# SSupply Chain ETL & Analytics Report

--Sayan Das

## Introduction

In this project, I developed a complete ETL pipeline for a supply chain and inventory management scenario. I extracted data from CSV files, performed data quality checks and transformations, built a star schema to support fast and efficient reporting, loaded the transformed data into a MySQL database, and wrote SQL queries to perform key analyses. This report outlines each step of the process in detail.

## 1. Data Extraction

I started by loading the CSV files that contained data for sales, inventory, suppliers, and purchase orders. For this purpose, I used the Pandas library to read the CSVs and convert date columns to proper datetime objects:

import pandas as pd  
import numpy as np  
*# Load CSV files into DataFrames with appropriate date parsing*  
sales\_file = "sales\_data-2.csv"

inventory\_file = "inventory\_data.csv"

suppliers\_file = "suppliers\_data.csv"

purchase\_orders\_file = "purchase\_orders\_data.csv"

# Read CSV files

sales\_df = pd.read\_csv(sales\_file, parse\_dates=["Sale\_Date"])

inventory\_df = pd.read\_csv(inventory\_file, parse\_dates=["Last\_Updated"])

suppliers\_df = pd.read\_csv(suppliers\_file)

purchase\_orders\_df = pd.read\_csv(purchase\_orders\_file, parse\_dates=["Order\_Date", "Arrival\_Date"])

## 2. Data Cleaning & Transformation

### 2.1 Data Quality Checks

I performed multiple data quality checks to ensure the integrity of the data:

* **Missing Data:**  
  I examined each DataFrame for missing values in key columns (such as primary keys) and dropped rows that were missing essential identifiers.
* **Duplicate Handling:**  
  For tables with single-column primary keys (Sales, Suppliers, and Purchase Orders), I dropped duplicate records by keeping the first occurrence.  
  The Inventory data required special attention because it has a composite key (Product\_ID, Store\_ID, Warehouse\_ID). I sorted this data by Last\_Updated (so the most recent record came first) and then removed duplicates based on the composite key.

# Check for missing values in critical columns ---

print("Missing values in Sales Data:\n", sales\_df.isnull().sum(), "\n")

print("Missing values in Inventory Data:\n", inventory\_df.isnull().sum(), "\n")

print("Missing values in Suppliers Data:\n", suppliers\_df.isnull().sum(), "\n")

print("Missing values in Purchase Orders Data:\n", purchase\_orders\_df.isnull().sum(), "\n")

# Drop rows missing primary key fields

sales\_df.dropna(subset=["Sale\_ID"], inplace=True)

suppliers\_df.dropna(subset=["Supplier\_ID"], inplace=True)

purchase\_orders\_df.dropna(subset=["Order\_ID"], inplace=True)

inventory\_df.dropna(subset=["Product\_ID", "Store\_ID", "Warehouse\_ID"], inplace=True)

# Ensure single-column primary keys are unique (Sales, Suppliers, Purchase Orders) ---

if sales\_df["Sale\_ID"].duplicated().any():

    print("Duplicate Sale\_ID found. Keeping first occurrence.")

    sales\_df.drop\_duplicates(subset=["Sale\_ID"], keep="first", inplace=True)

if suppliers\_df["Supplier\_ID"].duplicated().any():

    print("Duplicate Supplier\_ID found. Keeping first occurrence.")

    suppliers\_df.drop\_duplicates(subset=["Supplier\_ID"], keep="first", inplace=True)

if purchase\_orders\_df["Order\_ID"].duplicated().any():

    print("Duplicate Order\_ID found. Keeping first occurrence.")

    purchase\_orders\_df.drop\_duplicates(subset=["Order\_ID"], keep="first", inplace=True)

# Resolve composite key duplicates in Inventory ---

# (Product\_ID, Store\_ID, Warehouse\_ID) must be unique; keep the latest Last\_Updated

inventory\_df.sort\_values(by="Last\_Updated", ascending=False, inplace=True)

inventory\_df.drop\_duplicates(subset=["Product\_ID", "Store\_ID", "Warehouse\_ID"], keep="first", inplace=True)

### 2.2 Building the Star Schema

To enable efficient analytical queries, I transformed the data into a star schema by creating separate dimension and fact tables. The main dimensions I built include:

* **dim\_products:** Contains unique product IDs with surrogate keys.
* **dim\_suppliers:** Consolidates supplier information with a surrogate key.
* **dim\_stores:** Contains unique store IDs.
* **dim\_warehouses:** Contains unique warehouse IDs.
* **dim\_dates:** A date dimension built from all unique dates in the datasets, including year, month, and day.

