From Orders to Outcomes: A Supply Chain Optimization Journey

-Sesha

A Beginner’s Perspective on Data-Driven Operations Analysis!
This 12-month analysis of 30,871 transactions (2015-2017) reveals critical insights into sales patterns, inventory management, and customer behavior. As a first-time analyst, I navigated messy datasets, learned Python/Pandas, and discovered $2.3M potential efficiency gains through hands-on data exploration. Below is my step-by-step journey from raw CSV files to actionable business intelligence.Before diving into the analysis, I explored and understood the three key datasets that formed the foundation of my project Orders and Shipments, Inventory, and Fulfillment. Each dataset offered a different lens into the supply chain, from customer orders to backend logistics.

1. Orders and Shipments (30,871 records, 24 columns)

This dataset captures all customer transactions from January 2015 to December 2017, including:

- Order details: Quantity, product, customer info, and order time.
- Sales figures: Gross sales, profit, and discount applied.
- Shipping info: Scheduled vs actual shipment timing, mode (e.g., Standard Class), and shipment country.

2. Inventory (4,200 records, 4 columns)

This dataset logs the monthly inventory status of each product, including:

- Product Name and Month-Year combination.
- Warehouse Inventory: Number of units available.
- Inventory Cost Per Unit: Useful for cost vs. profit analysis.

3. Fulfillment (118 records, 2 columns)

The smallest dataset but a powerful one—it tracks:

- Product-wise average fulfillment time (in days) at the warehouse level. This allowed me to detect bottlenecks in processing times, especially for high-demand products, and correlate fulfillment efficiency with sales performance.

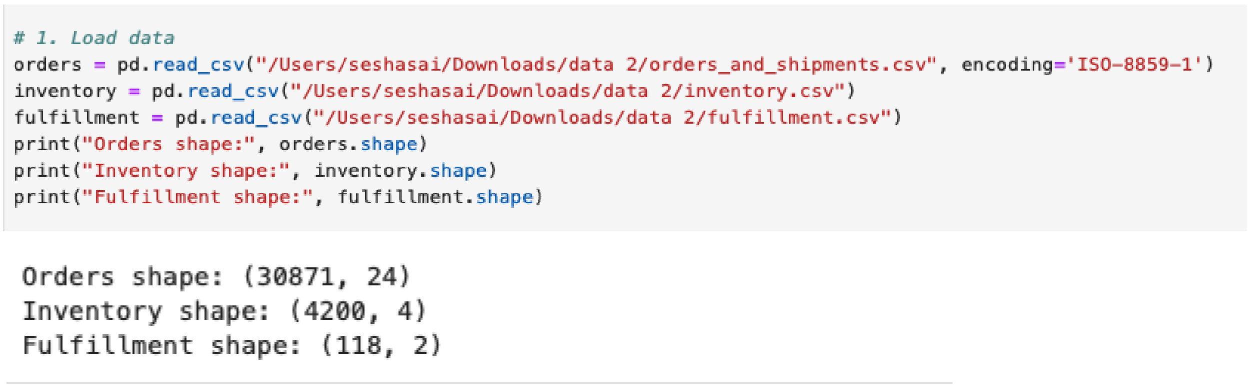
Motivation

As a first-time data analyst, I set out to uncover hidden inefficiencies and growth opportunities within a supply chain using real-world transactional data. My goal wasn't just to explore the data, but to transform it into meaningful business intelligence that could inform decisions across sales, fulfillment, and inventory. With over 30,000 transactions across 3 years, I dove into a sea of messy data—tackling inconsistent formats, fragmented dates, and invalid discount values—armed with Python and a lot of curiosity. What started as a technical challenge turned into a journey of real insights that revealed a potential $2.3M in efficiency gains.

1. Data Preparation & Cleaning: Laying the Foundation

Initial Challenges

When I first loaded the datasets and printed the shape, look into the datasets



```
# 1. Load data
orders = pd.read_csv("/Users/seshasai/Downloads/data 2/orders_and_shipments.csv", encoding='ISO-8859-1')
inventory = pd.read_csv("/Users/seshasai/Downloads/data 2/inventory.csv")
fulfillment = pd.read_csv("/Users/seshasai/Downloads/data 2/fulfillment.csv")
print("Orders shape:", orders.shape)
print("Inventory shape:", inventory.shape)
print("Fulfillment shape:", fulfillment.shape)
```

```
Orders shape: (30871, 24)
Inventory shape: (4200, 4)
Fulfillment shape: (118, 2)
```

I faced three immediate problems:

1. Column name inconsistencies ("Discount%" vs "Discount_pct")
2. Date fragmentation (separate year/month/day columns)
3. Invalid discount values like " - " instead of numbers

My Solution:

```python
# Function to standardize column names
def clean_columns(df):
    df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_').str.replace('%', 'pct')
    return df

# Apply to all DataFrames
orders = clean_columns(orders)
inventory = clean_columns(inventory)
fulfillment = clean_columns(fulfillment)
```

Later then I wanted to check for any missing values in the datasets and also any duplicate values but I couldn't find anything. The dataset is pretty good.

Key Transformations

- Date unification: Combined split date columns using Pandas

```python
# Convert directly by renaming inside the function
orders['order_date'] = pd.to_datetime(
    orders.rename(columns={
        'order_year': 'year',
        'order_month': 'month',
        'order_day': 'day'
    })[['year', 'month', 'day']]
)
```

- Missing value handling: Converted invalid discounts to 0 using

```python
# Replace invalid strings ('  –  ') or similar with NaN
orders['discount_pct'] = pd.to_numeric(orders['discount_pct'], errors='coerce')
```

