Retail Data Analytics Project

#### 

#### 

**By : Sarthak Jain**

**Date : 15-10-25**

**Satbeer Singh Bhatia Prasad Adsul**

**Global Delivery Manager Project Owner**

**Table of Contents**

| **Section** |
| --- |
| **1. Introduction** |
| 1.1 Problem Statement |
| 1.2 Objectives of the Project |
| 1.3 Scope of the Project |
| **2. Project Overview** |
| 2.1 Project Aim |
| 2.2 Key Technologies Us |
| 2.2.1 Apache Spark (PySpark) |
| 2.2.2 Apache Airflow |
| 2.2.3 Docker |
| 2.2.4 MySQL |
| 2.3 Configurations Guide |
| **3. System Model** |
| 3.1 Architecture |
| 3.2 Database Design |
| **4. Methodology** |
| 4.1 Medallion Architecture |
| 4.2 Star Schema Design |
| 4.3 Version Control |
| 4.4 Testing and Bug Fixing |
| **5. Implementation** |
| 5.1 Stage 1:Ingestion |
| 5.2 Stage 2: Transformation |

| **6. Results and Discussion** |
| --- |
| 6.1 Airflow Orchestration |
| 6.2 Business Insights |
| **7. Challenges Faced** |
| **8. Conclusion and Future Work** |
| 8.1 Conclusion |
| 8.2 Future Scope |

1. **Introduction**

### Problem Statement

Raw retail transactional data is often large, inconsistent, and highly inconsistent schemas, containing critical quality issues such as duplicates, missing values, and inconsistent formatting. Utilizing this data directly leads to inaccurate business reports and poor operational decisions.

**Project Focus:** The Retail Data Analytics Pipeline project is designed to process, clean, transform, and analyze large volumes of retail transaction data. This pipeline follows a structured Bronze–Silver–Gold architecture, ensuring data reliability, scalability, and usability for analytics and auditing.

### Objectives of the Project

To overcome the challenges in manual expense tracking, primary objective of this project is :

* Ingest raw retail data from sources (like : CSV file).
* Perform data cleaning and transformation (PySpark).
* Modular code structure.
* Create a dimensional model (star schema) with fact and dimension tables for analytics.
* Enable data quality checks, logging, and orchestration using Apache Airflow.
* Scalable deployment (Docker)
  1. **Scope of the project**

This project demonstrates the design and implementation of a Retail Data Analysis Pipeline using Modern data engineering best practices. We can run the pipelines locally in our machine and use Apache Airflow as well using DAGs.



## Project Overview

### Background and Motivation

This project implements a scalable ETL (Extract, Transform, Load) pipeline for retail data analysis, using Apache Airflow for orchestration, PySpark for processing, and MySQL for data storage. The pipeline ingests raw CSV files into a "bronze" layer (raw, validated data), transforms them into a "silver" layer (cleaned, enriched data), and builds a data warehouse with dimension and fact tables for analytics.

### Project Aim

The central aim is to deploy a production-grade, data pipeline that can process incoming transactional data incrementally and provide a clean, centralized Gold Layer (MySQL database) for business intelligence tools.

**2.3 Key Technologies Used**

To achieve the project objectives, the following key technologies were adopted for development:

### Apache Spark (PySpark) - 3.5.6



Spark is the powerful engine we use for handling and processing large amounts of data quickly. We write all our transformation and cleaning logic using PySpark, which is Spark's easy-to-use Python interface. It allows us to manage complex tasks like data imputation and modeling by distributing the workload across multiple systems.



### Apache Airflow - 2.11.0

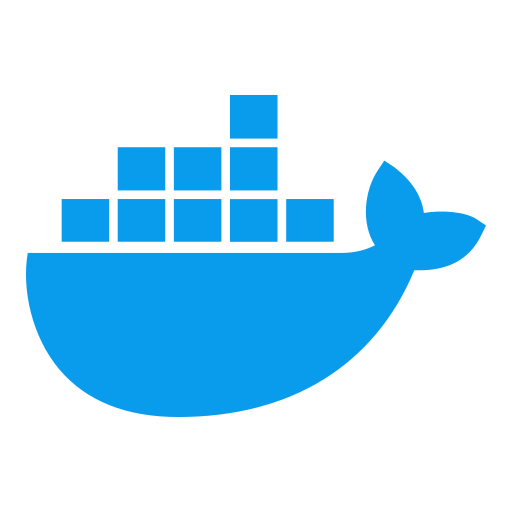
Airflow acts as the central control for the entire pipeline, managing the sequence of all data jobs. It ensures that data cleaning only starts after the raw data has been fully ingested, providing notifications viaaa email and automated retries if any job fails.

