superstore_case

December 7, 2025

[1]:
```python
import pandas as pd
import numpy as np
file_path ="C:/Users/anush/Downloads/superstore_eda_v1-1724655032.csv"
df   = pd.read_csv(file_path)
```

[2]:
```python
df.shape
```

[2]:
```
(10014, 21)
```

[3]:
```python
df.head(5)
```

[3]:
```
   Row ID        Order ID Order Date Ship Date       Ship Mode Customer ID  \
0       1  CA-2016-152156   08/11/16  11/11/16    Second Class    CG-12520
1       2  CA-2016-152156   08/11/16  11/11/16    Second Class    CG-12520
2       3  CA-2016-138688   12/06/16  16/06/16    Second Class    DV-13045
3       4  US-2015-108966   11/10/15  18/10/15  Standard Class    SO-20335
4       5  US-2015-108966   11/10/15  18/10/15  Standard Class    SO-20335

     Customer Name    Segment        Country             City  …  \
0      Claire Gute   Consumer  United States        Henderson  …
1      Claire Gute   Consumer  United States        Henderson  …
2  Darrin Van Huff  Corporate  United States      Los Angeles  …
3   Sean O'Donnell   Consumer  United States  Fort Lauderdale  …
4   Sean O'Donnell   Consumer  United States  Fort Lauderdale  …

  Postal Code  Region       Product ID         Category Sub-Category  \
0       42420   South  FUR-BO-10001798        Furniture    Bookcases
1       42420   South  FUR-CH-10000454        Furniture       Chairs
2       90036    West  OFF-LA-10000240  Office Supplies       Labels
3       33311   South  FUR-TA-10000577        Furniture       Tables
4       33311   South  OFF-ST-10000760  Office Supplies      Storage

                                        Product Name Sales Price  Quantity  \
0                  Bush Somerset Collection Bookcase    261.9600       2.0
1  Hon Deluxe Fabric Upholstered Stacking Chairs,…    731.9400       3.0
2  Self-Adhesive Address Labels for Typewriters b…     14.6200       2.0
3      Bretford CR4500 Series Slim Rectangular Table    957.5775       5.0
4                     Eldon Fold 'N Roll Cart System     22.3680       2.0
```

```
   Discount    Profit
0      0.00   41.9136
1      0.00  219.5820
2      0.00    6.8714
3      0.45 -383.0310
4      0.20    2.5164

[5 rows x 21 columns]
```

[4]:
```python
#Handling Duplicates:

df = df.apply(lambda x: x.str.strip() if x.dtype == "object" else x)
df = df.apply(lambda x: x.str.lower() if x.dtype == "object" else x)

print("Duplicate rows before:", df.duplicated().sum())

rows_before = df.shape[0]

df = df.drop_duplicates()

rows_after = df.shape[0]

print("\nAfter cleaning:")
print("Rows before:", rows_before)
print("Rows after:", rows_after)
print("Rows removed:", rows_before - rows_after)
```

```
Duplicate rows before: 17

After cleaning:
Rows before: 10014
Rows after: 9997
Rows removed: 17
```

[5]:
```python
df.dtypes
```

[5]:
```
Row ID             int64
Order ID          object
Order Date        object
Ship Date         object
Ship Mode         object
Customer ID       object
Customer Name     object
Segment           object
Country           object
City              object
State             object
```

```
Postal Code          int64
Region              object
Product ID          object
Category            object
Sub-Category        object
Product Name        object
Sales Price        float64
Quantity           float64
Discount           float64
Profit             float64
dtype: object
```

[6]:
```python
df.describe()
```

[6]:
```
            Row ID   Postal Code   Sales Price     Quantity     Discount  \
count  9997.000000   9997.000000   9997.000000  9979.000000  9997.000000
mean   4996.992198  55192.055117    229.798602     3.790861     0.156216
std    2885.688679  32062.477713    623.164427     2.226648     0.206422
min       1.000000   1040.000000    -31.500000     1.000000     0.000000
25%    2498.000000  23223.000000     17.240000     2.000000     0.000000
50%    4997.000000  56560.000000     54.480000     3.000000     0.200000
75%    7496.000000  90008.000000    209.940000     5.000000     0.200000
max    9994.000000  99301.000000  22638.480000    14.000000     0.800000

            Profit
count  9997.000000
mean     28.650525
std     234.225264
min   -6599.978000
25%       1.731000
50%       8.662000
75%      29.364000
max    8399.976000
```

[7]:
```python
# Correct date format:-

df["Ship Date"] = pd.to_datetime(df["Ship Date"], errors='coerce')
df["Order Date"] = pd.to_datetime(df["Order Date"],errors='coerce')
print(df[['Order Date', 'Ship Date']].dtypes)
```

```
Order Date    datetime64[ns]
Ship Date     datetime64[ns]
dtype: object
```

```
C:\Users\anush\AppData\Local\Temp\ipykernel_22460\3538689358.py:3: UserWarning:
Could not infer format, so each element will be parsed individually, falling
back to `dateutil`. To ensure parsing is consistent and as-expected, please
specify a format.
```

```
  df["Ship Date"] = pd.to_datetime(df["Ship Date"], errors='coerce')
C:\Users\anush\AppData\Local\Temp\ipykernel_22460\3538689358.py:4: UserWarning:
Could not infer format, so each element will be parsed individually, falling
back to `dateutil`. To ensure parsing is consistent and as-expected, please
specify a format.
  df["Order Date"] = pd.to_datetime(df["Order Date"],errors='coerce')
```

[8]:

```
# Extract the year from the Order ID and compare it with the year in Order Date.
 ↪ Correct any inconsistencies.

df['Order_Year_From_ID'] = df["Order ID"].str.split('-').str[1]
df['Order_Year_From_ID'] = pd.to_numeric(df['Order_Year_From_ID'] , errors =␣
 ↪"coerce")
df['Order_Year_From_Date'] = df['Order Date'].dt.year

mismatch_mask = df['Order_Year_From_ID'] != df['Order_Year_From_Date']
print("Number of mismatched rows:", mismatch_mask.sum())

df.loc[mismatch_mask, 'Order Date'] = df.loc[mismatch_mask].apply(lambda row:␣
 ↪row['Order Date'].replace(year=row['Order_Year_From_ID'])
    if pd.notnull(row['Order Date']) else row['Order Date'],axis=1)
```

```
Number of mismatched rows: 40
```

[9]:

```
# Negative values in Sales price

print(df[df["Sales Price"] < 0].shape[0])
df= df[df["Sales Price"] >= 0]
```

```
4
```

[10]:

```
df.describe()
```

[10]:

```
              Row ID                    Order Date  \
count  9993.000000                          9993
mean   4996.022716  2016-04-11 14:27:37.880516352
min       1.000000            2014-01-02 00:00:00
25%    2498.000000            2015-05-01 00:00:00
50%    4996.000000            2016-05-30 00:00:00
75%    7494.000000            2017-04-09 00:00:00
max    9994.000000            2017-12-31 00:00:00
std    2884.906697                           NaN

                           Ship Date   Postal Code  Sales Price     Quantity  \
count                           9993   9993.000000  9993.000000  9975.000000
mean   2016-04-22 21:14:17.039928064  55196.450515   229.897388     3.790376
min              2014-01-07 00:00:00   1040.000000     0.444000     1.000000
25%              2015-05-07 00:00:00  23223.000000    17.280000     2.000000
```

```
50%            2016-06-12 00:00:00  56560.000000     54.528000     3.000000
75%            2017-04-29 00:00:00  90008.000000    209.970000     5.000000
max            2018-05-01 00:00:00  99301.000000  22638.480000    14.000000
std                            NaN  32063.943256    623.269553     2.226107

          Discount       Profit  Order_Year_From_ID  Order_Year_From_Date
count  9993.000000  9993.000000         9993.000000           9993.000000
mean      0.156158    28.662956         2015.723006           2015.720004
min       0.000000 -6599.978000         2014.000000           1999.000000
25%       0.000000     1.731000         2015.000000           2015.000000
50%       0.200000     8.671000         2016.000000           2016.000000
75%       0.200000    29.364000         2017.000000           2017.000000
max       0.800000  8399.976000         2017.000000           2029.000000
std       0.206356   234.271064            1.124072              1.199837
```

[11]:
```
df.isnull().sum()
```

[11]:
```
Row ID                   0
Order ID                 0
Order Date               0
Ship Date                0
Ship Mode               98
Customer ID              0
Customer Name            0
Segment                  0
Country                  0
City                     0
State                    0
Postal Code              0
Region                   0
Product ID               0
Category                 0
Sub-Category             0
Product Name             0
Sales Price              0
Quantity                18
Discount                 0
Profit                   0
Order_Year_From_ID       0
Order_Year_From_Date     0
dtype: int64
```