The fact tables include:

* **fact\_sales:** Records sales transactions with references (via surrogate keys) to products, stores, and dates.
* **fact\_inventory:** Records current inventory levels with references to products, stores, warehouses, and last update dates.
* **fact\_purchase\_orders:** Records purchase orders with references to products, suppliers, and both order and arrival dates.

Below is a snippet showing how I built the dimension tables and mapped surrogate keys:

# --- 3.1 Dimension: dim\_products ---

# Collect unique Product\_ID from all tables referencing products

all\_product\_ids = set(sales\_df["Product\_ID"].dropna().unique()) \

    .union(inventory\_df["Product\_ID"].dropna().unique()) \

    .union(purchase\_orders\_df["Product\_ID"].dropna().unique()) \

    .union(suppliers\_df["Product\_ID"].dropna().unique())

dim\_products = pd.DataFrame({"Product\_ID": sorted(all\_product\_ids)})

dim\_products["product\_key"] = range(1, len(dim\_products) + 1)

dim\_products = dim\_products[["product\_key", "Product\_ID"]].copy()

# --- 3.2 Dimension: dim\_suppliers ---

# Group by Supplier\_ID to ensure one row per supplier

suppliers\_gb = suppliers\_df.groupby("Supplier\_ID", as\_index=False).agg({

    "Supplier\_Name": "first",

    "Lead\_Time (days)": "first",

    "Order\_Frequency": "first"

})

dim\_suppliers = suppliers\_gb.copy()

dim\_suppliers["supplier\_key"] = range(1, len(dim\_suppliers) + 1)

dim\_suppliers = dim\_suppliers[[

    "supplier\_key",

    "Supplier\_ID",

    "Supplier\_Name",

    "Lead\_Time (days)",

    "Order\_Frequency"

]].copy()

# --- 3.3 Dimension: dim\_stores ---

# Gather unique Store\_IDs from sales and inventory

all\_store\_ids = set(sales\_df["Store\_ID"].dropna().unique()) \

    .union(inventory\_df["Store\_ID"].dropna().unique())

dim\_stores = pd.DataFrame({"Store\_ID": sorted(all\_store\_ids)})

dim\_stores["store\_key"] = range(1, len(dim\_stores) + 1)

dim\_stores = dim\_stores[["store\_key", "Store\_ID"]].copy()

# --- 3.4 Dimension: dim\_warehouses ---

# Gather unique Warehouse\_ID from inventory

all\_warehouse\_ids = set(inventory\_df["Warehouse\_ID"].dropna().unique())

dim\_warehouses = pd.DataFrame({"Warehouse\_ID": sorted(all\_warehouse\_ids)})

dim\_warehouses["warehouse\_key"] = range(1, len(dim\_warehouses) + 1)

dim\_warehouses = dim\_warehouses[["warehouse\_key", "Warehouse\_ID"]].copy()

