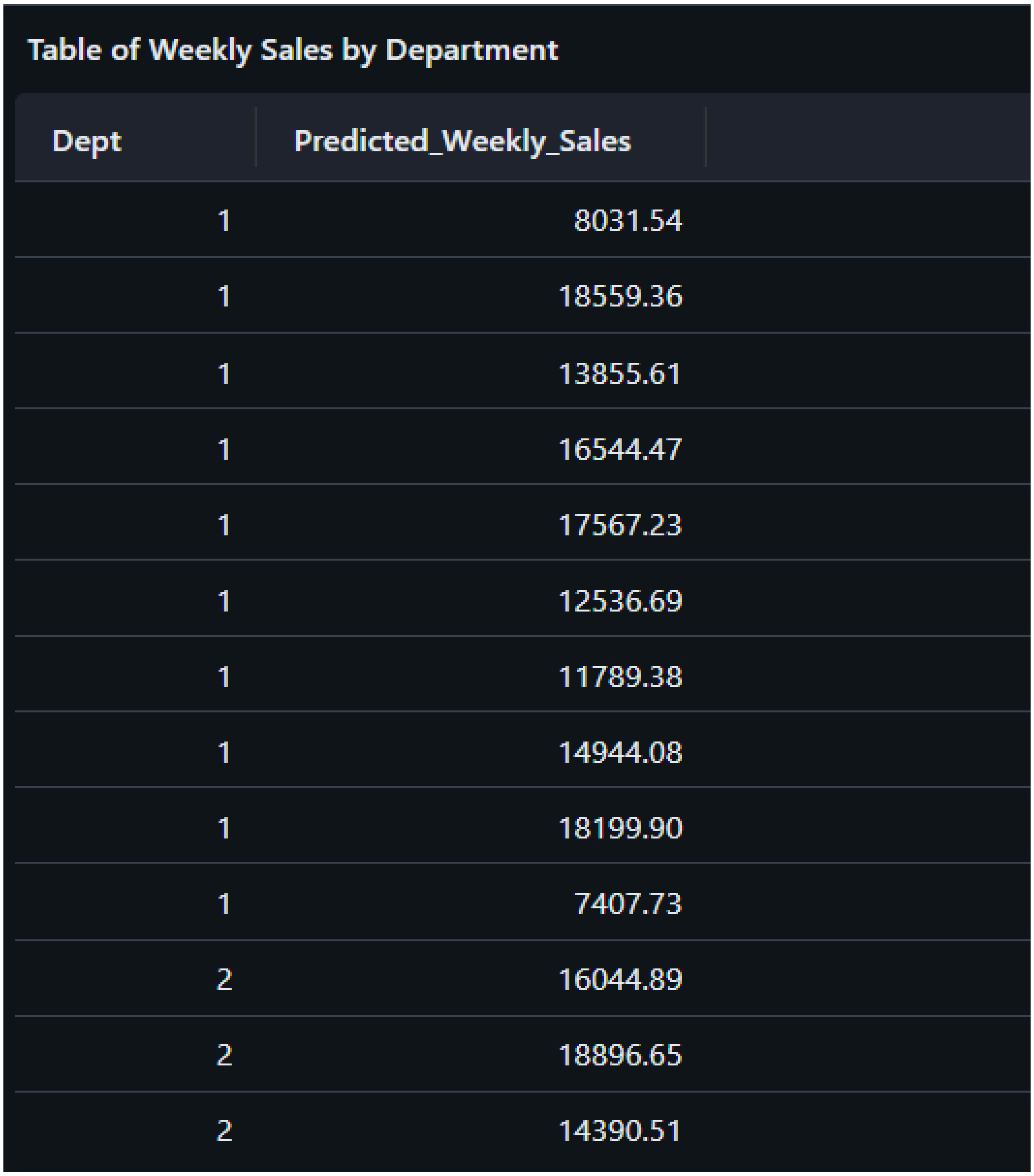
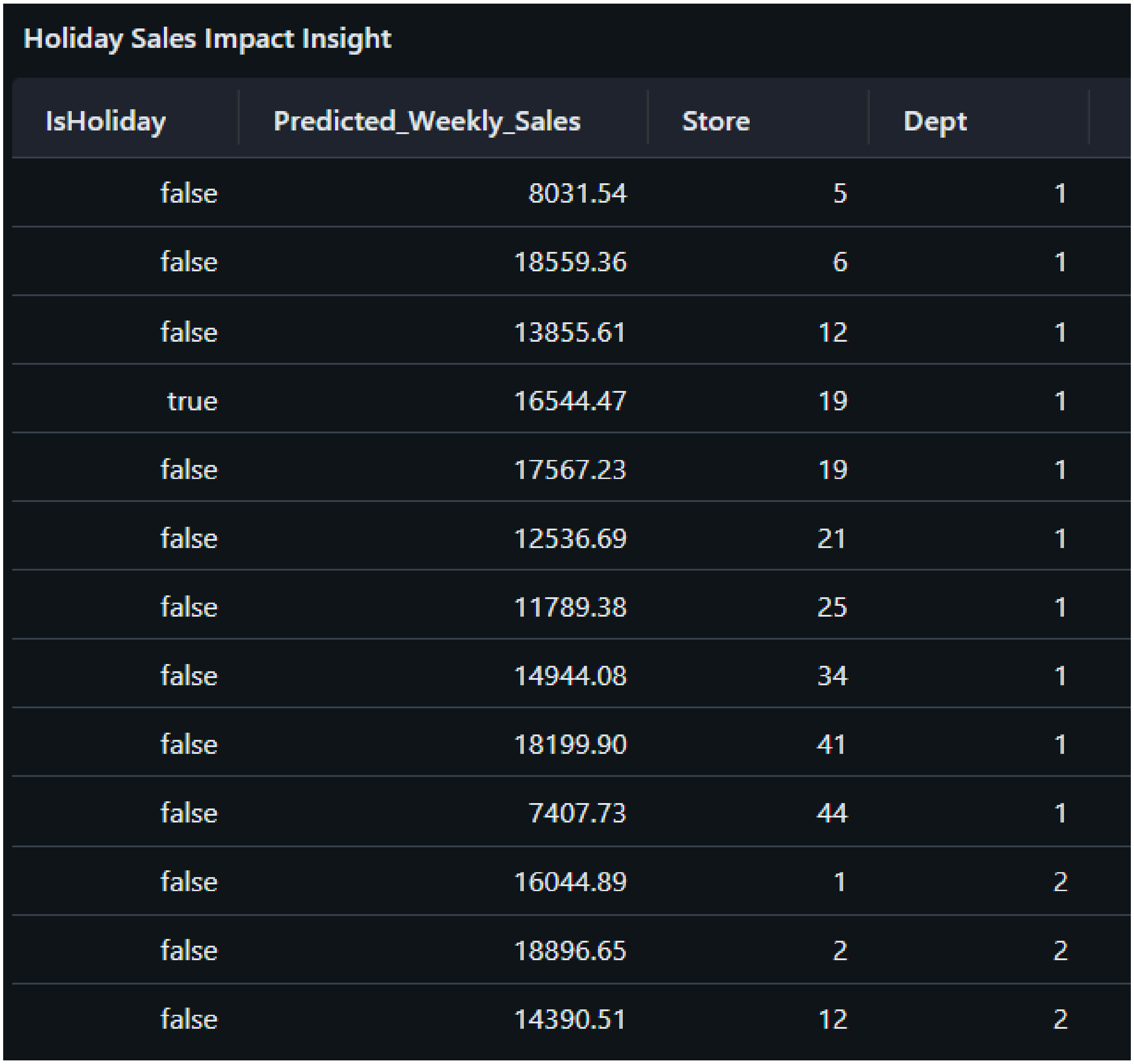


Table of Weekly Sales by Department

Dept	Predicted_Weekly_Sales
1	8031.54
1	18559.36
1	13855.61
1	16544.47
1	17567.23
1	12536.69
1	11789.38
1	14944.08
1	18199.90
1	7407.73
2	16044.89
2	18896.65
2	14390.51

6. **Holiday Sales Impact**:

   A table showing the effect of holidays on sales performance.



