Technical Document: Olist E-commerce End-to-End Data Pipeline

1. **System Overview**

This project aims to build a modern data processing system that integrates traditional Business Intelligence (BI) pipelines. The system is based on the Olist e-commerce dataset and implements an end-to-end workflow, from raw data ingestion and cleaning to modeling and deep dive to business analysis.

Scope and Objectives

- **Data Engineering:** Build a robust ELT pipeline using Meltano and BigQuery, transforming scattered CSV files into a queryable data warehouse.
- **Business Analysis:** Construct dimensional models (Star Schema) using dbt to support multi-dimensional analysis of orders, products, customers, and reviews.

The project covers the entire Data Engineering Lifecycle:

- Ingestion: Meltano (EL)
- Storage: Google BigQuery
- Transformation: dbt (Data Build Tool)
- Analysis: Python

Data Sources and Lineage

Raw Data Sources

The data sources consist of a series of CSV files, loaded using the **tap-csv** extractor. The main entities include:

- **orders:** Core order information
- **order_items:** Order line items
- **customers / raw_sellers:** Buyer and seller information
- **reviews:** Review text
- **products:** Product attributes
- **payments:** Payment information

2. Phase-by-Phase Implementation Plan

Phase 1: Data Ingestion (EL)

Objective: Automated loading of raw CSV data into Google BigQuery.

- Dataset: Brazilian E-Commerce Dataset by Olist.
- Tech Stack:
    - Engine: Meltano (CLI-based ELT tool).
    - Extractor: tap-csv (Configured to read data/raw/*.csv).
    - Loader: target-bigquery (Configured with GCP Service Account).
- Implementation Details:
    - Source Configuration: meltano.yml defines mapping from local files (e.g., olist_orders_dataset.csv) to BigQuery tables (orders).
    - Outcome: Raw tables created in BigQuery dataset olist_raw.

Phase 2: Data Warehouse Design

Objective: Design a schema optimized for OLAP (Online Analytical Processing).

- Architecture: Raw data Schema.
- Tech Stack: Google BigQuery.
- Schema Definition:
    - Fact Tables (Verbs):
        - fct_order_items: Central transaction table (Grain: Line Item). Includes keys to all dimensions and metric columns (price, freight, review_score).
        - fct_payments: Financial transaction table (Grain: Payment Attempt). Logic linked via order_id.
    - Dimension Tables (Nouns): -
        - dim_customers: Denormalized table containing User Profile + Geolocation (Lat/Lng integrated from raw geo data).
        - dim_products: Contains Product ID and English Category Names.
        - dim_sellers: Contains Seller ID and Location (City/State).

Phase 3: ELT (Extract, Load, Transform)

Objective: Clean raw data and model it for business analysis.

- Tech Stack: dbt (data build tool) with dbt-bigquery adapter.
- Workflow:
    - Staging Layer (models/staging):
        - Cleaning: Cast strings to TIMESTAMP/NUMERIC.
        - Standardization: * stg_products.sql: Joins raw_products with raw_category_translation to replace Portuguese categories with English (e.g., beleza_saude → health_beauty).

        - stg_customers.sql: Distinguishes between customer_id (order key) and customer_unique_id (user key).
    - Marts Layer (models/marts):
        - Logic: Joins Staging tables to create the Star Schema.
        - Business Logic Implementation:
            - Filtering: fct_order_items includes logic to handle order_status (e.g., filtering out cancelled orders if required).
            - Derived Metrics: Calculated total_item_value (price + freight_value) at the row level.
- Addressing Specific Analysis Tasks (via SQL Models):
    - Show number of listing products based on location: Solved by joining fct_order_items with dim_sellers (using seller_state).
    - Category Analysis: Solved by dim_products (Standardized English names).
    - Revenue per Category: Aggregation on fct_order_items joined with dim_products.

Phase 5: Data Analysis

Analysis 1: Generate insights of key categories for marketing efforts.

