QuantifAI Data Engineering challange

- Project Overview :

  This project is a complete data engineering solution, addressing real-world messy data integration from three recently acquired e-commerce platforms. The goal is to :

  Phase 1 : Data Exploration and Analysis

  Phase 2 : ETL pipeline development

  Phase3 : Dashboard Creation

  Phase 4 : Using Gemini AI to reconcile mismatched schema data

- **Phase1 : Data Discovery And Analysis :**

1. **Loaded All Raw Datasets**: Imported customer, order, and product data from multiple inconsistent formats (JSON, CSV).
2. **Explored Data Structure**: Used head(), info(), and describe() to understand schema, types, and basic stats.
3. **Identified Data Quality Issues**: Noted nulls, duplicates, inconsistent naming, mixed formats, and invalid values.
4. **Mapped Redundant Fields**: Found overlapping fields like customer_name/full_name, qty/quantity, and item_id/product_id and performed Exploratory data analysis (EDA).
5. **Planned Cleaning Strategy**: Defined steps for merging, standardizing, handling nulls, and enforcing valid references for ETL.

  - **Data Quality Issues that I found after analyzing datasets :**

Customers

1. Duplicate fields: customer_name, full_name, email, email_address
2. Mixed gender values: M, F, Female, Other, "
3. Mixed date formats: registration_date, reg_date
4. Inconsistent status: ACTIVE, active, Inactive, etc.
5. Nulls in: phone, age, birth_date, preferred_payment, etc.

