



Building an End-to-End Data Pipeline: Unlocking E-commerce Insights from the Olist Dataset

By Gowry

TABLE OF CONTENTS

| Section | Title & Subsections | Pages |
|---------|---|-----------------------|
| 1 | Project Overview <ul style="list-style-type: none">• Problem Statement• Scope• Objectives | 3 |
| 2 | Data Source | 4 |
| 3 | Data Architecture & Workflow <ul style="list-style-type: none">• Cloud Storage Recommendation• Data Architecture• Teck Stack• End-to-End Workflow<ul style="list-style-type: none">◦ Data Ingestion (Bronze)◦ Data Transformation (Silver)◦ Data Modeling (Gold)• Data Architecture Best Practices | 5-13 |
| 4 | Data Validation & Checks <ul style="list-style-type: none">• Validation Summary Table | 14 - |
| 5 | Data Schema Modeling (Gold Lakehouse) | |
| 6 | Data Analysis & Visualisation <ul style="list-style-type: none">• Vendor Performance Dashboard• Recommendations | 18-20 |
| 7 | Challenges & Solutions | 21 |
| 8 | Future Improvements | 22 |

1- PROJECT OVERVIEW

Problem Statement

To deliver relevant business intelligence to Olist's management regarding how the vendors are performing on the platform across the time period of 2016 - 2018.

Which vendors are performing well in terms of sales, ratings and delivery? How can Olist use vendor rankings to improve sales and overall marketplace performance?

Project Scope

- Creating a centralized, structured dataset from raw Olist E-commerce data from 2016 to 2018.
- Automating the data pipeline to support scalable and reproducible analysis.
- Providing actionable insights and recommendations to the Olist management to make informed decisions on Vendor Performance on the platform.

Objectives

- Build An End-To-End Data Engineering Pipeline

- Recommend a suitable cloud data storage solution to Olist's management
- Ingest, clean, transform, and store Olist E-commerce data from 2016-2018, covering the entire workflow from raw data to structured datasets.
- Maintain data consistency and reliability through validation and cleaning processes.

- Implement Vendor Performance-Specific Statistics And Generate Insights

- Compute Gross Merchandise Value for vendors to evaluate their individual contributions
- Calculate the average days of handover (order purchase date to order delivered carrier date) to determine time taken for order to reach customer.
- Categorise vendors into 4 categories to determine performance to support informed decision-making for next step actions

- Visualize Vendor Performance For Stakeholders

- Create interactive dashboard for actionable insights and monitoring of vendor performance

2 - DATA SOURCE

Data source inventory table

| Dataset | Data Format | Files | Description | Location | Notes |
|---|-------------|--|---|------------------------|---|
| Olist Brazilian E-Commerce Dataset | .csv | olist_customers_dataset.csv olist_geolocation_dataset.csv olist_order_items_dataset.csv olist_order_payments_dataset.csv olist_order_reviews_dataset.csv olist_orders_dataset.csv olist_products_dataset.csv olist_sellers_dataset.csv product_category_name_translation.csv (9 files) | Raw data from Olist e-commerce platform for location, sellers info, customers info, reviews and order related info. | Kaggle | - Relational dataset - Historical data from 2016-2018 |
| City Codes dataset | .csv | br-city-codes.csv (1 file) | Brazilian city codes, city names and state information | github | - Additional resource used as geolocations raw data is inconsistent |

3 - DATA ARCHITECTURE & WORKFLOW

Cloud Storage Recommendation

Microsoft Fabric was selected as the optimal data and analytics platform due to its provision of a unified, all-in-one Software-as-a-Service (SaaS) solution. This strategic consolidation simplifies the entire data lifecycle, from initial ingestion and engineering through to advanced analysis and visualization.

Specifically, Fabric provides a superior value proposition over traditional, piecemeal cloud data architectures by offering:

- Unified Environment
- Reduced Complexity & Overhead
- Cost Efficiency & Predictability
- Core BI Strength