- Tech Stack: Python (Pandas, Matplotlib) + Bigquery.
- Workflow:
    1. Connector: Use google-cloud-bigquery to fetch fct_order_items into a generic Dataframe.
    2. In-Memory OLAP: Use DuckDB to query the Dataframe using SQL for high-performance aggregation.
    3. Data Pre-processing:
        - Examine the data to extract key components either directly or by tabulating a more compact dataset for focused analysis e.g. order_prod_time_price_citystate for the analysis of one of the business questions.
        - Apply progressive data amalgamation to enable checks and cleaning-up.
    4. Feature engineering:
        - Create new columns to facilitate further analysis e.g. splitting up time-stamp to date and time columns.
        - Truncate redundant data to streamline focused analysis e.g. niche_list for one of the business questions.
    5. Aggregation Logic:

        - Identify the logical Key of the desired dataset to facilitate targeted grouping e.g. customer_id and not customer_unique_id should be used when investigating product sales counts.
        - Apply appropriate aggregation logic e.g. day, time, product category to pin-point the critical business consideration figures.
    6. Validation:
        - Cross-referencing figures e.g. sales figures against the source database totals or sub-totals.
    7. Visualizations:
        - Sales overview charts.
        - Sales volume overview by categories.
        - Top 10 low- and high-price product bar charts.
        - Day of week, time of day order pie charts.
        - Stacked bar chart of targeted product category sales by city.

Analysis 2: How delivery performance impacts customer experience and loyalty on the Olist e-commerce platform.

Using historical order, review, customer, and geolocation data, we examine:

1. How delivery delays affect customer satisfaction
2. Whether late delivery influences repeat purchase behaviour
3. Which regions pose the highest operational risk due to late delivery

The analysis culminates in three core visualisations that translate operational delivery metrics into customer-centric and business-relevant insights.

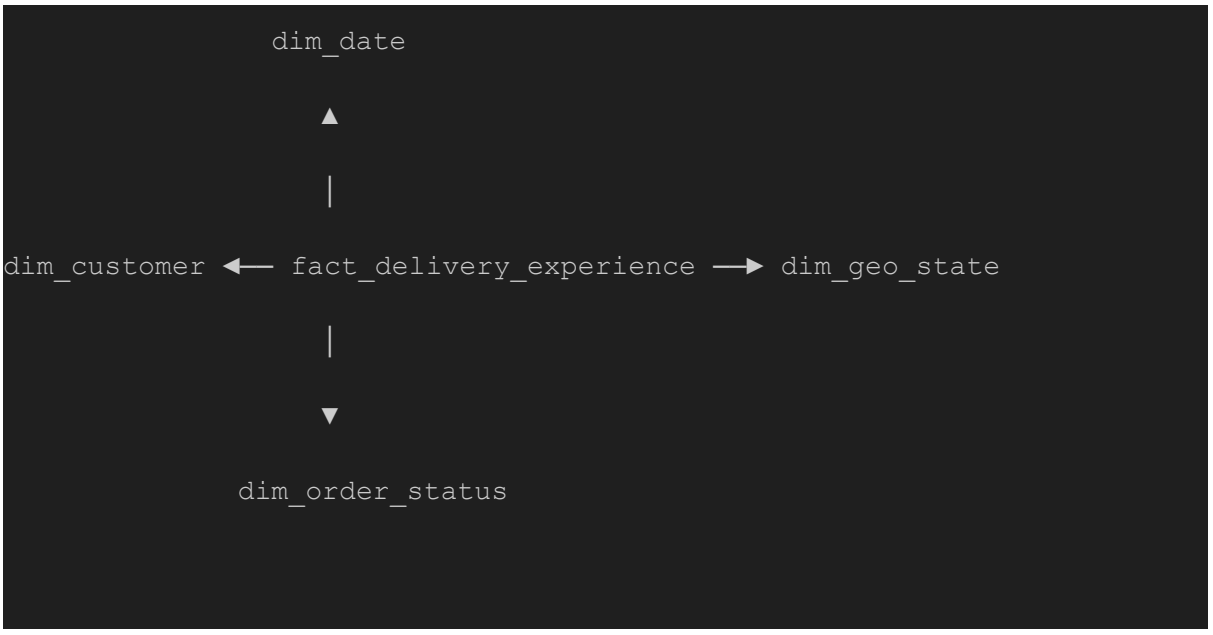
Data Sources

The following datasets from the Olist public dataset were used:

- **Orders dataset**: order timestamps and delivery estimates
- **Order reviews dataset**: customer review scores
- **Customers dataset**: customer identifiers and state-level location
- **Geolocation dataset**: latitude and longitude by zip code

Each dataset was ingested in its raw form to preserve data fidelity before transformation and analysis.

Schema Design



Engineering & Metric Definitions

To support business interpretation, several engineered features were created from the raw data:

Delivery Delay (Days)

Delivery delay was calculated as:

> **Delivery Delay (days) = Actual delivery date − Estimated delivery date**

- Negative values indicate early delivery
- Zero indicates on-time delivery
- Positive values indicate late delivery

Delivery Delay Buckets

Delivery delays were grouped into business-friendly categories:

- **On time / Early**: delay ≤ 0 days
- **1–3 days late**
- **4–7 days late**
- **8+ days late**

These buckets allow executives to reason about delivery performance without relying on raw timestamps.

