In real life, this system would be used by:

* **Fuel Station Managers** to monitor daily sales, refills, and inventory.
* **Operations Teams** to track tank levels, detect shortages, and schedule refills.
* **Business Analysts and Executives** to analyze sales trends and performance.
* **Logistics/Refinery Teams** to plan shipments based on real-time demand.

## **Bronze Layer?**

The **Bronze layer** is where we dump the raw data exactly as it arrives — no cleaning, no filtering. This allows us to:

* **Preserve the original data**
* Capture any data **errors or weird patterns**
* Trace back if needed

| **Field Name** | **What It Means** | **Why It’s Useful** |
| --- | --- | --- |
| StationID | Fuel station ID | Identify where the sale happened |
| PumpID | Pump number that sold the fuel | Track pump-wise activity |
| TankID | Underground tank where fuel is stored | Needed to check tank inventory |
| TransactionDate | Date when the sale happened | For daily reporting and time trends |
| FuelType | Petrol or Diesel | For product-level tracking |
| VolumeSold\_Liters | How much fuel was sold | To track consumption and revenue |
| TankLevel\_PostTxn | How much fuel remained after the sale | Needed to trigger refill alerts later |
| AmountCollected | Total money collected from the sale | For sales/revenue analysis |
| ReorderLevel | Minimum level before refill is needed | Used later to compare with TankLevel\_PostTxn |
| CreatedBy | Who uploaded the data | For audit trail |
| CreatedTime | When the data was uploaded | Track freshness of data |
| ModifiedBy | Who last modified the data | Also for audit trail |
| ModifiedTime | When the data was modified | To detect updates or changes |

### **2. Why use different ReorderLevel values?**

👉 In real life, **each tank has a different size**.  
 You can’t set one fixed reorder level for all tanks. For example:

* A tank holding 5000L should have a reorder level like **1000L**.
* A smaller tank (2000L) might reorder at **400L**.

So, **ReorderLevel should depend on the tank’s capacity**.

✅ Better approach:

* Store tank capacities in a reference table (like a lookup file).
* Then, in Silver layer, compare TankLevel\_PostTxn to that tank’s specific ReorderLevel.

### **❓3. How does IoT help in refilling fuel tanks?**

👉 IoT = Internet of Things = Smart sensors installed **inside the fuel tanks**.

These sensors:

* Continuously **measure how much fuel is left** in the tank.
* Send this live data to Azure Blob or IoT Hub in real-time.

💡 So, when the fuel level gets **close to the reorder level**, the system:

* **Automatically triggers an alert**
* Notifies logistics teams to **schedule a refill** in advance
* Prevents tanks from running dry (no stockouts)

✅ **This data becomes part of your pipeline** and is checked in the Silver/Gold layers to send alerts and track consumption patterns.

✅ Here's the **Tank Reference Table** you can download:  
 📄 tank\_reference\_table.csv

### **🛢️ What Is This Table For?**

This file contains important reference data for each tank:

| **TankID** | **TankCapacity\_Liters** | **ReorderLevel\_Liters** |
| --- | --- | --- |
| TK1 | 5000 | 1000 |
| TK2 | 4000 | 800 |
| TK3 | 3000 | 600 |

### **🧠 Why Do You Need It?**

* In the **Silver layer**, you'll use this file to **join** with your sales data.
* This allows you to:  
  + **Check tank-specific reorder levels**
  + **Detect low inventory** (when tank level after transaction is below reorder level)
  + Keep logic dynamic — if tank sizes change, you just update this file.