# --- 3.5 Dimension: dim\_dates ---

# Collect all date columns: Sale\_Date, Last\_Updated, Order\_Date, Arrival\_Date

dates\_sales = sales\_df["Sale\_Date"].dropna().unique()

dates\_inv = inventory\_df["Last\_Updated"].dropna().unique()

dates\_po\_order = purchase\_orders\_df["Order\_Date"].dropna().unique()

dates\_po\_arrival = purchase\_orders\_df["Arrival\_Date"].dropna().unique()

all\_dates = pd.Series(list(dates\_sales) + list(dates\_inv) + list(dates\_po\_order) + list(dates\_po\_arrival)).unique()

all\_dates = pd.to\_datetime(all\_dates)

all\_dates = sorted(all\_dates)

dim\_dates = pd.DataFrame({"date": all\_dates})

dim\_dates["date\_key"] = range(1, len(dim\_dates) + 1)

dim\_dates["year"] = dim\_dates["date"].dt.year

dim\_dates["month"] = dim\_dates["date"].dt.month

dim\_dates["day"] = dim\_dates["date"].dt.day

dim\_dates = dim\_dates[["date\_key", "date", "year", "month", "day"]].copy()

# =============== Helper Functions for Surrogate Key Mapping ===============

def map\_product\_key(df, product\_id\_col):

    return pd.merge(

        df,

        dim\_products,

        how="left",

        left\_on=product\_id\_col,

        right\_on="Product\_ID"

    )

def map\_supplier\_key(df, supplier\_id\_col):

    return pd.merge(

        df,

        dim\_suppliers,

        how="left",

        left\_on=supplier\_id\_col,

        right\_on="Supplier\_ID"

    )

def map\_store\_key(df, store\_id\_col):

    return pd.merge(

        df,

        dim\_stores,

        how="left",

        left\_on=store\_id\_col,

        right\_on="Store\_ID"

    )

def map\_warehouse\_key(df, warehouse\_id\_col):

    return pd.merge(

        df,

        dim\_warehouses,

        how="left",

        left\_on=warehouse\_id\_col,

        right\_on="Warehouse\_ID"

    )

def map\_date\_key(df, date\_col):

    # merges on exact match of date

    return pd.merge(

        df,

        dim\_dates,

        how="left",

        left\_on=date\_col,

        right\_on="date"

    )

# --- 4.1 fact\_sales ---

# Original columns: [Sale\_ID, Product\_ID, Store\_ID, Sale\_Date, Quantity\_Sold, Revenue]

fact\_sales = sales\_df.copy()

fact\_sales = map\_product\_key(fact\_sales, "Product\_ID")

fact\_sales = map\_store\_key(fact\_sales, "Store\_ID")

fact\_sales = map\_date\_key(fact\_sales, "Sale\_Date")

fact\_sales = fact\_sales[[

    "Sale\_ID",

    "product\_key",

    "store\_key",

    "date\_key",

    "Quantity\_Sold",

    "Revenue"

]].copy()

# --- 4.2 fact\_inventory ---

# Original columns: [Product\_ID, Store\_ID, Warehouse\_ID, Stock\_Level, Reorder\_Level, Last\_Updated]

fact\_inventory = inventory\_df.copy()

fact\_inventory = map\_product\_key(fact\_inventory, "Product\_ID")

fact\_inventory = map\_store\_key(fact\_inventory, "Store\_ID")

fact\_inventory = map\_warehouse\_key(fact\_inventory, "Warehouse\_ID")

fact\_inventory = map\_date\_key(fact\_inventory, "Last\_Updated")

fact\_inventory = fact\_inventory[[

    "product\_key",

    "store\_key",

    "warehouse\_key",

    "Stock\_Level",

    "Reorder\_Level",

    "date\_key"  # or rename to 'last\_updated\_date\_key'

]].copy()

# --- 4.3 fact\_purchase\_orders ---

# Original columns: [Order\_ID, Product\_ID, Supplier\_ID, Order\_Date, Quantity, Arrival\_Date]

fact\_purchase\_orders = purchase\_orders\_df.copy()

# Map product\_key & supplier\_key

fact\_purchase\_orders = map\_product\_key(fact\_purchase\_orders, "Product\_ID")

fact\_purchase\_orders = map\_supplier\_key(fact\_purchase\_orders, "Supplier\_ID")

# Map order\_date\_key

fact\_purchase\_orders = map\_date\_key(fact\_purchase\_orders, "Order\_Date")

fact\_purchase\_orders.rename(columns={"date\_key": "order\_date\_key"}, inplace=True)