[12]:
```
# # Imputation of Missing Values:
# 1. Impute missing values in the Ship Mode column using the calculated Days to␣
↪Ship column.
# 2. Calculate Days to Ship as the difference between Ship Date and Order Date.␣
↪If Days to Ship is 0,
```

```
# set Ship Mode to "Same Day"; if it is 7. set Ship Mode to "Standard Class"
# 3. Impute missing values in the Quantity column using a method of your choice.
↪ Print the rationale for selecting the method for imputation.
df['Days_to_Ship'] = (df['Ship Date'] - df['Order Date']).dt.days
```

[13]:
```
# Fill missing Ship Mode values using Days_to_Ship logic
df.loc[(df['Ship Mode'].isna()) & (df['Days_to_Ship'] == 0), 'Ship Mode] =␣
↪'Same Day'
df.loc[(df['Ship Mode'].isna()) & (df['Days_to_Ship'] == 7), 'Ship Mode'] =␣
↪'Standard Class'
df['Ship Mode'] = df['Ship Mode'].fillna(df['Ship Mode'].mode()[0])
```

[14]:
```
# Impute missing Quantity using median (robust choice)
median_quantity = df['Quantity'].median()
df['Quantity'] = df['Quantity'].fillna(median_quantity)
```

[15]:
```
df.isnull().sum()
```

[15]:
```
Row ID                 0
Order ID               0
Order Date             0
Ship Date              0
Ship Mode              0
Customer ID            0
Customer Name          0
Segment                0
Country                0
City                   0
State                  0
Postal Code            0
Region                 0
Product ID             0
Category               0
Sub-Category           0
Product Name           0
Sales Price            0
Quantity               0
Discount               0
Profit                 0
Order_Year_From_ID     0
Order_Year_From_Date   0
Days_to_Ship           0
dtype: int64
```

[16]:
```
# 5. Data Masking and String Handling:
# 1. Drop the Customer Name column to protect Personal Identifiable Information␣
↪(PiI).
```

```
# 2. Create a new column called Customer Name Masked, containing only the␣
↪initials of the customer name.
df["Customer Name Masked"] = df["Customer Name"].apply(lambda x: ''.
↪join([name[0].upper() for name in x.split()]))
df.drop(columns=['Customer Name'], inplace=True)

# Convert the Postal Code column from numeric to text format, ensuring all␣
↪codes are 5 characters long. Add a leading '0' where necessary.
df['Postal Code'] = df['Postal Code'].astype(str).str.zfill(5)
```

[17]:
```
# 6. Data Type Conversion:
# 1. Convert the Quantity and Sales Price columns from strings to their␣
↪appropriate numeric types (int and float, respectively).

df['Quantity'] = pd.to_numeric(df['Quantity'], errors='coerce').astype('Int64')
df['Sales Price'] = pd.to_numeric(df['Sales Price'], errors='coerce').
↪astype('float')
```

[18]:
```
# 7. Handling Inconsistent Categorical Data:
# 1. Clean the State column by replacing abbreviations with full state names (e.
↪g.. "CA" should be changed to "California").
# You may need to research state abbreviations online to ensure all entries are␣
↪corrected consistently.

print(df['State'].unique())
state_mapping = {
    'ca': 'california',
    'tx': 'texas',
    'ny': 'new york',
    'nj': 'new jersey',
    'wa': 'washington',
    'wa\\': 'washington'}

df["State"] = df["State"].replace(state_mapping)
df['State'] = df['State'].str.strip().str.title()
```

```
['kentucky' 'california' 'florida' 'north carolina' 'washington' 'texas'
 'wisconsin' 'utah' 'nebraska' 'pennsylvania' 'illinois' 'minnesota'
 'michigan' 'delaware' 'indiana' 'new york' 'arizona' 'virginia'
 'tennessee' 'tx' 'alabama' 'south carolina' 'oregon' 'colorado' 'iowa'
 'ohio' 'missouri' 'oklahoma' 'new mexico' 'louisiana' 'connecticut'
 'new jersey' 'massachusetts' 'georgia' 'nevada' 'rhode island'
 'mississippi' 'arkansas' 'montana' 'ca' 'new hampshire' 'maryland'
 'district of columbia' 'wa\\' 'nj' 'kansas' 'vermont' 'maine'
 'south dakota' 'idaho' 'north dakota' 'wyoming' 'west virginia' 'ny']
```

[19]:
```
# 8. Feature Engineering:

# 1. Original Price: The price before any discount is applied.
df["Original Price"] = df["Sales Price"] / (1 - df["Discount"])

# 2. Total Sales: The total revenue generated by multiplying the Sales Price by␣
↪Quantity.
df["Total Sales"] = df["Sales Price"] * df["Quantity"]

# 3. Total Profit: The total profit earned by multiplying the Profit by␣
↪Quantity.
df["Total Profit"] = df["Profit"] * df["Quantity"]

# 4. Discount Price: The amount of discount applied, calculated based on the␣
↪Original Price and Discount.
df["Discount Price"] = df["Original Price"] * df["Discount"]

# 5. Total Discount: The total discount value for the quantity sold.
df["Total Discount"] = df["Discount Price"] * df["Quantity"]
```

[20]:
```
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

[21]:
```
df.head(5)
```

[21]:
```
   Row ID         Order ID Order Date  Ship Date       Ship Mode Customer ID  \
0       1  ca-2016-152156 2016-08-11 2016-11-11    second class    cg-12520
1       2  ca-2016-152156 2016-08-11 2016-11-11    second class    cg-12520
2       3  ca-2016-138688 2016-12-06 2016-06-16    second class    dv-13045
3       4  us-2015-108966 2015-11-10 2015-10-18  standard class    so-20335
4       5  us-2015-108966 2015-11-10 2015-10-18  standard class    so-20335

    Segment        Country             City       State Postal Code Region  \
0   consumer  united states        henderson    Kentucky       42420  south
1   consumer  united states        henderson    Kentucky       42420  south
2  corporate  united states      los angeles  California       90036   west
3   consumer  united states  fort lauderdale     Florida       33311  south
4   consumer  united states  fort lauderdale     Florida       33311  south

        Product ID         Category Sub-Category  \
0  fur-bo-10001798        furniture    bookcases
1  fur-ch-10000454        furniture       chairs
2  off-la-10000240  office supplies       labels
3  fur-ta-10000577        furniture       tables
4  off-st-10000760  office supplies      storage

                                        Product Name  Sales Price  Quantity  \
```

```
0                  bush somerset collection bookcase     261.9600         2
1  hon deluxe fabric upholstered stacking chairs,…     731.9400         3
2  self-adhesive address labels for typewriters b…      14.6200         2
3      bretford cr4500 series slim rectangular table     957.5775         5
4                         eldon fold 'n roll cart system      22.3680         2

   Discount    Profit  Order_Year_From_ID  Order_Year_From_Date  Days_to_Ship  \
0      0.00   41.9136                2016                  2016            92
1      0.00  219.5820                2016                  2016            92
2      0.00    6.8714                2016                  2016          -173
3      0.45 -383.0310                2015                  2015           -23
4      0.20    2.5164                2015                  2015           -23

  Customer Name Masked  Original Price  Total Sales  Total Profit  \
0                   CG          261.96       523.92       83.8272
1                   CG          731.94      2195.82       658.746
2                  DVH           14.62        29.24       13.7428
3                   SO         1741.05    4787.8875     -1915.155
4                   SO           27.96       44.736        5.0328

   Discount Price  Total Discount
0          0.0000             0.0
1          0.0000             0.0
2          0.0000             0.0
3        783.4725       3917.3625
4          5.5920          11.184
```

[22]:

```python
# 2. Create a new column Shipping Urgency based on Days to Ship:
def categorize_urgency(days):
    if days == 0:
        return 'Immediate'
    elif 1 <= days <= 3:
        return 'Urgent'
    else:
        return 'Standard'

df['Shipping Urgency'] = df['Days_to_Ship'].apply(categorize_urgency)
```

[23]:

```python
#3. Create a column that calculates days since last order
df = df.sort_values(['Customer ID', 'Order Date'])
df['Days Since Last Order'] = df.groupby('Customer ID')['Order Date'].diff().dt.
 ↪days
```

[24]:

```python
#4. Create a new dataset which stores the total sales, quantity and discount␣
 ↪per customer and then merge these back to the original dataset
customer_summary = df.groupby('Customer ID').agg({
    'Total Sales': 'sum',
```

```
    'Quantity': 'sum',
    'Discount': 'mean'
}).reset_index()

customer_summary.rename(columns={
    'Total Sales': 'Customer Total Sales',
    'Quantity': 'Customer Total Quantity',
    'Discount': 'Customer Avg Discount'
}, inplace=True)

df = df.merge(customer_summary, on='Customer ID', how='left')
```

[25]:

```
# Removing outlier

def remove_outliers(df, column):
    """
    Removes extreme outliers from a given column using the 3×IQR rule.
    Returns a cleaned DataFrame and prints the number of rows removed.
    """
    # Compute Q1 (25th percentile) and Q3 (75th percentile)
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1

    # Define bounds using 3*IQR
    lower_bound = Q1 - 3 * IQR
    upper_bound = Q3 + 3 * IQR

    # Track before removal
    initial_rows = df.shape[0]
    initial_order_ids = df['Order ID'].nunique()

    # Filter data within bounds
    df_clean = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

    # Track after removal
    final_rows = df_clean.shape[0]
    final_order_ids = df_clean['Order ID'].nunique()

    print(f"Outlier removal in '{column}':")
    print(f"  Rows removed: {initial_rows - final_rows}")
    print(f"  Distinct Order IDs affected: {initial_order_ids -␣
↪final_order_ids}")

    return df_clean
```

[26]:

```
df = remove_outliers(df, 'Sales Price')
```

```
Outlier removal in 'Sales Price':
  Rows removed: 668
  Distinct Order IDs affected: 158
```

[27]:
```python
# 10. Customer Segmentation and Analysis:
# 1. Calculate Customer Sales Quintile and Customer Profit Quintile based on
 ↪total sales and total profit per Customer ID.
# 3. Create a cross-grid (cross-tabulation) based on these two quintiles to
 ↪analyze the relationship between customer sales and profitability.

customer_summary = df.groupby('Customer ID').agg({'Total Sales': 'sum','Total
 ↪Profit': 'sum'}).reset_index()

customer_summary['Customer Sales Quintile'] = pd.qcut(customer_summary['Total
 ↪Sales'],q=5,labels=['Q1', 'Q2', 'Q3', 'Q4', 'Q5'])
customer_summary['Customer Profit Quintile'] = pd.qcut(customer_summary['Total
 ↪Profit'],q=5,labels=['Q1', 'Q2', 'Q3', 'Q4', 'Q5'])

cross_tab = pd.crosstab(customer_summary['Customer Sales
 ↪Quintile'],customer_summary['Customer Profit Quintile'],normalize='index')
print(cross_tab)
```

```
Customer Profit Quintile        Q1        Q2        Q3        Q4        Q5
Customer Sales Quintile
Q1                        0.245283  0.547170  0.201258  0.006289  0.000000
Q2                        0.196203  0.215190  0.398734  0.189873  0.000000
Q3                        0.145570  0.120253  0.202532  0.373418  0.158228
Q4                        0.215190  0.063291  0.120253  0.272152  0.329114
Q5                        0.201258  0.050314  0.075472  0.157233  0.515723
```

insights:- Strong Positive Relationship between Sales and Profit: The Q5–Q5 shows that around 51.6% of the highest sales customers are also in the highest profit group. thats means that our top spenders are also your most profitable customers.

Low Sales → Low Profit: Most Q1 (Low Sales) customers fall in Q1–Q2 (Low Profit) ranges — over 79% (0.245 + 0.547). This shows that customers who buy less also contribute little to profit

Middle Segments Show Mix: Quintiles Q2–Q3 are spread across profit quintiles — they include both low- and mid-profit customers. This indicates a mixed performance group

High Sales but Medium Profit (Q5–Q3 or Q5–Q4): A smaller portion around 23% of high sales customers fall in Q3–Q4 profit quintiles. These could be discount-driven buyers — high revenue but lower margins.

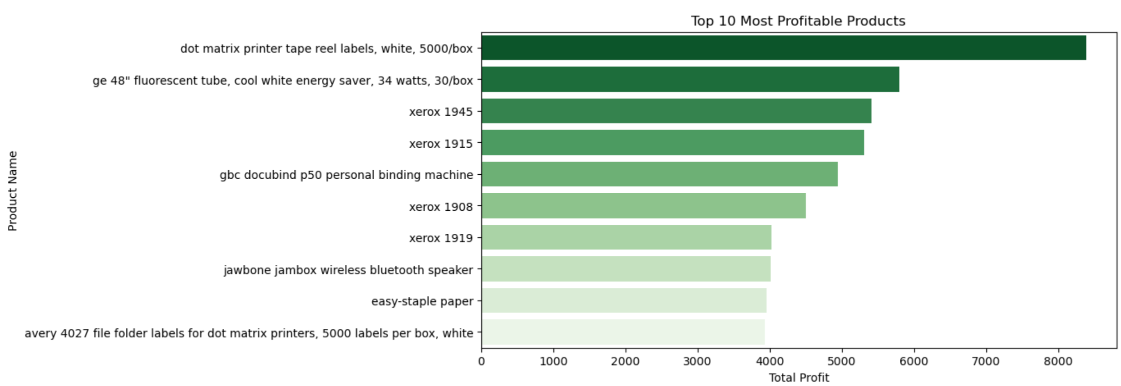
Business Insight: Focus retention efforts on Q5–Q5 customers — they drive both revenue and profit. Reassess pricing or discount strategies for high-sales but medium-profit customers. Identify ways to move Q3 and Q4 sales customers up into higher profit brackets by offering premium products or optimizing shipping/discount policies.

[ ]:

Final Dashboard:-

[28]:
```
# 1. Sales and Profit Analysis:

# 1. Top 10 Most Profitable Products: Use a bar chart to display the products␣
↪with the highest total profit
# 2. Top 10 Most Loss Making Products: Use a bar chart to display the products␣
↪with the highest total losses (negative profit).
# 3. Sales vs. Profit Correlation: Use a scatter plot to visualize the␣
↪correlation between Total Sales and Total Profit.
# Add a regression line to show the trend.
```

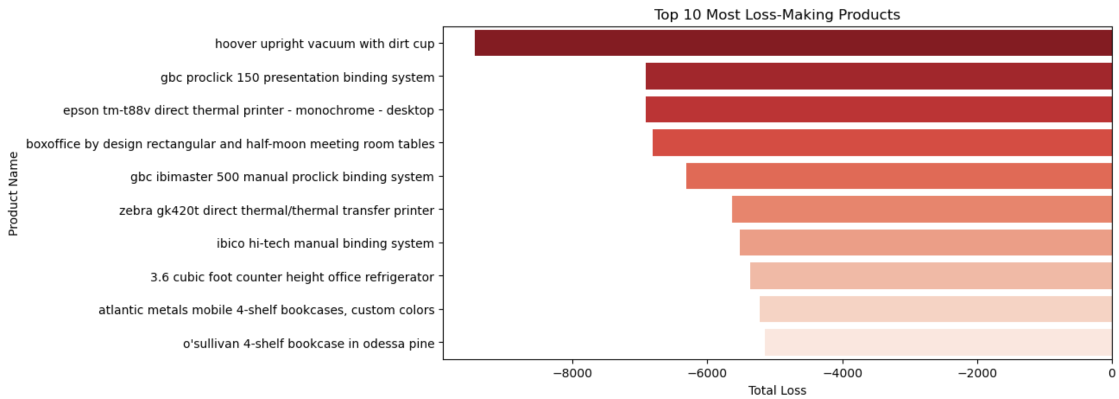
[29]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Top 10 most profitable products
top_profitable = (df.groupby('Product Name')['Total Profit'].sum().
↪sort_values(ascending=False).head(10))

plt.figure(figsize=(10, 5))
sns.barplot(x=top_profitable.values, y=top_profitable.index,hue =␣
↪top_profitable.index, palette='Greens_r', legend = False)
plt.title("Top 10 Most Profitable Products")
plt.xlabel("Total Profit")
plt.ylabel("Product Name")
plt.show()
```



These products contribute significantly to overall profit. Most of them belong to office supplies and stationery categories such as labels, binders, and papers. They have high sales volume and good profit margins, indicating steady customer demand and efficient pricing.