Repeat Buyer Indicator

A customer was labelled as a repeat buyer if they placed **more than one order** across the dataset period. This binary flag was used to calculate repeat purchase rates.

Visualisation 1 — Delivery Delay vs Customer Satisfaction

Metric Used

- **Average Review Score** (1–5 scale)

What the Visual Shows

This bar chart compares average customer review scores across delivery delay categories.

Key Insight

- On-time / early deliveries average **~4.3 out of 5**
- Deliveries delayed by **8+ days** average **~1.7 out of 5**
- This represents an approximate **60% decline in customer satisfaction**

Interpretation

Customer satisfaction is **highly sensitive to delivery delays**, with a sharp decline once delays exceed 3–4 days. This indicates a clear threshold beyond which customer experience deteriorates rapidly.

Visualisation 2 — Delivery Delay vs Repeat Purchase Rate

Metric Used

- **Repeat Purchase Rate**
  (Percentage of customers who made more than one purchase)

What the Visual Shows

This bar chart illustrates how repeat purchase behaviour changes across delivery delay categories.

Key Insight

- On-time / early deliveries: **~6.6–6.8% repeat rate**
- 8+ days late deliveries: **~5.7–6.0% repeat rate**
- Approximately **10–15% relative decline in loyalty**

Interpretation

Unlike satisfaction, loyalty declines more gradually. Customers may tolerate poor delivery once, but repeated exposure to delays erodes trust over time. This makes loyalty a **lagging indicator** of delivery performance issues.

Visualisation 3 — Regional Late Delivery Risk Map

Metrics Used

- **Percentage of late deliveries**
- **Average number of days late (when late)**
- **Late Delivery Risk Score**

Risk Score Definition

The late delivery risk score was defined as:

Late Risk Score = (% of late orders) × (average days late when late)

This represents the **expected late-delivery days per order** for a given region.
It is **not a score out of 5**, but a composite operational risk metric.

What the Visual Shows

A choropleth map highlighting regional delivery risk across Brazilian states, with markers indicating the highest-risk areas.

Key Insight

Three states consistently emerge as high-risk:

- **RR — Roraima**
- **SE — Sergipe**
- **AP — Amapá**

These states combine:

- High frequency of late deliveries
- Severe delays when deliveries are late

Interpretation

Delivery risk is **not evenly distributed**. Targeted interventions in these regions are likely to deliver disproportionately high returns compared to blanket nationwide improvements.

Business Implications

Across the three visualisations, several consistent themes emerge:

1. Customer satisfaction reacts **immediately** to delivery delays
2. Loyalty erosion occurs **gradually** and is harder to detect early
3. Regional delivery performance varies significantly and requires targeted action

Delivery performance should therefore be treated as a **leading indicator of customer experience**, not merely an operational metric.

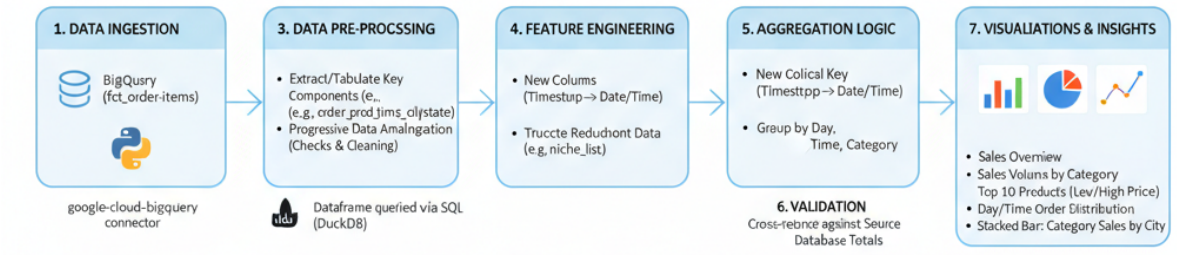
Recommendations

Based on the findings:

- Intervene **before delivery delays exceed 3 days**
- Monitor review score declines as an **early warning signal**
- Treat delivery performance as part of the **end-to-end customer experience**
- Prioritise **RR, SE, and AP** for:
    - Logistics partner review
    - SLA adjustments
    - Regional routing or capacity optimisation

