Logistics Cost & Delivery Analytics Platform

**Capstone Project Group 1**

1.Project Overview

The Logistics Cost & Delivery Analytics project is an end-to-end data engineering capstone designed to simulate a real-world enterprise logistics analytics platform. The project focuses on building a scalable, reliable, and governed data pipeline capable of transforming raw operational shipment data into meaningful business insights for leadership and operations teams.

The solution processes daily shipment extracts generated by regional warehouses and integrates them with reference data such as warehouses, carriers, and regions. These datasets contain common real-world data challenges including missing values, duplicate records, cost outliers, and delivery anomalies. Addressing these challenges is a core objective of the project, ensuring that downstream analytics are based on accurate and trustworthy data.

The architecture follows the Medallion pattern (Bronze, Silver, and Gold layers), which enables progressive data refinement and clear separation of responsibilities across ingestion, transformation, and analytics layers. Raw data is first ingested into the Bronze layer without modification to preserve source integrity. The Silver layer applies data cleansing, standardization, deduplication, and data quality validation using Delta Lake. Finally, the Gold layer presents a business-ready star schema optimized for reporting and analytics.

Automation and scalability are key design considerations in this project. Azure Data Factory orchestrates batch ingestion with incremental watermark logic to ensure efficient daily processing and rerun capability. Azure Databricks provides a distributed processing environment for transformations, data quality enforcement, and dimensional modeling. Power BI consumes the curated Gold layer to deliver interactive dashboards and KPIs for shipment cost governance, delivery performance, carrier SLA compliance, and data quality monitoring.

Overall, this project demonstrates the application of modern data engineering best practices using Azure-native services. It showcases how cloud-based analytics platforms can be designed to handle imperfect data, support incremental growth, and deliver actionable insights to business stakeholders in a logistics domain.

2.Introduction

In today’s data-driven logistics environment, organizations generate large volumes of operational data across warehouses, carriers, and regional networks. While this data holds significant potential for improving cost efficiency and delivery performance, it often arrives in fragmented formats with inconsistent quality, making reliable analytics difficult. Without a structured data engineering framework, leadership teams struggle to obtain accurate, timely, and trusted insights for decision-making.

This capstone project, **Logistics Cost & Delivery Analytics**, focuses on designing and implementing an end-to-end data engineering solution using modern Azure cloud services. The project addresses common challenges such as missing shipment costs, duplicate records, delivery anomalies, and performance bottlenecks in reporting systems. By applying industry-standard data architecture patterns, the solution ensures data reliability, scalability, and analytical readiness.

The project leverages the **Medallion Architecture (Bronze, Silver, and Gold layers)** to progressively refine raw logistics data into a clean, governed, and business-consumable format. Azure Data Factory is used for orchestrating batch ingestion with incremental watermark logic, Azure Data Lake Storage Gen2 acts as the central data repository, and Azure Databricks with Delta Lake enables scalable transformations, data quality enforcement, and dimensional modeling. Finally, Power BI is used to deliver interactive dashboards and a semantic model optimized for business users.

The primary objective of this project is to demonstrate how a robust data engineering pipeline can transform raw logistics data into meaningful insights such as shipment cost governance, delivery performance analysis, carrier SLA compliance, and data quality transparency. This solution closely mirrors real-world enterprise data platforms and showcases best practices commonly adopted in production-grade analytics systems.

3. Abstract

The **Logistics Cost & Delivery Analytics** capstone project presents a comprehensive cloud-based data engineering solution designed to analyze shipment costs, delivery performance, and data quality metrics across regional logistics operations. The project simulates a real-world logistics analytics scenario in which daily shipment data is ingested from multiple warehouses and processed using a scalable and governed architecture.

The solution is built using Azure Data Lake Storage Gen2, Azure Data Factory, Azure Databricks, Delta Lake, and Power BI. Raw CSV datasets are ingested into the Bronze layer through automated batch pipelines with incremental watermarking. The data is then cleansed, standardized, deduplicated, and validated in the Silver layer, where data quality rules identify null values, duplicates, outliers, and delivery anomalies. Invalid or non-conforming records are isolated into a quarantine zone to preserve data integrity.

In the Gold layer, the processed data is modeled into a star schema consisting of fact and dimension tables optimized for analytical workloads. Surrogate keys and incremental merge strategies are implemented to support efficient updates and high query performance. The curated dataset is consumed by Power BI to create interactive dashboards that provide insights into shipment cost trends, on-time delivery performance, carrier SLA compliance, fragile shipment handling, and overall data quality health.

This project demonstrates the practical application of modern data engineering principles, including Medallion Architecture, incremental data processing, data quality governance, and business intelligence integration. The final outcome is a production-ready analytics platform that delivers trusted, high-quality insights to stakeholders and serves as a strong foundation for enterprise-scale logistics analytics.

4. Business Problem Statement

Regional warehouses submit daily shipment data containing operational and financial metrics. However, leadership faced challenges due to:

* Missing or invalid shipment costs
* Duplicate shipment records
* Delivery date anomalies
* Lack of a centralized, trusted reporting dataset

Business Objectives

* Enforce shipment cost governance
* Track delivery performance by region, warehouse, and carrier
* Monitor fragile shipment handling and SLA compliance
* Provide a trusted semantic model for Power BI reporting

3. Solution Overview

To address these challenges, an automated data pipeline was built that:

1. Lands raw data in Azure Data Lake (Bronze)
2. Applies cleansing, deduplication, and data quality checks (Silver)
3. Models data into a star schema optimized for analytics (Gold)
4. Publishes Power BI dashboards for operational and executive insights

4. Architecture & Technology Stack

4.1 Azure Services Used

* Azure Data Lake Storage Gen2 – Raw and curated data storage
* Azure Data Factory – Orchestration, batch ingestion, watermarking
* Azure Databricks – Data transformations, Delta Lake, data quality logic
* Power BI – Semantic modeling and visualization

4.2 Medallion Architecture

* Bronze Layer: Raw CSV files partitioned by ingest date
* Silver Layer: Cleaned, typed Delta tables with DQ flags
* Gold Layer: Star schema (Fact & Dimensions) for BI consumption

5. Data Sources & Dataset Design

5.1 Source Files

* shipments\_daily.csv – Daily shipment transactions (~3,000 records)
* warehouses.csv – Warehouse master data
* carriers.csv – Carrier reference data
* regions.csv – Regional mapping
* delivery\_events\_microbatch.csv – Near-real-time delivery updates (optional)

5.2 Embedded Data Quality Challenges

* Null shipment costs
* Missing carrier identifiers
* Duplicate shipment IDs
* Cost outliers
* Invalid or missing delivery dates

6. Data Quality Framework

A structured data quality framework was implemented in the Silver layer.

Key Rules Applied

* Null shipment costs flagged and excluded from KPIs
* Duplicate shipment IDs deduplicated using latest ingest timestamp
* Cost outliers identified using percentile-based thresholds
* Date anomalies detected and flagged
* Invalid records routed to a quarantine zone

DQ Metrics Generated

* Null cost percentage
* Duplicate record count
* Outlier count
* Date anomaly count

7. Data Modeling (Gold Layer)

7.1 Dimension Tables

* DimDate
* DimWarehouse
* DimRegion
* DimCarrier (includes UNKNOWN carrier handling)

7.2 Fact Table

* FactShipments
  + Shipment cost
  + Delivery status
  + Fragile indicator
  + On-time delivery flag
  + Data quality indicators

Surrogate keys and incremental MERGE patterns were used to ensure scalability and performance.

8. Incremental Loading & Automation

* Azure Data Factory implements a watermark-based incremental load
* Control files track last successful ingestion date
* Pipelines support reruns and failure recovery
* Daily scheduled ingestion with logging and validation

9. Power BI Reporting & Analytics

9.1 Key KPIs

* Total Shipment Cost (excluding invalid records)
* Average Cost by Region, Warehouse, and Carrier
* Delivery Success Rate
* On-time Delivery Percentage
* Fragile Shipment Failure Rate
* Data Quality Metrics

9.2 Dashboard Pages

* Executive Overview
* Carrier Performance Analysis
* Warehouse & Regional Operations
* Data Quality & Freshness Monitoring

10. Project Execution Plan

- **System Architecture Overview**

The **Logistics Cost & Delivery Analytics** platform is built using a modern, cloud-native architecture on Microsoft Azure. The system is designed to support scalable data ingestion, robust data quality enforcement, incremental processing, and high-performance analytics. The architecture follows industry best practices and is aligned with the **Medallion Architecture pattern (Bronze, Silver, and Gold layers)** to ensure data reliability and maintainability.

The solution integrates multiple Azure services, each responsible for a specific role in the data lifecycle, from raw data ingestion to business intelligence reporting.

**High-Level Architecture Description**

At a high level, the system architecture consists of four main layers:

1. **Data Source Layer**
2. **Data Ingestion & Orchestration Layer**
3. **Data Processing & Storage Layer**
4. **Analytics & Visualization Layer**

Each layer is loosely coupled, enabling independent scaling, easier troubleshooting, and clear separation of responsibilities.

**Sprint 0: Architecture & Setup**

Sprint 0 focuses on establishing the foundational architecture required for building a scalable, enterprise-grade logistics analytics platform. This sprint covers data sourcing, cloud environment setup, storage organization, and initial service configuration using Azure-native components.

1. Data Source Layer

The data source layer represents the operational systems responsible for generating logistics data. In this project, regional warehouses act as the primary source systems, producing daily shipment extracts in CSV format. These files contain detailed shipment transaction records along with supporting reference data such as warehouse information, carrier details, and regional mappings.

In addition to scheduled batch data, optional micro-batch delivery event files are introduced to simulate near-real-time updates to shipment delivery statuses. This approach reflects real-world logistics environments where delivery confirmations and status changes may arrive asynchronously after shipment creation.

Data Sourcing Strategy

The original project specification provided approximately 3,000 records to represent 30 days of shipment activity. To better simulate enterprise-scale data processing and validate system scalability, the dataset was synthetically expanded to approximately 2.5 million records.

This data generation approach allowed the platform to be tested under higher data volumes while preserving realistic data characteristics such as:

Missing and null values

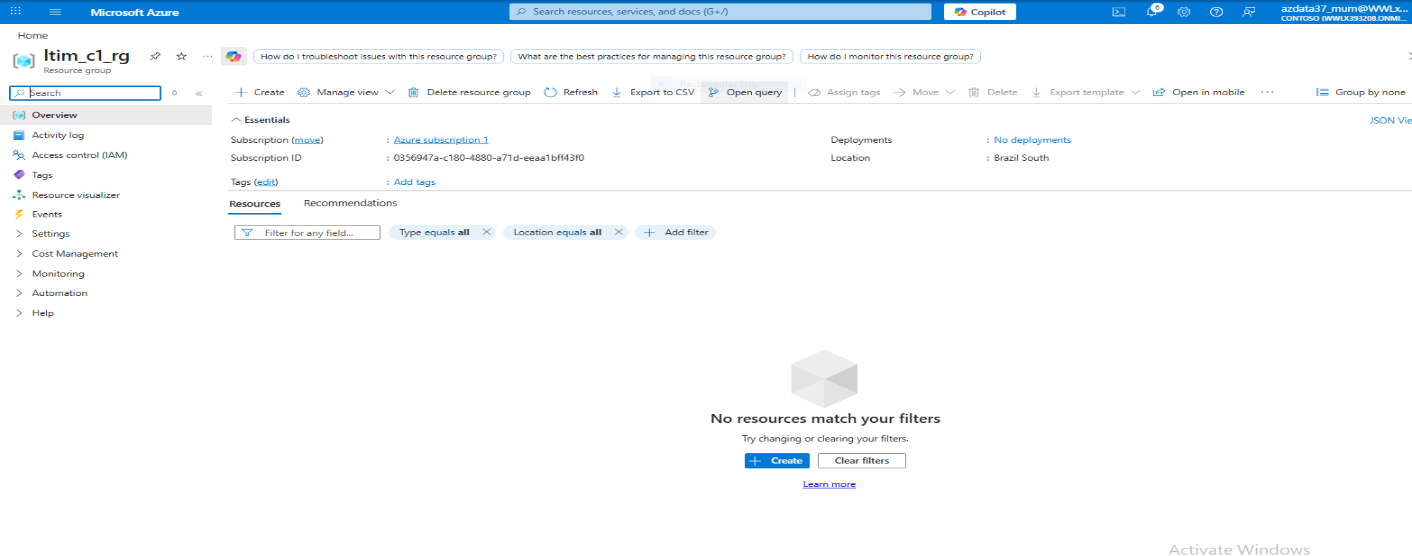
Duplicate shipment records

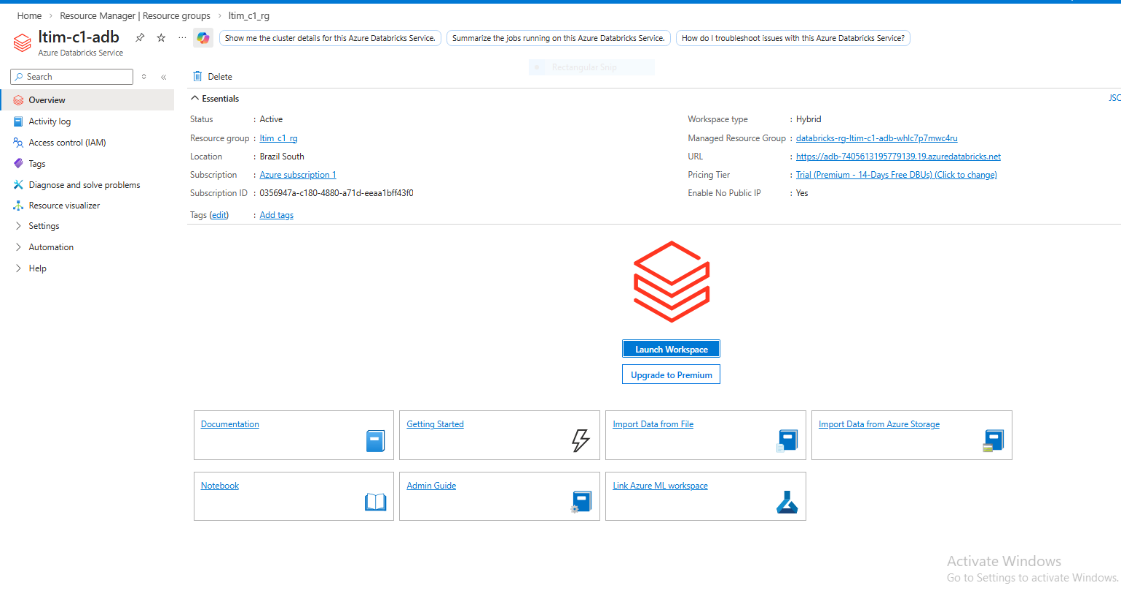
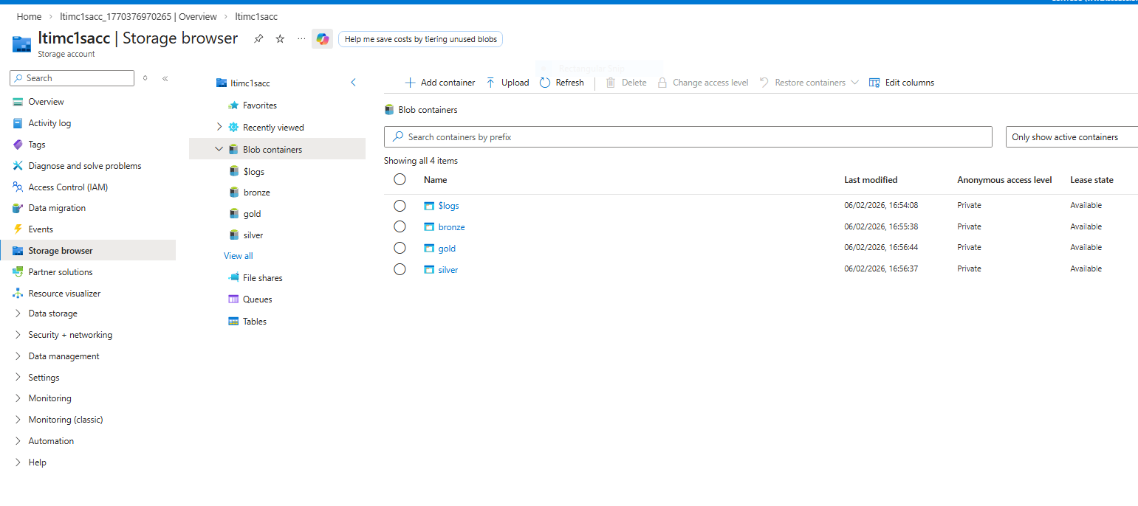
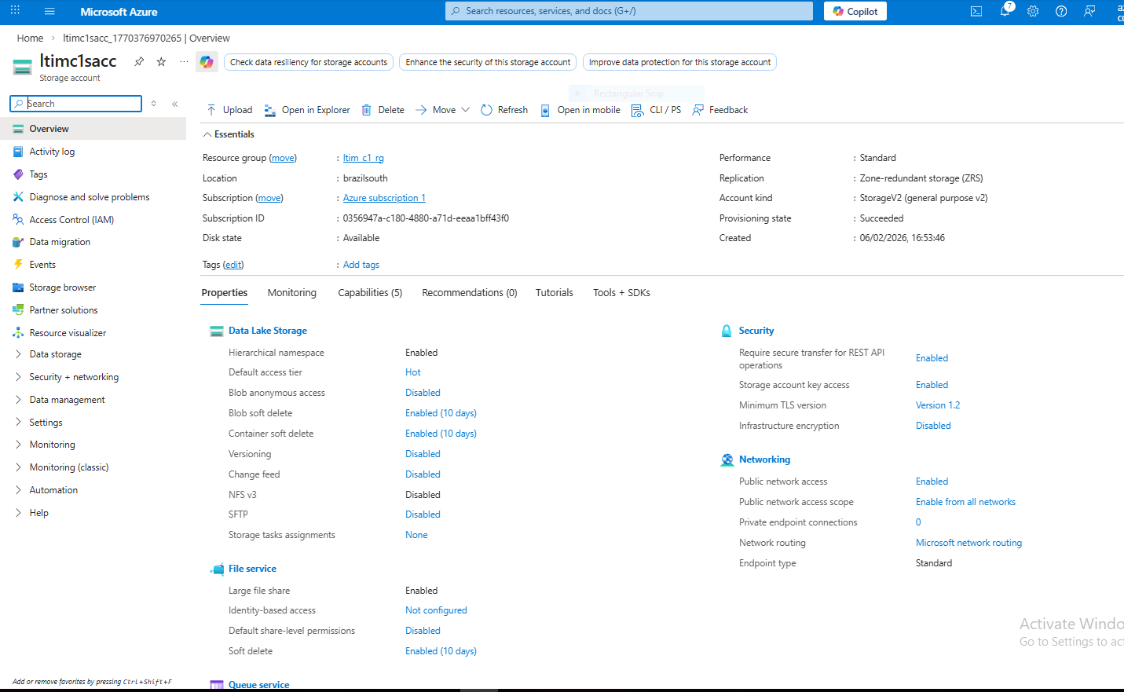
Cost outliers

Delivery date anomalies

The scaled dataset ensured that ingestion pipelines, transformation logic, and analytical queries could handle large volumes efficiently, closely mirroring production workloads.

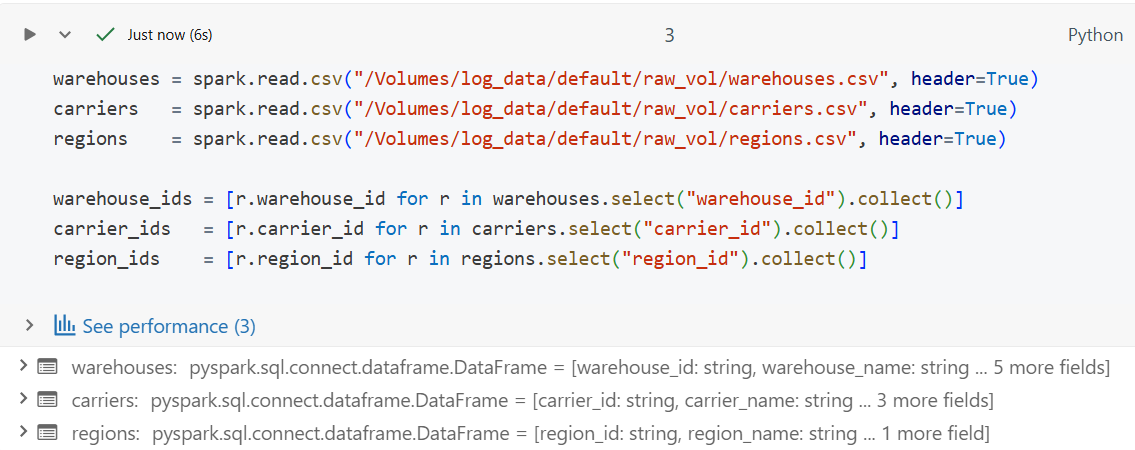
* **AZURE & DATABRICK SETUP**

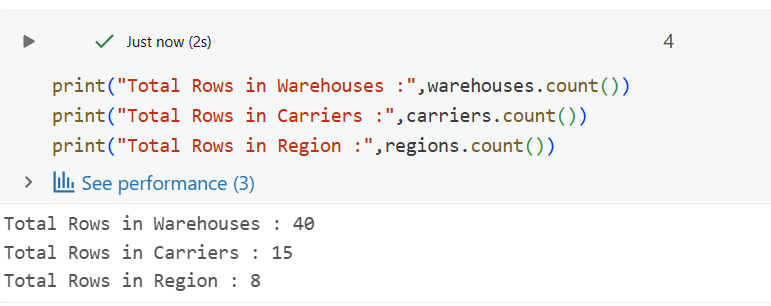
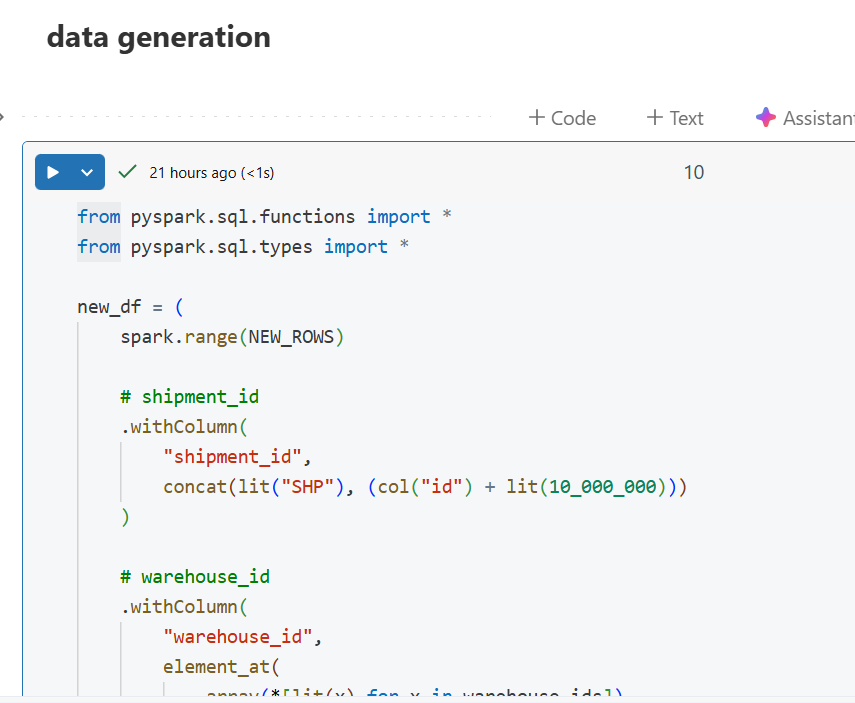




* **Source Systems – Daily Shipment & Reference Data**

-Validate the given Data and identify the Fact table and Dim table and identify the Joins (Foreign Keys) and understand the star Schema. Generate Extra Records on the Fact table and validate the tables again

**-** Loading the DIM table data as *CSV*

* + Total row Count of the Given data
* Entire code to Generate New Data into a Spark DF
  + 

from pyspark.sql.functions import \*

from pyspark.sql.types import \*

new\_df = (

    spark.range(NEW\_ROWS)

    # shipment\_id

    .withColumn(

        "shipment\_id",

        concat(lit("SHP"), (col("id") + lit(10\_000\_000)))

    )

  # warehouse\_id

    .withColumn(

        "warehouse\_id",

        element\_at(

            array(\*[lit(x) for x in warehouse\_ids]),

            ((col("id") % len(warehouse\_ids)) + 1).cast("int")

        )

    )

    # region\_id

    .withColumn(

        "region\_id",

        element\_at(

            array(\*[lit(x) for x in region\_ids]),

            ((col("id") % len(region\_ids)) + 1).cast("int")

        )

    )

    # carrier\_id (allow ~5% NULLs)

    .withColumn(

        "carrier\_id",

        when(col("id") % 20 == 0, lit(None))

        .otherwise(

            element\_at(

                array(\*[lit(x) for x in carrier\_ids]),

                ((col("id") % len(carrier\_ids)) + 1).cast("int")

            )

        )

    )

    # shipment\_cost

    .withColumn("shipment\_cost", round(rand() \* 5000 + 50, 2))

    # delivery\_status

    .withColumn("delivery\_status", lit("Delivered"))

