**QuantifAI Data Engineering Challenge - Documentation**

**Project Overview**

This project implements a complete data engineering solution for TechCorp's unified data pipeline challenge.

**Phase 1: Data Discovery & Quality Assessment**

**Manual Data Exploration**

I started by manually examining each dataset to understand the structure and identify data quality issues. This hands-on approach helped me get familiar with the business context.

**Key Issues Identified Manually:**

1. Customers\_messy\_data.json :

* There are fields with same meaning but different names customer\_id, cust\_id ; email, email\_address; phone, phone\_number; registration\_date,. reg\_date; status, customer\_status
* For some customers phone number is written in phone or phone\_number or some has it in both.Some have different postal\_code and zip\_code.
* Some where ‘customer\_status’ is Active while for the same ‘status’ is InActive.
* There are multiple entries with inconsistency of capital-small words e.g. Active, active.

1. products\_inconsistent\_data.json :

* The dataset have both product\_id and item\_id, as well as product\_name and item\_name, which refer to the same entry and names.
* The fields category and product\_category contain conflicting or unrelated values (e.g., category: *electronics*, product\_category: *Books*).
* The is\_active field uses different representations for boolean values: 1, "yes", "no", false.

1. Orders\_unstructured\_data.csv :

* quantity and qty appear to be duplicate fields holding the different values.
* unit\_price vs. price, status = CANCELLED vs order\_status = processing. E.g.: unit\_price = 130.62, price = 404.94 with qty = 4. 130.62 × 4 = 522.48, not 404.94

**Automated summariser Utility**

I developed a custom summarise utility (utils/summarise.py) that generates comprehensive summaries of each dataset.

**Summariser Logic:**

* It goes through each column in the dataset and gives a small summary of what’s in it.
* First, it ignores any missing values (like empty cells) so it only looks at real data.
* It checks what kind of data is in the column (like text, numbers, etc.) and tells if there’s more than one type.
* It grabs a few example values from each column (first 5 unique ones) to see what kind of data is present.
* It counts the no. of columns which has null value.
* It counts how many different values are in the column.
* If the data in the column is all text, it checks for the case sensitivity.
* It checks If a column has a mix data types (like numbers and text in the same column).
* It puts together all of this info into a list of summaries, one summary for each column.
* At the end, we get a bunch of helpful details about data so we can spot what’s clean, what’s messy, and what needs fixing.

**Jupyter Notebook: LLM-Assisted Data Quality Analysis**

**Notebook:** notebooks/llm\_validation.ipynb

I created a Jupyter notebook that gives AI-powered analysis to systematically identify data quality issues across all three datasets.

**Notebook Structure:**

* Imported essential libraries: pandas, FPDF, LangChain, Google Gemini AI. Set up environment variables for API access
* Loaded all three raw datasets (customers, products, orders)
* Applied custom summarize\_dataframe() utility to generate column-wise summaries
* Created structured data profiles for each dataset
* Implemented Pydantic output parser (IssueOutput) for structured LLM responses
* Configured LangChain with Google Gemini 2.0 Flash model
* Created specialized prompt template for data quality issue detection
* def generate\_data\_issue\_report(dataset\_name, summary\_text, sample\_records\_text):
  + *# LLM chain for systematic issue identification*
  + *# Analyzes column summaries + random samples*
  + *# Returns structured list of data quality issues*
* Created utility to extract 10 random records from each dataset
* Handles both JSON and CSV formats appropriately

**LLM-Assisted Analysis in Notebook**

After getting the summaries, I used LLM to analyse 10 random samples from each dataset along with my manual observations + summaries.

**Analysis Process:**

1. Fed summary to LLM
2. Provided 10 random samples from dataset
3. Combined LLM analysis with my manual observations
4. Generated comprehensive PDF reports for dataset

**Major Data Quality Issues Found by LLM’s are stored in pdf format in the reports directory.**

* *Customers Dataset Issues:* customers\_data\_quality\_report.pdf
* *Orders Dataset Issues:* orders\_data\_quality\_report.pdf
* *Products Dataset Issues:* products\_data\_quality\_report.pdf

**Phase 2: Data Cleaning & Transformation**

**Cleaning Strategy**

Based on the analysis, I developed dataset-specific cleaning strategies implemented in separate Python scripts by assistance with AI.

