Superstore-customer behaviour RFM analysis

Week 1: Data Engineering & Schema Design

Project Overview

This document presents the complete implementation of Week 1 tasks based on the Superstore dataset. It includes cleaning and standardization of data ,entity relationship diagram,schema design, SQL scripts etc.

1. DATA CLEANING & STANDARDIZATION

- Create cleaned staging view with standardized data,
- Standardize sales_id (remove whitespace, convert to uppercase),
- Date parsing with multiple format support,
- Standardize customer names (proper case, remove extra spaces),
- Standardize product names,
- Standardize category (proper case),
- Handle NULL/invalid quantities (default to 1),
- Handle NULL/invalid prices (use 0 or calculate from revenue),
- Handle NULL costs,
- Calculate/verify revenue,
- Standardize salesperson names,
- Standardize region (uppercase)

1. ERD (Entity Relationship Diagram)

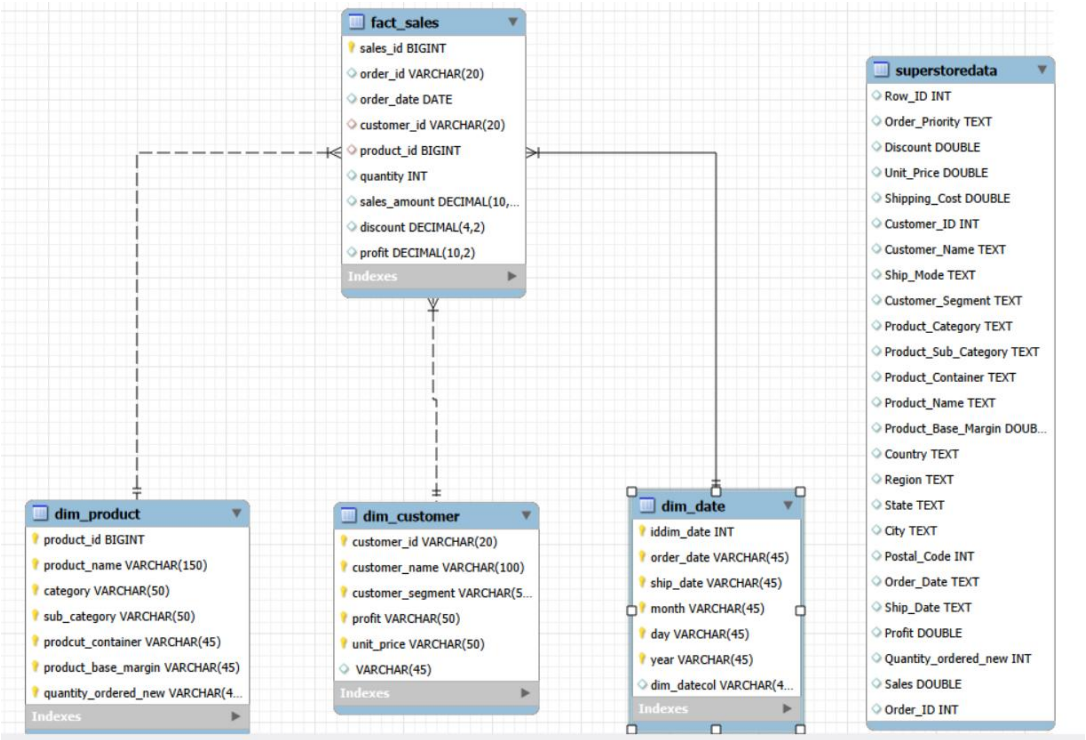
The data model follows a Star Schema design:

Fact Table:
• fact_sales (sales transactions)

Dimension Tables:
• dim_customer (customer details)
• dim_product (product details)

Relationships:
• fact_sales.customer_id → dim_customer.customer_id
• fact_sales.product_id → dim_product.product_id

- Fact_sales.order_date→dim_date.order_date



This structure enables fast analytical queries.

