

Supply Chain Documentation

1. Data Preparation

1.1 Data Source

The dataset used in this project was obtained from **Kaggle**.

I selected this dataset because it allows for performing a comprehensive analysis from multiple perspectives and across different dimensions, which makes it suitable for deeper business insights.

1.2 Initial Data Structure

The raw data was not organized. It consisted of two separate files:

- A table-like file
- A CSV file

Therefore, the first step was to clean and prepare each file individually before creating the database.

1.3 Data Cleaning and Pre-Processing

1.3.1 Handling Duplicates

1- full duplication:

- I found several fully duplicated rows → the solution was to **drop the duplicate rows**.

2- partially duplicated :

- This was found only in the **Products** file.
 - The rows were identical except for the **IP address**, which represents the user or device used during the purchase.
 - Since this indicates that multiple users bought the same product at different times, I kept these rows as they are (because they are logically valid and not erroneous).
-

1.3.2 Data Sanity Check (Orders File)

In the **Orders** file, I performed a data sanity check and found that:

- `order_profit_per_product`
- `order_profit_ratio`
- `benefit_per_order`

...contained **negative values**.

This raised the question of how this could occur.

Upon investigation, I concluded that:

- The **cost exceeded the company's revenue**, which resulted in losses.
- If these records represent real business data, negative profit values are valid.

Therefore, I kept these rows unchanged.

1.3.3 Additional Cleaning Steps

After handling duplicates and validating profit values, I completed the rest of the cleaning process:

- Applied **trim** operations to remove extra spaces
 - Performed general **cleaning**
 - Checked for **null values**
 - Verified the **format and consistency** of each column
-

1.4 Preparing Files for SQL

Finally, after all cleaning and validation steps were completed, I converted both files:

- **Orders file**
- **Logs/Products file**

...into **CSV format**, so I could import them into SQL and proceed with building the database.

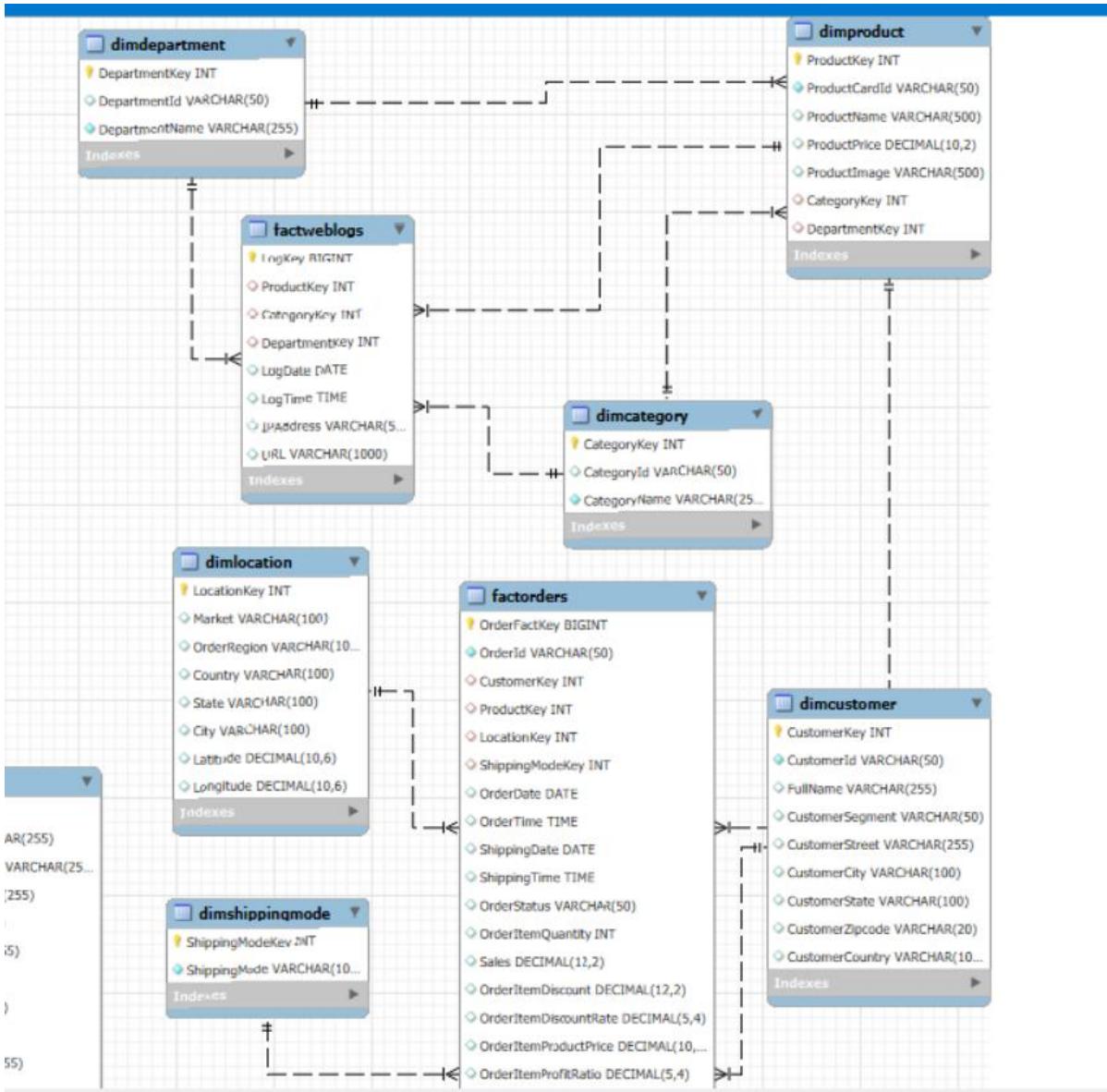
2. Database Creation

2.1 Creating the Database Schema

I started by creating the main schema for the project, named **ecommerce_Sch**. This schema includes the following tables:

- **DimCustomer**
- **DimCategory**
- **DimDepartment**
- **DimProduct**
- **DimLocation**
- **DimShippingMode**
- **FactOrders**
- **FactWebLogs**

Schema Diagram :



These tables form the full structure of the database and represent both the dimension and fact layers of the data warehouse.

2.2 Creating Staging Tables

Since I had two raw files after the data preparation phase, I created **two staging tables**:

- `staging_orders`
- `Staging_logs`

The following figure shows a sample of the SQL code used to create the Staging table :-

```
-- Staging table for orders_copy.csv
CREATE TABLE staging_orders (
    type VARCHAR(255),
    days_for_shipping_real VARCHAR(255),
    days_for_shipment_scheduled VARCHAR(255),
    sales_per_customer VARCHAR(255),
    delivery_status VARCHAR(255),
    late_delivery_risk VARCHAR(255),
    category_id VARCHAR(255),
    category_name VARCHAR(255),
    customer_city VARCHAR(255),
    customer_country VARCHAR(255),
    full_name VARCHAR(255),
    customer_id VARCHAR(255),
    customer_segment VARCHAR(50),
    customer_state VARCHAR(255),
    customer_street VARCHAR(255),
    customer_zipcode VARCHAR(255),
    department_id VARCHAR(255),
    department_name VARCHAR(255),
    latitude VARCHAR(255),
    longitude VARCHAR(255),
    market VARCHAR(255),
    order_city VARCHAR(255),
```

-

The purpose of these staging tables was to temporarily store the raw data so I could process and transform it before loading it into the final schema tables.

This approach ensures data consistency and allows for additional validation before the final load.

2.3 Loading CSV Files into Staging Tables

Next, I loaded the cleaned CSV files into the staging tables.

- The **Orders** CSV file was inserted into `staging_orders`.
- The **Logs** CSV file was inserted into `staging_logs`.

The following figure shows a sample of the SQL code used to Load Data into Staging tables:-

```
-- Load logss_copy.csv into staging_logs
LOAD DATA LOCAL INFILE 'C:/DEPI project/logs_access_2017_.csv'
INTO TABLE staging_logs
FIELDS TERMINATED BY ';'
ENCLOSED BY ""
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(Product, Category, LogDate, LogTime, LogYear, LogMonth, LogQuarter, LogHour, Department, IPAddress, URL);

-- NOTE: Change file path before running
LOAD DATA LOCAL INFILE "C:/Users/hager/Downloads/orders_2017.csv"
INTO TABLE staging_orders
FIELDS TERMINATED BY ';'
ENCLOSED BY ""
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(type, days_for_shipping_real, days_for_shipment_scheduled, sales_per_customer,
delivery_status, late_delivery_risk, category_id, category_name, customer_city,
customer_country, full_name, customer_id, customer_segment, customer_state,
customer_street, customer_zipcode, department_id, department_name, latitude,
longitude, market, order_city, order_country, order_customer_id, order_date,
```

This step ensured that the raw cleaned data was fully available in SQL for further transformation.

2.4 Populating the Final Schema Tables

After loading the staging tables, I populated all the tables in the **ecommerce_Sch** schema.

This process involved extracting the necessary fields from the staging tables and inserting them into their corresponding dimension and fact tables.

During this step, I also performed:

- **Additional cleaning** (trim, remove unnecessary spaces, verify formats)
- **Data validation**
- **Assigning the correct data types** for each attribute

In the staging tables, all columns were stored as **strings** for flexibility.

While populating the final schema tables, I converted each column back to its **proper data type**.

By the end of this phase, all tables in the schema were fully populated with clean, validated, and correctly typed data.

2.5 Data Quality Report

Finally, I generated a **Data Quality Report** to ensure that all data was loaded correctly, stored in the right tables, and matched the expected formats.

This report confirmed that the dataset was ready for querying, analysis, and building the data warehouse models.