### **🔄 How to Use This in the Pipeline?**

#### **Step-by-Step:**

1. **In the Silver Layer (Databricks/ADF):**
   * Load your raw transaction data (from Bronze)
   * Load this tank reference table
   * Perform a **join** on TankID

**Add new logic:** python  
Copy code  
if TankLevel\_PostTxn < ReorderLevel\_Liters:

RecordStatus = "LowInventory"

1. Also check for:  
   * Negative VolumeSold\_Liters → "NegativeSales"
   * Duplicates → Use drop\_duplicates() and tag as "Duplicate"

| **File** | **Purpose** |
| --- | --- |
| bronze\_layer\_sample\_dataset.csv | Raw transaction logs, no cleaning yet |
| tank\_reference\_table.csv | Lookup for tank capacity and reorder level |

### **What Does the tank\_reference\_table.csv Do?**

The **tank\_reference\_table.csv** is a **lookup file** that holds details for each tank, including:

| **TankID** | **TankCapacity\_Liters** | **ReorderLevel\_Liters** |
| --- | --- | --- |
| TK1 | 5000 | 1000 |
| TK2 | 4000 | 800 |
| TK3 | 3000 | 600 |

### **🧠 Why Is This Useful?**

In the real world:

* **Every tank is different** — size, capacity, refill threshold.
* You can’t hardcode reorder levels in your sales dataset.

So instead:  
 ✅ You **store tank info separately** and  
 ✅ You **join this file** with your Bronze data during processing (in Silver layer).

### **🔄 What Happens After the Join?**

Let’s say your data shows:

* Tank TK2 has 700L left (TankLevel\_PostTxn)
* From reference: TK2's reorder level = 800L

Then:  
 ➡️ You mark this record in Silver as **"LowInventory"** ➡️ In Gold, you trigger a **refill alert**

🔌 IoT Devices / POS Systems (Fuel Stations)

│

▼

🗂️ Raw CSV Data (e.g., station\_logs\_2025-07-15.csv)

│

▼

☁️ Upload to Azure Blob Storage → `/raw/` folder

│

▼

🔄 Azure Data Factory copies it → Azure Data Lake → `/raw/` zone (Bronze Layer)

│

▼

📦 Store as-is (No cleaning, No checking)

### **Plan to Add Realistic Abnormalities to Bronze Dataset**

I was modifying your original **Bronze dataset** to include the following realistic problems:

| **Row** | **Issue** | **Notes** |
| --- | --- | --- |
| 1 | Zero Volume Sold | Volume is 0 — possibly a test or entry mistake |
| 2 | Missing Fuel Type | FuelType is blank — can't process or analyze |
| 3 | Future-Dated Transaction | Dated 2035 — clearly not a valid past sale |
| 4 | Wrong Fuel Type | "Gasolene" is not accepted — could break pipelines |
| 5 | Tank Level Increased After Sale | Level should drop, not rise — indicates manipulation or wrong input |
| 6 | Negative Amount Collected | Financial error or reversal not handled correctly |
| 7 | Extremely High Price Per Liter | ₹50,000 for 100 liters = ₹500/litre — unrealistic sale |

Let me know if you'd like help with:

* **Flagging these in Silver layer**
* **Creating rules to clean/validate them**
* **Automatically sending alerts for each case**

## **What is the Silver Layer?**

The **Silver Layer** is where we:

✅ Clean the raw data  
 ✅ Validate it  
 ✅ Flag abnormalities  
 ✅ Remove duplicates  
 ✅ Join with reference data (like tank capacity)  
 ✅ Store cleaned and error-flagged data separately

This prepares your data for final reporting in the **Gold Layer**.

📁 Azure Blob Storage (/raw/)

│

▼

📦 Azure Data Factory

│

├── Ingest `bronze\_layer\_updated.csv` ✅

└── Ingest `tank\_reference\_table.csv` ✅

│

▼

📓 Azure Databricks Notebook (Python or PySpark)

│

├── Join with tank reference table ✅

├── Clean bad formats ✅

├── Remove duplicates ✅

├── Flag abnormal records (status) ✅

│ ├─ Negative Sales

│ ├─ Missing fields

│ ├─ Wrong fuel type

│ ├─ Tank level increase

│ └─ Abnormal pricing

│

├── Store clean records ➝ `/cleaned/` folder ✅

└── Store failed/bad records ➝ `/failures/` folder ✅

│

▼

📧 Azure Logic App

└── Send email alert for Low Inventory ✅

Absolutely! Let's now dive into the **Silver Layer** of your **Petroleum Industry Data Engineering project using Azure**, explained in **simple and clear language** — with a full **step-by-step plan**, **flow diagram**, and all tasks you need to perform using **Azure Data Factory, Databricks, Logic Apps, and Data Lake**.