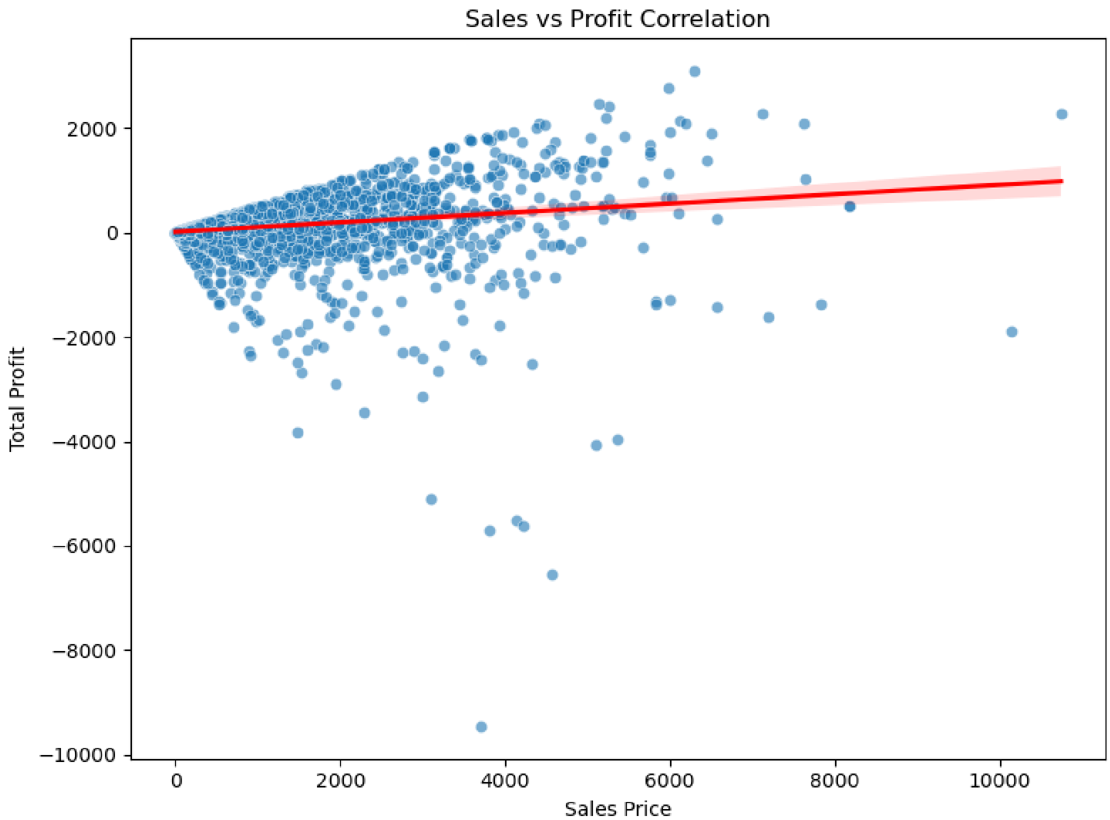
Actions:- They should be prioritized for promotion, inventory stocking, and marketing focus

```
[30]: top_loss = (df.groupby('Product Name')['Total Profit'].sum().
       ↪sort_values(ascending=True).head(10))
      plt.figure(figsize=(10, 5))
      sns.barplot(x=top_loss.values, y=top_loss.index,hue = top_loss.index,␣
       ↪palette='Reds_r', legend = False)
      plt.title("Top 10 Most Loss-Making Products")
      plt.xlabel("Total Loss")
      plt.ylabel("Product Name")
      plt.show()
```



These products generate negative profits every sale results in a loss. Many are office equipment and technology-related items like printers, binding machines, and box systems, which usually have higher costs or are sold under heavy discounts. Heavy discounts or promotional sales. High shipping or handling costs (especially for large/heavy products). Returns or warranty replacements increasing cost. Low demand, leading to inventory holding cost or clearance sales

```
[31]: plt.figure(figsize=(8,6))
      sns.scatterplot(data=df, x="Total Sales", y="Total Profit", alpha=0.6)
      sns.regplot(data=df, x="Total Sales", y="Total Profit", scatter=False,␣
       ↪color="red")
      plt.title("Sales vs Profit Correlation")
      plt.xlabel("Sales Price")
      plt.ylabel("Total Profit")
      plt.tight_layout()
      plt.show()
```



[32]:
```
correlation = df["Total Sales"].corr(df["Total Profit"])
print(f"Correlation between Sales and Profit: {correlation:.2f}")
```

```
Correlation between Sales and Profit: 0.23
```

The scatter plot shows a slightly positive correlation between Sales and Profit — meaning that generally, higher sales tend to bring higher profit. However, the relationship is not very strong (the points are widely spread). You can notice that some orders with very high sales still have low or negative profit. This usually happens when high discounts or shipping costs reduce profit margins. The red regression line slopes upward but slightly — indicating a weak positive trend between total sales and total profit There are some points far below the line (negative profit despite good sales).

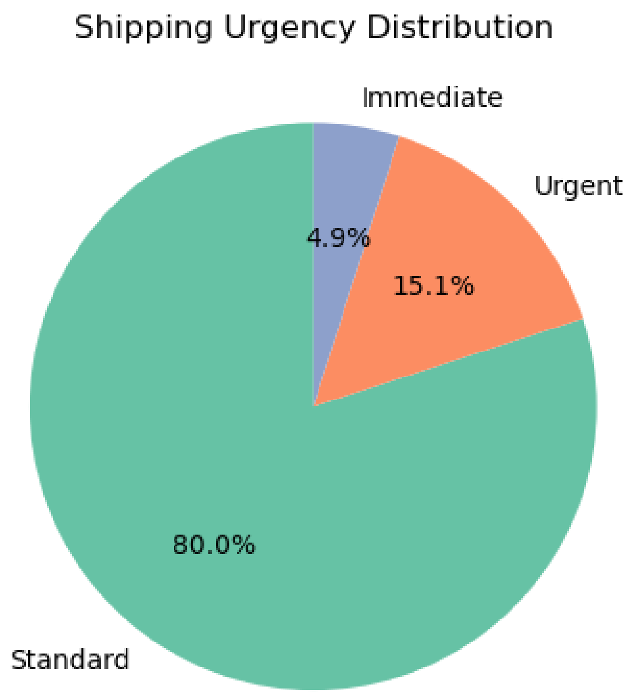
These represent loss-making transactions, likely due to: Heavy discounts High-cost shipping modes Returns or refunds

[34]:
```
# 3. Shipping and Delivery Analysis:

# 1. Distribution of Shipping Urgency: Visualize the distribution of orders by␣
↪Shipping Urgency using a pie chart or bar chart
# 2. Days to Ship vs. Profit: Use a box plot to explore the distribution of␣
↪Profit across different Days to Ship categories.
```

```
# This will help analyze whether faster shipping correlates with higher or
 ↪lower profitability.
# 3. Shipping Mode and Profitability: Create a grouped bar chart to compare the
 ↪profitability of different shipping modes
# (eg.. Standard Class,First Class).
# 4. Using pivot table, determine which shipping modes are most preferred
 ↪across different regions and analyze the impact on total sales and profit.
# Create a pivot table that shows the count of Order IDs, total Sales, and
 ↪total Profit for each Region and Ship Mode. Identify and print your insights
```

[60]:

```
df['Shipping Urgency'].value_counts().plot(kind='pie', autopct='%1.1f%%',
 ↪startangle=90, colors=['#66c2a5','#fc8d62','#8da0cb'])
plt.ylabel('')
plt.title('Shipping Urgency Distribution')
plt.show()
```

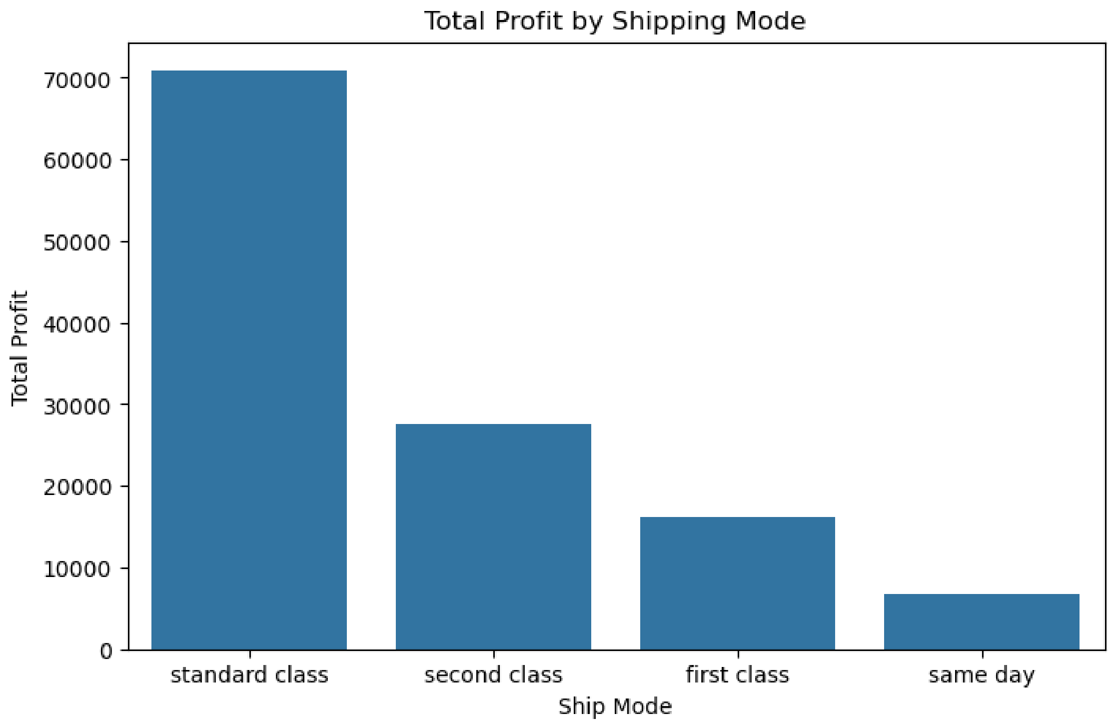


The data shows that Standard Shipping accounts for ~80% of all orders, while Urgent orders make up 15.1%, and Immediate (Same Day) orders only 4.9%. This indicates that most customers prefer cost-effective standard delivery over faster options. The low percentage of Immediate orders suggests that customers are not highly time-sensitive and are likely price-conscious. From a business perspective, this pattern highlights an opportunity to optimize logistics around standard shipping, as it handles the majority of orders. The company can also limit high-cost same-day options to select products or regions to reduce operational expenses.