**Customers Cleaning (cleaners/clean\_customers.py):**

**Step 1:** Removes extra columns like cust\_id, email\_address, phone\_number, etc., and merges them into a single version like customer\_name, email, phone, etc.

**Step 2:** phone numbers => (555) 123-4567.Format dates also.City and state names are mapped to proper names(like "la" =>"Los Angeles").Use same case for each entry in column.

**Step 3:** total\_orders, total\_spent, loyalty\_points, age => datatype numbers. Keeps zip\_code =>text

**Step 5:** Checks email addresses are in the right format. Verifies age and birthdate if don’t match correct it using b’date. Remove duplicate customer IDs.

**Step 6:** Shows how many rows/columns were changed and gives an overview of missing values and data types after cleaning. And stores it as customers\_cleaned\_data.json.

**Orders Cleaning (cleaners/clean\_orders.py):**

**Step 1:** Remove extra versions of columns like cust\_id, ord\_id. keep just customer\_id and order\_id to avoid confusion.

**Step 2:** Makes sure both order\_date and order\_datetime follow proper datetime format

**Step 3:** If one of the date fields is missing, it fills it from the other one. If orders are marked as shipped or delivered but don’t have a tracking number, generate random tracking number.

**Step 4:** Normalizes statuses like delivered, Delivered, etc. => Capital case (e.g. DELIVERED). Merge both status and order\_status into just one status.

**Step 5:** Compares quantity vs qty. Merge into one and store the larger one.

**Step 6:** order\_total => price × quantity + tax + shipping − discount. Remove the old price column and updates it. total\_amount => price × quantity.

**Step 8:** Makes sure there are no negative numbers in important columns like quantity, price, tax, etc. remove discount if they’re more than the total price.

**Step 9:** Prints a small summary of missing values and unique values per column And store the cleaned df as orders\_cleaned\_data.csv.

**Products Cleaning (cleaners/clean\_products.py):**

**1.** Empty strings in fields like brand, color, and supplier IDs => NaNs

**2.** Fields like price, cost, and weight => numeric values.

**3.** category and product\_category values (CLOTHING, clothing, etc.) => Cap-start (e.g. Clothing). Adds a new column final\_category that combines the two if one is missing.

**4.** weird characters (\_, -)=> “ ”, and turns names into proper case (e.g. sony-corp => Sony Corp).

**5.** "yes", 1, "TRUE", "inactive" => True or False. Adds a flag is\_active\_flag\_issue when a product is marked active but missing price or stock\_quantity.

**6.** "m", "XS", "onesize" => "M", "XS", "One Size", etc.

**8.** Handles different date formats (e.g., ISO timestamps) and turns them into proper datetime objects. Applies this to created\_date and last\_updated.

**9.** Validates dimension strings like "10x5x2", if not match remove it.

**10.** category\_mismatch flag if both fields exist but don’t match.

**11.** Prices > 0 (< 0 get turned into NaN). Stock quantities > 0(< 0 set to 0). Ratings 0 to 5

**12.** keepboth item\_id and product\_id for now .

**13.** Prints out the remaining missing value % by column. Returns the cleaned dataset and saves it as products\_cleaned\_data.json file.

**Challenges:**

the whole thing was pretty challenging for me, mainly because I wasn’t that into Python before this. Most of my background wasn’t in Python, so jumping into a full project where everything was in Python — from data cleaning to writing scripts and creating database tables was definitely not easy at first.

