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()
```

```python
# --- 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.

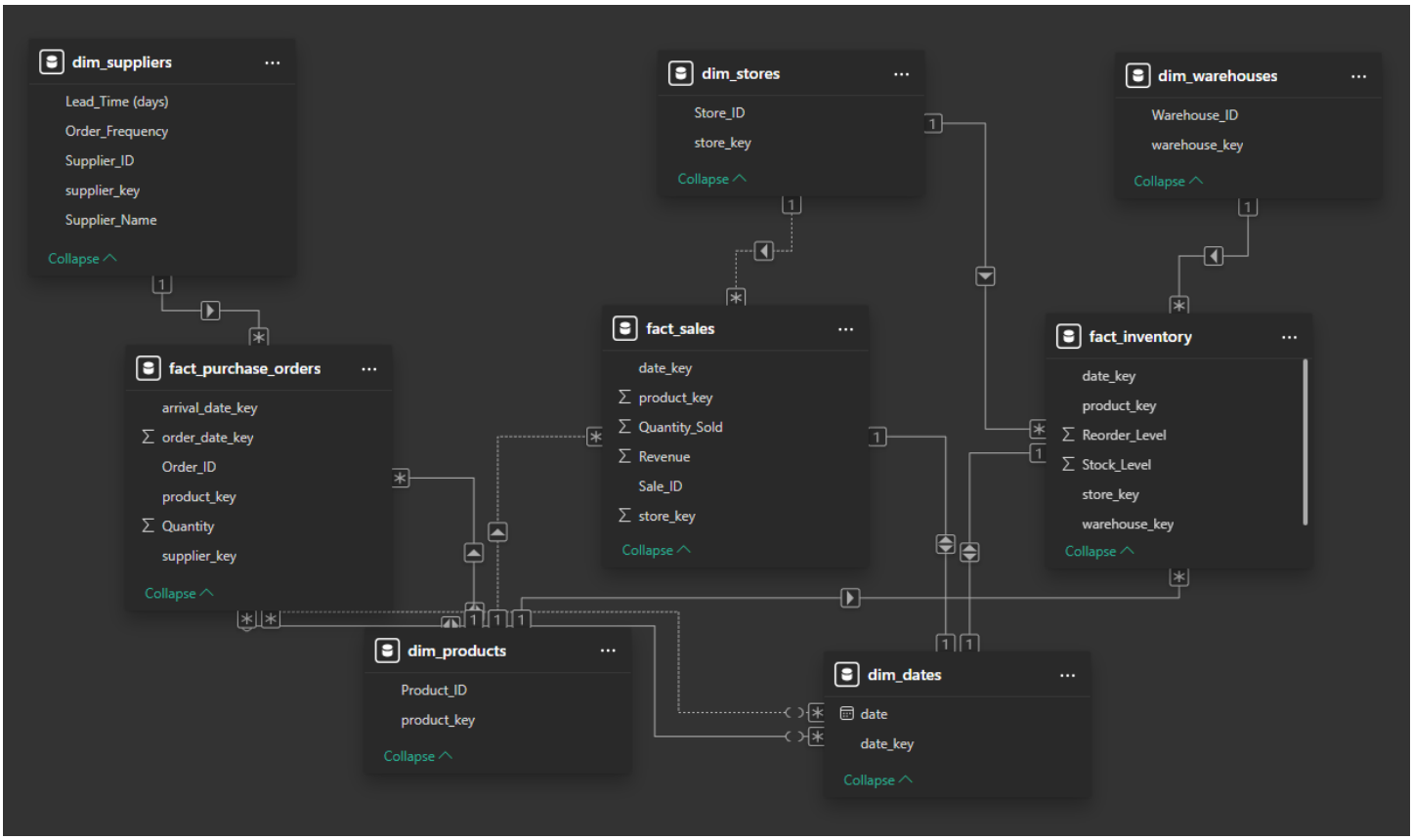
```
from sqlalchemy import create_engine
import pymysql

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:

```sql
SELECT
  dp.Product_ID,
  SUM(fs.Quantity_Sold) AS Total_Sales
FROM fact_sales fs
JOIN dim_products dp
  ON fs.product_key = dp.product_key
JOIN dim_dates dd
  ON fs.date_key = dd.date_key
WHERE dd.date >= CURDATE() - INTERVAL 3 MONTH
GROUP BY dp.Product_ID
ORDER BY Total_Sales DESC
LIMIT 10;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Product_ID	Total_Sales
P002	249641
P001	244610
P004	239675
P005	235576
P003	233004

Slow-Moving Products:

```sql
SELECT
  dp.Product_ID,
  SUM(fs.Quantity_Sold) AS Total_Sales
FROM fact_sales fs
JOIN dim_products dp
  ON fs.product_key = dp.product_key
JOIN dim_dates dd
  ON fs.date_key = dd.date_key
WHERE dd.date >= CURDATE() - INTERVAL 3 MONTH
GROUP BY dp.Product_ID
ORDER BY Total_Sales ASC
LIMIT 10;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Product_ID	Total_Sales
P003	233004
P005	235576
P004	239675
P001	244610
P002	249641

4.2 Reporting Products Below Reorder Level

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

```
36 •  SELECT
37      dp.Product_ID,
38      ds.Store_ID,
39      dw.Warehouse_ID,
40      fi.Stock_Level,
41      fi.Reorder_Level,
42      dd.date AS Last_Updated
43    FROM fact_inventory fi
44    JOIN dim_products dp
45      ON fi.product_key = dp.product_key
46    JOIN dim_stores ds
47      ON fi.store_key = ds.store_key
48    JOIN dim_warehouses dw
49      ON fi.warehouse_key = dw.warehouse_key
50    JOIN dim_dates dd
51      ON fi.date_key = dd.date_key
52    WHERE fi.Stock_Level < fi.Reorder_Level;
53
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Product_ID	Store_ID	Warehouse_ID	Stock_Level	Reorder_Level	Last_Updated
P005	S102	W002	66	119	2024-02-25 00:00:00
P004	S104	W001	62	79	2024-03-23 00:00:00
P004	S101	W001	43	165	2024-05-08 00:00:00
P005	S101	W003	91	167	2024-05-10 00:00:00
P001	S103	W002	34	118	2024-05-28 00:00:00
P005	S104	W003	11	194	2024-06-02 00:00:00
P002	S104	W002	156	193	2024-06-29 00:00:00
P003	S101	W001	109	160	2024-07-01 00:00:00
P002	S101	W001	97	168	2024-07-03 00:00:00
P002	S102	W003	121	173	2024-07-09 00:00:00
P002	S103	W003	79	141	2024-07-10 00:00:00
P004	S101	W003	100	136	2024-07-30 00:00:00
P003	S101	W002	104	144	2024-07-31 00:00:00
P005	S103	W001	164	180	2024-08-17 00:00:00
P001	S104	W003	140	184	2024-08-18 00:00:00