2. SQL Schema Creation (Week 1)

```
use thansimoldb;


-- Dimension: Customer
CREATE TABLE dim_customer (
   customer_id VARCHAR(20) PRIMARY KEY,
   customer_name VARCHAR(100) NOT NULL,
```

```
    region VARCHAR(50) NOT NULL,
    state VARCHAR(50) NOT NULL,
    city VARCHAR(50) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    INDEX idx_region (region),
    INDEX idx_state (state),
    INDEX idx_city (city)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- Dimension: Date (for time-based analysis)
CREATE TABLE dim_date (
    date_id INT UNSIGNED PRIMARY KEY,
    date_value DATE NOT NULL UNIQUE,
    year SMALLINT NOT NULL,
    quarter TINYINT NOT NULL,
    month TINYINT NOT NULL,
    month_name VARCHAR(10) NOT NULL,
    day TINYINT NOT NULL,
    day_of_week TINYINT NOT NULL,
    day_name VARCHAR(10) NOT NULL,
    week_of_year TINYINT NOT NULL,
    is_weekend BOOLEAN NOT NULL,
    is_holiday BOOLEAN DEFAULT FALSE,
    INDEX idx_year_month (year, month),
```

```
    INDEX idx_quarter (quarter)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- Dimension: Product
CREATE TABLE dim_product (
    product_id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    product_name VARCHAR(150) NOT NULL,
    category VARCHAR(50) NOT NULL,
    sub_category VARCHAR(50) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    UNIQUE KEY uk_product (product_name, category, sub_category),
    INDEX idx_category (category),
    INDEX idx_sub_category (sub_category)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- FACT_TABLE
CREATE TABLE fact_sales (
    sales_id BIGINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    order_id VARCHAR(20) NOT NULL,
    order_date_id INT UNSIGNED NOT NULL,
    customer_id VARCHAR(20) NOT NULL,
    product_id INT UNSIGNED NOT NULL,
    quantity INT NOT NULL DEFAULT 0,
    sales_amount DECIMAL(12,2) NOT NULL DEFAULT 0.00,
```

```
    discount DECIMAL(5,4) NOT NULL DEFAULT 0.0000,
    profit DECIMAL(12,2) NOT NULL DEFAULT 0.00,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);



CREATE TABLE fact_sales (
    sales_id INT AUTO_INCREMENT PRIMARY KEY,
    order_date_id INT NOT NULL,
    product_id INT NOT NULL,
    customer_id INT NOT NULL,
    quantity INT,
    sales DECIMAL(10,2),
    profit DECIMAL(10,2),

    CONSTRAINT fk_date
        FOREIGN KEY (order_date_id)
        REFERENCES dim_date(date_id),

    CONSTRAINT fk_product
        FOREIGN KEY (product_id)
        REFERENCES dim_product(product_id),

    CONSTRAINT fk_customer
        FOREIGN KEY (customer_id)
        REFERENCES dim_customer(customer_id)
```

```
)
ENGINE = InnoDB;

-- Clean and populate dim_customer
INSERT INTO dim_customer (customer_id, customer_name, region, state, city)
SELECT DISTINCT
   TRIM(customer_id) AS customer_id,
   TRIM(COALESCE(customer_name, 'Unknown')) AS customer_name,
   TRIM(COALESCE(region, 'Unknown')) AS region,
   TRIM(COALESCE(state, 'Unknown')) AS state,
   TRIM(COALESCE(city, 'Unknown')) AS city
FROM thansimoldb
WHERE customer_id IS NOT NULL
  AND TRIM(customer_id) != ''
ON DUPLICATE KEY UPDATE
   customer_name = VALUES(customer_name),
   region = VALUES(region),
   state = VALUES(state),
   city = VALUES(city);

-- Clean and populate dim_product
INSERT INTO dim_product (product_name, category, sub_category)
SELECT DISTINCT
   TRIM(COALESCE(Product_Name, 'Unknown')) AS product_name,
   TRIM(COALESCE(Product_Category, 'Unknown')) AS category,
   TRIM(COALESCE(Product_Sub_Category, 'Unknown')) AS sub_category
```

```
FROM superstoredata
WHERE Product_Name IS NOT NULL
  AND TRIM(Product_Name) != ''
ON DUPLICATE KEY UPDATE
    category = VALUES(category),
    sub_category = VALUES(sub_category);



-- Populate dim_date (generates dates for a 10-year range)
CREATE TABLE dim_date (
    date_id INT PRIMARY KEY,
    date_value DATE NOT NULL,
    year INT,
    quarter INT,
    month INT,
    month_name VARCHAR(20),
    day INT,
    day_of_week INT,
    day_name VARCHAR(20),
    week_of_year INT,
    is_weekend TINYINT(1)
);

DELIMITER $$
```

```
CREATE PROCEDURE populate_dim_date (
   IN p_start_date DATE,
   IN p_end_date DATE
)
BEGIN
   DECLARE v_date DATE;
   SET v_date = p_start_date;

   WHILE v_date <= p_end_date DO

      INSERT INTO dim_date (
         date_id,
         date_value,
         year,
         quarter,
         month,
         month_name,
         day,
         day_of_week,
         day_name,
         week_of_year,
         is_weekend
      )
      VALUES (
         DATE_FORMAT(v_date, '%Y%m%d'),
         v_date,
```

```
        YEAR(v_date),
        QUARTER(v_date),
        MONTH(v_date),
        MONTHNAME(v_date),
        DAY(v_date),
        DAYOFWEEK(v_date),
        DAYNAME(v_date),
        WEEKOFYEAR(v_date),
        IF(DAYOFWEEK(v_date) IN (1,7), 1, 0)
    );

    SET v_date = DATE_ADD(v_date, INTERVAL 1 DAY);

  END WHILE;
END$$

DELIMITER ;

CALL populate_dim_date('2022-01-01', '2022-12-31');

SELECT COUNT(*) FROM dim_date;
SELECT * FROM dim_date LIMIT 5;

-- Disable safe updates for bulk operations
INSERT INTO fact_sales (
  order_id,
```

```
    order_date_id,
    customer_id,
    product_id,
    quantity,
    sales_amount,
    discount,
    profit
)
SELECT
    TRIM(s.order_id) AS order_id,
    CAST(DATE_FORMAT(
        STR_TO_DATE(s.order_date, '%m/%d/%Y'),
        '%Y%m%d'
    ) AS UNSIGNED) AS order_date_id,
    TRIM(s.customer_id) AS customer_id,
    p.product_id,
    COALESCE(s.Quantity_Ordered_new, 0) AS quantity,
    COALESCE(s.sales, 0) AS sales_amount,
    COALESCE(s.discount, 0) AS discount,
    COALESCE(s.profit, 0) AS profit
FROM superstoredata AS s
INNER JOIN dim_product AS p
    ON TRIM(s.Product_Name) = p.product_name
   AND TRIM(COALESCE(s.Product_Category, 'Unknown')) = p.category
   AND TRIM(COALESCE(s.Product_Sub_Category, 'Unknown')) = p.sub_category
INNER JOIN dim_customer AS c
```

```
   ON TRIM(s.customer_id) = c.customer_id
WHERE s.order_id IS NOT NULL
  AND s.customer_id IS NOT NULL
  AND s.order_date IS NOT NULL
  AND s.sales > 0
  AND STR_TO_DATE(s.order_date, '%m/%d/%Y') IS NOT NULL;



SELECT 'Orphaned Customer Records' AS check_name, COUNT(*) AS count
FROM fact_sales fs
LEFT JOIN dim_customer dc ON fs.customer_id = dc.customer_id
WHERE dc.customer_id IS NULL

UNION ALL

SELECT 'Orphaned Product Records', COUNT(*)
FROM fact_sales fs
LEFT JOIN dim_product dp ON fs.product_id = dp.product_id
WHERE dp.product_id IS NULL

UNION ALL

SELECT 'Orphaned Date Records', COUNT(*)
FROM fact_sales fs
LEFT JOIN dim_date dd ON fs.order_date_id = dd.date_id
WHERE dd.date_id IS NULL
```

```
UNION ALL

SELECT 'Negative Sales', COUNT(*)
FROM fact_sales
WHERE sales_amount < 0

UNION ALL

SELECT 'Negative Quantity', COUNT(*)
FROM fact_sales
WHERE quantity < 0;

-- Query 1: Monthly Sales Performance (< 2 seconds)
SELECT
    dd.year,
    dd.month_name,
    dc.region,
    COUNT(DISTINCT fs.order_id) AS total_orders,
    SUM(fs.quantity) AS total_quantity,
    SUM(fs.sales_amount) AS total_sales,
    SUM(fs.profit) AS total_profit,
    AVG(fs.sales_amount) AS avg_order_value
FROM fact_sales fs
INNER JOIN dim_date dd ON fs.order_date_id = dd.date_id
INNER JOIN dim_customer dc ON fs.customer_id = dc.customer_id
```

```
GROUP BY dd.year, dd.month, dd.month_name, dc.region
ORDER BY dd.year DESC, dd.month DESC, total_sales DESC;

-- Query 2: Top Products by Category (< 2 seconds)
SELECT
    dp.category,
    dp.sub_category,
    dp.product_name,
    COUNT(DISTINCT fs.order_id) AS order_count,
    SUM(fs.quantity) AS units_sold,
    SUM(fs.sales_amount) AS revenue,
    SUM(fs.profit) AS profit,
    ROUND(SUM(fs.profit) / SUM(fs.sales_amount) * 100, 2) AS profit_margin_pct
FROM fact_sales fs
INNER JOIN dim_product dp ON fs.product_id = dp.product_id
GROUP BY dp.category, dp.sub_category, dp.product_name
HAVING revenue > 1000
ORDER BY revenue DESC
LIMIT 50

-- Query 3: Customer Segmentation (< 2 seconds)
SELECT
dc.customer_id,
    dc.customer_name,
    dc.region,
    dc.state,
```

```
    COUNT(DISTINCT fs.order_id) AS total_orders,
    SUM(fs.sales_amount) AS lifetime_value,
    AVG(fs.sales_amount) AS avg_order_value,
    MAX(dd.date_value) AS last_order_date,
    CASE
        WHEN SUM(fs.sales_amount) > 10000 THEN 'VIP'
        WHEN SUM(fs.sales_amount) > 5000 THEN 'High Value'
        WHEN SUM(fs.sales_amount) > 1000 THEN 'Medium Value'
        ELSE 'Low Value'
    END AS customer_segment
FROM fact_sales fs
INNER JOIN dim_customer dc ON fs.customer_id = dc.customer_id
INNER JOIN dim_date dd ON fs.order_date_id = dd.date_id
GROUP BY dc.customer_id, dc.customer_name, dc.region, dc.state
ORDER BY lifetime_value DESC;

-- Query 4: Year-over-Year Comparison (< 2 seconds)
SELECT
    dd.year,
    dd.quarter,
    dp.category,
    SUM(fs.sales_amount) AS sales,
    SUM(fs.profit) AS profit,
    COUNT(DISTINCT fs.customer_id) AS unique_customers
FROM fact_sales fs
INNER JOIN dim_date dd ON fs.order_date_id = dd.date_id
```

```
INNER JOIN dim_product dp ON fs.product_id = dp.product_id

GROUP BY dd.year, dd.quarter, dp.category

ORDER BY dd.year, dd.quarter, sales DESC;


ANALYZE TABLE dim_customer;

ANALYZE TABLE dim_product;

ANALYZE TABLE dim_date;

ANALYZE TABLE fact_sales;
```

Week 2: Python Logic & Analytics

Project Overview

This document presents the complete implementation of Week 2 tasks based on the Superstore dataset. It includes schema design, SQL scripts, Python logic, RFM analysis, and Market Basket Analysis.