Holiday Sales Impact Insight

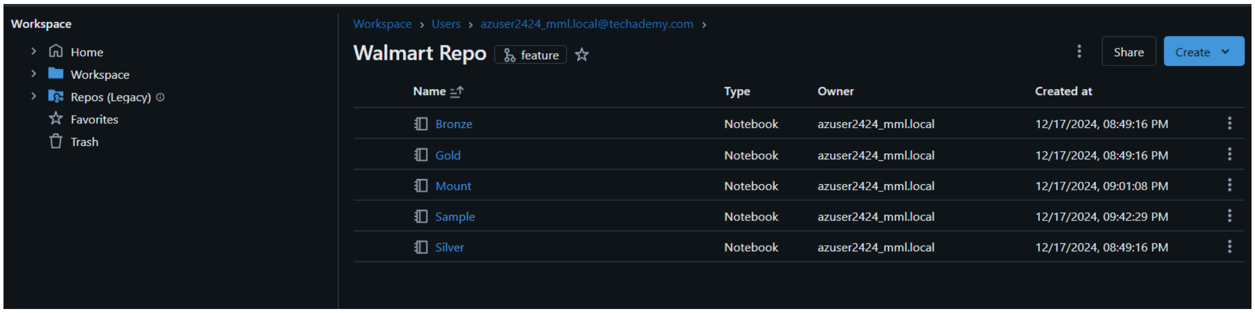
IsHoliday	Predicted_Weekly_Sales	Store	Dept
false	8031.54	5	1
false	18559.36	6	1
false	13855.61	12	1
true	16544.47	19	1
false	17567.23	19	1
false	12536.69	21	1
false	11789.38	25	1
false	14944.08	34	1
false	18199.90	41	1
false	7407.73	44	1
false	16044.89	1	2
false	18896.65	2	2
false	14390.51	12	2

These visualizations and tables collectively help in monitoring, managing, and forecasting stock and sales, contributing to efficient stock management and decision-making.

Continuous Integration and Continuous Deployment(CI/CD)

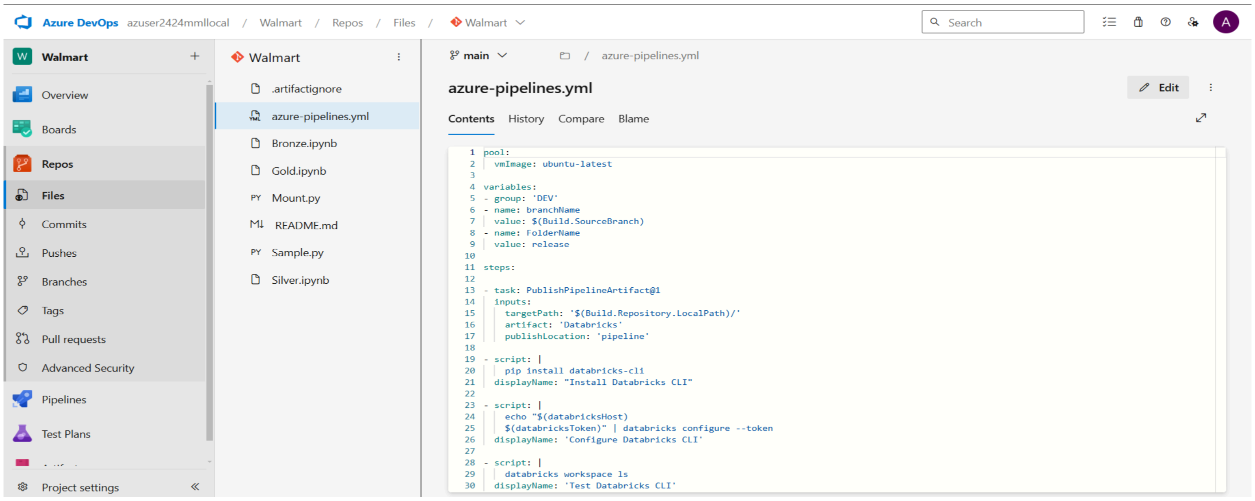
For Continuous Integration and Continuous Deployment (CI/CD), I used Azure DevOps to automate the process and streamline updates for my project. The process involves several steps that ensure the latest changes in the Databricks notebooks are deployed efficiently.

1. **Azure Databricks Setup**: I first cloned the Azure Databricks Walmart project folders, which contain all the Bronze, Silver, and Gold notebooks of the project. These notebooks were crucial in the data pipeline process, from raw data ingestion (Bronze) to refined data (Silver) and the final analysis (Gold).



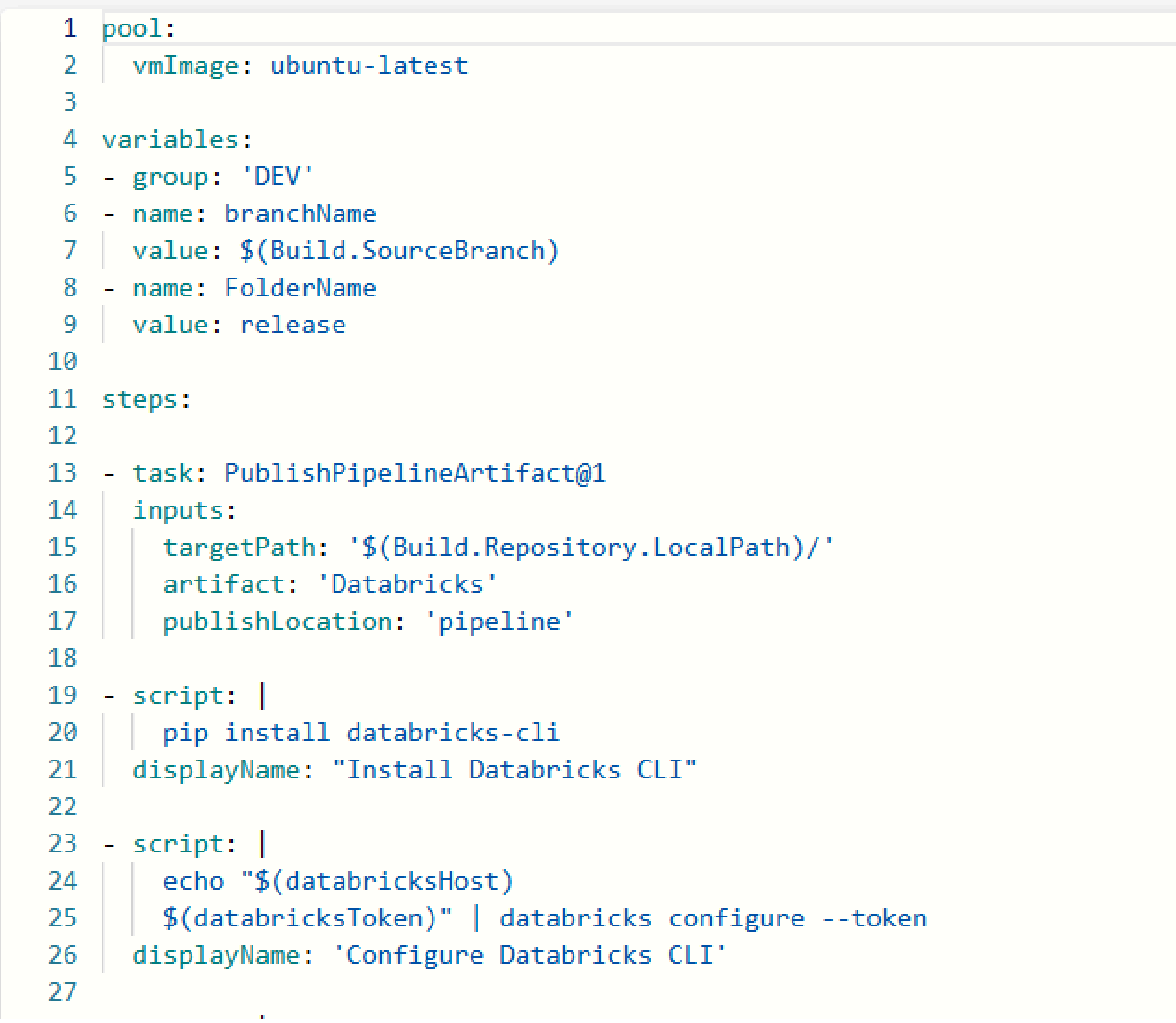
Name	Type	Owner	Created at
Bronze	Notebook	azuser2424_mml.local	12/17/2024, 08:49:16 PM
Gold	Notebook	azuser2424_mml.local	12/17/2024, 08:49:16 PM
Mount	Notebook	azuser2424_mml.local	12/17/2024, 09:01:08 PM
Sample	Notebook	azuser2424_mml.local	12/17/2024, 09:42:29 PM
Silver	Notebook	azuser2424_mml.local	12/17/2024, 08:49:16 PM