**Orders Dataset Normalization Challenge:** The orders dataset was particularly challenging and required significant time investment. nothing was clearly defined for how the **orders dataset** should be structured when I started. It was just one csv with everything mixed in like order info, quantity, item\_id, product\_id, and customer\_id, etc. I noticed there's an item\_id, product\_id column in the orders dataset. Alos it has total\_amount and order\_total then I thought that this may refer to two different things combined together. One may be order\_items that refers to item-level details for each order, like quantity, price, and which product was ordered. And Order details refers to general info about the order — like who placed it, the status, tracking, payment, shipping, tax, etc.

I considered and recalculated all these values while cleaning it self

* total\_amount = quantity \* unit\_price
* order\_total = total\_amount + shipping\_cost + tax – discount.

Many more challeges were finding out the proper relation between the tables, along with correct keys for them. But on the plus side, at the same time I was also exploring Gen AI and that actually helped a lot. I used it to understand concepts faster, debug code, and even get some advice for cleaning the data or structuring tables.

**Phase 3: Database Design & Implementation**

**Entity-Relationship Analysis**

After cleaning the data, I manually analyzed the relationships between entities to design a proper normalized database schema.

**Thought process:**No relationships were given to me initially, I need to understand and figure out manually from the structure of the raw data. I looked at the columns in each dataset and tried to think in terms of how they might be connected in a real e-commerce system. As it was related to e-commerece platform my previous experience with creating an e-commerce website helped me visualise the relationships.

in the orders dataset, I noticed there was a customer\_id, which clearly links each order to a customer, so that became a foreign key from the customers table.

Similarly, product\_id as foreign key from the products table.

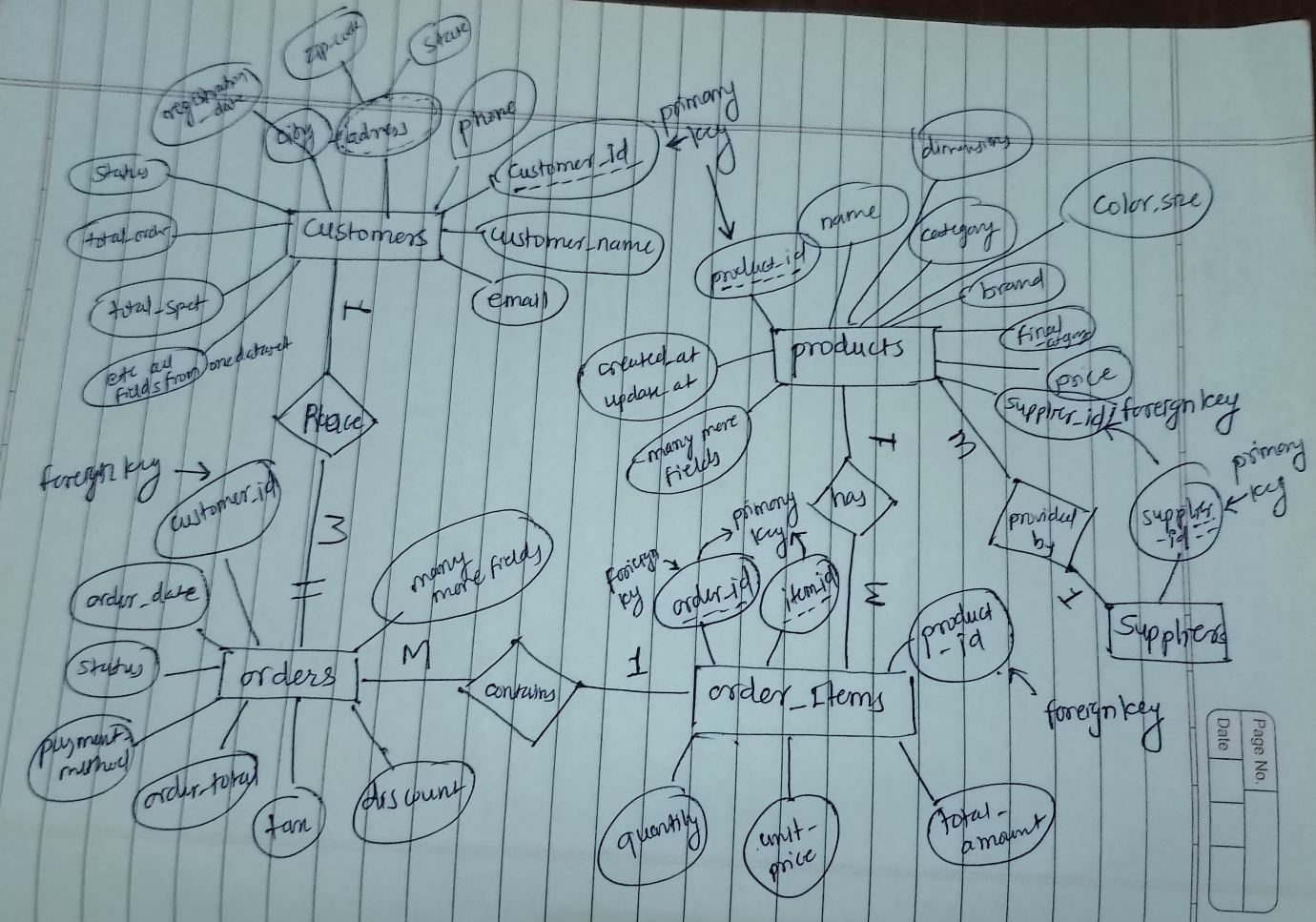
In my ecomerce website I had my order items separately in different collections and the order details in one different collection. Which made me think to separate the orders dataset to two tables order consisting the order related things and order\_items consisting the items quantity, pricing related things.

