TRAINING PROJECT REPORT

HEXAWARE

E-commerce Product Demand & Sales Analysis

Data Engineering: Batch 4

HANIF MOHAMMED
KATHYAYINI
INDHUJA RAVI
HARSHA

1. Project Overview

In the dynamic landscape of e-commerce, accurate demand forecasting is paramount for optimizing inventory management, supply chain efficiency, and marketing strategies. This project establishes a robust, end-to-end data and MLOps pipeline designed to address these challenges. The primary objective is to ingest raw e-commerce transaction data, transform it into a refined format, apply advanced machine learning models for predicting future demand, and automate the entire model lifecycle from training to deployment. The ultimate goal is to provide actionable insights through accessible visualizations, enabling data-driven decision-making for business stakeholders.

2. Solution Architecture and Workflow

The solution employs a modern data architecture, adhering to the principles of the **medallion architecture** (Bronze, Silver, Gold layers) to ensure data quality, scalability, and accessibility. This is complemented by a robust MLOps framework for continuous integration and continuous delivery (CI/CD) of machine learning models, all orchestrated within the Azure cloud ecosystem.

Architectural Components:

- **Azure Data Lake Storage Gen2 (ADLS Gen2):** Serves as the central data lake, hosting the Bronze, Silver, and Gold layers. Its hierarchical namespace and HDFS compatibility make it ideal for big data analytics.
- **Azure Databricks:** The core compute engine for data ingestion, transformation, and machine learning model development. Its Apache Spark-based environment provides scalable processing capabilities.
- **MLflow:** Integrated within Databricks, MLflow is used for tracking machine learning experiments, managing model versions, and registering models for deployment.
- **Azure Data Factory (ADF):** The orchestration service responsible for scheduling and managing the end-to-end data pipeline, ensuring data flows smoothly between layers and triggering notebook executions.
- **Azure DevOps:** Facilitates the CI/CD pipeline for machine learning models, automating the promotion of models through various stages in the MLflow Model Registry.
- **Azure Data Explorer (ADX):** A fast, fully managed data analytics service optimized for ad-hoc queries and interactive dashboards on large volumes of data, used here for visualizing the final forecasts.

Workflow Overview:

1. **Data Ingestion (Bronze Layer):** Raw e-commerce data is securely brought into ADLS Gen2.
2. **Data Transformation (Silver Layer):** Raw data undergoes cleansing, enrichment, and aggregation, transforming it into a structured, queryable format.

3. **Forecasting & MLflow Tracking (Gold Layer):** Machine learning models consume the Silver layer data to generate demand forecasts. All model training details and artifacts are meticulously tracked in MLflow.
4. **Model Promotion (MLOps CI/CD):** A streamlined process, automated by Azure DevOps, moves validated models from development to production stages.
5. **Pipeline Orchestration:** Azure Data Factory ensures the sequential and scheduled execution of all data processing and model training steps.
6. **Data Visualization:** The final, curated forecasts from the Gold layer are ingested into Azure Data Explorer for interactive analysis and dashboarding, providing business users with actionable insights.

3. Detailed Project Phases

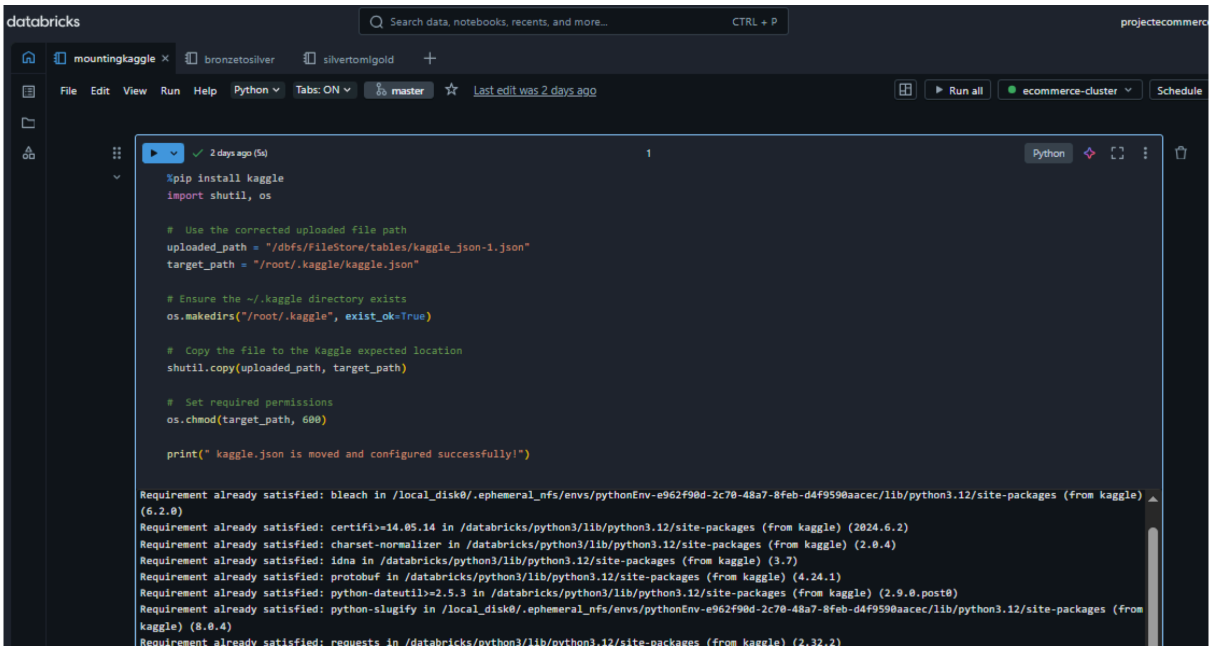
3.1. Data Ingestion (Databricks Notebook: `mountingkaggle`)

This initial phase is responsible for acquiring the raw e-commerce dataset and establishing the **Bronze Layer** in Azure Data Lake Storage Gen2. The Bronze layer serves as a raw data landing zone, preserving the original state of the ingested data for auditability and future reprocessing.

- **Objective:** Securely download the RetailRocket e-commerce dataset from Kaggle and upload its constituent parts (events, item properties, category tree) into designated folders within the `bronzelayer` container in Azure Blob Storage.
- **Dataset Context:** The RetailRocket dataset is a well-known public dataset for e-commerce, containing user behavior data (events like views, add-to-cart, transactions) and item metadata. Its structure (multiple CSV files) necessitates careful handling during ingestion.
- **Key Steps and Technical Details:**
  - **Kaggle API Integration:** The `kaggle` Python library is installed and configured within the Databricks environment. This involves placing the `kaggle.json` API token securely in the `/root/.kaggle/` directory and setting appropriate file permissions (`os.chmod(target_path, 600)`) to ensure secure authentication.
  - **Data Download:** The `kaggle.api.dataset_download_files()` function is used to download the `retailrocket/ecommerce-dataset` to a temporary location (`/dbfs/tmp/`) within the Databricks file system (DBFS). The `unzip=False` parameter is initially used to download the main zip file.
  - **File Extraction:** The downloaded `.zip` file is then extracted using Python's `zipfile` module to a dedicated directory (`/dbfs/tmp/ecommerce_dataset`) to access the individual CSV files.
  - **Azure Storage Configuration:** The Azure Storage account name (`storageecommerce`) and its access key (`storage_account_key`) are configured using `spark.conf.set()`. This grants Spark clusters the necessary permissions to read from and write to ADLS Gen2.

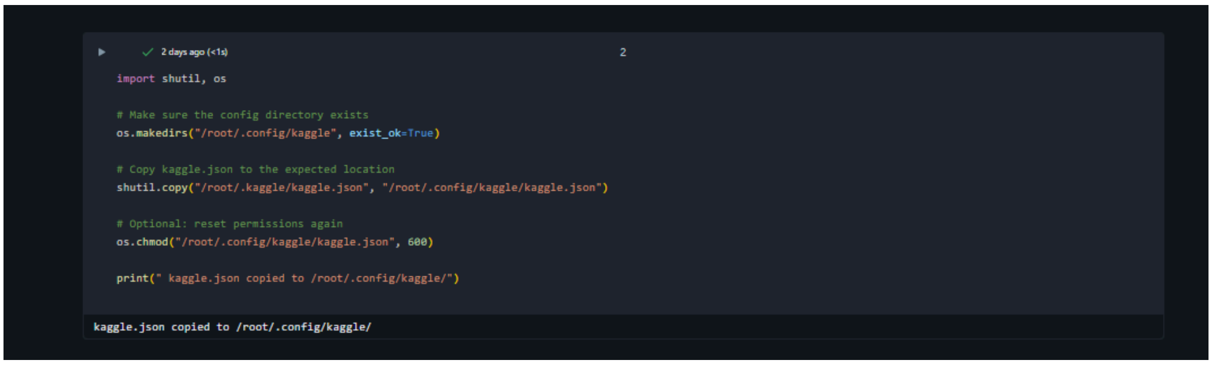
    ○ **Data Upload to Bronze:** A Python loop iterates through the extracted CSV files (`events.csv`, `item_properties_part1.csv`, `item_properties_part2.csv`, `category_tree.csv`). For each file, `dbutils.fs.cp()` is used to copy the data from the temporary DBFS path to its corresponding folder within the `bronzelayer` container in Azure Blob Storage (e.g., `wasbs://bronzelayer@storageecommerce.blob.core.windows.net/events/events.csv`). This ensures data is organized logically within the raw layer.
- **Tools Used:** Azure Databricks, Python (with `kaggle`, `shutil`, `os`, `zipfile` libraries), Azure Blob Storage (Bronze Layer).

Notebook: mountingkaggle



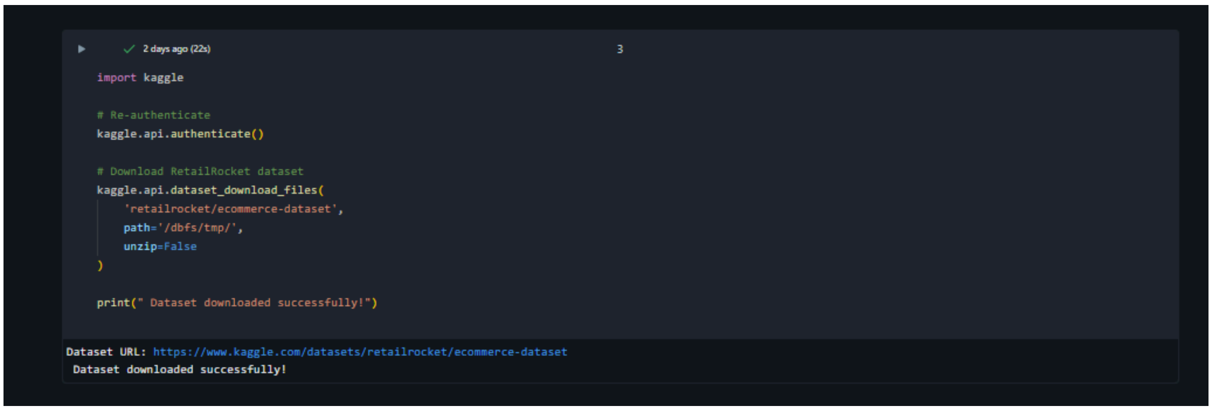
```
%pip install kaggle
import shutil, os

#  Use the corrected uploaded file path
uploaded_path = "/dbfs/FileStore/tables/kaggle_json-1.json"
target_path = "/root/.kaggle/kaggle.json"

# Ensure the ~/.kaggle directory exists
os.makedirs("/root/.kaggle", exist_ok=True)

#  Copy the file to the Kaggle expected location
shutil.copy(uploaded_path, target_path)

#  Set required permissions
os.chmod(target_path, 600)

print(" kaggle.json is moved and configured successfully!")
```

```
Requirement already satisfied: bleach in /local_disk0/.ephemeral_nfs/envs/pythonEnv-e962f90d-2c70-48a7-8feb-d4f9590aacec/lib/python3.12/site-packages (from kaggle)
(6.2.0)
Requirement already satisfied: certifi>=14.05.14 in /databricks/python3/lib/python3.12/site-packages (from kaggle) (2024.6.2)
Requirement already satisfied: charset-normalizer in /databricks/python3/lib/python3.12/site-packages (from kaggle) (2.0.4)
Requirement already satisfied: idna in /databricks/python3/lib/python3.12/site-packages (from kaggle) (3.7)
Requirement already satisfied: protobuf in /databricks/python3/lib/python3.12/site-packages (from kaggle) (4.24.1)
Requirement already satisfied: python-dateutil>=2.5.3 in /databricks/python3/lib/python3.12/site-packages (from kaggle) (2.9.0.post0)
Requirement already satisfied: python-slugify in /local_disk0/.ephemeral_nfs/envs/pythonEnv-e962f90d-2c70-48a7-8feb-d4f9590aacec/lib/python3.12/site-packages (from
kaggle) (8.0.4)
Requirement already satisfied: requests in /databricks/python3/lib/python3.12/site-packages (from kaggle) (2.32.2)
```



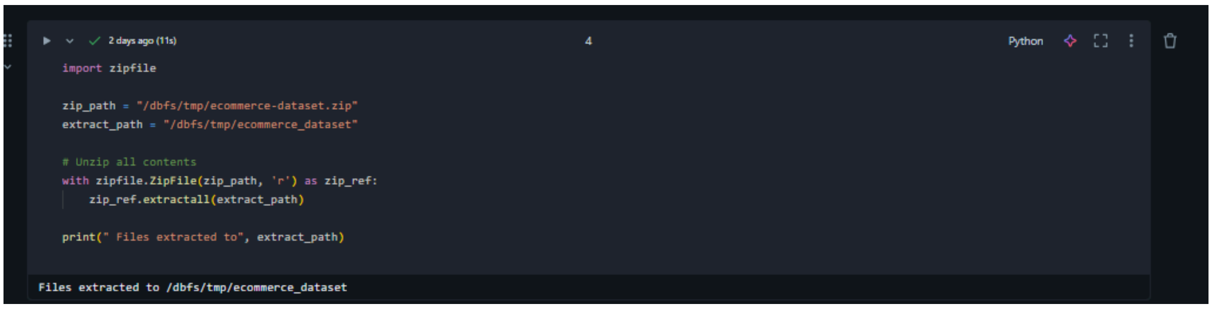
```
import shutil, os

# Make sure the config directory exists
os.makedirs("/root/.config/kaggle", exist_ok=True)

# Copy kaggle.json to the expected location
shutil.copy("/root/.kaggle/kaggle.json", "/root/.config/kaggle/kaggle.json")

# Optional: reset permissions again
os.chmod("/root/.config/kaggle/kaggle.json", 600)

print(" kaggle.json copied to /root/.config/kaggle/")
```

```
kaggle.json copied to /root/.config/kaggle/
```



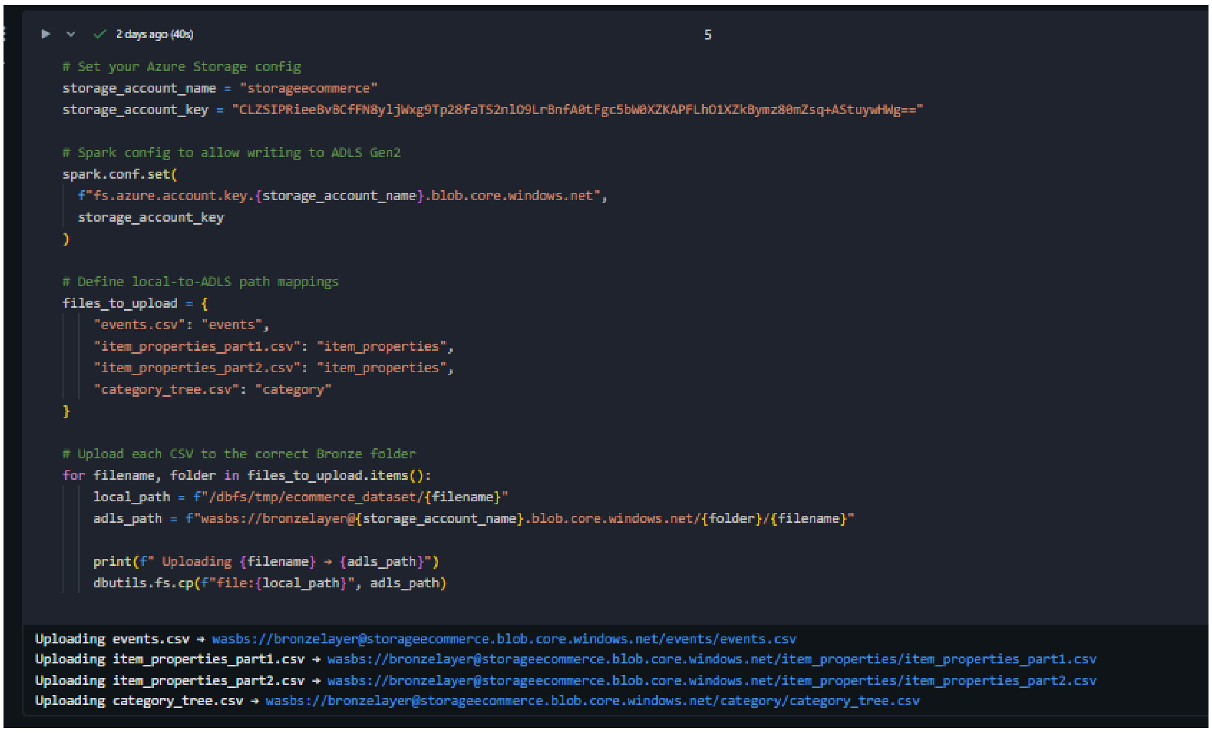
```
import kaggle

# Re-authenticate
kaggle.api.authenticate()

# Download RetailRocket dataset
kaggle.api.dataset_download_files(
    'retailrocket/ecommerce-dataset',
    path='/dbfs/tmp/',
    unzip=False
)

print(" Dataset downloaded successfully!")
```

```
Dataset URL: https://www.kaggle.com/datasets/retailrocket/ecommerce-dataset
 Dataset downloaded successfully!
```



```
import zipfile

zip_path = "/dbfs/tmp/ecommerce-dataset.zip"
extract_path = "/dbfs/tmp/ecommerce_dataset"

# Unzip all contents
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print(" Files extracted to", extract_path)
```

```
Files extracted to /dbfs/tmp/ecommerce_dataset
```



```
# Set your Azure Storage config
storage_account_name = "storageecommerce"
storage_account_key = "CLZSIPRieeBvBCfFN8yljWxg9Tp28faTS2nlO9LrBnfA0tFgc5bW0XZKAPFLhO1XZkBymz80mZsq+AStuywHWg=="

# Spark config to allow writing to ADLS Gen2
spark.conf.set(
  f"fs.azure.account.key.{storage_account_name}.blob.core.windows.net",
  storage_account_key
)

# Define local-to-ADLS path mappings
files_to_upload = {
    "events.csv": "events",
    "item_properties_part1.csv": "item_properties",
    "item_properties_part2.csv": "item_properties",
    "category_tree.csv": "category"
}

# Upload each CSV to the correct Bronze folder
for filename, folder in files_to_upload.items():
    local_path = f"/dbfs/tmp/ecommerce_dataset/{filename}"
    adls_path = f"wasbs://bronzelayer@{storage_account_name}.blob.core.windows.net/{folder}/{filename}"

    print(f" Uploading {filename} → {adls_path}")
    dbutils.fs.cp(f"file:{local_path}", adls_path)
```

```
Uploading events.csv → wasbs://bronzelayer@storageecommerce.blob.core.windows.net/events/events.csv
Uploading item_properties_part1.csv → wasbs://bronzelayer@storageecommerce.blob.core.windows.net/item_properties/item_properties_part1.csv
Uploading item_properties_part2.csv → wasbs://bronzelayer@storageecommerce.blob.core.windows.net/item_properties/item_properties_part2.csv
Uploading category_tree.csv → wasbs://bronzelayer@storageecommerce.blob.core.windows.net/category/category_tree.csv
```

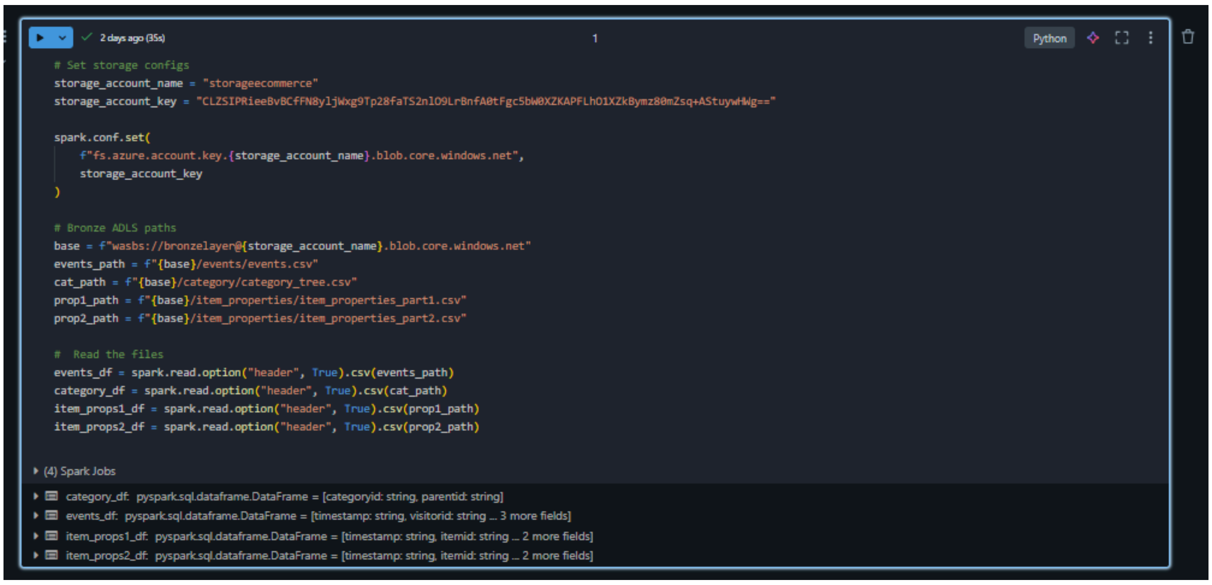
3.2. Data Transformation (Databricks Notebook: `bronzetosilver`)

This phase is critical for transforming the raw, potentially messy data from the Bronze layer into a clean, structured, and query-optimized format in the **Silver Layer**. The Silver layer is designed for enterprise-wide consumption, providing a single source of truth for refined data.

- **Objective:** Process the raw e-commerce events and item properties to derive meaningful features, specifically calculating daily sales counts per item, and store this in a Delta Lake format in the Silver layer.
- **Key Steps and Technical Details:**
  - **Azure Storage Configuration:** Similar to the ingestion notebook, Azure Storage account details are configured for Spark to access the Bronze layer.
  - **Reading Bronze Data:** Raw CSV files from the `bronzelayer` are read into PySpark DataFrames using `spark.read.option("header", True).csv()`. This includes `events.csv`, `category_tree.csv`, and the two parts of `item_properties.csv`.
  - **Data Combination and Deduplication:** The two `item_properties` DataFrames (`item_props1_df`, `item_props2_df`) are combined using `union()`. Duplicate records based on `itemid`, `property`, and `timestamp` are removed using `dropDuplicates()` to ensure data integrity.
  - **Transaction Filtering:** The `events_df` is filtered to isolate only "transaction" events using `col("event") == "transaction"`, as these are directly relevant to sales demand.
  - **Date Conversion:** The `timestamp` column in the `transactions_df` is converted to a `date` type using `withColumn("event_date", to_date("timestamp"))`. This standardizes the date format for subsequent daily aggregations.
  - **Item-Category Mapping Extraction:** From the combined `item_props_df`, records where `property == 'categoryid'` are filtered to extract the `itemid` and its corresponding `categoryid`, creating `item_category_df`. This provides a lookup for item categorization.
  - **Data Joining:**
    - `transactions_df` is joined with `item_category_df` on `itemid` using a `left` join to enrich transaction data with category information.
    - This intermediate DataFrame (`tx_item_df`) is then optionally joined with `category_df` (from `category_tree.csv`) on `categoryid` to get the full category hierarchy, if needed for deeper analysis.
  - **Daily Demand Aggregation:** The core transformation involves grouping the enriched transaction data by `itemid` and `event_date`. Aggregations are performed to calculate:
    - `daily_sales_count`: Using `count("*")` to count the number of transactions for each item on a given day.
    - `categoryid`: Using `max("categoryid")` (assuming a single category ID per item per day after joins).