2. **Pipeline Creation**: I created a CI/CD pipeline in Azure DevOps using YAML code. This pipeline is designed to trigger automatically whenever there are any changes in the Databricks notebooks.

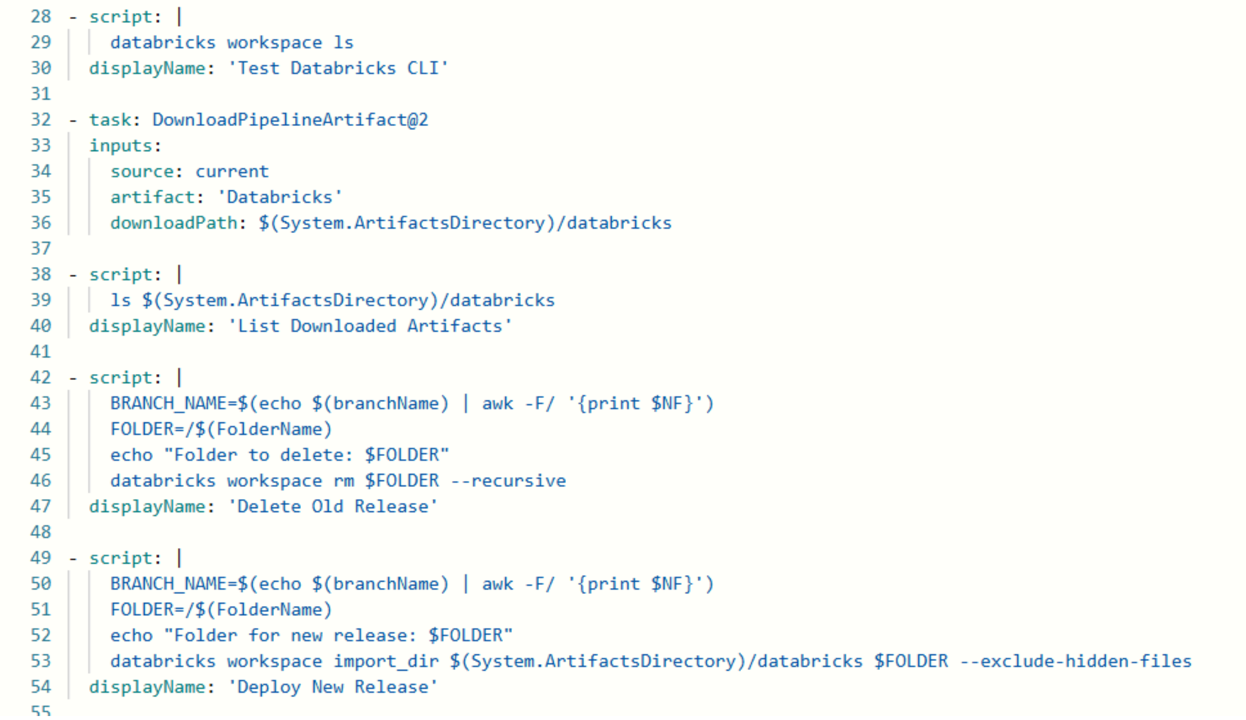


azure-pipelines.yml

```yaml
pool:
  vmImage: ubuntu-latest

variables:
- group: 'DEV'
- name: branchName
  value: $(Build.SourceBranch)
- name: FolderName
  value: release

steps:

- task: PublishPipelineArtifact@1
  inputs:
    targetPath: '$(Build.Repository.LocalPath)/'
    artifact: 'Databricks'
    publishLocation: 'pipeline'

- script: |
    pip install databricks-cli
  displayName: "Install Databricks CLI"

- script: |
    echo "$(databricksHost)
    $(databricksToken)" | databricks configure --token
  displayName: 'Configure Databricks CLI'

- script: |
    databricks workspace ls
  displayName: 'Test Databricks CLI'
```

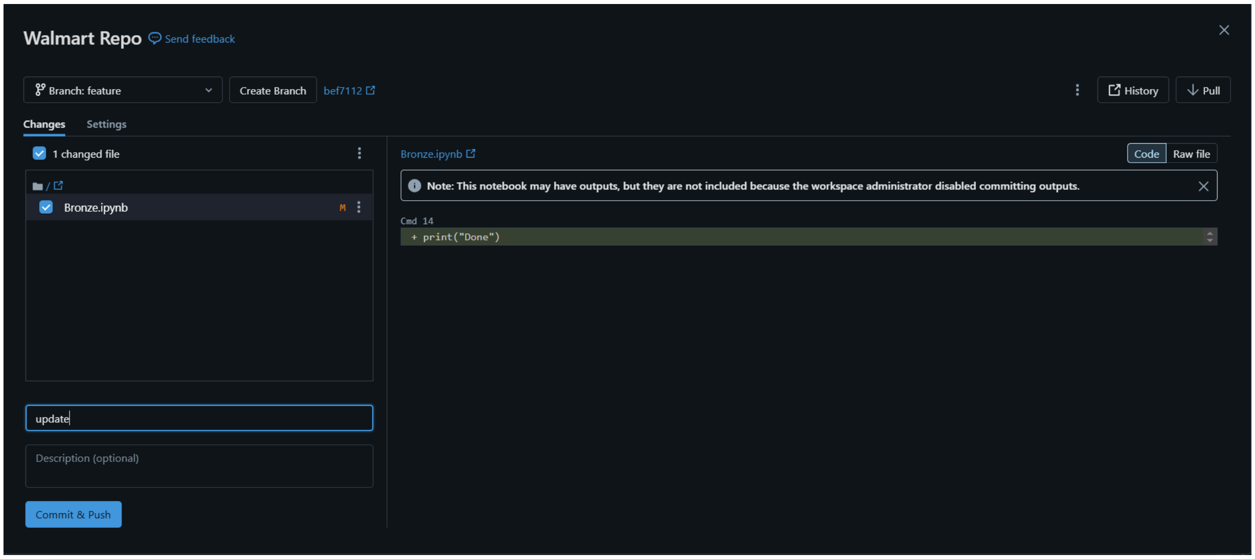


```yaml
pool:
  vmImage: ubuntu-latest

variables:
- group: 'DEV'
- name: branchName
  value: $(Build.SourceBranch)
- name: FolderName
  value: release

steps:

- task: PublishPipelineArtifact@1
  inputs:
    targetPath: '$(Build.Repository.LocalPath)/'
    artifact: 'Databricks'
    publishLocation: 'pipeline'

- script: |
    pip install databricks-cli
  displayName: "Install Databricks CLI"

- script: |
    echo "$(databricksHost)
    $(databricksToken)" | databricks configure --token
  displayName: 'Configure Databricks CLI'