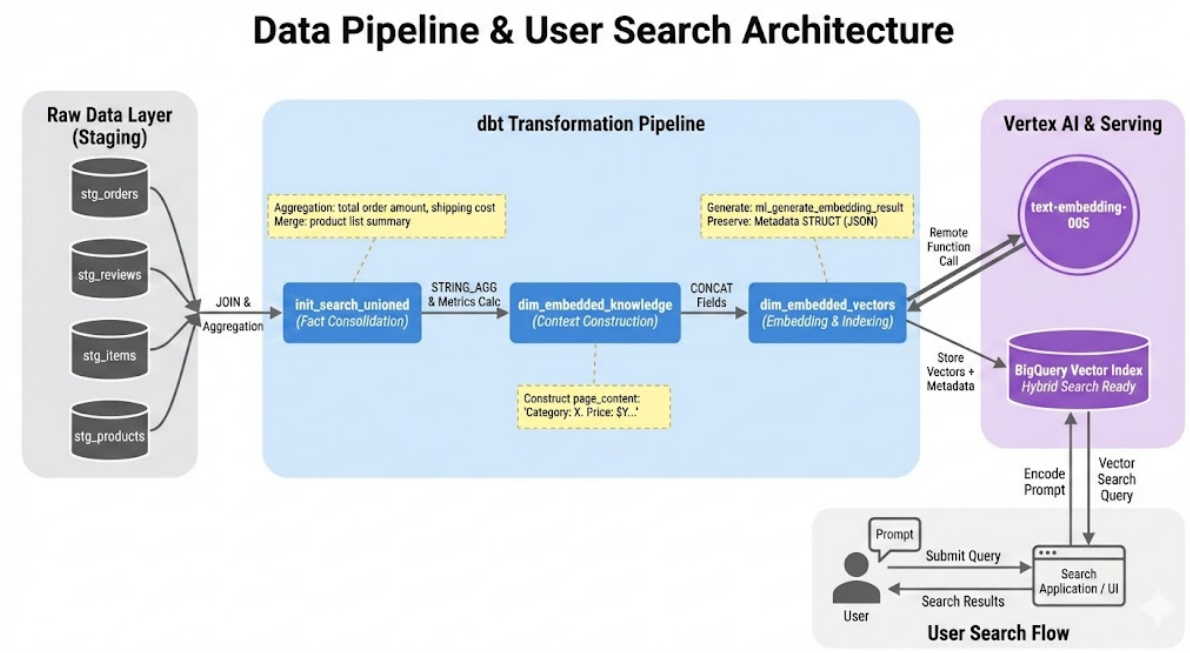
Conclusion

This analysis demonstrates that delivery performance has a direct and measurable impact on customer satisfaction, loyalty, and regional operational risk. By combining operational metrics with customer outcomes, Olist can move from reactive issue resolution to proactive experience management.



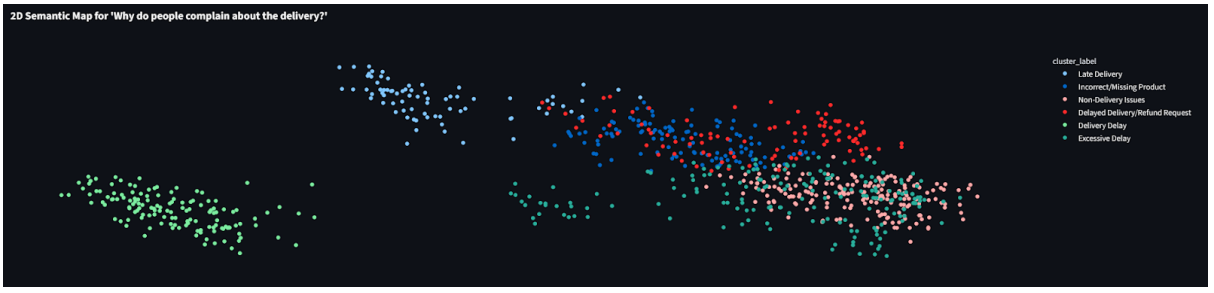
Analysis 3 :Qualitative/Insight Analyst based on Hybrid RAG (Voice of the Customer)

- Tech Stack: Meltano → Bigquery → DBT→RAG
- Workflow:
    - Data Preparation: dbt aggregates raw e-commerce data into a “wide table” and constructs semantic text.
    - Vectorization: BigQuery calls the Vertex AI Embedding API to generate vectors in-database and build indexes.
    - Query Routing: A Python application determines user intent and routes requests to either “SQL analytical queries” or “vector semantic search.”
    - Hybrid Retrieval: Combines vector distance (Cosine) + keyword matching (Keyword) + metadata filtering (Metadata Filter) to retrieve relevant context.
    - Answer Generation: The LLM generates the final answer based on the retrieved context.