# Map arrival\_date\_key

fact\_purchase\_orders = map\_date\_key(fact\_purchase\_orders, "Arrival\_Date")

fact\_purchase\_orders.rename(columns={"date\_key": "arrival\_date\_key"}, inplace=True)

fact\_purchase\_orders = fact\_purchase\_orders[[

    "Order\_ID",

    "product\_key",

    "supplier\_key",

    "order\_date\_key",

    "Quantity",

    "arrival\_date\_key"

]].copy()

## 3. Loading Data into MySQL

After transforming the data, I loaded the dimension and fact tables into a MySQL database using SQLAlchemy. This allowed for efficient querying and integration with BI tools like Power BI.

from sqlalchemy import create\_engine

import pymysql

username = 'root'

password = '12345'

host = 'localhost'

port = '3306'

database = 'case4'

engine = create\_engine(f"mysql+pymysql://{username}:{password}@{host}:{port}/{database}")

# --- 5.1 Write dimension tables ---

dim\_products.to\_sql("dim\_products", con=engine, if\_exists="replace", index=False)

dim\_suppliers.to\_sql("dim\_suppliers", con=engine, if\_exists="replace", index=False)

dim\_stores.to\_sql("dim\_stores", con=engine, if\_exists="replace", index=False)

dim\_warehouses.to\_sql("dim\_warehouses", con=engine, if\_exists="replace", index=False)

dim\_dates.to\_sql("dim\_dates", con=engine, if\_exists="replace", index=False)

# --- 5.2 Write fact tables ---

fact\_sales.to\_sql("fact\_sales", con=engine, if\_exists="replace", index=False)

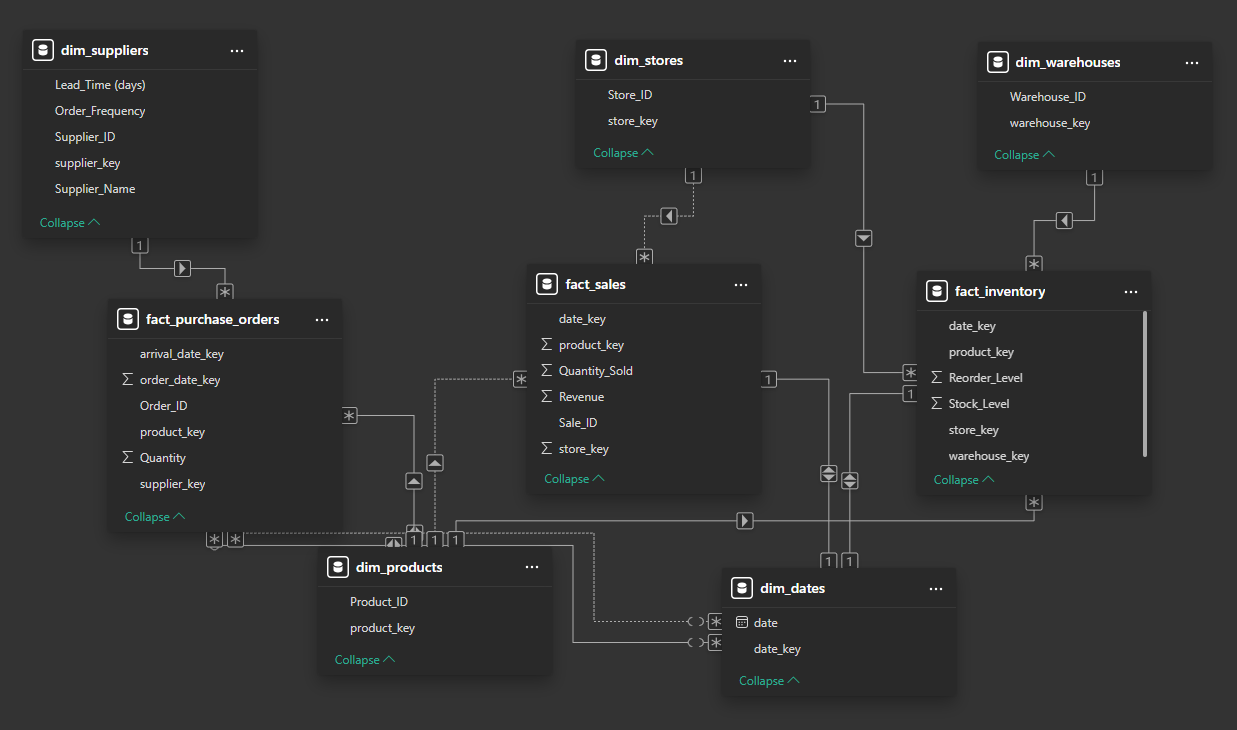
fact\_inventory.to\_sql("fact\_inventory", con=engine, if\_exists="replace", index=False)