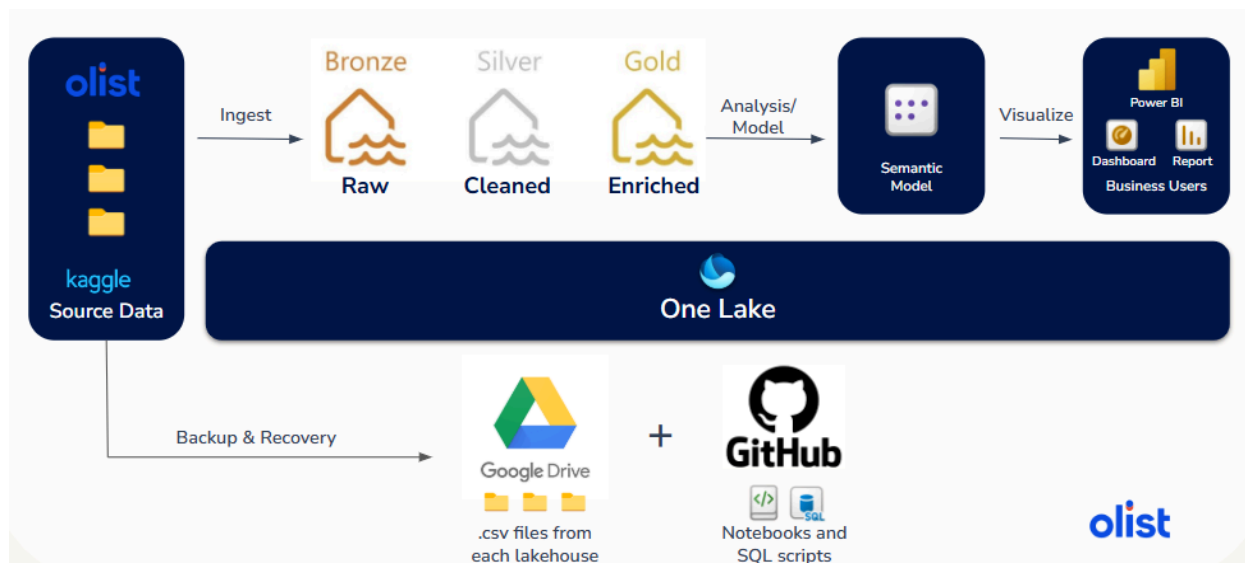
For Olist management's consideration, a side-by-side comparison with a traditional Azure setup using the Total Cost of Ownership (TCO) is illustrated:

| Cost Factor | Traditional Azure Data Platform (Example: ADLS Gen2 Hot Tier) | Microsoft Fabric (OneLake Storage) |
|--------------------------------------|---|--|
| Price/GB/Month | ≈\$0.020–\$0.023 | ≈\$0.023 |
| Total Storage Cost (10TB / 10000GB) | ≈\$200–\$230 USD/Month | ≈\$230 USD/Month |
| Capacity SKU Cost | ~\$6944.50/Month 1 year commitment for DWU1000 (approx. in East US) with Dedicated SQL, PLUS Spark, Pipelines, Serverless SQL, etc. | ~\$5,002.67/Month (1 year commitment for F64 for enterprise grade capacity) |
| Key Difference | Requires separate billing for every data operation (reads, writes, deletes, etc.) which can drastically increase the final bill. | Storage is decoupled from compute, and transactions/operations are included in the Fabric Capacity (CU) cost, simplifying the storage bill. |
| Total Cost of Ownership (TCO) Impact | The monthly salary of a dedicated Data Engineer often outweighs the marginal savings in direct cloud storage and compute, making the platform substantially more expensive to operate. | Reduced need for highly specialized infrastructure personnel drastically lowers the operational cost and allows team members to focus on delivering business value. |

This project was hence developed using Microsoft Fabric, which provides an end-to-end Software as a Service (SaaS) solution which Olist can use for data ingestion, transformation, modelling and analysis under a unified environment.

Data Architecture

Leveraging a **Medallion Architecture**, the systematic flow of data was ensured, guaranteeing progressive **quality, governance, and reliability** from raw ingestion to final analytical reporting.



Tech Stack

| Tool | Remarks |
|--|---|
| Microsoft Fabric Workspace | A workspace dedicated for e-commerce related data. The management can choose to set up separate workspaces for other internal functions such as HR. |
| Lakehouses | Three separate lakehouses within the workspace representing the medallion structure. |
| Notebooks (PySpark, PySparkSQL) | For data ingestion and transformation. |
| | Automated data validation and quality checks integrated within the data pipeline to ensure data accuracy, completeness and consistency before progressing to the next stage. |
| Dataflow Gen 2 | For data clean up from Bronze to Silver lakehouse for analysis by downstream users. |
| Power BI | For data visualization, reporting and creation of dashboards. |
| Diagnostics Lakehouse (Monitoring KQL Database) | Lakehouse configured for read-only access to monitor usage of the workspace and investigate issues with diagnostics events. Crucial for auditing, cost analysis, and troubleshooting. |
| Backup & Recovery | <p>Google drive - store lakehouse tables as .csv files for easy reproducibility to other data storage solutions.</p> <p>Github - Use Fabric's Github integration feature for code version control and backup.</p> |