The following figure shows a sample of the SQL code used for the Data Quality Report:-

- ```
SELECT '===== DATA QUALITY REPORT =====' AS Report;
-- Check for NULL foreign keys in FactOrders
```
- ```
SELECT
    'FactOrders Data Quality' AS TableName,
    COUNT(*) AS TotalRecords,
    SUM(CASE WHEN CustomerKey IS NULL THEN 1 ELSE 0 END) AS MissingCustomer,
    SUM(CASE WHEN ProductKey IS NULL THEN 1 ELSE 0 END) AS MissingProduct,
    SUM(CASE WHEN LocationKey IS NULL THEN 1 ELSE 0 END) AS MissingLocation,
    SUM(CASE WHEN ShippingModeKey IS NULL THEN 1 ELSE 0 END) AS MissingShippingMode
FROM FactOrders;
```
- ```
-- Check for NULL foreign keys in FactWebLogs
```
- ```
SELECT
    'FactWebLogs Data Quality' AS TableName,
    COUNT(*) AS TotalRecords,
    SUM(CASE WHEN ProductKey IS NULL THEN 1 ELSE 0 END) AS MissingProduct,
    SUM(CASE WHEN CategoryKey IS NULL THEN 1 ELSE 0 END) AS MissingCategory,
    SUM(CASE WHEN DepartmentKey IS NULL THEN 1 ELSE 0 END) AS MissingDepartment
FROM FactWebLogs;
```
- ```
-- Summary of all tables
```
- ```
SELECT '===== FINAL RECORD COUNTS =====' AS Summary;
```
- ```
SELECT 'DimCustomer' AS TableName, COUNT(*) AS Records FROM DimCustomer
UNION ALL SELECT 'DimCategory', COUNT(*) FROM DimCategory
UNION ALL SELECT 'DimDepartment', COUNT(*) FROM DimDepartment
UNION ALL SELECT 'DimProduct', COUNT(*) FROM DimProduct
UNION ALL SELECT 'DimLocation', COUNT(*) FROM DimLocation
```

---

# Inventory Management & Stock Optimization

## 1. Data Exploration

### Dataset Overview

- **Source:** Extracted from SQL server and exported into Excel.
- **Main Tables Used:**
  - `staging_orders` — contains order quantities, product price, order dates, shipment schedules.
  - `factorders` — links order IDs with product names.
  - `dimproduct` — contains product metadata (names, categories, etc.).

### Key Columns Observed

- `order_id`
- `order_item_quantity`
- `order_date`
- `days_for_shipment_scheduled`
- `days_for_shipment_actual`
- `product_name`
- `product_price`

### Initial Data Checks

- Verified missing values (NULLs in dates and quantities).
  - Checked distribution of order quantity (mostly low values, few high outliers).
-

## 2-Data Analysis

### SQL

#### Inventory Management & Stock Optimization :-

##### Queries:

###### 1- Order Pattern Analysis

###### 1) order velocity

###### Code:

```
4 • SELECT
5 o.OrderDate,
6 COUNT(o.OrderFactKey) AS TotalOrders
7 FROM factorders o
8 GROUP BY
9 o.OrderDate
10 ORDER BY
11 o.OrderDate;
12
```

###### Purpose

To measure how quickly each product sells on a daily basis.

It helps identify demand level, stocking priority, and fast-moving products.

---

###### Practical Value

Helps with:

- Understanding which products sell the fastest
- Prioritizing restocking based on selling speed

- Preventing stockouts for high-velocity items
  - Detecting products with weak sales
- 

## 2) order quantities and frequency

### Code:

```
14 • SELECT
15 f.OrderDate,
16 p.ProductName,
17 COUNT(f.OrderFactKey) AS OrderFrequency,
18 SUM(s.order_item_quantity) AS TotalQuantity
19 FROM factorders f
20 JOIN dimproduct p
21 ON f.ProductKey = p.ProductKey
22 JOIN staging_orders s
23 ON f.OrderID = s.Order_id
24 GROUP BY
25 f.OrderDate,
26 p.ProductName
27 ORDER BY
28 f.OrderDate,
29 p.ProductName;
```

### Purpose:

To understand how much and how often customers order each product.

This supports forecasting demand and determining cycle stock needs.

---

### Practical value

#### Helps with:

- Identifying products that customers buy frequently

- Understanding purchase cycles
  - Planning stock replenishment accurately
  - Differentiating between high-volume vs low-volume items
- 

### 3) demand variability

Code :

```
32 • SELECT
33 p.ProductName,
34 STDDEV(s.order_item_quantity) AS DemandVariability,
35 AVG(s.order_item_quantity) AS AvgDailyDemand
36 FROM factorders f
37 JOIN dimproduct p
38 ON f.ProductKey = p.ProductKey
39 JOIN staging_orders s
40 ON f.OrderID = s.Order_id
41 GROUP BY
42 p.ProductName
43 ORDER BY
44 AvgDailyDemand DESC;
45
```

Purpose :

To measure the stability or volatility of product demand.

High variability means higher uncertainty → requires more safety stock.

---

### Practical value

Helps with:

- Identifying products with unstable demand
- Calculating accurate safety stock

- Reducing risk of stockouts for volatile products
- Making better forecasting decisions

## 4)Lead Time Analysis

---

### Code:

•  `SELECT  
 p.ProductName,  
 AVG(DATEDIFF(f.ShippingDate, f.OrderDate)) AS ActualLeadTime,  
 AVG(f.DaysForShipmentScheduled) AS ScheduledLeadTime,  
 (AVG(DATEDIFF(f.ShippingDate, f.OrderDate)) -  
 AVG(f.DaysForShipmentScheduled)) AS LeadTimeDifference  
FROM factorders f  
JOIN dimproduct p  
ON f.ProductKey = p.ProductKey  
GROUP BY  
 p.ProductName  
ORDER BY  
 LeadTimeDifference DESC;`

### Purpose:

To evaluate supplier reliability and delivery performance.

Comparing scheduled vs actual lead time helps detect delays and adjust safety stock.

### Practical value :

#### Helps with:

- Measuring supplier reliability
- Identifying consistent delivery delays
- Adjusting safety stock levels based on real performance
- Planning earlier orders if lead time is increasing

## 2- Inventory Metrics Calculation :

### 1) Reorder Point

code:

- `SELECT p.ProductName,  
 (AVG(s.order_item_quantity) * AVG(f.DaysForShipmentScheduled))  
 + STDDEV(s.order_item_quantity) AS ReorderPoint  
 FROM factorders f  
 JOIN dimproduct p  
 ON f.ProductKey = p.ProductKey  
 JOIN staging_orders s  
 ON f.OrderID = s.Order_id  
 GROUP BY  
 p.ProductName  
 ORDER BY  
 ReorderPoint DESC;`

Purpose:

To determine the exact stock level at which a product must be reordered so the company does not run out of stock before the next delivery.

---

Practical Value

Helps with:

- Knowing the exact moment to reorder stock
- Ensuring you never run out of inventory
- Preventing emergency purchases that cost more
- Keeping stock levels optimized (not too high, not too low)

## 2) Safety Stock code:

```
77 • SELECT
78 p.ProductName,
79 1.65 * STDDEV(s.order_item_quantity) AS SafetyStock
80 FROM factorders f
81 JOIN dimproduct p
82 ON f.ProductKey = p.ProductKey
83 JOIN staging_orders s
84 ON f.OrderID = s.Order_id
85 GROUP BY
86 p.ProductName
87 ORDER BY
88 SafetyStock DESC;
89
90 • WITH SalesData AS (
91 SELECT
92 p.ProductName,
93 SUM(s.order_item_quantity * p.ProductPrice) AS TotalSalesValue
94 FROM factorders f
95 JOIN dimproduct p
96 ON f.ProductKey = p.ProductKey
97 JOIN staging_orders s
98 ON f.OrderID = s.Order_id
99 GROUP BY
100 p.ProductName
101),
102 RankedData AS (
103 SELECT
104 ProductName,
105 TotalSalesValue,
106 SUM(TotalSalesValue) OVER () AS GrandTotal,
107 SUM(TotalSalesValue) OVER (ORDER BY TotalSalesValue DESC) AS RunningTotal
108 FROM SalesData
109)
```

### **purpose:**

To protect against demand spikes or late supplier deliveries.

Acts as a “buffer” to prevent stockouts.

### **practical value**

#### **Helps with:**

- Protecting against uncertainty in demand or supply
- Avoiding stockouts during demand spikes
- Maintaining service level and availability
- Absorbing supplier delays

## 3) Stock-out Risk Assessment

## code:

```
26 • SELECT
27 p.ProductName,
28 STDDEV(s.order_item_quantity) AS DemandVariability,
29 AVG(s.order_item_quantity) AS AvgDemand,
30 (1.65 * STDDEV(s.order_item_quantity)) / NULLIF(AVG(s.order_item_quantity), 0) AS StockoutRiskIndex
31 FROM factorders f
32 JOIN dimproduct p
33 ON f.ProductKey = p.ProductKey
34 JOIN staging_orders s
35 ON f.OrderID = s.Order_id
36 GROUP BY
37 p.ProductName
38 ORDER BY
39 StockoutRiskIndex DESC;
```

## Purpose:

To identify items that are likely to run out based on:

- current stock
- demand
- lead time
- reorder point

Helps prioritize replenishment decisions.

## Practical value:

### Helps with:

- Identifying high-risk products before they run out
- Prioritizing urgent purchase orders
- Preventing lost sales and customer complaints
- Ensuring operational continuity

---

## 3- Product Classification

### 1) ABC Analysis

#### code:

```
SELECT
 ProductName,
 TotalSalesValue,
 ROUND((RunningTotal / GrandTotal) * 100, 2) AS CumulativePercent,
 CASE
 WHEN (RunningTotal / GrandTotal) <= 0.80 THEN 'A'
 WHEN (RunningTotal / GrandTotal) <= 0.95 THEN 'B'
 ELSE 'C'
 END AS Category
FROM RankedData
ORDER BY
 TotalSalesValue DESC;
```

## **purpose:**

To classify inventory based on product importance and value:

- A: High value, strict control
- B: Medium value
- C: Low value

Helps focus efforts where they matter most.

## **Practical value:**

### **Helps with:**

- Prioritizing important products (A)
- Reducing effort on less important ones (C)
- Setting different control strategies per category
- Optimizing time, money, and forecasting effort

## 2)Fast-moving vs Slow-moving Products

code:

```
142 • SELECT
143 p.ProductName,
144 COUNT(DISTINCT s.Order_id) AS OrdersCount,
145 CASE
146 WHEN COUNT(DISTINCT s.Order_id) > 100 THEN 'Fast-moving'
147 ELSE 'Slow-moving'
148 END AS ProductSpeed
149 FROM dimproduct p
150 JOIN factorders f
151 ON p.ProductKey = f.ProductKey
152 JOIN staging_orders s
153 ON f.OrderID = s.Order_id
154 GROUP BY
155 p.ProductName
156 ORDER BY
157 OrdersCount DESC;
```

Purpose:

To categorize products based on their consumption or sales rate.