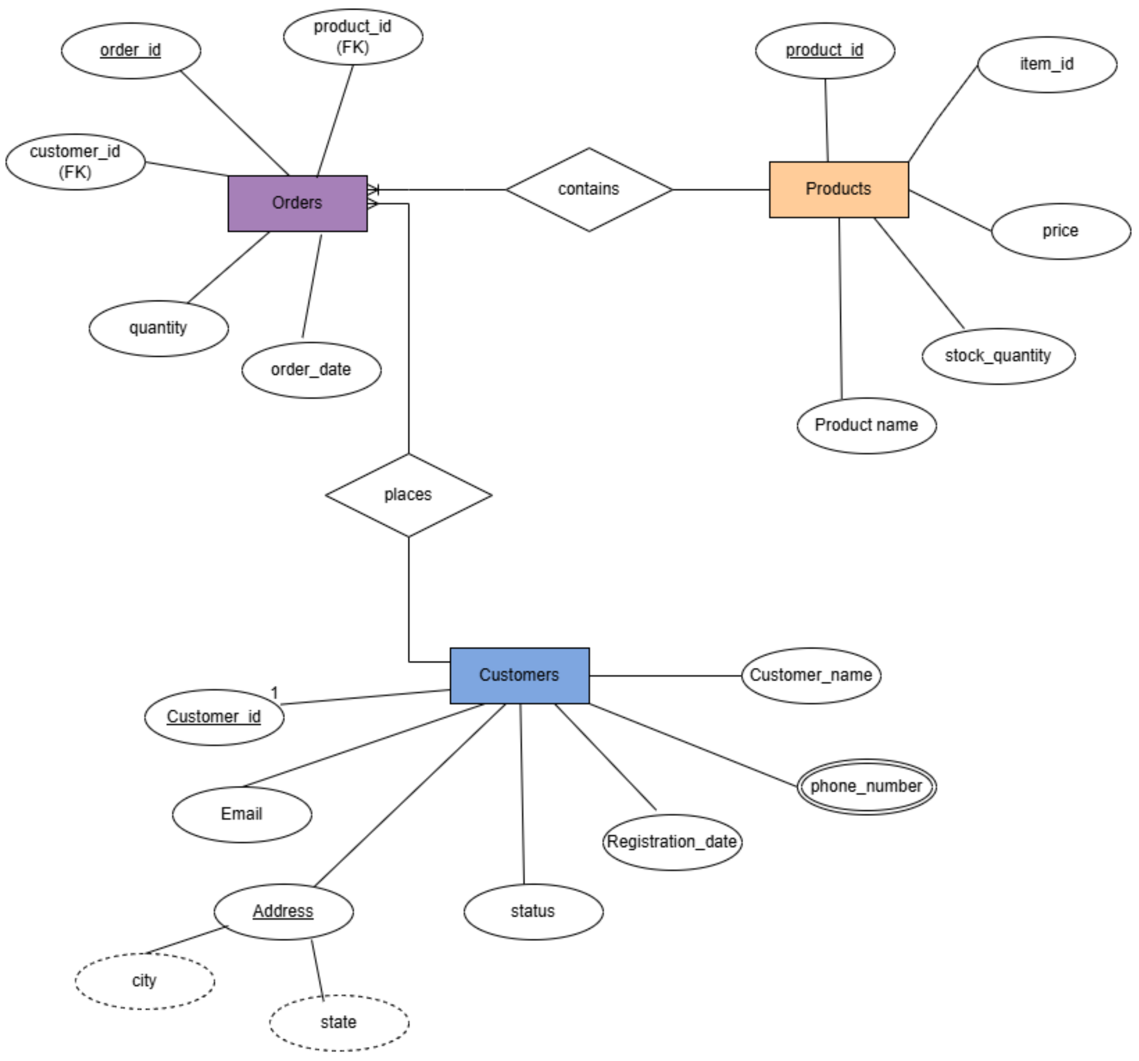
Orders

1. Duplicate fields: order_id vs ord_id, qty vs quantity
2. Mixed formats in order_date
3. Inconsistent status, missing order_datetime
4. Nulls in tracking_number, notes

Products

1. Duplicate fields: product_id vs item_id, product_name vs item_name
2. Inconsistent boolean in is_active field: yes, no, 0, False, etc.
3. Mixed casing in brand/category
4. Nulls in: description, supplier_id, created_date

ER –diagram:



- **Data Cleaning Strategy Finalization :**

A structured cleaning process was applied to ensure consistent, reliable, and joinable data across all datasets.

Customers

- Merged duplicate fields (customer_name, full_name; email, email_address)
- Standardized registration_date, status, and gender
- Retained customer_id as primary key; removed cust_id
- Filled missing preferred_payment and zip_code; dropped incomplete records
- Removed duplicates using ID, name, and email

Orders

- Consolidated fields (ord_id, qty, cust_id) into standardized columns
- Unified order_timestamp from multiple date fields
- Normalized status values (shipped, SHIPPED, Returned, etc.)
- Verified total_amount = quantity × price
- Replaced missing tracking_number with default placeholder

Products

- Kept product_id, product_name; removed duplicates like item_name
- Standardized brand, category, is_active values
- Cleaned numeric data; handled outliers and filled missing weight, rating
- Parsed dates; removed invalid or future records

Validation

- Ensured referential integrity between customers, orders, and products
- Enforced data type consistency and clean joins for ETL

Phase 2: ETL Pipeline Development

This phase focused on **building a robust ETL pipeline** that ingests raw data, performs modular cleaning, and loads the cleaned data into a normalized SQLite database for analysis.

Implemented :

1. **Modular Cleaning Functions**
   - Wrote reusable clean_customers(), clean_orders(), and clean_products() functions.
   - Used combine_first() to merge redundant fields (e.g., cust_id with customer_id).
   - Standardized formats (e.g., dates, status fields), cleaned nulls with mode/median, and handled data type conversions.
2. **Data Validation**
   - Checked for essential fields (email, product_id, etc.) and dropped invalid rows.
   - Imputed missing values using business logic (preferred_payment, zip_code, etc.).
3. **Foreign Key Filtering**
   - Ensured referential integrity by filtering orders that reference valid customers and products only.
4. **Database Creation with SQLite**
   - Used pandas.to_sql() to create normalized tables: customers, products, and orders.
   - Added indexes to optimize query performance on key fields.