    # fragile

    .withColumn("is\_fragile", when(rand() < 0.25, "Y").otherwise("N"))

    # shipment\_date

    .withColumn(

        "shipment\_date",

        date\_add(lit("2023-01-01"), ((col("id") % 365).cast("int")))

    )

    # delivery\_days (RANDOM 1–7)

    .withColumn(

        "delivery\_days",

        (floor(rand() \* 7) + 1).cast("int")

    )

    # delivery\_date (aligned with delivery\_days)

    .withColumn(

        "delivery\_date",

        date\_add(col("shipment\_date"), col("delivery\_days"))

    )

    # package weight

    .withColumn("package\_weight\_kg", round(rand() \* 50 + 0.5, 2))

    # declared value

    .withColumn("declared\_value\_inr", round(rand() \* 500000 + 500, 2))

    # payment type

    .withColumn("payment\_type", when(rand() < 0.65, "Prepaid").otherwise("COD"))

    # priority

    .withColumn(

        "priority\_level",

        when(rand() < 0.5, "Low")

        .when(rand() < 0.85, "Medium")

        .otherwise("High")

    )

    # created timestamp

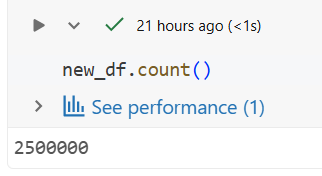
    .withColumn("created\_ts", current\_timestamp())

    # DROP Spark-generated ID

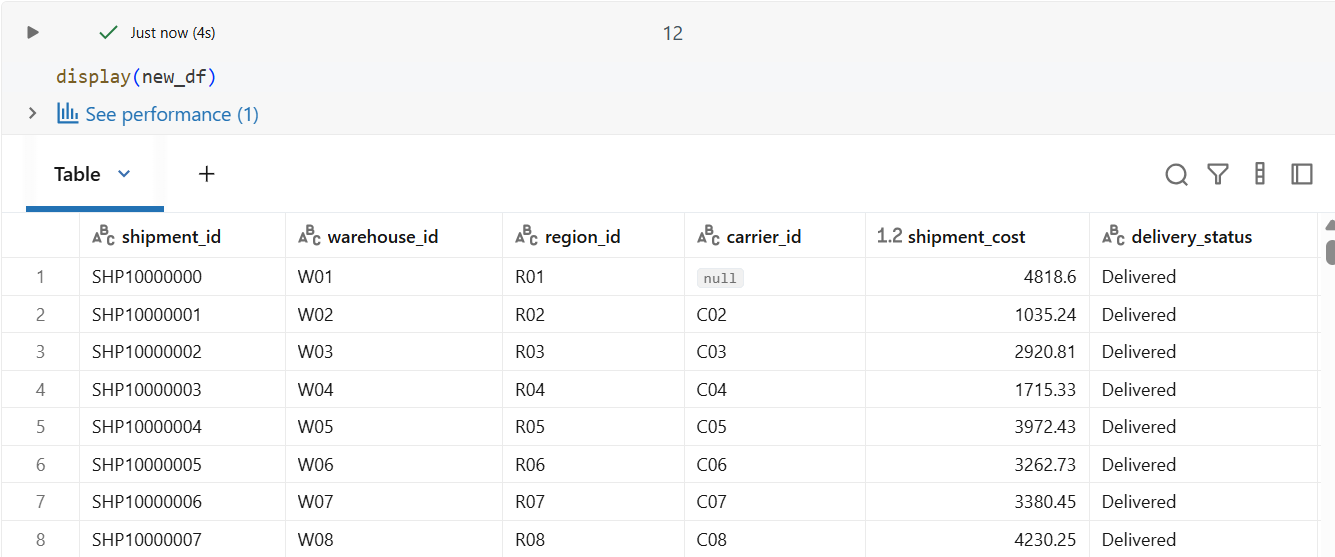
    .drop("id")

)

Count of New Data generated

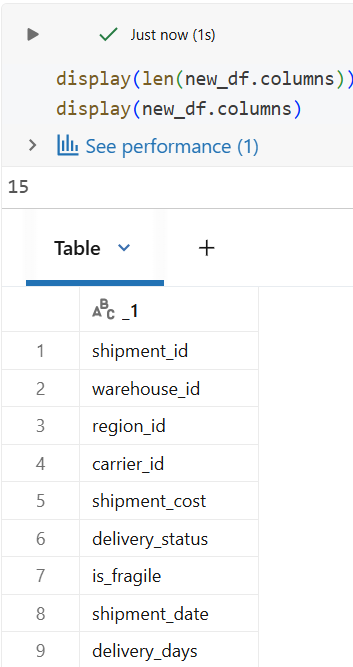


Sample Preview of New Data generated

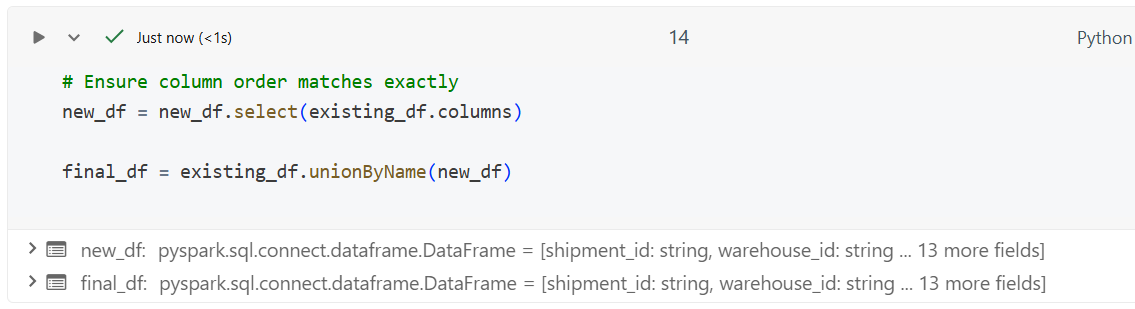


**Validating Data**

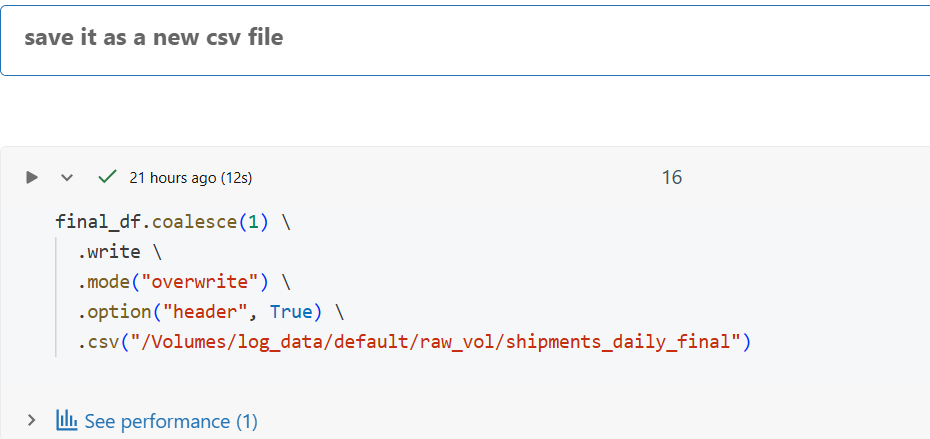
New Columns count is same as old Column

****

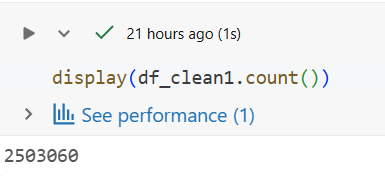
Ensuring Column Matches Exactly



Saving it as new *CSV* File



Final **row count** after adding more data



**Verifying the newly created data has same count of unique values in the foreign keys**

Taking count of Unique values in Foreign Keys



Verifying the Counts of the New Foreign Key Columns

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

The old data and the newly Generated data have the same number of Rows saying the new data can be joined without breaking the Schema.

**Sprint 1: Bronze ingestion**

**2. Azure Data Factory (ADF) Setup – Ingestion & Orchestration Layer-**

Azure Data Factory (ADF) serves as the central orchestration and ingestion engine for the platform. ADF pipelines are responsible for:

* Parameterized ingestion of daily shipment files
* Incremental loading using a watermark-based strategy
* Validation checks such as file existence and row count
* Pipeline execution logging and run metadata capture

ADF ensures reliable and repeatable data ingestion while supporting reruns and recovery from failures.

📸 **Figure 2: Azure Data Factory Ingestion Pipeline**  
Azure Data Factory Incremental Ingestion Pipeline

Introduction

The Project Implements an incremental batch ingestion pipeline using Azure Data Factory (ADF) to move shipment data from Azure Data Lake Storage (ADLS - Raw layer) to bronze layer, while maintaining a watermark-based control mechanism to ensure:

* No duplicate ingestion
* Controlled incremental loads
* Recoverability
* Auditability

Pipeline is Fully Automated, parameterises and monitored, with alerting enabled for successful runs.

1. Source Data Structure

*raw/*

*└── shipments/*

*└── shipment\_daily\_YYYY-MM-DD.csv  
  
Each batch represent data for specific ingestion date*

2. Watermark Design

**Purpose:**

The watermark tracks the last successfully processed date and operational metadata required for incremental ingestion

**Watermark File (JSON)**

Stored in ADLS

*control/shipments\_watermark.json*

**Watermark Structure**

{

"entity":"shipments",

"last\_successfull\_date":"2023-01-22",

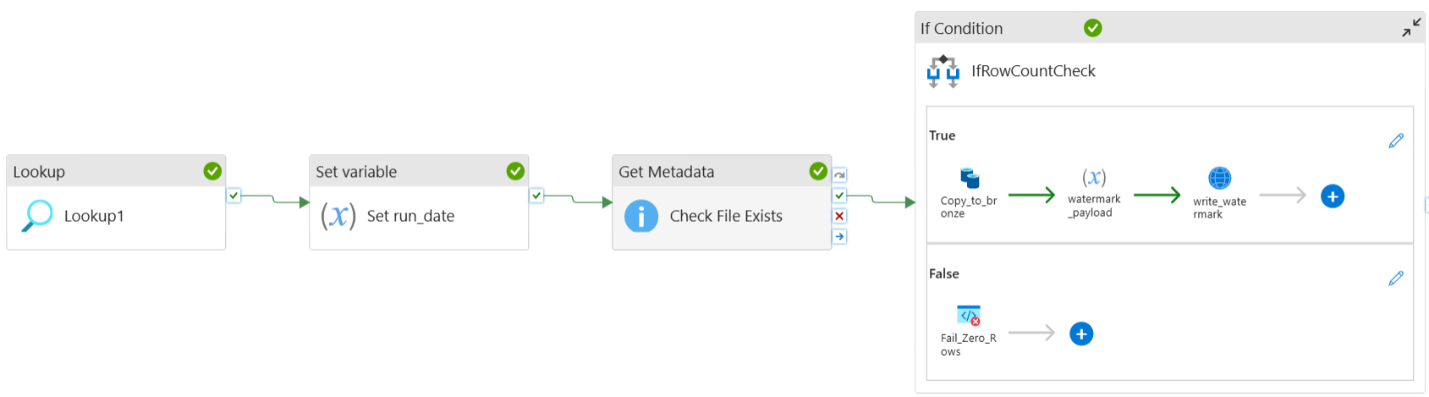
"last\_update\_ts":"2026-0210T06:57:42.2073290Z",