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:

```sql
SELECT
  supplier_key,
  Supplier_ID,
  Supplier_Name,
  `Lead_Time (days)` AS Lead_Time,
  `Order_Frequency`
FROM dim_suppliers
WHERE `Lead_Time (days)` > (SELECT AVG(`Lead_Time (days)`) FROM dim_suppliers)
ORDER BY `Lead_Time (days)` DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

supplier_key	Supplier_ID	Supplier_Name	Lead_Time	Order_Frequency
1	SUP001	Murray-Ramirez	9	Weekly
2	SUP002	Thomas, Meyer and Campbell	9	Weekly
7	SUP007	Meyers, Miller and Young	9	Weekly
10	SUP010	Wright PLC	9	Monthly
29	SUP029	Duncan, Davidson and Martin	9	Monthly
33	SUP033	Murphy-Wise	9	Weekly
34	SUP034	James, Mullen and Cooper	9	Weekly
35	SUP035	Smith PLC	9	Monthly
36	SUP036	Powers, Olson and Sanchez	9	Weekly
43	SUP043	Fox Group	9	Biweekly
46	SUP046	Harris, Patterson and Harris	9	Biweekly
49	SUP049	Obrien-Jones	9	Biweekly
4	SUP004	Evans, Brown and Turner	8	Weekly
8	SUP008	Edwards-Barnes	8	Weekly
16	SUP016	Johnson Group	8	Biweekly
37	SUP037	Kerr-Palmer	8	Weekly
38	SUP038	Johnson, Turner and Carpen...	8	Biweekly
50	SUP050	Sosa, Cain and Cummings	8	Weekly
13	SUP013	Norman, Welch and Foster	7	Biweekly
14	SUP014	Brown LLC	7	Biweekly
17	SUP017	Ayers Ltd	7	Monthly
45	SUP045	Castro-Townsend	7	Weekly

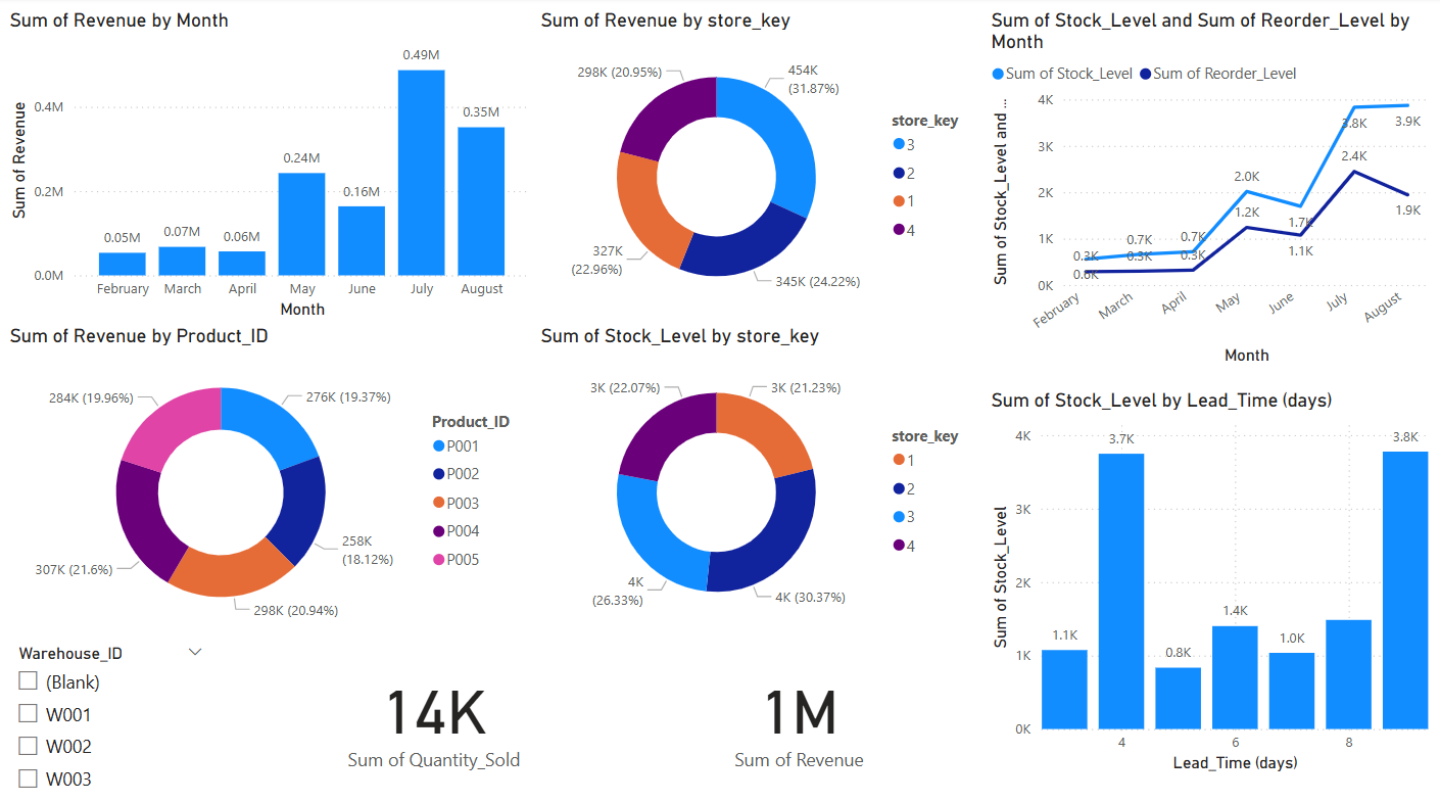
Suggest Alternative Suppliers:

```sql
SELECT
  dp.Product_ID,
  ds1.Supplier_ID AS High_LeadTime_Supplier,
  ds1.Supplier_Name AS High_LeadTime_Supplier_Name,
  ds1.`Lead_Time (days)` AS High_Lead_Time,
  ds2.Supplier_ID AS Alternative_Supplier,
  ds2.Supplier_Name AS Alternative_Supplier_Name,
  ds2.`Lead_Time (days)` AS Alternative_Lead_Time
FROM fact_purchase_orders fpo1
JOIN dim_products dp
  ON fpo1.product_key = dp.product_key
JOIN dim_suppliers ds1
  ON fpo1.supplier_key = ds1.supplier_key
JOIN fact_purchase_orders fpo2
  ON fpo1.product_key = fpo2.product_key
JOIN dim_suppliers ds2
  ON fpo2.supplier_key = ds2.supplier_key
WHERE ds1.`Lead_Time (days)` > ds2.`Lead_Time (days)`
ORDER BY dp.Product_ID, ds1.`Lead_Time (days)` DESC, ds2.`Lead_Time (days)` ASC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

Product_ID	High_LeadTime_Supplier	High_LeadTime_Supplier_Name	High_Lead_Time	Alternative_Supplier	Alternative_Supplier_Name	Alternative_Lead_Time
P001	SUP034	James, Mullen and Cooper	9	SUP039	Ortega-Manning	3
P001	SUP046	Harris, Patterson and Harris	9	SUP039	Ortega-Manning	3
P001	SUP036	Powers, Olson and Sanchez	9	SUP039	Ortega-Manning	3
P001	SUP046	Harris, Patterson and Harris	9	SUP039	Ortega-Manning	3
P001	SUP010	Wright PLC	9	SUP039	Ortega-Manning	3
P001	SUP010	Wright PLC	9	SUP039	Ortega-Manning	3
P001	SUP010	Wright PLC	9	SUP039	Ortega-Manning	3
P001	SUP046	Harris, Patterson and Harris	9	SUP039	Ortega-Manning	3
P001	SUP033	Murphy-Wise	9	SUP039	Ortega-Manning	3
P001	SUP029	Duncan, Davidson and Martin	9	SUP039	Ortega-Manning	3
P001	SUP033	Murphy-Wise	9	SUP039	Ortega-Manning	3
P001	SUP002	Thomas, Meyer and Campbell	9	SUP039	Ortega-Manning	3
P001	SUP010	Wright PLC	9	SUP039	Ortega-Manning	3
P001	SUP033	Murphy-Wise	9	SUP039	Ortega-Manning	3
P001	SUP010	Wright PLC	9	SUP039	Ortega-Manning	3
P001	SUP033	Murphy-Wise	9	SUP039	Ortega-Manning	3
P001	SUP049	Obrien-Jones	9	SUP039	Ortega-Manning	3
P001	SUP033	Murphy-Wise	9	SUP039	Ortega-Manning	3
P001	SUP010	Wright PLC	9	SUP039	Ortega-Manning	3
P001	SUP049	Obrien-Jones	9	SUP039	Ortega-Manning	3
P001	SUP033	Murphy-Wise	9	SUP039	Ortega-Manning	3

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.