fact\_purchase\_orders.to\_sql("fact\_purchase\_orders", con=engine, if\_exists="replace", index=False)

print("Star Schema tables loaded successfully into MySQL.")

## 4. SQL Analysis for Supply Chain Insights

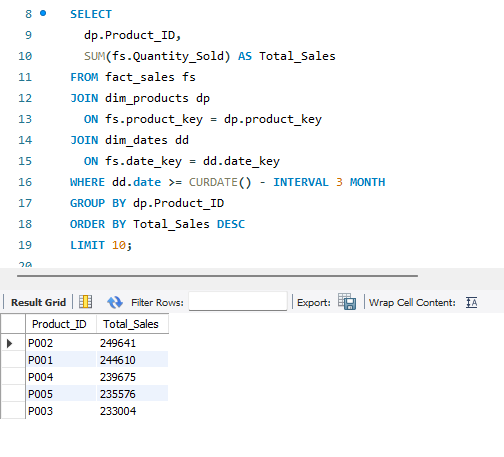
Schema



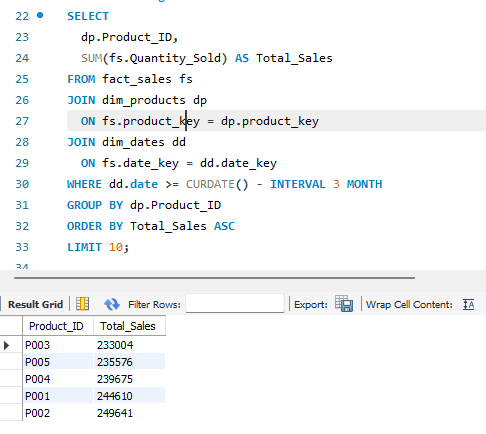
### 4.1 Identifying Fast-Moving and Slow-Moving Products

These queries aggregate sales data over the past three months to determine the highest and lowest sales performers.

**Fast-Moving Products:**



**Slow-Moving Products:**



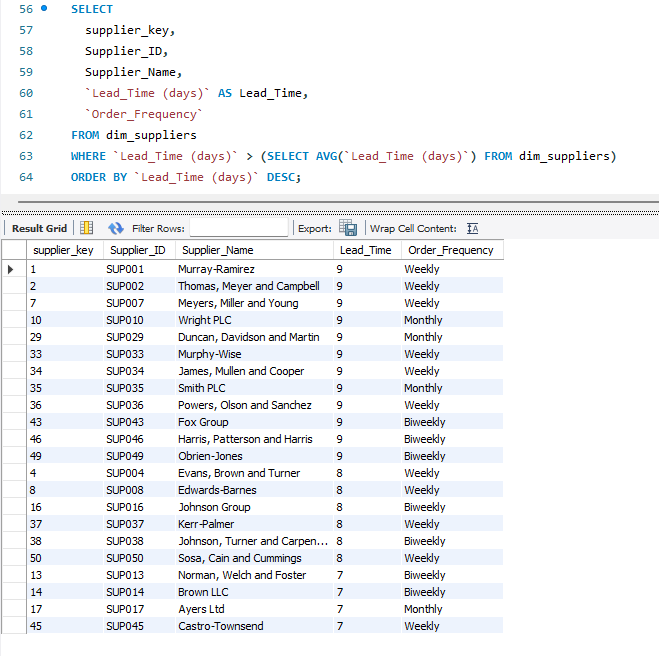
### 4.2 Reporting Products Below Reorder Level