"rows\_added":821237,

"files\_added":1

}

3. Pipeline Architecture Overview

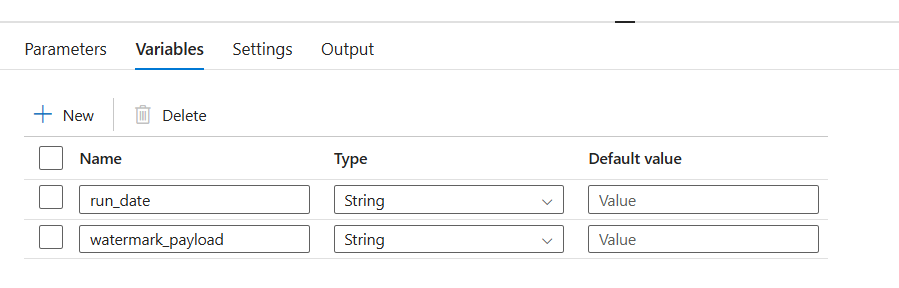


**Pipeline Flow**

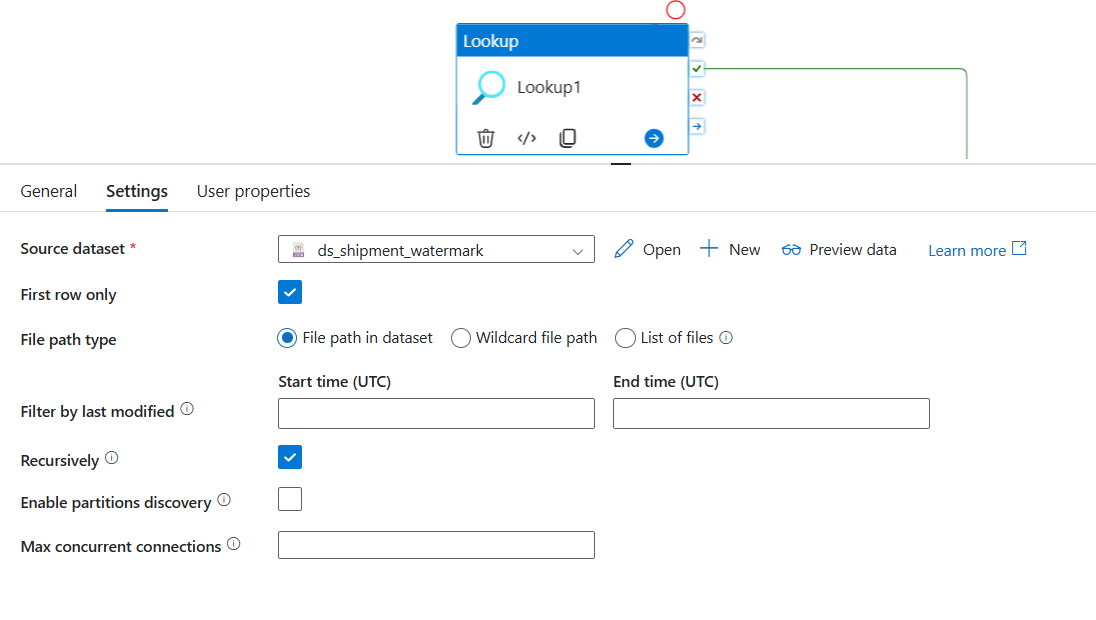
1. **Lookup Activity:** Read watermark using lookup activity
2. **Set variable (*run\_date*):** Calculate the next run Rate
3. **Get Metadata:** Check if the Expected source file exists
4. **Copy Data:** Copy data from Raw to Bronze layer
5. **If Condition:** Check the Row Counts condition
   1. **Truth:** If Row exists
      1. **Copy Data:** Data Movement from raw to bronze
      2. **Set Variable (*watermark\_payload*):** Update the watermark variable
      3. **Web Activity:** Update the watermark control file

4. Activity Level Explanation

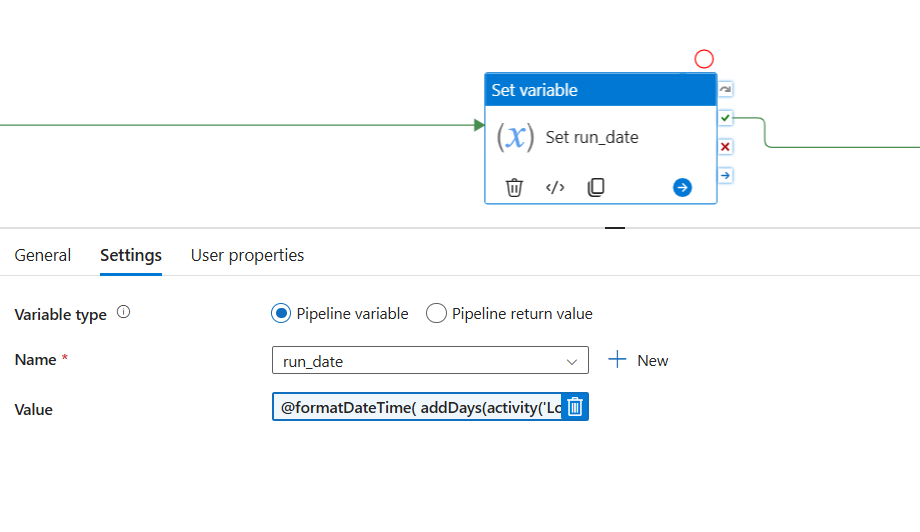
* **Pipeline Variable Initialisation:** Declaring variables in the pipeline canvas to access and modify the variables throughput the pipeline execution.



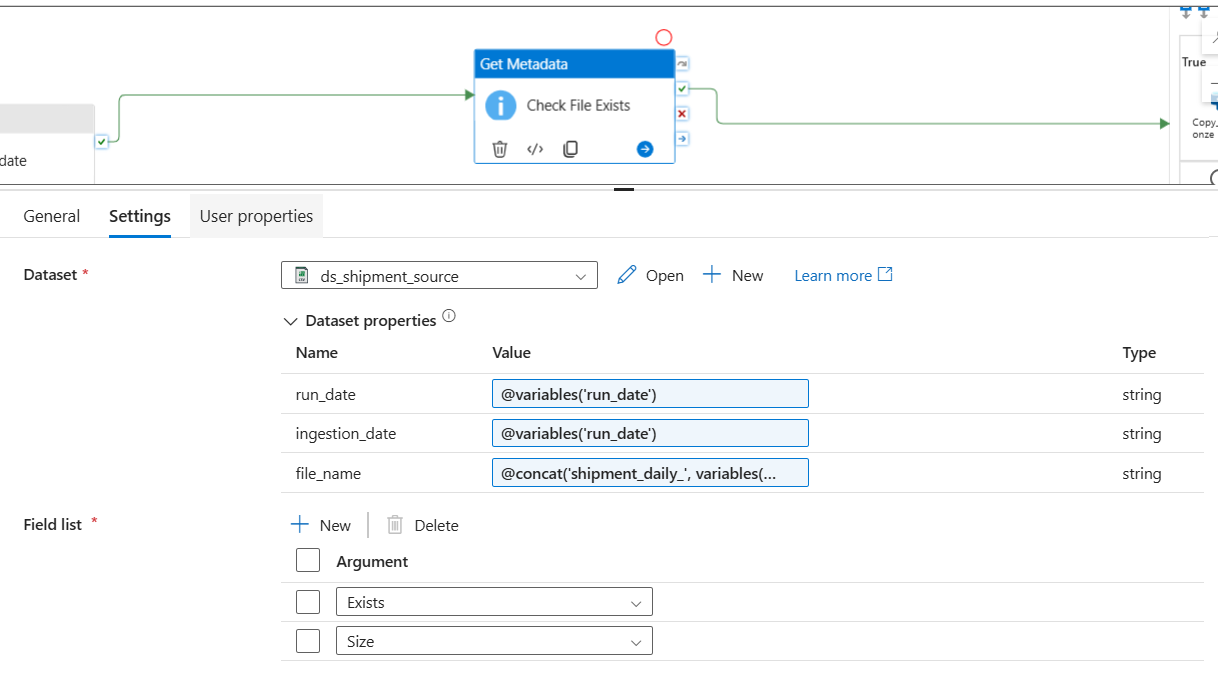
* **Lookup Activity:**  Reads the last successful ingestion date from the watermark file.



* **Set Variable (run\_date):** Computes the next ingestion date based on the watermark value.



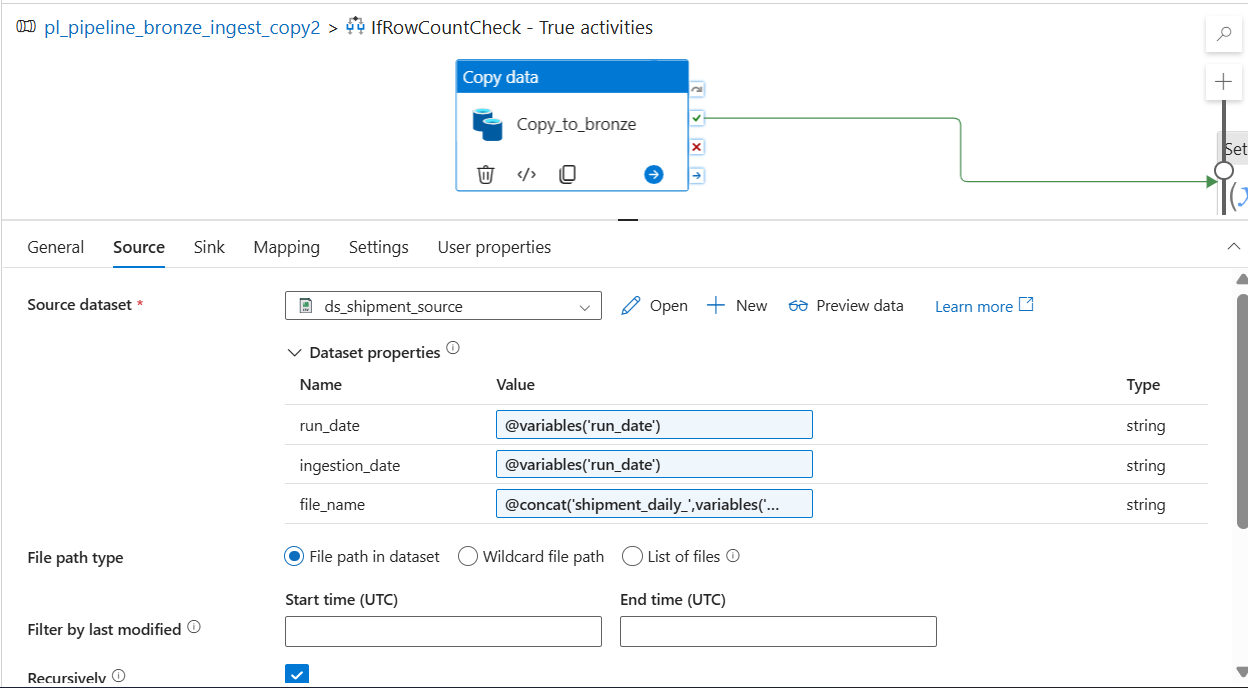
* **Get Metadata**: Validates whether the expected daily file exists before ingestion.



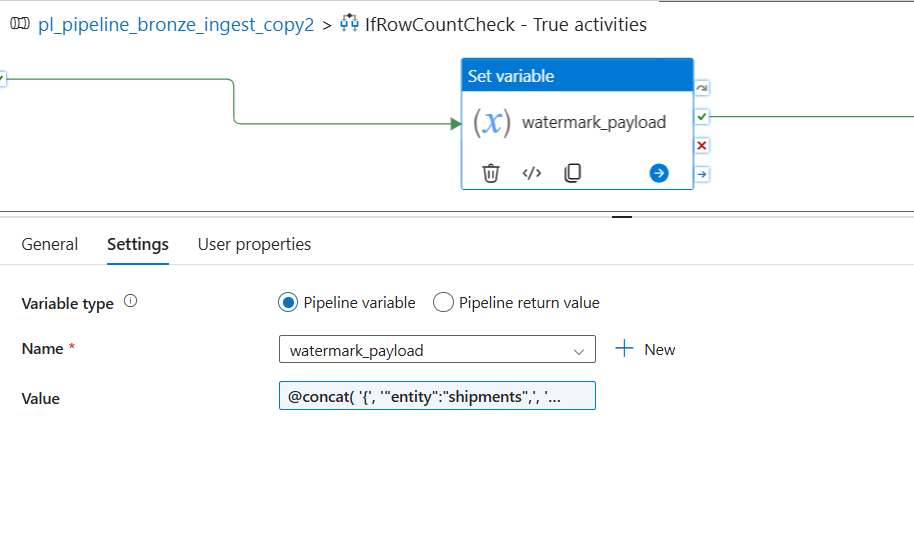
* **If Condition**: Ensures data is copied only when the source file exists.



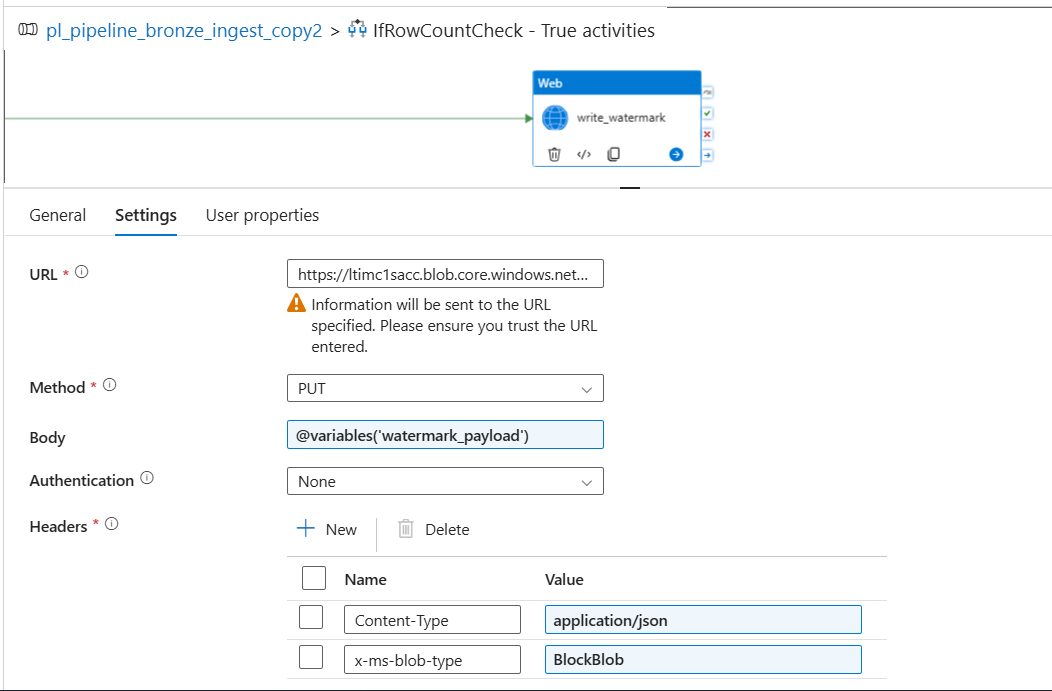
* **Copy Activity**: Moves shipment data from Raw storage to the Bronze layer.