**

### Docker - 4.44.3



Docker ensures that our entire complex environment, including the Spark engine and the Airflow scheduler, runs consistently on any machine. It packages the software and its dependencies into isolated units called containers, eliminating the common problem of "it is working on my machine." This makes our project portable and easy for any developer to deploy.



### MySQL - 8.0



MySQL serves as our final Gold Layer, which is the structured data warehouse optimized for fast reporting. It holds the clean, modeled Star Schema tables and is the source of truth for the Power BI dashboard. We rely on its quick analytical queries.



### 2.4 Configuration Guide

The pipeline runs on a Dockerized Airflow/Spark environment connected to a local MySQL instance.

**Step 1: .env File Configuration**

The .env file stores sensitive/environment-specific variables. Create it in my project root and add required keys (e.g., from config.py).

**Key Variables:**

JDBC\_URL=jdbc:mysql://host.docker.internal:3307/(database name) (MySQL connection, use host.docker.internal for Docker).

JDBC\_USER= {user accordingly}

JDBC\_PASS={password accordingly)

BRONZE\_TABLE= bronze\_table\_name

SILVER\_TABLE=silver\_table\_name

DQ\_DIR=/opt/airflow/logs/dq\_report (Airflow log path).

QUARANTINE\_DIR=/opt/airflow/logs/quarantine. (For quarantine data)

JAR\_PATH=/opt/airflow/jars/mysql-connector-j-9.4.0.jar (Spark JDBC driver).

SMTP\_EMAIL=(email accordingly) (For Airflow alerts).

SMTP\_PASS=(pass accordingly) (Gmail app password).

**Step 2: Docker Configuration**

docker-compose.yml and Dockerfile handle containerization.

Dockerfile: Builds a custom Airflow image with PySpark and MySQL JAR.

docker-compose.yml: Defines services (Airflow, MySQL, Redis, Postgres).