4. Python → MySQL Connection

```
import pandas as pd
from sqlalchemy import create_engine

engine = create_engine(
   "mysql+pymysql://root:password@localhost:3306/dakshitadb"
)

query = '''
SELECT order_id, order_date, customer_id, sales_amount, quantity
FROM fact_sales;
'''
```

```
df = pd.read_sql(query, engine)
```

5. RFM Calculation Logic

```
import datetime as dt

snapshot_date = df['order_date'].max() + dt.timedelta(days=1)

rfm = df.groupby('customer_id').agg({
   'order_date': lambda x: (snapshot_date - x.max()).days,
   'order_id': 'nunique',
   'sales_amount': 'sum'
}).reset_index()

rfm.columns = ['customer_id', 'Recency', 'Frequency', 'Monetary']
```

6. RFM Scoring & Segmentation

```
rfm['R_score'] = pd.qcut(rfm['Recency'], 5, labels=[5,4,3,2,1])
rfm['F_score'] = pd.qcut(rfm['Frequency'].rank(method='first'), 5, labels=[1,2,3,4,5])
rfm['M_score'] = pd.qcut(rfm['Monetary'], 5, labels=[1,2,3,4,5])

rfm['RFM_Score'] = (
   rfm['R_score'].astype(int) +
   rfm['F_score'].astype(int) +
   rfm['M_score'].astype(int)
)

def segment_customer(row):
   if row['RFM_Score'] >= 13:
      return 'Champion'
   elif row['RFM_Score'] >= 10:
      return 'Loyal'
   elif row['RFM_Score'] >= 7:
      return 'Potential'
   else:
      return 'Hibernating'

rfm['Segment'] = rfm.apply(segment_customer, axis=1)
```

7. Validation Check

```
rfm.groupby('Segment')['Monetary'].mean().sort_values(ascending=False)
```

8. Market Basket Analysis (Week 2)

```
from mlxtend.frequent_patterns import apriori, association_rules

basket = df.groupby(['order_id', 'product_id'])['quantity'].sum().unstack().fillna(0)
basket = basket.applymap(lambda x: 1 if x > 0 else 0)

frequent_items = apriori(basket, min_support=0.01, use_colnames=True)

rules = association_rules(frequent_items, metric='lift', min_threshold=1)
```

Conclusion

Week 1 focused on data modeling and SQL optimization, while Week 2 implemented business logic using Python. RFM segmentation successfully identified high-value customers (Champions), and Market Basket Analysis laid the foundation for cross-sell strategies.

Week 3 : DASHBOARD CONSTRUCTION

1. Create Star Schema:

The data model follows a Star Schema design:

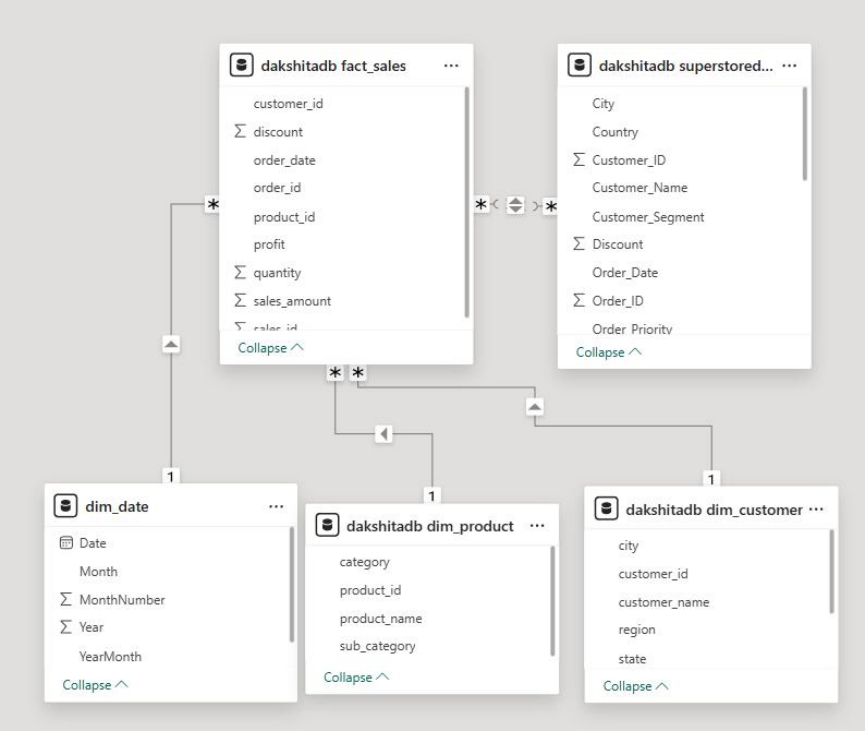
Fact Table: fact_sales
Dim_Customer (CustomerID)

Dim_Product (ProductID)

Dim_Date (Date)

Relationships:

- fact_sales.customer_id → dim_customer.customer_id
- fact_sales.product_id → dim_product.product_id
- fact_sales.date → dim_date.date



2. Build the Interactive RFM Matrix Visual

Step 1: Create RFM Measures:

dax

```
frequency = DISTINCTCOUNT('dakshitadb fact_sales'[order_id])

monetary = SUM('dakshitadb fact_sales'[sales_amount])

recency = DATEDIFF(MAX('dakshitadb fact_sales'[order_date]),TODAY(),DAY)
```

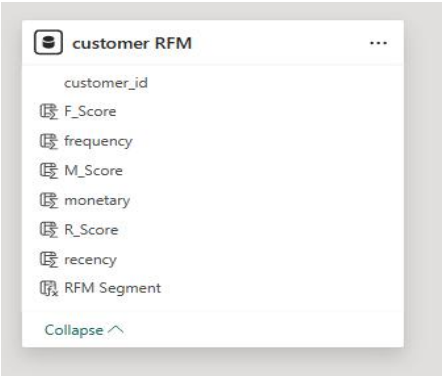
Step 2: Create RFM Scores (1–5)

```
R_Score =

SWITCH(TRUE(),

    [Recency] <= 30, 5,

    [Recency] <= 60, 4,

    [Recency] <= 90, 3,
```

```
    [Recency] <= 180, 2,
    1
)
F_Score =
SWITCH(TRUE(),
    [frequency] <= 30, 5,
    [frequency] <= 60, 4,
    [frequency] <= 90, 3,
    [frequency] <= 180, 2,
    1
)
M_Score =
SWITCH(TRUE(),
    [monetary]<= 30, 5,
    [monetary] <= 60, 4,
    [monetary] <= 90, 3,
    [monetary] <= 180, 2,
    1
)
```

Step 3: RFM Segment

```
RFM Segment = IF([R_Score]>=3 && [F_Score]>=3 &&
[M_score]>=2,"champions", IF([R_Score]>2,"loyal customer","at risk"))
```



frequency	recency	monetary	customer_id	R_Score	F_Score	M_Score	RFM Segment
1320	3844	1864109.43	711	1	1	1	At Risk
1320	3844	1864109.43	699	1	1	1	At Risk
1320	3844	1864109.43	3	1	1	1	At Risk
1320	3844	1864109.43	2571	1	1	1	At Risk
1320	3844	1864109.43	2570	1	1	1	At Risk
1320	3844	1864109.43	1595	1	1	1	At Risk
1320	3844	1864109.43	5	1	1	1	At Risk
1320	3844	1864109.43	2308	1	1	1	At Risk
1320	3844	1864109.43	11	1	1	1	At Risk
1320	3844	1864109.43	2355	1	1	1	At Risk
1320	3844	1864109.43	14	1	1	1	At Risk
1320	3844	1864109.43	2578	1	1	1	At Risk
1320	3844	1864109.43	1933	1	1	1	At Risk
1320	3844	1864109.43	1998	1	1	1	At Risk
1320	3844	1864109.43	15	1	1	1	At Risk
1320	3844	1864109.43	3191	1	1	1	At Risk
1320	3844	1864109.43	272	1	1	1	At Risk
1320	3844	1864109.43	269	1	1	1	At Risk
1320	3844	1864109.43	16	1	1	1	At Risk
1320	3844	1864109.43	21	1	1	1	At Risk
1320	3844	1864109.43	18	1	1	1	At Risk
1320	3844	1864109.43	19	1	1	1	At Risk
1320	3844	1864109.43	2912	1	1	1	At Risk
1320	3844	1864109.43	1781	1	1	1	At Risk
1320	3844	1864109.43	1672	1	1	1	At Risk
1320	3844	1864109.43	24	1	1	1	At Risk
1320	3844	1864109.43	2498	1	1	1	At Risk