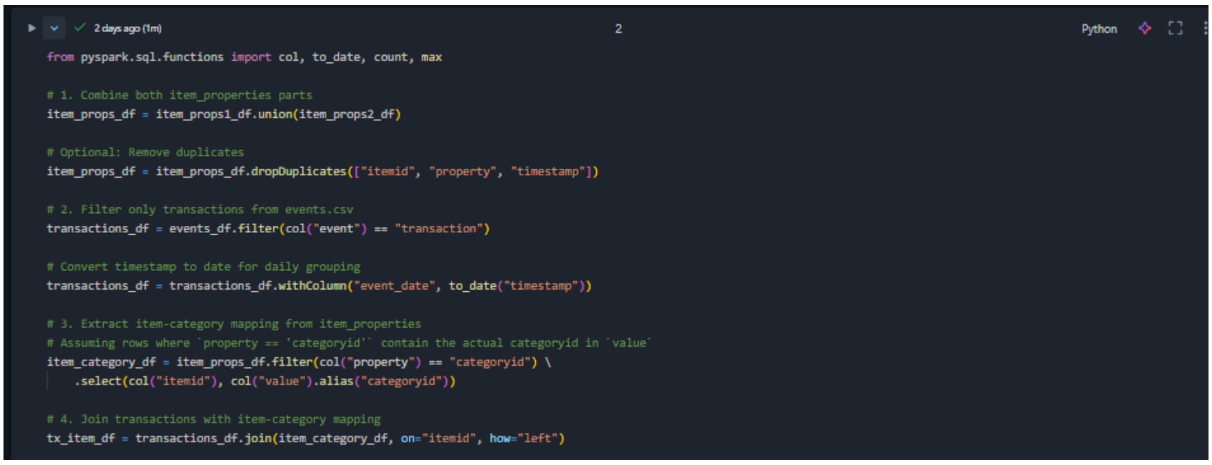
    - **Saving to Silver Layer (Delta Format):** The resulting `daily_demand_df` is saved to the `silverlayer` in Azure Blob Storage using the **Delta Lake format** (`.write.format("delta").mode("overwrite").save(silver_path)`). Delta Lake provides ACID properties, schema enforcement, and versioning, making the Silver layer reliable and performant for downstream analytics.
- **Tools Used:** Azure Databricks, PySpark (with `pyspark.sql.functions`), Pandas, Azure Blob Storage (Silver Layer - Delta Lake).

Notebook: bronzetosilver

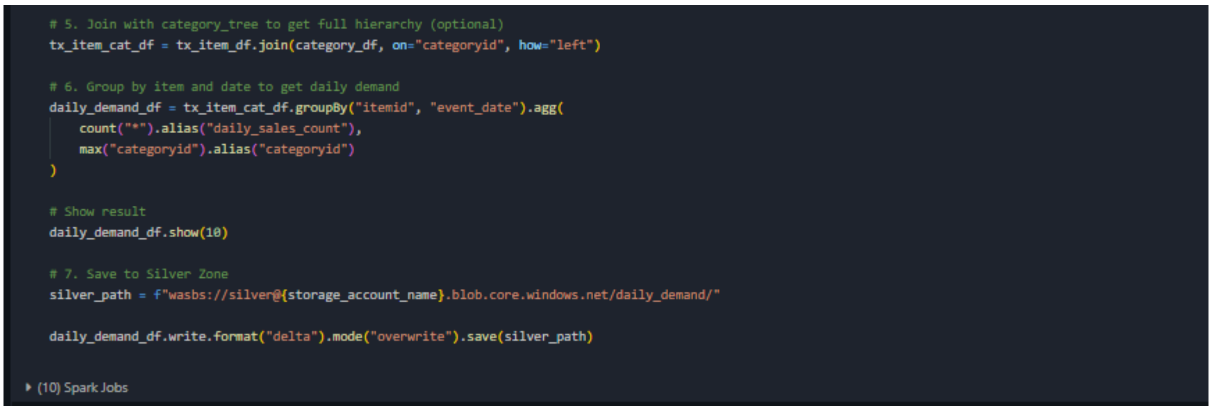


```
# Set storage configs
storage_account_name = "storageecommerce"
storage_account_key = "CLZSIPRieeBvBCfFN8yljWxg9Tp28faTS2nlO9LrBnfA0tFgc5bW0XZKAPFLhO1XZkBymz80mZsq+AStuywHWg=="

spark.conf.set(
    f"fs.azure.account.key.{storage_account_name}.blob.core.windows.net",
    storage_account_key
)

# Bronze ADLS paths
base = f"wasbs://bronzelayer@{storage_account_name}.blob.core.windows.net"
events_path = f"{base}/events/events.csv"
cat_path = f"{base}/category/category_tree.csv"
prop1_path = f"{base}/item_properties/item_properties_part1.csv"
prop2_path = f"{base}/item_properties/item_properties_part2.csv"

#  Read the files
events_df = spark.read.option("header", True).csv(events_path)
category_df = spark.read.option("header", True).csv(cat_path)
item_props1_df = spark.read.option("header", True).csv(prop1_path)
item_props2_df = spark.read.option("header", True).csv(prop2_path)
```

```
(4) Spark Jobs
category_df:  pyspark.sql.dataframe.DataFrame = [categoryid: string, parentid: string]
events_df:  pyspark.sql.dataframe.DataFrame = [timestamp: string, visitorid: string ... 3 more fields]
item_props1_df:  pyspark.sql.dataframe.DataFrame = [timestamp: string, itemid: string ... 2 more fields]
item_props2_df:  pyspark.sql.dataframe.DataFrame = [timestamp: string, itemid: string ... 2 more fields]
```

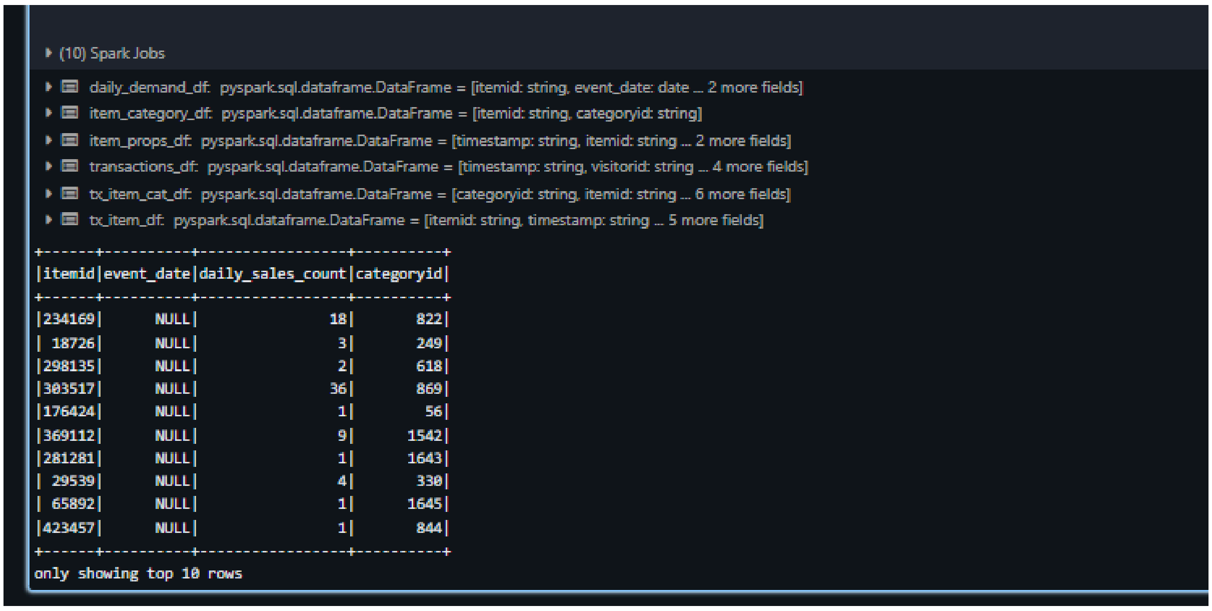


```
from pyspark.sql.functions import col, to_date, count, max

# 1. Combine both item_properties parts
item_props_df = item_props1_df.union(item_props2_df)

# Optional: Remove duplicates
item_props_df = item_props_df.dropDuplicates(["itemid", "property", "timestamp"])

# 2. Filter only transactions from events.csv
transactions_df = events_df.filter(col("event") == "transaction")

# Convert timestamp to date for daily grouping
transactions_df = transactions_df.withColumn("event_date", to_date("timestamp"))

# 3. Extract item-category mapping from item_properties
# Assuming rows where `property == 'categoryid'` contain the actual categoryid in `value`
item_category_df = item_props_df.filter(col("property") == "categoryid") \
    .select(col("itemid"), col("value").alias("categoryid"))

# 4. Join transactions with item-category mapping
tx_item_df = transactions_df.join(item_category_df, on="itemid", how="left")
```



```python
# 5. Join with category_tree to get full hierarchy (optional)
tx_item_cat_df = tx_item_df.join(category_df, on="categoryid", how="left")

# 6. Group by item and date to get daily demand
daily_demand_df = tx_item_cat_df.groupBy("itemid", "event_date").agg(
    count("*").alias("daily_sales_count"),
    max("categoryid").alias("categoryid")
)

# Show result
daily_demand_df.show(10)

# 7. Save to Silver Zone
silver_path = f"wasbs://silver@{storage_account_name}.blob.core.windows.net/daily_demand/"

daily_demand_df.write.format("delta").mode("overwrite").save(silver_path)
```

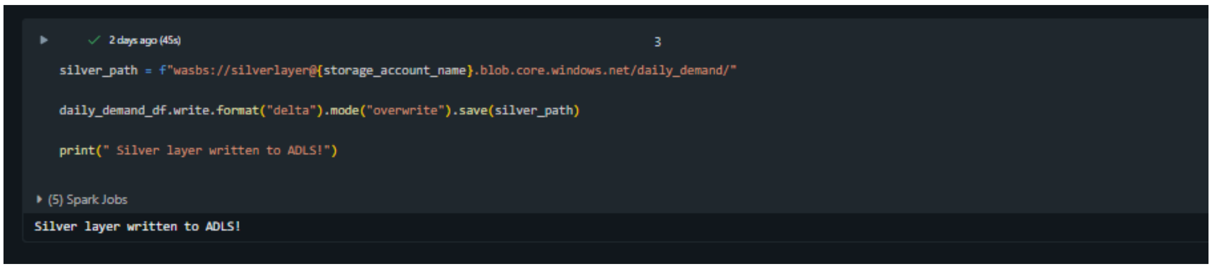
(10) Spark Jobs



(10) Spark Jobs

```
daily_demand_df:  pyspark.sql.dataframe.DataFrame = [itemid: string, event_date: date ... 2 more fields]
item_category_df:  pyspark.sql.dataframe.DataFrame = [itemid: string, categoryid: string]
item_props_df:  pyspark.sql.dataframe.DataFrame = [timestamp: string, itemid: string ... 2 more fields]
transactions_df:  pyspark.sql.dataframe.DataFrame = [timestamp: string, visitorid: string ... 4 more fields]
tx_item_cat_df:  pyspark.sql.dataframe.DataFrame = [categoryid: string, itemid: string ... 6 more fields]
tx_item_df:  pyspark.sql.dataframe.DataFrame = [itemid: string, timestamp: string ... 5 more fields]
```

```
+------+----------+-----------------+----------+
|itemid|event_date|daily_sales_count|categoryid|
+------+----------+-----------------+----------+
|234169|      NULL|               18|       822|
| 18726|      NULL|                3|       249|
|298135|      NULL|                2|       618|
|303517|      NULL|               36|       869|
|176424|      NULL|                1|        56|
|369112|      NULL|                9|      1542|
|281281|      NULL|                1|      1643|
| 29539|      NULL|                4|       330|
| 65892|      NULL|                1|      1645|
|423457|      NULL|                1|       844|
+------+----------+-----------------+----------+
only showing top 10 rows
```



✓ 2 days ago (45s) 3

```python
silver_path = f"wasbs://silverlayer@{storage_account_name}.blob.core.windows.net/daily_demand/"

daily_demand_df.write.format("delta").mode("overwrite").save(silver_path)

print(" Silver layer written to ADLS!")
```

(5) Spark Jobs

```
Silver layer written to ADLS!
```

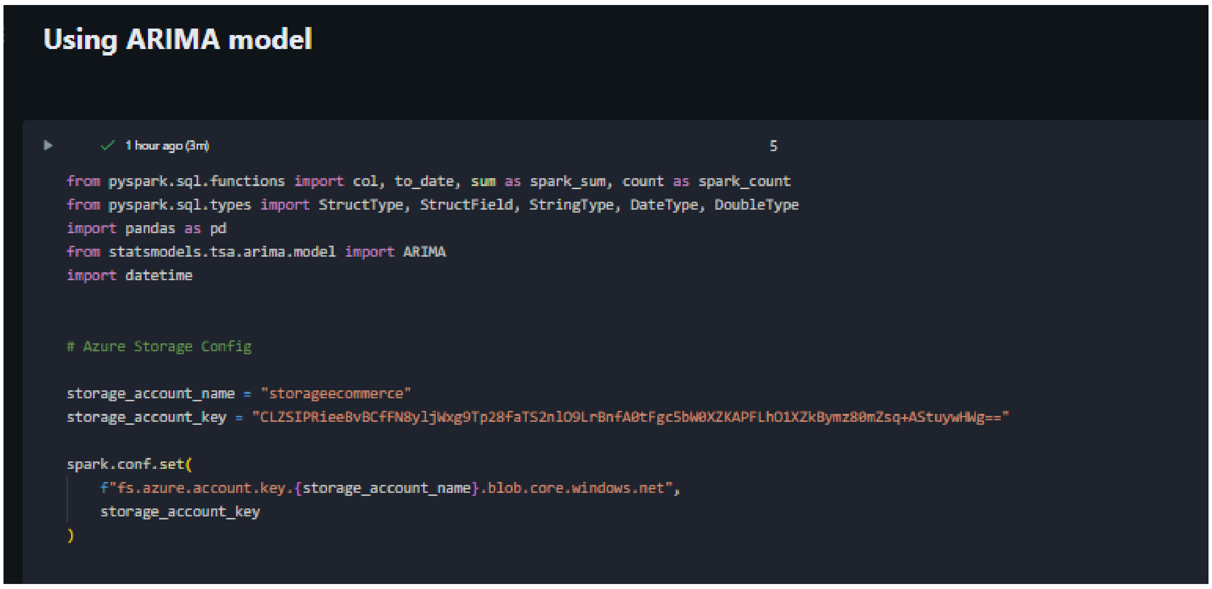
3.3. Forecasting Model Development (Databricks Notebook: `silvertomlgold`)

This phase focuses on applying machine learning models to generate demand forecasts and meticulously tracking these models using MLflow. The output is stored in the **Gold Layer**, which is optimized for direct consumption by business intelligence tools and applications.

- **Objective:** Generate 7-day demand forecasts for the top 5 most active e-commerce items using the ARIMA model (with a fallback/alternative of Simple Moving Average) and store these forecasts in the Gold Layer, while ensuring all model training details are logged with MLflow.
- **Key Steps and Technical Details:**
  - **Configuration and Data Loading:** Azure Storage configuration and paths for Silver and Gold layers are defined. The `daily_demand` data is loaded from the Silver layer (Delta format), selecting `itemid`, `event_date`, and `daily_sales_count`.
  - **Data Preparation for Forecasting:**
    - The loaded data is grouped by `event_date` and `itemid` to get `daily_sales_count` (y), and `event_date` is aliased as `ds`. `itemid` is aliased as `group` to align with common time series forecasting library conventions.
    - `item_day_counts` is calculated to identify the top 5 most active items (those with the most recorded sales days). This ensures forecasting focuses on relevant, frequently transacted items.
    - The `daily_df` is then filtered to include only these `top_items`.
  - **ARIMA Forecast Function (`arima_forecast`):** This is a Pandas UDF (`applyInPandas`) designed to run the ARIMA model for each `group` (itemid).
    - **Input Handling:** It receives a Pandas DataFrame (`pdf`) for a specific item. The `ds` column is explicitly converted to `pd.to_datetime()` and then set as the index for time series operations, ensuring `statsmodels` compatibility.
    - **Data Sufficiency Check:** A crucial check `if len(ts) < 2:` is included. ARIMA models require a minimum number of data points (at least 2, but more for meaningful forecasts) to fit. If insufficient data, the function gracefully returns a DataFrame with NaN values for `yhat`, preventing pipeline failures and indicating missing forecasts.
    - **ARIMA Model Training:** An **ARIMA(1,1,1)** model is instantiated and fitted (`model.fit()`) to the time series (`ts`). The (1,1,1) order implies:
      - **p=1 (AR order):** Uses one lagged observation in the model.
      - **d=1 (Differencing order):** The time series is differenced once to make it stationary.
      - **q=1 (MA order):** Uses one lagged forecast error in the model.
      - This order is a common starting point for many time series.

        - **Forecast Generation:** `model_fit.forecast(steps=7)` generates predictions for the next 7 days.
        - **Future Dates Generation:** `future_dates` are generated as Pandas Timestamps, extending from the last date in the historical series.
        - **Result DataFrame Construction:** A Pandas DataFrame `result` is created with `group`, `ds` (future dates), and `yhat` (forecasted values).
        - **MLflow Tracking:** This is a core MLOps component.
            - `with mlflow.start_run(run_name=f"ARIMA_{group}")`: creates a new MLflow run for each item's forecast.
            - `mlflow.log_param()`: Logs model type ("ARIMA"), order ("(1,1,1)"), and the specific `group` (item ID).
            - `mlflow.log_metric("train_count", len(ts))`: Records the number of data points used for training.
            - `mlflow.sklearn.log_model(model_fit, f"model_arima_{group}")`: Logs the trained `model_fit` object as an MLflow artifact, making it reproducible and ready for deployment.
        - **Error Handling:** A `try-except` block catches any exceptions during ARIMA fitting or forecasting, printing an error message and returning `NaN` forecasts for that specific item.
    - **Application of ARIMA:** The `arima_forecast` Pandas UDF is applied to the `filtered_df` grouped by `group (itemid)` using `applyInPandas()`. This efficiently distributes the forecasting task across the Spark cluster.
    - **Final Date Casting:** The `forecast_df` undergoes a final cast of the `ds` column to `DateType` (`col("ds").cast("date")`) to ensure compatibility with Spark's date handling, resolving previous `NULL` issues by ensuring the Pandas Timestamps are correctly interpreted.
    - **Data Saving to Gold Layer:** The final `forecast_df` containing the 7-day forecasts is saved to the `goldlayer` in Azure Blob Storage in two formats:
        - **Delta format:** (`.write.format("delta").mode("overwrite").save(gold_path)`) for analytical consumption and future pipeline stages.
        - **CSV format:** (`.write.option("header", "true").mode("overwrite").csv(gold_csv_path)`) for easy human readability and consumption by tools that prefer flat files.
    - **Simple Moving Average Method (Baseline):** As an alternative or baseline, a simple moving average forecasting method is also implemented. This calculates the average daily sales for each top item and uses this average as the forecast for the next 7 days. This provides a quick, interpretable benchmark.
- **Tools Used:** Azure Databricks, PySpark, Pandas, `statsmodels.tsa.arima.model.ARIMA`, MLflow, Azure Blob Storage (Gold Layer - Delta and CSV).

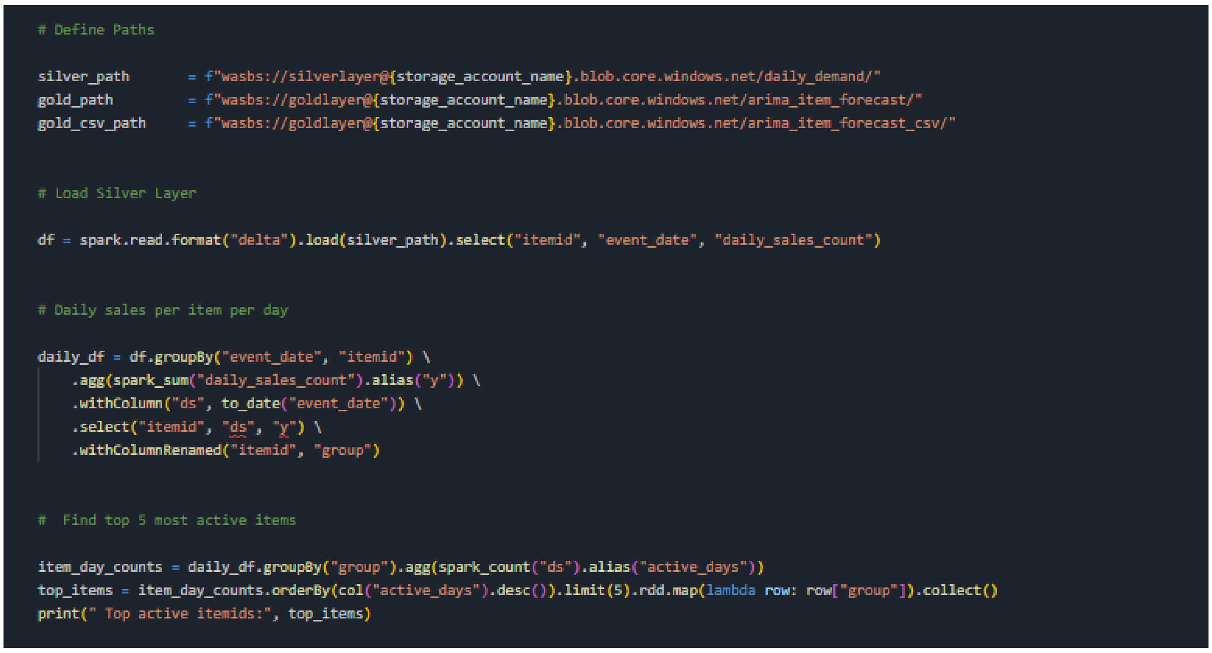
Notebook: silvertomlgold



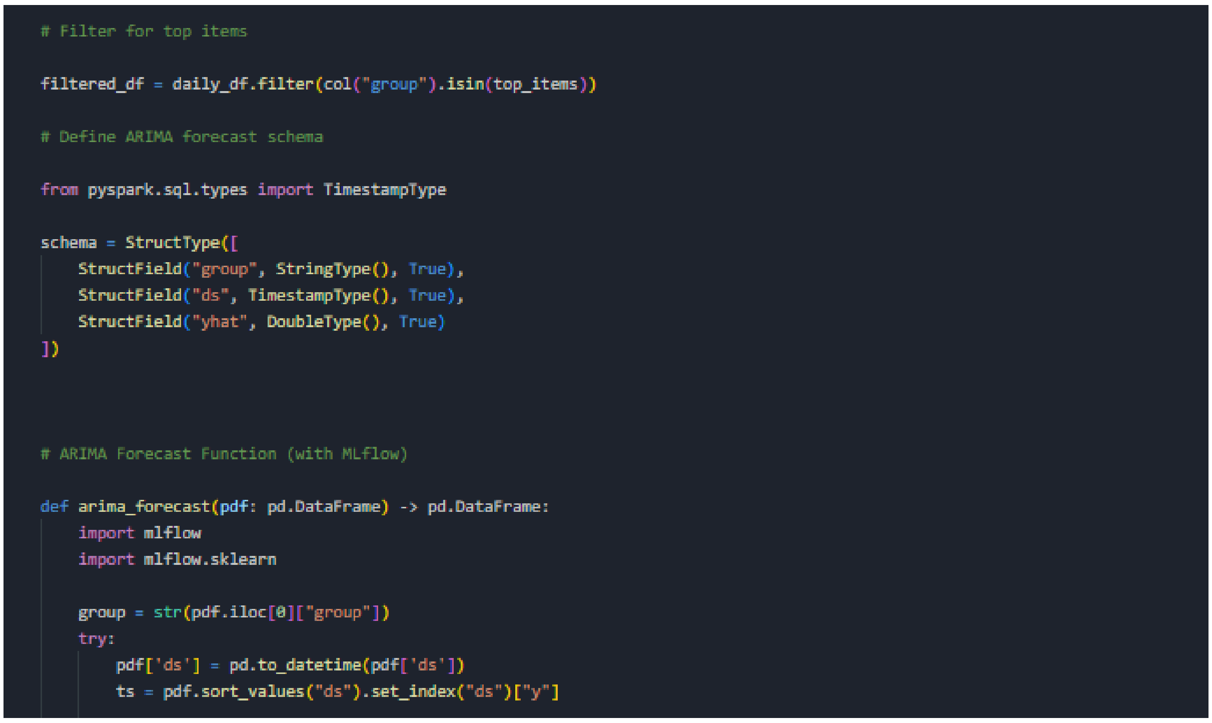
Using ARIMA model

1 hour ago (3m)   5

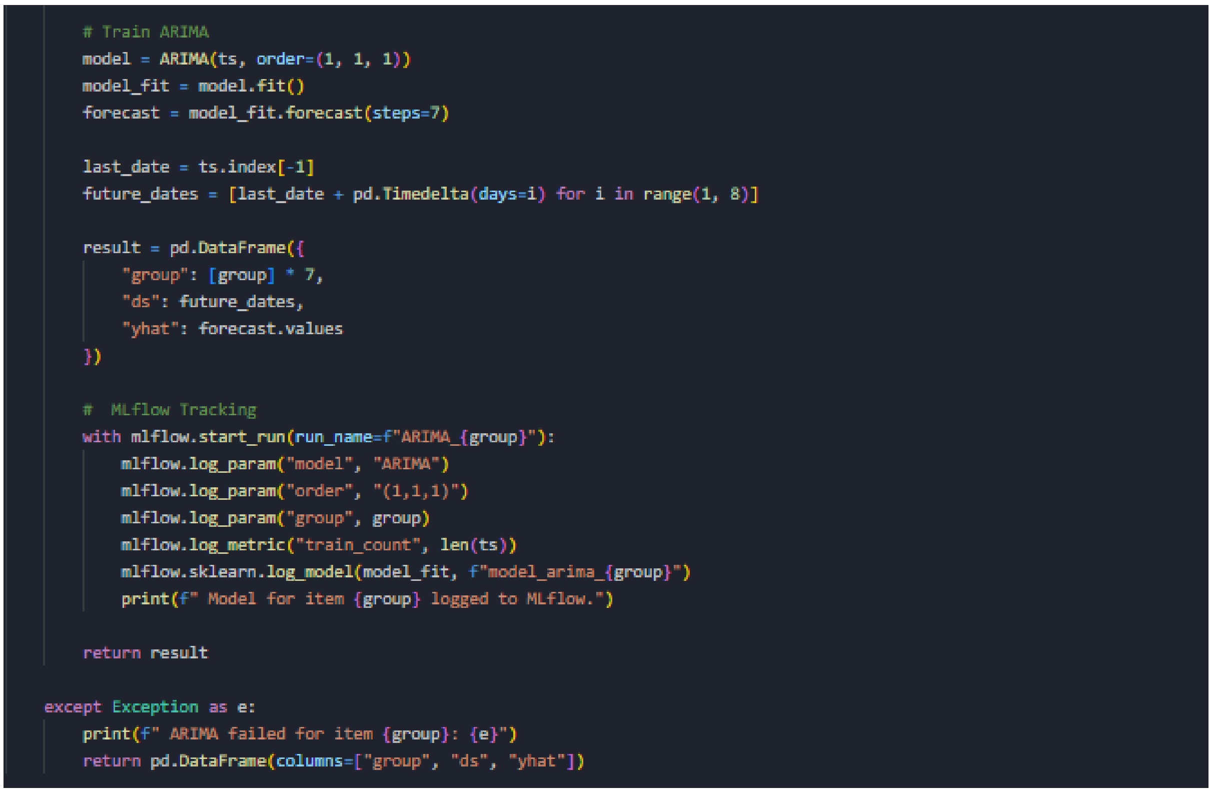
```python
from pyspark.sql.functions import col, to_date, sum as spark_sum, count as spark_count
from pyspark.sql.types import StructType, StructField, StringType, DateType, DoubleType
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
import datetime


# Azure Storage Config

storage_account_name = "storageecommerce"
storage_account_key = "CLZSIPRieeBvBCfFN8yljWxg9Tp28faTS2nlO9LrBnfA0tFgc5bW0XZKAPFLhO1XZkBymz80mZsq+AStuywHWg=="

spark.conf.set(
    f"fs.azure.account.key.{storage_account_name}.blob.core.windows.net",
    storage_account_key
)
```