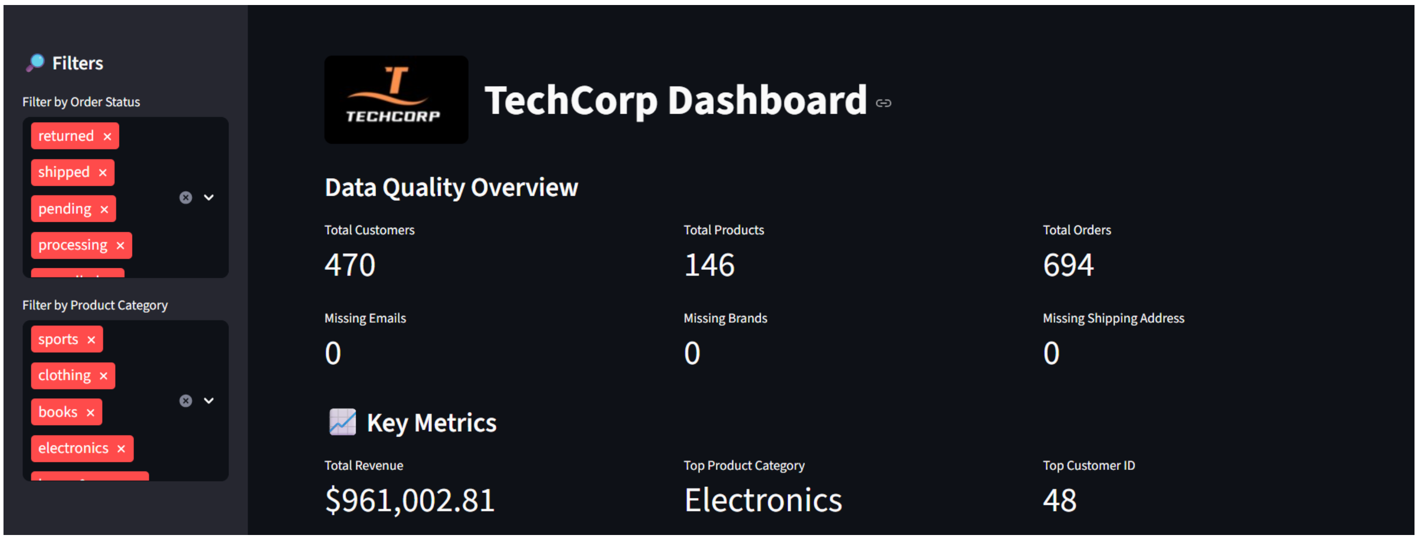
5. **File Outputs**
   - Saved cleaned .csv files to dataset/cleaned/
   - Populated **TechCorp_cleaned.db** database for dashboard and future querying.

Phase 3 : Building Streamlit Dashboard

Created an user-friendly analytics dashboard using **Streamlit** that will connect to the cleaned SQLite database and offers actionable insights.

Implemented :

1. **Clean UI Layout**
   - Sidebar with interactive filters (order status, product category).
   - Main area with KPIs, charts, and data tables.
   - Brand logo and title shown side-by-side using st.columns().
2. **Business KPIs & Metrics**
   - Total Revenue (from order_value).
   - Top Product Category (based on frequency).
   - Top Customer ID (based on order total).
3. **Visual Insights**
   - **Sales Trend**: Line chart showing monthly revenue using plotly.express.
   - **Customer Segments**: Pie chart showing segmentation distribution.
   - **Top Products Table**: List of highest-selling products.
4. **Data Quality Snapshot**
   - Metrics for missing values across customers, orders, and products.
   - Helps business users monitor data health at a glance.
5. **Search & Explore**
   - Filtered product table with live search capability using category filters.
   - Instant insights with dropdown/multiselect for granular exploration.