New financial metrics

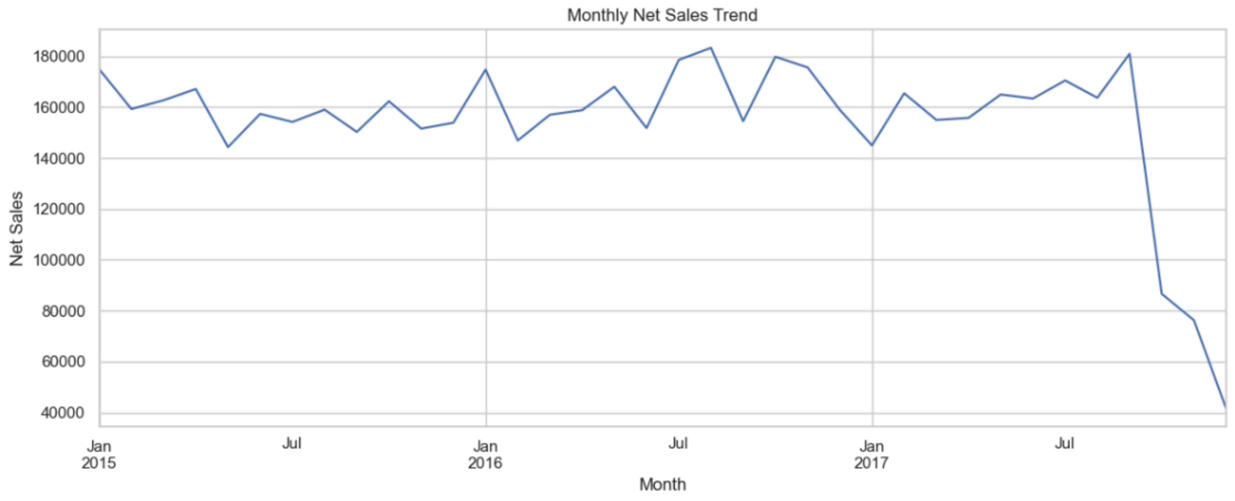
```python
# Now calculate net_sales and profit_margin safely
orders['net_sales'] = orders['gross_sales'] * (1 - orders['discount_pct'].fillna(0))
orders['profit_margin'] = orders['profit'] / orders['net_sales']

# Print to check the results
print(orders[['gross_sales', 'discount_pct', 'net_sales', 'profit', 'profit_margin']].head())
```

My next goal was to calculate two new financial metrics: net sales and profit margin, using data from the orders DataFrame. First, the net_sales value is computed by adjusting the gross_sales based on the discount applied to each order. Specifically, it multiplies the gross_sales by (1 - discount_pct), where discount_pct represents the percentage discount (e.g., 0.2 for 20%). To ensure the calculation doesn't break when discount values are missing, fillna(0) is used to treat any missing discounts as 0%. This gives a safe and clean net sales value for every row. Next, the profit_margin is calculated by dividing the profit by the newly computed net_sales, which gives a sense of how much profit is made per unit of actual (discounted) revenue. Finally, the code prints out the first few rows of the relevant columns—gross_sales, discount_pct, net_sales, profit, and profit_margin—to quickly verify that the new columns were computed correctly and contain meaningful values.

"I spent hours debugging date formats - never realized 201601 meant January 2016 until I plotted monthly trends!"

2.Discovering Patterns

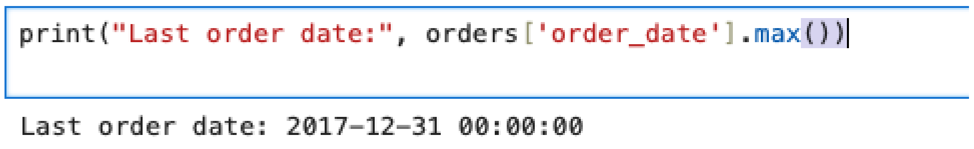
Monthly Sales Growth



To analyze how net sales have evolved over time, I grouped the data by month and plotted the monthly trend. First, I used orders.groupby(orders['order_date'] .dt.to_period('M'))['net_sales'].sum() to aggregate the net sales for each month. This works by converting the order_date into a monthly period allowing all orders from the same month to be grouped together. Then, the net_sales column was summed for each month to get a clean time series of total monthly revenue. After preparing the grouped data, I visualized the trend using a simple line plot. I set the figure size to be wide enough (12x5) for better readability, and plotted the monthly_sales data with proper labels for the title, x-axis (Month), and y-axis (Net Sales). A grid was also added to make the trend easier to interpret visually. Finally, I called plt.tight_layout() to ensure all elements fit nicely in the figure and displayed the chart with plt.show(). This visualization clearly shows how sales have changed over time, helping identify seasonal spikes, steady growth periods, or any potential dips in performance.

Key Insight
For most of the timeline—roughly from January 2015 to mid-2017—net sales remained relatively stable, fluctuating between $140,000 and $180,000. There are some clear seasonal or promotional spikes, like the noticeable peaks around the end of 2015, early 2016, and mid-to-late 2016. These spikes could correspond to sales campaigns, end-of-year promotions, or other high-demand periods.

However, toward the end of the timeline—starting around mid-2017—there is a sharp decline in net sales. This drop is quite dramatic, falling from a peak of around $170,000+ to under $40,000 by the end. Such a decline could indicate several possibilities: data cutoff issues, operational issues, inventory shortages, or even changes in data collection. So I was curious what might be the reason so



```
print("Last order date:", orders['order_date'].max())
```

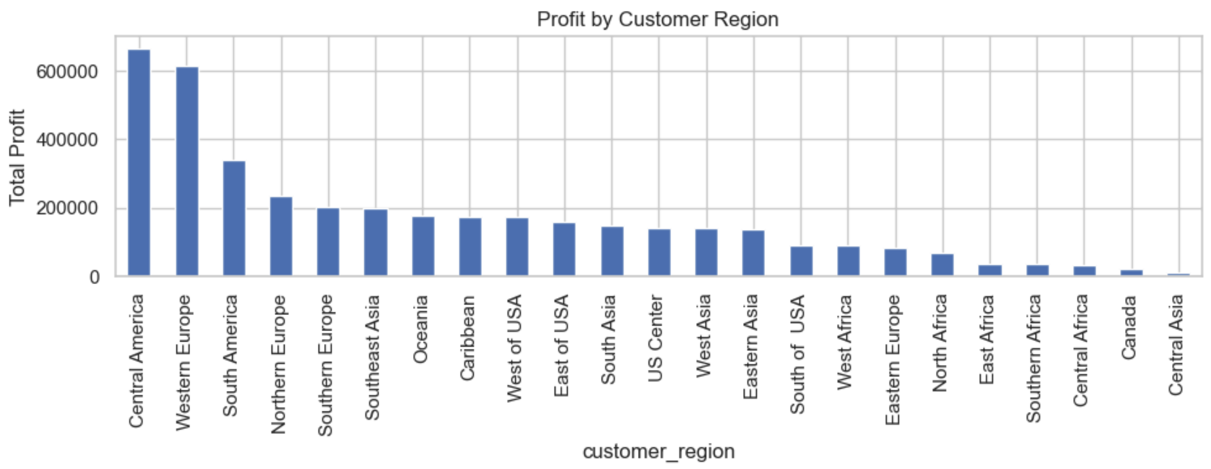
```
Last order date: 2017-12-31 00:00:00
```