3. UX Review – Client-Ready Dashboard Checklist

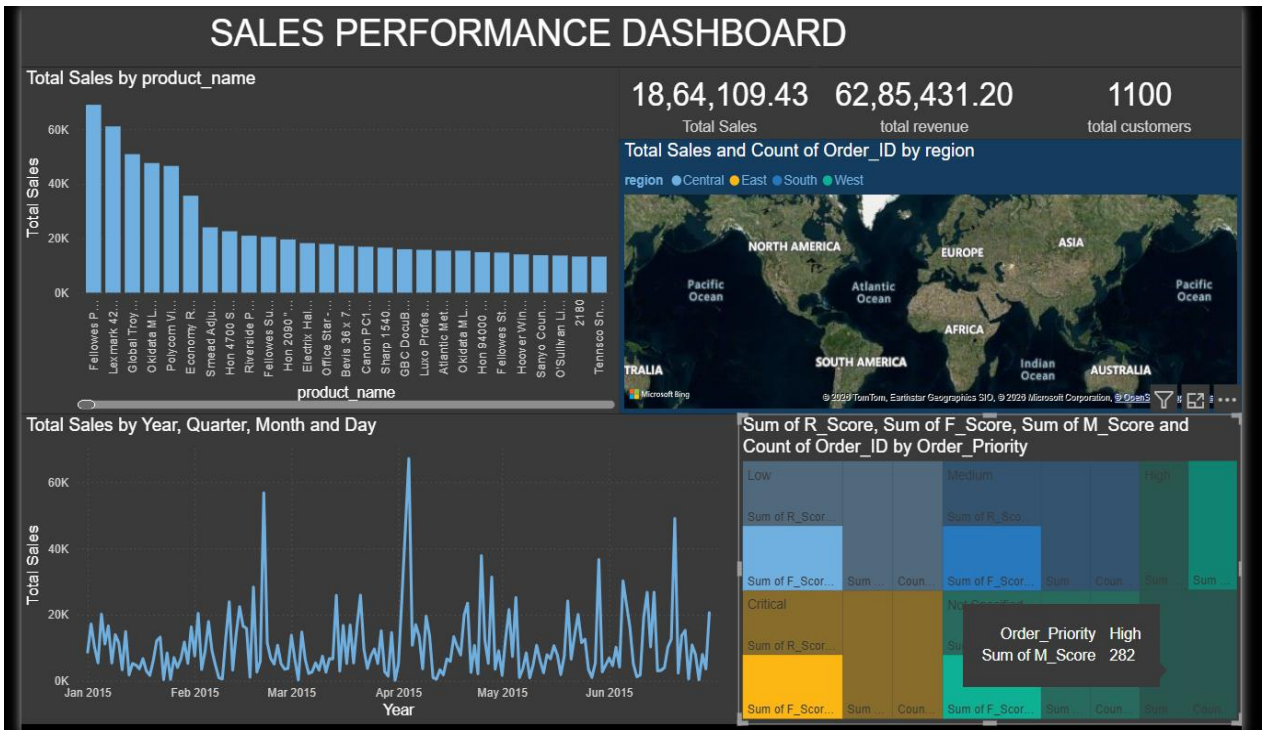
This dashboard delivers:

Executive KPIs (Sales, Growth)

Customer Intelligence (RFM Segments)

Secure Regional Views (RLS)

Clean, professional UX



Week 4: AUTOMATION AND HANDOFF

Description:

Schedule the entire end-to-end Python script to run automatically (e.g., every Sunday night)

using cron or Windows Task Scheduler. Final Presentation Deck, clearly outlining key actionable insights (e.g., "Region X is experiencing a 15% customer churn increase").

Full automation test:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

df = pd.read_csv('SuperStoreUS_dataset.csv',encoding='latin1')

df.head()
```

	Row ID	Order Priority	Discount	Unit Price	Shipping Cost	Customer ID	Customer Name	Ship Mode	Customer Segment	Product Category	...	Regior
0	20847	High	0.01	2.84	0.93	3	Bonnie Potter	Express Air	Corporate	Office Supplies	...	Wes
1	20228	Not Specified	0.02	500.98	26.00	5	Ronnie Proctor	Delivery Truck	Home Office	Furniture	...	Wes
2	21776	Critical	0.06	9.48	7.29	11	Marcus Dunlap	Regular Air	Home Office	Furniture	...	Eas
3	24844	Medium	0.09	78.69	19.99	14	Gwendolyn F Tyson	Regular Air	Small Business	Furniture	...	Centra
4	24846	Medium	0.08	3.28	2.31	14	Gwendolyn F Tyson	Regular Air	Small Business	Office Supplies	...	Centra

5 rows × 25 columns

features

df.dtypes

df.shape

df.head()

df.tail()

df.describe()

	Row ID	Discount	Unit Price	Shipping Cost	Customer ID	Product Base Margin	Postal Code	Pr
count	1952.000000	1952.000000	1952.000000	1952.000000	1952.000000	1936.000000	1952.000000	1952.000
mean	19916.479508	0.048975	109.079221	12.968151	1735.376537	0.515186	51534.769467	114.793
std	5957.595627	0.031378	393.481301	17.414631	991.078006	0.137055	29362.828420	1141.112
min	64.000000	0.000000	1.140000	0.490000	3.000000	0.350000	1001.000000	-16476.838
25%	19121.000000	0.020000	6.480000	3.230000	875.000000	0.380000	28560.000000	-84.485
50%	21164.500000	0.050000	20.990000	6.150000	1738.000000	0.525000	48765.500000	1.476
75%	23483.250000	0.080000	100.972500	14.362500	2578.250000	0.590000	78550.000000	116.201
max	26389.000000	0.210000	6783.020000	164.730000	3403.000000	0.850000	99362.000000	9228.225

```
18]: df.columns
```

```
18]: Index(['Row ID', 'Order Priority', 'Discount', 'Unit Price', 'Shipping Cost',
       'Customer ID', 'Customer Name', 'Ship Mode', 'Customer Segment',
       'Product Category', 'Product Sub-Category', 'Product Container',
       'Product Name', 'Product Base Margin', 'Country', 'Region',
       'State or Province', 'City', 'Postal Code', 'Order Date', 'Ship Date',
       'Profit', 'Quantity ordered new', 'Sales', 'Order ID'],
      dtype='object')
```

```
20]: df.columns = (df.columns.str.strip().str.lower().str.replace(' ','_'))
```

```
21]: df.columns
```

```
21]: Index(['row_id', 'order_priority', 'discount', 'unit_price', 'shipping_cost',
       'customer_id', 'customer_name', 'ship_mode', 'customer_segment',
       'product_category', 'product_sub-category', 'product_container',
       'product_name', 'product_base_margin', 'country', 'region',
       'state_or_province', 'city', 'postal_code', 'order_date', 'ship_date',
       'profit', 'quantity_ordered_new', 'sales', 'order_id'],
      dtype='object')
```

```python
# Regional aggregation (NO indentation at start)

regional_stats = (
    df.groupby('region')
      .agg({
          'sales': 'sum',
          'profit': 'sum',
          'order_id': 'nunique',
          'customer_id': 'nunique',
      })
      .round(2)
)

# Calculate KPIs
regional_stats['profit_margin'] = (
    regional_stats['profit'] / regional_stats['sales'] * 100
).round(2)

regional_stats['avg_order_value'] = (
    regional_stats['sales'] / regional_stats['order_id']
).round(2)

# Rename columns AFTER calculations
regional_stats.columns = [
    'Sales',
    'Profit',
    'Orders',
    'Customers',
    'Profit Margin (%)',
```

```python
# Sort by Sales
regional_stats = regional_stats.sort_values('Sales', ascending=False)

print(regional_stats)
```

```
              Sales    Profit  Orders  Customers  Profit Margin (%)  \
region
East     592171.49  85291.40     323        305              14.40
West     526776.57  75844.79     336        250              14.40
Central  448284.70  77365.47     400        323              17.26
South    357105.12 -14424.05     326        252              -4.04

         Avg Order Value
region
East             1833.35
West             1567.79
Central          1120.71
South            1095.41
```

```python
# Convert order_date column to datetime
df['order_date'] = pd.to_datetime(df['order_date'], errors='coerce')