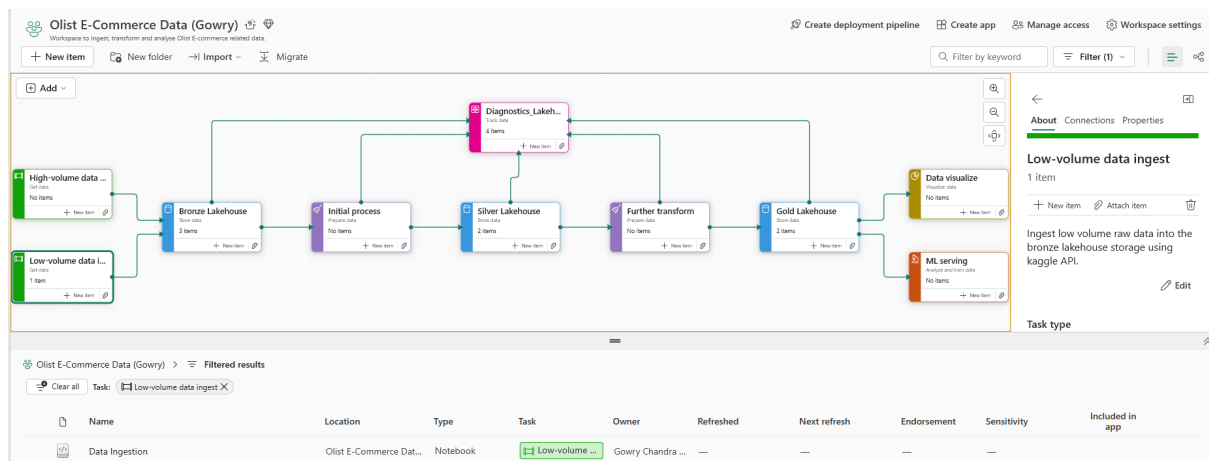
End-to-End Workflow

1. Data Ingestion (Bronze Lakehouse)

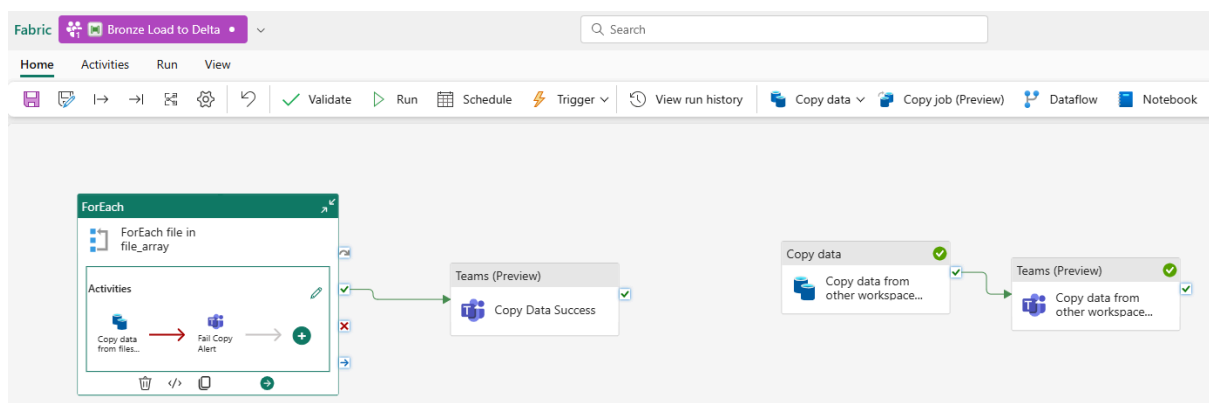
- Raw datasets were ingested into the Bronze Lakehouse directly from Kaggle via API.
- Data is stored in their original .csv format to preserve source fidelity.

Bronze Load to Delta

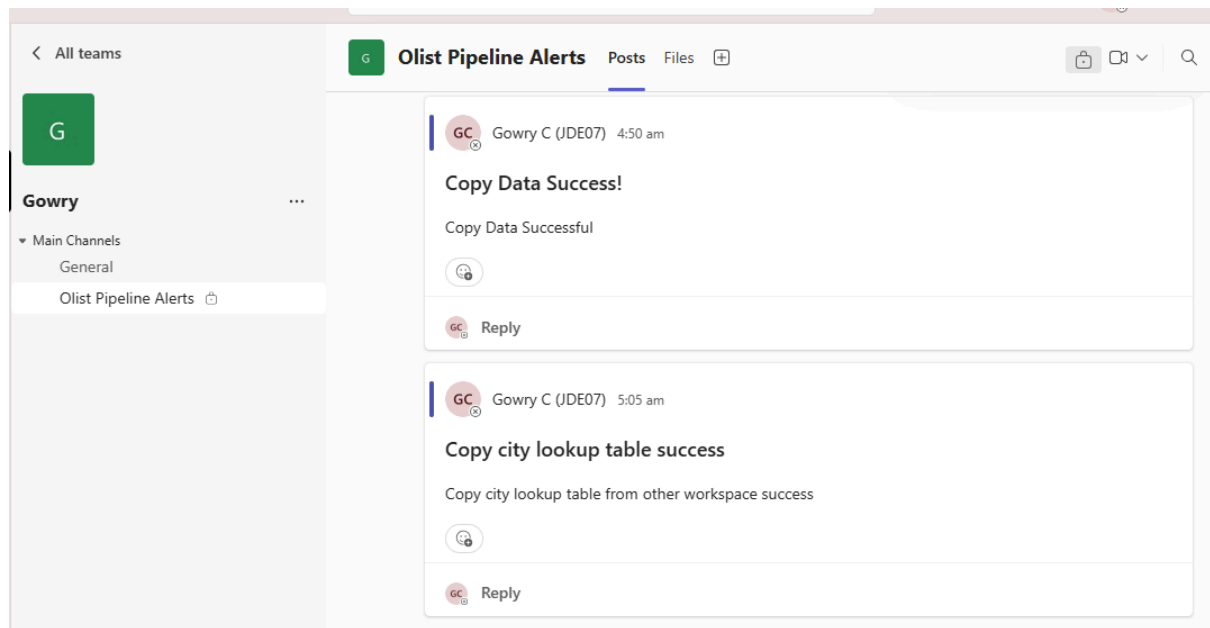
- Automate data ingestion pipeline in Bronze Lakehouse built with Copy Data Pipeline Activity to convert csv files to Delta tables.
- Automate data ingestion of reference table from another workspace with Copy Data Pipeline Activity.
- Store any failures in a log file within the Monitoring Lakehouse.
- Send Alerts on Success and Failure to Teams Channel.