```



```
- script: |
    databricks workspace ls
  displayName: 'Test Databricks CLI'

- task: DownloadPipelineArtifact@2
  inputs:
    source: current
    artifact: 'Databricks'
    downloadPath: $(System.ArtifactsDirectory)/databricks

- script: |
    ls $(System.ArtifactsDirectory)/databricks
  displayName: 'List Downloaded Artifacts'

- script: |
    BRANCH_NAME=$(echo $(branchName) | awk -F/ '{print $NF}')
    FOLDER=/$(FolderName)
    echo "Folder to delete: $FOLDER"
    databricks workspace rm $FOLDER --recursive
  displayName: 'Delete Old Release'

- script: |
    BRANCH_NAME=$(echo $(branchName) | awk -F/ '{print $NF}')
    FOLDER=/$(FolderName)
    echo "Folder for new release: $FOLDER"
    databricks workspace import_dir $(System.ArtifactsDirectory)/databricks $FOLDER --exclude-hidden-files
  displayName: 'Deploy New Release'

```

3. **Version Control and Pull Requests**: If changes are made to the feature branch in the Azure DevOps repository, a notification is triggered for a pull request to the main branch. Once the pull request is reviewed and merged, the changes are incorporated into the main branch, and the feature branch is deleted.



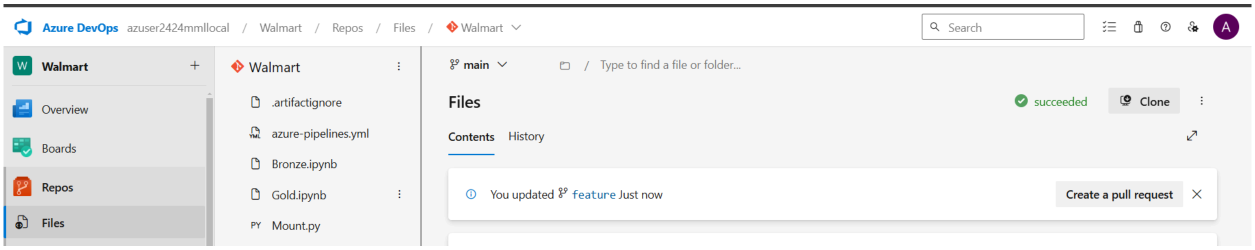
☑ Bronze.ipynb

update

Description (optional)

Note: This notebook may have outputs, but they are not included because the workspace administrator disabled committing outputs.

```
Cmd 14
+ print("Done")
```


Azure DevOps azuser2424mmllocal / Walmart / Repos / Files / Walmart
Search
Walmart
Overview
Boards
Repos
Files
main
Type to find a file or folder...
succeeded
Clone
Contents
History

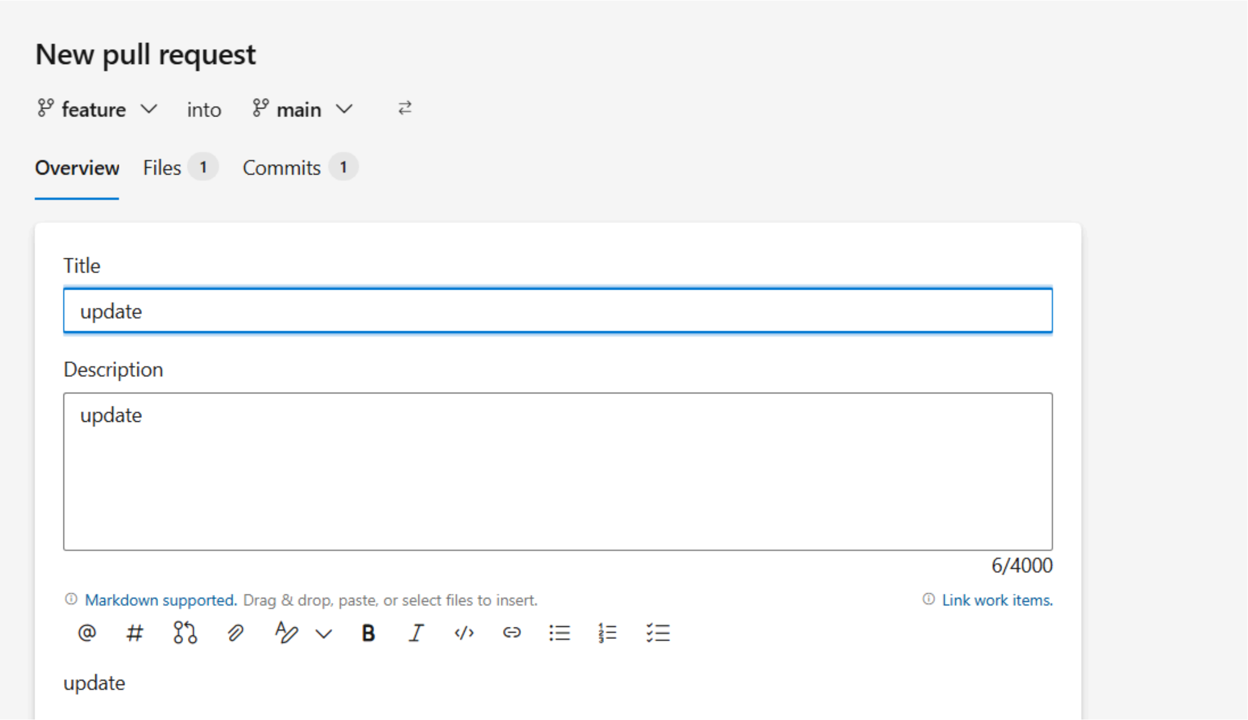
Walmart

- .artifactignore
- azure-pipelines.yml
- Bronze.ipynb
- Gold.ipynb
- Mount.py

Files

You updated feature Just now

Create a pull request


Overview
Files 1
Commits 1
Markdown supported. Drag & drop, paste, or select files to insert.
Link work items.