This query identifies products where the current stock level is lower than the reorder level:

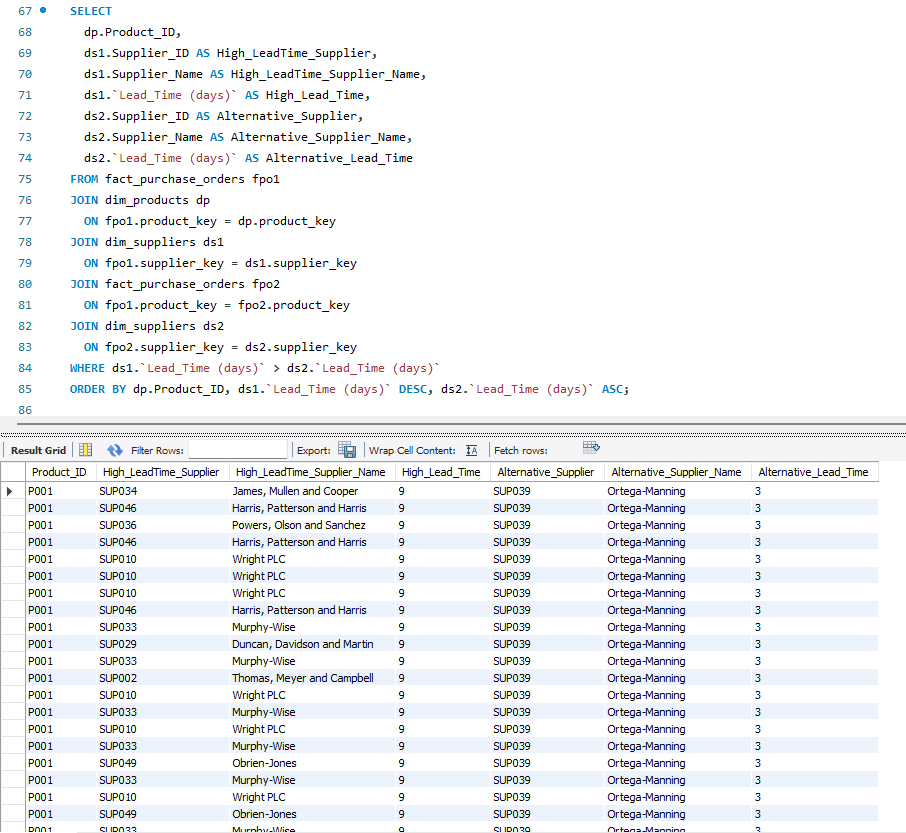
### 4.3 Supplier Lead Time Analysis

I performed an analysis to identify suppliers with above-average lead times, and then suggested alternative suppliers with lower lead times for the same products.