## **🌟 What is the Silver Layer?**

The **Silver Layer** is where we:

✅ Clean the raw data  
 ✅ Validate it  
 ✅ Flag abnormalities  
 ✅ Remove duplicates  
 ✅ Join with reference data (like tank capacity)  
 ✅ Store cleaned and error-flagged data separately

This prepares your data for final reporting in the **Gold Layer**.

## **📊 Silver Layer: Visual Flow**

📁 Azure Blob Storage (/raw/)

│

▼

📦 Azure Data Factory

│

├── Ingest `bronze\_layer\_updated.csv` ✅

└── Ingest `tank\_reference\_table.csv` ✅

│

▼

📓 Azure Databricks Notebook (Python or PySpark)

│

├── Join with tank reference table ✅

├── Clean bad formats ✅

├── Remove duplicates ✅

├── Flag abnormal records (status) ✅

│ ├─ Negative Sales

│ ├─ Missing fields

│ ├─ Wrong fuel type

│ ├─ Tank level increase

│ └─ Abnormal pricing

│

├── Store clean records ➝ `/cleaned/` folder ✅

└── Store failed/bad records ➝ `/failures/` folder ✅

│

▼

📧 Azure Logic App

└── Send email alert for Low Inventory ✅

## **💡 What Tools Are Used?**

| **Tool** | **Role** |
| --- | --- |
| Azure Data Factory | Copies CSV files from Blob into Data Lake or Databricks |
| Azure Databricks | Cleans data, joins with reference, flags issues, writes outputs |
| Azure Data Lake | Stores the Silver layer outputs → /cleaned/ and /failures/ |
| Azure Logic Apps | Sends email alert when tank level is below reorder threshold |

## **✅ Step-by-Step Breakdown (Silver Layer)**

### **1️⃣ Ingest Data (Azure Data Factory)**

* Use **ADF Copy Activity** to:  
  + Ingest bronze\_layer\_updated.csv (raw sales data)
  + Ingest tank\_reference\_table.csv (lookup table)
  + Store both in temporary location or send to **Azure Databricks** directly.

**You are reading from:**

* Source: Azure Blob /raw/

**And sending to:**

* Destination: Azure Data Lake or Databricks Notebook

### **2️⃣ Data Transformation (Azure Databricks)**

Launch a **Databricks notebook** with the following tasks:

#### **🔗 a. Join with Tank Reference Table**

Join the two datasets using TankID to bring in:

* Tank capacity
* Reorder level

#### **🧹 b. Clean the Data**

Check and fix or remove:

* Wrong date formats
* Null or missing fields (like FuelType)
* Normalize TransactionDate format
* Convert data types (e.g., liters as float)

#### **⚠️ c. Flag Abnormalities (Add RecordStatus)**

Create a column called RecordStatus with values like:

* Valid
* NegativeSales
* Duplicate
* LowInventory
* FutureTransaction
* ZeroVolume
* NullFuelType
* InvalidFuelType
* AbnormalPrice
* TankLevelIncreased

#### **📑 d. Remove Duplicates**

Use dropDuplicates() or group + count based on:

* StationID, PumpID, TankID, TransactionDate, VolumeSold\_Liters, etc.

#### **🗂️ e. Store Outputs**

Split and store:

* ✅ **Valid records** → /cleaned/cleaned\_station\_logs.parquet
* ❌ **Invalid/failed records** → /failures/failed\_records.parquet

These are stored in **Azure Data Lake**.