Analyzable Question Types:

- **Descriptive Statistics and Aggregation Queries (Descriptive Statistics) :** Quantitative analysis of specific metrics, where precise answers can be obtained solely through SQL aggregation.
    - **Example Questions:**
        - "What was the average shipping cost in Rio de Janeiro during the fourth quarter of 2017?"
        - "List the top 5 best-selling products in the 'health_beauty' category."
        - "How many orders in the São Paulo region are in a 'shipped' but 'not delivered' status?"
- **Qualitative Attribution and Diagnostic Analysis (Diagnostic & Root Cause Analysis) :** Open-ended questions focused on "why" or "specific behaviors," requiring a combination of structured filtering and unstructured text mining.
    - **Example Questions:**
        - "Why were ratings generally low for the 'bed_bath_table' category in March 2018?"
        - "Find all orders that were delivered on time but still received negative reviews—what were users mainly complaining about?"
        - "Analyze the specific feedback users gave about logistics packaging in the 'pet supplies' category."
    - **Why do people complain about the delivery?**



Delivery Complaint Analysis: Semantic Map Insights

1. Macro Perspective: Two Distinct Types of Delivery Crises
   The semantic map reveals that delivery complaints naturally split into two separate clusters—the isolated island on the left vs. the dense continent on the right—indicating that we are facing two fundamentally different types of operational issues, not a single problem.
2. The Red Island — Pure "Speed" Issues
    - Visual Features: Red cluster at the lower-left corner (Delivery Time Delay) is isolated, showing a clear semantic gap from the right-side clusters.
    - Business Interpretation:
        - Represents purely transportation delays.
        - Customers complain specifically about "slow," "late," or "still on the way." No wrong deliveries or lost items—just

carrier inefficiency.
    - Action Recommendation: Focus on logistics KPIs or switch to faster carriers; no warehouse intervention required.
3. The Chaos Continent — Complex “Fulfillment” Issues
    - Visual Features: Right-side cluster is dense, mixing deep blue (Missing), light blue (Wrong), pink (Not Arrived), and green (Delayed Product).
    - Business Interpretation:
      Represents fulfillment failures.
        - Complaints about wrong or missing products are clustered with “not arrived,” showing customers perceive these as the same pain point: not receiving what they wanted.
        - Causes are likely warehouse errors—picking mistakes, inventory mismanagement, or lost items—beyond simple logistics speed.
    - Action Recommendation: Warehouse operations must review picking processes and inventory accuracy; changing carriers will not solve these issues.
4. Executive Summary
    - Delivery complaints are driven by two core causes:
        - Transportation Issue (Red Island): Purely speed-related.
        - Warehouse Issue (Chaos Continent): Wrong, missing, or lost items.

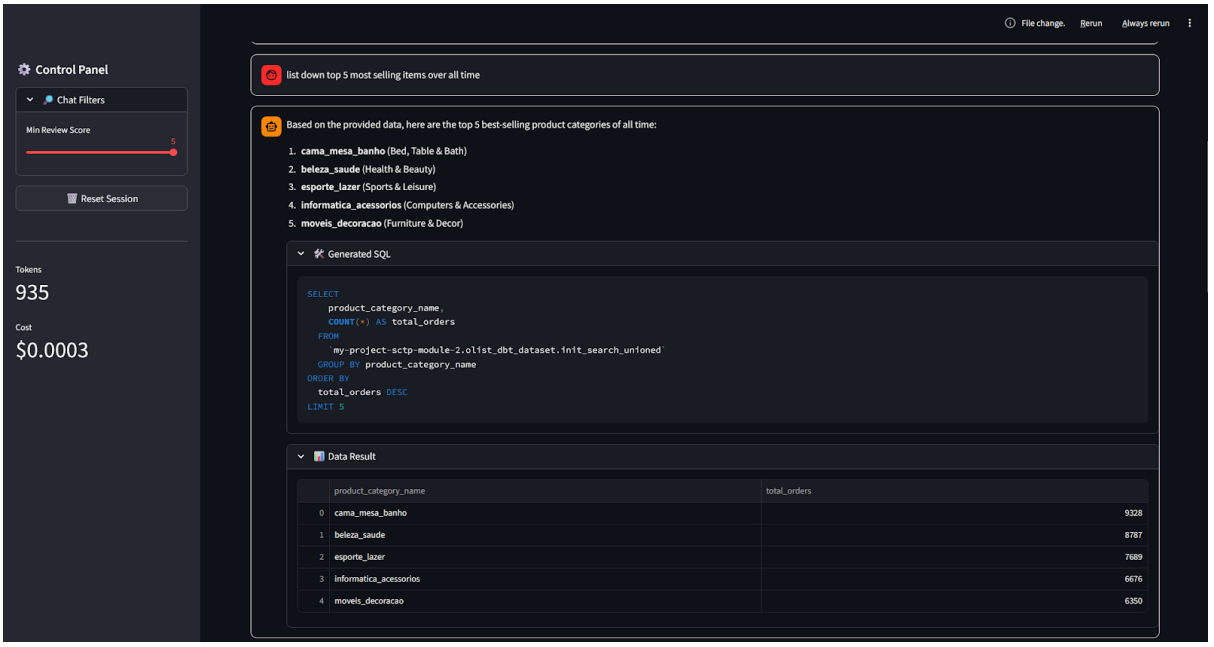
    - RAG Value: Traditional KPIs label all as “delivery failures,” but semantic mapping shows that warehouse issues are more complex and concentrated—these are the high-priority areas for intervention.