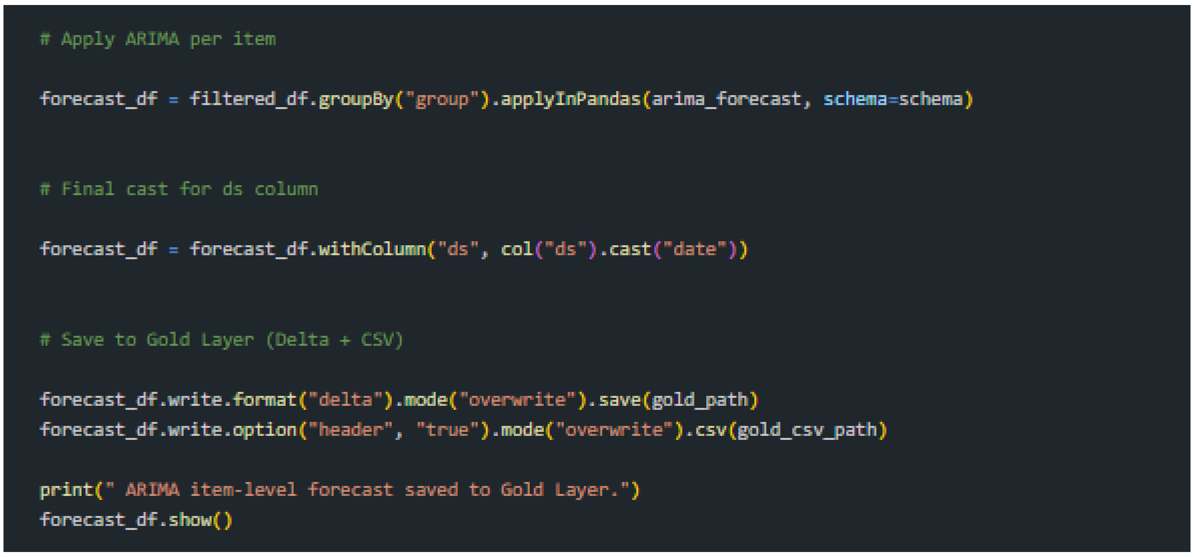
```python
# Define Paths

silver_path      = f"wasbs://silverlayer@{storage_account_name}.blob.core.windows.net/daily_demand/"
gold_path        = f"wasbs://goldlayer@{storage_account_name}.blob.core.windows.net/arima_item_forecast/"
gold_csv_path    = f"wasbs://goldlayer@{storage_account_name}.blob.core.windows.net/arima_item_forecast_csv/"


# Load Silver Layer

df = spark.read.format("delta").load(silver_path).select("itemid", "event_date", "daily_sales_count")


# Daily sales per item per day

daily_df = df.groupBy("event_date", "itemid") \
    .agg(spark_sum("daily_sales_count").alias("y")) \
    .withColumn("ds", to_date("event_date")) \
    .select("itemid", "ds", "y") \
    .withColumnRenamed("itemid", "group")


#  Find top 5 most active items

item_day_counts = daily_df.groupBy("group").agg(spark_count("ds").alias("active_days"))
top_items = item_day_counts.orderBy(col("active_days").desc()).limit(5).rdd.map(lambda row: row["group"]).collect()
print(" Top active itemids:", top_items)
```



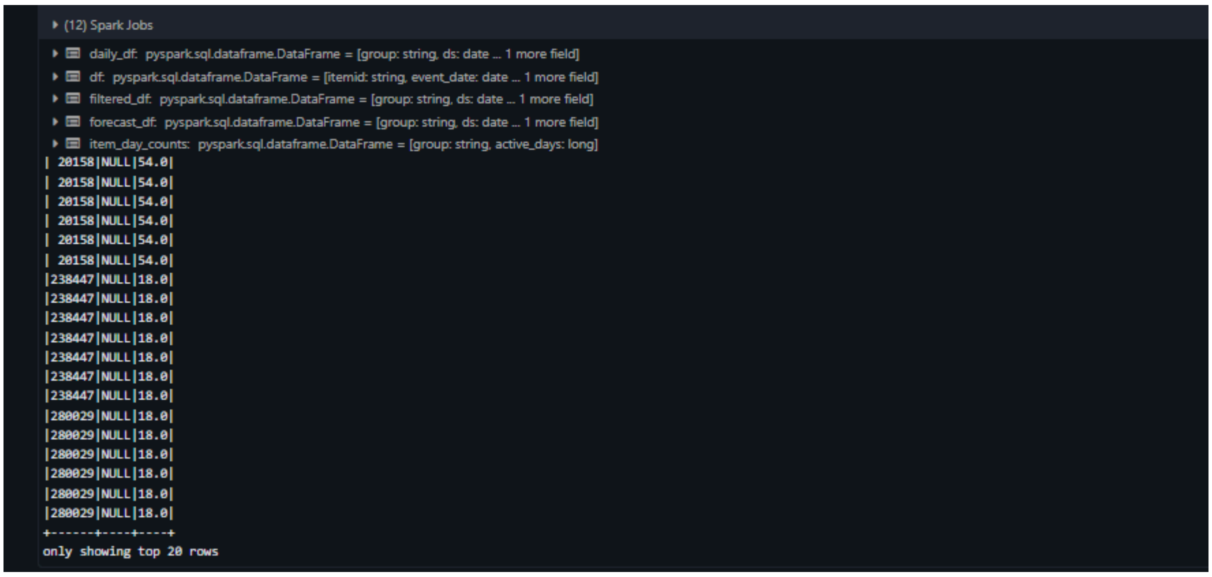
```python
# Filter for top items

filtered_df = daily_df.filter(col("group").isin(top_items))

# Define ARIMA forecast schema

from pyspark.sql.types import TimestampType

schema = StructType([
    StructField("group", StringType(), True),
    StructField("ds", TimestampType(), True),
    StructField("yhat", DoubleType(), True)
])



# ARIMA Forecast Function (with MLflow)

def arima_forecast(pdf: pd.DataFrame) -> pd.DataFrame:
    import mlflow
    import mlflow.sklearn

    group = str(pdf.iloc[0]["group"])
    try:
        pdf['ds'] = pd.to_datetime(pdf['ds'])
        ts = pdf.sort_values("ds").set_index("ds")["y"]
```



```python
        # Train ARIMA
        model = ARIMA(ts, order=(1, 1, 1))
        model_fit = model.fit()
        forecast = model_fit.forecast(steps=7)

        last_date = ts.index[-1]
        future_dates = [last_date + pd.Timedelta(days=i) for i in range(1, 8)]

        result = pd.DataFrame({
            "group": [group] * 7,
            "ds": future_dates,
            "yhat": forecast.values
        })

        #  MLflow Tracking
        with mlflow.start_run(run_name=f"ARIMA_{group}"):
            mlflow.log_param("model", "ARIMA")
            mlflow.log_param("order", "(1,1,1)")
            mlflow.log_param("group", group)
            mlflow.log_metric("train_count", len(ts))
            mlflow.sklearn.log_model(model_fit, f"model_arima_{group}")
            print(f" Model for item {group} logged to MLflow.")

        return result

    except Exception as e:
        print(f" ARIMA failed for item {group}: {e}")
        return pd.DataFrame(columns=["group", "ds", "yhat"])
```

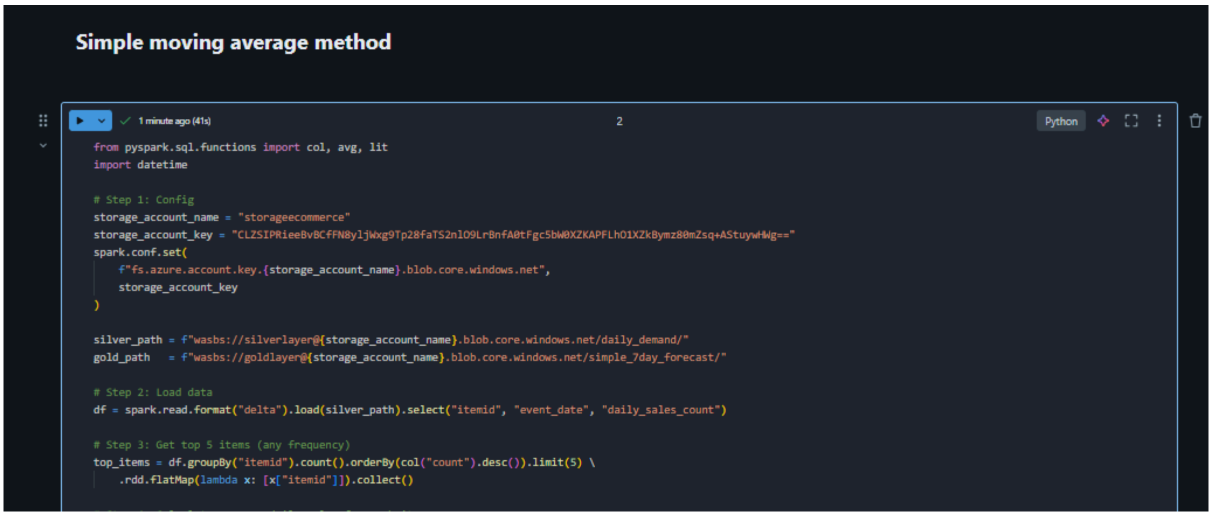


```
# Apply ARIMA per item

forecast_df = filtered_df.groupBy("group").applyInPandas(arima_forecast, schema=schema)


# Final cast for ds column

forecast_df = forecast_df.withColumn("ds", col("ds").cast("date"))


# Save to Gold Layer (Delta + CSV)

forecast_df.write.format("delta").mode("overwrite").save(gold_path)
forecast_df.write.option("header", "true").mode("overwrite").csv(gold_csv_path)

print(" ARIMA item-level forecast saved to Gold Layer.")
forecast_df.show()
```



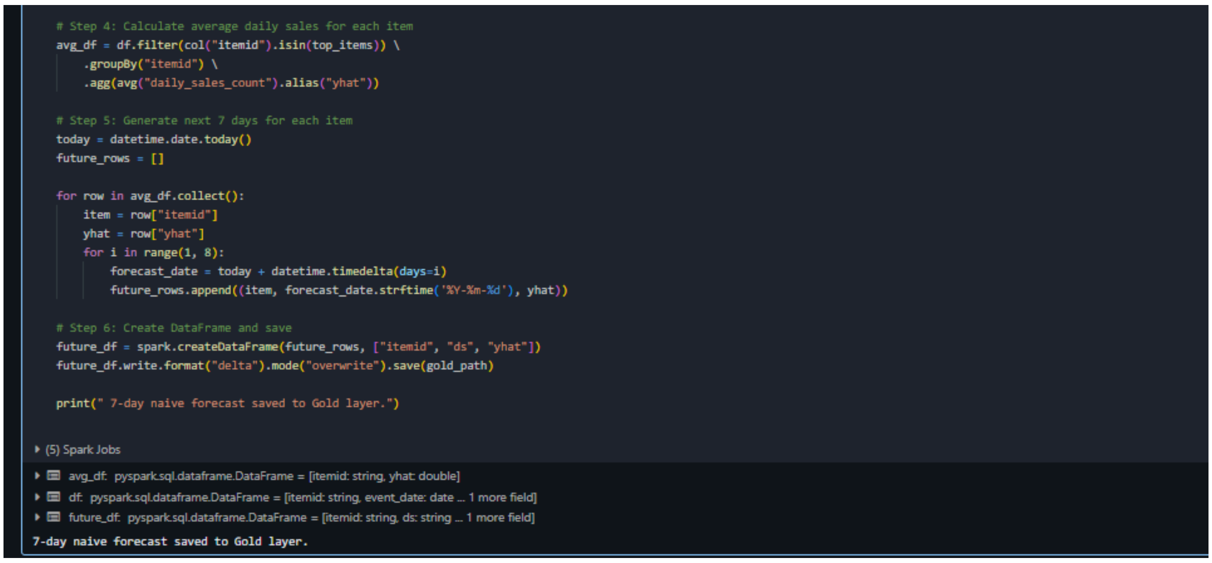
```
▸ (12) Spark Jobs
▸ daily_df:  pyspark.sql.dataframe.DataFrame = [group: string, ds: date ... 1 more field]
▸ df:  pyspark.sql.dataframe.DataFrame = [itemid: string, event_date: date ... 1 more field]
▸ filtered_df:  pyspark.sql.dataframe.DataFrame = [group: string, ds: date ... 1 more field]
▸ forecast_df:  pyspark.sql.dataframe.DataFrame = [group: string, ds: date ... 1 more field]
▸ item_day_counts:  pyspark.sql.dataframe.DataFrame = [group: string, active_days: long]
| 20158|NULL|54.0|
| 20158|NULL|54.0|
| 20158|NULL|54.0|
| 20158|NULL|54.0|
| 20158|NULL|54.0|
| 20158|NULL|54.0|
|238447|NULL|18.0|
|238447|NULL|18.0|
|238447|NULL|18.0|
|238447|NULL|18.0|
|238447|NULL|18.0|
|238447|NULL|18.0|
|238447|NULL|18.0|
|280029|NULL|18.0|
|280029|NULL|18.0|
|280029|NULL|18.0|
|280029|NULL|18.0|
|280029|NULL|18.0|
|280029|NULL|18.0|
+------+----+----+
only showing top 20 rows
```

Method 2: Using Simple moving averages to reduce time complexity and job runs



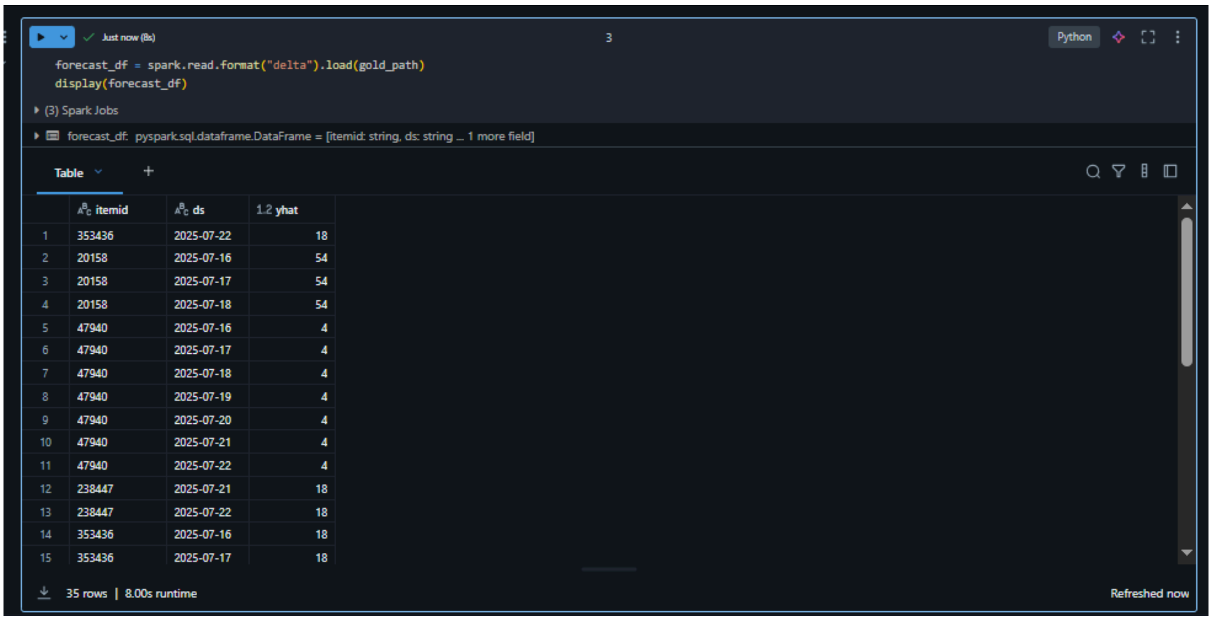
Simple moving average method

```
from pyspark.sql.functions import col, avg, lit
import datetime

# Step 1: Config
storage_account_name = "storageecommerce"
storage_account_key = "CLZSIPRieeBvBCfFN8yljWxg9Tp28faTS2nlO9LrBnfA0tFgc5bW0XZKAPFLhO1XZkBymz80mZsq+AStuywHWg=="
spark.conf.set(
    f"fs.azure.account.key.{storage_account_name}.blob.core.windows.net",
    storage_account_key
)

silver_path = f"wasbs://silverlayer@{storage_account_name}.blob.core.windows.net/daily_demand/"
gold_path   = f"wasbs://goldlayer@{storage_account_name}.blob.core.windows.net/simple_7day_forecast/"

# Step 2: Load data
df = spark.read.format("delta").load(silver_path).select("itemid", "event_date", "daily_sales_count")

# Step 3: Get top 5 items (any frequency)
top_items = df.groupBy("itemid").count().orderBy(col("count").desc()).limit(5) \
    .rdd.flatMap(lambda x: [x["itemid"]]).collect()
```



```
# Step 4: Calculate average daily sales for each item
avg_df = df.filter(col("itemid").isin(top_items)) \
    .groupBy("itemid") \
    .agg(avg("daily_sales_count").alias("yhat"))

# Step 5: Generate next 7 days for each item
today = datetime.date.today()
future_rows = []

for row in avg_df.collect():
    item = row["itemid"]
    yhat = row["yhat"]
    for i in range(1, 8):
        forecast_date = today + datetime.timedelta(days=i)
        future_rows.append((item, forecast_date.strftime('%Y-%m-%d'), yhat))

# Step 6: Create DataFrame and save
future_df = spark.createDataFrame(future_rows, ["itemid", "ds", "yhat"])
future_df.write.format("delta").mode("overwrite").save(gold_path)

print(" 7-day naive forecast saved to Gold layer.")
```

```
(5) Spark Jobs
avg_df:  pyspark.sql.dataframe.DataFrame = [itemid: string, yhat: double]
df:  pyspark.sql.dataframe.DataFrame = [itemid: string, event_date: date ... 1 more field]
future_df:  pyspark.sql.dataframe.DataFrame = [itemid: string, ds: string ... 1 more field]
7-day naive forecast saved to Gold layer.
```



```
forecast_df = spark.read.format("delta").load(gold_path)
display(forecast_df)
```

```
(3) Spark Jobs
forecast_df:  pyspark.sql.dataframe.DataFrame = [itemid: string, ds: string ... 1 more field]
```

	itemid	ds	yhat
1	353436	2025-07-22	18
2	20158	2025-07-16	54
3	20158	2025-07-17	54
4	20158	2025-07-18	54
5	47940	2025-07-16	4
6	47940	2025-07-17	4
7	47940	2025-07-18	4
8	47940	2025-07-19	4
9	47940	2025-07-20	4
10	47940	2025-07-21	4
11	47940	2025-07-22	4
12	238447	2025-07-21	18
13	238447	2025-07-22	18
14	353436	2025-07-16	18
15	353436	2025-07-17	18

35 rows | 8.00s runtime

Refreshed now

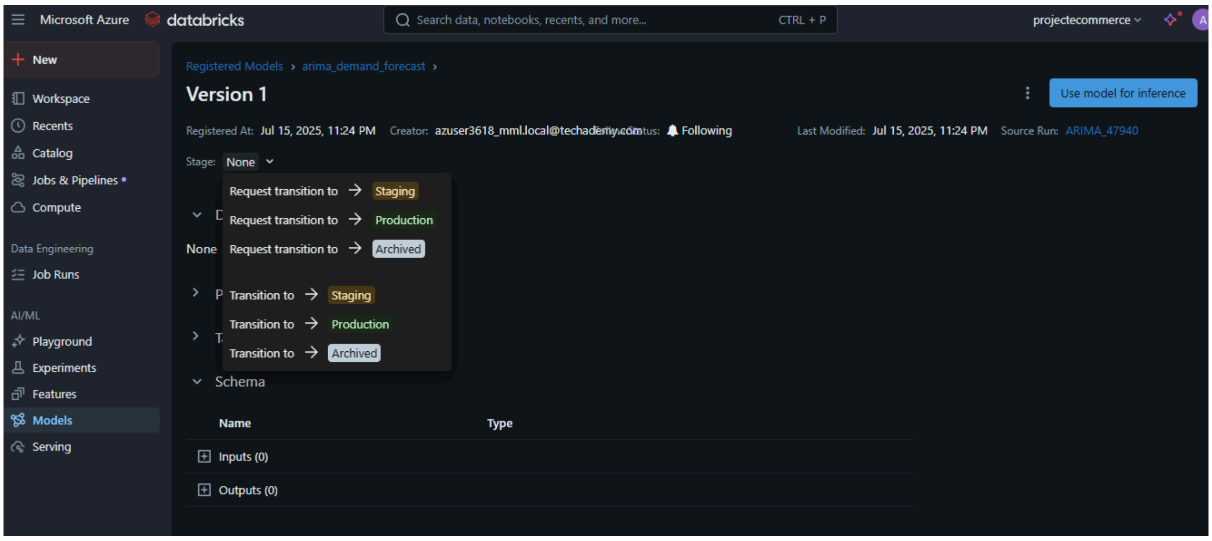
3.4. ML CI/CD Pipeline (Azure DevOps & MLflow Model Registry)

This phase is a cornerstone of MLOps, demonstrating how to automate the promotion of trained machine learning models through different lifecycle stages using Azure DevOps.