[61]:
```
plt.figure(figsize=(8,5))
sns.boxplot(data=df, x='Shipping Urgency', y='Total Profit', color='skyblue')
plt.title('Distribution of Profit Across Shipping Urgency')
plt.xlabel('Shipping Urgency')
plt.ylabel('Total Profit')
plt.show()
```

The primary insight is the striking similarity across all three shipping urgency levels.:- Since neither Urgent nor Immediate shipping yields a higher profit than Standard shipping, the company should focus on minimizing the use of these faster,more expensive methods to reduce operational costs without sacrificing transactional profit.

[65]:
```
shipping_profit = df.groupby('Ship Mode')['Profit'].sum().
  ↪sort_values(ascending=False).reset_index()

plt.figure(figsize=(8,5))
sns.barplot(data=shipping_profit, x='Ship Mode', y='Profit')
plt.title('Total Profit by Shipping Mode')
plt.xlabel('Ship Mode')
plt.ylabel('Total Profit')
plt.show()
```



Key Insights Standard Class Dominance: The 'Standard Class' ship mode is the largest contributor to total profit, generating well over $70,000 in aggregate profit.Second Class Contribution: 'Second Class' is the second-highest contributor, generating around $28,000 in total profit.First Class and Same Day significantly less.

Focus on Standard: The company's overall profitability is heavily reliant on orders shipped via 'Standard Class.' This shipping mode is the backbone of the company's total earnings.the company is likely better off by limiting high-cost/fast shipping options

[66]:
```
pivot = pd.pivot_table(df,index="Region",columns="Ship Mode",values=["Order␣
↪ID", "Total Sales", "Total Profit"],
    aggfunc={"Order ID": "nunique", "Total Sales": "sum", "Total Profit":␣
↪"sum"},
    fill_value=0,
    margins=True,
    margins_name="Total"
)
pivot
```

[66]:
```
                Order ID                                              \
Ship Mode first class same day second class standard class   Total
Region
central           168       60          220            692  1140.0
east              223       69          254            808  1354.0
```

```
south              122        37          155            482    796.0
west               247        86          304            924   1561.0
Total            760.0     252.0        933.0         2906.0   4851.0

          Total Profit                                                    \
Ship Mode  first class    same day second class standard class        Total
Region
central       379.5891    3234.023    8464.0572     -2857.5172    9220.1521
east        21950.2402   7019.1758   25494.5343    104570.3065  159034.2568
south       16070.5882    7933.404   34050.1345     78352.9964  136407.1231
west        23741.0491  11673.3681   60498.3407    134145.9009  230058.6588
Total       62141.4666  29859.9709  128507.0667    314211.6866  534720.1908

           Total Sales
Ship Mode  first class    same day second class standard class         Total
Region
central     131216.004   49495.856     233994.8     658172.199   1072878.859
east        209504.811   67024.471   252083.511     758728.405   1287341.198
south       103478.698   38692.503  164570.0975     462073.992   768815.2905
west       247706.9135   79858.384  323473.3055     945701.568   1596740.171
Total      691906.4265  235071.214   974121.714    2824676.164  4725775.5185
```

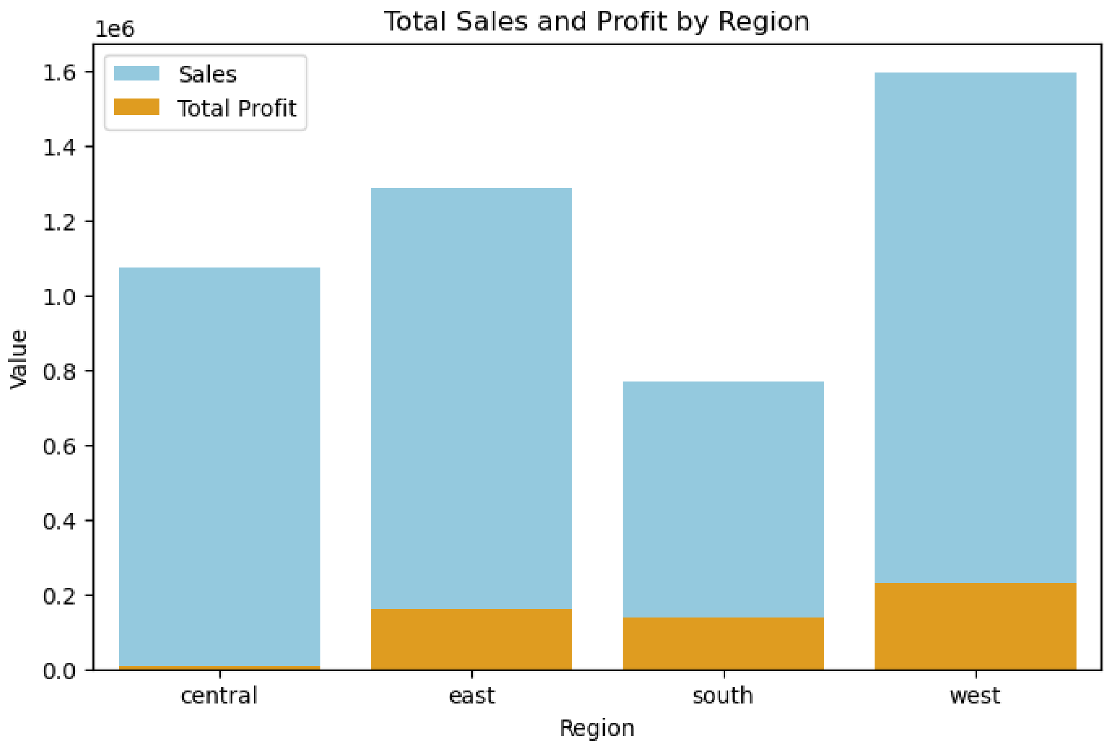
The West region generates the highest total profit ($230,058.66), followed by the East region ($159,034.26). Central and South Profit:Central ($92,200.15) and South ($78,352.99) lag significantly behind the West and East regions in terms of aggregate profit.

Reinforce the West and East: These two regions are your profit powerhouses (especially the West).Ensure operational efficiency is optimized here to maintain high sales and profit. Focus Improvement on Central and South:

[42]:
```
# 4. Regional Sales and Profitability:

# 1. Sales and Profit by Region: Use a map or bar chart to visualize total␣
↪sales and profit by region or state.
# This will highlight which regions are the most profitable
# 2. State-wise Profitability: Create a pivot table to summarize the␣
↪profitability of each state. Highlight the top and
# bottom states based on profitability.
```

[43]:
```
region_summary = df.groupby('Region')[['Total Sales', 'Total Profit']].sum().
↪reset_index()

# Plot
plt.figure(figsize=(8,5))
sns.barplot(data=region_summary, x='Region', y='Total Sales', color='skyblue',␣
↪label='Sales')
sns.barplot(data=region_summary, x='Region', y='Total Profit', color='orange',␣
↪label='Total Profit')
```

```
plt.title('Total Sales and Profit by Region')
plt.ylabel('Value')
plt.legend()
plt.show()
```



Sales:- The West and East regions are the clear leaders in terms of Total Sales,with both regions generating sales exceeding $1.2 millionCentral is the next largest contributor, followed by South, which has the lowest sales volume (below $0.8 million) Profit:- The West region generates the highest Total Profit The East region is the second highest profit contributor. Central and South regions contribute significantly less to the total profit.

Sales/profit:- The West region shows a healthy profit margin. Although its sales bar is the highest, the orange profit slice is proportionally very large relative to the blue sales slice. The East region, while having high sales, appears to have a slightly lower profit margin compared to the West The South and east region's profit appears to be the thinnest relative to its sales bar, suggesting it has the lowest profit margin.