Task: Upload raw data via API



Task: Automated Copy Data Pipeline Activity



Task: Teams Channel Alerts

2. Data CleanUp (Silver Lakehouse)

- Data cleanup on Bronze Delta tables.
- Dataflow Gen 2 tool used for simplification and centralizing of process.
- Maintains scalability for large datasets.

Data Cleaning (Bronze to Silver)

- Consistent data types applied across the different table columns.
- String to Integer for product dimension columns.
- String to Datetime for all timestamp related columns.
- Price values to currency .
- String values trimmed, removed spaces and replace “-” with space.

Reviews table

- Remove additional information from the review_id row (total 5 affected)

| | |
|---|--|
| 2 | ,2017-03-12 00:00:00,2017-03-12 11:38:03 "3a6fd9264a564dd7ab0c54d6a5b0bbe0" |
|---|--|

Example of review_id record with additional information

- Remove invalid rows which have no review_id or other identifiable information to populate.

| review_id | order_id | review_score | review_comment_title | review_comment_message |
|-----------|--------------------------------------|-------------------------------------|----------------------|------------------------|
| 1 | O produto mesmo conectado na ener... | ao tocar musica fica reproduzindo a | [Error] | 2018-01-02 17:25:57 |

Example of invalid data in reviews table

- Remove all true duplicates i.e all columns have repeated values (Total: 66)

Seller & Customers

- Used city_look_up reference table to get correct values of city and state based on the zip code prefix

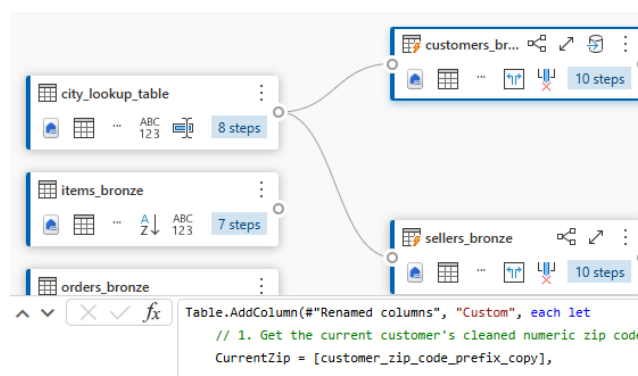


Diagram view of steps performed in Dataflow Gen2

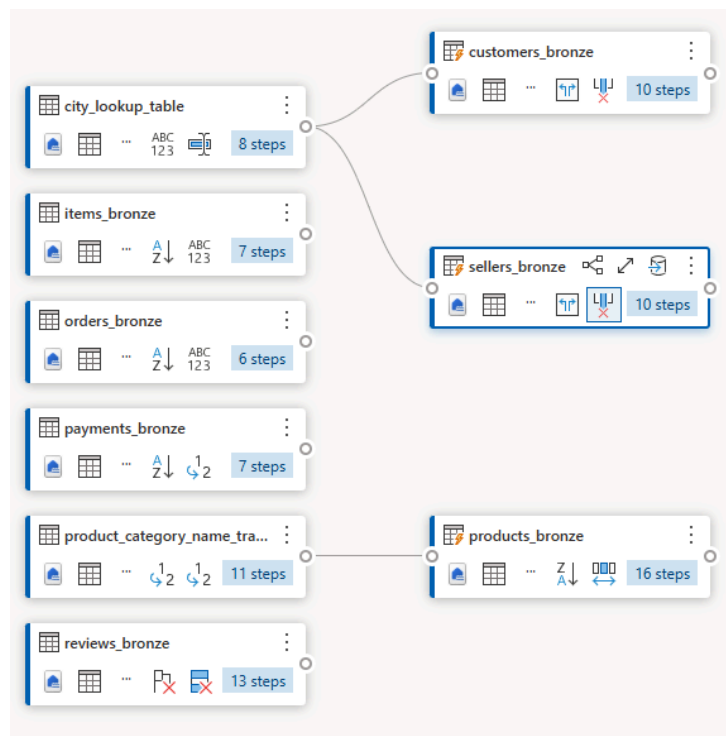


Diagram view of overall data cleanup steps

3. Data Modelling (Gold Lakehouse)

- Galaxy Star Schema implemented.
- PySparkSQL in one Notebook used to populate schema tables.
- Maintains scalability for large datasets.
- Readability of Code.