- **Objective:** Implement an automated pipeline in Azure DevOps that transitions the trained ARIMA model (Version 1) from the "Staging" to the "Production" stage within the MLflow Model Registry.
- **Key Steps and Technical Details:**
  - **Model Registration in MLflow:** After successful training in the `silvertomlgold` notebook, the ARIMA model is registered in the MLflow Model Registry under a specific name (e.g., `arima_demand_forecast`). This creates a versioned entry for the model.
  - **Manual Stage Transition (Initial Setup for Approval Workflow):** To simulate a real-world MLOps workflow, the model (Version 1) is initially transitioned manually:
    - From "None" to "Staging" in the Databricks MLflow UI (e.g., for testing/QA).
    - A request is then made to transition it from "Staging" to "Production," which is subsequently approved. This sets up the model in a state where the automated pipeline can take over.
  - **Azure DevOps Pipeline Creation (`azure-pipelines.yml`):**
    - A new YAML pipeline is created in Azure DevOps, linked to the project's Azure Repos Git repository (`devopsecommerce`).
    - **`trigger: main`**: Configures the pipeline to automatically run whenever changes are pushed to the `main` branch, enabling continuous integration.
    - **`pool: vmImage: ubuntu-latest`**: Specifies that the pipeline jobs will run on a Microsoft-hosted Ubuntu agent, providing a managed execution environment.
    - **`variables`**:
      - `DATABRICKS_HOST`: The URL of the Databricks workspace (e.g., `https://adb-xxxx.azuredatabricks.net`).
      - `DATABRICKS_TOKEN`: A **highly sensitive** Databricks personal access token. This variable is configured as a **secret variable** in Azure DevOps Pipeline settings to prevent its value from being exposed in logs.
      - `MODEL_NAME`: The registered name of the MLflow model (`arima_demand_forecast`).
      - `MODEL_VERSION`: The specific version of the model to be promoted (`1`).
      - `STAGE`: The target stage for promotion (`Production`).
    - **`jobs` (`PromoteModel`):**
      - Contains a `script` step that executes a `curl` command.

        - **`curl -X POST "$DATABRICKS_HOST/api/2.0/mlflow/model-versions/transition-stage"`**: This command directly calls the MLflow Model Registry REST API.
        - **`-H "Authorization: Bearer $DATABRICKS_TOKEN"`**: Authenticates the request using the securely stored Databricks token.
        - **`-H "Content-Type: application/json"`**: Specifies the request body format.
        - **`-d '{...}'`**: The JSON payload containing the `name` (model name), `version`, `stage` (target stage), and `archive_existing_versions` (set to `false` to keep previous versions).
        - `displayName: 'Promote Model to Production'`: Provides a clear name for the step in the pipeline logs.
    - **Pipeline Execution & Troubleshooting:**
        - Initial attempts to run the pipeline resulted in a common Azure DevOps error: **"No hosted parallelism has been granted."** This is a security measure for new Azure DevOps organizations.
        - **Resolution:** This required submitting a request form (`https://aka.ms/azpipelines-parallelism-request`) to Microsoft to grant free hosted parallelism. The pipeline could only succeed after this approval was received.
- **Tools Used:** MLflow Model Registry (Databricks), Azure DevOps Pipelines (YAML), `curl` (for REST API interaction), Azure Repos Git.

Creating a model in MLflow Databricks:



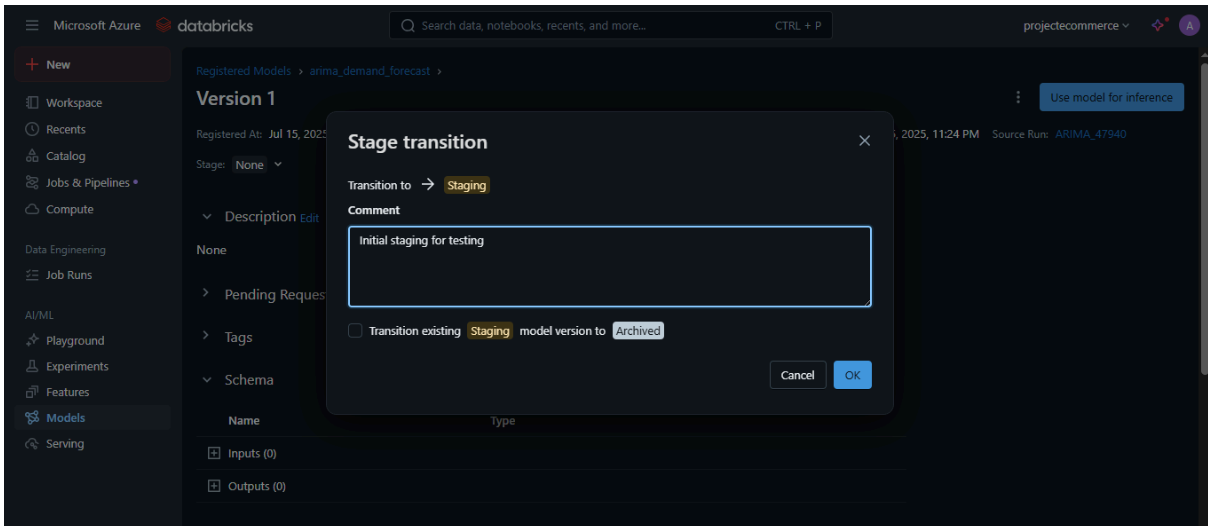
Registered Models › arima_demand_forecast ›

Version 1

Registered At: Jul 15, 2025, 11:24 PM Creator: azuser3618_mml.local@techademy.com Status: Following Last Modified: Jul 15, 2025, 11:24 PM Source Run: ARIMA_47940

Stage: None

Setting the version1 arima_demanded_forecast model to staging:



Version 1

Registered At: Jul 15, 2025 ... 2025, 11:24 PM Source Run: ARIMA_47940

Stage: None

Description Edit

None

Pending Reques

Tags

Schema

Name	Type
Inputs (0)	
Outputs (0)	

Stage transition

Transition to → Staging

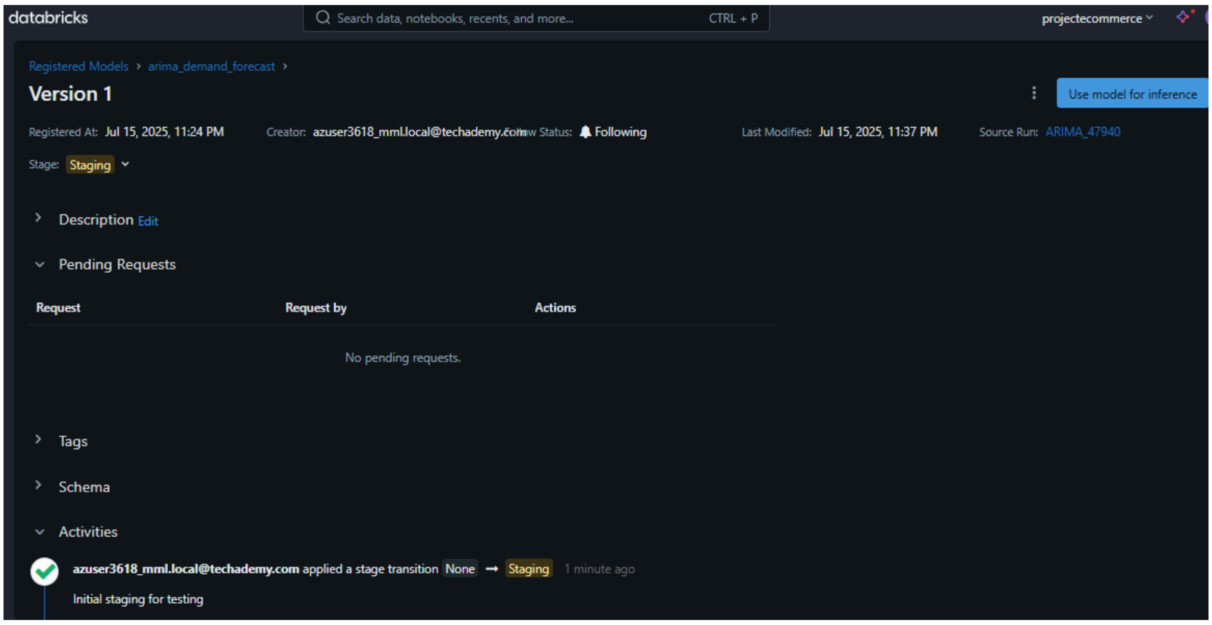
Comment

Initial staging for testing

Transition existing Staging model version to Archived

Cancel OK

Model is now at staging phase(Testing or Quality Assurance):


databricks
Search data, notebooks, recents, and more... CTRL + P
projectecommerce
Registered Models > arima_demand_forecast >
Use model for inference

Version 1

Registered At: Jul 15, 2025, 11:24 PM Creator: azuser3618_mml.local@techademy.com Status: Following Last Modified: Jul 15, 2025, 11:37 PM Source Run: ARIMA_47940

Stage: Staging

Description Edit

Pending Requests

Request	Request by	Actions
No pending requests.		

Tags

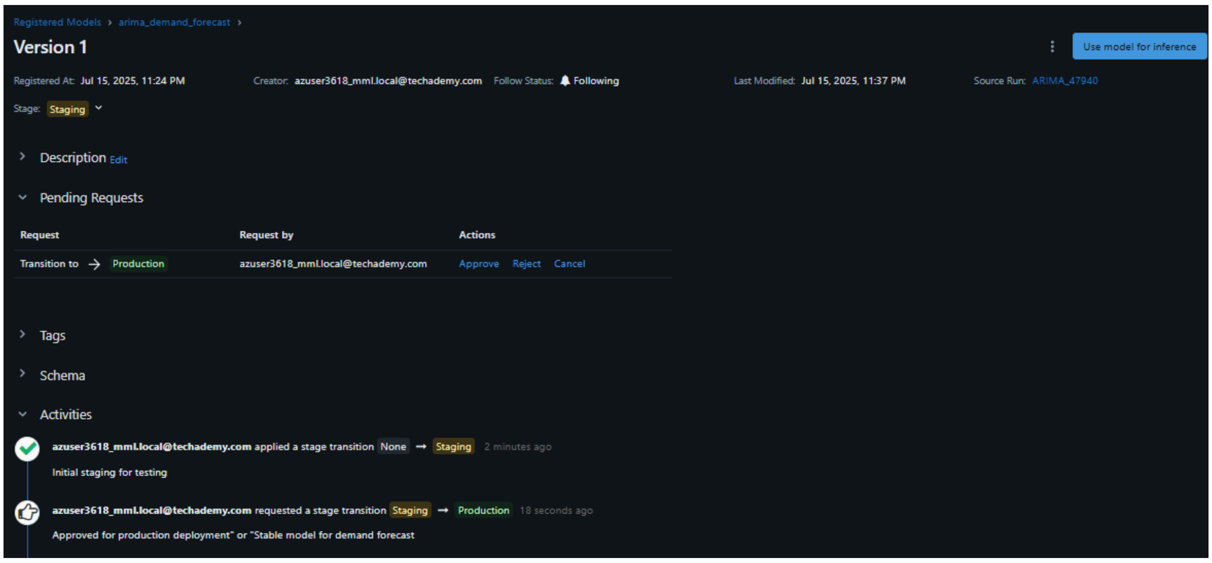
Schema

Activities

azuser3618_mml.local@techademy.com applied a stage transition None → Staging 1 minute ago

Initial staging for testing

Requesting to change it for production(Ready for Deployment) change:



Registered Models › arima_demand_forecast ›

Version 1

Registered At: Jul 15, 2025, 11:24 PM   Creator: azuser3618_mml.local@techademy.com   Follow Status: Following   Last Modified: Jul 15, 2025, 11:37 PM   Source Run: ARIMA_47940

Stage: Staging

Description Edit

Pending Requests

Request	Request by	Actions
Transition to → Production	azuser3618_mml.local@techademy.com	Approve Reject Cancel

Tags

Schema

Activities

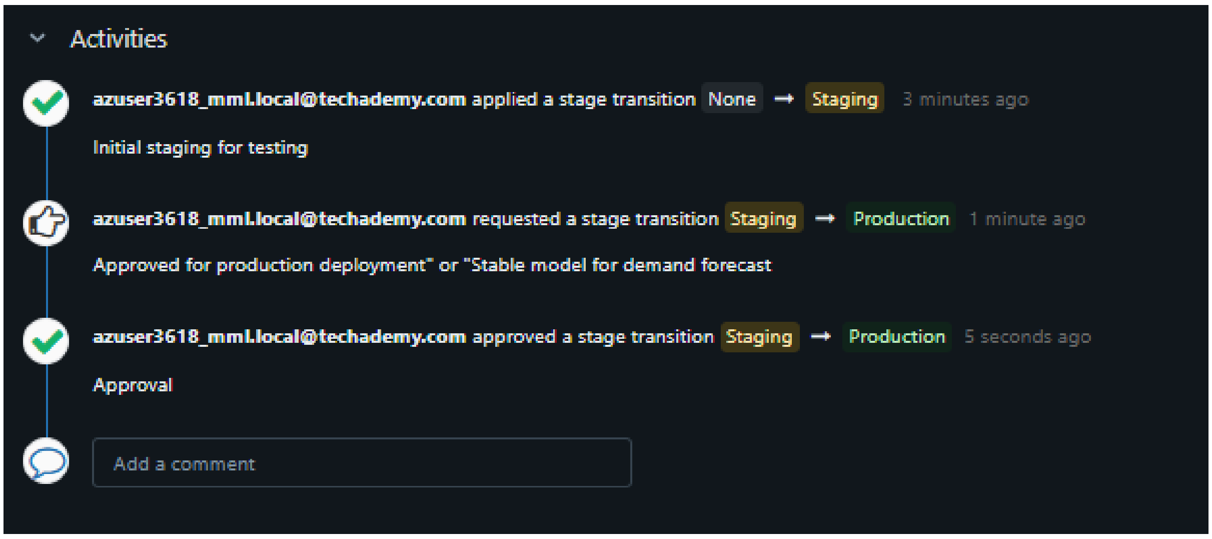
azuser3618_mml.local@techademy.com applied a stage transition None → Staging 2 minutes ago

Initial staging for testing

azuser3618_mml.local@techademy.com requested a stage transition Staging → Production 18 seconds ago

Approved for production deployment" or "Stable model for demand forecast

Approved for production phase:



Activities

azuser3618_mml.local@techademy.com applied a stage transition None → Staging 3 minutes ago

Initial staging for testing

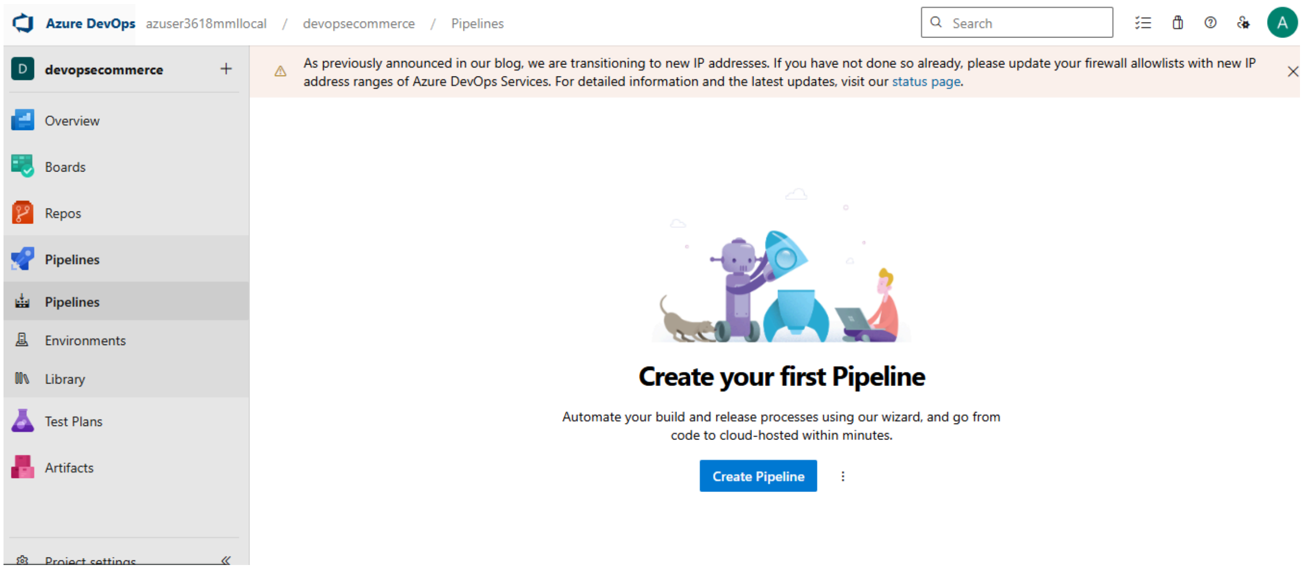
azuser3618_mml.local@techademy.com requested a stage transition Staging → Production 1 minute ago

Approved for production deployment" or "Stable model for demand forecast

azuser3618_mml.local@techademy.com approved a stage transition Staging → Production 5 seconds ago

Approval

Creating Azure devops release pipeline:



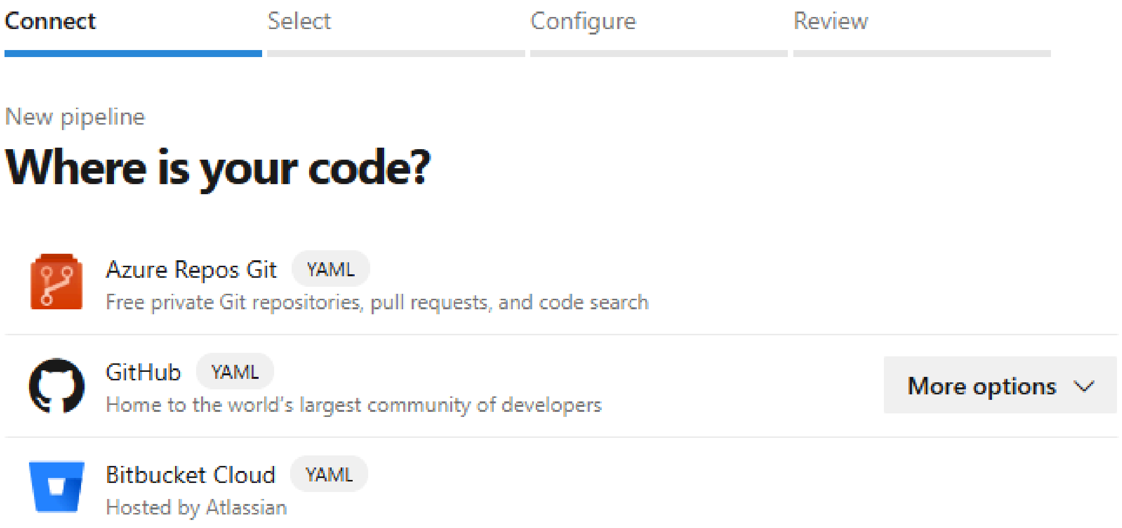
Azure DevOps azuser3618mmllocal / devopsecommerce / Pipelines

As previously announced in our blog, we are transitioning to new IP addresses. If you have not done so already, please update your firewall allowlists with new IP address ranges of Azure DevOps Services. For detailed information and the latest updates, visit our status page.

Create your first Pipeline

Automate your build and release processes using our wizard, and go from code to cloud-hosted within minutes.

Our source is Azure repos git:



New pipeline

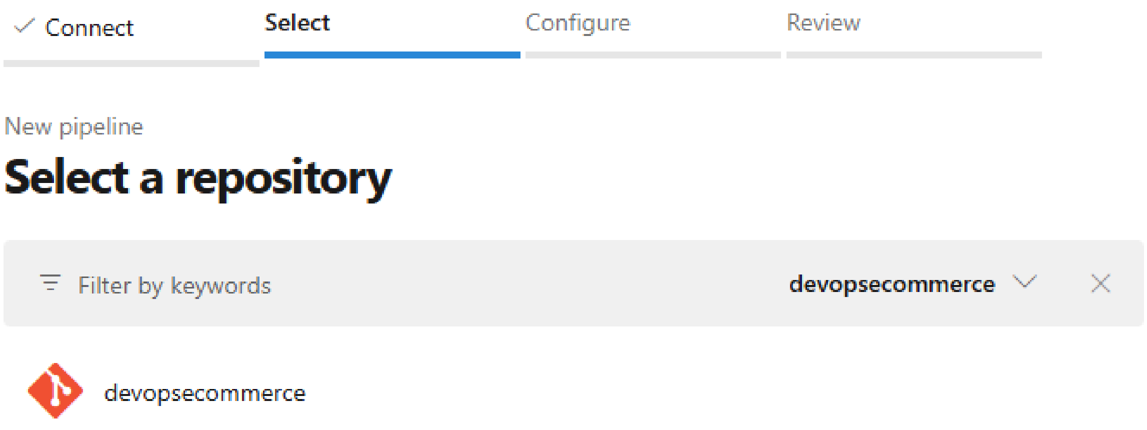
Where is your code?

Azure Repos Git YAML
Free private Git repositories, pull requests, and code search

GitHub YAML
Home to the world's largest community of developers

Bitbucket Cloud YAML
Hosted by Atlassian

Our project is devopsecommerce:

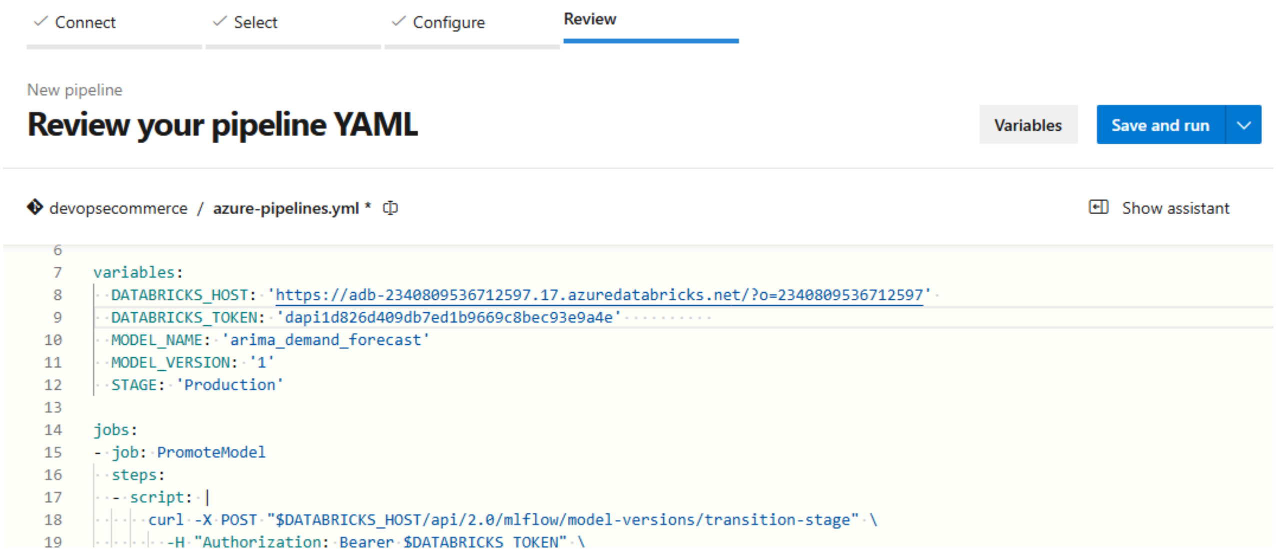


New pipeline

Select a repository

devopsecommerce

Editing our custom YAML code with our Databricks token:



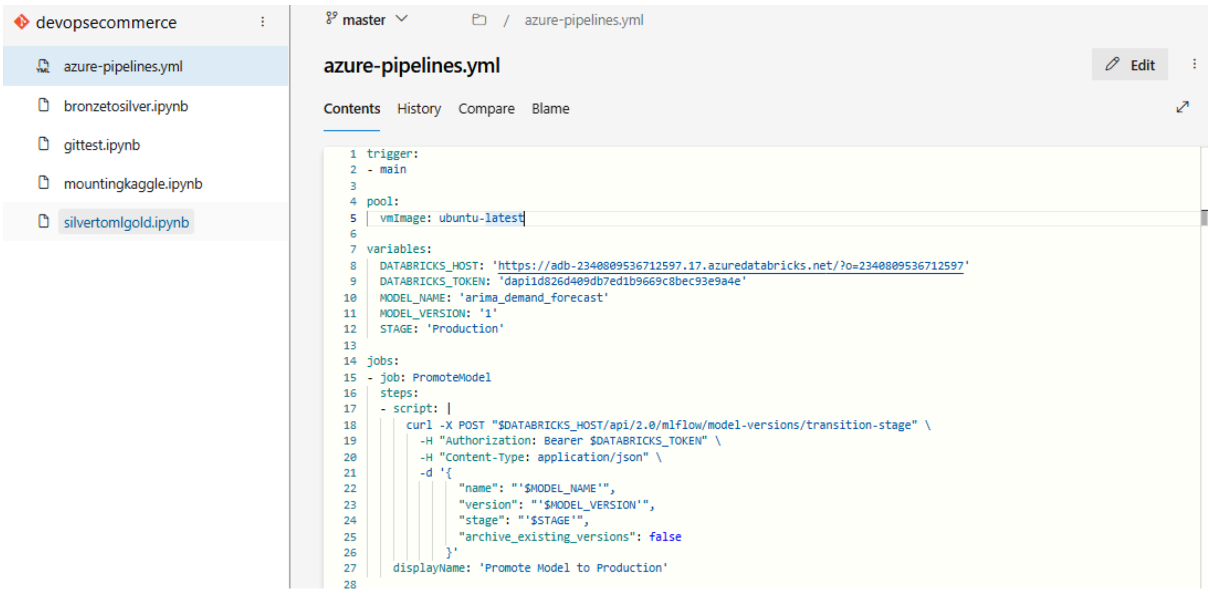
New pipeline

Review your pipeline YAML

devopsecommerce / azure-pipelines.yml *

```yaml

variables:
  DATABRICKS_HOST: 'https://adb-2340809536712597.17.azuredatabricks.net/?o=2340809536712597'
  DATABRICKS_TOKEN: 'dapi1d826d409db7ed1b9669c8bec93e9a4e'
  MODEL_NAME: 'arima_demand_forecast'
  MODEL_VERSION: '1'
  STAGE: 'Production'

jobs:
- job: PromoteModel
  steps:
  - script: |
      curl -X POST "$DATABRICKS_HOST/api/2.0/mlflow/model-versions/transition-stage" \
        -H "Authorization: Bearer $DATABRICKS_TOKEN" \
```