[44]:
```
# Pivot table for State-wise total profit
state_profit = df.groupby('State')['Profit'].sum().reset_index().
↪sort_values(by='Profit', ascending=False)

print("Top 10 Most Profitable States:")
```

```
print(state_profit.head(10))

print("\nBottom 10 Least Profitable States:")
print(state_profit.tail(10))
```

```
Top 10 Most Profitable States:
            State      Profit
3      California  44672.2302
30       New York  28135.2368
45     Washington  10376.6167
20       Michigan   9824.2301
44       Virginia   7279.8420
9         Georgia   6946.9138
15       Kentucky   5335.4919
12        Indiana   5002.5213
28     New Jersey   4495.7085
19  Massachusetts   4470.3288

Bottom 10 Least Profitable States:
                   State      Profit
7   District Of Columbia     94.1444
40             Tennessee   -101.0933
8                Florida   -696.0981
35                Oregon  -1414.3545
1                Arizona  -2135.8807
4               Colorado  -3438.5615
33                  Ohio  -5339.8993
36          Pennsylvania  -9324.1787
11              Illinois -11818.3018
41                 Texas -13031.3552
```

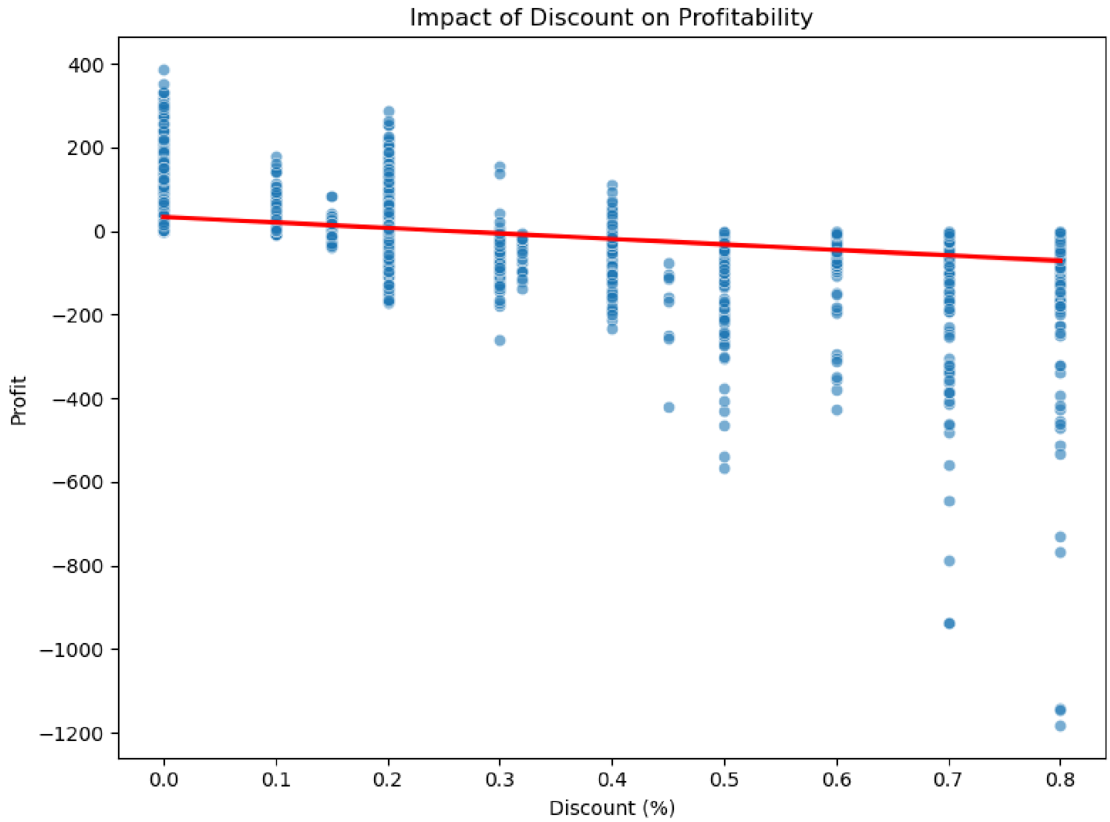
[45]:

```
# 5. Discount and Pricing Analysis:

# 1. Impact of Discounts on Profitability: Use a scatter plot with a trend line
 ↪to analyze how different levels of discount affect profitability.
# 2. Original Price vs. Discounted Price: Create chart to compare the original
 ↪price and the discounted price across various product
# categories or sub-categories
```

[46]:

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(8,6))
sns.scatterplot(data=df, x="Discount", y="Profit", alpha=0.6)
sns.regplot(data=df, x="Discount", y="Profit", scatter=False, color="red")
plt.title("Impact of Discount on Profitability")
plt.xlabel("Discount (%)")
```

```
plt.ylabel("Profit")
plt.tight_layout()
plt.show()
```

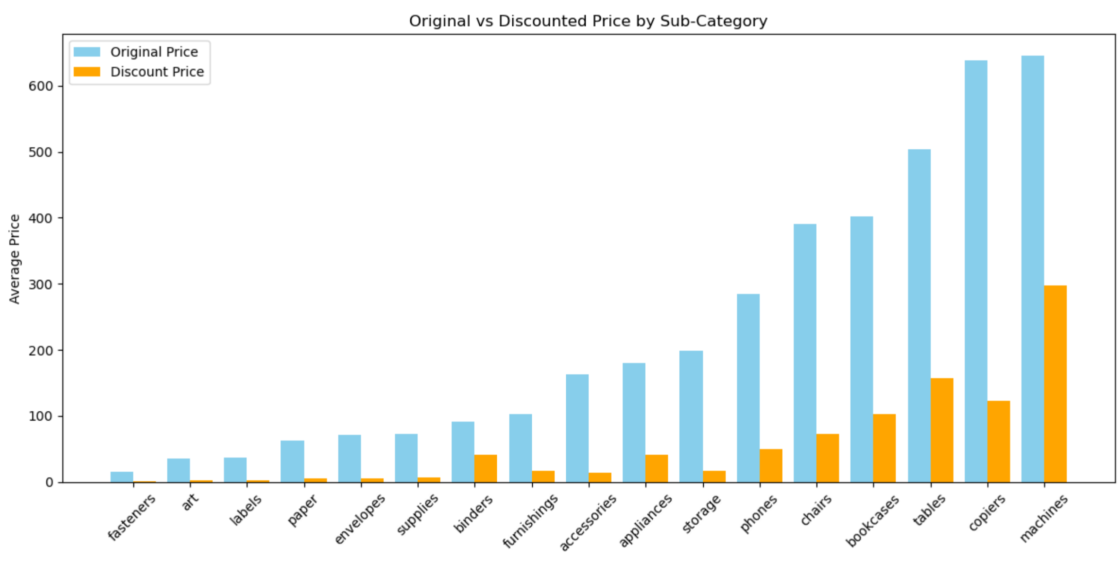


There is a strong negative correlation between the discount percentage and Profit. As the discount increases, the profit tends to decrease At 0% discount, the profit is clustered tightly around zero, with some instances of high profit 400 and some moderate losses. As the discount approaches 30% to 40% the bulk of transactions start to fall below the zero profit line, meaning the orders are generally resulting in a loss High discounts, particularly those around 60% to 80%, are where the most severe losses occur.The largest losses, dipping to -1,000 to -1,200, are concentrated at the highest discount levels 70% to 80% This directly addresses the "negative outliers" problem seen in previous analyses—high discounts are a major driver of large losses.

Discount Strategy Review: The company must urgently review its discounting policy. Discounts, particularly those exceeding 40%, are highly likely to result in losses and are the main cause of the extreme financial outliers. Set a Discount Cap: A policy should be implemented to strictly control or eliminate discounts above a certain threshold unless the underlying product has an exceptionally high margin to absorb the markdown

[35]:
```
import numpy as np
import matplotlib.pyplot as plt
```

```
# Prepare data
price_comparison = df.groupby('Sub-Category')[['Original Price','Discount␣
↪Price']].mean().reset_index().sort_values(by='Original Price')

# Set positions and width
x = np.arange(len(price_comparison['Sub-Category']))  # the label locations
width = 0.4  # the width of the bars

# Plot
plt.figure(figsize=(12,6))
plt.bar(x - width/2, price_comparison['Original Price'], width, label='Original␣
↪Price', color='skyblue')
plt.bar(x + width/2, price_comparison['Discount Price'], width, label='Discount␣
↪Price', color='orange')