Data Modelling (Silver to Gold)

- Preserved data for sellers, customers and products as dimension tables.
- Date dimension table created to supplement analysis.
- Fact_reviews table supplemented with order_id, order_status and customer_id.
- Fact_orders table with aggregated columns of total_order_value and total_freight_value.

The screenshot shows a Microsoft Fabric notebook titled "Gold Lakehouse Table Population". The interface includes a top bar with tabs for "Data Cleaning (Bronze to Silver)" and "Gold Lakehouse Table Population". Below the top bar is a navigation pane with "Home", "Edit", "AI tools", "Run", and "View" options. The main workspace displays two code blocks. The first code block, labeled [4], contains PySpark SQL code for creating and writing to the "dim_dates" table. The second code block, labeled [19], contains PySpark SQL code for joining "silver_items" and "silver_orders" tables, selecting specific columns, and writing the result to the "fact_order_items" table. Both code blocks show successful execution status with timestamps.

```
14 .withColumn("day_of_week", F.date_format("date_id", "EEEE")) \
15 .orderBy("date_id")
16
17 spark.sql("DROP TABLE IF EXISTS GoldLakehouse.dim_dates")
18
19 dim_dates.write.format("delta").mode("overwrite").saveAsTable("GoldLakehouse.dim_dates")

[4] ✓ - Command executed in 28 sec 198 ms by Gowry C (JDE07) on 9:20:19 PM, 10/12/25

1 # Populate fact_order_items
2
3 # Read silver items and orders tables
4 silver_items = spark.table("olist_items_cleaned")
5 silver_orders = spark.table("olist_orders_cleaned")
6
7 # Join items with orders to get date_id
8 gold_order_items = silver_items.join(
9     silver_orders,
10     silver_items["order_id"] == silver_orders["order_id"],
11     "inner"
12 ).select(
13     F.col("order_item_id").cast("INT").alias("order_item_id"),
14     F.col("olist_items_cleaned.order_id").alias("order_id"),
15     F.col("product_id").alias("product_id"),
16     F.col("seller_id").alias("seller_id"),
17     F.col("shipping_limit_date").cast("timestamp").alias("shipping_limit_date"),
18     F.col("price").cast("FLOAT").alias("price"),
19     F.col("freight_value").cast("FLOAT").alias("freight_value"),
20     F.col("order_purchase_timestamp").cast("date").alias("date_id")
21 )
22
23 # Overwrite GoldLakehouse.fact_order_items
24 gold_order_items.write.format("delta").mode("overwrite").option("overwriteSchema", "true").saveAsTable("GoldLakehouse.fact_order_items")
25

[19] ✓ - Command executed in 14 sec 670 ms by Gowry C (JDE07) on 12:11:53 PM, 10/07/25
```

Using PySpark SQL to populate Gold lakehouse Delta Tables

| | Name | Location | Type | Task | Owner |
|--|---------------------------------|---------------------------|----------|----------------|-------------------|
| | Gold Lakehouse Table Population | datasquirrels/Migratio... | Notebook | Gold Lakeho... | Gowry Chandra ... |

Singular Notebook to populate Gold Lakehouse

Data Architecture Best Practices

Separate Diagnostic Lakehouse

- Set up of Diagnostic Lakehouse with Monitoring Datahouse. This will house all the failure logs for easy analysis in a read-only format to ensure data traceability.
- Use a centralised system for logging the errors with a separate lakehouse for data clarity.

Optimizing Write (Bronze Delta tables)

- Set the Delta Lake table optimization by writing them to parquet file format using V-Order in Copy Data pipeline activity.

Parallel Execution

- Increase processing speed and reduce runtime by executing copy data tasks concurrently.

Real-Time Alerts

- Sending real time notifications for failures on automated pipeline processes.







Data Recovery

- Data persistence for backup and recovery in a different system for reproducibility.

Fabric Tools Usage

- Usage of different tools to ensure scalability and fast processing for large data.