# Verify
df[['order_date']].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1952 entries, 0 to 1951
Data columns (total 1 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   order_date  791 non-null    datetime64[ns]
dtypes: datetime64[ns](1)
memory usage: 15.4 KB
```

Convert order_date column to datetime

df['order_date'] = pd.to_datetime(df['order_date'], errors='coerce')

Verify

df[['order_date']].info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1952 entries, 0 to 1951
Data columns (total 1 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   order_date  791 non-null    datetime64[ns]
dtypes: datetime64[ns](1)
memory usage: 15.4 KB
```

```python
from datetime import timedelta

# Ensure order_date is datetime
df['order_date'] = pd.to_datetime(df['order_date'])

recent_date = df['order_date'].max()
cutoff_date = recent_date - timedelta(days=90)

active_customers = df[df['order_date'] >= cutoff_date]['customer_id'].unique()
all_customers = df['customer_id'].unique()
churned_customers = set(all_customers) - set(active_customers)

insights = []  # initialize list

for region in df['region'].unique():
    region_customers = df[df['region'] == region]['customer_id'].unique()
    region_churned = len(set(region_customers) & churned_customers)

    churn_rate = (
        (region_churned / len(region_customers)) * 100
        if len(region_customers) > 0 else 0
    )

    if churn_rate > 12:
        insights.append({
            'Priority': 'CRITICAL',
            'Category': 'Customer Retention',
            'Finding': f'{region} region has {churn_rate:.1f}% customer churn',
            'Impact': f'{region_churned} customers at risk',
            'Action': f'Launch retention campaign in {region} immediately'
        })
```

```python
pd.DataFrame(insights)
```

	Priority	Category	Finding	Impact	Action
0	CRITICAL	Customer Retention	West region has 78.4% customer churn	196 customers at risk	Launch retention campaign in West immediately
1	CRITICAL	Customer Retention	East region has 72.8% customer churn	222 customers at risk	Launch retention campaign in East immediately
2	CRITICAL	Customer Retention	Central region has 75.5% customer churn	244 customers at risk	Launch retention campaign in Central immediately
3	CRITICAL	Customer Retention	South region has 74.2% customer churn	187 customers at risk	Launch retention campaign in South immediately

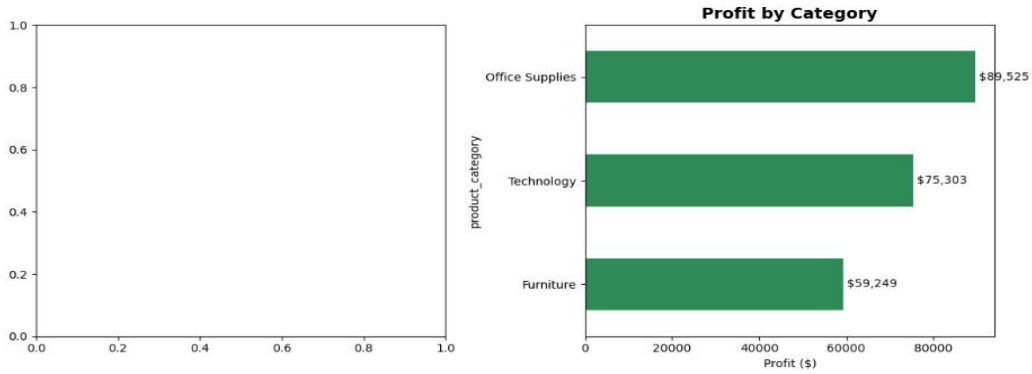
```
# Category performance analysis

category_stats = df.groupby('product_category').agg(
    sales=('sales', 'sum'),
    profit=('profit', 'sum')
)

# Profit margin %
category_stats['margin'] = (category_stats['profit'] / category_stats['sales']) * 100
category_stats = category_stats.round(2)

# Best & worst categories
best_category = category_stats['profit'].idxmax()
worst_margin = category_stats['margin'].idxmin()

# Best category insight
insights.append({
    'Priority': 'OPPORTUNITY',
    'Category': 'Product Strategy',
    'Finding': f'{best_category} category showing strongest performance',
    'Impact': f'{category_stats.loc[best_category, "profit"]:.2f} profit generated',
    'Action': f'Increase {best_category} inventory by 30%'
})
```

```
# Worst margin warning
if category_stats.loc[worst_margin, 'margin'] < 5:
    insights.append({
        'Priority': 'WARNING',
        'Category': 'Profitability',
        'Finding': (
            f'{worst_margin} category has only '
            f'{category_stats.loc[worst_margin, "margin"]:.2f}% margin'
        ),
        'Impact': 'Low profitability affecting overall performance',
        'Action': f'Review pricing and supplier costs for {worst_margin}'
    })

category_stats
```

product_category	sales	profit	margin
Furniture	660704.31	59249.45	8.97
Office Supplies	551368.62	89525.01	16.24
Technology	712264.95	75303.16	10.57

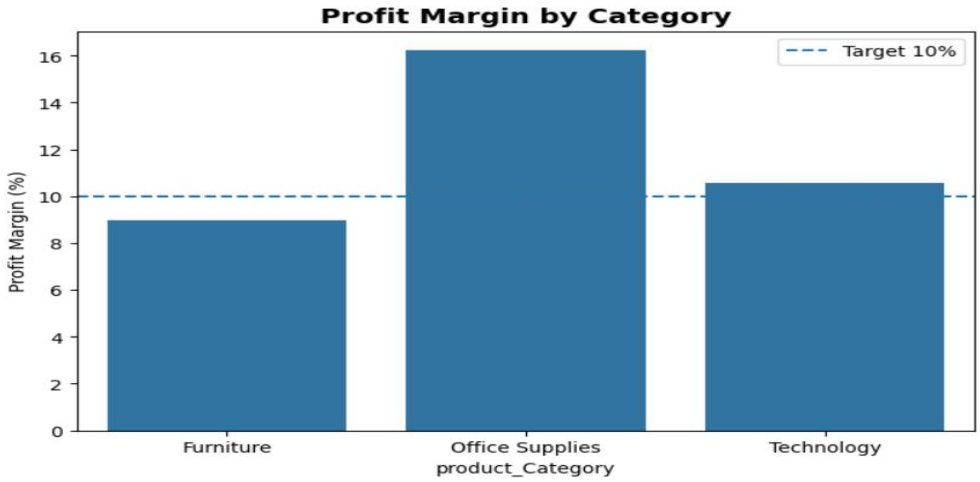
```
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Category Profit

category_profit = (

  df.groupby('product_category')['profit']

  .sum()

  .sort_values()

)
```

```
category_profit.plot(kind='barh', ax=axes[1], color='seagreen')
axes[1].set_title('Profit by Category', fontsize=14, fontweight='bold')
axes[1].set_xlabel('Profit ($)')
for i, v in enumerate(category_profit.values):
    axes[1].text(v, i, f' ${v:,.0f}', va='center')
plt.tight_layout()
plt.show()
```




```
category_stats = df.groupby('product_category').agg(
    sales=('sales', 'sum'),
    profit=('profit', 'sum')
).reset_index()
category_stats['margin_pct'] = (
    category_stats['profit'] / category_stats['sales'] * 100
)
plt.figure(figsize=(8,5))
sns.barplot(data=category_stats, x='product_category', y='margin_pct')
plt.axhline(10, linestyle='--', label='Target 10%')
plt.title('Profit Margin by Category', fontsize=14, fontweight='bold')
plt.ylabel('Profit Margin (%)')
plt.xlabel('product_Category')
plt.legend()
```

plt.show()

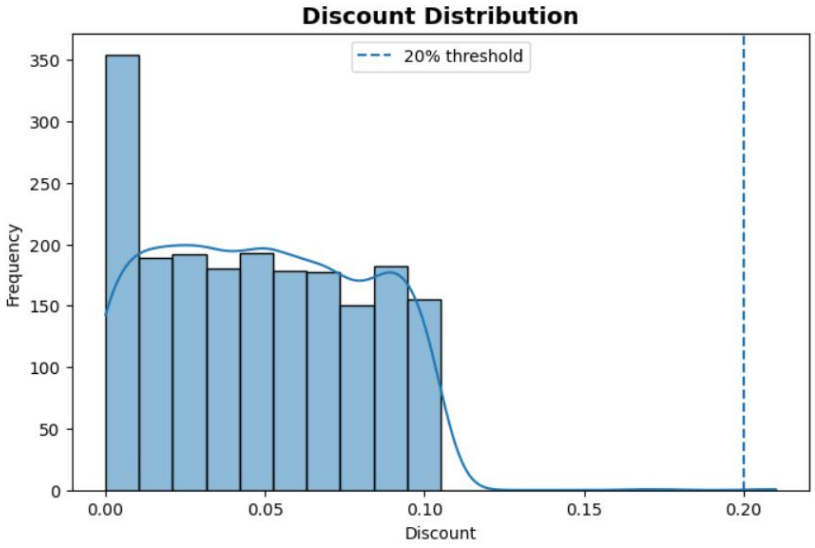



```
# Discount impact analysis

high_discount = df[df['discount'] > 0.2]

if len(high_discount) > 0:
    high_discount_profit = high_discount['profit'].sum()
    total_profit = df['profit'].sum()

    impact_pct = (abs(high_discount_profit) / total_profit) * 100

    insights.append({
        'Priority': 'WARNING',
        'Category': 'Discount Policy',
        'Finding': f'Discounts >20% impacting profit by {impact_pct:.1f}%',
        'Impact': f'{abs(high_discount_profit):.2f} profit loss',
        'Action': 'Cap discounts at 15% for low-margin products'
    })
```

```
pd.DataFrame(insights)
```