* **Set Variable (watermark\_payload):** Constructs a JSON payload containing updated watermark values.

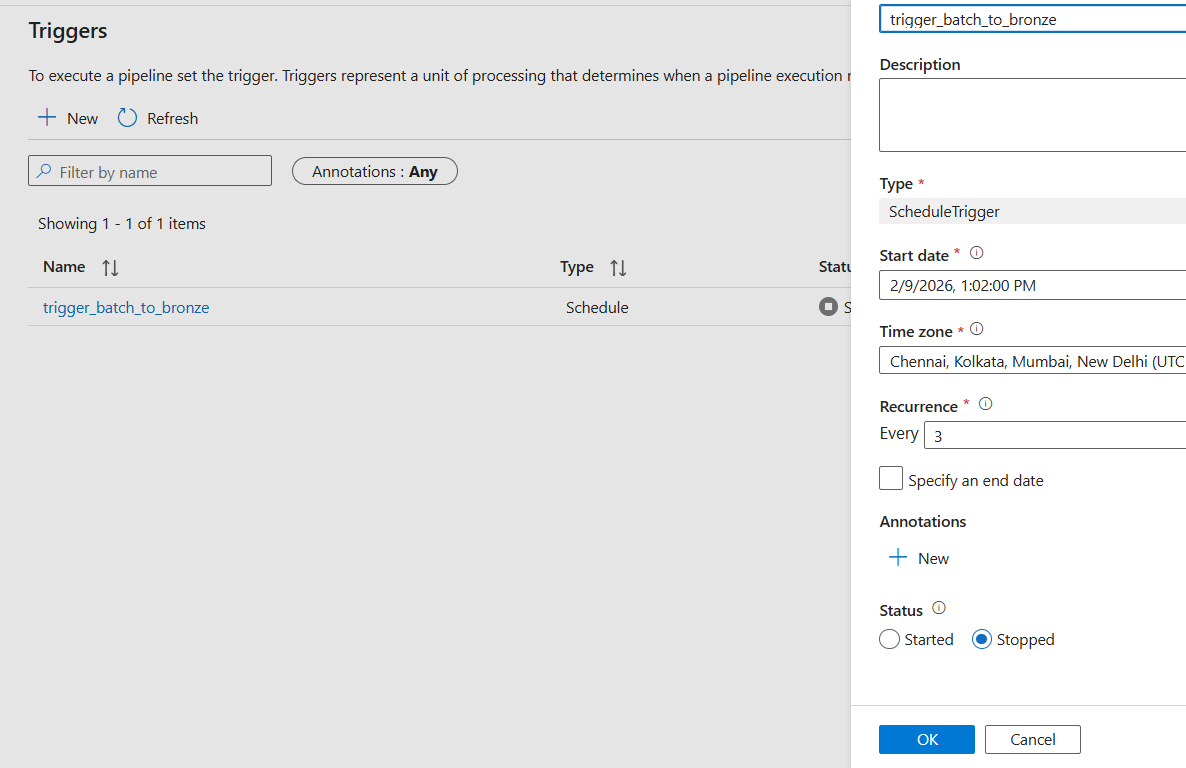


* **Web Activity**: Overwrites the watermark JSON file using an HTTP PUT operation.



5. Trigger Automation

The pipeline is automated using an Azure Data Factory Schedule Trigger configured to run once per 3 minutes (Ideally once per day). No pipeline parameters are required, as the watermark controls incremental execution.



Overwrites the watermark JSON file using an HTTP PUT operation.

**Sprint 2: Silver Transformations & Data Quality Architecture**

**Objective**

To transform raw Bronze data into clean, standardized, and trusted datasets by applying data quality rules, deduplication logic, and anomaly detection.

**Architecture Overview**

In Sprint 2, **Azure Databricks** acts as the primary processing engine, consuming raw CSV data from the **Bronze layer** stored in **Azure Data Lake Storage Gen2**. The processed outputs are written as **Delta tables** into the **Silver layer**, enabling ACID transactions, schema enforcement, and incremental updates.

**Key Architectural Components**

* **Source:** Bronze-layer raw CSV files partitioned by ingest date
* **Processing Engine:** Azure Databricks (PySpark)
* **Storage Format:** Delta Lake
* **Output Zones:** Silver and Quarantine zones in ADLS Gen2

**Data Flow**

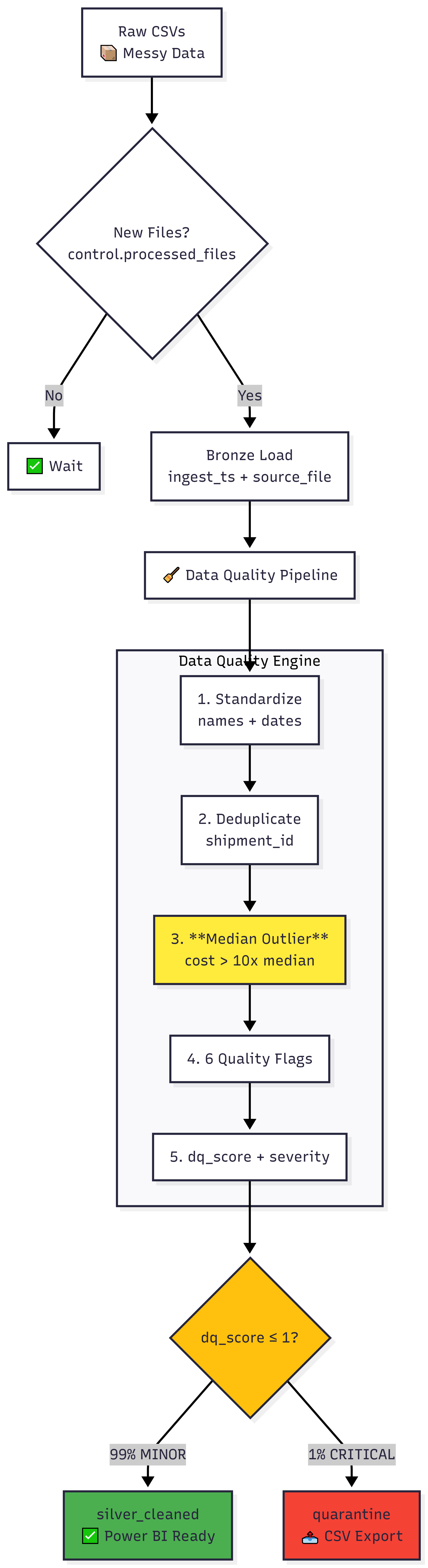
1. Databricks notebooks read raw shipment and reference data from the Bronze layer.
2. Data is type-casted and standardized (dates, numeric fields, flags).
3. Data quality rules are applied:
   * Null shipment cost detection
   * Duplicate shipment identification
   * Cost outlier detection
   * Delivery date anomaly checks
4. Invalid or non-conforming records are routed to a **Quarantine zone**.
5. Clean and validated records are written to **Silver Delta tables**.
6. A **Data Quality Summary table** is generated for monitoring and reporting.

**Architecture Benefits**

* Early detection and isolation of bad data
* Reusable and auditable data quality framework
* Delta Lake ensures reliability and performance for downstream processing

**Sprint 2 – Silver Layer Transformation & DQ Architecture**

1. **Data Flow & Medallion Architecture**

****

The pipeline follows a structured path to ensure data quality at every stage:

* **Bronze Layer (Ingestion):** Raw CSV files are ingested from the landing zone (/Volumes/logistics/bronze/shipments\_vol/).
* **Incremental Loading:** To ensure efficiency, we use a control table (‘processed\_files’) and a left anti join procedure to identify and process only new files.
* **Silver Layer (Cleansing):** Data is normalized, typed, and validated. Records are assigned DQ scores and routed to either "Cleaned" or "Quarantine" tables.
* **Gold Layer (Modeling):** Data is structured into a Star Schema optimized for Power BI performance.

**2. Detailed Data Cleaning & Transformations**

The Silver layer acts as the primary engine for data integrity. The following steps were performed in the Databricks notebook:

**A. Standardization**

* **Normalization:** All column names were converted to lowercase, and spaces were replaced with underscores for SQL compatibility.
* **Trimming & Casing:** Columns like carrier\_id and delivery\_status were trimmed of whitespace and forced to uppercase.
* **Type Casting:** Numeric fields like shipment\_cost were cast to double, and date strings were converted to date types.

**B. Deduplication Strategy**

* Duplicates with the same shipment\_id often occur due to late re-sends.
* **How:** We used a Spark Window function partitionBy("shipment\_id") and orderBy(F.col("ingest\_ts").desc()).
* **Result:** Only the most recent record (Row Number 1) is retained, ensuring the latest shipment status is reported.

**C. Median-Based Outlier Detection**

* **Why:** Standard deviation is easily skewed by extreme values. A median-based approach is more robust for detecting data entry errors.
* **How:** We calculated the median cost using approxQuantile("shipment\_cost", [0.5], 0.01).
* **Rule:** Any record where the cost is **greater than 10x the median** is flagged as dq\_cost\_outlier.

1. **Data Quality (DQ) & Quarantine**

We implemented a scoring system to govern data health. Every row is evaluated against specific business rules:

**DQ Rule Table**

| **Rule Name** | **Logic** |
| --- | --- |
| dq\_cost\_null | Checks if financial data is missing. |
| Dq\_date\_anomaly | Checks if shipment\_date is later than delivery\_date. |
| Dq\_invalid\_status | Validates that status is “DELIVERED”, “IN\_TRANSIT”, or “CANCELLED”. |
| Dq\_cost\_zero | Flags records with a cost of zero or less. |

**Routing & Efficiency**

* **DQ Score:** We sum these flags into a dq\_score.
* **Severity:** \* **MINOR (Score 0):** Clean data, routed to silver\_cleaned.
* **MAJOR/CRITICAL (Score ≥ 1):** Routed to **Quarantine**.
* **De-quarantining:** Quarantined data is exported to CSV for business correction, while the Gold layer remains 100% "clean" and trustworthy.

**Sprint 3: Gold Modeling Architecture**

**Objective**

To create a business-ready analytical data model using a star schema that supports high-performance reporting and incremental data updates.

**Architecture Overview**

Sprint 3 introduces the **Gold layer**, where curated Silver data is transformed into dimensional models. Azure Databricks continues to be used for transformations, while **Delta Lake MERGE operations** enable incremental loading of fact and dimension tables.

**Key Architectural Components**

* **Source:** Silver Delta tables
* **Processing Engine:** Azure Databricks
* **Storage Format:** Delta Lake
* **Target Schema:** Star Schema (Fact & Dimensions)
* **Optimization:** Partitioning and Z-Ordering

**Data Model Components**

* **Dimension Tables:** DimDate, DimWarehouse, DimRegion, DimCarrier
* **Fact Table:** FactShipments
* **Surrogate Keys:** Generated for all dimensions
* **Unknown Handling:** Default “UNKNOWN” dimension records for missing references

**Data Flow**

1. Silver Delta tables are read into Databricks.
2. Dimension tables are built and incrementally updated.
3. Surrogate keys are generated and managed.
4. FactShipments is created by joining facts with dimensions.
5. Incremental MERGE logic ensures only new or changed records are processed.
6. Gold tables are optimized for BI query performance.

**Architecture Benefits**

* BI-optimized star schema design
* Scalable incremental processing
* Improved query performance for analytics tools

**Sprint 3 – Gold Layer Star Schema Architecture**  
**Gold Layer: Star Schema Design**

The data model is designed as a **Star Schema** to ensure high performance in Power BI.

**Dimension Tables (Context)**

* **dim\_date:** Created from unique shipment dates to allow for Time Intelligence (YoY, MTD).
* **dim\_carrier / dim\_warehouse:** These use **Surrogate Keys (SK)** generated via row\_number().
* **Unknown Handling:** Missing carrier IDs are mapped to a -1 | UNKNOWN record to maintain referential integrity.