4 - DATA VALIDATION CHECK & QUALITY

| Name | Task | Type | Owner |
|--|---|----------|--------------------|
|  Gold Lakehouse Validation |  Gold Validation | Notebook | Gowry Chandra Sega |
|  Silver Lakehouse Validation |  Silver Lakehouse V... | Notebook | Gowry Chandra Sega |
|  Silver Validation SQL Script |  Silver Lakehouse V... | Notebook | Gowry Chandra Sega |

PySpark Notebooks and SQL Script to perform validation checks

Silver Lakehouse Data Validation

| Data Quality Dimension | Purpose | Specified Checks |
|------------------------|---|---|
| Completeness | Are all required data values present? (Focus on Nulls) | Verified no Null values in all mandatory ID fields (<code>order_id</code> , <code>customer_id</code> , <code>product_id</code> , <code>seller_id</code>). |
| Validity | Does the data conform to defined schema, format, or range? (Focus on Schema & Outliers) | <p>Enforced consistent data types: IDs as STRING, dates as DATE/TIMESTAMP, and monetary fields as INT/FLOAT.</p> <p>Checked for invalid values in critical fields (e.g., non-negative prices/freight, review scores strictly between 1–5).</p> |
| Consistency | Is the data logical, and does it align across tables? (Focus on Dates & Foreign Keys) | <p>Validated referential integrity (Foreign Keys) to ensure all referenced IDs exist and prevent orphaned fact records.</p> <p>Confirmed logical date sequences (purchase → approval → shipment → delivery) to ensure a realistic order lifecycle.</p> <p>Only 1 record was found not to have a logical date sequence. No</p> |

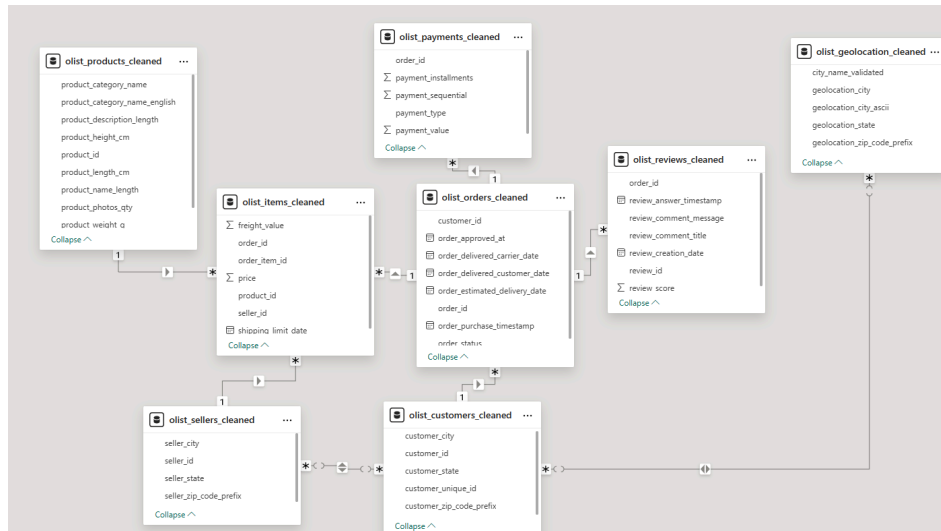
| | | |
|------------|---|--|
| | | further changes were made and it was excluded from analysis. |
| Uniqueness | Are there duplicate records where they should not exist? (Focus on Primary Keys & Duplicates) | <p>Enforced uniqueness in all Dimension tables (customers, sellers, products) to prevent double-counting.</p> <p>Confirmed and handled known duplication issues in primary identifiers - <code>order_id</code> & <code>review_id</code></p> |
| Integrity | Does the data comply with core business rules? (Focus on Business Logic) | <p>Verified financial integrity by ensuring total payments match the calculated order total (items + freight).</p> <p>Enforced the rule that each order must have ≥ 1 item and ≥ 1 payment record to prevent incomplete facts.</p> |

Gold Lakehouse Data Validation

| Data Quality Dimension | Check (What is verified) | Why It's Critical (Impact on downstream users) |
|---------------------------|---|--|
| Integrity (Star Schema) | Referential Integrity: Every fact table Foreign Key (e.g., <code>customer_id</code> , <code>date_id</code> , <code>product_id</code>) must exist in the corresponding dimension table. | Guarantees Reliable Joins: Ensures the star schema works flawlessly, preventing broken relationships and orphaned fact records in BI tools. |
| Completeness | Mandatory Key Check: Verify that no surrogate keys (e.g., <code>date_id</code> , <code>customer_id</code>) are NULL in the final Fact tables. | Prevents Missing Data: Ensures all fact records can successfully connect to their dimension attributes, preventing data loss during analysis. * Exception of 8 records where only the <code>order_id</code> is available. |
| Accuracy (Reconciliation) | Metric Alignment: Review average scores in <code>fact_reviews</code> must exactly match the aggregated review scores used in the final <code>fact_orders</code> table. | Ensures Financial Truth: Guarantees that aggregated measures are consistent across all views, preserving trust in core business metrics. |
| Performance (Usability) | Query Efficiency: Validate that Star Schema joins between Facts and Dimensions complete within acceptable time thresholds. | Keeps customer reviews consistent across tables. |