Filters

Filter by Order Status: returned × shipped × pending × processing ×

Filter by Product Category: sports × clothing × books × electronics ×

TechCorp Dashboard

Data Quality Overview

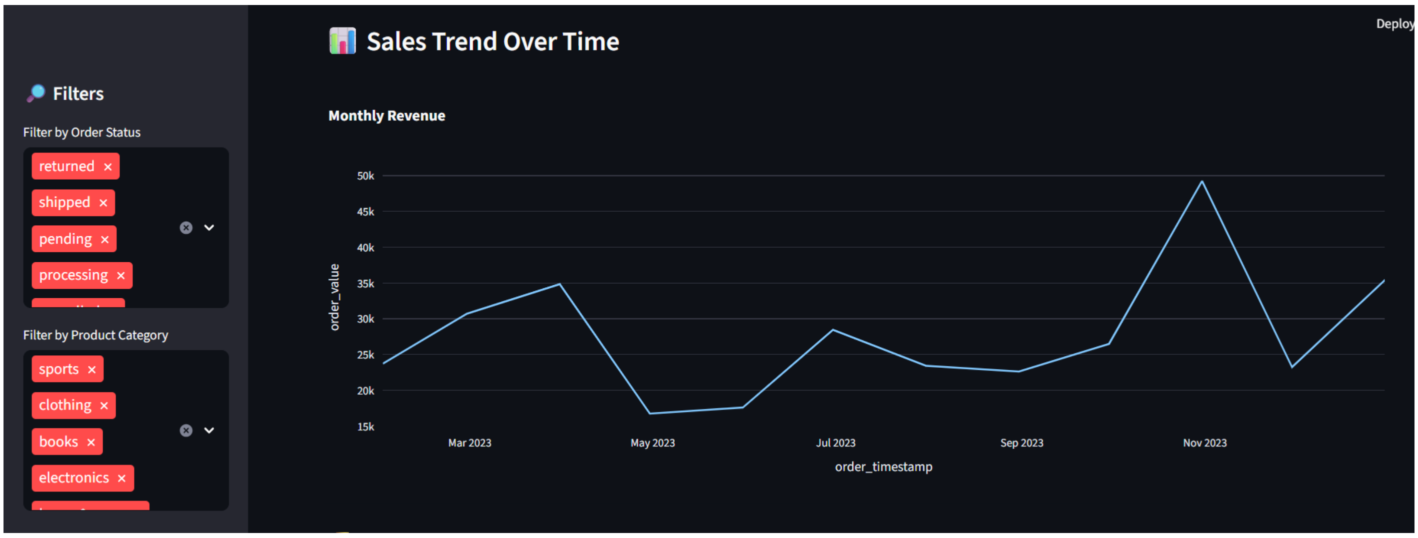
Total Customers: 470
Total Products: 146
Total Orders: 694