### **3️⃣ Alerting (Azure Logic App)**

Set up a Logic App that:

* Watches /cleaned/ file for updates
* Filters for rows where TankLevel\_PostTxn < ReorderLevel
* Sends a **refill alert** via **Outlook email** with:  
  + StationID
  + TankID
  + FuelType
  + Remaining Fuel
  + Threshold crossed

### **📁 Folder Structure in Data Lake**

| **Folder Path** | **Purpose** |
| --- | --- |
| /raw/ | Bronze layer raw files (original CSVs) |
| /cleaned/ | Silver layer cleaned, valid data |
| /failures/ | Invalid/abnormal flagged records |
| /reference/ | Tank reference table, lookups |

## **🧠 BONUS: KPIs You Can Calculate in Silver**

You can also prepare some **pre-aggregated fields** here:

* **PricePerLiter** = AmountCollected / VolumeSold\_Liters
* **DailyVolumePerStation**
* **TotalRevenuePerStation**
* **TotalLowInventoryAlerts**

You’ll use these in Gold Layer dashboards.

## **✅ Summary Checklist**

| **Step** | **Action** | **Tool** | **Output** |
| --- | --- | --- | --- |
| 1 | Ingest raw & reference files | Azure Data Factory | Bronze to Data Lake |
| 2 | Join, clean, validate, flag, and split valid/invalid records | Azure Databricks | /cleaned/, /failures/ |
| 3 | Trigger email alerts for low fuel | Azure Logic Apps | Refill alert in Outlook |

Would you like me to now:

* Help you write the **Databricks notebook code**?
* Show how to set up the **Logic App email alert**?

Let’s continue building!

Absolutely! You’re already covering all the core elements of a professional Silver layer — but here are **additional smart ideas and enhancements** you can implement in the **Silver layer** to make your pipeline even more robust, intelligent, and realistic based on your petroleum industry use case:

## **💡 Additional Enhancements for Silver Layer**

### **1️⃣ Add Derived Metrics/KPIs for Insight**

Instead of just cleaning, make the Silver layer smarter by **generating helpful fields**.

| **Derived Field** | **Formula / Use** | **Benefit** |
| --- | --- | --- |
| PricePerLiter | AmountCollected / VolumeSold\_Liters | Detect abnormal pricing |
| EstimatedRefillDate | Predict when tank will empty (see #2) | Helps logistics teams |
| SaleType | e.g., Retail / B2B based on volume | Helps segment customer behavior |

### **2️⃣ Tank Depletion Rate Tracking**

You can estimate **how fast a tank is emptying** and predict **when it needs to be refilled**, even before it crosses the reorder level.

**How to do it:**

* Track how much fuel is sold per day from each tank.
* Calculate average daily usage (rolling average).
* Estimate: EstimatedRefillDate = TankLevel / AvgDailyUsage

### **3️⃣ Track Station Performance (Basic Trend Data)**

In Silver layer, pre-calculate:

* Total sales per day per station
* High-performing vs. low-performing stations
* % of fuel type sold (Petrol vs. Diesel)

This can feed into Gold layer dashboards but saves processing time later.

### **4️⃣ Fuel-Type Validation by Tank**

Cross-check if the FuelType matches what's allowed in a tank (from reference data):

* If TK1 is Petrol, but Diesel is logged → flag as "FuelTypeMismatch"

Add this to your reference table:

TankID,FuelType,TankCapacity\_Liters,ReorderLevel\_Liters

TK1,Petrol,5000,1000

TK2,Diesel,4000,800

### **5️⃣ Time-Based Duplication Detection**

Check for same StationID + PumpID + Timestamp within a 1–2 minute window. These could be **accidental double entries** (especially if same volume sold).