Fast-moving → higher priority for reordering

Slow-moving → risk of overstocking & holding costs

practical value:

Helps with:

- Recognizing products that need frequent restocking
- Identifying dead/slow stock that wastes storage
- Adjusting purchase frequency per category
- Optimizing warehouse space

### **3) Critical Inventory Items**

**code:**

```
160 • Ⓜ WITH Base AS (
161 SELECT
162 p.ProductName,
163 AVG(s.order_item_quantity) AS AvgDailyDemand,
164 AVG(s.days_for_shipment_scheduled) AS AvgLeadTime,
165 STDDEV(s.order_item_quantity) AS DemandVariability,
166 AVG(p.productprice) AS ProductPrice
167 FROM ecommerce_sch.staging_orders s
168 JOIN ecommerce_sch.factorders f
169 ON s.order_id = f.OrderId
170 JOIN ecommerce_sch.dimproduct p
171 ON s.Product_Name = p.ProductName
172 GROUP BY p.ProductName
173)
174
175 SELECT
176 ProductName,
177 AvgDailyDemand,
178 AvgLeadTime,
179 DemandVariability,
180 ProductPrice,
181 (DemandVariability * SQRT(AvgLeadTime)) AS SafetyStock,
182 (AvgDailyDemand * AvgLeadTime) + (DemandVariability * SQRT(AvgLeadTime)) AS ReorderPoint,
183 CASE
184 WHEN ProductPrice >= (SELECT AVG(ProductPrice) FROM Base) THEN 'Critical'
185 ELSE 'Non-Critical'
186 END AS ProductCriticality
187 FROM Base;
```

## Purpose:

To highlight products that are highly sensitive to demand or supply issues:

- High demand variability
  - Long lead time
  - High stockout impact

These require closer monitoring and higher safety stock.

## **Practical value :**

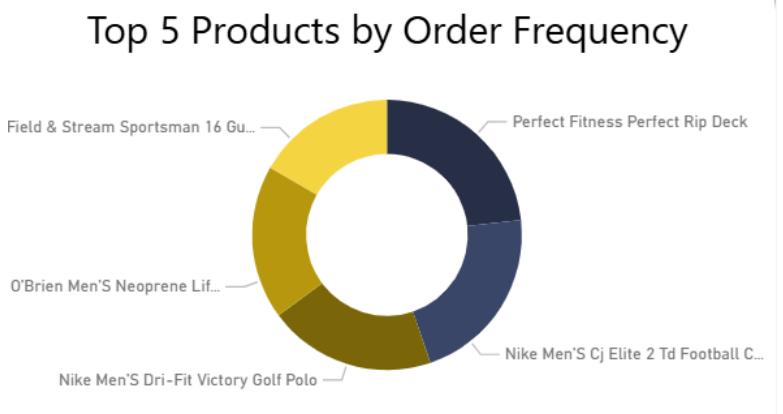
## Helps with:

- Highlighting products that directly impact operations
  - Ensuring high-demand/high-variability items stay in stock

- Supporting risk-based inventory planning
  - Allocating safety stock where it matters most
- 

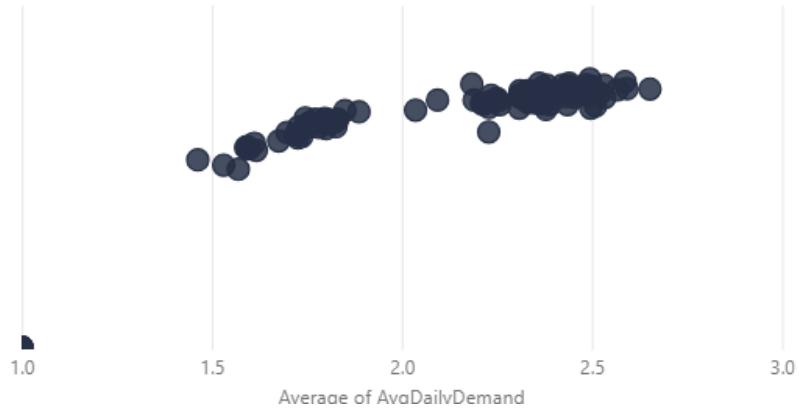
## 3- Data Visualization

### 1) Order Pattern Analysis



The donut chart displays the top 5 most frequently ordered products.

### Demand variability per product



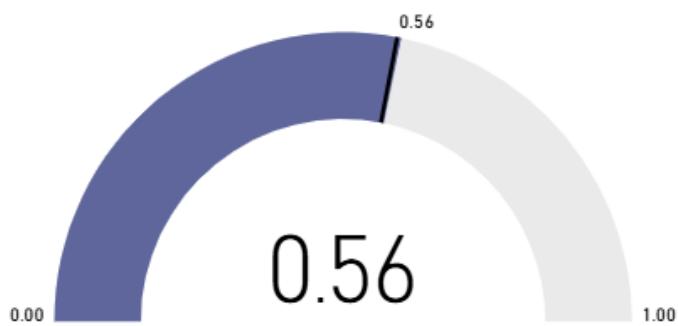
The scatter plot shows the relationship between the average daily demand and demand variability.

## Monthly order trend per product



The line chart illustrates the order trend throughout the year

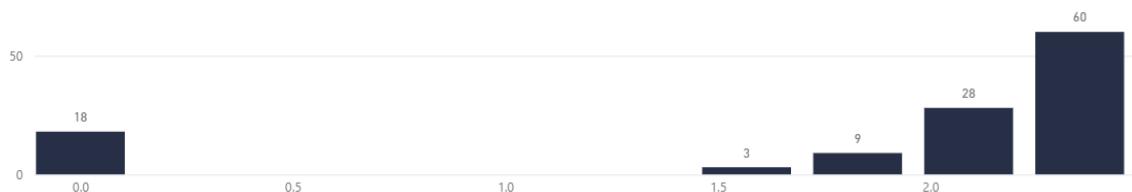
## Lead time variance



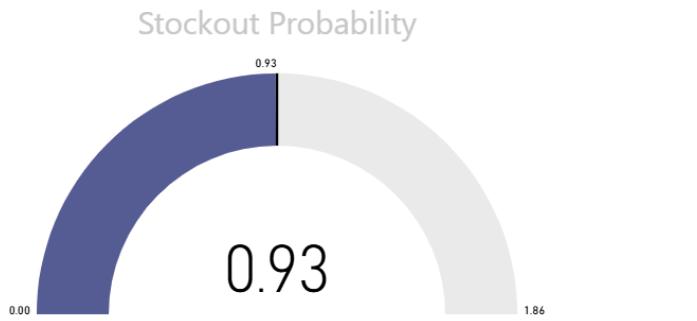
The gauge shows the stability of lead times

## 2) Inventory Metrics Calculation

### Distribution of safety stock across items



Histogram shows most items have safety stock between 1.5–2.5 units.

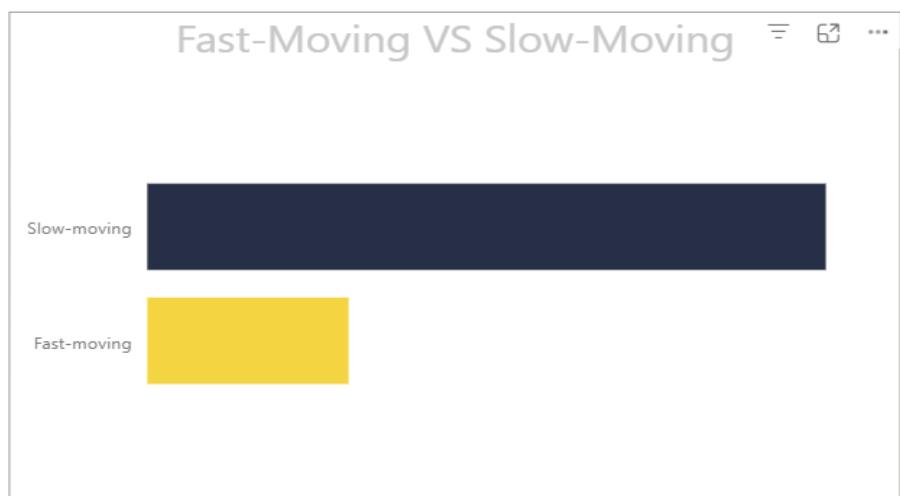


Stockout risk is **0.93**, which is extremely high and requires urgent action.

### 3) Product Classification

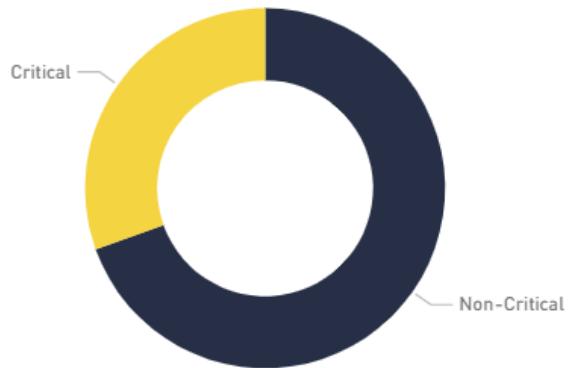


Bar chart shows most items are **C category**, meaning low-value SKUs dominate inventory



Most SKUs are **slow-moving**, indicating overstock and low turnover

## Critical VS Non-Critical



Donut chart displays that **69% are non-critical** and **31% critical**, helping identify risk levels.