**Setup Steps:**

Build: docker-compose –build -d

Run: docker-compose up -d

Stop: docker-compose down

Delete image: docker-compose down -v

## System Model

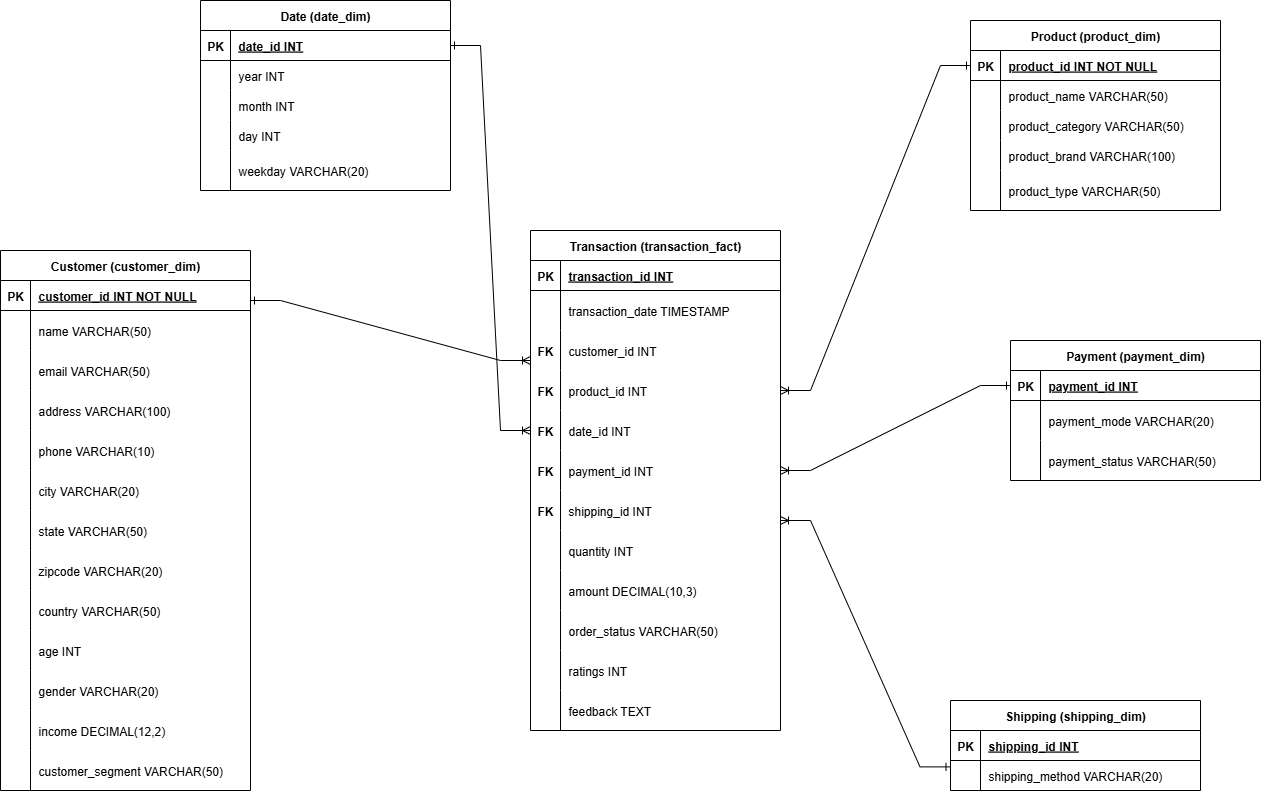
### Architecture

The software architecture follows the Medallion Architecture, separating data integrity and quality concerns into distinct stages managed by an Airflow pipeline.

* Bronze Layer (Ingestion): Raw, storage of source data.
* Silver Layer (Transformation): Cleaned, filtered, and transformed data, ready for modeling.
* Gold Layer (Reporting): The final, query-optimized Star Schema (MySQL).



### Database Design



The Gold Layer is designed as a Star Schema to ensure high performance for analytical queries.

| **Table** | **Purpose** |
| --- | --- |
| silver\_retail | Stores metrics and measures related to each transaction of the customer. |
| silver\_customer | Contains all descriptive attributes about each unique customer. |
| silver\_product | Contains attributes about the products. |
| silver\_shipping | Contains information of shipping. |
| silver\_payment | Contains all the payment details of the customer. |
| silver\_date | Contains all the dates for date related queries |

for more reference : dir - store/US1 &US2

[RDA\_Schema\_Documentation.docx](https://docs.google.com/document/d/1CYaDpRZtjOSud14yfvRi2jTUIc6iaXpK/edit?usp=sharing&ouid=105004890288741700290&rtpof=true&sd=true)

## Methodology

### Medallion Architecture: Implementation Flow

Our methodology strictly follows the Medallion Architecture, ensuring data quality and structural integrity across two core stages. The Airflow DAG orchestrates these stages sequentially.

**4.1.1 Ingestion (Bronze Layer)**

This stage reads CSVs, validates schema, cleans, and writes to MySQL bronze table.

It Scans `/opt/airflow/data/raw or /data/raw` for new CSVs (incremental via metadata\_table timestamps)

It Reads CSV with header/inferSchema, applies utils (lower\_columns, validate\_filename).

Then it Validates the filename and schema then writes to MySQL database.

**4.1.2 Cleaning and Transformation (Silver Layer)**

This stage transforms bronze to silver (cleaning, rules, imputation), builds dimensions (e.g., customer, date), and creates a fact table (silver\_retail).

It reads a bronze table, applies clean\_transform (utils: nulls, dates), apply\_business\_rules (quarantine invalids), transform\_to\_silver (impute amounts, add timestamps). Builds dims (e.g., silver\_customer via upsert\_dimension), joins to fact table (silver\_retail) and Computes metrics (e.g., null counts), writes to dq\_report table and CSV.

### Star Schema Design

The Star Schema defines the analytical structure of the Gold Layer database, optimized for fast querying and BI tool performance. The model consists of a central Fact Table (silver\_retail) linked via foreign keys to the smaller Dimension Tables (silver\_customer ,silver\_product…etc). This structure simplifies data retrieval for analytical reports.

### Version Control

We use Git for version control across all code assets to ensure a stable and auditable development history. All source code is managed in a central repository. Airflow further handles the versioning of the DAG file (ingest\_clean\_dag.py), providing a transparent audit trail of changes to the pipeline's workflow.