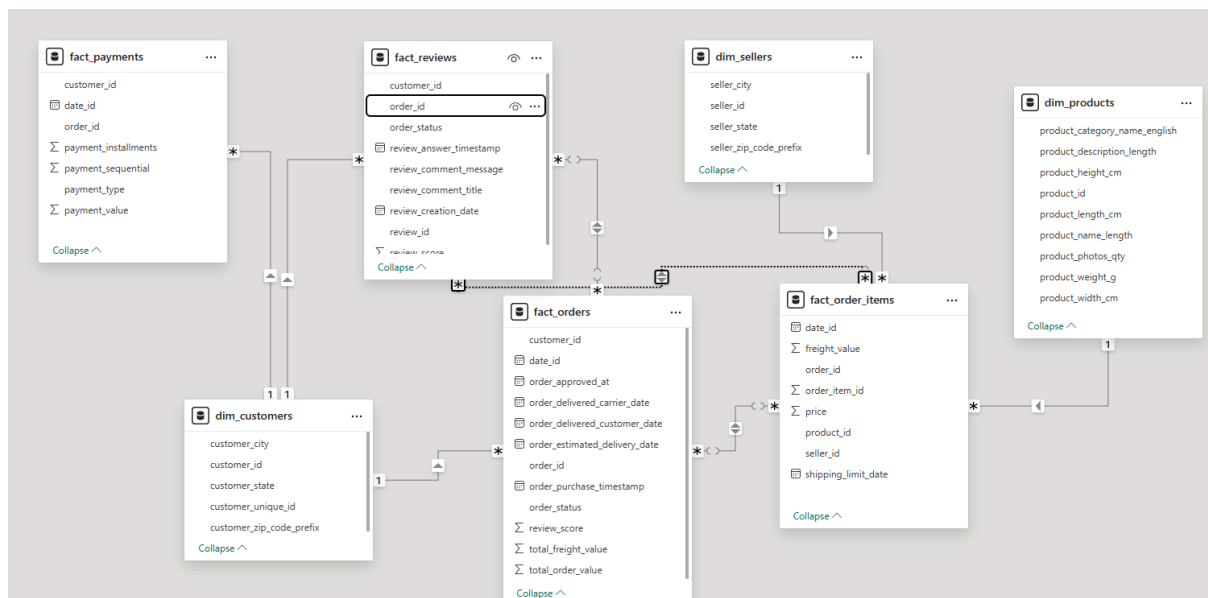
5 - DATA SCHEMA MODELING (GOLD LAKEHOUSE)

Silver Lakehouse Schema



Final Gold Lakehouse Schema (Galaxy Schema)

Multiple interconnected Fact tables that share common dimension tables.



6 - DATA ANALYSIS AND VISUALISATION

Vendor Performance Summary & Insights

- Audience: Olist Leadership (Executives, VPs).
- Goal: Explain who the best/worst vendors are and why, leading to a tactical plan for improvement.
- Dashboard Type: Explanatory (Tells a clear story, pre-attentive data) and Tactical (Focuses on metrics that drive action, like fixing late delivery vendors).

1. Gross Merchandise Value

- Calculated as the product price multiplied by the number of items sold.

$$\text{GMV} = \text{Price} \times \text{No. of Items Sold}$$

- Measure each vendor's sales contribution.

2. Average Days to Handover

- Defined as the time between `order_purchase_timestamp` and `order_delivered_carrier_date`.
- Used as a measure of delivery performance among vendors.
- Overall vendor average was established at **3.67 days**.

3. Benchmarks

- Median GMV (Due to skewness of data): **\$832.41**
 - Sellers with GMV less than \$10 and more than \$200,000.
- Average Review Score: **3.98**

4. Categorisation of Vendors

Two-by-two matrix classifying all vendors into four distinct groups using both benchmarks:

- 1- Top Vendors (High GMV + High Rating)
- 2- At-Risk Vendors (High GMV + Low Rating)
- 3- Growth Potential Vendors (Low GMV + High Rating)
- 4- Low Priority Vendors (Low GMV + Low Rating)

Dashboard & Insights

Summary text

- Key descriptive statistics for at-a-glance insights.
- Majority of high impact sales concentrated with Top Vendors and At-Risk Vendors.
- Immediate priority to **stabilise At-Risk vendor performance** to mitigate critical customer churn risk.

KPI Cards

- High-level overview of key vendor performance metrics:
 - Overall Total Sales Value (GMV) is \$13.76M.
 - Vendors Average Review Score of 3.98.
 - Overall Average On-Time Delivery Rate of 85%.

Top 10 & Bottom 10 Table visuals

- At-Risk category is a pressing concern: contributing over \$582,000 in GMV in top 10
- Bottom 10 vendors, largely categorized as Growth Potential.
- Reinforces the strategic necessity of focusing resources on stabilizing At-Risk vendors performance.