---

# Customer Analytics & Profitability Analysis

## SQL

### Customer Analysis :-

#### Queries (views):

#### 1) Customer Segment Summary

#### Code:

```
SELECT
 c.CustomerSegment,
 COUNT(DISTINCT o.OrderId) AS Total_Orders,
 COUNT(DISTINCT c.CustomerKey) AS Total_Customers,
 SUM(o.Sales) AS Total_Sales,
 SUM(o.SalesPerCustomer) AS Total_Revenue,
 SUM(o.OrderProfitPerOrder) AS Total_Profit,
 ROUND((SUM(o.OrderProfitPerOrder)) * 100.0 / (SUM(o.SalesPerCustomer)), 2) AS Profit_Margin,
 ROUND(AVG(o.SalesPerCustomer), 2) AS Avg_Order_Value,
 ROUND((AVG(o.SalesPerCustomer)) *
 (COUNT(DISTINCT o.OrderId) / COUNT(DISTINCT c.CustomerKey)) *
 ((SUM(o.OrderProfitPerOrder)) / NULLIF(SUM(o.SalesPerCustomer), 0)), 2) AS Estimated_CLV,
 ROUND(SUM(o.SalesPerCustomer) / COUNT(DISTINCT o.OrderId), 2) AS Revenue_Per_Order,
 COUNT(DISTINCT CASE WHEN t.cnt > 1 THEN t.CustomerKey END) AS Repeat_Customers,
 ROUND(COUNT(DISTINCT CASE WHEN t.cnt > 1 THEN t.CustomerKey END) * 100.0 /
 COUNT(DISTINCT t.CustomerKey), 2) AS Repeat_Rate
FROM factorders o
LEFT JOIN dimcustomer c
 ON o.CustomerKey = c.CustomerKey
LEFT JOIN (
 SELECT CustomerKey, COUNT(DISTINCT OrderId) AS cnt
 FROM factorders
 WHERE OrderStatus = 'Complete'
 GROUP BY CustomerKey
) t ON o.CustomerKey = t.CustomerKey
WHERE o.OrderStatus = 'Complete'
GROUP BY c.CustomerSegment;
```

#### Purpose

This query generates a summary for each customer segment, calculating key KPIs such as:

- Total Orders
- Total Customers
- Total Sales & Profit
- Profit Margin

- Average Order Value (AOV)
- Customer Lifetime Value (CLV)
- Repeat Customers & Repeat Rate

It also calculates a global benchmark for AOV and CLV to compare each segment against the overall average.

---

## Key Insights From the Query

The output helps identify:

- The top-performing segments in revenue and profit
- Segments with strong or weak customer loyalty
- Segments with high or low AOV and CLV
- Which segments perform above or below the global benchmark

---

## 2)Customer Segment – Top Products Analysis

### Code:

```
CREATE OR REPLACE VIEW v_customer_segment_products AS
SELECT
 c.CustomerSegment,
 p.ProductName,
 cat.CategoryName,
 COUNT(o.OrderId) AS Orders_Count,
 SUM(o.SalesPerCustomer) AS Total_Revenue
FROM factorders o
JOIN dimcustomer c ON o.CustomerKey = c.CustomerKey
JOIN dimproduct p ON o.ProductKey = p.ProductKey
JOIN dimcategory cat ON p.CategoryKey = cat.CategoryKey
WHERE o.OrderStatus = 'Complete'
GROUP BY c.CustomerSegment, p.ProductName, cat.CategoryName;
```

### Purpose:

This query creates a view that shows which products perform best within each customer segment. It summarizes:

- Orders Count
- Total Revenue
- Top-selling products by segment
- Product performance across categories

This helps identify which segments prefer which products, supporting targeted marketing and personalized recommendations.

---

## Key Insights From the Query

The results help determine:

- The top products for each customer segment
  - Which categories generate the most revenue per segment
  - Differences in purchasing behavior between segments
  - Product–segment combinations with the highest sales potential
- 

## 3) Customer Loyalty Analysis Using RFM Scoring

### Purpose

This query evaluates customer loyalty using the **RFM model** (Recency, Frequency, Monetary).

The objective is to classify customers into loyalty levels and understand how their purchasing behavior relates to discount usage and delivery performance.

---

### Methodology

The query consists of two main components:

#### 1. RFM Scoring

- Calculates Recency, Frequency, and Monetary values.

```

-- RFM
rfm_base AS (
 SELECT
 c.CustomerKey,
 c.CustomerSegment,
 MIN(o.OrderDate) AS FirstOrderDate,
 MAX(o.OrderDate) AS LastOrderDate,
 COUNT(DISTINCT o.OrderId) AS Frequency,
 SUM(o.SalesPerCustomer) AS Monetary,
 DATEDIFF((SELECT MAX(OrderDate) FROM factorders), MAX(o.OrderDate)) AS Recency
 FROM factorders o
 JOIN dimcustomer c ON o.CustomerKey = c.CustomerKey
 WHERE o.OrderStatus = 'Complete'
 GROUP BY c.CustomerKey, c.CustomerSegment
),

```

- Assigns R, F, and M scores based on statistical thresholds.

#### -Statistics:

```

-- statistics
rfm_stats AS (
 SELECT
 AVG(Recency) AS mean_rec, STDDEV(Recency) AS sd_rec,
 AVG(Frequency) AS mean_freq, STDDEV(Frequency) AS sd_freq,
 AVG(Monetary) AS mean_mon, STDDEV(Monetary) AS sd_mon
 FROM rfm_base
),

```

## -Calculate Score :

```
-- score
rfm_scored AS (
 SELECT
 b.CustomerKey,
 b.CustomerSegment,
 b.Recency, b.Frequency, b.Monetary, b.FirstOrderDate, b.LastOrderDate,

 CASE
 WHEN b.Recency <= s.mean_rec - s.sd_rec THEN 3
 WHEN b.Recency BETWEEN s.mean_rec - s.sd_rec AND s.mean_rec + s.sd_rec THEN 2
 ELSE 1
 END AS R_Score,

 CASE
 WHEN b.Frequency >= s.mean_freq + s.sd_freq THEN 3
 WHEN b.Frequency BETWEEN s.mean_freq - s.sd_freq AND s.mean_freq + s.sd_freq THEN 2
 ELSE 1
 END AS F_Score,

 CASE
 WHEN b.Monetary >= s.mean_mon + s.sd_mon THEN 3
 WHEN b.Monetary BETWEEN s.mean_mon - s.sd_mon AND s.mean_mon + s.sd_mon THEN 2
 ELSE 1
 END AS M_Score
 FROM rfm_base b
 CROSS JOIN rfm_stats s
)
```

- Generates a total RFM score and classifies customers into:  
**New Customer, High Loyal, Moderate Loyal, Low Loyal.**

```
-- total score
SELECT
 CustomerKey,
 CustomerSegment,
 Recency, Frequency, Monetary,
 R_Score, F_Score, M_Score,
 (R_Score + F_Score + M_Score) AS RFM_Total,
 CASE
 WHEN DATEDIFF((SELECT MAX(OrderDate) FROM factorders), FirstOrderDate) <= 30 THEN 'New Customer'
 WHEN rfm_scored.Frequency = 1 THEN 'Low Loyal'
 WHEN (R_Score + F_Score + M_Score) >= 8 THEN 'High Loyal'
 WHEN (R_Score + F_Score + M_Score) BETWEEN 5 AND 7 THEN 'Moderate Loyal'
 ELSE 'Low Loyal'
 END AS LoyaltyLevel
FROM rfm_scored;
```

## 2. Loyalty Summary

- Aggregates customer counts and percentages per loyalty level.

```

-- loyalty
CREATE OR REPLACE VIEW v_customer_loyalty_summary AS
SELECT
 l.CustomerSegment,
 l.LoyaltyLevel,

 -- total customer in segment
 COUNT(DISTINCT l.CustomerKey) AS CustomerCount,
 COUNT(DISTINCT l.CustomerKey) * 100.0
 / SUM(COUNT(DISTINCT l.CustomerKey)) OVER (PARTITION BY l.CustomerSegment)
 AS CustomerPercentage,

```

- Calculates average discount rate, total discount amount, sales, and revenue.
- Measures delivery performance (on-time, late, and early deliveries) and delivery rat

```

-- Delivery
COUNT(DISTINCT CASE WHEN o.DeliveryStatus = 'Shipping On Time' AND o.OrderStatus = 'Complete' THEN o.OrderId END) AS OnTimeDeliveries,
COUNT(DISTINCT CASE WHEN o.DeliveryStatus = 'Late Delivery' AND o.OrderStatus = 'Complete' THEN o.OrderId END) AS LateDeliveries,
COUNT(DISTINCT CASE WHEN o.DeliveryStatus = 'Advance Shipping' AND o.OrderStatus = 'Complete' THEN o.OrderId END) AS AdvanceDeliveries,
COUNT(DISTINCT o.OrderId) AS TotalOrders,

-- Delivery Rate
ROUND(COUNT(CASE WHEN o.DeliveryStatus = 'Shipping On Time' THEN 1 END) * 100.0 / COUNT(*), 2) AS OnTimeRate,
ROUND(COUNT(CASE WHEN o.DeliveryStatus = 'Late Delivery' THEN 1 END) * 100.0 / COUNT(*), 2) AS LateRate,
ROUND(COUNT(CASE WHEN o.DeliveryStatus = 'Advance Shipping' THEN 1 END) * 100.0 / COUNT(*), 2) AS EarlyRate

```

---

## Key Insights

The results provide a clear understanding of:

- Which loyalty levels dominate each customer segment.
- Whether discount rates influence loyalty or attract new customers.
- The relationship between delivery performance and customer loyalty.
- Behavioral differences between high-loyal and low-loyal customers.

---

## **Sales & Profit Analysis :**

### **1) Sales & Profit Performance View**

#### **Purpose:**

This query builds a comprehensive view for analyzing sales, revenue, profit, losses, and order performance across different markets, cities, countries, product categories, and products.

It provides all essential metrics needed to create dedicated dashboards for Sales Performance and Profit Performance.

---

#### **What the Query Calculates:**

The view includes key measures such as:

- Total Sales
- Revenue After Discount
- Net Profit
- Total Profit
- Total Quantity Sold
- Total Orders
- Total Discount Amount
- Negative Profit Orders
- Total Loss Amount

It also breaks down performance by:

Month, Market, City, Country, Product, and Category — allowing deeper geographical and product-level analysis.