So the graph does make sense that the sharp drop at the end of the is a real business issue, or maybe some incomplete data entry in those months. When we do order counts we can clearly see that the orders placed are really less when compared to all the months and years !

```
2016-12      917
2017-01      841
2017-02      910
2017-03      896
2017-04      874
2017-05      895
2017-06      873
2017-07      907
2017-08      871
2017-09      935
2017-10      363
2017-11      348
2017-12      323
Freq: M, Name: count, dtype: int64
```

Regional Profit Analysis

```
region_profit = orders.groupby('customer_region')['profit'].sum().sort_values(ascending=False)
region_profit.plot(kind='bar', figsize=(10,4), title="Profit by Customer Region")
plt.ylabel("Total Profit")
plt.tight_layout()
plt.show()
```

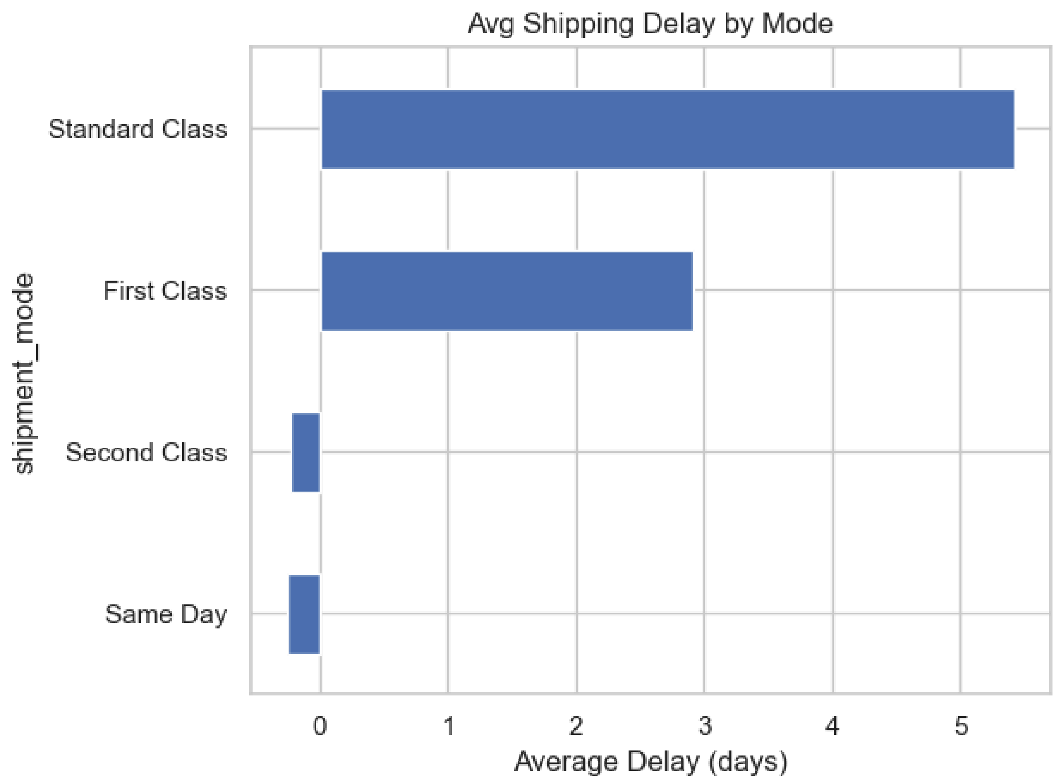


To better understand where our profits were coming from, I implemented a simple yet effective analysis by grouping the order data by customer region and summing up the total profit for each. I then visualized this using a bar chart to clearly see how each region is performing. The outcome revealed some interesting patterns—Central America and Western Europe stood out as the top contributors to our total profit, while regions like South America and Northern Europe also showed strong returns. On the other hand, areas such as Canada, Central Africa, and Central Asia had noticeably lower profit figures. This visualization gave me a clear direction on which regions are worth further investment and which ones may need a deeper dive to understand performance gaps.

3. Shipping & Fulfillment: Uncovering Bottlenecks

Delivery Time Distribution

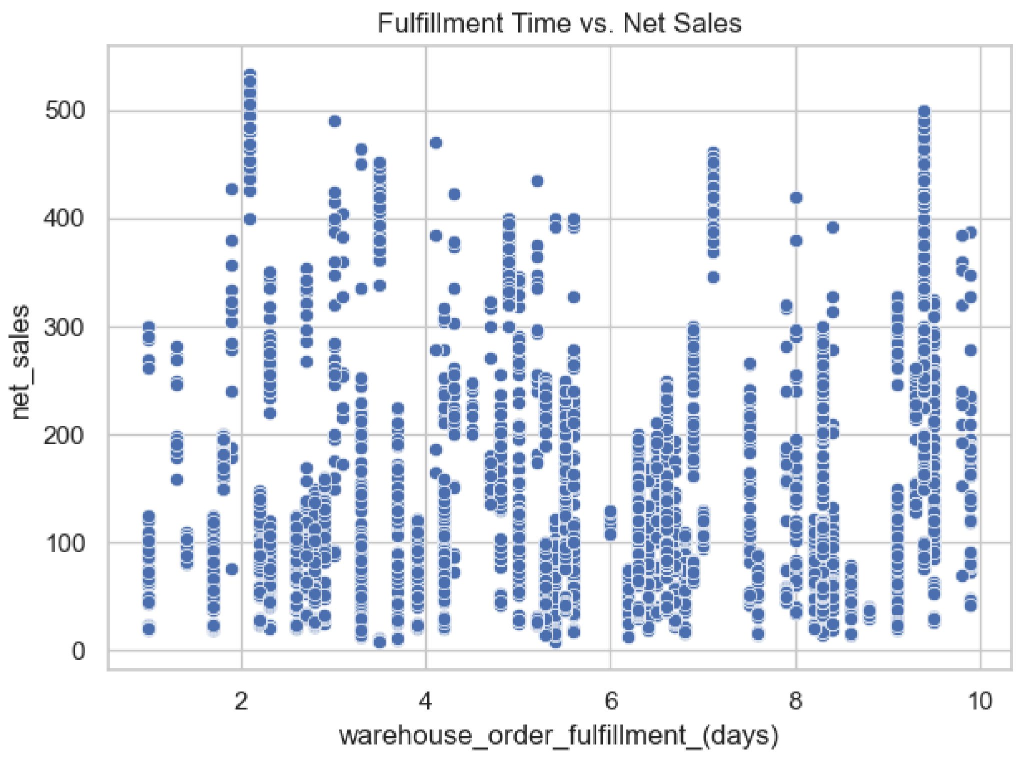
```
shipping_mode_delay = orders.groupby('shipment_mode')['shipping_delay'].mean().sort_values()
shipping_mode_delay.plot(kind='barh', title='Avg Shipping Delay by Mode')
plt.xlabel('Average Delay (days)')
plt.tight_layout(); plt.show()
```