### Testing and Bug Fixing

Data Quality (DQ) and system reliability are core components of our testing methodology.

* Quarantine Mechanism: The most critical test is the **Quarantine Layer**. Records failing business validation checks (outliers, negative amounts, invalid IDs) are automatically diverted to a separate **Parquet file**, preventing them from corrupting the final Gold Layer.
* Idempotency Test: The file-based logging ensures that the ingestion job can be safely re-run without duplicating files, providing a reliable recovery mechanism.
* Modularize the code and apply changes to fasten the execution of pipelines.

## Implementation

### Stage 1: Ingestion (Bronze Layer)

### Purpose : Extract and validate raw data from CSV files, handling errors and logging for reliability.

### Flow :

### Input: CSVs in `/opt/airflow/data/raw` or `/data/raw` (e.g., "retail\_YYYYMMDD\_HHMMSS.csv").

### Processing:

### Scan for new files (incremental via metadata\_table).

### Validate filename and schema (using utils.py functions).

### Clean: Lowercase columns, cast types, add metadata (ingestion time, source file).

### Quarantine: Bad files/rows to `/opt/airflow/logs/quarantine` or `logs/quarantine`.

### Output: Write to MySQL bronze\_retail table (append mode).

### Code Files Involved :

### ingest\_to\_bronze.py : Main script for ingestion.

### utils.py: Helpers like validate\_schema, cast\_and\_clean and metadata

### write.py: Writes to bronze and logs.

### Dependencies: Config for paths/DB and Airflow for scheduling.

### Performance : Optimized by using dynamic allocation.

### Stage 2: Transformation (Silver Layer)

### This stage involves the most complex logic:

**Purpose**: Clean and enrich bronze data, apply business rules, and build warehouse structures for analytics.

**Flow**:

**Input**: Data from bronze\_retail table.

**Processing**:

* Cleaning: Use clean\_transform (utils.py) for nulls, date parsing, type casting.
* Business Rules: Apply apply\_business\_rules (transform\_and\_report.py) to validate and quarantine invalids.
* Enrichment: Transform to silver schema (e.g., impute amounts, add timestamps).
* Warehouse Build: Create dimensions (e.g., silver\_customer) with surrogate keys; join to fact table (silver\_retail) after this we get the Silver tables in MYSQL and DQ metrics.

**Code Files Involved:**

clean\_to\_silver.py: Orchestrates transformation and cleaning.

dimensions.py: Builds dims , fact and upsert.

transform\_and\_report.py: Applies rules and metrics.

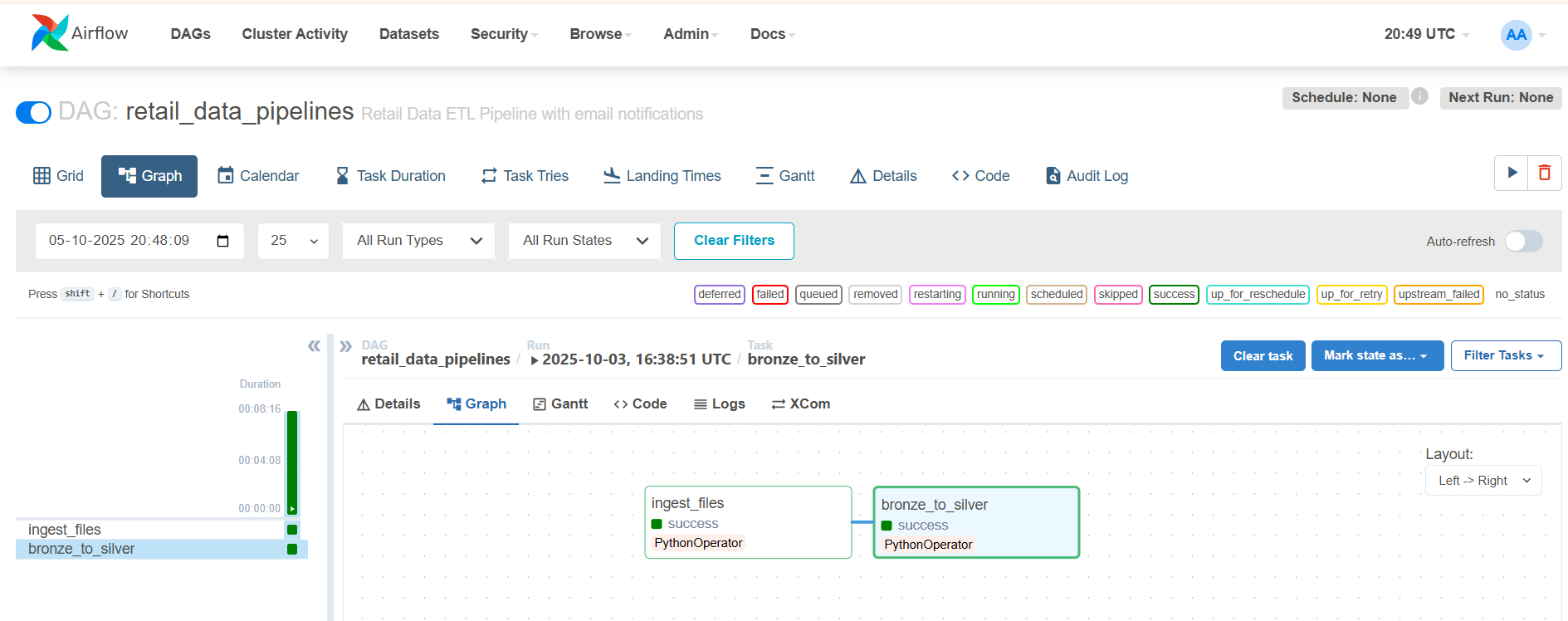
Dependencies: Stage 1 output.