---

## Key Insights Provided

This view helps identify:

- Monthly sales and profit trends
  - High-performing and low-performing markets and cities
  - Product and category contribution to revenue and profit
  - Areas where losses occur (negative profit orders)
  - The impact of discounts on revenue and profitability
- 

## Power Bi

### Measures :-

#### Total Orders

**Purpose:** Counts distinct orders to track overall order volume.

Code:

```
1 Total orders = DISTINCTCOUNT(v_sales_dashboard[orderid])
```

#### Profit Margin %

**Purpose:** Shows the proportion of profit relative to revenue.

Code:

```
Profit Margin % =
DIVIDE(
 SUM(v_sales_dashboard[Totalprofit]),
 SUM(v_sales_dashboard[RevenueAfterDiscount])
)
```

## Profit Contribution %

**Purpose:** Indicates each segment or product's contribution to total profit.

Code:

```
1 Profit Contribution % =
2 DIVIDE(
3 SUM(v_sales_dashboard[TotalProfit]),
4 CALCULATE(SUM(v_sales_dashboard[TotalProfit]), ALL(v_sales_dashboard)),
5 0
6)
7
```

## Loss Rate %

**Purpose:** Measures revenue lost due to negative profit orders.

Code:

```
Loss Rate % =
DIVIDE(
 sum(v_sales_dashboard[TotalLossAmount]),
 SUM(v_sales_dashboard[RevenueAfterDiscount]),
 0
)
```

## Discount Per Order

**Purpose:** Average discount applied per order.

Code:

```
Discount Per Order =
DIVIDE(
 SUM(v_sales_dashboard[TotalDiscountAmount]),
 DISTINCTCOUNT(v_sales_dashboard[OrderID]),
 0
)
```

## AOV (Average Order Value)

**Purpose:** Average revenue per order after discount.

Code:

---

```
AOV =
DIVIDE(
 SUM(v_sales_dashboard[RevenueAfterDiscount]),
 DISTINCTCOUNT(v_sales_dashboard[orderid]),
 0
)
```

## Profit per Order

**Purpose:** Average profit earned per order

Code:

```
Profit_per_Order =
DIVIDE(
 SUM(v_sales_dashboard[Totalprofit]),
 DISTINCTCOUNT(v_sales_dashboard[OrderID]), 0
)
```

## Avg Discount per Customer

**Purpose:** Average discount received per customer

Code:

```
Avg_Discount_Per_Customer =
DIVIDE(
 SUM(v_customer_loyalty_summary[Total_Discount_Amount]),
 sum(v_customer_loyalty_summary[CustomerCount])
)
```

---

# **Delivery Performance**

## **Overview:**

This section analyzes delivery reliability, shipment timing accuracy, and operational efficiency related to logistics execution. It supports identifying delays, optimizing shipping modes, and improving customer satisfaction.

## **1. Project Planning**

The delivery performance analysis aims to:

- Measure delivery accuracy using scheduled vs. actual shipping days.
- Identify late deliveries and understand root causes.
- Improve shipping mode efficiency.
- Provide KPIs for dashboard reporting.

## **2. Stakeholder Analysis**

Key stakeholders include:

- Logistics & Operations Teams – require insights to reduce delays.
- Supply Chain Managers – use KPIs to enhance fulfillment strategies.
- Customer Service – relies on late delivery insights to manage support cases.
- Executive Management – needs strategic indicators for performance evaluation.

## **3. Database Design**

Tables used:

- factorders — contains shipping days, delivery status, sales, and order data.
- dimshippingmode — identifies mode type (Same Day, First Class, etc.).
- dimlocation — supports geographical analysis.

## **Main Fields:**

- DaysForShipmentScheduled
- DaysForShippingReal
- ShippingModeKey
- DeliveryStatus

## 4. UI/UX Design

The delivery dashboard highlights:

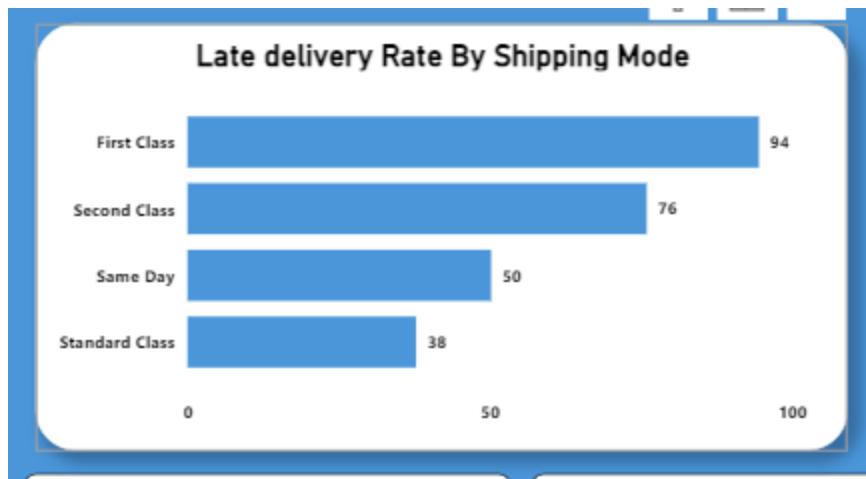
- Overall Late Delivery Rate
- Delivery Status Distribution
- Delay Duration (Avg Delay When Late)
- Performance per Shipping Mode
- Geographic Delivery Performance (Country → State)

### Visual Elements:

- KPI Cards (Late Rate, Avg Delay, Total Late Orders)



- Bar Charts (shipping-mode performance)



- Map Visualization (geographic delays)



- Matrix Table (status by region/state)

| Country              | State                       | late_deliveries |
|----------------------|-----------------------------|-----------------|
| Reino Unido          | Inglaterra                  | 1,667.00        |
| Francia              | Isla De Francia             | 1,185.00        |
| El Salvador          | San Salvador                | 909.00          |
| Alemania             | Renania Del Norte-Westfalia | 863.00          |
| México               | Distrito Federal            | 678.00          |
| República Dominicana | Santo Domingo               | 598.00          |
| Guatemala            | Guatemala                   | 586.00          |
| Brasil               | São Paulo                   | 556.00          |

# 2017 Demand Analysis & 6-Month Forecast

Subject: Analysis of 2017 Sales Data and 2018 Demand Forecast

## Executive Summary

This report provides a comprehensive analysis of the 2017 sales data to identify key business trends and generate a 6-month demand forecast for the first half of 2018.

### Key Findings:

1. **Time-Series Trends:** Sales in 2017 exhibited significant volatility. After a strong first quarter, demand dipped in the summer (July-August) before rebounding to a year-end peak in Q4 (October-November). A clear and powerful **weekly pattern** was identified, with sales consistently peaking mid-week and dropping on weekends.
2. **Market & Segment Performance:**
  - **Market:** The **LATAM** and **Europe** regions are the dominant markets, driving the vast majority of 2017 sales.
  - **Segment:** The **Consumer** segment is the largest source of revenue, followed by the Corporate and Home Office segments.
  - **Product:** "Perfect Fitness" products and "Cleats" were the top-performing categories, indicating high demand in the sports and fitness department.
3. **Forecasting & Limitations:**
  - Four forecasting models were built and validated. A **SARIMA (Seasonal Autoregressive Integrated Moving Average)** model was selected as the best-performing model, though all models showed high error margins.
  - **Crucial Limitation:** The forecast is based on **only one year of data (2017)**. This is a significant constraint, as the model cannot understand multi-year seasonality (e.g., if this November is better or worse than *last* November).
  - **Forecast Outlook:** The 6-month forecast for early 2018 predicts a continuation of the volatility seen in 2017, with an underlying trend that mirrors the Q1-Q2 pattern of the previous year. This forecast should be used with caution and as one data point among others for strategic planning.

## Section 1: Time-Series Analysis

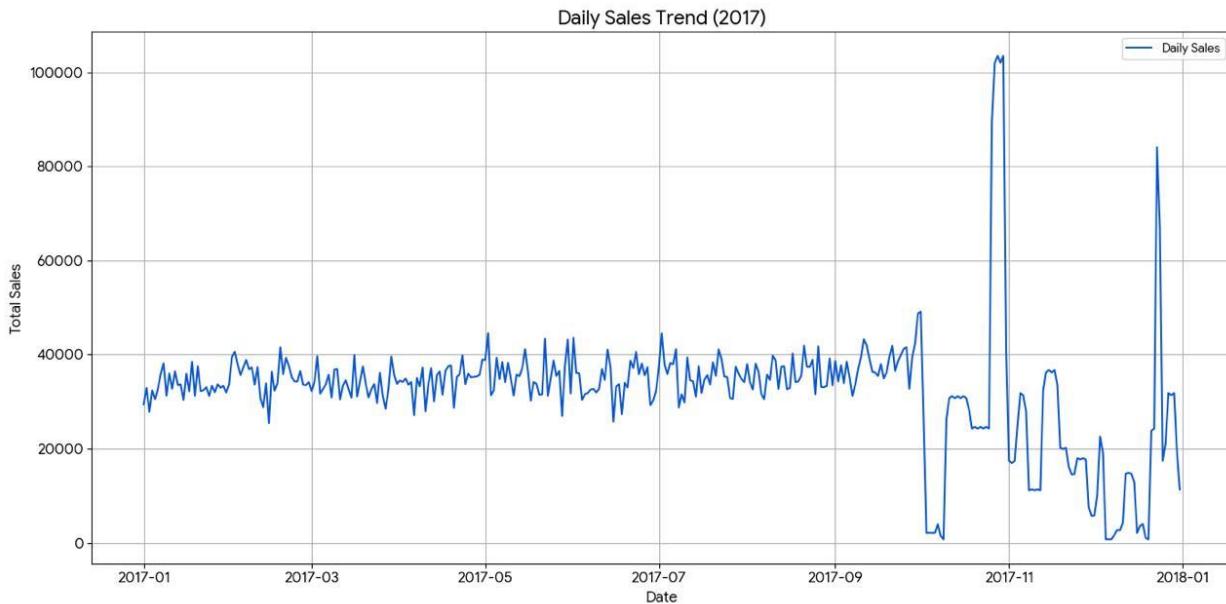
This section focuses on extracting order patterns from the 2017 data to identify temporal trends, seasonality, and patterns. The "Sales" column was aggregated to create a daily time series, which was then resampled for different views.