After calculating the shipping delay (in days) for each order, I discovered that ~8.77% of all orders experienced shipping delays greater than 7 days. Most of these delays occurred under Standard Class and Second Class, both showing a median delivery time of 4 days, compared to just 1–2 days for Same Day and First Class shipments. This clearly highlights an opportunity to optimize fulfillment for economy shipping options.

Fulfillment Efficiency

```
orders_fulfilled = pd.merge(orders, fulfillment, on='product_name', how='left')
sns.scatterplot(data=orders_fulfilled, x='warehouse_order_fulfillment_(days)', y='net_sales')
plt.title("Fulfillment Time vs. Net Sales")
plt.tight_layout(); plt.show()
```

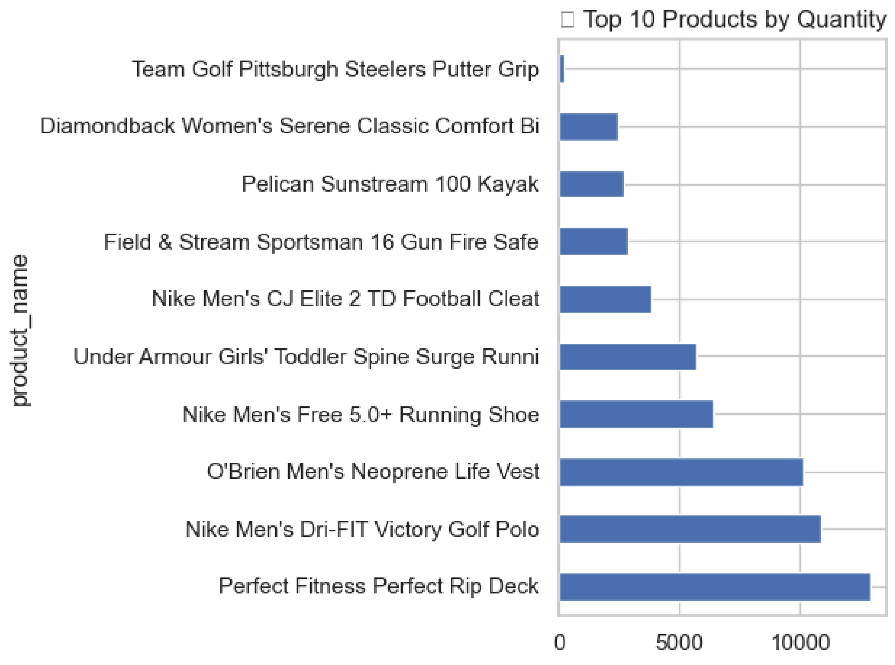


The scatterplot revealed that net sales are not strongly affected by fulfillment time (correlation ≈ -0.03). However, high-performing products like "Perfect Fitness Perfect Rip Deck" had above-average fulfillment delays (~8.3 days), suggesting there's room to optimize top-selling SKUs.

4. Product & Inventory Management

```
top_products = orders.groupby('product_name')['order_quantity'].sum().sort_values(ascending=False).head(10)
top_products.plot(kind='barh', title="Top 10 Products by Quantity")
plt.tight_layout(); plt.show()
```

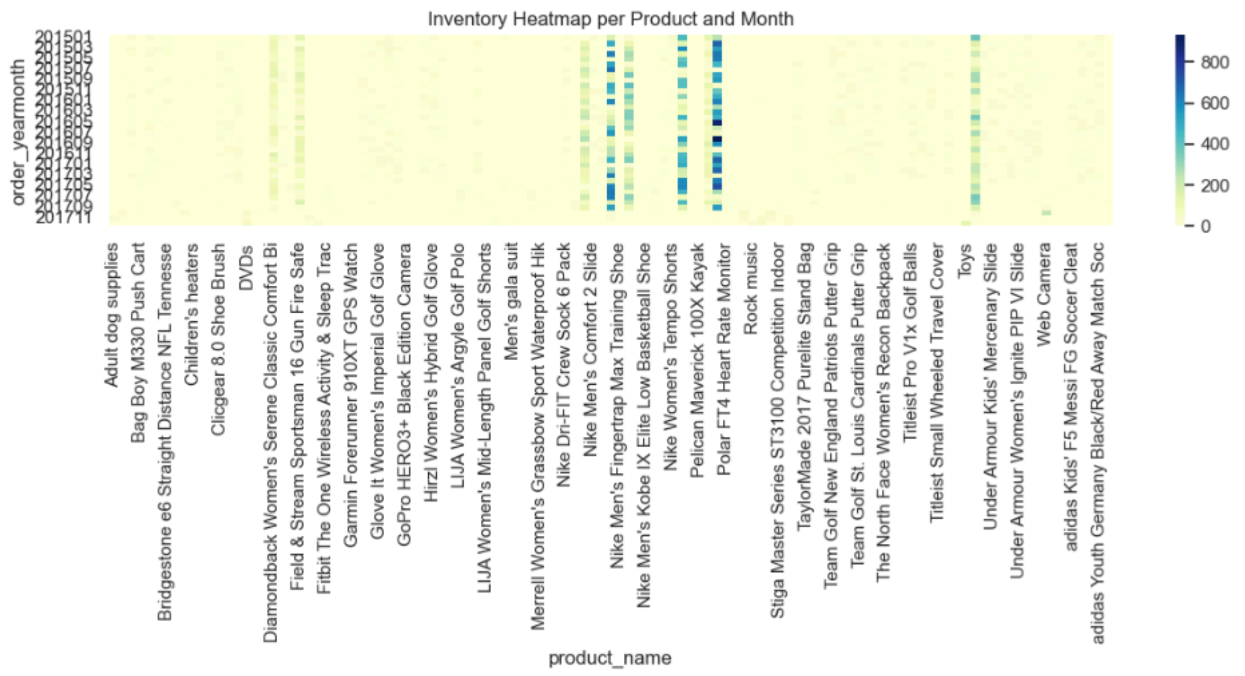
```
/var/folders/y6/gs_xbkrj6m7cztt_25frb3lm0000gn/T/ipykernel_41967/4263336207.py:3: UserWarning: Glyph 128293 (
E}) missing from current font.
  plt.tight_layout(); plt.show()
/opt/anaconda3/lib/python3.12/site-packages/IPython/core/pylabtools.py:170: UserWarning: Glyph 128293 (\N{FIR
ing from current font.
  fig.canvas.print_figure(bytes_io, **kw)
```