Hybrid RAG Features and GUI

An intent classification layer that routes incoming queries into two distinct pipelines:

- Statistical Route: For queries requiring aggregation or math, the system acts as a Text-to-SQL engine, generating and executing precise SQL queries against BigQuery to return deterministic results.

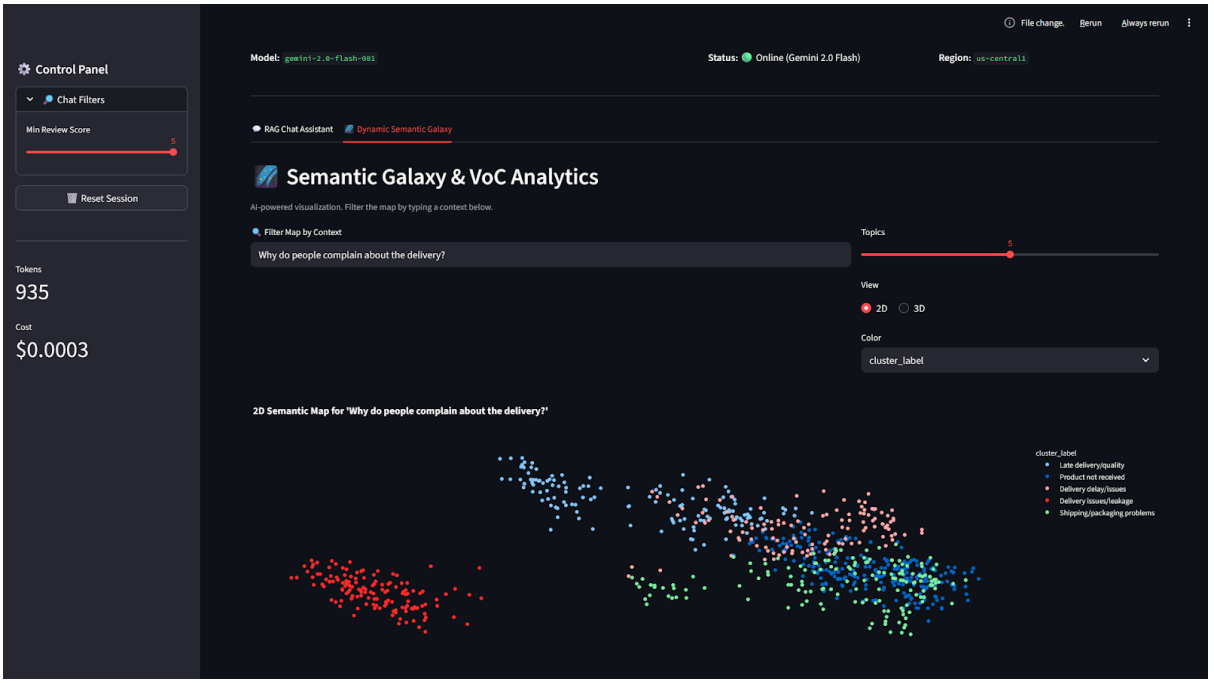
- Semantic Route: For qualitative or open-ended exploration, the system utilizes vector search to retrieve relevant unstructured context.