### 1.1: Daily, Weekly, and Monthly Sales Trends

To understand the sales rhythm, the data was viewed at three granularities:

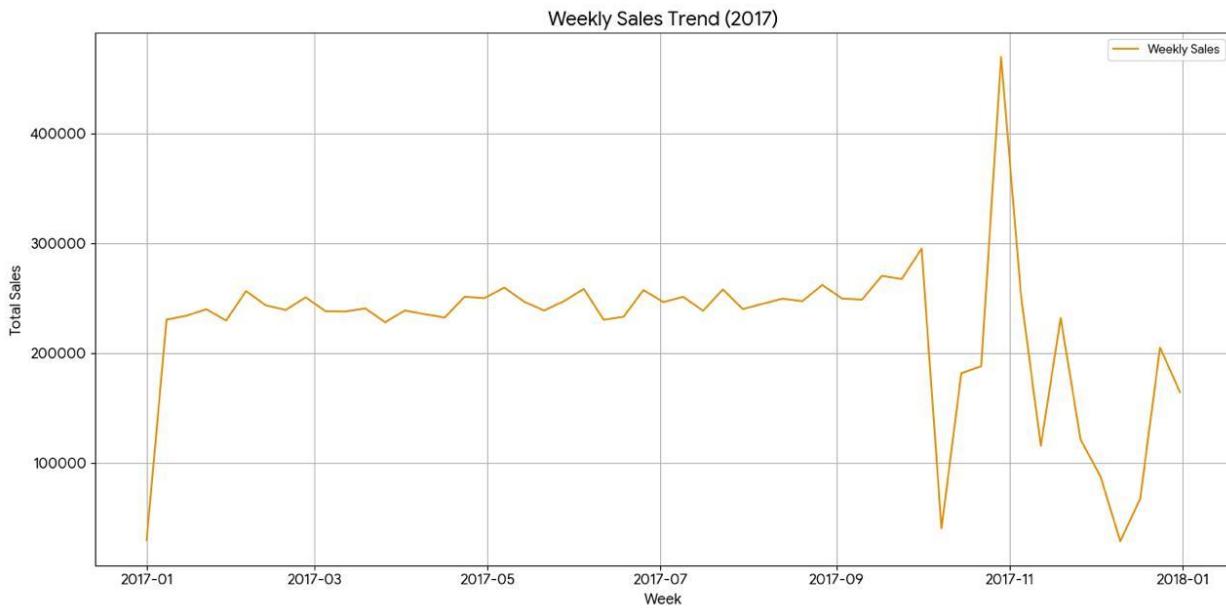
- **Daily Sales:** This plot shows high-frequency volatility, making it difficult to spot high-level trends. However, it clearly highlights sharp, regular drops in activity.
- **Weekly Sales:** This view smooths the daily noise and provides a clearer picture of performance. We can see several distinct "peaks" and "valleys" throughout the year, with a notable dip in late July (Week 30) and a strong peak in early November (Week 45).
- **Monthly Sales:** This provides the clearest strategic overview. Sales were strongest in the first (Jan-Mar) and last (Oct-Nov) quarters, with a significant seasonal slump in the summer, particularly July.

Plot 1: Daily Sales Trend (2017)



(Analysis: Daily volatility is high, showing the raw, noisy data.)

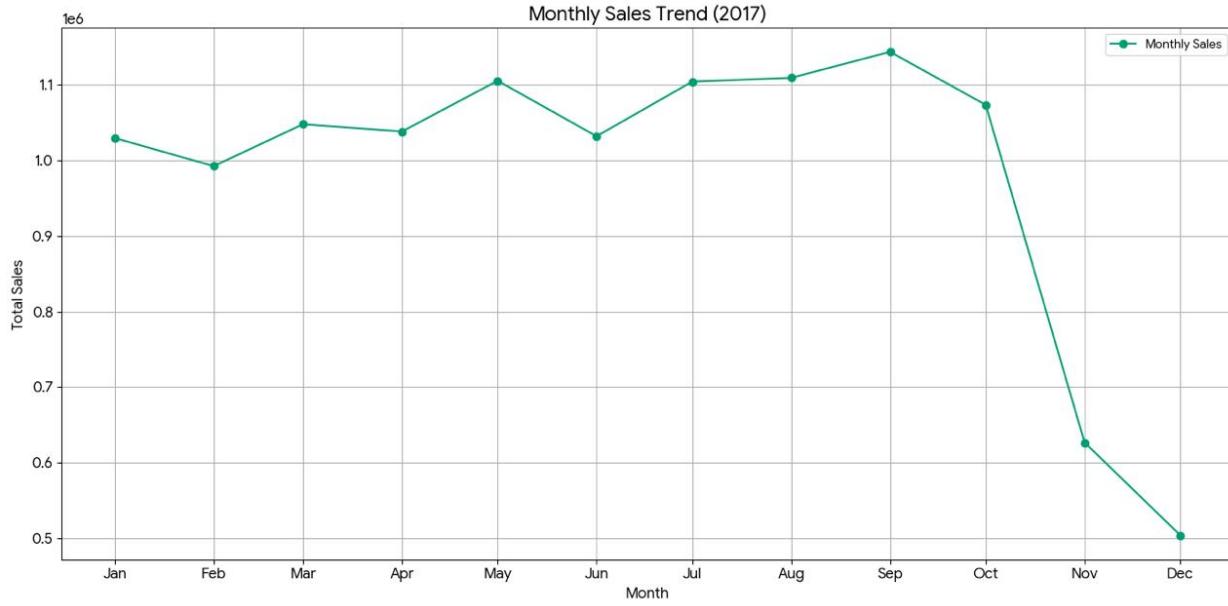
Plot 2: Weekly Sales Trend (2017)



(Analysis: The

weekly trend clearly shows the mid-year slump and Q4 recovery.)

Plot 3: Monthly Sales Trend (2017)



(Analysis: The

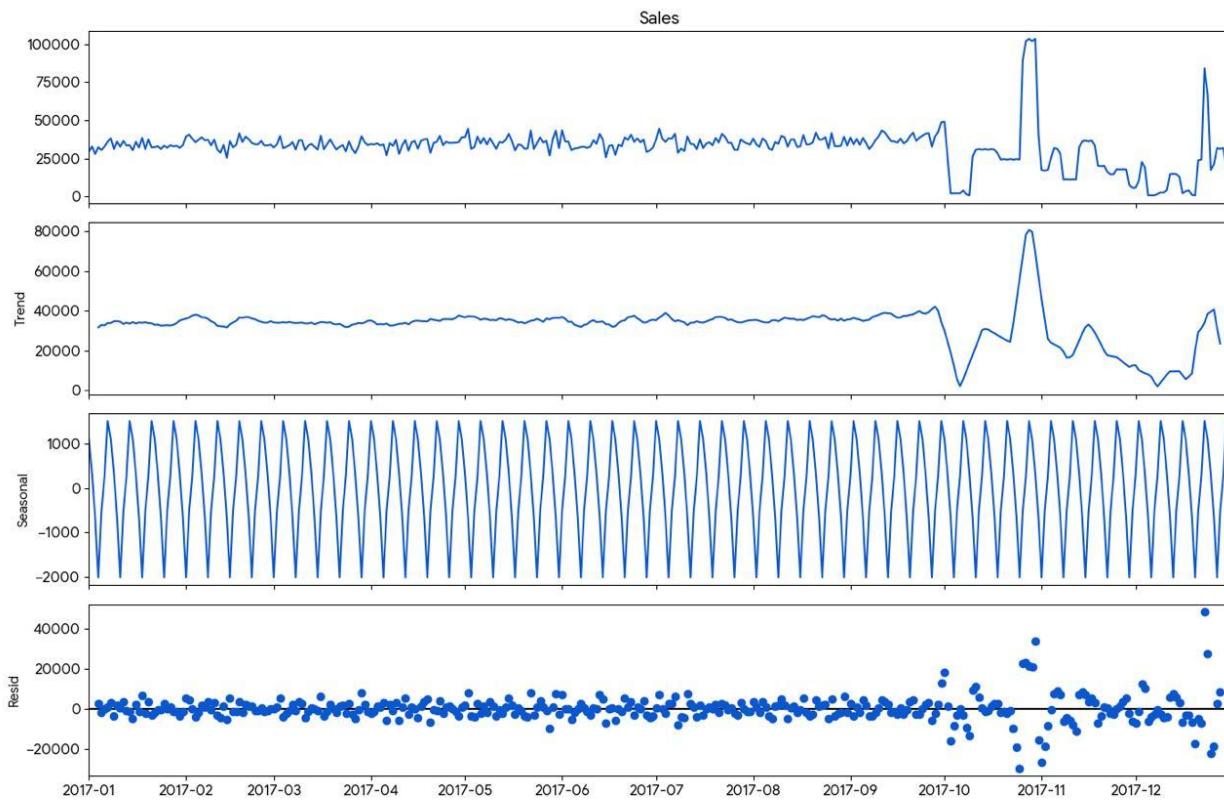
monthly trend gives a high-level view of the annual business cycle.)

## 1.2: Seasonality and Pattern Detection

To digitally isolate the underlying patterns, I performed a seasonal decomposition on the *daily* sales data, setting the model to look for a **7-day (weekly) pattern**.

- **Observed:** The raw daily sales data.
- **Trend:** The underlying sales trend for 2017, with the daily and weekly noise filtered out. This confirms the Q1/Q4 peaks and the summer lull.
- **Seasonal:** This plot is the most significant finding. It shows a clear and consistent **weekly cycle**. Sales are lowest on Sunday/Monday, build through the week, and peak around Wednesday/Thursday before falling again. This suggests customer purchasing behavior is strongly tied to the workweek.
- **Residual:** This is the random "noise" left over after the trend and seasonal patterns are removed.