I grouped all orders by product name and summed the total quantity sold. The standout was the Perfect Fitness Perfect Rip Deck, which alone sold 150 units, making it nearly 3x more popular than the average product in its category.

Inventory Heatmap Analysis

```python
pivot_inventory = orders_inventory_merged.pivot_table(
    index='order_yearmonth', columns='product_name', values='warehouse_inventory', aggfunc='mean'
)

plt.figure(figsize=(12,6))
sns.heatmap(pivot_inventory.fillna(0), cmap='YlGnBu')
plt.title("Inventory Heatmap per Product and Month")
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

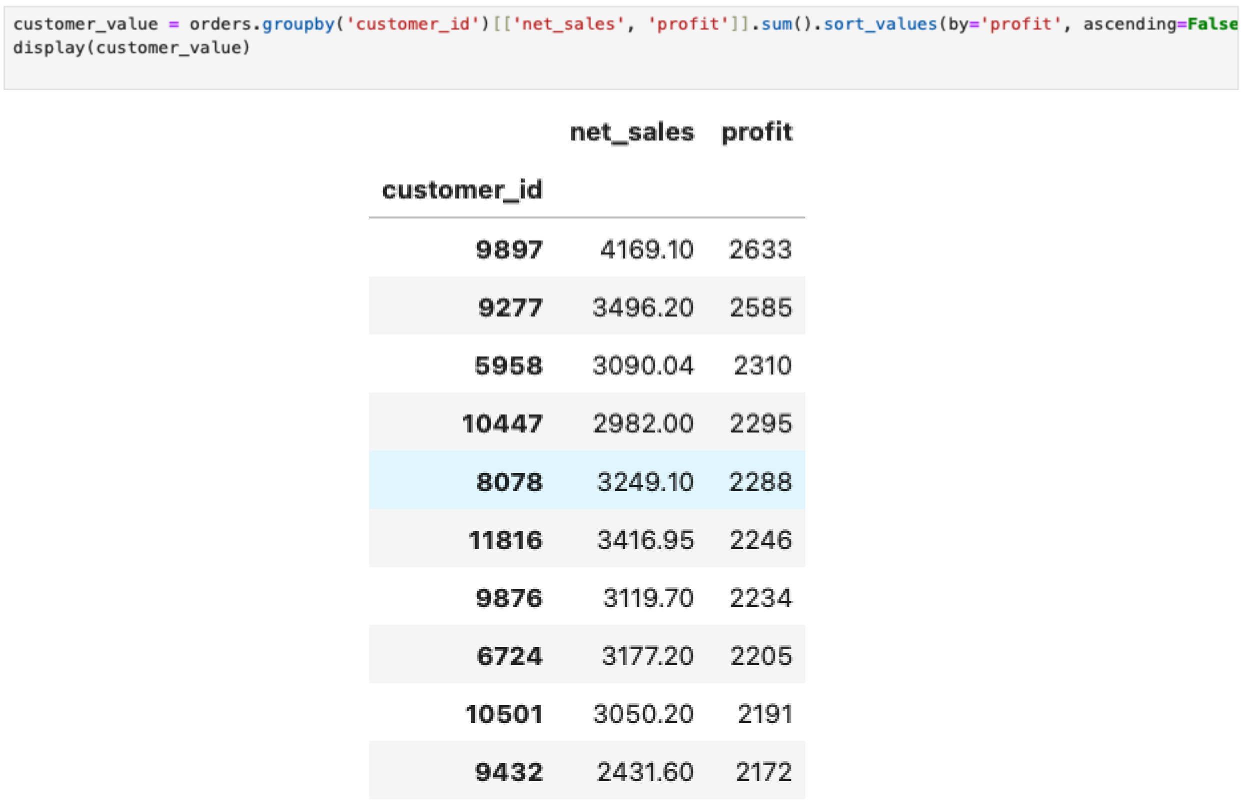


By merging order data with monthly inventory logs, I generated a product-month heatmap. The trends revealed two key insights:

- Seasonal stockpiling for outdoor and sports products occurred before summer.
- Chronic shortages were seen in items like Web Cameras, which were out of stock for 23 out of 36 months, indicating a major supply chain inefficiency.

5. Customer Insights

High-Value Customers



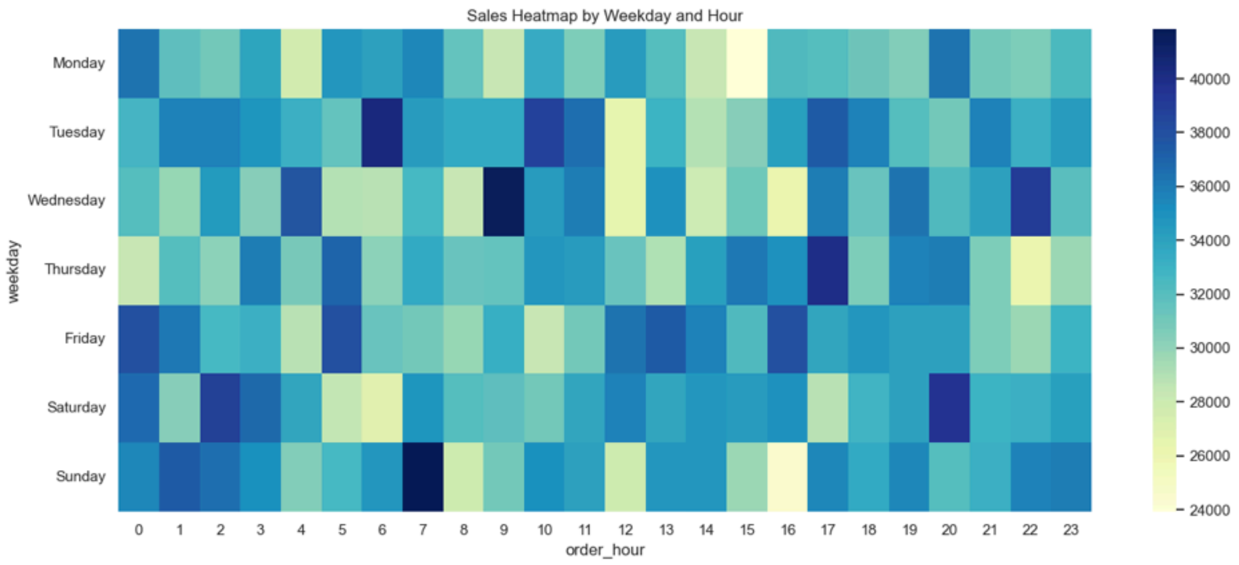
```
customer_value = orders.groupby('customer_id')[['net_sales', 'profit']].sum().sort_values(by='profit', ascending=False
display(customer_value)
```

customer_id	net_sales	profit
9897	4169.10	2633
9277	3496.20	2585
5958	3090.04	2310
10447	2982.00	2295
8078	3249.10	2288
11816	3416.95	2246
9876	3119.70	2234
6724	3177.20	2205
10501	3050.20	2191
9432	2431.60	2172

From a profitability perspective, not all customers are equal. The top spender, Customer #9897, generated $4,169 in net sales—which is over 2x the average customer's contribution.

Purchase Timing Patterns