	Priority	Category	Finding	Impact	Action
0	CRITICAL	Customer Retention	West region has 78.4% customer churn	196 customers at risk	Launch retention campaign in West immediately
1	CRITICAL	Customer Retention	East region has 72.8% customer churn	222 customers at risk	Launch retention campaign in East immediately
2	CRITICAL	Customer Retention	Central region has 75.5% customer churn	244 customers at risk	Launch retention campaign in Central immediately
3	CRITICAL	Customer Retention	South region has 74.2% customer churn	187 customers at risk	Launch retention campaign in South immediately
4	OPPORTUNITY	Product Strategy	Office Supplies category showing strongest per...	89525.01 profit generated	Increase Office Supplies inventory by 30%
5	WARNING	Discount Policy	Discounts >20% impacting profit by 0.0%	17.75 profit loss	Cap discounts at 15% for low-margin products
6	OPPORTUNITY	Seasonal Planning	April shows 37% higher sales than average	179495.39 revenue potential	Increase inventory 4–6 weeks before April

```
# discount distribution
plt.figure(figsize=(8,5))
sns.histplot(df['discount'], bins=20, kde=True)
plt.axvline(0.2, linestyle='--', label='20% threshold')
plt.title('Discount Distribution', fontsize=14, fontweight='bold')
plt.xlabel('Discount')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



```python
# Highlight issues
print("⚠️ DISCOUNT INSIGHTS:")
print("-" * 60)

high_discount = df[df['discount'] > 0.2]

if len(high_discount) > 0:
    high_discount_profit = high_discount['profit'].sum()
    high_discount_sales = high_discount['sales'].sum()

    print(f"• Orders with >20% discount: {len(high_discount)}")
    print(f"• Total profit from high discounts: ${high_discount_profit:,.2f}")

    if high_discount_sales != 0:
        profit_margin = (high_discount_profit / high_discount_sales) * 100
        print(f"• Profit margin at >20% discount: {profit_margin:.2f}%")

    if high_discount_profit < 0:
        print("🚨 WARNING: High discounts resulting in NEGATIVE profits!")

print()
```

```
⚠️ DISCOUNT INSIGHTS:
------------------------------------------------------------
• Orders with >20% discount: 1
• Total profit from high discounts: $-17.75
• Profit margin at >20% discount: -11.54%
🚨 WARNING: High discounts resulting in NEGATIVE profits!
```

```python
   #create customer stats
customer_stats = df.groupby('customer_id').agg({
            'sales': 'sum',
            'profit': 'sum',
            'order_id': 'nunique',
            'order_date': 'max'
        }).reset_index()
# rename columns
customer_stats = customer_stats.rename(columns={'sales':'Total Sales',
                                'profit':'Total Profit', 'order_id':'Order Count', 'order_date':'Last
                                            })
    # Segment customers by value
customer_stats['Segment'] = pd.cut(
    customer_stats['Total Sales'],
    bins=[0, 500, 2000, 10000, float('inf')],
    labels=['Low Value', 'Medium Value', 'High Value', 'VIP']
)

# Segment-level summary
segment_summary = customer_stats.groupby('Segment', observed=True).agg({
    'Total Sales': 'sum',
    'Total Profit': 'sum',
    'Order Count': 'sum'
}).round(2)

# Customer count per segment
segment_summary['Customer Count'] = (
    customer_stats.groupby('Segment', observed=True).size()
)
```

```python
# Average customer value
segment_summary['Avg Customer Value'] = (
    segment_summary['Total Sales'] / segment_summary['Customer Count']
).round(2)

print(segment_summary)
```

```
              Total Sales  Total Profit  Order Count  Customer Count  \
Segment
Low Value       102129.87     -32383.56          687             602
Medium Value    315162.11      21927.29          391             291
High Value      862719.24     152285.79          314             197
VIP             644326.66      82248.09           93              40

              Avg Customer Value
Segment
Low Value                 169.65
Medium Value             1083.03
High Value               4379.29
VIP                     16108.17
```

```python
# Seasonal patterns analysis

monthly_sales = df.groupby(df['order_date'].dt.month)['sales'].sum()

peak_month = monthly_sales.idxmax()
avg_sales = monthly_sales.mean()
peak_sales = monthly_sales.loc[peak_month]

if peak_sales > avg_sales * 1.3:
    month_name = datetime(2000, int(peak_month), 1).strftime('%B')
    growth = ((peak_sales / avg_sales) - 1) * 100

    insights.append({
        'Priority': 'OPPORTUNITY',
        'Category': 'Seasonal Planning',
        'Finding': f'{month_name} shows {growth:.0f}% higher sales than average',
        'Impact': f'{peak_sales:.2f} revenue potential',
        'Action': f'Increase inventory 4-6 weeks before {month_name}'
    })
```

```python
monthly_sales
```

```
order_date
1.0    145694.55
2.0    114600.97
3.0     78898.24
4.0    179495.39
5.0    115201.98
6.0    149575.47
Name: sales, dtype: float64
```

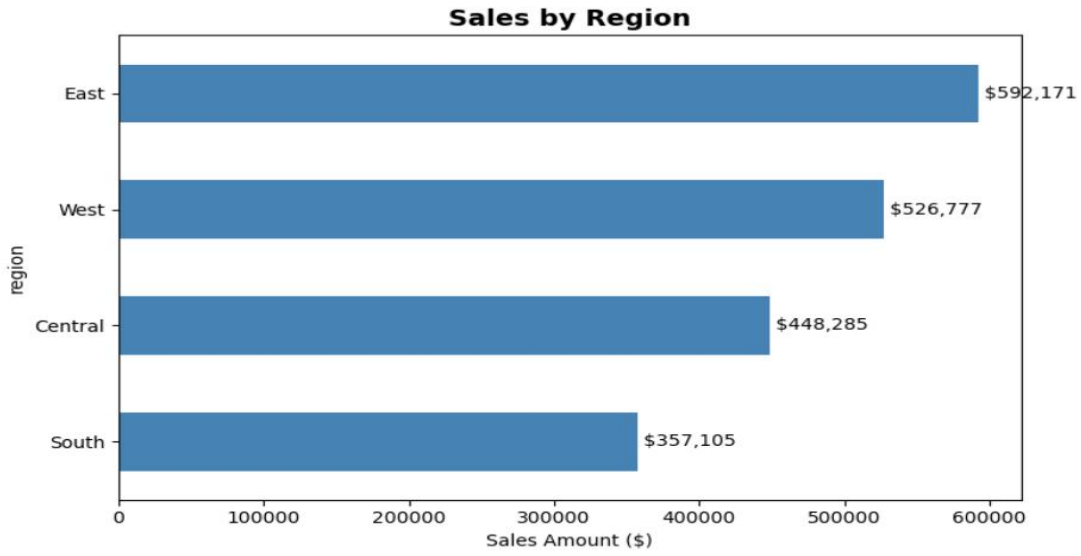
```python
# MONTHLY SALES TRENDS

# Ensure order_date is datetime
df['order_date'] = pd.to_datetime(df['order_date'])

# Create time columns
df['month'] = df['order_date'].dt.month
df['month_name'] = df['order_date'].dt.strftime('%B')
df['year'] = df['order_date'].dt.year

# Monthly aggregation
monthly_stats = (
    df.groupby(['year', 'month', 'month_name'])
      .agg({
          'sales': 'sum',
          'profit': 'sum',
          'order_id': 'nunique'
      })
      .round(2)
)

# Rename columns
monthly_stats.columns = ['Sales', 'Profit', 'Orders']

# Profit margin
monthly_stats['Profit Margin %'] = (
    monthly_stats['Profit'] / monthly_stats['Sales'] * 100
).round(2)
```