New pull request

feature into main

Title

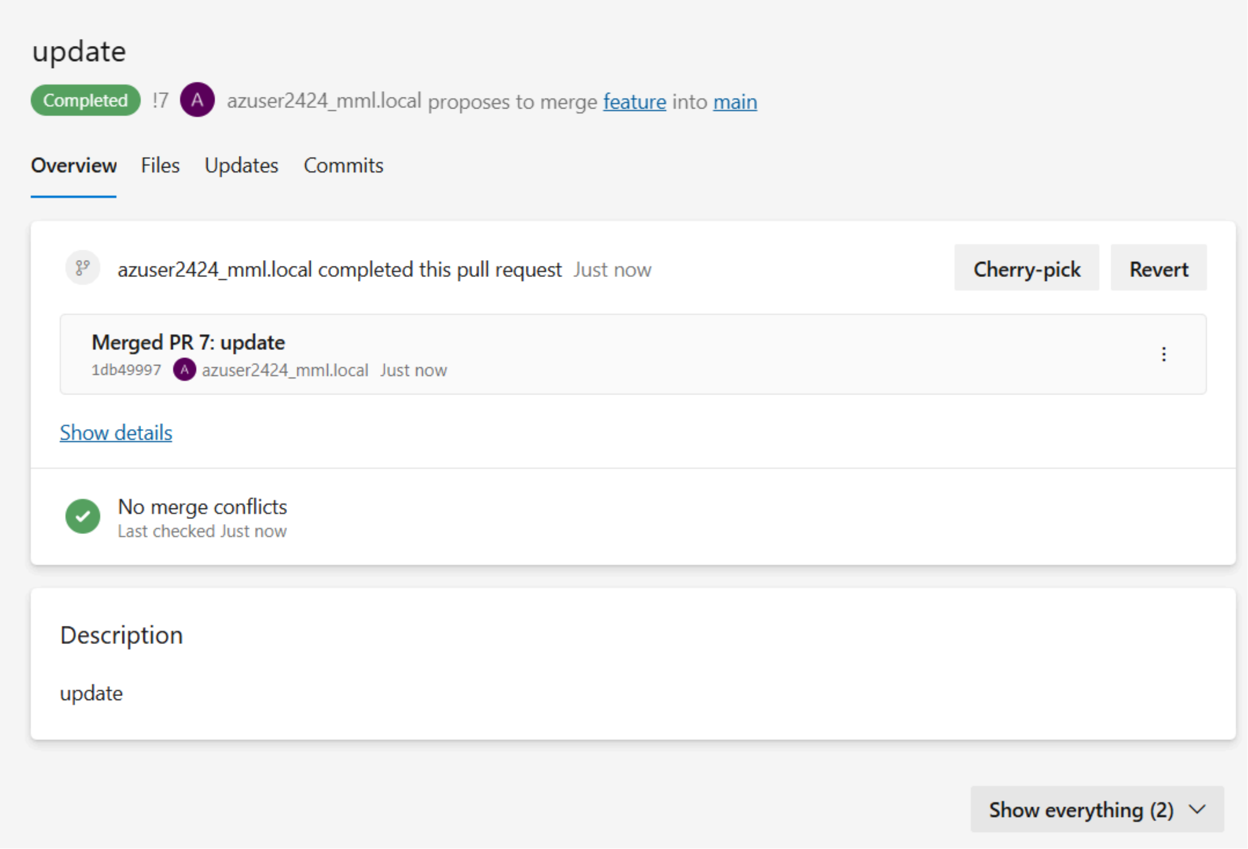
update

Description

update

6/4000

update



update

Completed !7 A azuser2424_mml.local proposes to merge feature into main

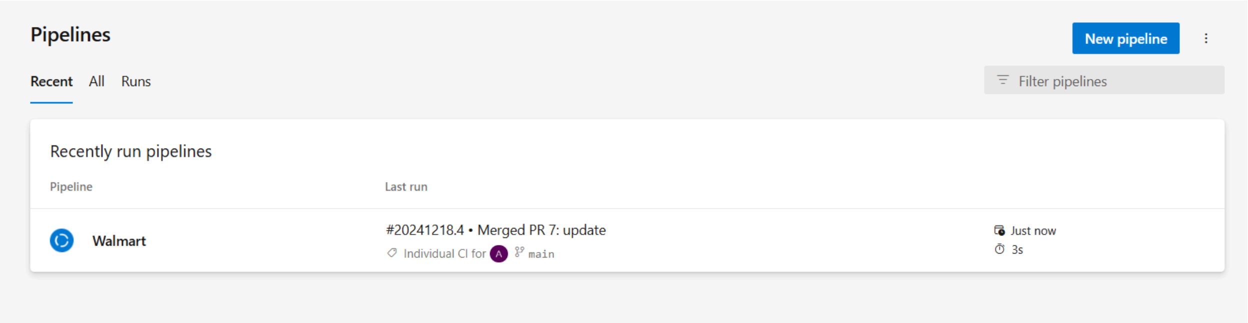
azuser2424_mml.local completed this pull request Just now

Merged PR 7: update
1db49997 A azuser2424_mml.local Just now

No merge conflicts
Last checked Just now

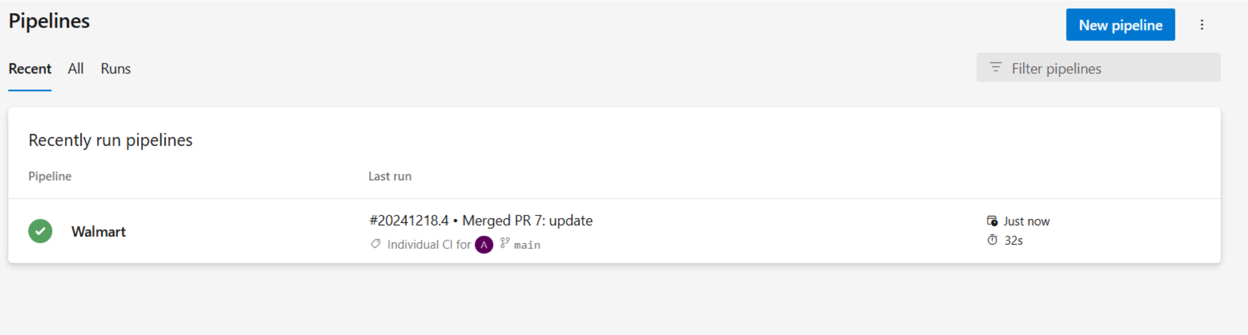
Description

update



Recently run pipelines

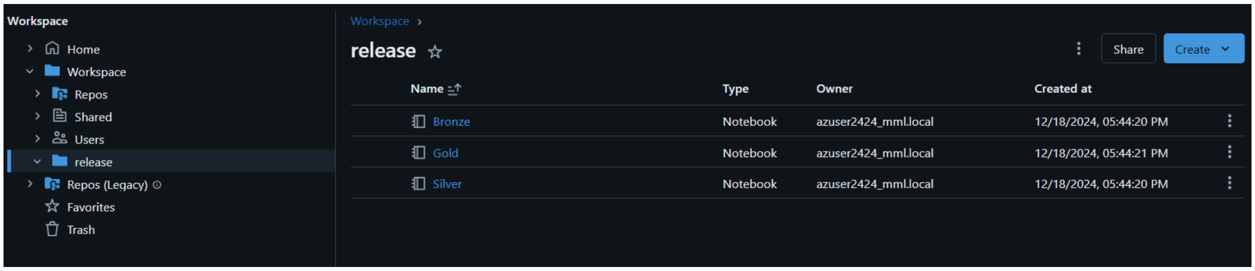
Pipeline	Last run	
Walmart	#20241218.4 • Merged PR 7: update Individual CI for A main	Just now 3s



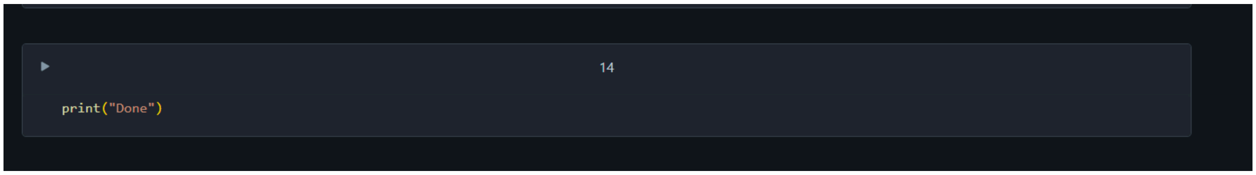
Recently run pipelines

Pipeline	Last run	
✓ Walmart	#20241218.4 • Merged PR 7: update Individual CI for A main	Just now 32s

4. **File Upload and Release Folder**: The pipeline will take all the updated files from the main branch and upload them to a folder named **release**, located within my Azure Databricks workspace. This ensures that the latest versions of the notebooks are available in a dedicated folder for execution.