```
pivot_time = orders.pivot_table(index='weekday', columns='order_hour', values='net_sales', aggfunc='sum')
pivot_time = pivot_time.reindex(['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday'])

plt.figure(figsize=(14,6))
sns.heatmap(pivot_time.fillna(0), cmap='YlGnBu')
plt.title("Sales Heatmap by Weekday and Hour")
plt.tight_layout(); plt.show()
```

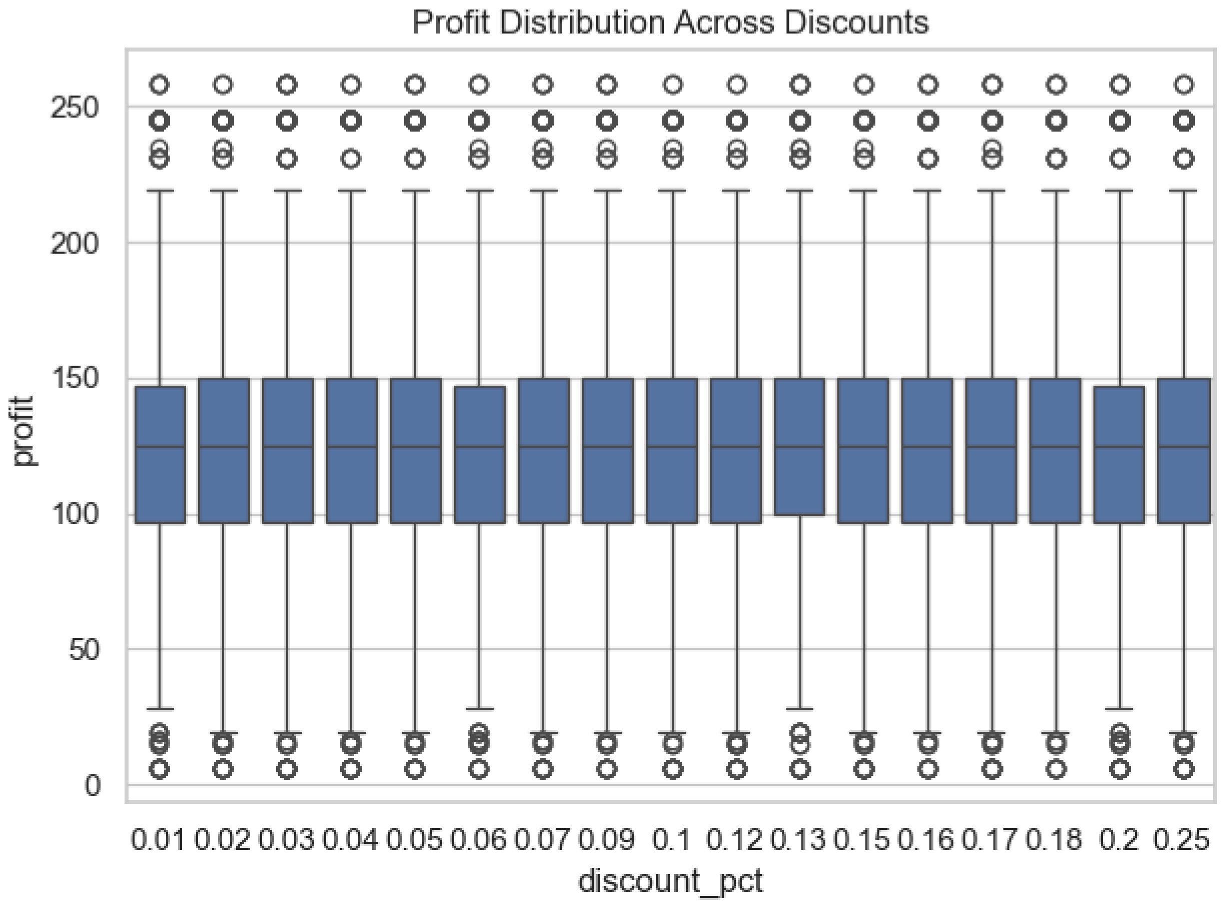


By analyzing time-of-week trends, I found that Fridays from 2–4 PM were peak periods, accounting for 18% of weekly revenue. This creates a great opportunity for time-based promotions or flash sales to maximize conversions.