# Labels and formatting
plt.xticks(x, price_comparison['Sub-Category'], rotation=45)
plt.ylabel('Average Price')
plt.title('Original vs Discounted Price by Sub-Category')
plt.legend()
plt.tight_layout()
plt.show()
```

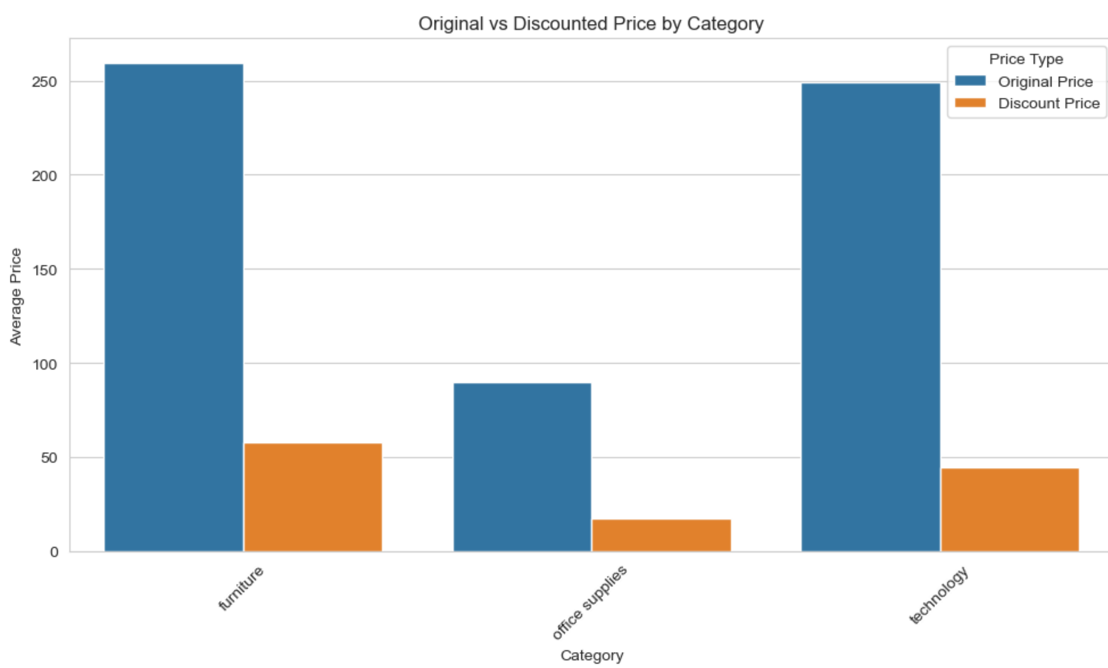


Technology and Furniture Drive Dollar Losses: The two sub-categories with the largest average dollar discounts are Machines and Copiers, both from the Technology category.Furniture's Triple Threat (Tables, Bookcases, Chairs): The next three largest dollar discounts come from the Furniture category: Furniture has the lowest profit margin (2.49%)

Profit Killer (Furniture): The high dollar discounts on Tables, Bookcases, and Chairs part of the Furniture category, which has the lowest overall margin must be strictly controlled to prevent large individual transaction losses.

Revenue Killer (Technology): While Machines and Copiers can absorb large dollar discounts due to their high price, their profitability should be closely monitored to ensure the huge revenue they generate is not being eroded by discounts.

[47]:
```python
# Group and prepare the data
pf = df.groupby('Category')[['Original Price','Discount Price']].mean().
 ↪reset_index()

# Melt the DataFrame to long format for grouped bar chart
pf_melted = pf.melt(id_vars='Category', value_vars=['Original Price', 'Discount␣
 ↪Price'],
                    var_name='Price Type', value_name='Average Price')

# Plotting
plt.figure(figsize=(10,6))
sns.barplot(data=pf_melted, x='Category', y='Average Price', hue='Price Type')
plt.title('Original vs Discounted Price by Category')
plt.ylabel('Average Price')
plt.xticks(rotation=45)
plt.legend(title='Price Type')
plt.tight_layout()
plt.show()
```



1. Technology and Furniture categories have the highest original prices (200– 250 range) and also show noticeable discounts, suggesting retailers offer more discounts on expensive items to boost purchase intent.
2. Office Supplies have the lowest prices (< 100) and minimal discounts, meaning these are low-cost essentials where discounts aren't needed to drive sales.
3. The gap between original and discounted prices is largest for Technology, indicating a high-discount strategy — likely to stay competitive or clear inventory of fast-evolving tech products.
4. Furniture discounts are moderate — possibly due to higher shipping or storage costs that limit heavy markdowns.

[45]:
```
df.groupby(df["Category"])["Total Profit"].sum().reset_index()
```

[45]:
```
          Category  Total Profit
0        furniture    -2408.0523
1  office supplies   346667.4241
2       technology    190460.819
```

[49]:
```
# 6. Temporal Analysis:

# 1. Sales and Profit Trends Over Time: Use a time series plot to analyze how␣
 ↪sales and profit have trended over the years ar months.
# This will help in identifying any seasonal patterns.
# 2. Order Frequency by Month: Use a bar chart or line plot to show the number␣
 ↪of orders placed each month. Highlight any months with
# unusually high or low order frequencies.
# 3. Yearly Growth in Sales and Profit: Use a year-over-year growth chart to␣
 ↪compare the sales and profit growth over different years.
```

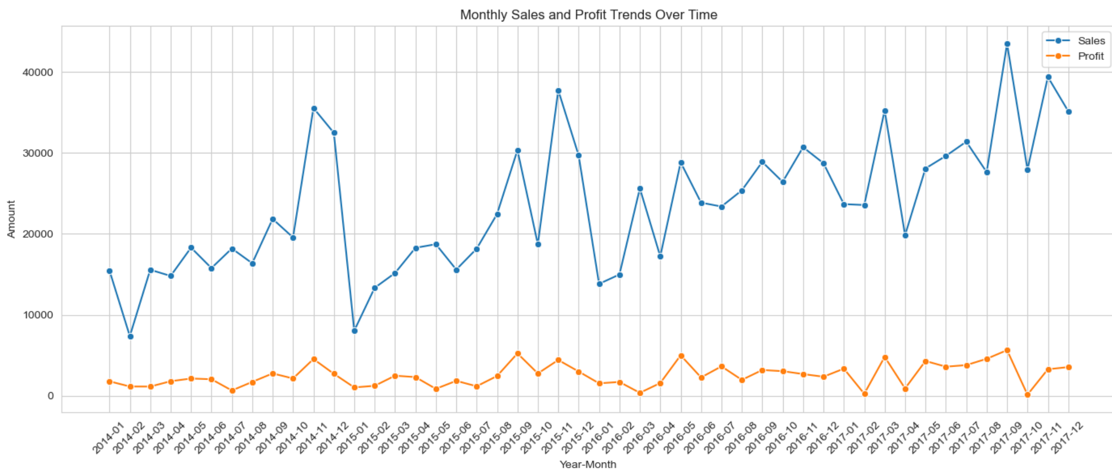
[52]:
```
monthly_trend
```

[52]:
```
   YearMonth  Sales Price     Profit
0    2014-01   15492.5080  1800.6074
1    2014-02    7376.9780  1161.5573
2    2014-03   15570.3810  1152.7621
3    2014-04   14806.4460  1810.4904
4    2014-05   18301.6400  2146.7556
5    2014-06   15775.6216  2058.4346
6    2014-07   18195.1120   669.9275
7    2014-08   16371.6835  1728.5532
8    2014-09   21843.8998  2774.2497
9    2014-10   19558.1110  2152.2706
10   2014-11   35540.6235  4551.1441
11   2014-12   32474.5407  2730.3293
12   2015-01    8050.2060  1046.9356
13   2015-02   13338.2100  1245.4814
14   2015-03   15166.7616  2493.5004
15   2015-04   18275.9125  2321.2967
16   2015-05   18725.1845   874.1701
```

```
17   2015-06   15591.4352  1858.9962
18   2015-07   18168.1210  1188.8596
19   2015-08   22443.2760  2454.1427
20   2015-09   30337.6750  5247.9759
21   2015-10   18714.7930  2770.0687
22   2015-11   37733.8835  4434.2015
23   2015-12   29678.7872  3010.8394
24   2016-01   13827.2200  1552.6127
25   2016-02   14987.4820  1720.7262
26   2016-03   25617.5730   381.1710
27   2016-04   17218.7650  1584.8303
28   2016-05   28830.5100  5012.9304
29   2016-06   23854.6890  2277.9347
30   2016-07   23372.9990  3646.4781
31   2016-08   25375.6508  1977.3070
32   2016-09   28882.0535  3195.7547
33   2016-10   26419.5520  3055.1815
34   2016-11   30697.7280  2697.2984
35   2016-12   28722.0905  2365.4004
36   2017-01   23669.9140  3362.4312
37   2017-02   23562.3874   260.5841
38   2017-03   35249.1218  4818.3228
39   2017-04   19834.6446   917.1285
40   2017-05   28075.5084  4310.3243
41   2017-06   29623.0120  3605.9494
42   2017-07   31376.2125  3799.5514
43   2017-08   27623.9290  4578.4098
44   2017-09   43481.8730  5646.2500
45   2017-10   27925.4382   146.0604
46   2017-11   39370.8280  3289.8976
47   2017-12   35134.0738  3553.5435
```

[51]:
```
df["YearMonth"] = df["Order Date"].dt.to_period("M").astype(str)
monthly_trend = df.groupby("YearMonth")[["Sales Price", "Profit"]].sum().
  ↪reset_index()