Missing Emails: 0
Missing Brands: 0
Missing Shipping Address: 0

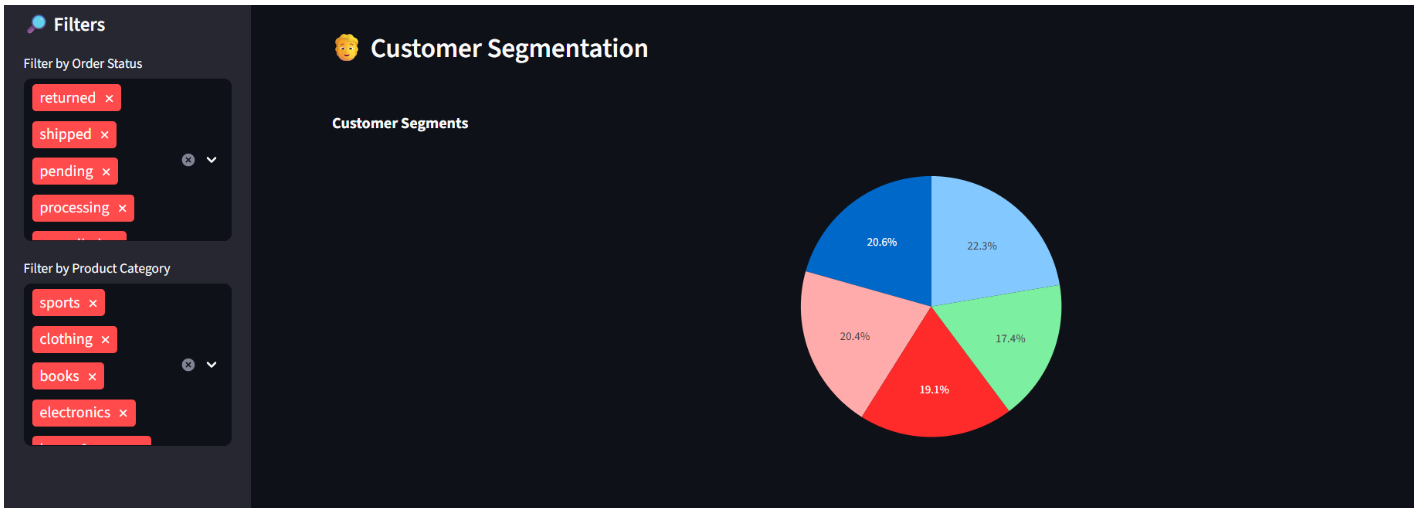
Key Metrics

Total Revenue: $961,002.81
Top Product Category: Electronics
Top Customer ID: 48

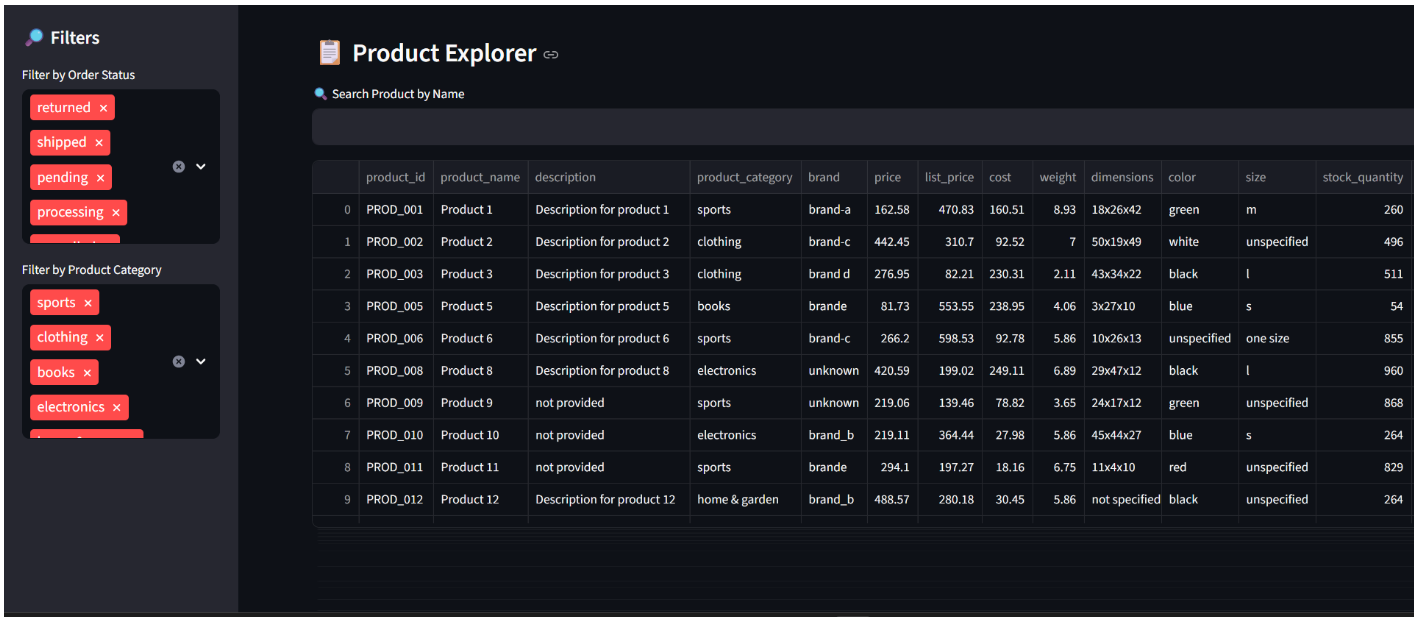
It Displays key business metrics like total revenue, product counts, and customer insights, alongside real-time data quality stats such as missing values.



Visualizes monthly revenue fluctuations based on filtered product categories and order statuses. This helps identify peak sales periods and seasonality.