I also noticed that supplier\_id existed in the products data, so I created a minimal suppliers table to link it through a foreign key

**Final Database Schema:** Based on my analysis, I designed the following 5 tables.

* **Customers Table:** Generated from customers\_cleaned\_data.json
* **Orders Table:** Generated from orders\_cleaned\_data.csv
* **Order\_Items Table:** Generated from orders\_cleaned\_data.csv
* **Products Table:** Generated from products\_cleaned\_data.json
* **Suppliers table:** Generated from suppliers\_id field in products\_cleaned\_data.json

**ER Diagram:**



**Database Implementation**

* **Schema Creation:** db/table.py - Creates all tables with proper constraints
* **Data Insertion:** db/insert.py - Loads cleaned data with validation

**Code Organization**

* **Modular Design:** Separate cleaning scripts for each dataset, separate folder for cleaned datasets, separate folder for database operations.
* **Utility Function:** Reusable function in utils/ directory
* **Clear Separation:** Raw data, cleaned data, and database files properly organized.

**Phase 4: Interactive Dashboard Development**

**Dashboard Features**

I created a Streamlit dashboard (app/dashboard.py) using AI to generate meaningful business insights + for pretty user-friendly UI.

**Dashboard Components:**

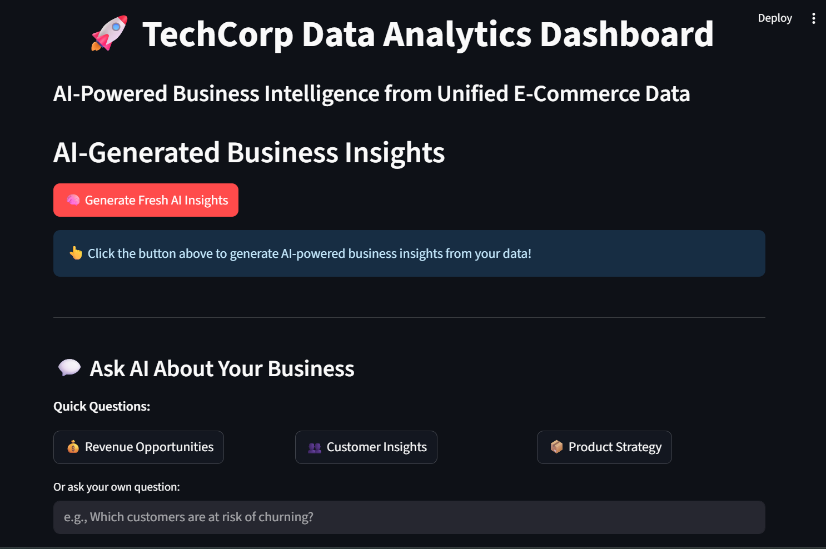
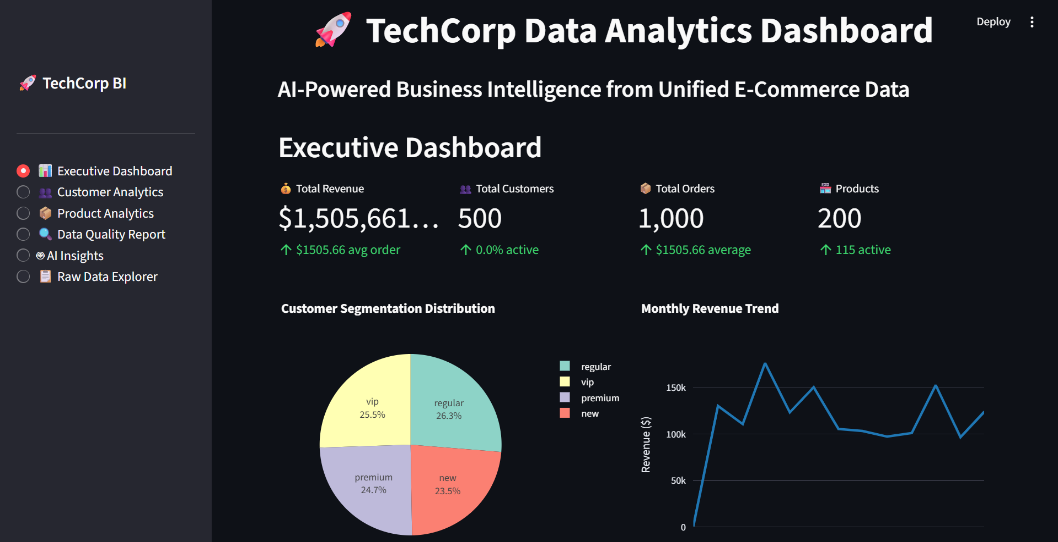
* **Starting Dashboard:** KPIs include Total Revenue, Orders, Customers, and Product Counts. Also includes visualizations like revenue trends and customer segmentation pie charts.