**Fact Table (Metrics)**

* **fact\_shipments:** Joins cleaned shipments with dimension keys.
* **Measures:** Includes shipment\_cost, delivery\_days, and handling flags.
* **Filtering:** shipment\_cost IS NULL records are explicitly excluded to protect financial KPIs.

**Sprint 4: Power BI & Validation Architecture**

**Objective**

To expose curated Gold data to business users through Power BI dashboards while validating data accuracy, freshness, and quality across all layers.

**Architecture Overview**

In Sprint 4, **Power BI** connects directly to the **Gold layer** using Databricks SQL endpoints or direct connectivity. This sprint focuses on semantic modeling, KPI creation, validation, and reconciliation to ensure business trust in the analytics.

**Key Architectural Components**

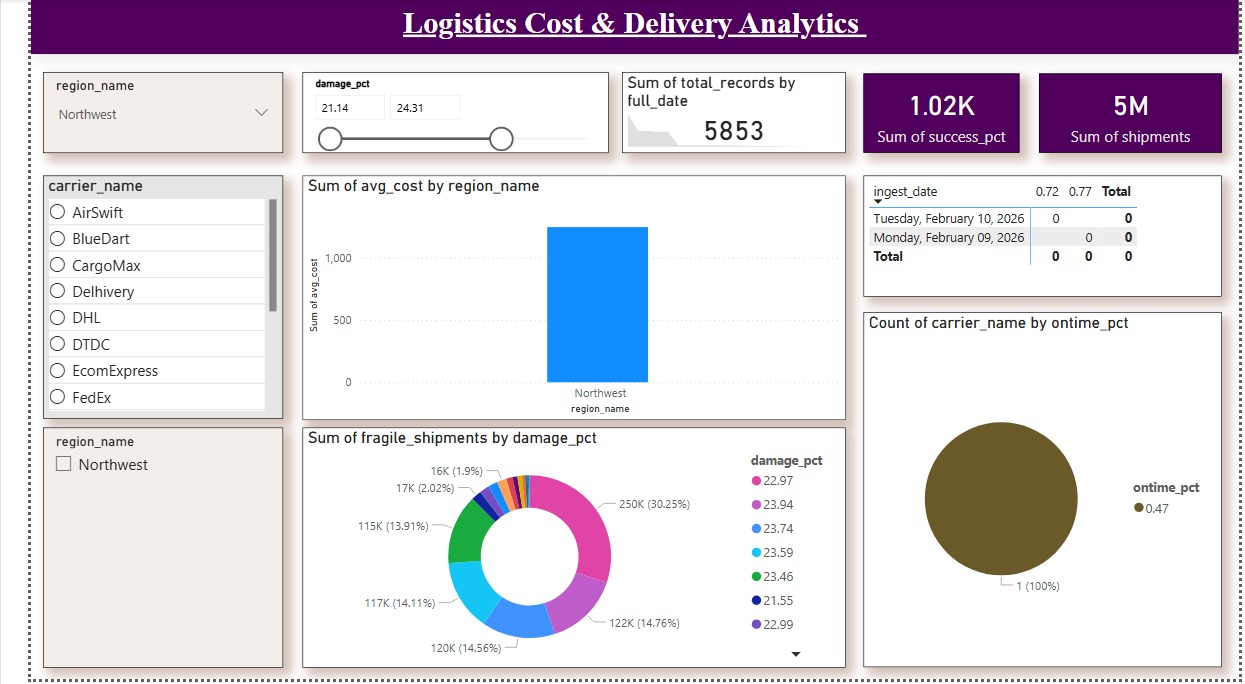
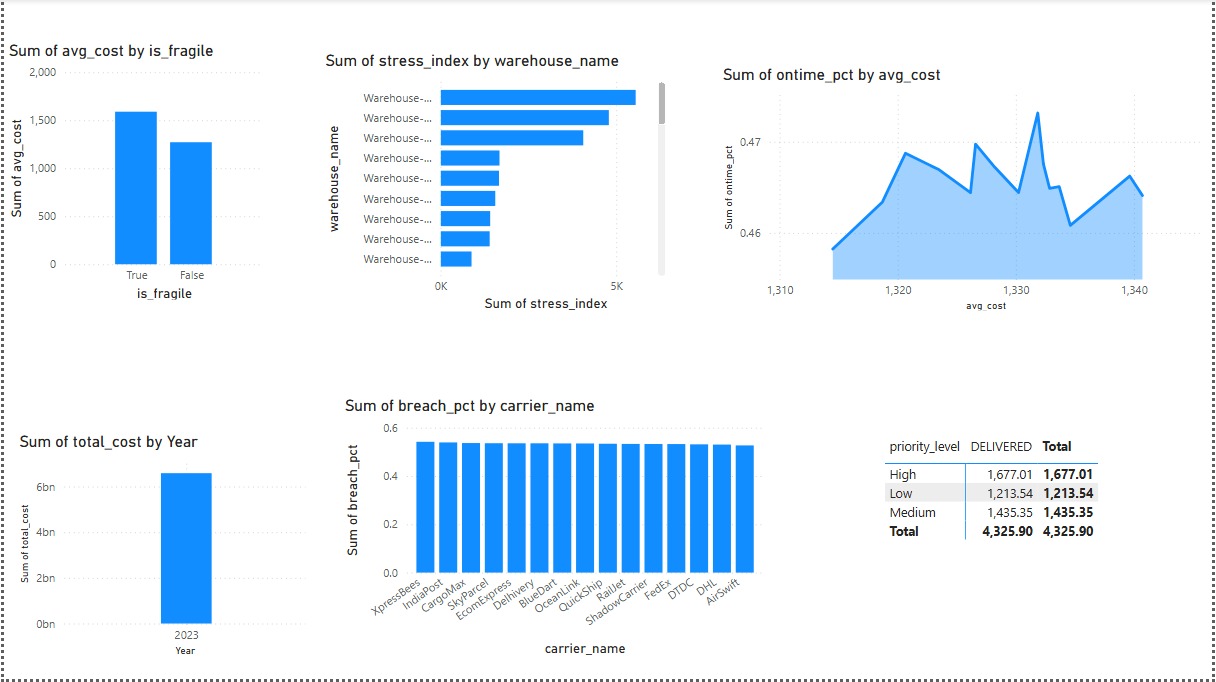
* **Source:** Gold Delta tables
* **Query Layer:** Databricks SQL / Direct Lake (as applicable)
* **Visualization Tool:** Power BI
* **Validation Layer:** Databricks notebooks and SQL checks

**Data Flow**

1. Power BI establishes a connection to Gold dimension and fact tables.
2. Relationships are defined using the star schema model.
3. Business measures and KPIs are created in the semantic layer.
4. Dashboards are built for executive, operational, and data quality views.
5. Validation checks reconcile record counts across Bronze, Silver, and Gold.
6. Data freshness and pipeline success metrics are verified.

**Architecture Benefits**

* High-performance analytics using curated Gold data
* Clear separation between data engineering and reporting layers
* End-to-end validation ensures data accuracy and trust

📸 **Figure: Sprint 4 – Power BI Analytics & Validation Architecture**  


Each sprint concluded with validated deliverables and acceptance checks.

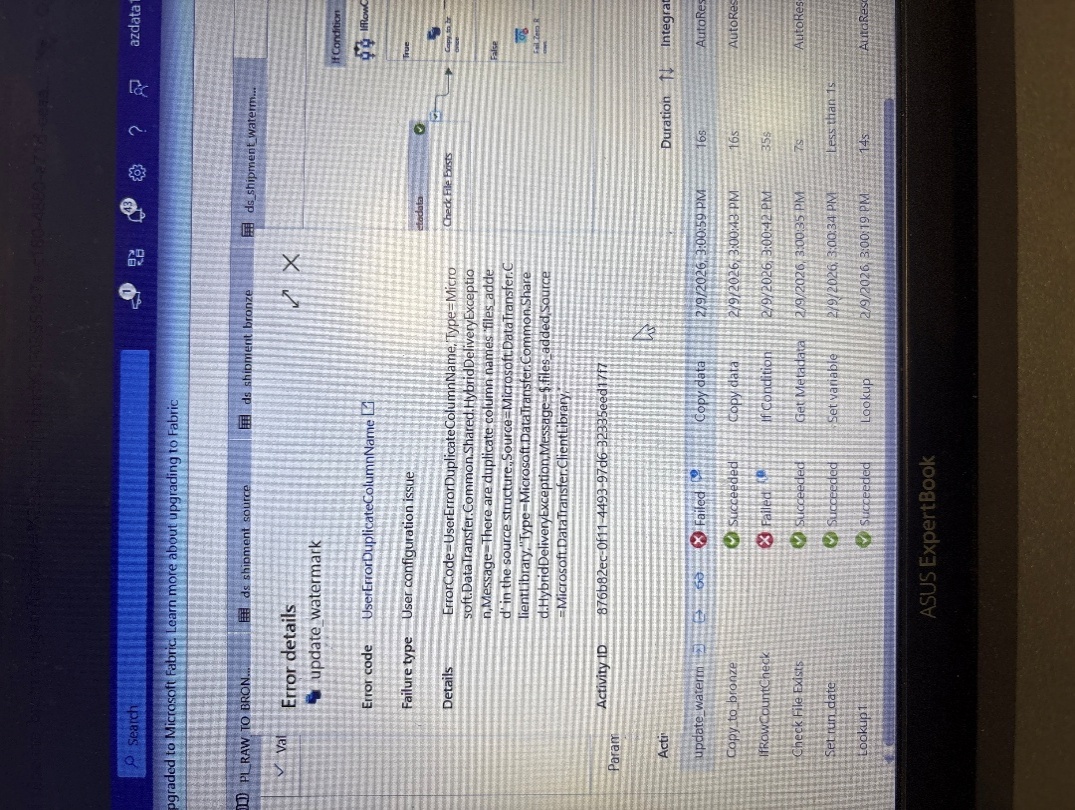
11. Challenges Faced During Implementation

11.1 Data Ingestion Challenges

* Schema drift in incoming CSV files
* Handling late-arriving and duplicate data
* Watermark management for reruns
* ADF pipeline Creating Issues when updating from same watermark file is access to update
* Error Faced and Resolutions

**Duplicate Column Errors**

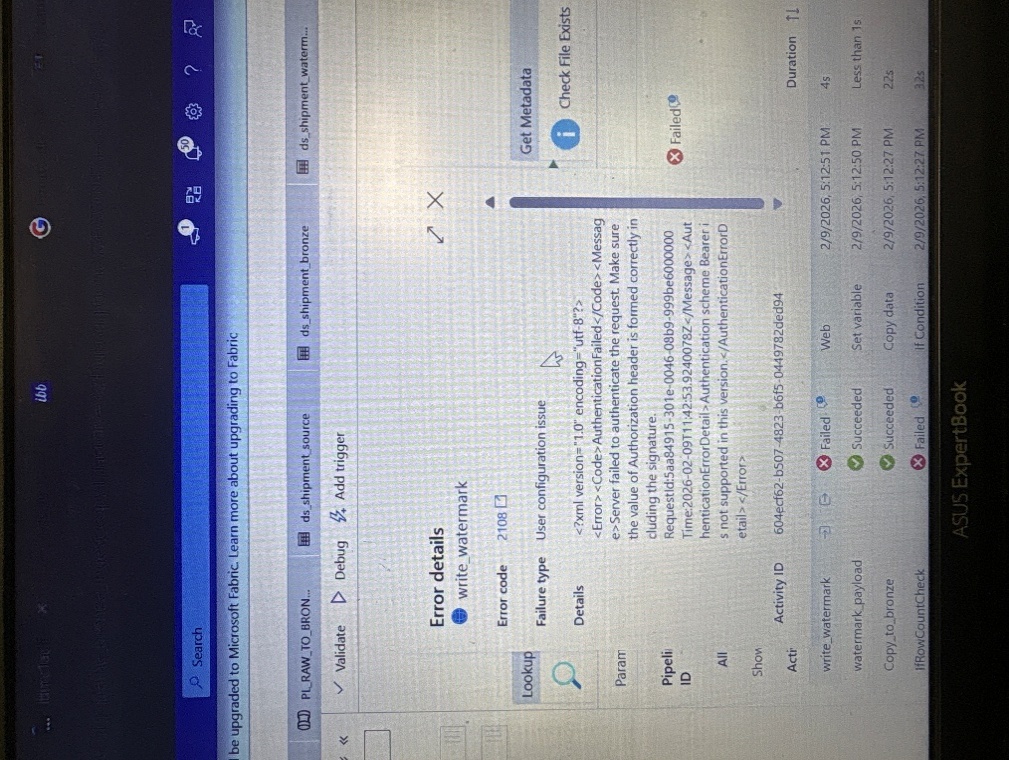
Caused by using Copy Activity to update JSON.