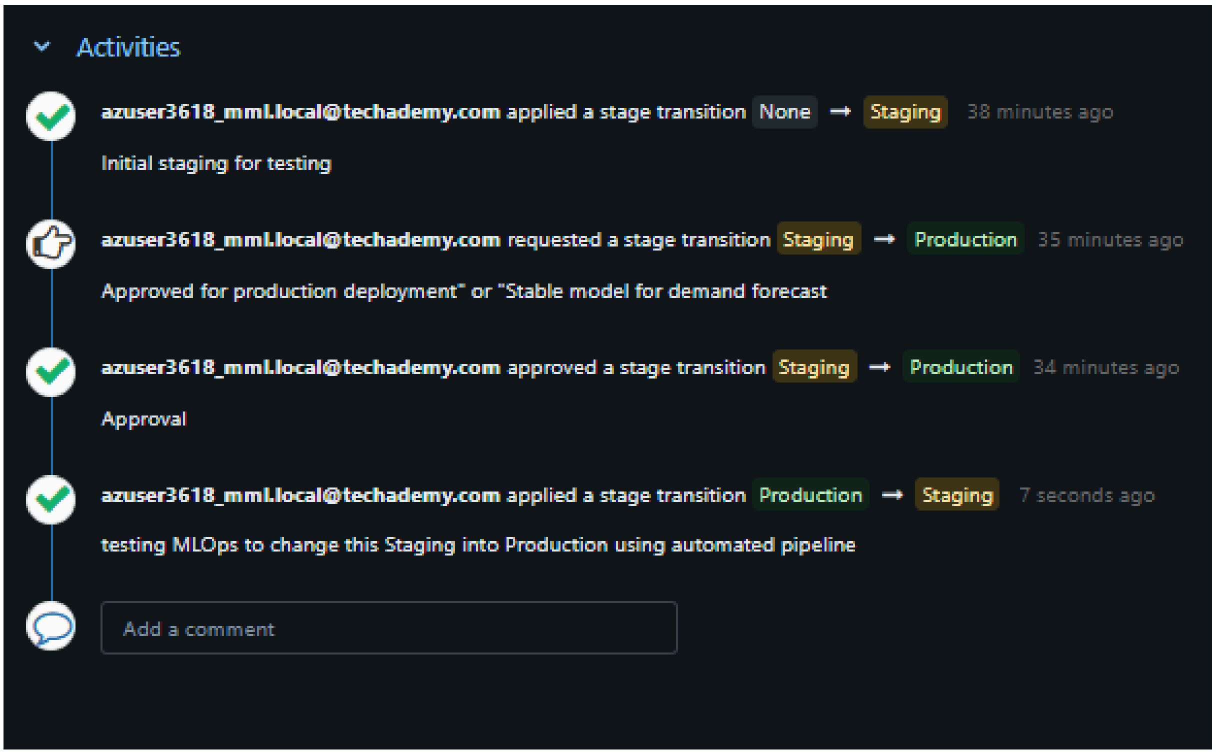
Saving the YAML file:



azure-pipelines.yml

```yaml
trigger:
- main

pool:
  vmImage: ubuntu-latest

variables:
  DATABRICKS_HOST: 'https://adb-2340809536712597.17.azuredatabricks.net/?o=2340809536712597'
  DATABRICKS_TOKEN: 'dapi1d826d409db7ed1b9669c8bec93e9a4e'
  MODEL_NAME: 'arima_demand_forecast'
  MODEL_VERSION: '1'
  STAGE: 'Production'

jobs:
- job: PromoteModel
  steps:
  - script: |
      curl -X POST "$DATABRICKS_HOST/api/2.0/mlflow/model-versions/transition-stage" \
        -H "Authorization: Bearer $DATABRICKS_TOKEN" \
        -H "Content-Type: application/json" \
        -d '{
              "name": "'$MODEL_NAME'",
              "version": "'$MODEL_VERSION'",
              "stage": "'$STAGE'",
              "archive_existing_versions": false
            }'
    displayName: 'Promote Model to Production'

```

Changing the production to staging for testing the pipeline:



Activities

azuser3618_mml.local@techademy.com applied a stage transition None → Staging 38 minutes ago

Initial staging for testing

azuser3618_mml.local@techademy.com requested a stage transition Staging → Production 35 minutes ago

Approved for production deployment" or "Stable model for demand forecast

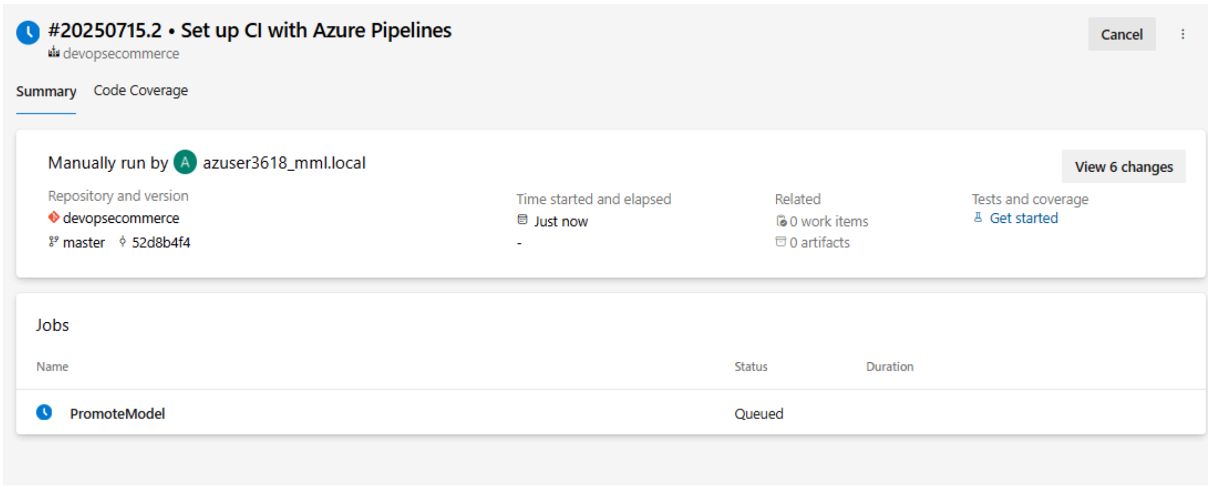
azuser3618_mml.local@techademy.com approved a stage transition Staging → Production 34 minutes ago

Approval

azuser3618_mml.local@techademy.com applied a stage transition Production → Staging 7 seconds ago

testing MLOps to change this Staging into Production using automated pipeline

Saving and Running by CI/CD pipeline:



#20250715.2 • Set up CI with Azure Pipelines
devopsecommerce

Summary | Code Coverage

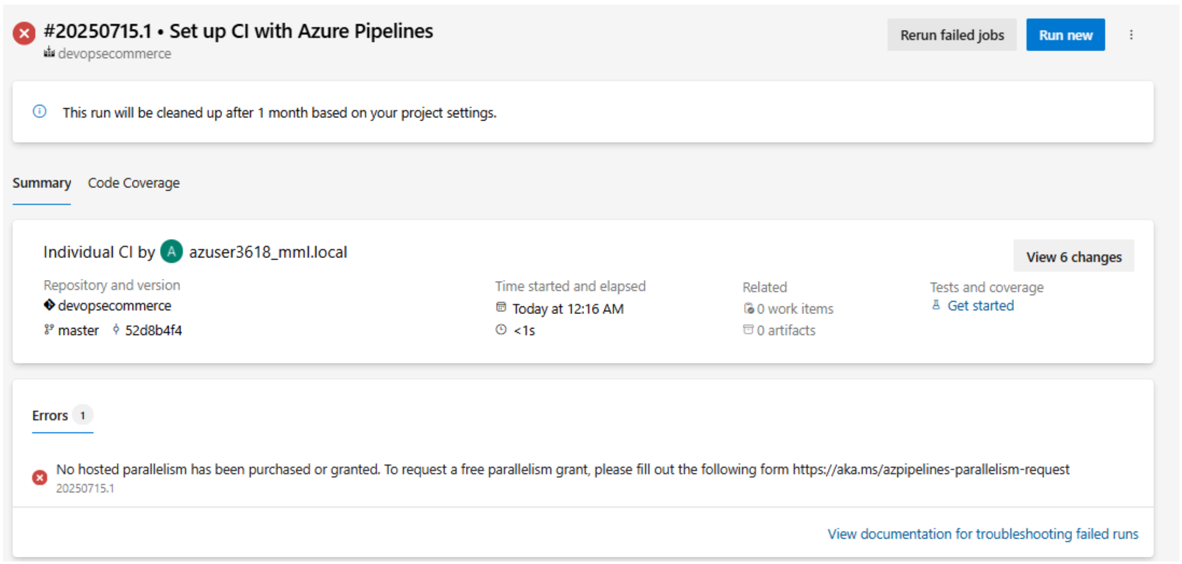
Manually run by azuser3618_mml.local

Repository and version: devopsecommerce, master, 52d8b4f4
Time started and elapsed: Just now, -
Related: 0 work items, 0 artifacts
Tests and coverage: Get started

Jobs

Name	Status	Duration
PromoteModel	Queued	

Azure pipeline fails:



#20250715.1 • Set up CI with Azure Pipelines
devopsecommerce

This run will be cleaned up after 1 month based on your project settings.

Summary | Code Coverage

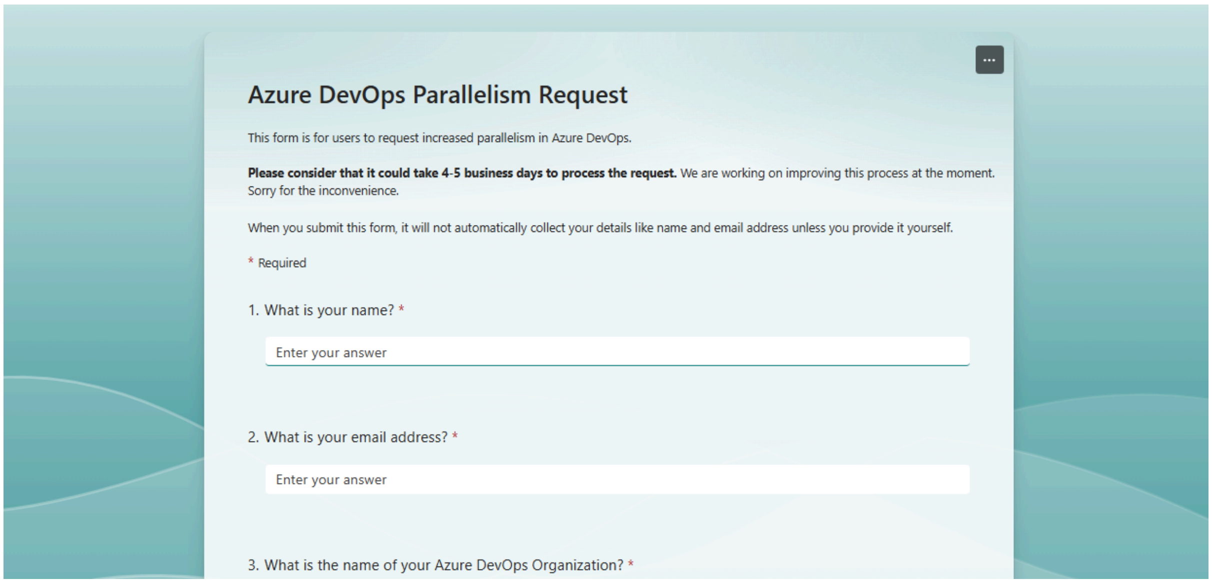
Individual CI by azuser3618_mml.local

Repository and version: devopsecommerce, master, 52d8b4f4
Time started and elapsed: Today at 12:16 AM, <1s
Related: 0 work items, 0 artifacts
Tests and coverage: Get started

Errors 1

No hosted parallelism has been purchased or granted. To request a free parallelism grant, please fill out the following form https://aka.ms/azpipelines-parallelism-request
20250715.1

View documentation for troubleshooting failed runs



Azure DevOps Parallelism Request

This form is for users to request increased parallelism in Azure DevOps.

Please consider that it could take 4-5 business days to process the request. We are working on improving this process at the moment. Sorry for the inconvenience.

When you submit this form, it will not automatically collect your details like name and email address unless you provide it yourself.

* Required

1. What is your name? *

Enter your answer

2. What is your email address? *

Enter your answer

3. What is the name of your Azure DevOps Organization? *

- The error message:
  "No hosted parallelism has been granted or granted. To request a free parallelism grant, please fill out the following form https://aka.ms/azpipelines-parallelism-request"

- Filling out the form with our azure credentials

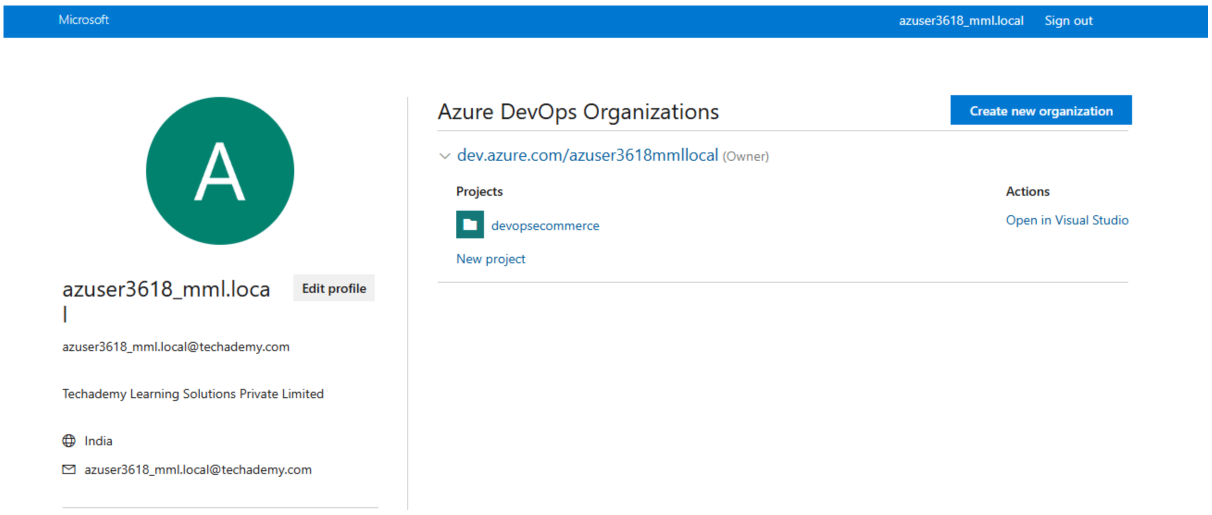
- Waiting for Approval: Microsoft reviews these requests, and it can take some time to receive an email notification once your request is approved.

- Rerun the Pipeline: Once you receive confirmation that parallelism has been granted, we can rerun the pipeline by "Run new" or "Rerun failed jobs".

- Without this grant, our pipelines cannot run on Microsoft-hosted agents

Proceeding with the Azure Data Factory for pipeline as Devops takes time to grant permission:

Azure Devops Homepage:



azuser3618_mml.local

azuser3618_mml.local@techademy.com

Techademy Learning Solutions Private Limited

India

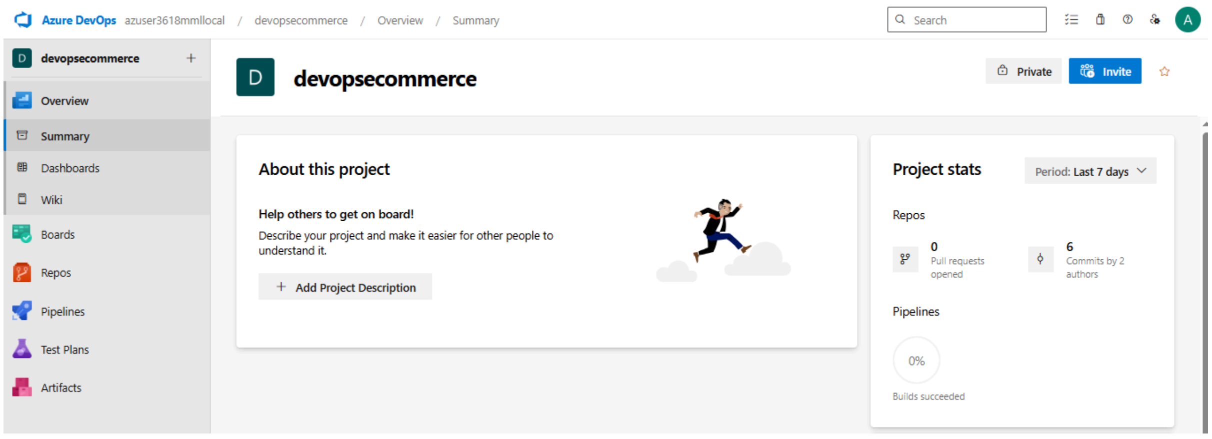
azuser3618_mml.local@techademy.com

Azure DevOps Organizations

dev.azure.com/azuser3618mmllocal (Owner)

Projects	Actions
devopsecommerce	Open in Visual Studio

Project named as "devopsecommerce":



devopsecommerce

About this project

Help others to get on board!

Describe your project and make it easier for other people to understand it.

Project stats

Repos

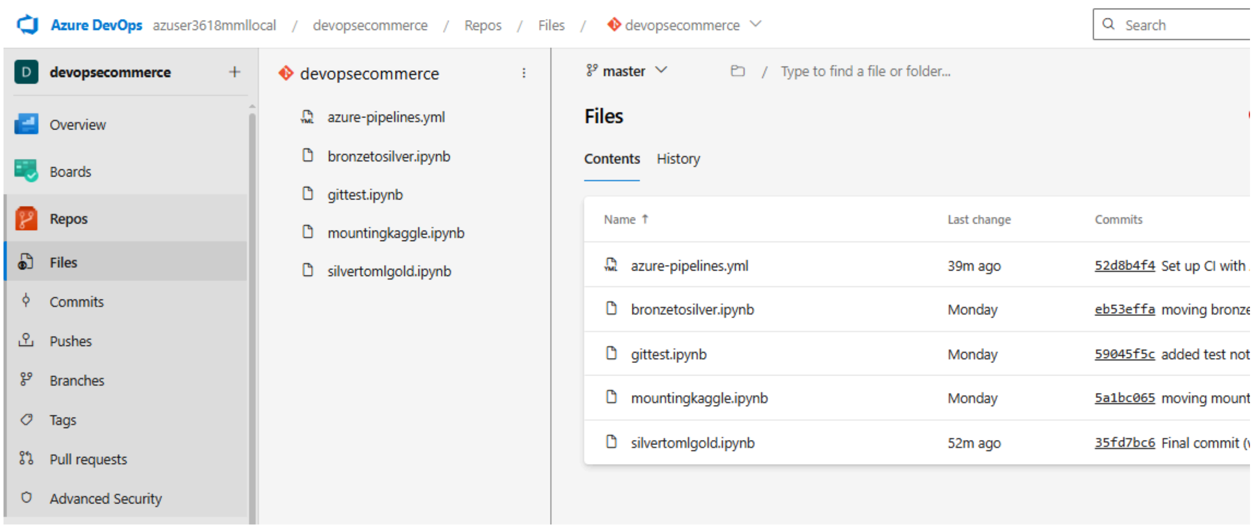
0 Pull requests opened

6 Commits by 2 authors

Pipelines

0% Builds succeeded

Files inside “ecommerce”:



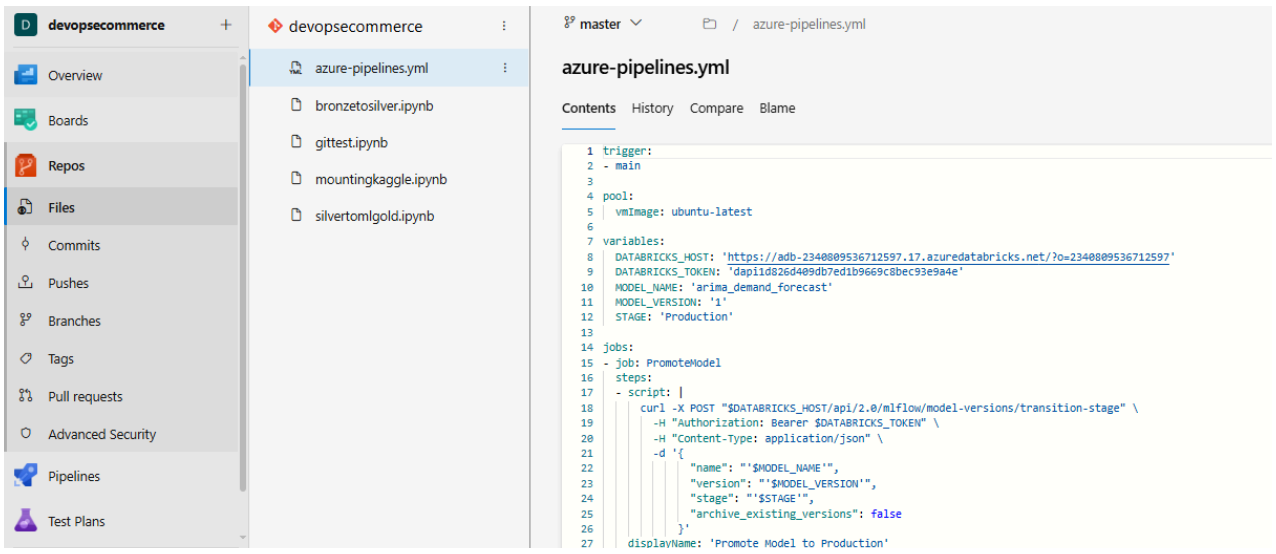
devopsecommerce

- azure-pipelines.yml
- bronzetosilver.ipynb
- gittest.ipynb
- mountingkaggle.ipynb
- silvertomlgold.ipynb

Files

Name ↑	Last change	Commits
azure-pipelines.yml	39m ago	52d8b4f4 Set up CI with
bronzetosilver.ipynb	Monday	eb53effa moving bronze
gittest.ipynb	Monday	59045f5c added test not
mountingkaggle.ipynb	Monday	5a1bc065 moving mount
silvertomlgold.ipynb	52m ago	35fd7bc6 Final commit (