Use this logic to flag:

* "TimeDuplicate" instead of just exact duplicates

### **6️⃣ Flag Suspicious Station Activity**

If one station shows:

* Extremely high fuel sales daily (e.g., more than 95% of tank)
* Tank Level increasing without a refill record
* Continuous negative or zero sales

➡ Flag that station for **"SuspiciousActivity"** or **"AuditRequired"**

This can be sent to management automatically.

### **7️⃣ Geo-Based Enhancements (Optional Advanced)**

If StationID is linked with a location:

* Track fuel demand by region
* Compare coastal vs. inland fuel demand

You can link to a separate station metadata file:

StationID,Location,Latitude,Longitude

STN001,Hyderabad,17.385,78.4867

### **8️⃣ Silver Metadata Logging Table**

Maintain a silver\_processing\_log.csv that tracks:

* When each file was processed
* Number of valid, invalid, duplicate, alert-triggering rows
* Processing time

This helps with **observability** and **auditing** in production pipelines.

## **🧠 Summary: Smart Additions for Silver Layer**

| **Feature** | **Impact** |
| --- | --- |
| Derived KPIs like Price/Liter | Detect errors + support dashboards |
| Tank depletion prediction | Forecast refills proactively |
| Fuel type validation per tank | Prevent contamination & catch config errors |
| Suspicious activity detection | Improve fraud/risk management |
| Metadata tracking | Add observability & governance |
| Regional sales analysis (geo) | Enable geo-based planning |

Would you like me to:

* Help generate these new derived columns and validations in code?
* Create a mock EstimatedRefillDate and PricePerLiter calculation in your dataset?

Let’s make your Silver layer intelligent — not just clean, but insightful!

✅ All the **transformations, joins, validations, and logic** for **cleaning and detecting abnormalities** will be done using **Azure Databricks** (via notebooks — PySpark or Pandas).  
 ➡️ This will produce two outputs:

* ✅ cleaned\_station\_logs.csv/parquet → **Valid cleaned records**
* ❌ abnormal\_records.csv/parquet → **Flagged/error records**

IoT/POS CSVs

↓

Azure Blob (/raw/)

↓ [Copy]

ADF Pipeline ─────────► Reference Table (/reference/)

↓ │

Azure Data Lake (/raw/) ▼

Databricks Notebook

├─ Join + Clean + Validate

├─ Flag Abnormalities (RecordStatus)

├─ Add Derived Fields (Price/Liter, etc.)

├─ Split:

│ ├─ ✅ Valid → /cleaned/

│ └─ ❌ Invalid → /failures/

└─ Save Logs (optional)

↓

Logic App watches for Low Inventory

↓

🔔 Sends Outlook Alert

Perfect — now let’s wrap it up with the **Gold Layer** of your project. This is where all the hard work in the Bronze and Silver layers pays off!

## **🥇 GOLD LAYER – Final Aggregation, KPIs & Visualization**

### **🎯 Main Goal of Gold Layer:**

Provide **business-ready, summarized, and visual data insights** — easy to understand, fast to access, and ready for dashboards.

This layer is where stakeholders (like operations, logistics, or management) **consume** the data to:

* Monitor performance
* Detect leakage
* Plan refills
* Track revenue

## **🔁 High-Level Flow**

/cleaned/ (from Silver)

↓

Databricks Notebook (Gold Layer)

↓

📊 Aggregate & Calculate KPIs

↓

Save summarized data ➝ Azure Data Lake (/aggregated/)

↓

📈 Visualize in Databricks (Dashboards/Charts)

## **✅ What is Done in Gold Layer?**

### **1️⃣ Read Cleaned Data**

* Read cleaned\_station\_logs.parquet from /cleaned/ (Silver output)