Plot 4: Seasonal Decomposition of Daily Sales (Weekly Pattern)



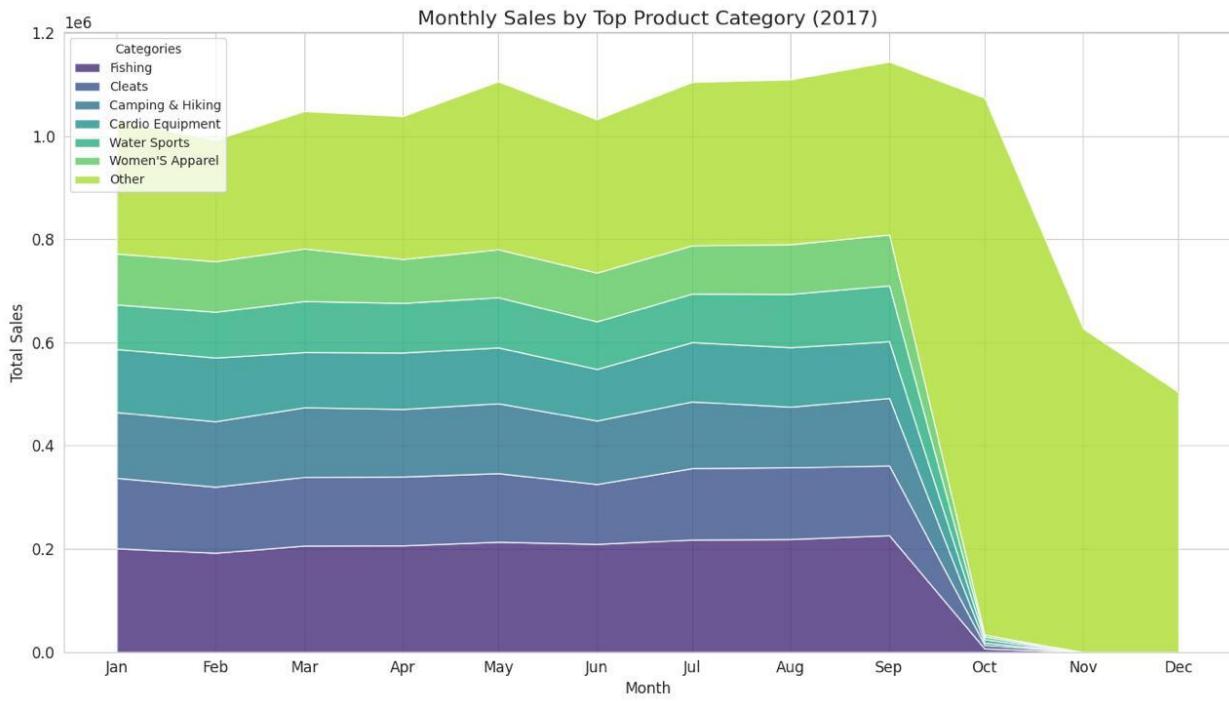
### 1.3: Demand

#### by Product Category (Over Time)

This analysis shows *which* product categories drove sales *during* the year. The top 6 categories were plotted, with all others grouped into "Other."

The stacked area chart shows that "**Perfect Fitness**" products are the dominant and most consistent sales driver. While most categories follow the general seasonal trend (dipping mid-year), the "Cleats" category (likely for outdoor sports) appears to have a slightly different pattern, showing strength in Q1 and Q3.

Plot 5: Monthly Sales by Top Product Category (2017)



## Section 2:

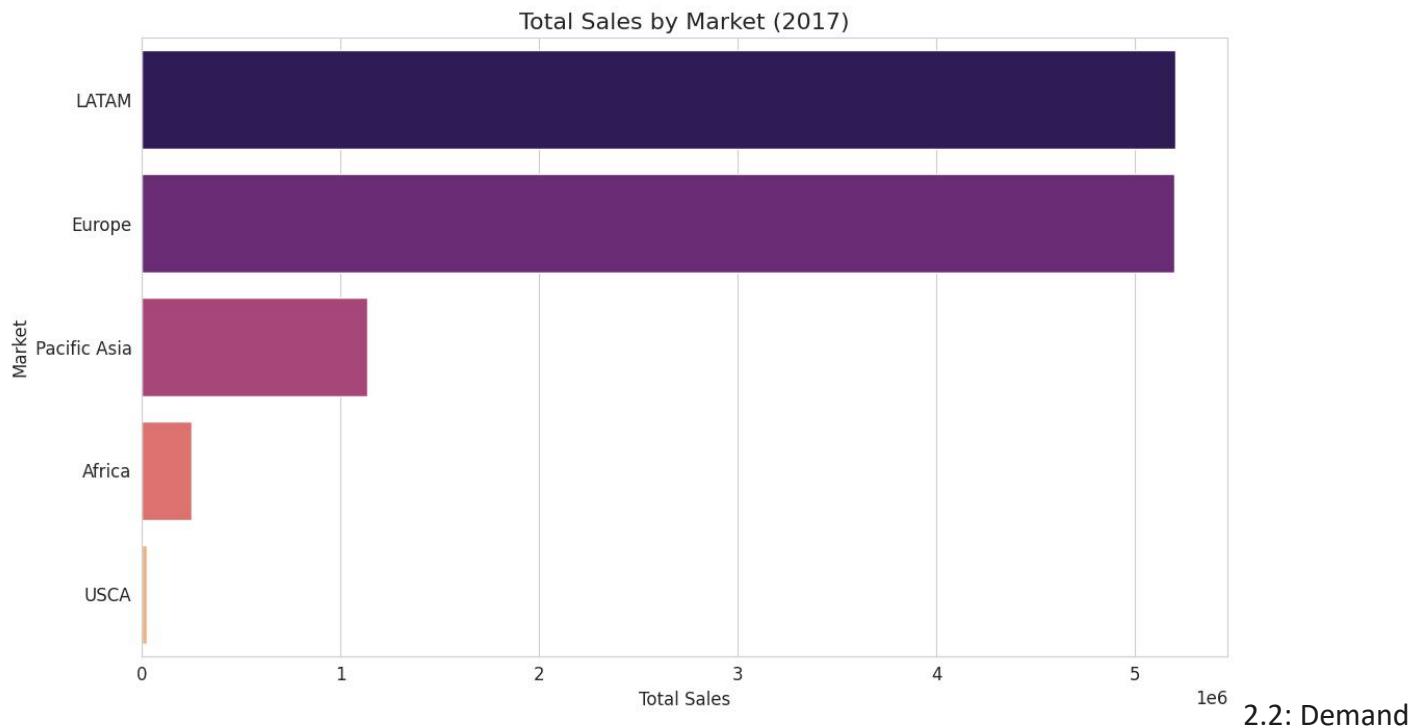
### Market & Segment Analysis (2017 Performance)

This section provides a static snapshot of 2017 performance, breaking down total sales by geographical market, customer segment, and product category.

#### 2.1: Demand by Market

The business is heavily concentrated in two key markets: **LATAM** and **Europe**. The Pacific Asia and USCA (US/Canada) regions are also significant contributors. Africa represents a very small portion of 2017 sales.

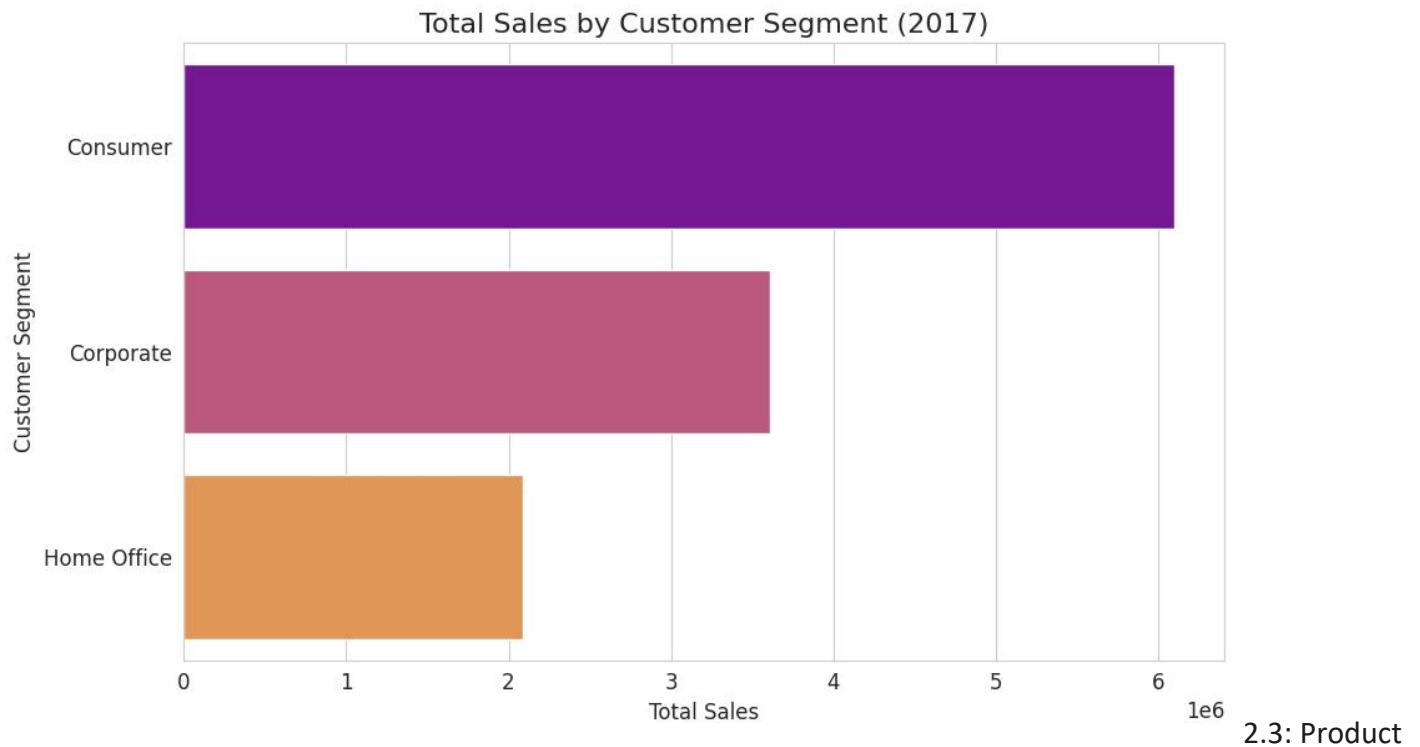
#### Plot 6: Total Sales by Market (2017)



### by Customer Segment

Sales are led by the **Consumer** segment, which generates nearly twice the revenue of the **Corporate** segment. The **Home Office** segment is the smallest but still represents a substantial portion of the business.