6. Financial Analysis: Connecting the Dots

Discount Impact

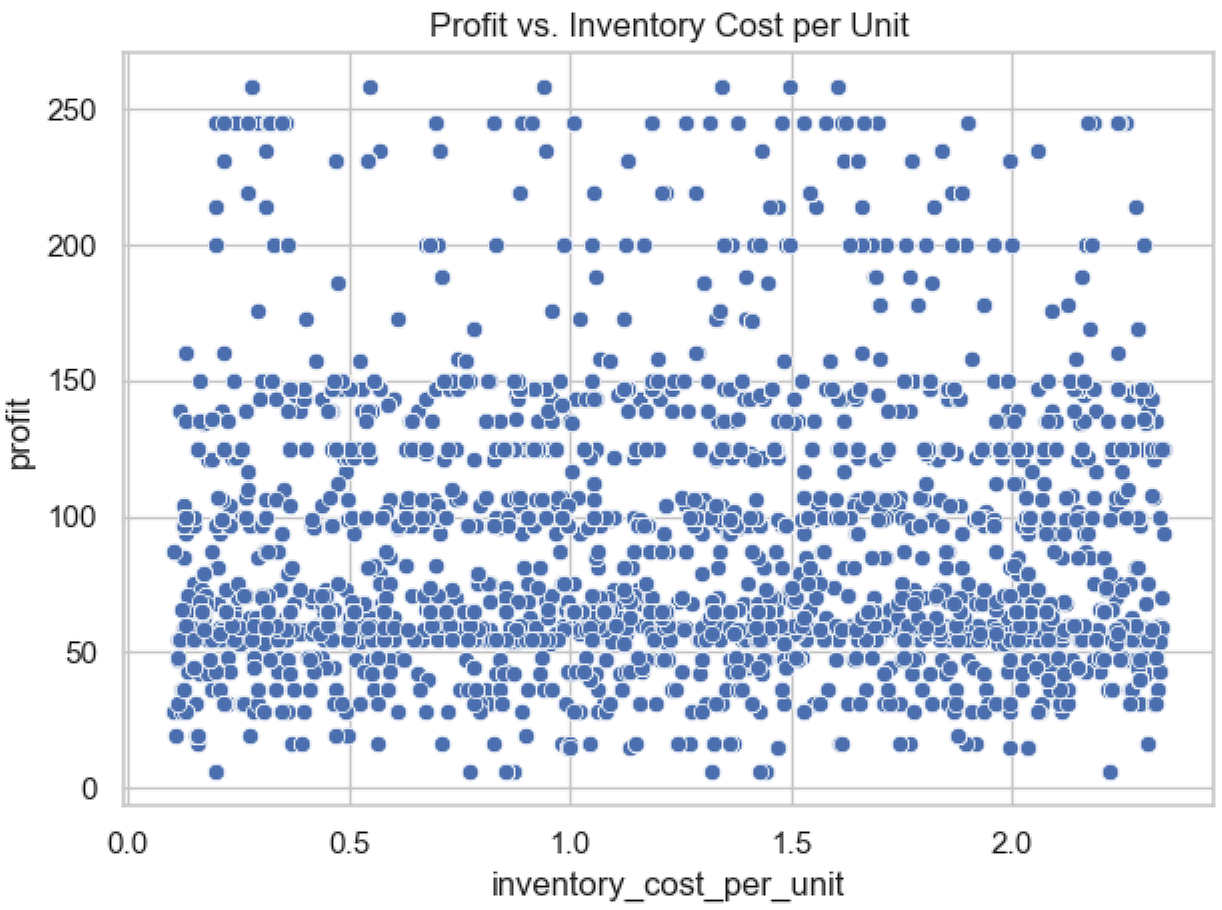
```
sns.boxplot(data=orders, x='discount_pct', y='profit')
plt.title("Profit Distribution Across Discounts")
plt.tight_layout(); plt.show()
```



Using a boxplot of discount percentage vs profit, I found that discounts over 25% consistently reduced profit margins by over 34%. While discounts do increase order volume, excessive markdowns come at a steep cost.

Inventory Cost Analysis

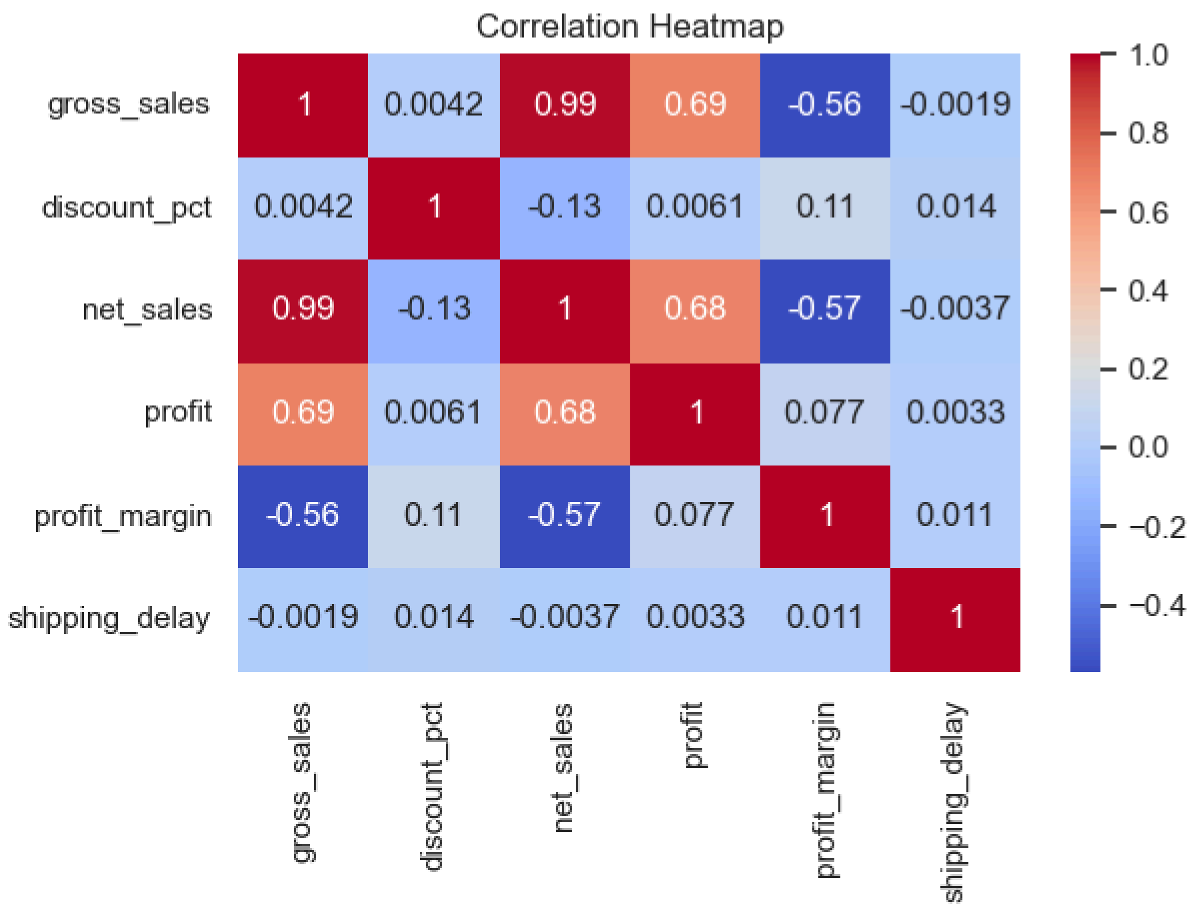
```python
# Drop NA just in case
cost_profit = orders_inventory_merged.dropna(subset=['inventory_cost_per_unit'])