```python
# Show last 12 months
print(monthly_stats.tail(12))
```

```
                          Sales    Profit  Orders  Profit Margin %
year   month month_name
2015.0 1.0   January     145694.55  -9670.96     109            -6.64
       2.0   February    114600.97   2945.27     101             2.57
       3.0   March        78898.24   3419.58      73             4.33
       4.0   April       179495.39  28674.10     101            15.97
       5.0   May         115201.98  20705.32      79            17.97
       6.0   June        149575.47  27449.67      95            18.35
```

```
import matplotlib.pyplot as plt

# 1. Regional Sales

regional = df.groupby('region')['sales'].sum().sort_values()

fig, axes = plt.subplots(1, 1, figsize=(8, 5))

regional.plot(kind='barh', ax=axes, color='steelblue')

axes.set_title('Sales by Region', fontsize=14, fontweight='bold')

axes.set_xlabel('Sales Amount ($)')

# Add value labels

for i, v in enumerate(regional.values):

   axes.text(v, i, f' ${v:,.0f}', va='center')

plt.tight_layout()
```

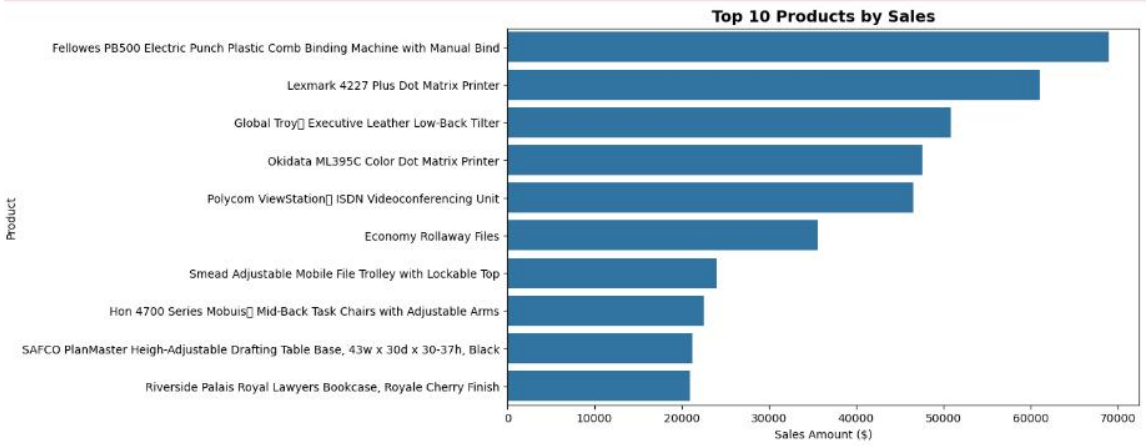
plt.show()




```
# TOP PRODUCTS ANALYSIS

product_stats = (
    df.groupby(['product_name', 'product_category'])
      .agg({
          'sales': 'sum',
          'profit': 'sum',
          'quantity_ordered_new': 'sum'
      })
      .round(2)
)

# TOP 5 PRODUCTS BY SALES
print("-" * 80)
top_sales = product_stats.sort_values('sales', ascending=False).head(5)
print("TOP 5 PRODUCTS BY SALES")
print(top_sales)

print("\n" + "-" * 80)

# TOP 5 PRODUCTS BY PROFIT
print("-" * 80)
top_profit = product_stats.sort_values('profit', ascending=False).head(5)
print("TOP 5 PRODUCTS BY PROFIT")
print(top_profit)
print("-" * 80)
```

```
TOP 5 PRODUCTS BY SALES
                                                                    sales  \
product_name                                product_category
Fellowes PB500 Electric Punch Plastic Comb Bind... Office Supplies   69013.48
Lexmark 4227 Plus Dot Matrix Printer               Technology        61071.73
Global Troy® Executive Leather Low-Back Tilter     Furniture         50871.21
Okidata ML395C Color Dot Matrix Printer            Technology        47611.09
Polycom ViewStation® ISDN Videoconferencing Unit   Technology        46538.69

                                                                     profit  \
product_name                                product_category
Fellowes PB500 Electric Punch Plastic Comb Bind... Office Supplies   23865.90
Lexmark 4227 Plus Dot Matrix Printer               Technology         3700.37
Global Troy® Executive Leather Low-Back Tilter     Furniture         18628.15
Okidata ML395C Color Dot Matrix Printer            Technology         7709.31
Polycom ViewStation® ISDN Videoconferencing Unit   Technology       -27621.25

                                                                    quantity_ordered_new
product_name                                product_category
Fellowes PB500 Electric Punch Plastic Comb Bind... Office Supplies                    54
Lexmark 4227 Plus Dot Matrix Printer               Technology                         35
Global Troy® Executive Leather Low-Back Tilter     Furniture                          96
Okidata ML395C Color Dot Matrix Printer            Technology                         34
Polycom ViewStation® ISDN Videoconferencing Unit   Technology                          7

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
TOP 5 PRODUCTS BY PROFIT
                                                                    sales  \
product name                                product category
```

```
top_products = (
  df.groupby('product_name', as_index=False)['sales']
    .sum()
    .sort_values('sales', ascending=False)
    .head(10)
)
plt.figure(figsize=(10,6))
sns.barplot(data=top_products, x='sales', y='product_name')
plt.title('Top 10 Products by Sales', fontsize=14, fontweight='bold')
plt.xlabel('Sales Amount ($)')
plt.ylabel('Product')
plt.show()
```



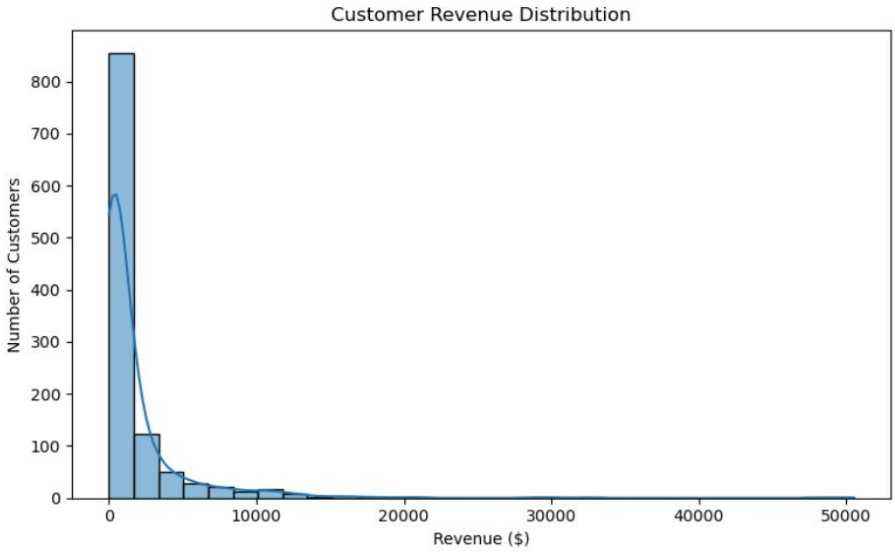

```
print("TOP 10 CUSTOMERS BY REVENUE:")
print("-" * 60)

top_customers = customer_stats.nlargest(
    10, 'Total Sales'
)[['Total Sales', 'Total Profit', 'Order Count']]

print(top_customers)
print()
```

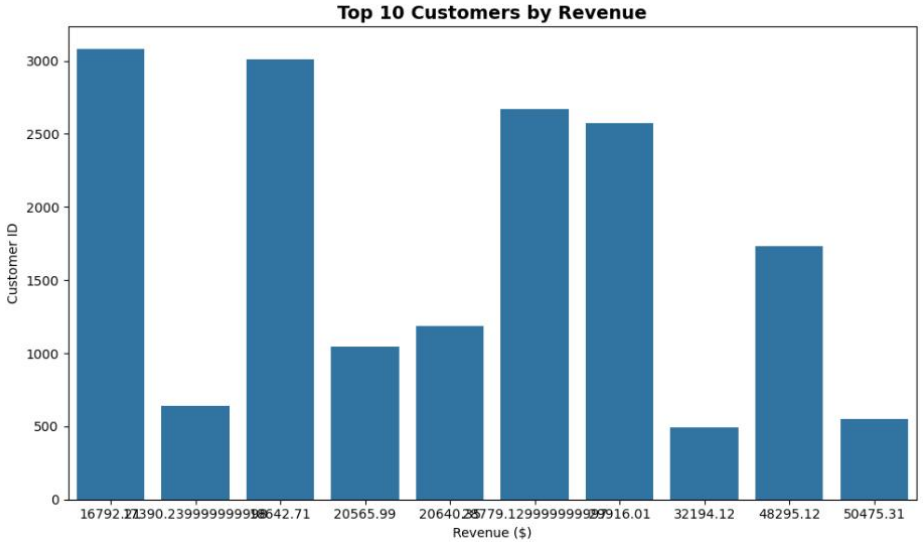
```
TOP 10 CUSTOMERS BY REVENUE:
------------------------------------------------------------
      Total Sales  Total Profit  Order Count
169       50475.31     3427.3460            4
563       48295.12     8533.4256            2
151       32194.12     1893.4200            4
847       29916.01     4234.9720            1
879       28779.13     1928.1800            1
387       20640.35       32.6760            2
343       20565.99     3678.9600            2
996       18642.71     2460.3160            2
202       17390.24     2280.9590            3
1015      16792.21      313.4900            4
```