Advanced Visibility: To make this production-ready, I built a Streamlit interface that includes:

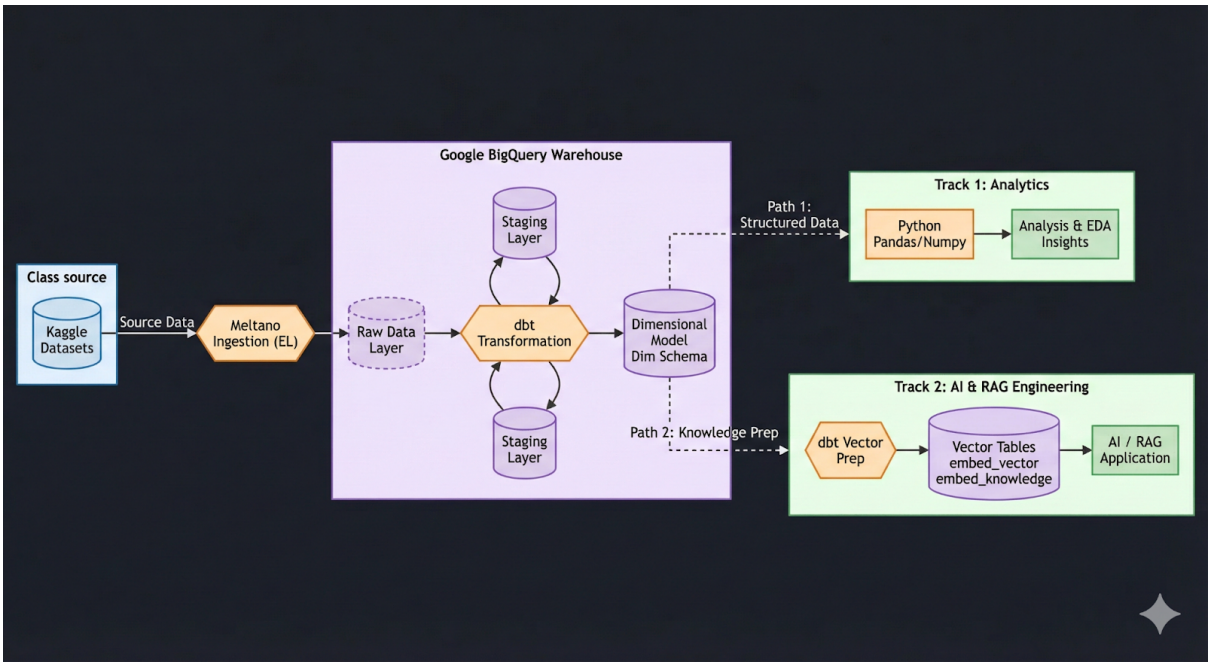
- Cost Transparency: Real-time calculation of Token usage and estimated cost per query.

- Semantic Galaxy: A 3D visualization using PCA and K-Means clustering to map the "Voice of Customer," allowing us to visually identify clusters of complaints like "delivery delays" or "damaged packaging".



3. Technical Design

Architecture Diagram



Database Data Model (ERD)

- Central Fact: fct_order_items (Key metrics: Price, Freight, Review Score).
- Dimensions: dim_customers (User info + Lat/Lng), dim_products (Category info), dim_sellers (Location).
- Auxiliary Fact: fct_payments (Payment type, Installments).

---

4. Implementation Details (Code Structure)

Setup & Environment

- Environment Manager: Conda (example - env: elt).
- Dependencies: dbt-bigquery, meltano.
- Cloud Provider: GCP (Service Account with BigQuery Admin access, Vertex AI access).

Code Structure (Repository)

This repository implements an **end-to-end analytics engineering pipeline** that ingests raw CSV data, transforms it into analytics-ready models, and enables downstream analytics, search, and retrieval-augmented generation (RAG) use cases.

The system follows modern data engineering principles, including:

- Layered data modeling (staging → marts)
- Declarative transformations using dbt
- Reproducible ingestion using Meltano
- Separation of ingestion, transformation, analytics, and application layers

---

2. High-Level Architecture

The project is organized around the full data lifecycle:

Raw CSV Data
→ Meltano Ingestion
→ dbt Transformations
→ Fact & Dimension Models
→ Analytics, Dashboards, and RAG

Each major directory corresponds to a distinct responsibility within this lifecycle.

---

3. Repository Structure

```
sctp_module2_team3_mini_proj/
├── meltano-ingestion/   # Data ingestion and raw storage
├── dbt/                 # Data transformation and modeling
├── dags/                # Workflow orchestration
├── notebooks/           # Analysis and validation
├── rag.py               # Retrieval-Augmented Generation logic
├── dashboard.py         # Visualization entry point
├── demo*.py             # Demonstration scripts
├── logs/                # Execution logs
├── scripts/             # Utility scripts
├── slides/              # Presentation materials
```

---

4. Data Ingestion Layer (`meltano-ingestion/`)

4.1 Purpose

The ingestion layer is responsible for **extracting raw datasets**, loading them into an analytical database, and preparing them for transformation.