* **Customer Analytics** : Insights into customer behavior including status distribution, segment-wise spending, and geographic spread.
* **Product Analytics**: shows top-performing products, price and stock distribution, and category performance using revenue and quantity metrics.
* **Data Quality Report**: Highlights missing or incomplete data fields and known data issues (like category mismatches or active flag inconsistencies), using color-coded metrics.
* **AI Insights**:
  + AI-generated business insights powered by Google Gemini via LangChain.
  + Interactive business question answering feature using pre-loaded metrics and structured prompts.
* **Raw Data Explorer**: Lets users explore all tables (customers, products, orders, order items, and suppliers) directly inside the app.

**AI-Generated Insights:** The dashboard uses **LangChain with Gemini-2.0 Flash** to deliver actionable business insights from structured data. The AI component does the following:

* Analyzes recent trends in revenue, customer behavior, and product performance.
* Identifies risk areas like inactive customers or low-stock items.
* Recommends data-driven actions such as which customer segment to focus on, or which products are underperforming.

**KPIs:**

* **Total Revenue**: The cumulative sales amount from all orders.
* **Average Order Value (AOV)**: The average revenue per order.
* **Total Orders**: Total number of orders placed.
* **Customer Retention Rate**: Percentage of customers that remain active over time.
* **Active Products**: Number of products currently available for sale.
* **Top Performing Products / Categories**: Based on total sales volume and revenue.

**Results & Outcomes**