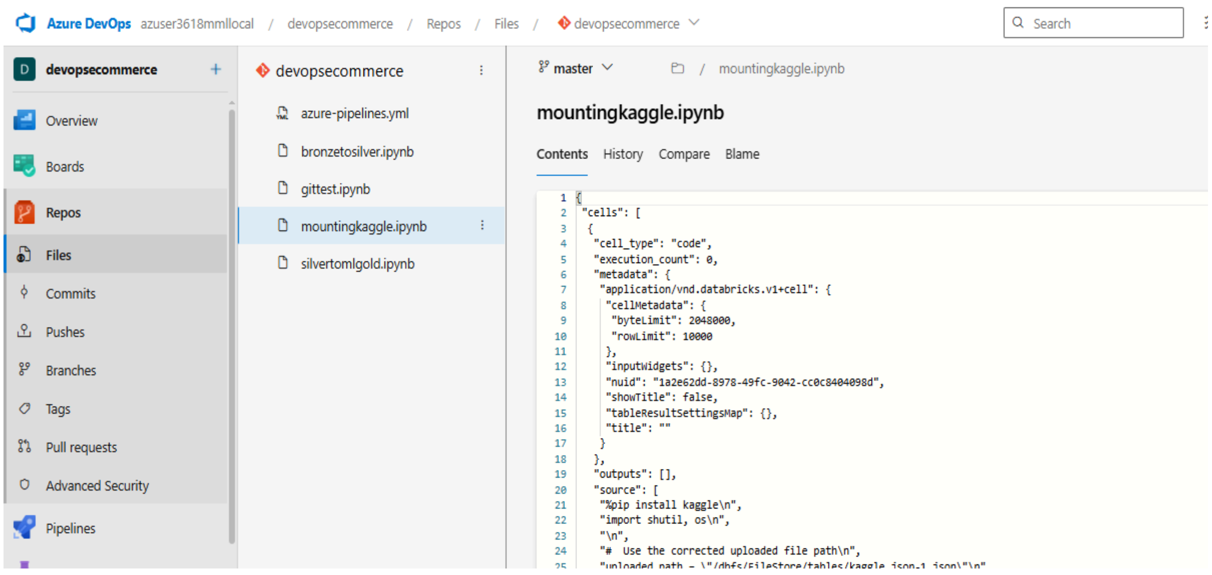
azure-pipeline.yml file:



azure-pipelines.yml

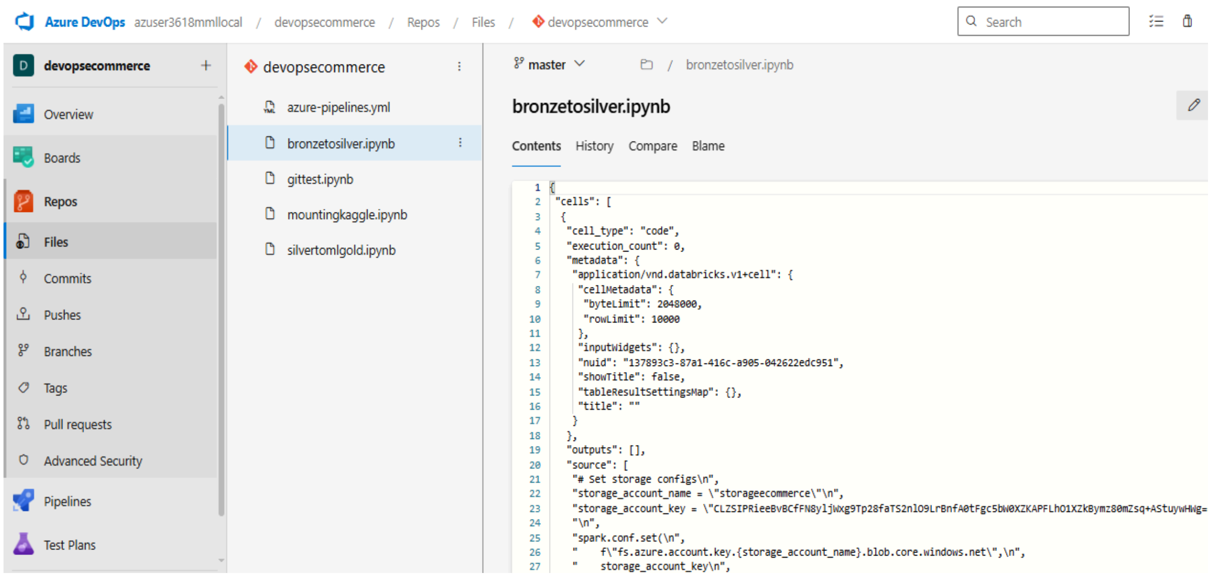
```yaml
trigger:
- main

pool:
  vmImage: ubuntu-latest

variables:
  DATABRICKS_HOST: 'https://adb-2340809536712597.17.azuredatabricks.net/?o=2340809536712597'
  DATABRICKS_TOKEN: 'dapi1d826d409db7ed1b9669c8bec93e9a4e'
  MODEL_NAME: 'arima_demand_forecast'
  MODEL_VERSION: '1'
  STAGE: 'Production'

jobs:
- job: PromoteModel
  steps:
  - script: |
      curl -X POST "$DATABRICKS_HOST/api/2.0/mlflow/model-versions/transition-stage" \
        -H "Authorization: Bearer $DATABRICKS_TOKEN" \
        -H "Content-Type: application/json" \
        -d '{
              "name": "'$MODEL_NAME'",
              "version": "'$MODEL_VERSION'",
              "stage": "'$STAGE'",
              "archive_existing_versions": false
            }'
    displayName: 'Promote Model to Production'
```

mountingkaggle.ipynb file:



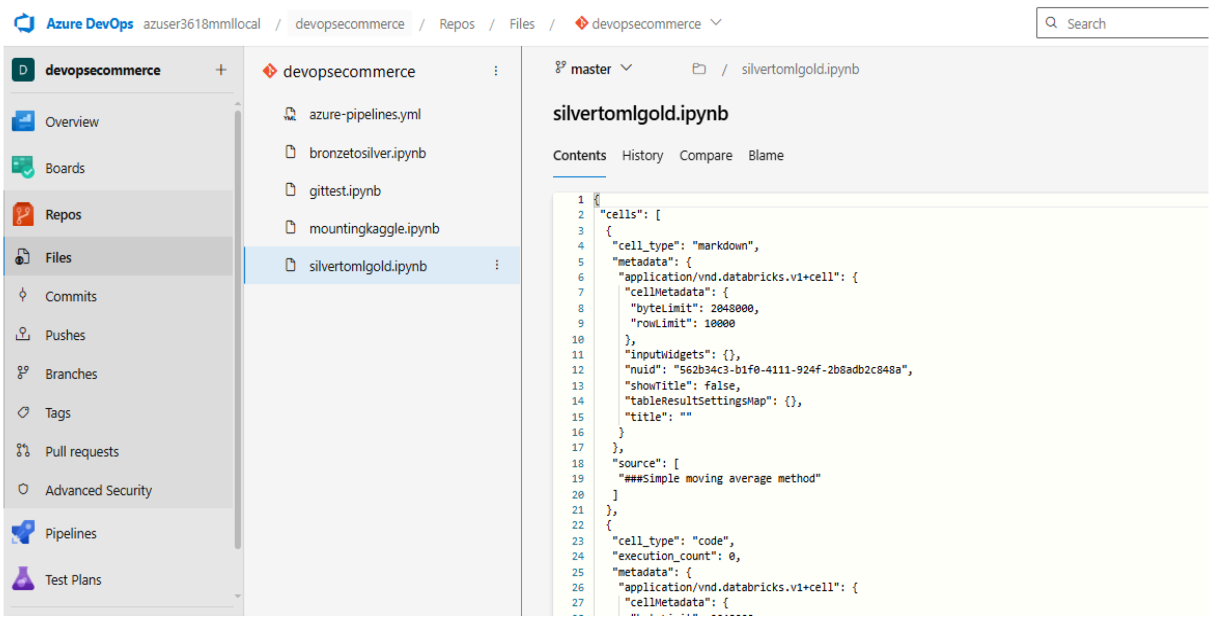
```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": 0,
   "metadata": {
    "application/vnd.databricks.v1+cell": {
     "cellMetadata": {
      "byteLimit": 2048000,
      "rowLimit": 10000
     },
     "inputWidgets": {},
     "nuid": "1a2e62dd-8978-49fc-9042-cc0c8404098d",
     "showTitle": false,
     "tableResultSettingsMap": {},
     "title": ""
    }
   },
   "outputs": [],
   "source": [
    "%pip install kaggle\n",
    "import shutil, os\n",
    "\n",
    "#  Use the corrected uploaded file path\n",
    "uploaded_path = \"/dbfs/FileStore/tables/kaggle_json-1.json\"\n"
```

bronzetosilver.ipynb file:



```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": 0,
   "metadata": {
    "application/vnd.databricks.v1+cell": {
     "cellMetadata": {
      "byteLimit": 2048000,
      "rowLimit": 10000
     },
     "inputWidgets": {},
     "nuid": "137893c3-87a1-416c-a905-042622edc951",
     "showTitle": false,
     "tableResultSettingsMap": {},
     "title": ""
    }
   },
   "outputs": [],
   "source": [
    "# Set storage configs\n",
    "storage_account_name = \"storageecommerce\"\n",
    "storage_account_key = \"CLZSIPRieeBvBCfFN8yljWxg9Tp28faTS2nlO9LrBnfA0tFgc5bW0XZKAPFLhO1XZkBymz80mZsq+AStuywHWg=
    "\n",
    "spark.conf.set(\n",
    "    f\"fs.azure.account.key.{storage_account_name}.blob.core.windows.net\",\n",
    "    storage_account_key\n",
```

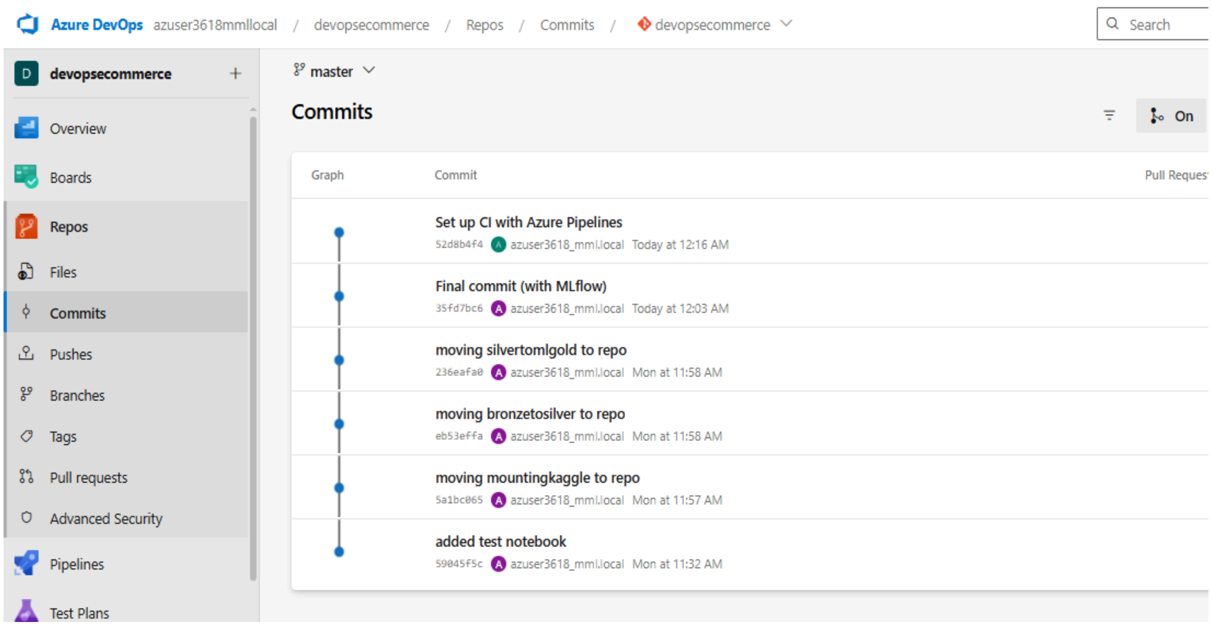
silvertomlgold.ipynb file:



silvertomlgold.ipynb

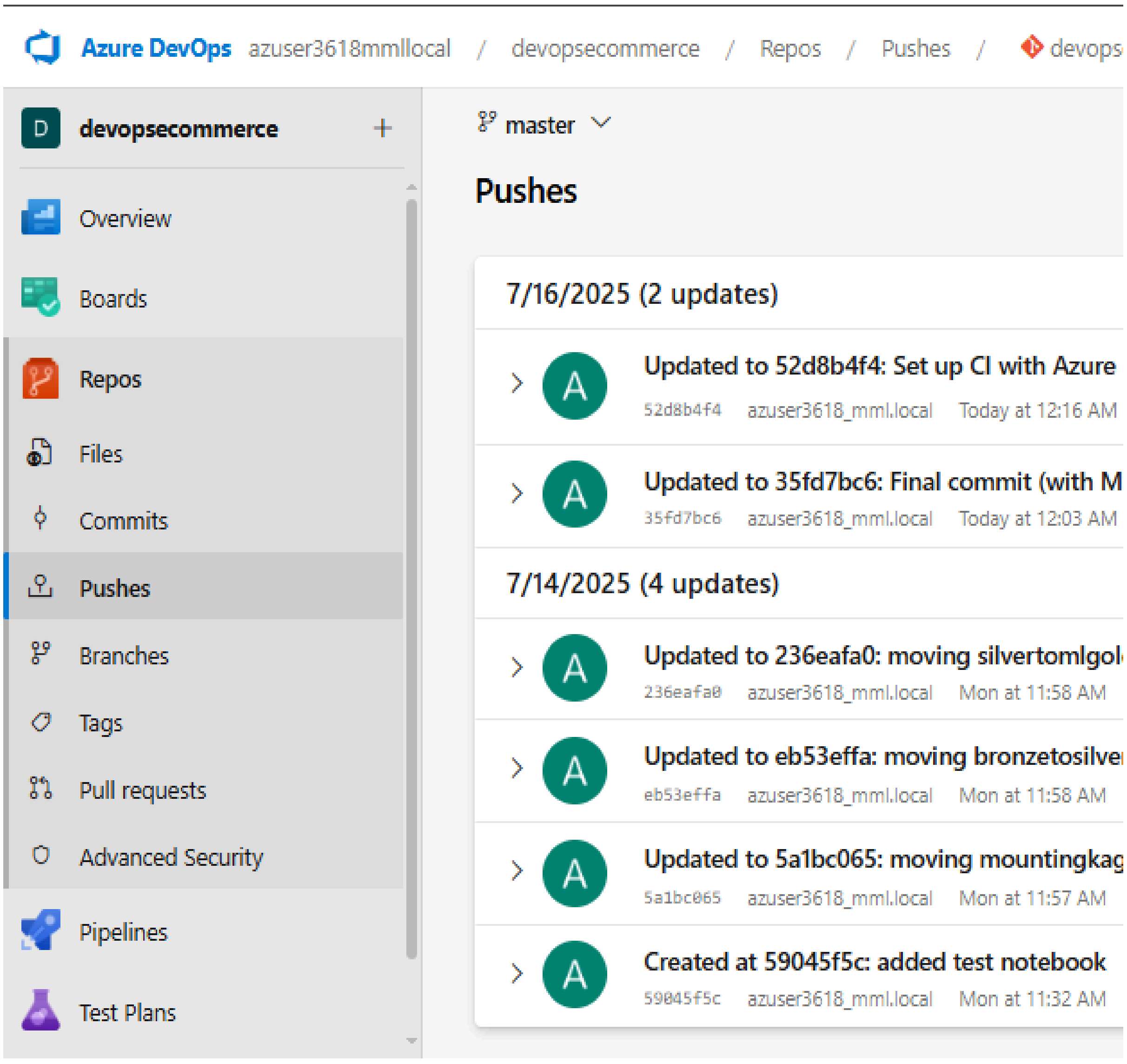
```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {
    "application/vnd.databricks.v1+cell": {
     "cellMetadata": {
      "byteLimit": 2048000,
      "rowLimit": 10000
     },
     "inputWidgets": {},
     "nuid": "562b34c3-b1f0-4111-924f-2b8adb2c848a",
     "showTitle": false,
     "tableResultSettingsMap": {},
     "title": ""
    }
   },
   "source": [
    "###Simple moving average method"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 0,
   "metadata": {
    "application/vnd.databricks.v1+cell": {
     "cellMetadata": {
```

Commits done to the project:



Graph	Commit	Pull Reques
●	Set up CI with Azure Pipelines 52d8b4f4 A azuser3618_mml.local Today at 12:16 AM	
●	Final commit (with MLflow) 35fd7bc6 A azuser3618_mml.local Today at 12:03 AM	
●	moving silvertomlgold to repo 236eafa0 A azuser3618_mml.local Mon at 11:58 AM	
●	moving bronzetosilver to repo eb53effa A azuser3618_mml.local Mon at 11:58 AM	
●	moving mountingkaggle to repo 5a1bc065 A azuser3618_mml.local Mon at 11:57 AM	
●	added test notebook 59045f5c A azuser3618_mml.local Mon at 11:32 AM	

Pushes done to the project:


Azure DevOps azuser3618mmllocal / devopsecommerce / Repos / Pushes / devops

devopsecommerce

Overview
Boards
Repos
Files
Commits
Pushes
Branches
Tags
Pull requests
Advanced Security
Pipelines
Test Plans

master

Pushes

7/16/2025 (2 updates)

Updated to 52d8b4f4: Set up CI with Azure
52d8b4f4 azuser3618_mml.local Today at 12:16 AM

Updated to 35fd7bc6: Final commit (with M
35fd7bc6 azuser3618_mml.local Today at 12:03 AM

7/14/2025 (4 updates)

Updated to 236eafa0: moving silvertomlgol
236eafa0 azuser3618_mml.local Mon at 11:58 AM

Updated to eb53effa: moving bronzetosilve
eb53effa azuser3618_mml.local Mon at 11:58 AM

Updated to 5a1bc065: moving mountingkag
5a1bc065 azuser3618_mml.local Mon at 11:57 AM

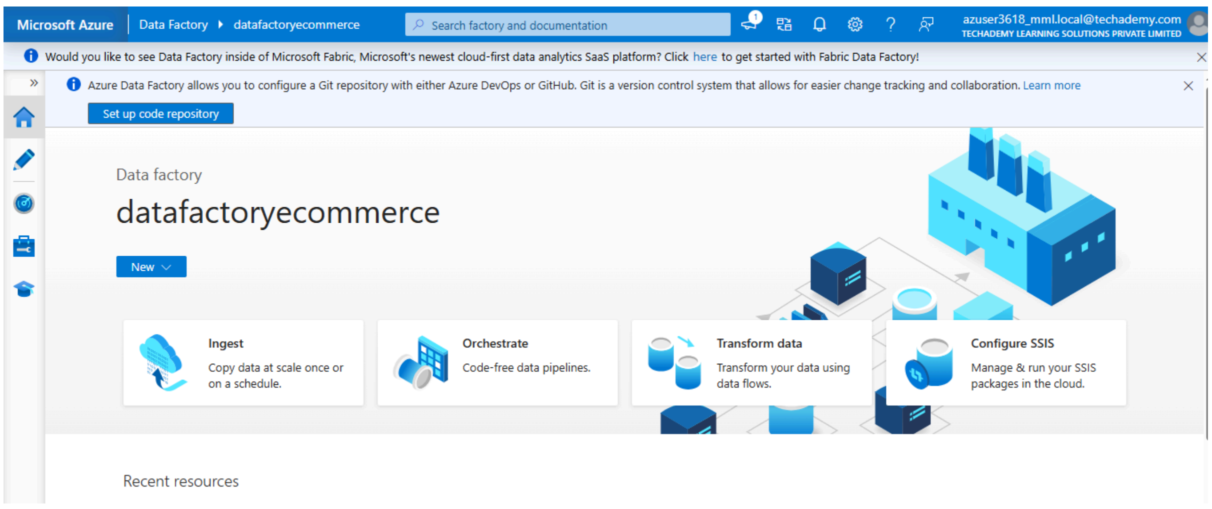
Created at 59045f5c: added test notebook
59045f5c azuser3618_mml.local Mon at 11:32 AM

3.5. Data Orchestration (Azure Data Factory)

Azure Data Factory (ADF) acts as the central orchestrator, tying together the various Databricks notebooks into a cohesive and scheduled data pipeline.

- **Objective:** Design and implement an ADF pipeline to sequentially execute the Databricks notebooks, ensuring a smooth flow of data from raw ingestion to final forecast generation.
- **Key Steps and Technical Details:**
  - **Creation of ADF Pipeline:** A new pipeline (`pipeline1`) is created within the Azure Data Factory Studio.
  - **Adding Databricks Notebook Activities:**
    - Three **Notebook activities** are added to the pipeline canvas, each configured to point to a specific Databricks notebook:
      - `kaggletodadlsgen2` (for data ingestion into Bronze).
      - `runbronzetosilver` (for data transformation to Silver).
      - `silvertomlgold` (for forecasting and saving to Gold).
    - Each Notebook activity is configured with the appropriate Databricks linked service (connecting ADF to your Databricks workspace) and the path to the notebook.
  - **Chaining Activities:** Success dependencies (green arrows) are drawn between the Notebook activities. This ensures that `runbronzetosilver` only starts after `kaggletodadlsgen2` successfully completes, and `silvertomlgold` only starts after `runbronzetosilver` succeeds. This sequential execution is critical for maintaining data integrity across the medallion layers.
  - **Monitoring Pipeline Runs:** ADF provides comprehensive monitoring capabilities. The "Pipeline runs" view allows tracking the status, duration, and logs of each activity within the pipeline, enabling quick identification and troubleshooting of any failures.
- **Tools Used:** Azure Data Factory, Azure Databricks Notebook Activity.

Data Factory Homepage:

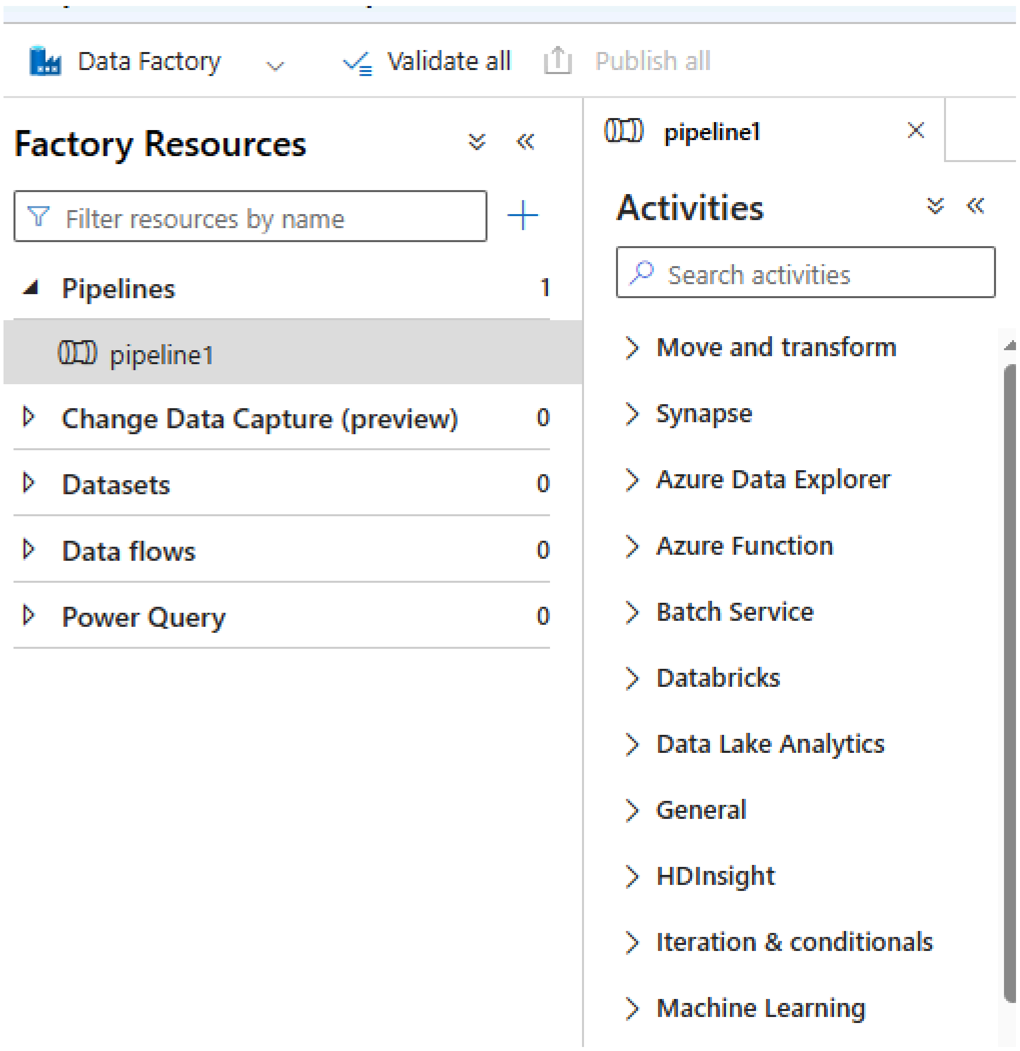

Microsoft Azure
Data Factory ▸ datafactoryecommerce
Search factory and documentation
azuser3618_mml.local@techademy.com
TECHADEMY LEARNING SOLUTIONS PRIVATE LIMITED

Would you like to see Data Factory inside of Microsoft Fabric, Microsoft's newest cloud-first data analytics SaaS platform? Click here to get started with Fabric Data Factory!