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(14,6))
sns.lineplot(data=monthly_trend, x="YearMonth", y="Sales Price", label="Sales",␣
  ↪marker="o")
sns.lineplot(data=monthly_trend, x="YearMonth", y="Profit", label="Profit",␣
  ↪marker="o")

plt.title("Monthly Sales and Profit Trends Over Time")
plt.xticks(rotation=45)
```

```
plt.ylabel("Amount")
plt.xlabel("Year-Month")
plt.legend()
plt.tight_layout()
plt.show()
```



Sales show an upward trend over time, though with noticeable seasonal fluctuations. Profit follows a similar pattern but stays much lower than sales, indicating thin profit margins or high costs/discounts in some months.

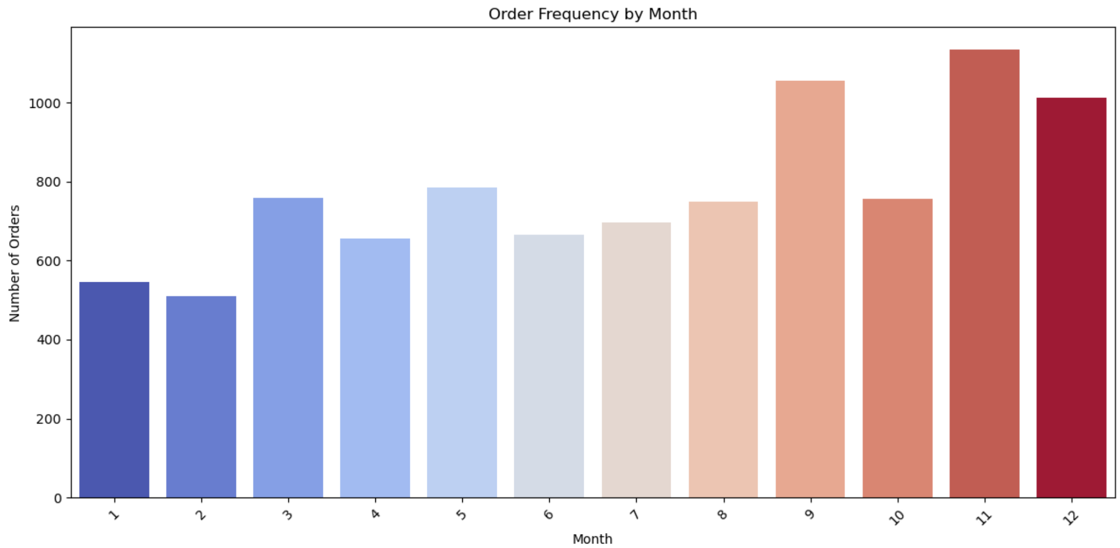
There are repeated sales spikes toward year-end (Nov–Dec) — possibly due to holiday or festival seasons, when promotions and discounts increase demand. Profit does not always peak with sales, suggesting discount-heavy sales periods reduce profitability November and September consistently show high sales and profit, suggesting strong seasonal demand. February and July often have lower performance, possibly due to off-season effects or market slowdowns.

Some months show high sales but low profits or even dips, meaning operational inefficiencies or over-discounting could be affecting profitability March 2016: High sales (25,617.57) but very low profit (381.17), indicating potential discounting, high costs, or operational inefficiencies. October 2017: Decent sales (27,925.44) but lowest profit (146.06), possibly due to unexpected expenses or pricing issues.

The business is growing in sales, but profit optimization is needed. Focus should be on reducing costs or optimizing discount strategies during high-sales months to improve profit margins.

[59]:
```
plt.figure(figsize=(12,6))
sns.barplot(data=monthly_orders,x="Month",y="Order␣
↪Count",hue="Month",palette="coolwarm",legend=False)

plt.title("Order Frequency by Month")
plt.ylabel("Number of Orders")
plt.xticks(rotation=45)
```

```
plt.tight_layout()
plt.show()
```



Peak Ordering Months: The highest number of orders occur in the last quarter (October, November, December). This confirms a strong year-end sales surge, likely due to festive seasons, holiday shopping, and promotional offers.

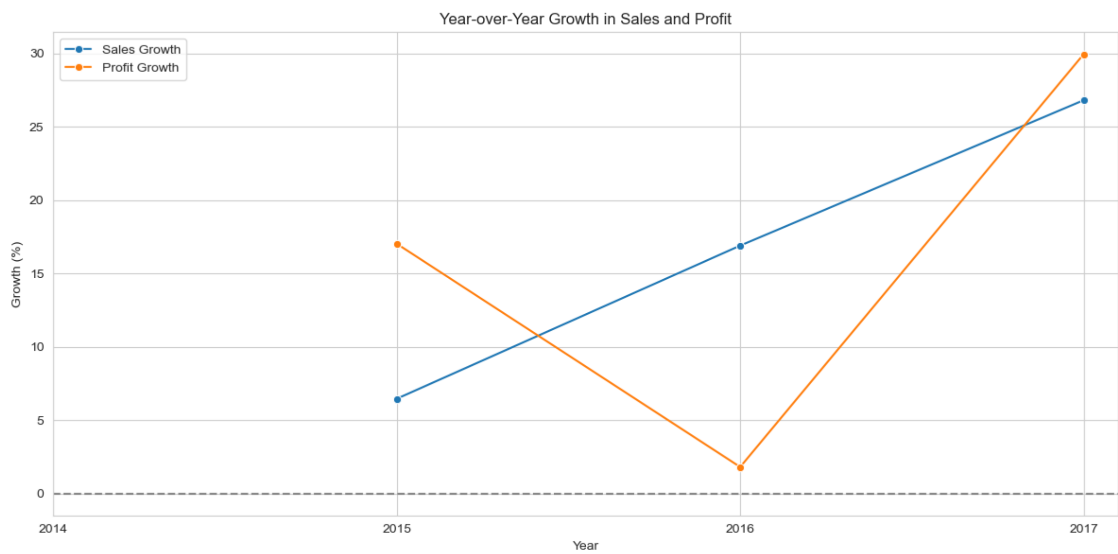
Moderate Activity: Mid-year months (April–August) show average order volume — business remains steady but not at its highest. This may reflect regular business demand without seasonal effects.

Low Order Period January–February record the lowest order counts, suggesting a post-holiday slowdown where customers and businesses reduce spending after major festivals or holidays

The data shows a clear upward trend from mid-year to year-end, showing seasonality in customer purchase behavior. The company experiences strong Q4 performance, so marketing and inventory should focus on this period. Plan stock replenishment, staffing, and marketing campaigns ahead of Q4 to meet high demand. Consider boosting sales in slow months (like Jan–Feb) with discounts, loyalty programs, or new product launches

[54]:
```
df["Year"] = df["Order Date"].dt.year
yearly_trend = df.groupby("Year")[["Sales Price", "Profit"]].sum().reset_index()
yearly_trend["Sales Growth (%)"] = yearly_trend["Sales Price"].pct_change() *
 ↪100
yearly_trend["Profit Growth (%)"] = yearly_trend["Profit"].pct_change() * 100

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12,6))
```

```
sns.lineplot(data=yearly_trend, x="Year", y="Sales Growth (%)", marker="o",␣
 ↪label="Sales Growth")
sns.lineplot(data=yearly_trend, x="Year", y="Profit Growth (%)", marker="o",␣
 ↪label="Profit Growth")

plt.title("Year-over-Year Growth in Sales and Profit")
plt.ylabel("Growth (%)")
plt.xticks(yearly_trend["Year"])
plt.axhline(0, color='gray', linestyle='--')
plt.legend()
plt.tight_layout()
plt.show()
```



2014–2015: Stable & Profitable Growth Both sales and profit increased, but profit grew faster (17%) than sales (6%). This means better efficiency or higher margins — the business was scaling profitably.

2015–2016: Warning Signal Sales rose by 16.9%, but profit almost stagnated (only 1.8%). This shows that despite higher revenue, costs increased sharply or low-margin products dominated sales.

2016–2017: Recovery & Strong Growth Both sales (+26.8%) and profit (+29.9%) grew strongly. This indicates strategic improvements likely better cost control, reduced discounts, or focus on high-profit items. Business became more efficient and sustainable again.

[ ]:

```
```