Resolved by switching to Web Activity with full overwrite.

**Authentication Failures**

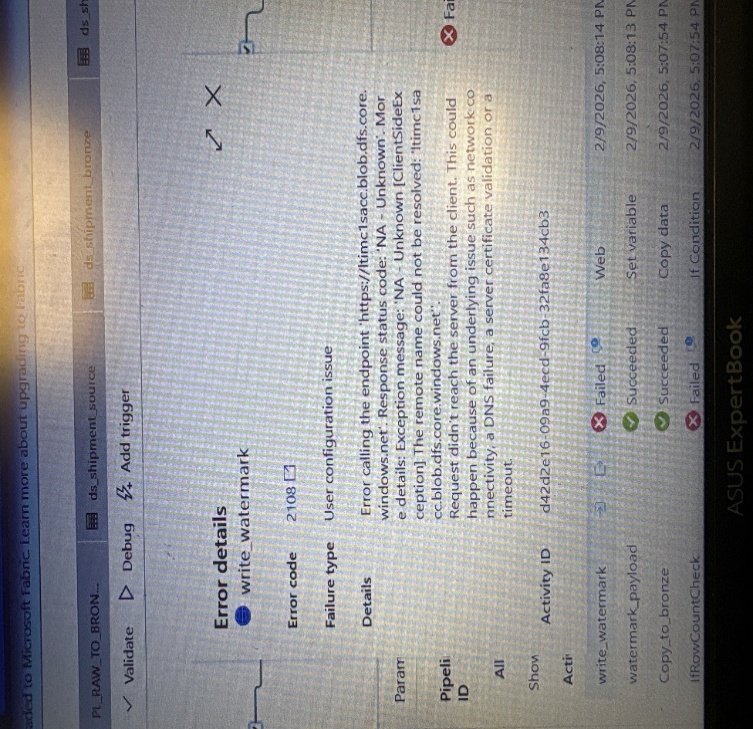
Managed Identity is not supported for Blob PUT operations.



Resolved by using SAS token authentication.

**DFS Endpoint Errors**

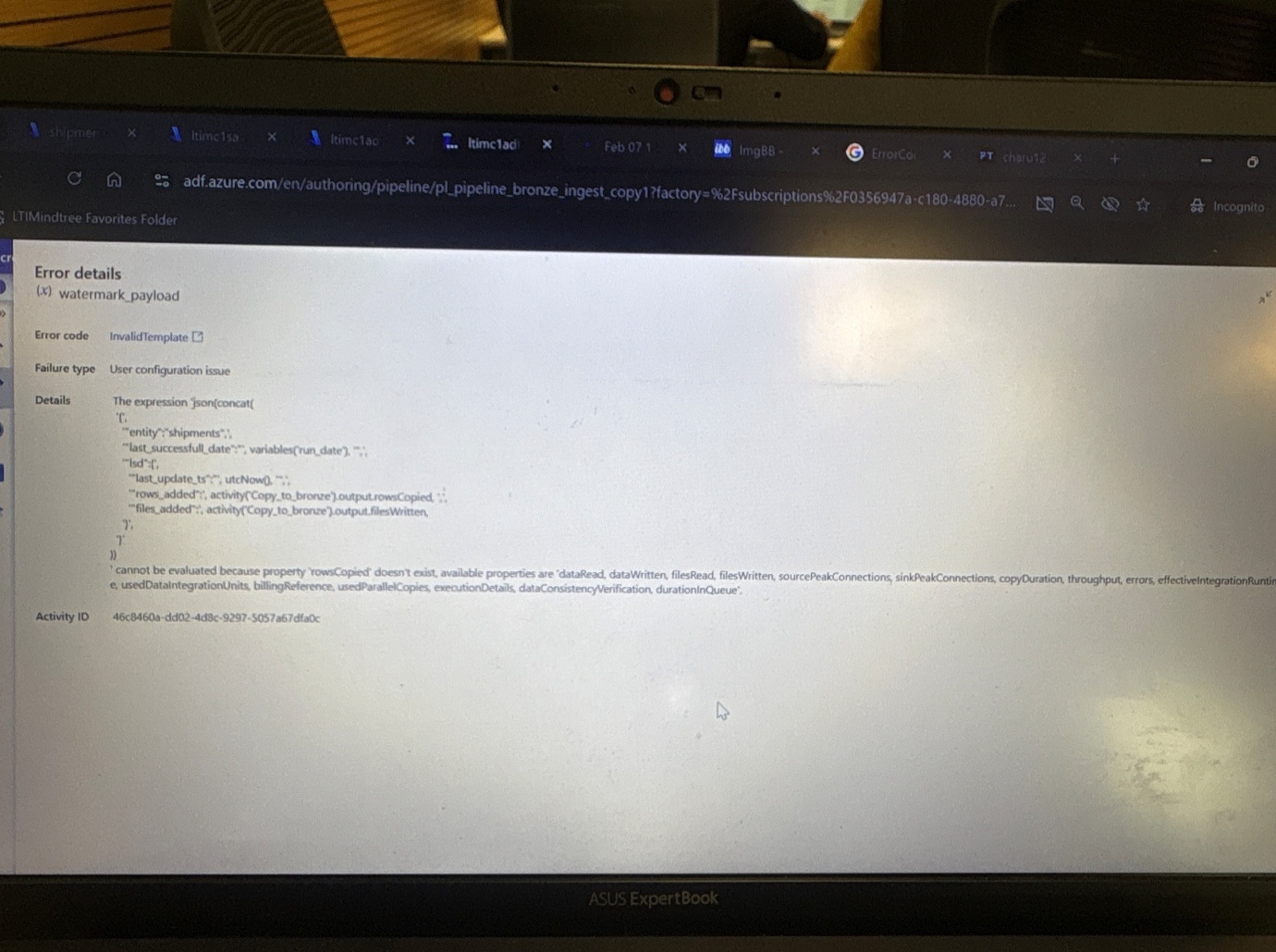
Web Activity does not support DFS endpoints.



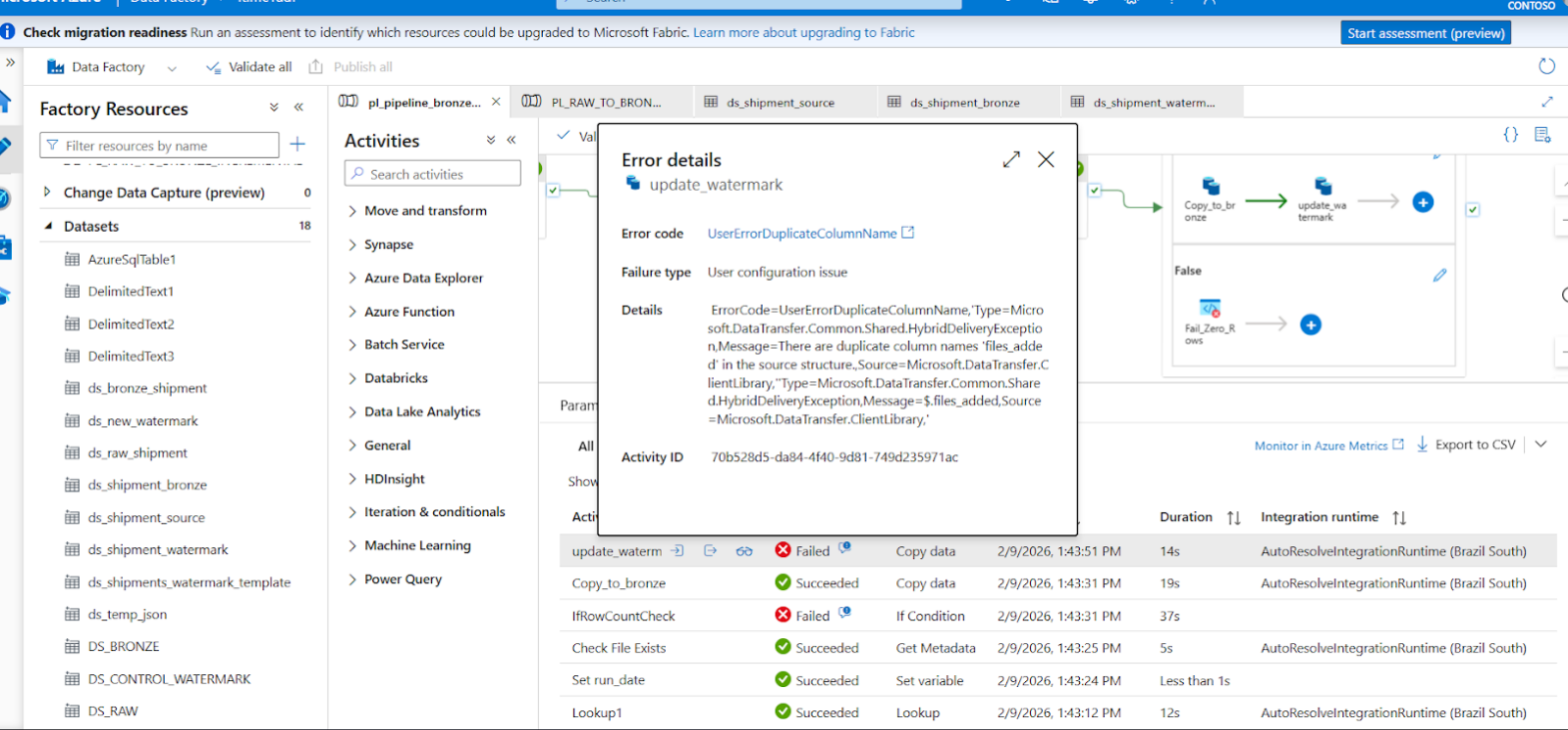
Resolved by using Blob endpoint instead.

**Invalid Copy Output References**

Incorrect usage of unsupported properties.