### **2️⃣ Aggregate and Calculate KPIs**

| **KPI / Report Name** | **Calculation/Logic** |
| --- | --- |
| 🏪 **Total Sales per Station** | Sum of VolumeSold\_Liters and AmountCollected by StationID daily |
| ⛽ **Tank Refill Report** | If tank level increased between two records → mark as refill |
| ⚠️ **Leakage Detection** | Tank level drops but **no sales recorded** → potential leak |
| 💰 **Revenue per Fuel Type** | Aggregate AmountCollected grouped by FuelType and date |
| 📊 **Daily KPI Summary** | Total volume sold, revenue, number of alerts (per day) |
| ⏱️ **Average Time Between Refills** | For each tank, compute days between refills |

### **3️⃣ Save Final Output to Data Lake**

Create and store these datasets as Parquet files:

| **Folder Path** | **Description** |
| --- | --- |
| /aggregated/daily\_kpis.parquet | All per-station, per-day KPIs |
| /aggregated/refill\_reports.parquet | Refill events with timestamps |
| /aggregated/leakage\_reports.parquet | Detected anomalies related to leakage |
| /aggregated/fuel\_sales\_by\_type.parquet | Sales grouped by fuel type |

🔐 These files are now **read-optimized and ready for visualization or Power BI**.

### **4️⃣ Visualization in Databricks**

Use **Databricks Notebooks** to:

* Load /aggregated/ files
* Create charts directly using %matplotlib, plotly, or built-in visualizer

You can build dashboards like:

* 🔵 Station-wise sales heatmap
* 📈 Daily trend of petrol vs diesel
* 🟡 Tank refill trend timeline
* 🚨 Live low-inventory or leakage alert board

## **🧠 Optional Gold Layer Add-ons**

| **Idea** | **Why It's Useful** |
| --- | --- |
| Upload dashboards to Power BI | For business team to view outside Databricks |
| Auto-generate email summary | Send daily KPI report to stakeholders via Logic App |
| Save snapshot CSVs weekly | For audit trail or backup purposes |

## **✅ Final Folder Structure Summary**

/raw/ → Bronze (raw CSVs)

/reference/ → Reference lookup tables

/cleaned/ → Silver (clean, valid records)

/failures/ → Silver (flagged bad records)

/aggregated/ → Gold (final KPIs, refill logs, leakage reports)

/logs/ → Job run metadata (optional)

## **🏁 Final Thoughts: Gold Layer Role**

* This layer is **the final product** that your end-users (logistics, sales, ops, management) will use.
* You’re now creating **real, decision-driving insights** from raw fuel data.
* It makes your pipeline not just technically sound — but **business impactful**.

| **Insight / Dashboard** | **Why It’s Valuable** |
| --- | --- |
| 🔔 **Low Fuel Alert** | Ensures no fuel shortage — improves customer experience |
| 📉 **Leakage Detection** | Saves money by identifying fuel loss early |
| ⛽ **Tank Refill Timeline** | Helps schedule fuel truck deliveries efficiently |
| 💰 **Revenue by Fuel Type/Station** | Shows which fuel is most profitable, and which station performs better |
| 🏷️ **Unusual Sales Flags** | Detects data errors, fraud, or billing issues |
| 📊 **Daily KPI Summary** | Gives decision-makers a quick health check of daily operations |

## **Real-World Use Case Scenario (Example)**

Let’s say a company runs **50+ petrol stations** across India. Using this project:

* Every station uploads daily sales
* Managers get **automated alerts** when fuel is low
* Logistics knows exactly **which tanks to refill, when, and how much**
* Owners can view dashboards showing **sales, trends, and issues in real time**
* Any **fraud or error** (like negative sales or fake data) is **caught immediately**

This kind of architecture is exactly what **fuel distribution companies**, **logistics operators**, or **retail station chains like HPCL, IOCL, BPCL, Shell** would use.

Absolutely! Here's a **complete end-to-end summary** of your **Petroleum Industry Data Engineering Project using Azure** — explained clearly and simply, along with improvement ideas at each stage.