Performance: Uses caching and broadcast joins to process all rows in optimum time.

## Results and Discussions

#### Airflow Orchestration (Pipeline Reliability)

The pipeline is successfully deployed with clear dependencies: Ingestion → Cleaning → Building tables. This structure provides reliability, ensuring that the heavy transformation workload only begins after the initial file ingest is confirmed complete. The separate tasks allow for easy recovery and minimal wasted processing time upon failure.



#### Business Insights Generation

The clean data enables generation of actionable business metrics using Power BI:

* **Product Performance:** SQL analysis confirms top/bottom selling brands and categories and average rating.
* **Operational Efficiency:** KPIs are accurately calculated in Power BI, highlighting bottlenecks in the order fulfillment process.
* **Customer Insights:** Segmentation by demographics and spending habits is made possible by the clean customer dimension table.

for more reference Power BI documentation : [Sarthak\_PowerBI\_Documentation.docx](https://docs.google.com/document/d/1e4eX8CWUUJn6zurVvUjG3DNc1yFo-ScD/edit?usp=sharing&ouid=105004890288741700290&rtpof=true&sd=true)



## Challenges Faced

While working on this project, several challenges were encountered that required creative solutions and optimizations are :

* **Schema Mismatches**: Extra/missing cols quarantined but may log errors if MySQL strict. Fix: Update StructType dynamically.
* **Incremental Gaps**: If metadata\_table fails it’s handled by try-except.
* **Airflow Init Fails**: Permissions (AIRFLOW\_UID=50000). Fix: Run as root once, then switch.
* **MySQL Port Conflict**: 3307 vs. 3306 change if local MySQL runs. Healthcheck retries help.
* **Spark in Docker**: OOM if low RAM. Fix: Set increase the memory.
* **Performance Considerations**:
* Ingestion: For 1M rows or more use repartition.
* DAG Scaling: CeleryExecutor handles parallelism; set max\_active\_tasks=2 to avoid overload.
* Docker Overhead: Startup ~2 min; use pre-built image. Volumes for I/O speed.
* Optimizations: Cache DFs in PySpark, batchsize=10000 for writes. Monitor with Airflow.
* Metrics: Track rows\_processed in ingestion\_log and used repartition.

## 8. Conclusion and Future Scope

### Conclusion

The Retail Data Analytics Pipeline successfully transforms raw retail transactions into a clean, structured, and analytics-ready dataset. The project follows modern data engineering practices with proper logging, monitoring, orchestration, and a well-designed dimensional model.

### Future Scope

To enhance this project's usability and functionality, the following improvements are planned for future development:

* + - Incorporate machine learning models for customer churn prediction.
    - **Airflow Alerting:** Implement automated monitoring and alerting on the Quarantine Data Path to notify stakeholders immediately when data quality issues spike.
    - **Performance Optimization:** Implement optimized partition schemes (e.g., partitioning the Fact table by Date) to further accelerate query times in the MySQL database.