Illustrates the distribution of customers across different market segments, supporting strategic targeting and personalization.



	product_id	product_name	description	product_category	brand	price	list_price	cost	weight	dimensions	color	size	stock_quantity
0	PROD_001	Product 1	Description for product 1	sports	brand-a	162.58	470.83	160.51	8.93	18x26x42	green	m	260
1	PROD_002	Product 2	Description for product 2	clothing	brand-c	442.45	310.7	92.52	7	50x19x49	white	unspecified	496
2	PROD_003	Product 3	Description for product 3	clothing	brand d	276.95	82.21	230.31	2.11	43x34x22	black	l	511
3	PROD_005	Product 5	Description for product 5	books	brande	81.73	553.55	238.95	4.06	3x27x10	blue	s	54
4	PROD_006	Product 6	Description for product 6	sports	brand-c	266.2	598.53	92.78	5.86	10x26x13	unspecified	one size	855
5	PROD_008	Product 8	Description for product 8	electronics	unknown	420.59	199.02	249.11	6.89	29x47x12	black	l	960
6	PROD_009	Product 9	not provided	sports	unknown	219.06	139.46	78.82	3.65	24x17x12	green	unspecified	868
7	PROD_010	Product 10	not provided	electronics	brand_b	219.11	364.44	27.98	5.86	45x44x27	blue	s	264
8	PROD_011	Product 11	not provided	sports	brande	294.1	197.27	18.16	6.75	11x4x10	red	unspecified	829
9	PROD_012	Product 12	Description for product 12	home & garden	brand_b	488.57	280.18	30.45	5.86	not specified	black	unspecified	264

Phase 4 : AI Reconcilliation phase :

What Was Done

- **Manual Schema Analysis**: I examined both the raw reconciliation dataset and the cleaned dataset to identify mismatched or semantically similar columns.
- **AI-Aided Mapping Strategy**: Using Gemini AI (via conceptual prompts), I constructed a mapping dictionary that aligns fields like:
    - client_reference → customer_id
    - full_customer_name → full_name
    - transaction_date → order_timestamp
    - contact_email → email
    - And others based on semantic meaning.
- **Python Script Implementation**:
  I implemented this mapping in gemini_schema_reconciliation.py, which:
    - Loads the raw reconciliation dataset
    - Renames columns using the AI-derived mapping
    - Saves the cleaned version as reconciliation_cleaned.csv
    - Optionally compares customer_id values with the main customer dataset to detect new/unmatched entries
- **Validation Step**:
  I validated the reconciliation output by comparing unique customer IDs with those in the cleaned data. This revealed **300 new customer entries**, confirming the mapping was effective and that the dataset contains previously unseen records.

Although I did not directly integrate the Gemini API in this project, I designed the reconciliation logic in a way that could easily be extended to support it. Here's the approach I followed and how it could work with Gemini API in future iterations

My Approach :

- **Identifying Schema Mismatches**
  I started by reviewing both the reconciliation dataset and the cleaned dataset to identify inconsistencies in column names. Common examples included client_reference vs customer_id and contact_email vs email.
- **Planning for AI-Assisted Mapping**
  Instead of manually mapping each field, I explored the idea of using Gemini AI to semantically understand and suggest appropriate field mappings based on their meanings.
- **Preparing the Input for Gemini**
  The plan involved passing two lists to Gemini — one containing the raw column names and the other the standardized column names — along with a clear prompt asking the model to generate a Python dictionary that maps one to the other.
- **Designing a Meaningful Prompt**
  The prompt would describe the task in simple language, instructing Gemini to match the raw column names to their closest equivalent in the cleaned schema, purely based on semantic understanding.
- **Receiving and Interpreting the Output**
  Gemini would return a dictionary-like structure mapping raw fields to standardized fields. This output would then be parsed and reviewed before use.
- **Applying the Mapping to Data**
  The generated mapping would be used to rename the columns in the reconciliation dataset, ensuring they align with the existing data structure used in the ETL pipeline.

- **Validating Reconciled Data**
  After renaming, the dataset would be validated by checking whether the customer_id and other key identifiers exist in the cleaned customer table. This helps confirm the effectiveness of the mapping.
- **Not Integrating the API (Current Scope)**
  Although the API integration was not implemented in this version due to project scope and key access, the entire logic and structure is designed to support Gemini API integration in the future.