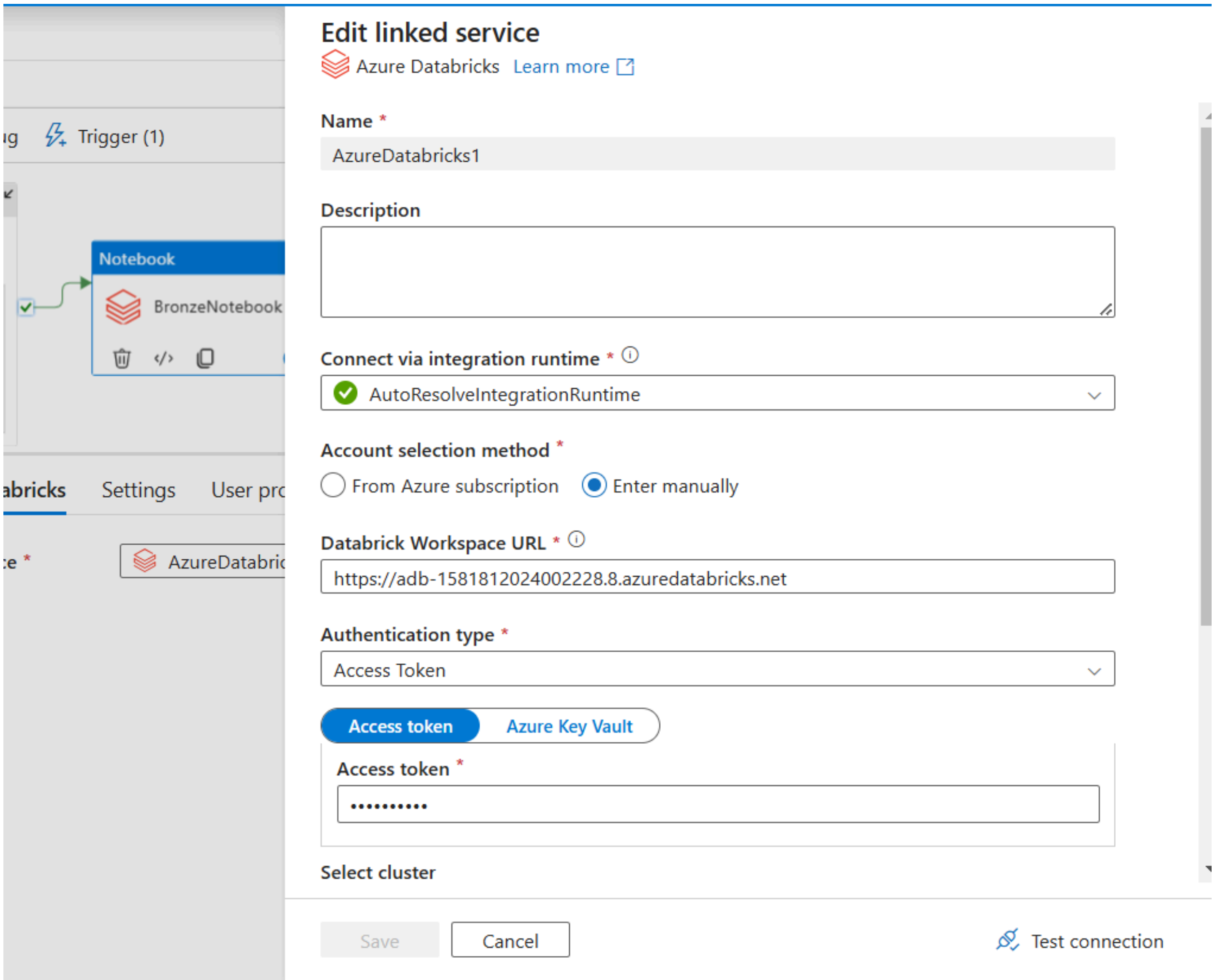


Name	Type	Owner	Created at
Bronze	Notebook	azuser2424_mml.local	12/18/2024, 05:44:20 PM
Gold	Notebook	azuser2424_mml.local	12/18/2024, 05:44:21 PM
Silver	Notebook	azuser2424_mml.local	12/18/2024, 05:44:20 PM



```
print("Done")
```

5. **Running Notebooks via ADF Pipeline**: From the release folder in Databricks, I use Azure Data Factory (ADF) to trigger the execution of the notebooks based on scheduled triggers. This enables automated, timely execution of the data processing and analysis pipelines.