```
plt.figure(figsize=(8, 5))

sns.histplot(

   customer_stats['Total Sales'],

   bins=30,

   kde=True

)

plt.title('Customer Revenue Distribution')

plt.xlabel('Revenue ($)')

plt.ylabel('Number of Customers')
```

plt.tight_layout()

plt.show()






```python
# KPI CARDS
total_revenue = customer_stats['Total Sales'].sum()
avg_customer_value = customer_stats['Total Sales'].mean()
top_customer_revenue = customer_stats['Total Sales'].max()

print(f"💰 Total Revenue: ${total_revenue:,.0f}")
print(f"👤 Avg Customer Value: ${avg_customer_value:,.0f}")
print(f"🏆 Top Customer Revenue: ${top_customer_revenue:,.0f}")
```

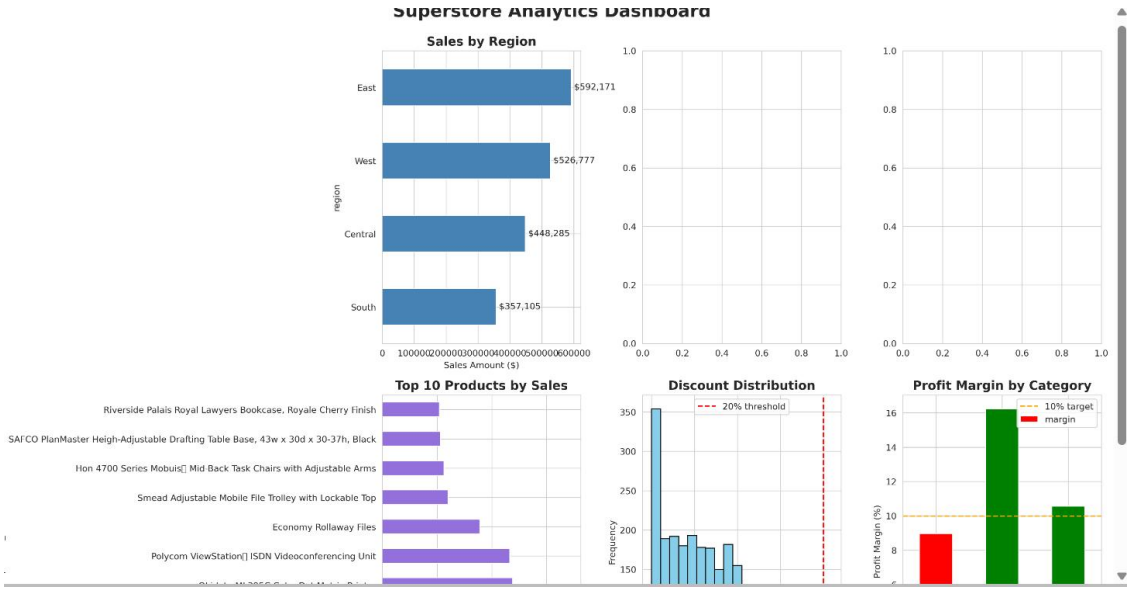
```
💰 Total Revenue: $1,924,338
👤 Avg Customer Value: $1,703
🏆 Top Customer Revenue: $50,475
```

```python
# Create and save visualization charts

import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Ensure order_date is datetime
df['order_date'] = pd.to_datetime(df['order_date'])

print("GENERATING VISUALIZATIONS")

sns.set_style("whitegrid")

fig, axes = plt.subplots(2, 3, figsize=(18, 12))
fig.suptitle(
    'Superstore Analytics Dashboard',
    fontsize=20,
    fontweight='bold',
    y=0.995
)

# 1. Regional Sales
regional = df.groupby('region')['sales'].sum().sort_values()
regional.plot(kind='barh', ax=axes[0, 0], color='steelblue')
axes[0, 0].set_title('Sales by Region', fontsize=14, fontweight='bold')
axes[0, 0].set_xlabel('Sales Amount ($)')

for i, v in enumerate(regional.values):
    axes[0, 0].text(v, i, f' ${v:,.0f}', va='center')
```

```python
# 4. Top 10 Products
top_products = df.groupby('product_name')['sales'].sum().nlargest(10)
top_products.plot(kind='barh', ax=axes[1, 0], color='mediumpurple')
axes[1, 0].set_title('Top 10 Products by Sales', fontsize=14, fontweight='bold')
axes[1, 0].set_xlabel('Sales Amount ($)')


# 5. Discount Distribution
df['discount_pct'] = df['discount'] * 100
axes[1, 1].hist(df['discount_pct'], bins=20, color='skyblue', edgecolor='black')
axes[1, 1].set_title('Discount Distribution', fontsize=14, fontweight='bold')
axes[1, 1].set_xlabel('Discount %')
axes[1, 1].set_ylabel('Frequency')
axes[1, 1].axvline(20, color='red', linestyle='--', label='20% threshold')
axes[1, 1].legend()


# 6. Profit Margin by Category
cat_stats = df.groupby('product_category').agg({'sales': 'sum', 'profit': 'sum'})
cat_stats['margin'] = (cat_stats['profit'] / cat_stats['sales']) * 100

colors = ['green' if x > 10 else 'red' for x in cat_stats['margin']]
cat_stats['margin'].plot(kind='bar', ax=axes[1, 2], color=colors)

axes[1, 2].set_title('Profit Margin by Category', fontsize=14, fontweight='bold')
axes[1, 2].set_ylabel('Profit Margin (%)')
axes[1, 2].set_xlabel('')
axes[1, 2].tick_params(axis='x', rotation=45)
axes[1, 2].axhline(10, color='orange', linestyle='--', label='10% target')
axes[1, 2].legend()
```

```python
# 6. Profit Margin by Category
cat_stats = df.groupby('product_category').agg({'sales': 'sum', 'profit': 'sum'})
cat_stats['margin'] = (cat_stats['profit'] / cat_stats['sales']) * 100

colors = ['green' if x > 10 else 'red' for x in cat_stats['margin']]
cat_stats['margin'].plot(kind='bar', ax=axes[1, 2], color=colors)

axes[1, 2].set_title('Profit Margin by Category', fontsize=14, fontweight='bold')
axes[1, 2].set_ylabel('Profit Margin (%)')
axes[1, 2].set_xlabel('')
axes[1, 2].tick_params(axis='x', rotation=45)
axes[1, 2].axhline(10, color='orange', linestyle='--', label='10% target')
axes[1, 2].legend()


plt.tight_layout()
plt.savefig('superstore_dashboard.png', dpi=300, bbox_inches='tight')
plt.close()

print("✓ Dashboard saved as 'SUPERSTORE_ANALYTICS_DASHBOARD.png'")
print("✓ All visualizations generated successfully")
```

```
GENERATING VISUALIZATIONS
💰 Total Revenue: $1,924,338
👤 Avg Customer Value: $1,703
🏆 Top Customer Revenue: $50,475
```



Conclusion:

I Scheduled the entire end-to-end Python script to run automatically (e.g., every Sunday night) using cron or Windows Task Scheduler. Final Presentation Deck, clearly outlining key actionable insights

(e.g., "Region X is experiencing a 15% customer churn increase").