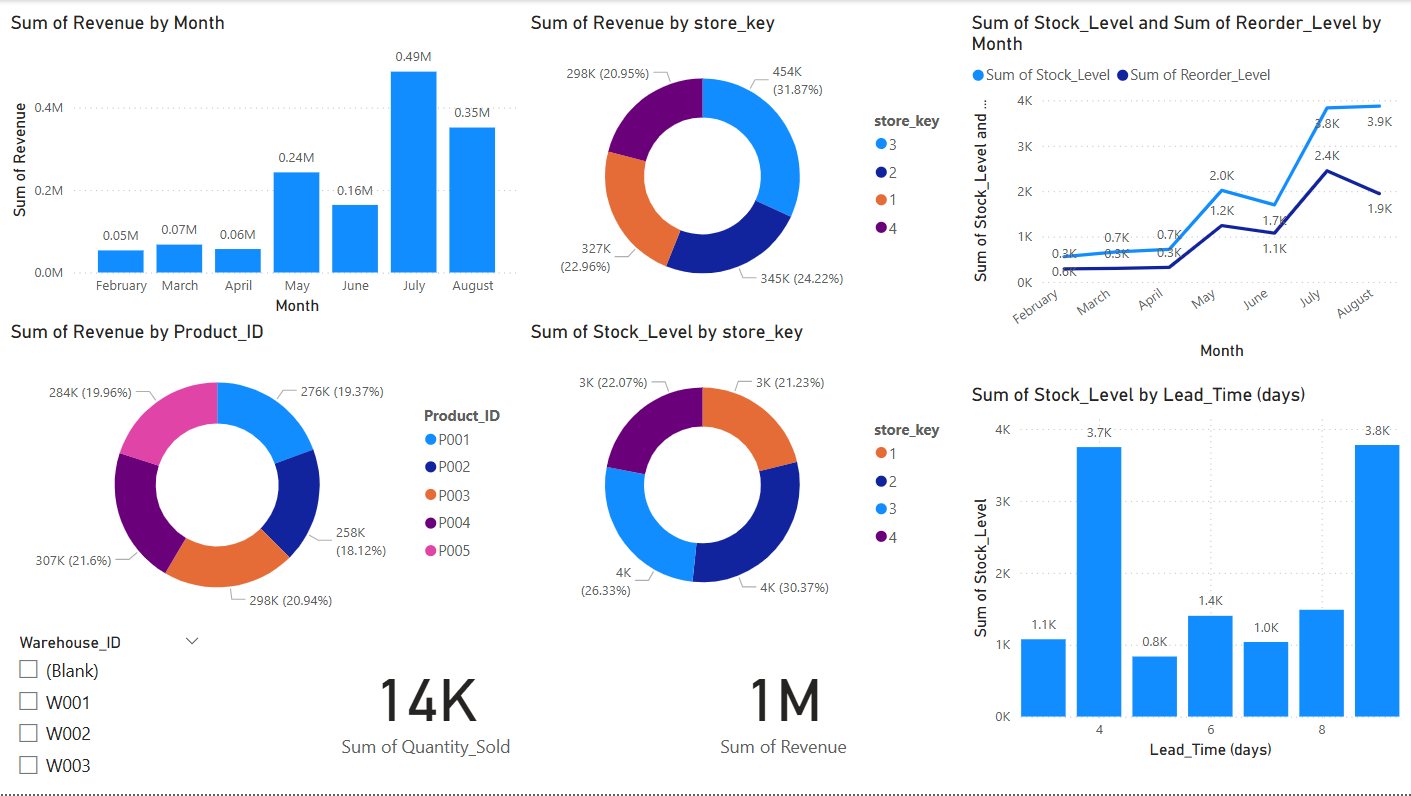
**Identify Suppliers with High Lead Times:**

****

**Suggest Alternative Suppliers:**



## 5. Power Bi Report



## Conclusion

**Monthly Revenue Trends**

* The **bar chart** showing “Sum of Revenue by Month” indicates that August has the highest revenue. This suggests a seasonal or promotional effect in late summer. If earlier months are lower, it might mean either a ramp-up in sales activity, a marketing push, or a seasonal demand spike.

**Revenue Distribution by Products and Stores**

* The **pie charts** for “Sum of Revenue by Product\_ID” and “Sum of Revenue by store\_key” show which products and which stores are driving the largest portion of overall revenue. A few products/stores may dominate the pie, implying either a focus on certain top-sellers or that some stores outperform others significantly.

**Inventory Levels vs. Reorder Levels**

* The **line chart** comparing “Sum of Stock\_Level” and “Sum of Reorder\_Level by Month” reveals how well current inventory matches the reorder thresholds over time. If the stock level frequently dips near or below the reorder line, it signals a risk of stockouts or a need to reorder more proactively. Conversely, if the stock level is always far above the reorder line, it might indicate overstocking or higher carrying costs.

**Store Inventory Distribution**

* The **pie chart** for “Sum of Stock\_Level by store\_key” highlights which stores carry the most inventory. If one store holds disproportionately high stock, you may want to investigate whether that location has higher demand or if it’s overstocked compared to others.