Clustered Bar Chart

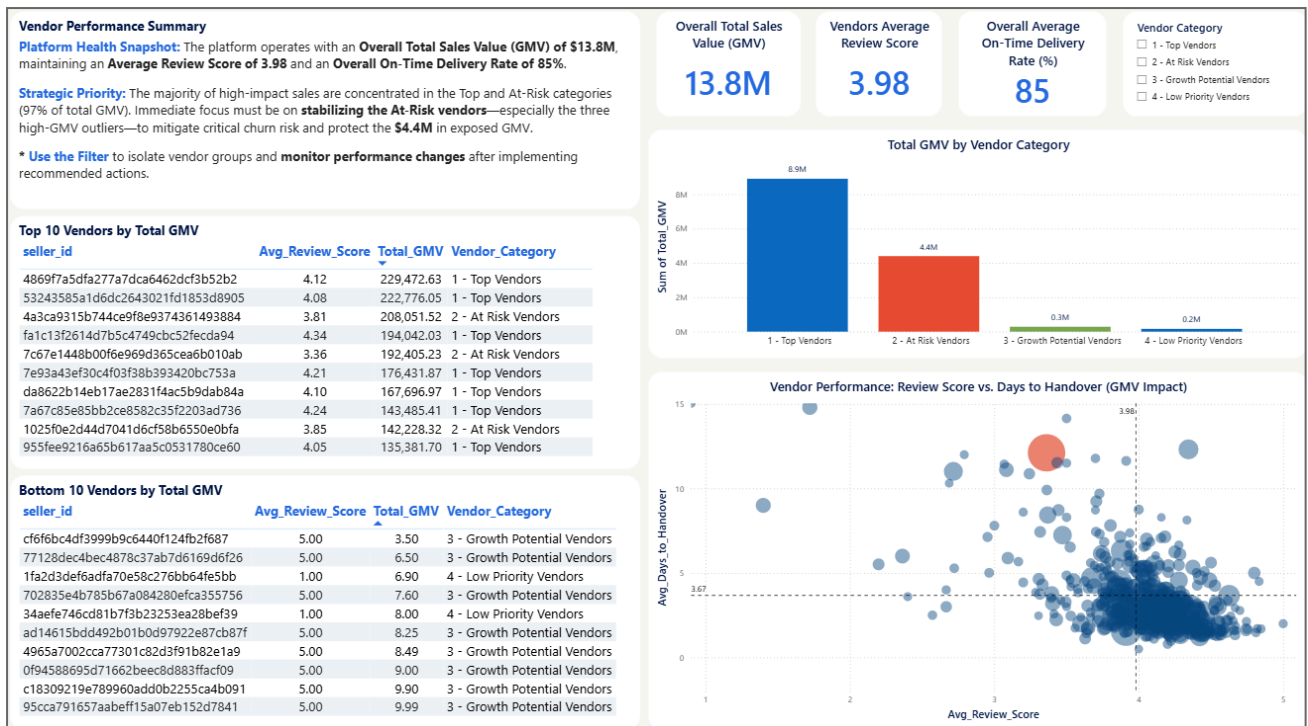
- Top Vendors (\$8.9M) and At-Risk Vendors (\$4.4M) driving ~ **97%** (\$13.3M).
- At-Risk group is a critical priority, as their GMV contribution is **14.6× greater than** the combined sales of the Growth Potential (\$0.3M) and Low Priority (\$0.2M).

Scatter Plot

- Pareto Principle to focus analysis on the 542 vendors contributing to 80% of the Total Gross Merchandise Value (GMV).
- Average Days to Handover (**Y-axis**) against Average Review Score (**X-axis**), with the **dot size representing Total GMV**, to diagnose performance drivers.
- Almost \$100,000 of GMV seen in the contrast orange bubble representing At-Risk Vendors.

Vendor Category Filter

- For users to get insights to each category individually
- Crucially, for monitoring of At-risk vendors performance after corrective actions are implemented.
- Dual purpose of dashboard to show current insights and monitor performance.



Vendor Performance Dashboard

7 - CHALLENGES & SOLUTIONS

Data Quality

Challenge: A lot of dirty data and inconsistent data within and across tables.

Solution: deep dive exploration to ensure good grasp of data before deciding replacement values, removal of nulls, data type consistency, and implementation of mean or median as benchmark.

Tools Usage

Challenge: Many tools within the Fabric environment provide the same output.

Solution: Reading documentation to understand the different purpose of tools before deciding which tools would ensure scalability and reproducibility for expanding data.

Data Recovery Platforms

Challenge: Recommendations for recovery need to be feasible and cost saving.

Solution: Using simple tools such as Google Drive and Github Integration which are already industry standards. They provide less friction for adoption.

8 - Future Improvements

While the current implementation successfully demonstrates the end-to-end data engineering workflow using Microsoft Fabric, several enhancements can be made to further strengthen scalability, automation, and analytical depth:

End-to-end automated pipeline

- Unable to do it this time due to time constraints and a lot of dirty data which required individualised attention.

Great Expectations for Data Validation

- Automate data testing, catch issues early and integrate into automated workflow.

Machine Learning

- As an e-commerce platform, employing suitable machine learning models such as product recommendations can be useful in increasing customer spend on the platform.

Dashboard

- Reports in other areas such as customer acquisition (Sales performance by region) and reviews analysis for a more complete analysis of the platform's performance to Olist Management.