Azure Data Factory allows you to configure a Git repository with either Azure DevOps or GitHub. Git is a version control system that allows for easier change tracking and collaboration. Learn more

Set up code repository
Data factory
datafactoryecommerce
New
Ingest
Copy data at scale once or on a schedule.
Orchestrate
Code-free data pipelines.
Transform data
Transform your data using data flows.
Configure SSIS
Manage & run your SSIS packages in the cloud.
Recent resources

Creating a Pipeline to orchestrate workflow:


Data Factory
Validate all
Publish all

Factory Resources

Filter resources by name

Pipelines	1
pipeline1	
Change Data Capture (preview)	0
Datasets	0
Data flows	0
Power Query	0

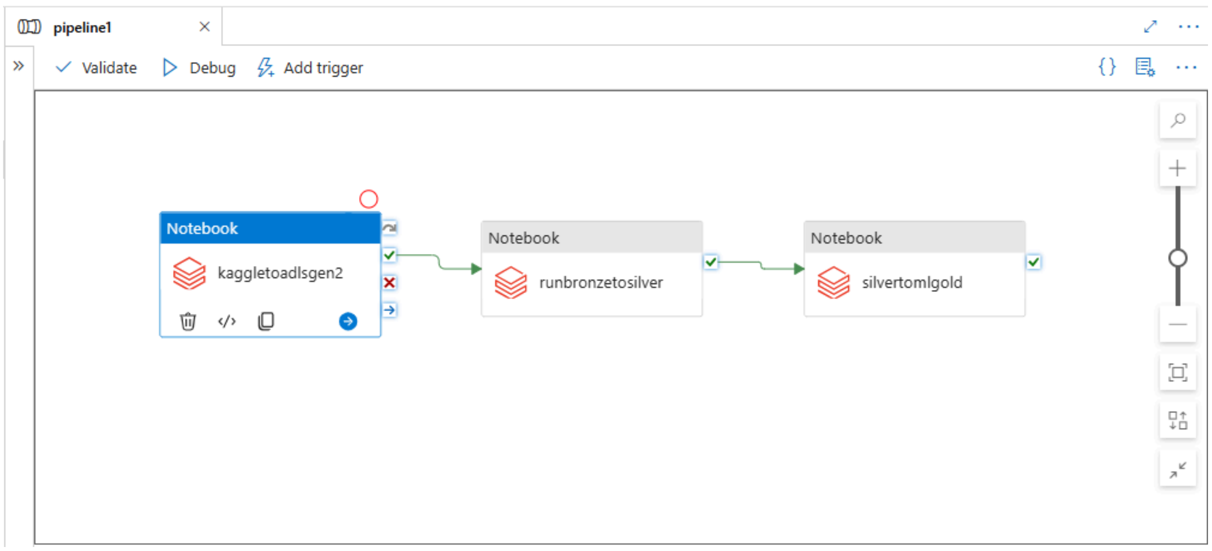
pipeline1

Activities

Search activities

- Move and transform
- Synapse
- Azure Data Explorer
- Azure Function
- Batch Service
- Databricks
- Data Lake Analytics
- General
- HDInsight
- Iteration & conditionals
- Machine Learning

Connecting the 3 notebooks from databricks to Azure Data Factory:


pipeline1

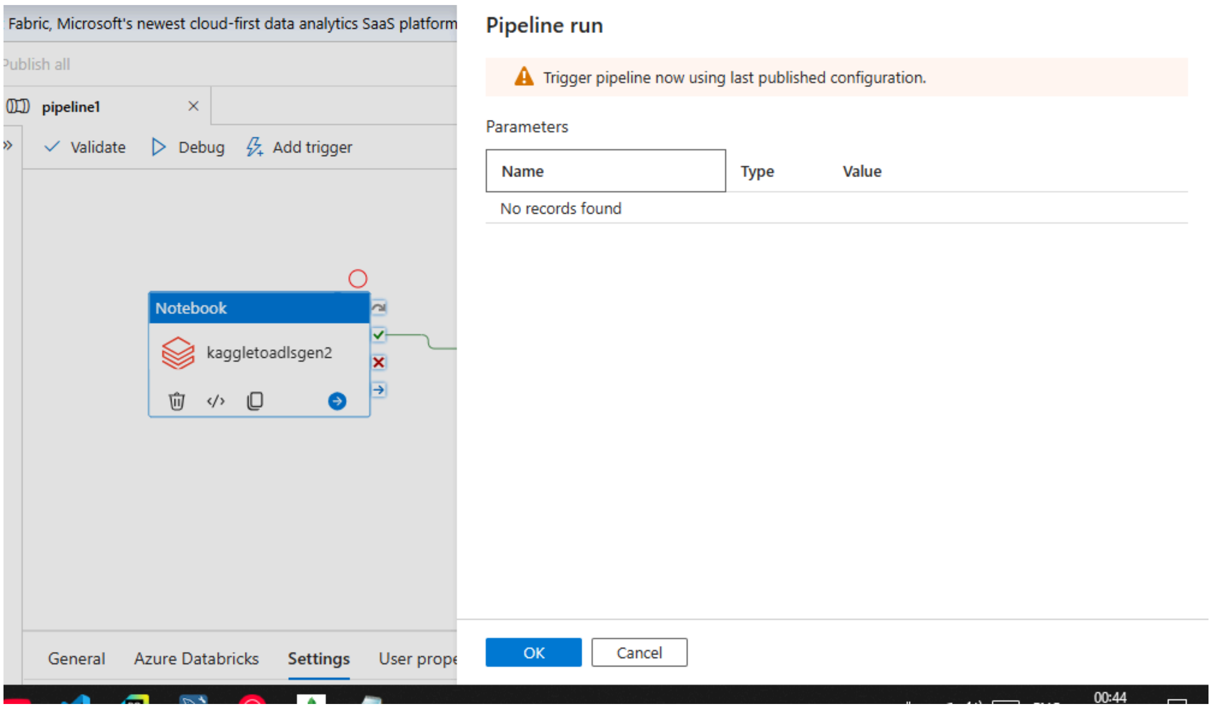
Validate
Debug
Add trigger

Notebook
kaggletoadlsgen2

Notebook
runbronzetosilver

Notebook
silvertomlgold

Running the pipeline:



Fabric, Microsoft's newest cloud-first data analytics SaaS platform

Publish all

pipeline1

Validate Debug Add trigger

Notebook

kaggletoadlsgen2

General Azure Databricks Settings User prope

Pipeline run

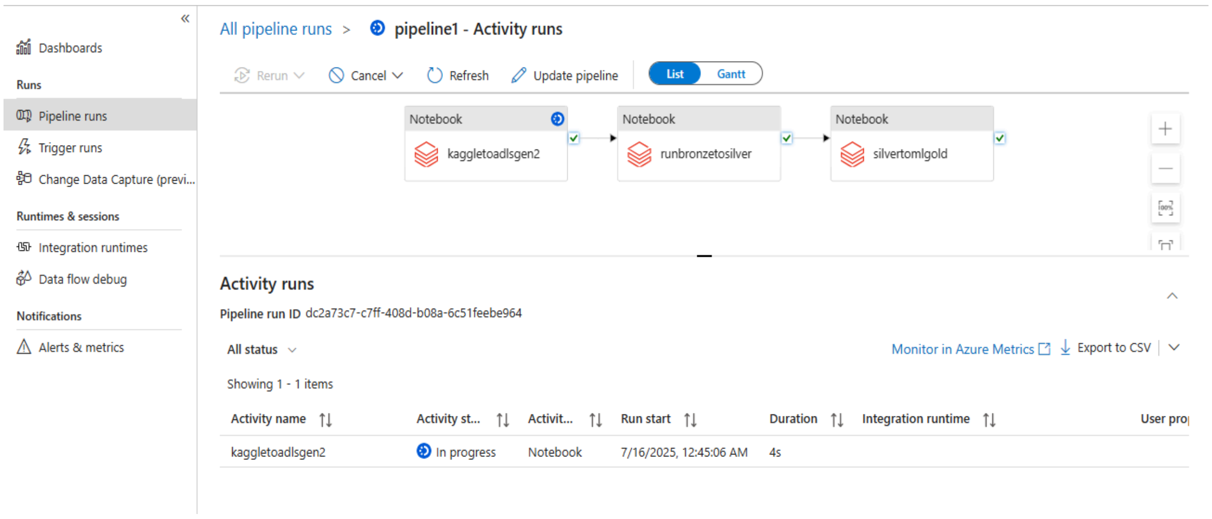
Trigger pipeline now using last published configuration.

Parameters

Name	Type	Value
No records found		

OK Cancel

Created Pipeline is running:



Activity runs

Pipeline run ID dc2a73c7-c7ff-408d-b08a-6c51feebe964

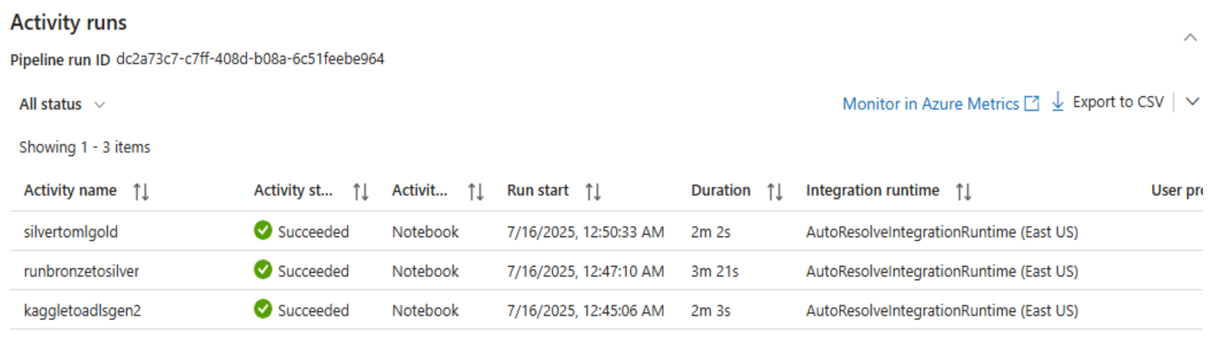
All status

Monitor in Azure Metrics Export to CSV

Showing 1 - 1 items

Activity name	Activity st...	Activit...	Run start	Duration	Integration runtime	User pro
kaggletoadlsgen2	In progress	Notebook	7/16/2025, 12:45:06 AM	4s		

Pipeline Run is Successful:



Activity runs

Pipeline run ID dc2a73c7-c7ff-408d-b08a-6c51feebe964

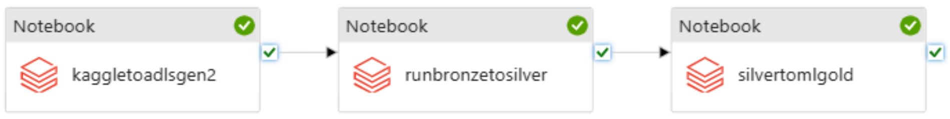
All status

Monitor in Azure Metrics | Export to CSV

Showing 1 - 3 items

Activity name	Activity st...	Activit...	Run start	Duration	Integration runtime	User pr
silvertomlgold	✓ Succeeded	Notebook	7/16/2025, 12:50:33 AM	2m 2s	AutoResolveIntegrationRuntime (East US)	
runbronzetosilver	✓ Succeeded	Notebook	7/16/2025, 12:47:10 AM	3m 21s	AutoResolveIntegrationRuntime (East US)	
kaggletoadlsgen2	✓ Succeeded	Notebook	7/16/2025, 12:45:06 AM	2m 3s	AutoResolveIntegrationRuntime (East US)	

Succeeded Pipeline:

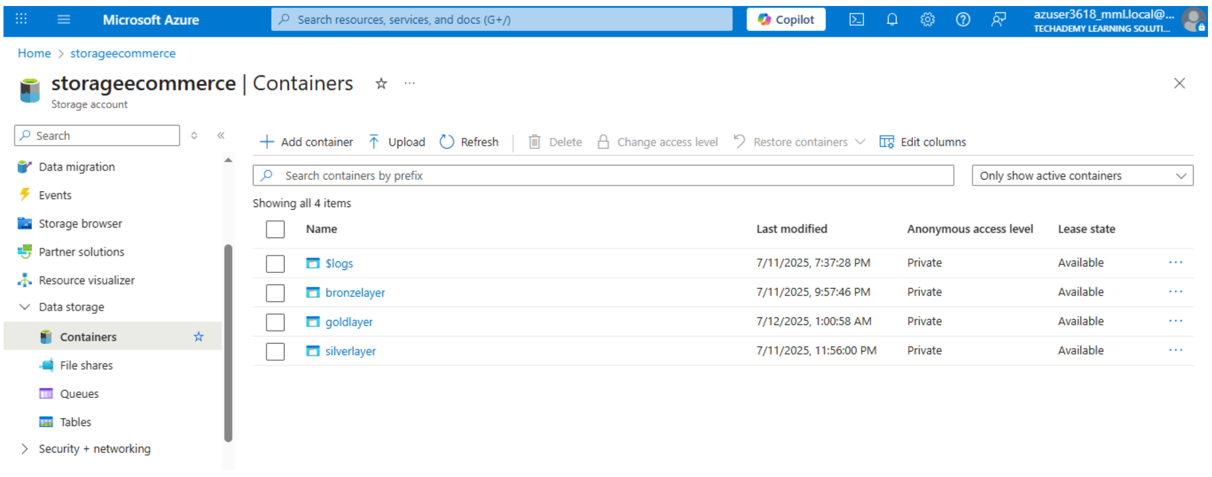


3.6. Data Storage Overview (Azure Blob Storage ADLS Gen 2)

The project leverages Azure Blob Storage, specifically its Data Lake Storage Gen2 capabilities, to implement a robust **medallion architecture**. This layered approach improves data quality, manageability, and accessibility.

- **Objective:** Store data in distinct layers (Bronze, Silver, Gold) within Azure Blob Storage, optimizing for different stages of data processing and consumption.
- **Key Layers and Their Purpose:**
    - **Bronze Layer (`bronzelayer` container):**
        - **Purpose:** Raw, immutable landing zone for source data. Data is stored in its original format (e.g., CSVs downloaded from Kaggle).
        - **Benefits:** Provides data lineage, auditability, and allows for reprocessing from raw sources if transformations need to change. It's the "single source of truth" for raw data.
    - **Silver Layer (`silverlayer` container):**
        - **Purpose:** Refined, cleaned, and conformed data. Data undergoes transformations, deduplication, and aggregation (e.g., `daily_demand` in Delta format).
        - **Benefits:** Ready for analytical use cases and feature engineering. Data is consistent and reliable. The use of **Delta Lake format** (`.delta` extension) provides ACID transactions, schema enforcement, schema evolution, and time travel capabilities, significantly enhancing data reliability and performance for Spark-based workloads.
    - **Gold Layer (`goldlayer` container):**
        - **Purpose:** Highly curated, aggregated, and business-ready data, optimized for direct consumption by business intelligence tools and applications. This includes the final demand forecasts.
        - **Benefits:** Provides a simplified view for end-users, optimized for reporting and dashboarding. Forecasts are available in both **Delta** (for programmatic access) and **CSV** (for broad compatibility) formats.
- **Tools Used:** Azure Blob Storage Containers, Delta Lake format.

Containers inside Storage account (Created by Automation):

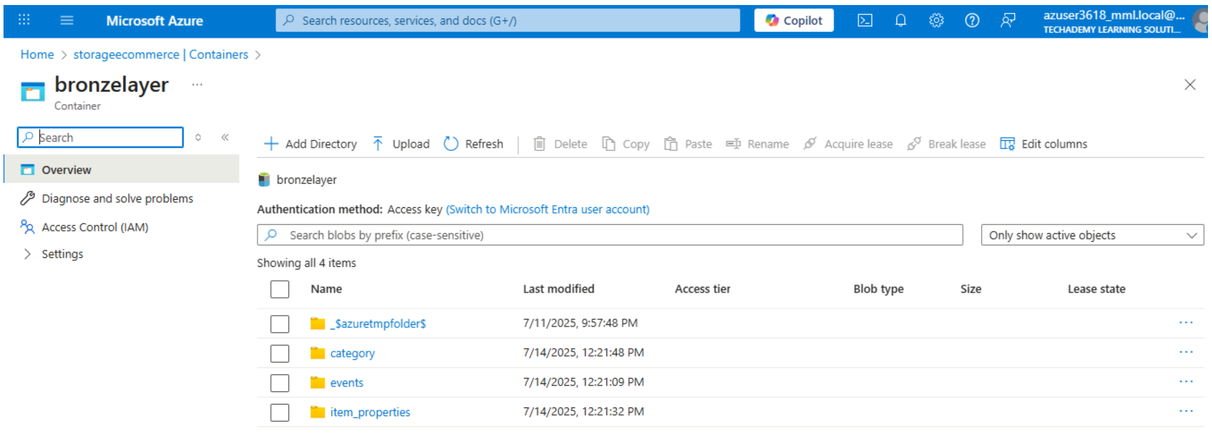


storageecommerce | Containers

Showing all 4 items

Name	Last modified	Anonymous access level	Lease state
$logs	7/11/2025, 7:37:28 PM	Private	Available
bronzelayer	7/11/2025, 9:57:46 PM	Private	Available
goldlayer	7/12/2025, 1:00:58 AM	Private	Available
silverlayer	7/11/2025, 11:56:00 PM	Private	Available

Files inside bronze layer:



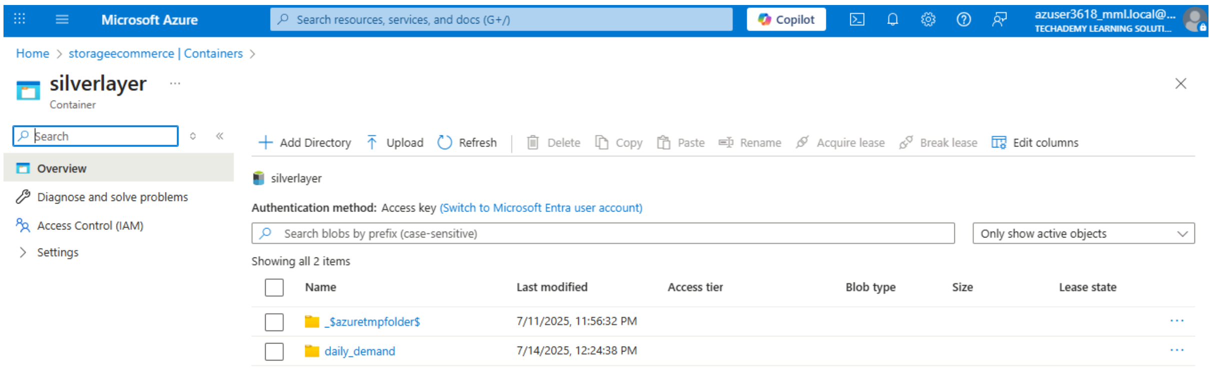
bronzelayer

Authentication method: Access key (Switch to Microsoft Entra user account)

Showing all 4 items

Name	Last modified	Access tier	Blob type	Size	Lease state
_$azuretmpfolder$	7/11/2025, 9:57:48 PM				
category	7/14/2025, 12:21:48 PM				
events	7/14/2025, 12:21:09 PM				
item_properties	7/14/2025, 12:21:32 PM				

Files inside silver layer:



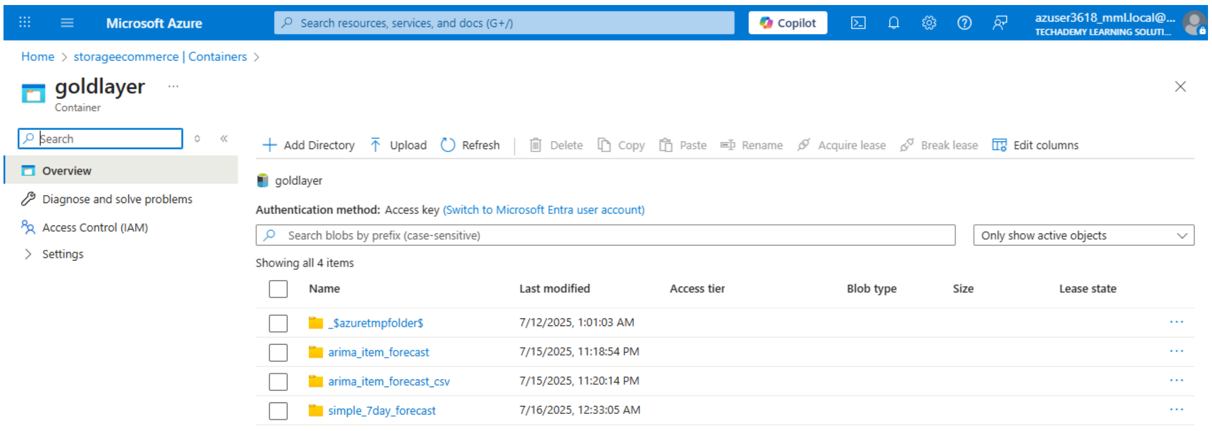
silverlayer

Authentication method: Access key (Switch to Microsoft Entra user account)

Showing all 2 items

Name	Last modified	Access tier	Blob type	Size	Lease state
_$azuretmpfolder$	7/11/2025, 11:56:32 PM				
daily_demand	7/14/2025, 12:24:38 PM				

Files inside Gold layer:



Authentication method: Access key (Switch to Microsoft Entra user account)

Search blobs by prefix (case-sensitive) | Only show active objects

Showing all 4 items

Name	Last modified	Access tier	Blob type	Size	Lease state
_$azuretmpfolder$	7/12/2025, 1:01:03 AM				
arima_item_forecast	7/15/2025, 11:18:54 PM				
arima_item_forecast_csv	7/15/2025, 11:20:14 PM				
simple_7day_forecast	7/16/2025, 12:33:05 AM				

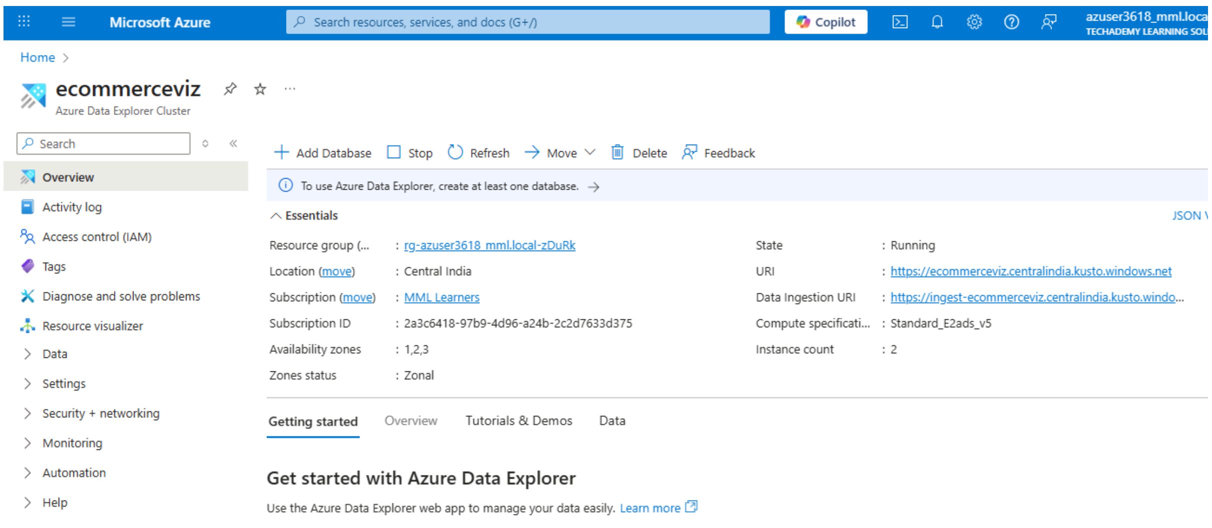
3.7. Azure Data Explorer:(Optional: Additional Functionality):

The final phase focuses on making the generated demand forecasts accessible and actionable through interactive visualizations. Azure Data Explorer (ADX) is chosen for its speed, scalability, and powerful Kusto Query Language (KQL).

- **Objective:** Ingest the forecasted data from the Gold Layer into an Azure Data Explorer cluster and create interactive dashboards for business insights.
- **Key Steps and Technical Details:**
    - **Azure Data Explorer Cluster and Database Setup:** An ADX cluster (`ecommerceviz.centralindia`) and a dedicated database (`forecasted db`) are provisioned in Azure. The cluster is optimized for high-volume, high-velocity data ingestion and querying.
    - **Data Ingestion from Gold Layer:**
        - The forecasted data (CSV or Parquet files from `goldlayer/arima_item_forecast_csv/` or `goldlayer/arima_item_forecast/`) is ingested into a new table (e.g., `ARIMAForecasts`) within the `forecasted db` using the ADX Web UI's ingestion wizard.
        - **Crucial Troubleshooting:** A common issue encountered was garbled output during initial queries, indicating incorrect data parsing. This was resolved by meticulously ensuring the **correct file format (CSV or Parquet)** was selected in the ADX ingestion wizard, matching the actual format of the files in the Azure Storage Gold layer. This ensures ADX correctly interprets the file structure.
    - **Querying Data with Kusto Query Language (KQL):**
        - In the ADX Web UI's "Query" tab, KQL is used to interact with the ingested data.
        - Basic queries like `ARIMAForecasts | take 10` are used to inspect the raw data and confirm successful ingestion.
        - **Analytical Queries:** KQL's powerful operators are used for aggregations and transformations, such as `summarize AverageForecast = avg(yhat) by group` to calculate average forecasts per item.
    - **Creating Visualizations:**
        - The `render` operator in KQL is used to transform query results into various chart types directly within the ADX Web UI. For instance, `| render barchart with (title="Average Forecasted Sales per Item", yaxis="Average Forecast", xaxis="Item ID")` creates a clear bar chart.
        - While the `ds` (date) column was removed from the final output for simplicity in the Python script, if it were present and correctly ingested as `datetime` type, KQL's `timechart` would be used to visualize trends over time.
    - **Dashboarding:**
        - Individual visualizations are pinned to new or existing Azure Data Explorer Dashboards.

    - These dashboards provide an interactive, shareable interface for business users to monitor demand forecasts without needing direct access to the underlying data or query language. They can be configured with auto-refresh for near real-time insights.
- **Tools Used:** Azure Data Explorer (Kusto Web UI), Kusto Query Language (KQL), ADX Dashboards.

Creating Data Explorer named "ecommerceviz":



ecommerceviz
Azure Data Explorer Cluster

To use Azure Data Explorer, create at least one database.