Edit linked service

Azure Databricks Learn more

Name *

AzureDatabricks1

Description

Connect via integration runtime *

AutoResolveIntegrationRuntime

Account selection method *

○ From Azure subscription ● Enter manually

Databrick Workspace URL *

https://adb-1581812024002228.8.azuredatabricks.net

Authentication type *

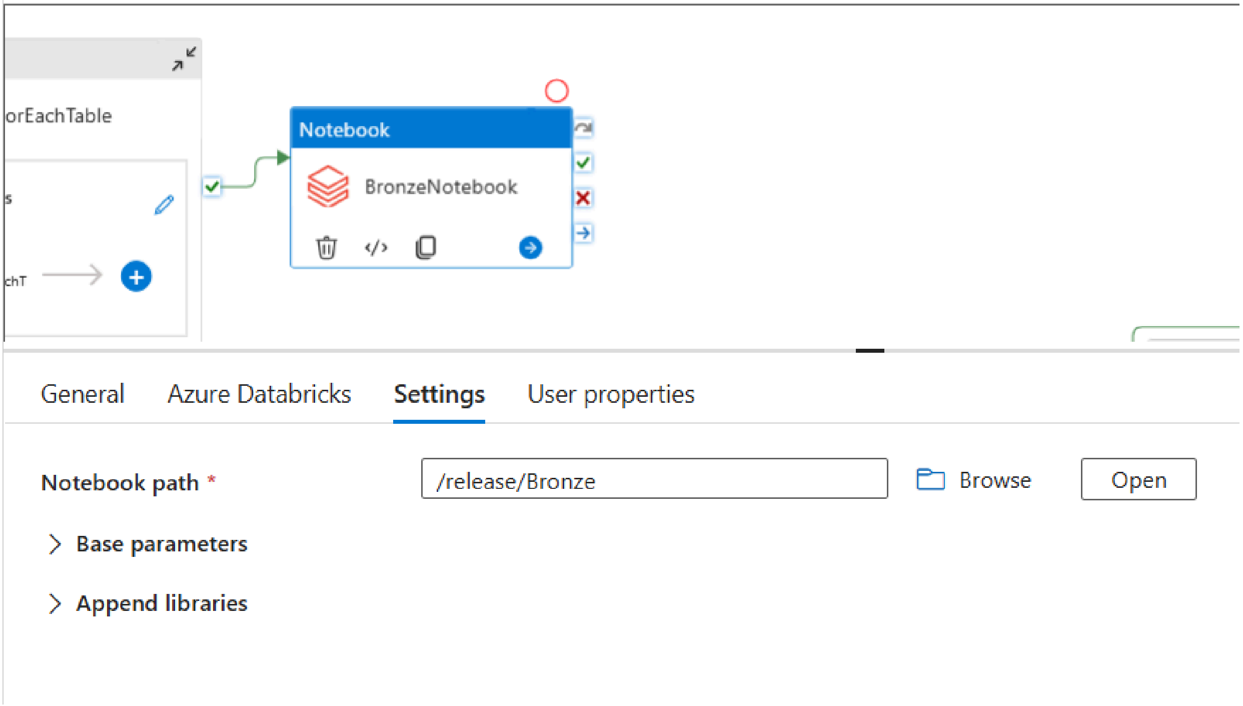
Access Token

Access token | Azure Key Vault

Access token *

••••••••••

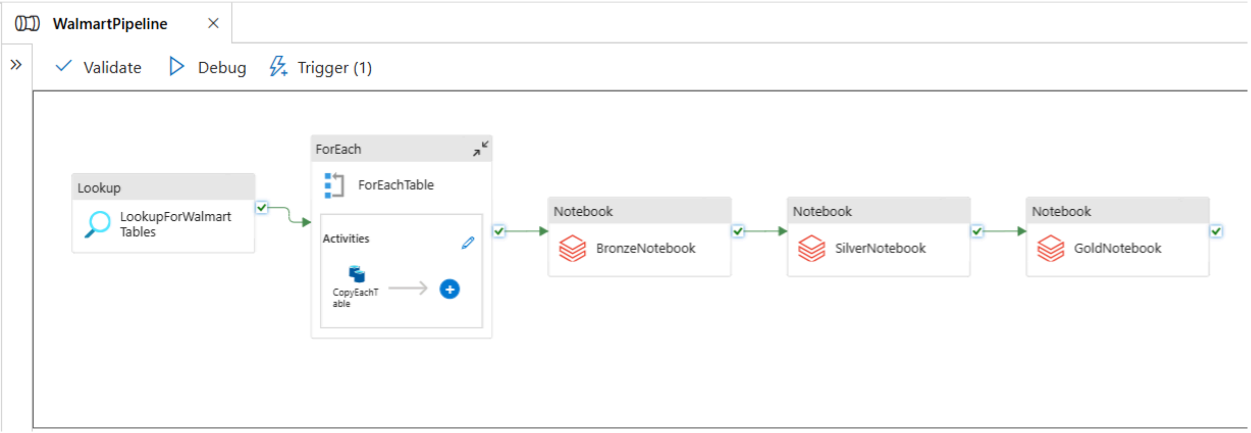
Select cluster

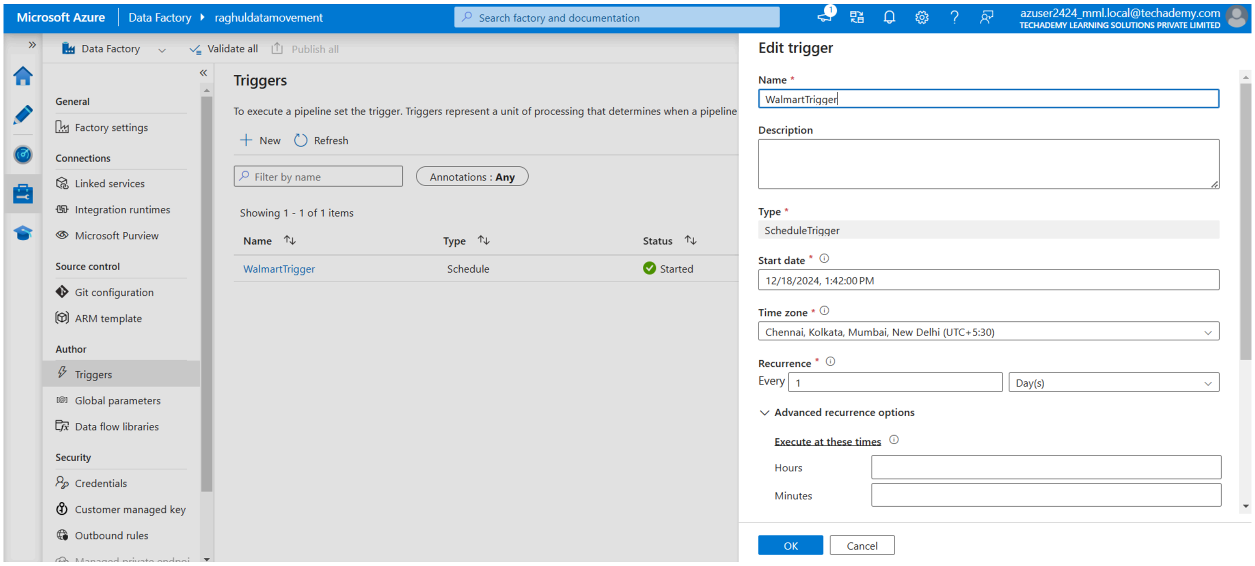


Notebook path * /release/Bronze

Base parameters

Append libraries





Triggers

To execute a pipeline set the trigger. Triggers represent a unit of processing that determines when a pipeline

Showing 1 - 1 of 1 items

Name	Type	Status
WalmartTrigger	Schedule	✓ Started

Edit trigger

Name * WalmartTrigger

Description

Type * ScheduleTrigger

Start date * 12/18/2024, 1:42:00 PM

Time zone * Chennai, Kolkata, Mumbai, New Delhi (UTC+5:30)

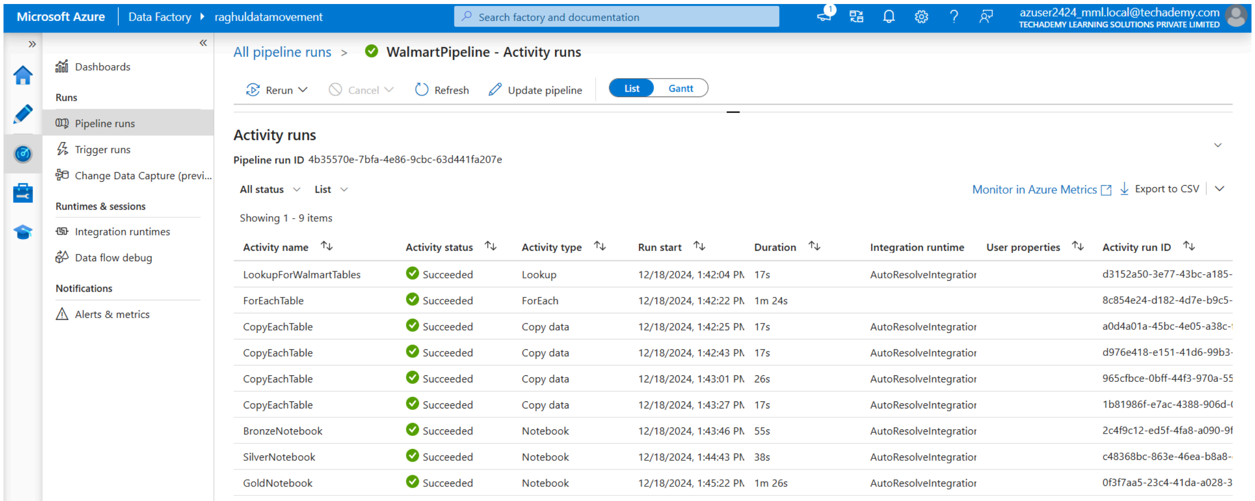
Recurrence * Every 1 Day(s)