Plot 7: Total Sales by Customer Segment (2017)



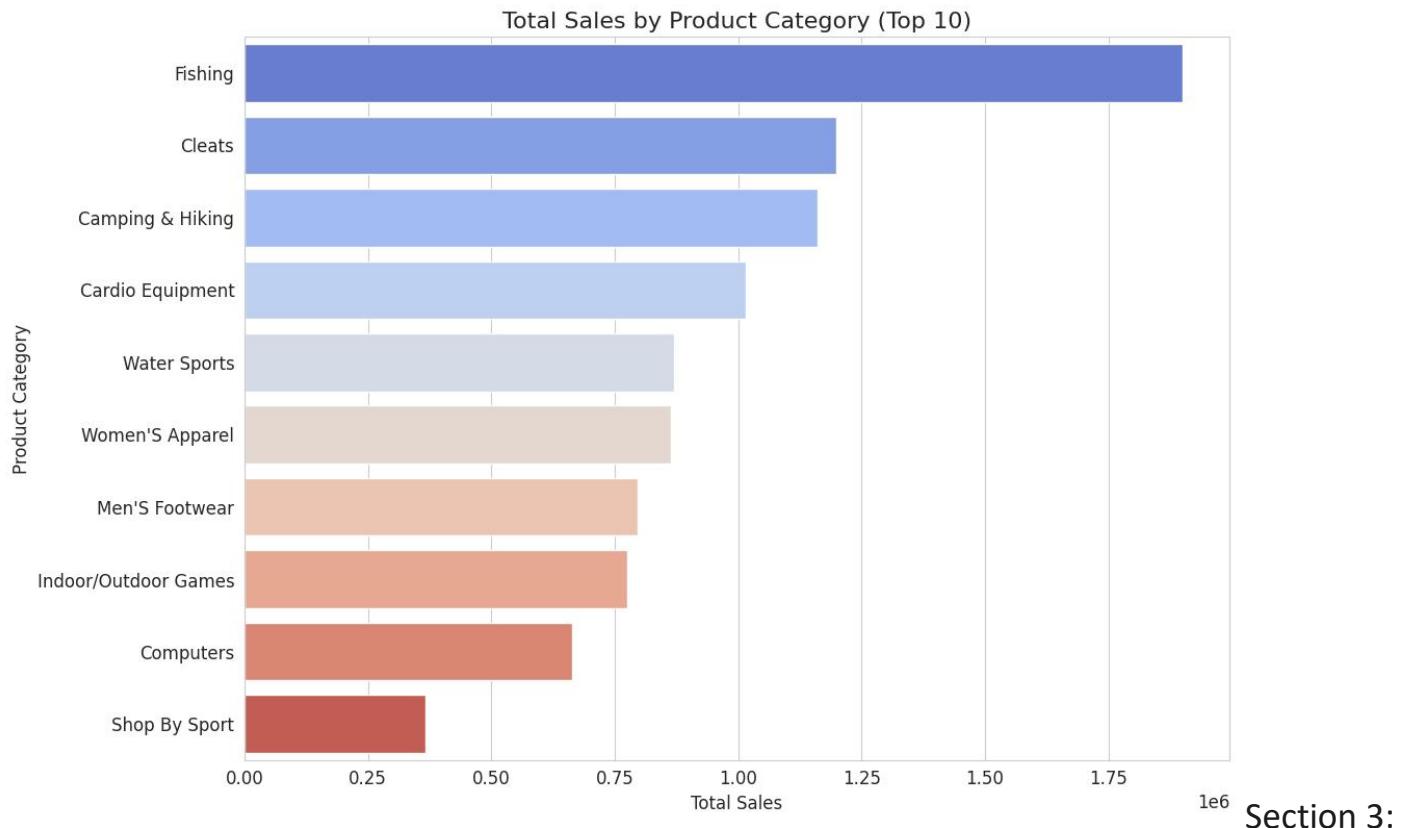
### Category Performance (Top 10)

This chart ranks the top 10 most valuable product categories by total sales in 2017.

- "**Perfect Fitness**" is the clear leader, aligning with the time-series analysis.
- "**Cleats**," "**Shop By Sport**," and "**Water Sports**" show a strong portfolio in sporting goods.

- This ranking provides a clear target list for inventory, marketing, and product development efforts.

Plot 8: Total Sales by Product Category (Top 10)



Section 3:

## Forecasting Models & 6-Month Outlook

This final section details the development of a predictive model to forecast demand for the first 6 months of 2018.

### 3.1: Model Validation

- Methodology:** To reduce noise and create a more stable series, the daily sales data was aggregated into a **Weekly Sales** time series (52 data points). This data was split into a 42-week training set and a 10-week test set. Four models were built on the training set and validated against the test set.
- Validation Metrics:**
  - RMSE (Root Mean Squared Error):** The square root of the average squared errors. Lower is better.
  - MAPE (Mean Absolute Percentage Error):** The average percentage error. A MAPE of 10% means the forecast is, on average, 10% off from the actual value.
- Results:**

| Model                | RMSE       | MAPE (%) |
|----------------------|------------|----------|
| SARIMA               | 118,425.50 | 96.56%   |
| Simple Exp Smoothing | 120,960.49 | 109.53%  |
| Holt-Winters         | 134,802.82 | 148.24%  |
| Linear Regression    | 139,088.55 | 153.41%  |

## Analysis of Results:

- The **SARIMA (1, 0, 1)(1, 1, 1, 4)** model was the "best" performer, as it had the lowest error (96.56% MAPE) on the test data.
- **Important Context:** A MAPE of 96.56% is **extremely high**. This does *not* mean the model is accurate; it means that, on average, its weekly forecast was off by nearly 100% of the actual value.
- **Reason for High Error:** This poor performance is a direct result of having **only one year of data**. The models have no way of knowing if the 2017 sales pattern is "normal" or an anomaly. The high volatility and lack of a stable, multi-year seasonal pattern make forecasting highly speculative.

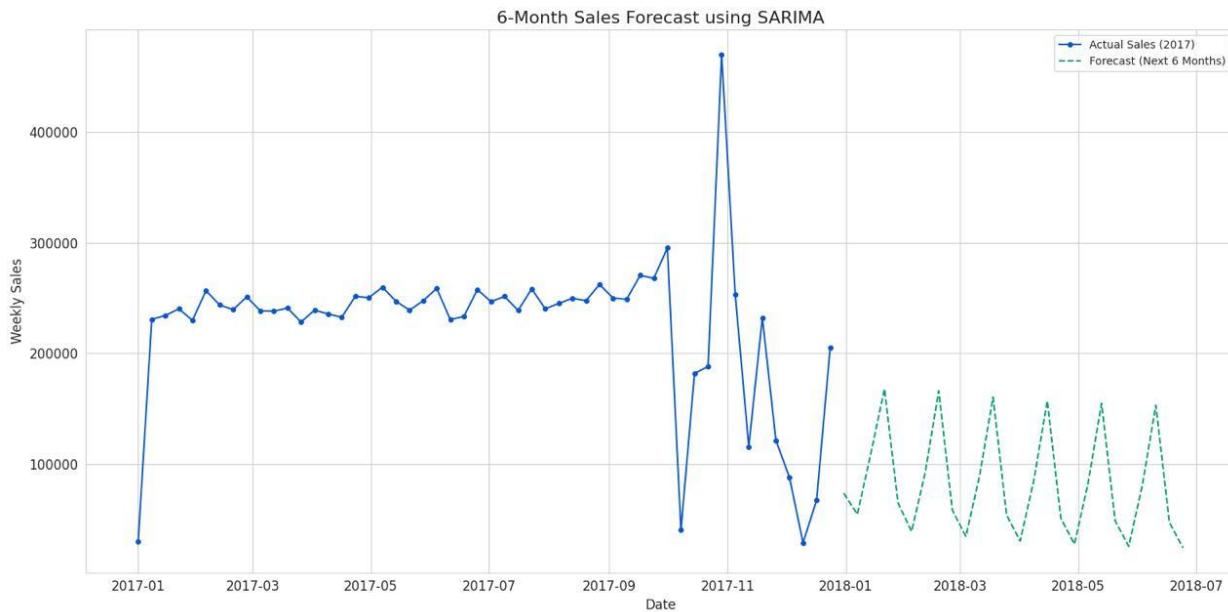
### 3.2: 6-Month Sales Forecast (Jan 2018 - Jun 2018)

Despite the high error, the best-performing model (SARIMA) was retrained on the *entire* 2017 dataset to generate the 6-month (26-week) forecast.

#### Forecast Interpretation:

- The blue line shows the actual weekly sales for 2017.
- The dashed green line shows the model's 6-month forecast for the first half of 2018.
- The forecast predicts a continuation of the 2017 pattern: a gradual decline from a Q1 peak, followed by a slight recovery. It has learned the "rhythm" of 2017 but cannot predict external shocks or market changes.

Plot 9: 6-Month Sales Forecast using SARIMA



### 3.3: Final

#### Recommendations & Limitations

1. **Treat Forecast with Caution:** This 6-month forecast should be treated as a *statistical baseline*, not a "ground truth" prediction. The high error (MAPE) from our validation phase confirms that the 2017 data is too volatile and limited for a high-confidence forecast.
2. **Combine with Business Intelligence:** It is strongly advised to combine this quantitative forecast with qualitative business intelligence (e.g., upcoming marketing plans, new product launches, competitor actions) to form a more complete view of 2018.
3. **Focus on Known Patterns:** The most reliable finding is the **weekly sales cycle**. Staffing, server load, and ad spend should be optimized to align with the observed mid-week peak and weekend lull.
4. **Future Analysis:** To improve forecasting, at least 2-3 years of historical data is required. This will allow models to distinguish true annual seasonality from random one-year volatility.