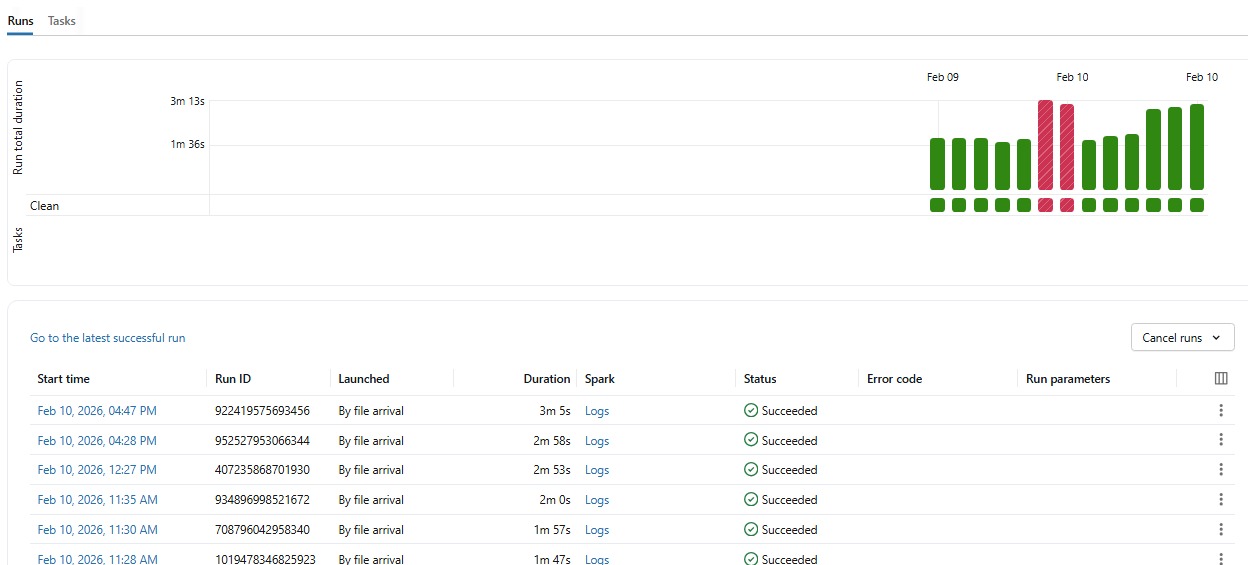


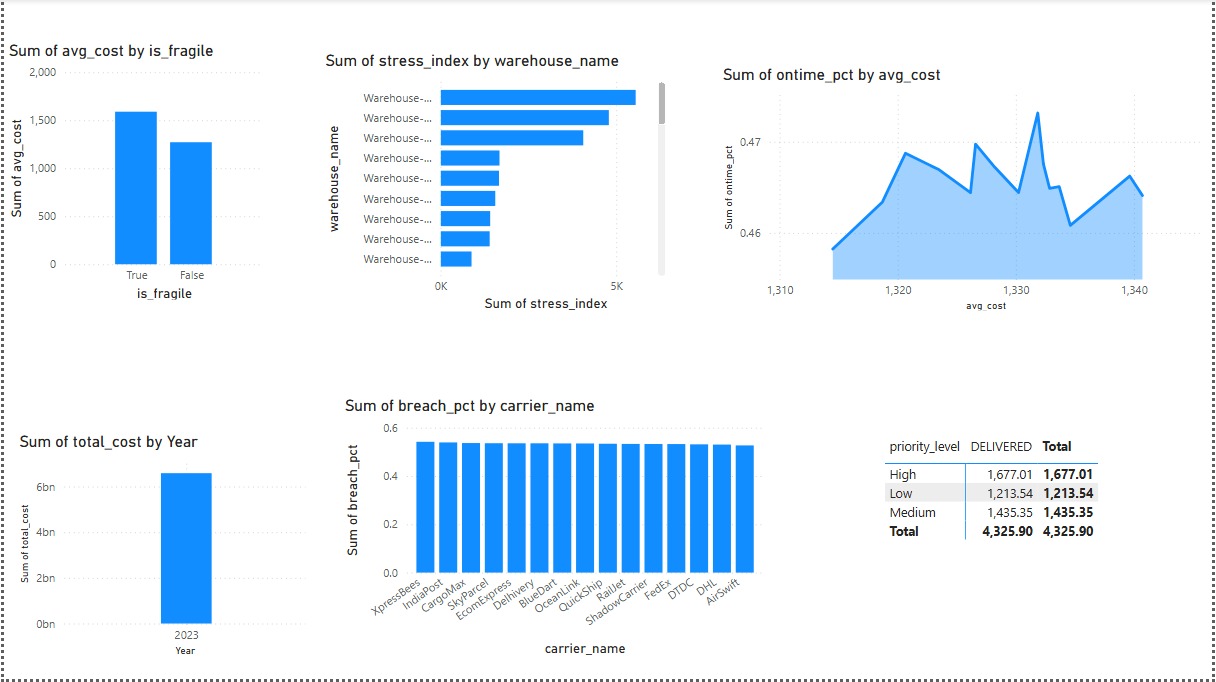
Resolved by using valid outputs such as *rowsRead* and *filesWritten*.



12. Success Highlights

* Automated daily ingestion with zero manual intervention
* Clean, reconciled data across Bronze, Silver, and Gold
* High-performing Power BI dashboards with accurate KPIs

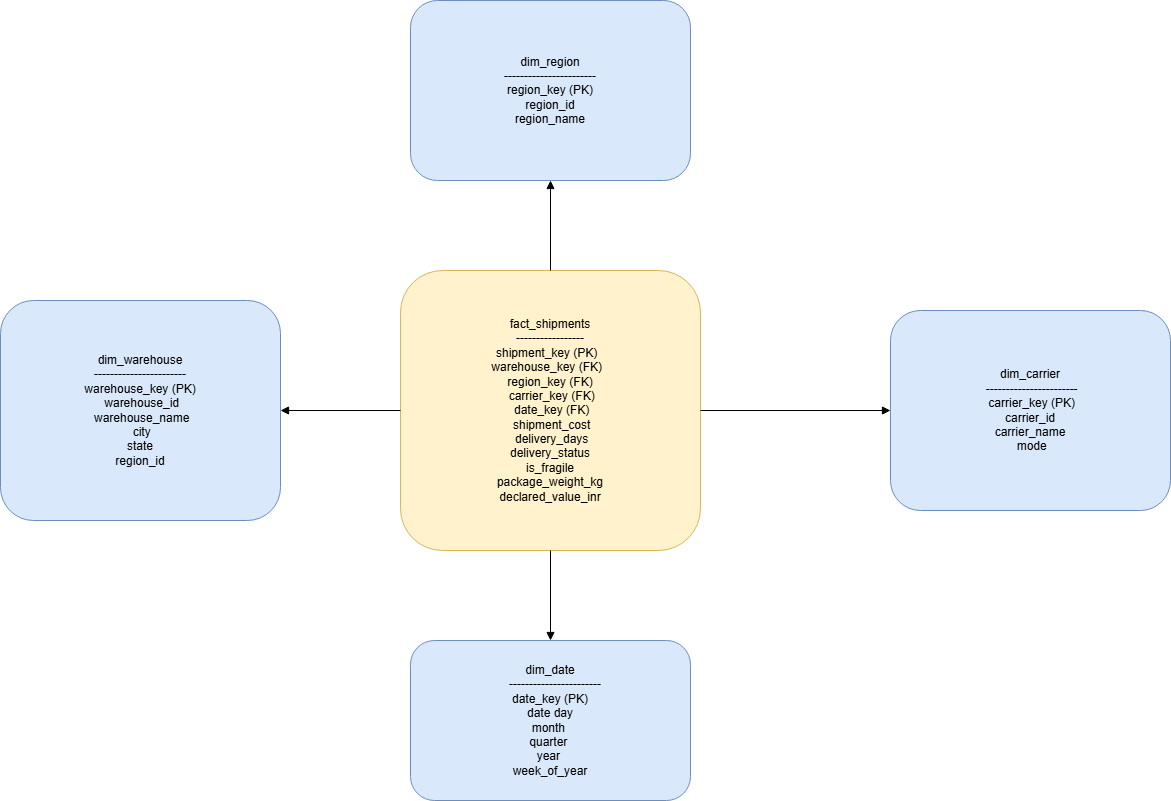
📸 Screenshot Placeholder – Successful ADF Run

📸 Screenshot Placeholder – Power BI Dashboard Overview

# 13. Flowcharts & Diagrams

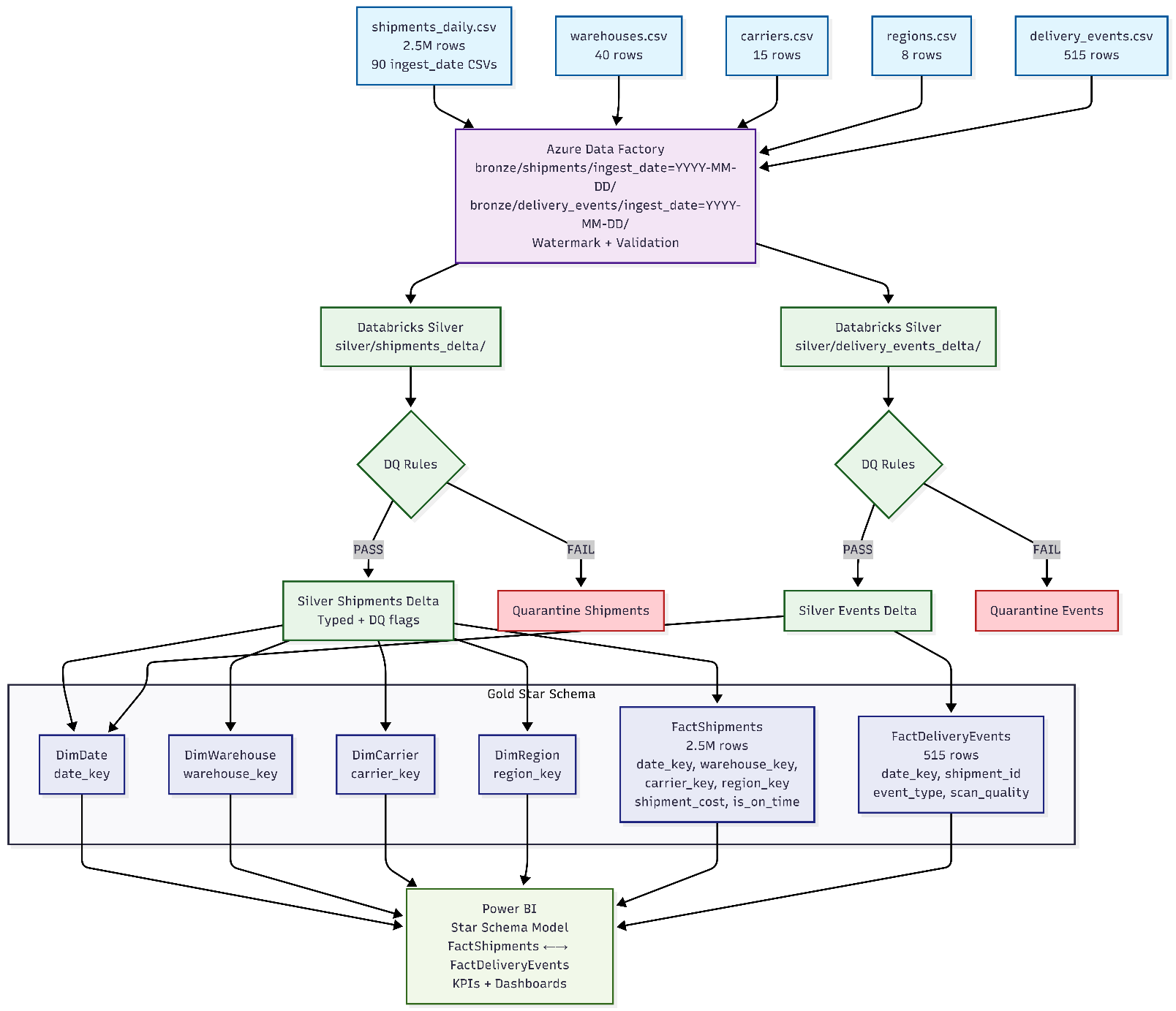
13.1 End-to-End Data Flow

📊 Diagram Title: End-to-End Logistics Data Pipeline Flow

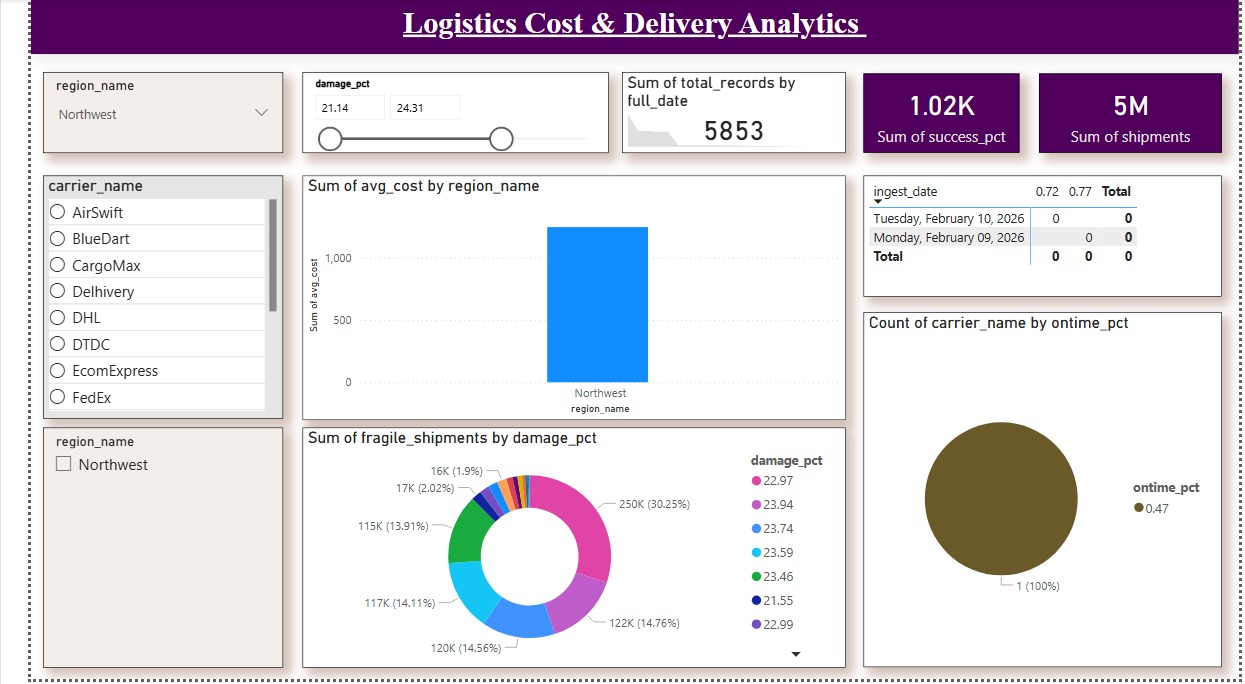


13.2 Medallion Architecture Flow

📊 Diagram Title: Bronze → Silver → Gold Transformation Flow



13.3 Power BI Semantic Model

📊 Diagram Title: Star Schema – Power BI Semantic Model

15. Conclusion & Future Enhancements

This capstone project demonstrates a production-ready data engineering solution using Azure-native services. The platform ensures data reliability, scalability, and analytical performance while delivering meaningful business insights through Power BI.

The architecture and implementation align with industry best practices and are suitable for real-world enterprise logistics analytics.

* Future Enhancements
* Real-time streaming using Event Hub
* Advanced anomaly detection using ML
* CI/CD automation for Databricks and ADF
* Row-level security in Power BI