Advanced recurrence options

Execute at these times

Hours

Minutes



All pipeline runs > ✓ WalmartPipeline - Activity runs

Activity runs

Pipeline run ID 4b35570e-7bfa-4e86-9cbc-63d441fa207e

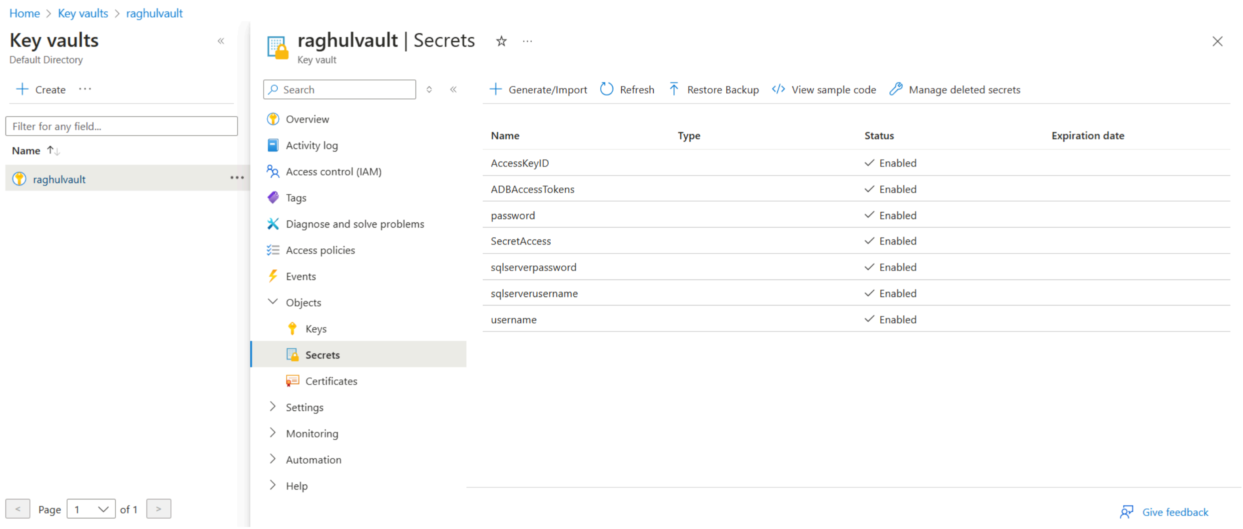
Showing 1 - 9 items

Activity name	Activity status	Activity type	Run start	Duration	Integration runtime	User properties	Activity run ID
LookupForWalmartTables	✓ Succeeded	Lookup	12/18/2024, 1:42:04 PM	17s	AutoResolveIntegratior		d3152a50-3e77-43bc-a185-
ForEachTable	✓ Succeeded	ForEach	12/18/2024, 1:42:22 PM	1m 24s			8c854e24-d182-4d7e-b9c5-
CopyEachTable	✓ Succeeded	Copy data	12/18/2024, 1:42:25 PM	17s	AutoResolveIntegratior		a0d4a01a-45bc-4e05-a38c-
CopyEachTable	✓ Succeeded	Copy data	12/18/2024, 1:42:43 PM	17s	AutoResolveIntegratior		d976e418-e151-41d6-99b3-
CopyEachTable	✓ Succeeded	Copy data	12/18/2024, 1:43:01 PM	26s	AutoResolveIntegratior		965cfbce-0bff-44f3-970a-55
CopyEachTable	✓ Succeeded	Copy data	12/18/2024, 1:43:27 PM	17s	AutoResolveIntegratior		1b81986f-e7ac-4388-906d-
BronzeNotebook	✓ Succeeded	Notebook	12/18/2024, 1:43:46 PM	55s	AutoResolveIntegratior		2c4f9c12-ed5f-4fa8-a090-9f
SilverNotebook	✓ Succeeded	Notebook	12/18/2024, 1:44:43 PM	38s	AutoResolveIntegratior		c48368bc-863e-46ea-b8a8-
GoldNotebook	✓ Succeeded	Notebook	12/18/2024, 1:45:22 PM	1m 26s	AutoResolveIntegratior		0f3f7aa5-23c4-41da-a028-3

By implementing this CI/CD process, I ensured that updates to the project notebooks are automatically integrated and deployed, streamlining the development and deployment process while maintaining version control and automated execution.

Security & Governance

For **Security and Governance** in my project, I utilized **Azure Key Vault** to securely store and manage sensitive information, such as API keys, database credentials, and other secrets. This ensures that sensitive data is protected and access is restricted to authorized users and services.



Name	Type	Status	Expiration date
AccessKeyID		✓ Enabled	
ADBAccessTokens		✓ Enabled	
password		✓ Enabled	
SecretAccess		✓ Enabled	
sqlserverpassword		✓ Enabled	
sqlserverusername		✓ Enabled	
username		✓ Enabled	

To enable collaboration and manage access control within the project, I leveraged **Azure Active Directory (AAD)**. I created an **Azure Active Directory group** and added my teammate to this group, granting her the necessary permissions. I provided her with access to the entire resource group, ensuring that both of us could collaborate effectively while maintaining proper access control and security.

This approach not only secured sensitive information but also streamlined the collaboration process by allowing both team members to work seamlessly on the project with appropriate access to resources and the ability to manage the infrastructure securely.

Conclusion

In conclusion, this project successfully integrates various aspects of data management, analytics, and cloud services to provide a comprehensive solution for Walmart’s stock and sales monitoring. The implementation of data pipelines using **Azure Databricks** allowed for seamless processing and analysis of large datasets, transforming raw data into actionable insights. By utilizing the **Bronze, Silver, and Gold** layers, we ensured that the data was cleaned, enriched, and optimized for reporting and decision-making.

The creation of insightful visualizations through **Databricks dashboards** enabled stakeholders to monitor key metrics such as weekly sales, stock depletion rates, reorder trends, and seasonal impacts. These visualizations empowered the team with real-time data to make informed decisions, driving efficient stock management and forecasting.

Furthermore, the integration of **Continuous Integration and Continuous Deployment (CI/CD)** using **Azure DevOps** ensured that any changes to the Databricks notebooks were automatically deployed and reflected in the workspace, streamlining updates and reducing manual interventions. This was complemented by a robust security and governance framework, with the use of **Azure Key Vault** for storing secrets and **Azure Active Directory (AAD)** for managing access and collaboration within the team.

Through the combined use of Azure’s advanced tools, cloud infrastructure, and data analytics capabilities, this project not only addressed Walmart’s immediate needs for stock and sales insights but also laid the foundation for future scalability and optimization in data-driven decision-making processes.

This project has been an excellent opportunity to apply and deepen my skills in cloud computing, data engineering, and project management, while also ensuring the security and governance of sensitive information throughout the process. The successful completion of this project demonstrates the power of cloud technologies and data analytics in transforming business operations and decision-making.