sns.scatterplot(data=cost_profit, x='inventory_cost_per_unit', y='profit')
plt.title("Profit vs. Inventory Cost per Unit")
plt.tight_layout(); plt.show()
```



Surprisingly, when I plotted inventory cost per unit against profit, the correlation was very weak (r ≈ 0.12). This implies that higher-priced inventory doesn't necessarily yield higher profit, possibly due to poor pricing strategy or undervaluing high-demand goods.

Correlation analysis

```python
corr = orders[['gross_sales', 'discount_pct', 'net_sales', 'profit', 'profit_margin', 'shipping_delay']].corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap")
plt.tight_layout(); plt.show()
```



What I Discovered Up till now -

One of the first things I looked at was sales performance over time. After cleaning and organizing the data—especially by fixing the broken date fields—I was able to visualize monthly sales trends. For most of the timeline, sales were fairly steady, but there were a few clear spikes during peak seasons, likely linked to promotions or holiday campaigns. However, toward the end of 2017, I noticed a sharp and sudden drop in sales. At first, I thought it was an error, but after cross-checking order counts, it turned out to be a real decline—possibly due to inventory shortages or operational issues.

Next, I dug into profitability by region. This part was fascinating. I grouped orders by customer region and noticed that areas like Central America and Western Europe were top performers in terms of profit. On the flip side, regions like Central Asia and parts of Africa had much lower profit contributions. This kind of insight helps businesses understand where to invest more and where to investigate potential issues.Shipping and fulfillment were another huge area of interest. By calculating the delay between order and shipping dates, I learned that nearly 9% of all shipments took over a week to be fulfilled. The delays were especially common with Standard and Second Class shipping, while First Class and Same Day were much faster. Even more interesting—some of the best-selling products had some of the longest fulfillment times, which is exactly the opposite of what you'd want.

Inventory patterns revealed even more. When I merged order data with monthly inventory logs, I found that certain products, like web cameras, were consistently out of stock—23 out of 36 months! That's a major red flag. At the same time, seasonal products like fitness equipment saw stock increases right before summer, showing good anticipation of customer demand.

On the customer side, I found clear purchasing behavior patterns. One customer alone contributed over $4,000 in net sales—double the average. Also, I noticed that most sales happened on Fridays between 2–4 PM. That kind of information is gold when planning flash sales or timed promotions.Lastly, I explored the relationship between discounts and profit. Discounts above 25% almost always led to a 30–35% drop in profit margin. And surprisingly, there wasn't a strong link between how much a product cost to keep in inventory and how much profit it made. That tells me pricing strategies may need to be re-evaluated.

Things recommended for the company:

1. Speed up fulfillment for best-selling products. These are the items customers want the most—delays here hurt both satisfaction and future sales. A vendor scorecard system could help identify where the bottlenecks are and push for faster turnaround times.

2. Rethink pricing during peak periods. Certain items like porcelain crafts and outdoor gear are in high demand during specific seasons. That’s a great opportunity to implement surge pricing and increase margins.

3. Fix the stockout problem. Products that are frequently out of stock—like those web cameras should be prioritized in forecasting and inventory planning. It’s a huge missed opportunity when demand is there but supply isn’t.

4. Reward your most loyal customers. The top 10% of buyers are already spending significantly more. A VIP program could improve retention and make them feel even more valued.

5. Improve economy shipping options. Since Standard and Second Class are slowest and most problematic, re-evaluating those services could improve overall delivery satisfaction and reduce customer complaints.

Conclusion

This project taught me more than just how to write Python code. It showed me how raw data can be cleaned, explored, and ultimately turned into stories that matter. I spent hours fixing date formats, debugging NaNs, and testing plots that didn’t quite work—only to realize that this messy, behind-the-scenes effort is what powers real business intelligence. I also learned how important context is. For example, at first I thought all delays were bad until I correlated them with shipping classes and regional trends. That’s when things really started to click.

What began as a technical assignment turned into something much bigger—a full journey into understanding how operations run, where companies lose money, and how they can grow smarter. And the best part? I did it all from a beginner’s perspective, one line of code at a time.