## **🛠️ PROJECT SUMMARY: BEGINNING TO END**

### **🔹 1. Bronze Layer – Raw Data Ingestion**

**What we do:**

* Receive raw .csv data from fuel stations (IoT or POS systems)
* Upload it to **Azure Blob Storage**
* Use **Azure Data Factory (ADF)** to copy data to **Azure Data Lake → /raw/**
* Add audit fields: CreatedBy, CreatedTime, etc.
* Maintain a reference table (tank\_reference\_table.csv) with TankID, fuel type, capacity, reorder level

**Purpose:** Safely store unprocessed data for further transformation

**Improvements:**

* Add schema validation in ADF (to reject corrupted or mismatched files)
* Add metadata log (bronze\_ingestion\_log.csv) for audit trail

### **🥈 2. Silver Layer – Data Cleaning & Validation**

**What we do:**

* Use ADF to send raw + reference data to **Azure Databricks Notebook**
* In Databricks:  
  + Join raw data with tank reference data
  + Clean formats, nulls, and data types
  + Add **RecordStatus** to flag abnormalities:  
    - Negative sales
    - Missing/invalid fuel type
    - Duplicate entries
    - Tank level increasing after sale
    - Abnormal price, future dates, zero volume
  + Add derived fields:  
    - PricePerLiter
    - (Optional) EstimatedRefillDate
  + Separate valid and invalid records
  + Save outputs:  
    - ✅ /cleaned/cleaned\_station\_logs.parquet
    - ❌ /failures/abnormal\_records.parquet

**Purpose:** Turn raw data into trusted, enriched data ready for insights

**Improvements:**

* Track and log % of abnormal records over time (for trend detection)
* Add advanced anomaly detection using ML (optional)
* Geo-enrich data with station location (lat/lon)

### **🥇 3. Gold Layer – Aggregation, KPIs, Visualization**

**What we do:**

* Read /cleaned/ data in **Databricks**
* Generate **aggregated KPIs**:  
  + Daily sales & revenue per station
  + Refill events (tank level increases)
  + Leakage detection (tank level drops without sales)
  + Fuel type sales comparison
  + Inventory health per station
* Save results to /aggregated/ folder:  
  + daily\_kpis.parquet
  + refill\_reports.parquet
  + leakage\_reports.parquet
* Visualize in **Databricks dashboards** or export to Power BI

**Purpose:** Give stakeholders actionable insights via KPIs and visual dashboards

**Improvements:**

* Schedule daily summary emails via **Logic Apps** (KPI report)
* Use Power BI for more flexible dashboards
* Save weekly snapshots for historical comparison

### **📧 Logic App Integration (Cross-layer)**

**What we do:**

* Watch /cleaned/ folder
* Detect if TankLevel\_PostTxn < ReorderLevel
* Send **refill alert email** to operations team

**Improvements:**

* Extend alerts to abnormal sales, high leakage, or missing data
* Store alerts in /logs/alerts\_log.csv

## **✅ FINAL OUTCOME**

You now have a **production-ready pipeline** that:

* Captures raw fuel data automatically
* Cleans and enriches it
* Detects problems
* Sends alerts
* Produces visual dashboards with KPIs

This system helps:

* ⛽ Fuel station managers avoid stockouts
* 🚚 Logistics teams schedule refills smartly
* 💰 Executives track revenue and performance
* 🔍 Audit teams detect fraud or data issues

## **🔄 FUTURE IMPROVEMENTS (Optional)**

| **Feature** | **Stage** | **Benefit** |
| --- | --- | --- |
| Schema evolution & validation | Bronze | Prevent bad data ingestion |
| ML anomaly detection | Silver | More advanced pattern recognition |
| Station geolocation mapping | Silver | Region-wise performance |
| Power BI Integration | Gold | Executive-friendly dashboards |
| ML Refill prediction | Gold | Predict future tank refill needs |
| Data quality score | Silver/Gold | Monitor reliability of each station's data |

Would you like a final **PDF or presentation version** of this summary for easy sharing or showcasing the project?