4.2 Key Components

```
meltano-ingestion/
├── meltano.yml          # Meltano project configuration
├── extract/             # Data extraction logic
├── load/                # Load targets
├── transform/           # Optional transformation hooks
├── orchestrate/         # Pipeline orchestration definitions
├── plugins/             # Meltano extractor and loader plugins
├── db/
│   ├── create_duckdb.py # DuckDB initialization script
│   ├── mini_proj.db     # DuckDB database file
│   └── *.csv            # Raw Olist datasets
```

4.3 Responsibilities

- Configure data pipelines using Meltano
- Ingest CSV-based Olist datasets
- Initialize and populate a DuckDB analytical database
- Provide a stable raw data source for dbt transformations

---

5. Transformation and Modeling Layer (`dbt/olist_dbt/`)

5.1 Purpose

This layer transforms raw ingested data into **clean, validated, and business-ready models** using dbt.

5.2 dbt Project Configuration

```
dbt_project.yml   # dbt project configuration
profiles.yml      # Database connection profiles
```

These files define:

- Model paths
- Materialization strategies
- Database connections
- Environment-specific settings

5.3 Model Layer Architecture

The dbt models follow a **three-layer architecture**:

```
models/
├── staging/     # Source-aligned, cleaned tables
├── marts/       # Business-facing fact and dimension tables
└── example/     # Sample dbt models
```

5.4 Staging Layer (`models/staging/`)

Purpose

The staging layer performs:

- Column renaming and standardization
- Data type casting
- Minimal transformations
- Schema alignment with source data

Core Staging Models

```
stg_customers.sql
stg_orders.sql
stg_order_items.sql
stg_products.sql
stg_payments.sql
stg_reviews.sql
stg_sellers.sql
stg_geolocation.sql
```

Each staging model corresponds directly to a raw source table.

5.5 Search and Semantic Layer

The project includes a dedicated **search and semantic preparation layer** to support vector search and RAG workflows.

```
staging/
├── search_layer/
│   ├── dim_embedded_knowledge.sql
│   ├── dim_embedded_vectors.sql
│   ├── init_search_unioned.sql
├── search_layer_data/
│   ├── stg_search_customers.sql
```

```
│   ├── stg_search_orders.sql
│   └── ...
```

Responsibilities

- Prepare textual data for embedding
- Generate vector-ready datasets
- Support semantic search and retrieval use cases

5.6 Marts Layer (`models/marts/core/`)

Purpose

The marts layer contains **analytics-ready tables** designed for reporting, dashboards, and downstream applications.

Dimension Tables
dim_customers.sql
dim_products.sql
dim_sellers.sql

Fact Tables
fct_order_items.sql
fct_payments.sql

These models:

- Join multiple staging tables
- Enforce business grain
- Apply business logic and aggregations

5.7 Data Quality and Testing

Schema tests are defined using `schema.yml` files and include:

- `not_null` constraints
- `unique` constraints
- Referential integrity (`relationships`)

Compiled and executed tests are materialized under:

dbt/target/compiled/
dbt/target/run/

---

6. dbt Build Artifacts (`dbt/target/`)

This directory contains **generated artifacts** produced by dbt runs, including:

- Compiled SQL
- Executed SQL
- Test queries
- Lineage metadata (`manifest.json`, `graph.gpickle`)

These files are outputs and are not manually maintained.

---

7. Analytics and Exploration (`notebooks/`)

Purpose

Jupyter notebooks are used for:

- Exploratory data analysis
- Data validation
- Metric verification
- Ad-hoc investigations

notebooks/
├── analyze_csv.ipynb
├── Farida/
├── Laslie/
└── Taufiq/

Contributor-specific folders reflect collaborative development.

---

8. Retrieval-Augmented Generation (`rag.py`)

Purpose

This module implements **Retrieval-Augmented Generation** logic using:

- Embedded knowledge tables
- Vector representations generated in dbt

Typical responsibilities include:

- Query embedding
- Semantic retrieval
- Context construction for LLM-based responses

---

9. Environment and Configuration

```
requirements.txt   # Python dependencies
proj_env.yml       # Environment configuration
```

These files ensure consistent setup across development environments.

---

10. Supporting Files

```
logs/     # Runtime and dbt logs
scripts/  # Utility scripts
slides/   # Presentation materials
```

---

11. Summary

This repository is structured to support **scalable analytics engineering**, with clear separation between ingestion, transformation, analytics, and AI-enabled search. The design promotes maintainability, data quality, and extensibility while supporting both traditional BI and advanced semantic use cases.