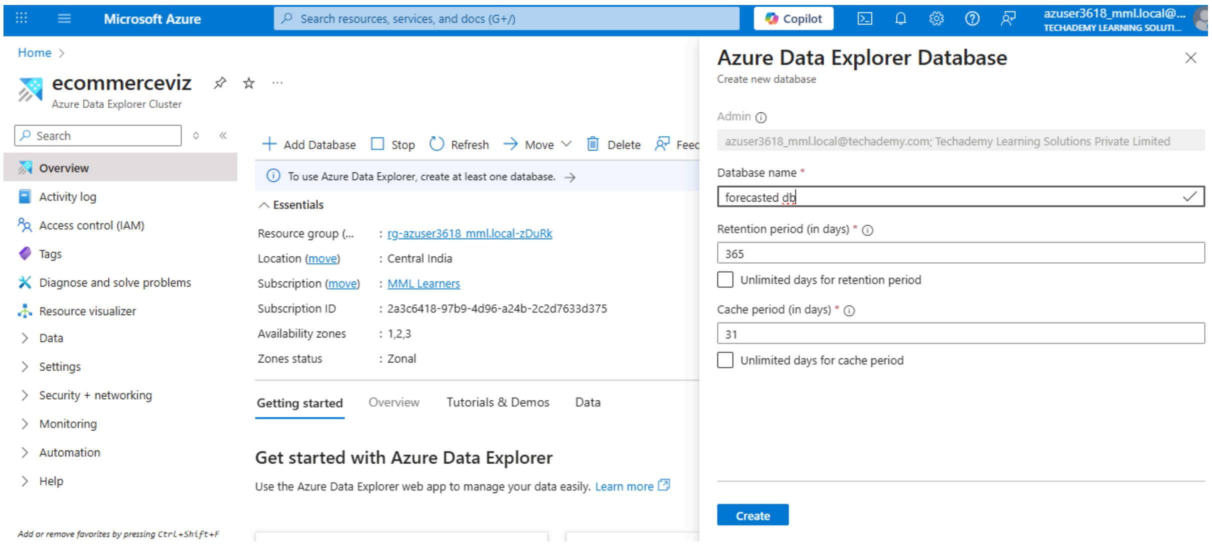
Essentials

Resource group (...	: rg-azuser3618_mml.local-zDuRk	State	: Running
Location (move)	: Central India	URI	: https://ecommerceviz.centralindia.kusto.windows.net
Subscription (move)	: MML Learners	Data Ingestion URI	: https://ingest-ecommerceviz.centralindia.kusto.windo...
Subscription ID	: 2a3c6418-97b9-4d96-a24b-2c2d7633d375	Compute specificati...	: Standard_E2ads_v5
Availability zones	: 1,2,3	Instance count	: 2
Zones status	: Zonal		

Get started with Azure Data Explorer

Use the Azure Data Explorer web app to manage your data easily. Learn more

Creating a database inside ecommerceviz
:



ecommerceviz
Azure Data Explorer Cluster

To use Azure Data Explorer, create at least one database.

Essentials

Resource group (...	: rg-azuser3618_mml.local-zDuRk
Location (move)	: Central India
Subscription (move)	: MML Learners
Subscription ID	: 2a3c6418-97b9-4d96-a24b-2c2d7633d375
Availability zones	: 1,2,3
Zones status	: Zonal

Get started with Azure Data Explorer

Use the Azure Data Explorer web app to manage your data easily. Learn more

Add or remove favorites by pressing Ctrl+Shift+F

Azure Data Explorer Database
Create new database

Admin
azuser3618_mml.local@techademy.com; Techademy Learning Solutions Private Limited

Database name *
forecasted db

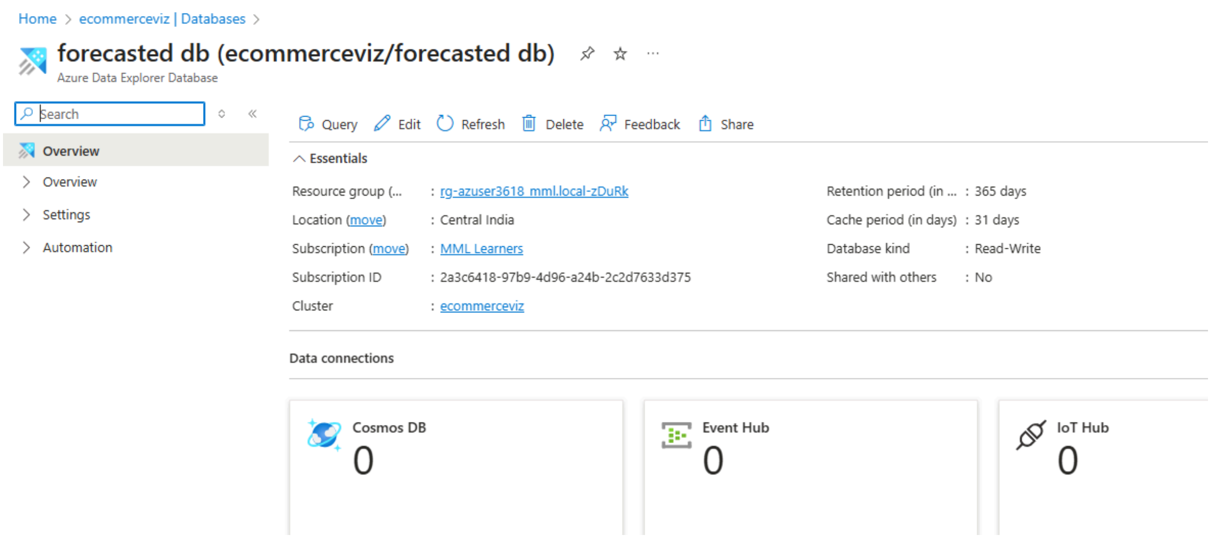
Retention period (in days) *
365

☐ Unlimited days for retention period

Cache period (in days) *
31

☐ Unlimited days for cache period

Created Database:


Home > ecommerceviz | Databases >

forcasted db (ecommerceviz/forecasted db)

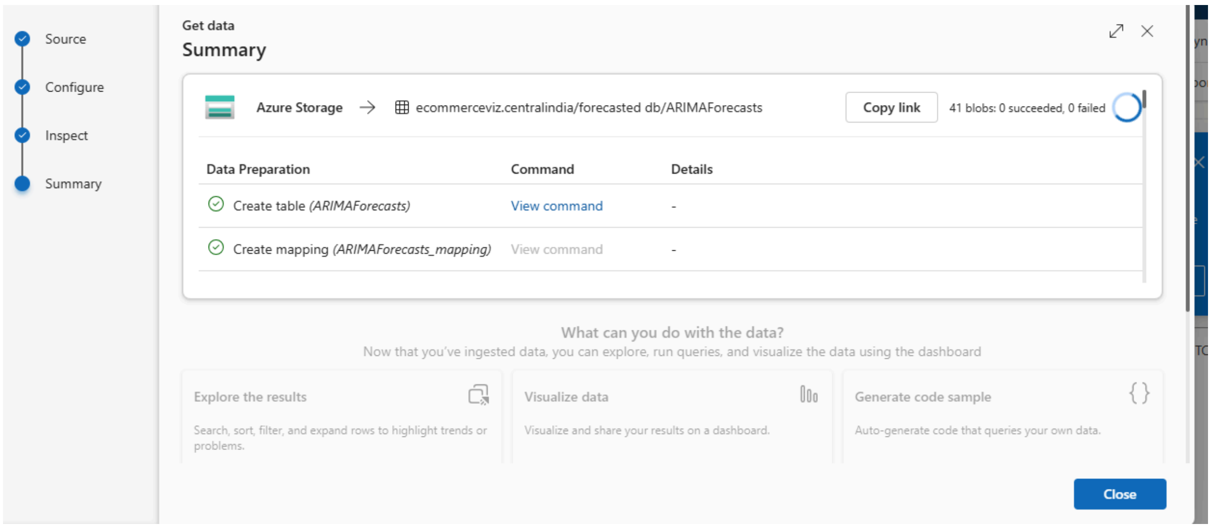
Azure Data Explorer Database

Resource group (...	: rg-azuser3618_mml.local-zDuRk
Location (move)	: Central India
Subscription (move)	: MML Learners
Subscription ID	: 2a3c6418-97b9-4d96-a24b-2c2d7633d375
Cluster	: ecommerceviz
Retention period (in ...	: 365 days
Cache period (in days)	: 31 days
Database kind	: Read-Write
Shared with others	: No

Data connections

Cosmos DB	Event Hub	IoT Hub
0	0	0

Ingesting the dataset from Gold layer:


Get data

Summary

Azure Storage → ecommerceviz.centralindia/forecasted db/ARIMAForecasts

41 blobs: 0 succeeded, 0 failed

Data Preparation	Command	Details
Create table *(ARIMAForecasts)*	View command	-
Create mapping *(ARIMAForecasts_mapping)*	View command	-

What can you do with the data?

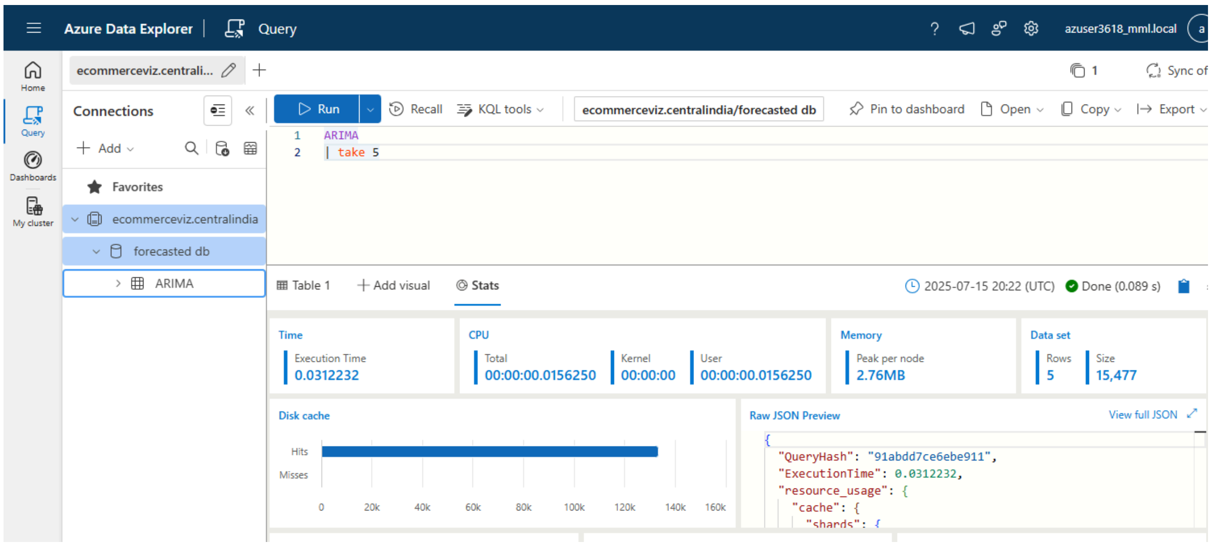
Now that you've ingested data, you can explore, run queries, and visualize the data using the dashboard

Explore the results — Search, sort, filter, and expand rows to highlight trends or problems.

Visualize data — Visualize and share your results on a dashboard.

Generate code sample — Auto-generate code that queries your own data.

Fetching first 5 rows of dataset:



```
ARIMA
| take 5
```

2025-07-15 20:22 (UTC) Done (0.089 s)

Time	CPU			Memory	Data set	
Execution Time	Total	Kernel	User	Peak per node	Rows	Size
0.0312232	00:00:00.0156250	00:00:00	00:00:00.0156250	2.76MB	5	15,477

Disk cache

```
{
  "QueryHash": "91abdd7ce6ebe911",
  "ExecutionTime": 0.0312232,
  "resource_usage": {
    "cache": {
      "shards": {
```

4. Conclusion

This project successfully implements a comprehensive and robust end-to-end data and MLOps pipeline for e-commerce demand forecasting within the Azure cloud. By leveraging Azure Databricks for scalable data processing, MLflow for experiment tracking and model management, Azure DevOps for automated CI/CD, Azure Data Factory for orchestration, and Azure Data Explorer for visualization, the solution provides:

- **Scalable Data Processing:** Efficient handling of large volumes of e-commerce data through a medallion architecture.
- **Automated Machine Learning Lifecycle:** Streamlined training, versioning, and promotion of forecasting models, reducing manual effort and improving reliability.
- **Actionable Insights:** Accessible and interactive dashboards that empower business users to make informed decisions regarding inventory, supply chain, and sales strategies.
- **Reproducibility and Auditability:** Through versioned data layers and MLflow tracking, the entire process is transparent and reproducible.

This end-to-end solution serves as a strong framework for managing and deploying machine learning models in a production environment, driving significant value for e-commerce operations.

Code used in this project

mountingkaggle.py

```
%pip install kaggle
import shutil, os

#  Use the corrected uploaded file path
uploaded_path = "/dbfs/FileStore/tables/kaggle_json-1.json"
target_path = "/root/.kaggle/kaggle.json"

# Ensure the ~/.kaggle directory exists
os.makedirs("/root/.kaggle", exist_ok=True)

#  Copy the file to the Kaggle expected location
shutil.copy(uploaded_path, target_path)

#  Set required permissions
os.chmod(target_path, 600)

print(" kaggle.json is moved and configured successfully!")
```

```
import shutil, os

# Make sure the config directory exists
os.makedirs("/root/.config/kaggle", exist_ok=True)

# Copy kaggle.json to the expected location
shutil.copy("/root/.kaggle/kaggle.json", "/root/.config/kaggle/kaggle.json")

# reset permissions again
os.chmod("/root/.config/kaggle/kaggle.json", 600)

print(" kaggle.json copied to /root/.config/kaggle/")
```

```
Import kaggle

# Re-authenticate
kaggle.api.authenticate()

# Download RetailRocket dataset
kaggle.api.dataset_download_files(
            ‘retailrocket/ecommerce-dataset’,
            path=’/dbfs/tmp/’,
            unzip=False
)

print(“Dataset downloaded succesfully”)




import zipfile

zip_path = "/dbfs/tmp/ecommerce-dataset.zip"
extract_path = "/dbfs/tmp/ecommerce_dataset"

# Unzip all contents
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print(" Files extracted to", extract_path)







# Set your Azure Storage config
storage_account_name = "storageecommerce"
storage_account_key =
"CLZSIPRieeBvBCfFN8yljWxg9Tp28faTS2nlO9LrBnfA0tFgc5bW0XZKAPFLhO1XZkBymz8
0mZsq+AStuywHWg=="

# Spark config to allow writing to ADLS Gen2
spark.conf.set(
  f"fs.azure.account.key.{storage_account_name}.blob.core.windows.net",
  storage_account_key
)

# Define local-to-ADLS path mappings
files_to_upload = {
    "events.csv": "events",
    "item_properties_part1.csv": "item_properties",
    "item_properties_part2.csv": "item_properties",
    "category_tree.csv": "category"
}

# Upload each CSV to the correct Bronze folder
```

```
for filename, folder in files_to_upload.items():
    local_path = f"/dbfs/tmp/ecommerce_dataset/{filename}"
    adls_path =
f"wasbs://bronzelayer@{storage_account_name}.blob.core.windows.net/{folder}/{filename}"

    print(f" Uploading {filename} → {adls_path}")
    dbutils.fs.cp(f"file:{local_path}", adls_path)
```

bronzetosilver.py

```
# Set storage configs

storage_account_name = "storageecommerce”
storage_account_key =
"CLZSIPRieeBvBCFFN8y1jwxg9Tp28faTS2n109LrBnfA8tFgc5bw@XZKAPFLhO1XZkBy@z
8@mZsq+AStuywtwg="
spark.conf.set(
"fs.azure.account.key.(storage_account_name}.blob.core.windows.net",
storage_account_key
)

# Bronze ADLS paths

Base = f"wasbs://bronzelayer@{storage_account_name}.blob.core.windows.net"
events_path = f"{base}/events/events.csv"
cat_path= f{base}/category/category_tree.csv"
prop1_path= f"{base}/item_properties/item_properties_part1.csv"
prop2_path = f"{base}/item_properties/item_properties_part2.csv"

# Read the files

events_df spark.read.option("header", True).csv(events_path)
category_df spark.read.option("header", True).csv(cat_path)
item_props1_df spark.read.option("header", True).csv(prop1_path)
item_props2_df spark.read.option("header", True).csv (prop2_path)




from pyspark.sql.functions import col, to_date, count, max

# Combine both item_properties parts
item_props_df = item_props1_df.union(item_props2_df)

# Remove duplicates
item_props_df = item_props_df.dropDuplicates(["itemid", "property", "timestamp"])
```

```
# Filter only transactions from events.csv
transactions_df = events_df.filter(col("event") == "transaction")

# Convert timestamp to date for daily grouping
transactions_df = transactions_df.withColumn("event_date", to_date("timestamp"))

#  Extract item-category mapping from item_properties
# Assuming rows where `property == 'categoryid'` contain the actual categoryid in `value`
item_category_df = item_props_df.filter(col("property") == "categoryid") \
    .select(col("itemid"), col("value").alias("categoryid"))

# Join transactions with item-category mapping
tx_item_df = transactions_df.join(item_category_df, on="itemid", how="left")

# Join with category_tree to get full hierarchy (optional)
tx_item_cat_df = tx_item_df.join(category_df, on="categoryid", how="left")

#  Group by item and date to get daily demand
daily_demand_df = tx_item_cat_df.groupBy("itemid", "event_date").agg(
    count("*").alias("daily_sales_count"),
    max("categoryid").alias("categoryid")
)

# Show result
daily_demand_df.show(10)

#  Save to Silver Zone
silver_path =
f"wasbs://silver@{storage_account_name}.blob.core.windows.net/daily_demand/"

daily_demand_df.write.format("delta").mode("overwrite").save(silver_path)




# Writing silverlayer to ADLS
silver_path =
f"wasbs://silverlayer@{storage_account_name}.blob.core.windows.net/daily_demand/"

daily_demand_df.write.format("delta").mode("overwrite").save(silver_path)

print(" Silver layer written to ADLS!")
```

silvertomlgold.py

Using ARIMA ML model:

```
from pyspark.sql.functions import col, to_date, sum as spark_sum, count as spark_count
from pyspark.sql.types import StructType, StructField, StringType, DateType, DoubleType
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
import datetime

# Azure Storage Config
storage_account_name = "storageecommerce"
storage_account_key =
"CLZSIPRieeBvBCfFN8yljWxg9Tp28faTS2nlO9LrBnfA0tFgc5bW0XZKAPFLhO1XZkBymz8
0mZsq+AStuywHWg=="

spark.conf.set(
    f"fs.azure.account.key.{storage_account_name}.blob.core.windows.net",
    storage_account_key
)

# Define Paths
silver_path      =
f"wasbs://silverlayer@{storage_account_name}.blob.core.windows.net/daily_demand/"
gold_path        =
f"wasbs://goldlayer@{storage_account_name}.blob.core.windows.net/arima_item_forecast/"
gold_csv_path    =
f"wasbs://goldlayer@{storage_account_name}.blob.core.windows.net/arima_item_forecast_
csv/"

# Load Silver Layer
df = spark.read.format("delta").load(silver_path).select("itemid", "event_date",
"daily_sales_count")

# Daily sales per item per day
daily_df = df.groupBy("event_date", "itemid") \
    .agg(spark_sum("daily_sales_count").alias("y")) \
    .withColumn("ds", to_date("event_date")) \
    .select("itemid", "ds", "y") \
    .withColumnRenamed("itemid", "group")

#  Find top 5 most active items
item_day_counts = daily_df.groupBy("group").agg(spark_count("ds").alias("active_days"))
top_items = item_day_counts.orderBy(col("active_days").desc()).limit(5).rdd.map(lambda
row: row["group"]).collect()
print(" Top active itemids:", top_items)

# Filter for top items
filtered_df = daily_df.filter(col("group").isin(top_items))
```

```
# Define ARIMA forecast schema
schema = StructType([
   StructField("group", StringType(), True),
   StructField("ds", DateType(), True), # Expecting Python datetime.date objects
   StructField("yhat", DoubleType(), True)
])

#Define ARIMA forecast 'schema
from pyspark.sql.types import TimestampType
schema=StructType([
         StructField("group", StringType(), True),
         StructField("ds", Timestamp Type(), True),
         StructField("yhat", DoubleType(), True)
])

#ARIMA Forecast Function (with MLflow)
def arima_forecast(pdf: pd.DataFrame) -> pd.DataFrame:
         import mlflow
         import mlflow.sklearn
         group = str(pdf.iloc[0]["group"])
         try:
             pdf['ds'] = pd.to_datetime(pdf['ds'])
             ts = pdf.sort_values("ds").set_index("ds")["y"]

# Train ARIMA
model = ARIMA(ts, order(1, 1, 1))
model_fit = model.fit()
forecast = model_fit.forecast(steps=7)
last_date = ts.index[-1]
future_dates = [last_date + pd.Timedelta (days=1) for i in range(1, 8)]
result = pd.DataFrame({
         "group": [group] 7,
         "ds": future_dates,
         "yhat": forecast.values
})

#MLflow Tracking
with mlflow.start_run(run_name = "ARIMA_{group}"):
         mlflow.log_param("model", "ARIMA")
         mlflow.log_param("order", "(1,1,1)")
         mlflow.log_param("group", group)
         mlflow.log_metric("train count", len(ts))
         mlflow.sklearn.log model(model_fit, f"model_arima_{group}")
         print(f" Model for item {group} logged to MLflow.")

return result
```

```
except Exception as e:

        print(f" ARIMA failed for item (group): ()")
        return pd.DataFrame(columns["group", "ds", "yhat"])


# Apply ARIMA per item
forecast_df = filtered_df.groupBy("group").applyInPandas(arima_forecast, schema=schema)

#  Final cast for ds column (might be redundant but safe to keep)
forecast_df = forecast_df.withColumn("ds", col("ds").cast("date"))


# Save to Gold Layer (Delta + CSV)
forecast_df.write.format("delta").mode("overwrite").save(gold_path)
forecast_df.write.option("header", "true").mode("overwrite").csv(gold_csv_path)

print(" ARIMA item-level forecast saved to Gold Layer.")
forecast_df.show()
```

Simple moving average method:

```
from pyspark.sql.functions import col, avg, lit
import datetime

# Config
storage_account_name = "storageecommerce"
storage_account_key =
"CLZSIPRieeBvBCFFN8y1jWxg9Tp28faTS2109LrBnAtFgc5bW8XZKPFLh01XZkBymz88mZ
sq+AStuywHWg=="
spark.conf.set(
        f"fs.azure.account.key.{storage_account_name}.blob.core.windows.net",
        storage_account_key
)
silver_path =
f"wasbs://silverlayer@{storage_account_name}.blob.core.windows.net/daily_demand/"
gold_path =
f"wasbs://goldlayer@{storage_account_name}.blob.core.windows.net/simple_7day_forecast/
"

# Load data
df = spark.read.format("delta").load(silver_path).select("itemid", "event_date",
"daily_sales_count")

# Get top 5 items (any frequency)
top_items= df.groupBy("itemid").count().orderBy(col("count").desc()).limit(5) \
.rdd.flatMap(lambda x: [x["itemid"]]).collect()
```

```
# Calculate average daily sales for each item
avg_df = df.filter(col("itemid").isin(top_items)) \
.groupBy("itemid") \
.agg(avg("daily_sales_count").alias("yhat"))

# Generate next 7 days for each item
today = datetime.date.today() future_rows = []
for row in avg_df.collect():
        item = row["itemid"]
        yhat = row["yhat"]
        for i in range(1, 8):
                forecast_date = today + datetime.timedelta(days=i)
                future_rows.append((item, forecast_date.strftime('%Y-%m-%d'), yhat))

# Create DataFrame and save
future_df = spark.createDataFrame(future_rows, ["itemid", "ds", "yhat"])
future_df.write.format("delta").mode("overwrite").save(gold_path)
print(" 7-day naive forecast saved to Gold layer.")

forecast_df = spark.read.format("delta").load(gold_path)
display(forecast_df)
```

azure-pipelines.yml

```
trigger:
- main

pool:
  vmImage: ubuntu-latest

variables:
  DATABRICKS_HOST:
'https://adb-2340809536712597.17.azuredatabricks.net/?o=2340809536712597'
  DATABRICKS_TOKEN: 'dapi1d826d409db7ed1b9669c8bec93e9a4e'
  MODEL_NAME: 'arima_demand_forecast'
  MODEL_VERSION: '1'
  STAGE: 'Production'

jobs:
- job: PromoteModel
  steps:
  - script: |
      curl -X POST "$DATABRICKS_HOST/api/2.0/mlflow/model-versions/transition-stage" \
        -H "Authorization: Bearer $DATABRICKS_TOKEN" \
    -H "Content-Type: application/json" \
    -d '{
          "name": "'"$MODEL_NAME"'",
          "version": "'"$MODEL_VERSION"'",
          "stage": "'"$STAGE"'",
          "archive_existing_versions": false
        }'
displayName: 'Promote Model to Production'
```