
SUPER
STORE
SUPER STORE

Super Store Final Project

Project Goal:

This project aims to analyze sales data from a Superstore to gain insights into sales trends, product performance, customer behavior, and profitability. These insights can then be used to inform data-driven decision-making and business growth strategies.

Data Sources:

The project uses an Excel file as data sources contains three sheets 'Orders', 'People', and 'Returns '. These sheets contain information about orders, customers, and product returns.

Disclaimer:

As a Data Analyst, my goal is to extract meaningful insights from sales data to support business decision-making. Initially, I received a dataset from the Administration that lacks key financial and transactional metrics required for a meaningful analysis.The administration dataset only contains **sales** as a measure, while crucial financial metrics such as **profit, discount, and quantity** are absent.
The alternative dataset includes **Profit**, allowing for **profitability analysis**, which is essential for understanding business success beyond revenue. In addition to a **returns** column, which is essential for understanding product return rates ,customer satisfaction and to analyze **return reasons, refund impact on revenue, and areas for improvement in product quality or shipping methods**.

Project Workflow:

1. Data Cleaning and Preparation:

Data Import and Inspection: The project begins by importing data from the Excel files using Pandas. It then inspects the data for missing values, inconsistencies, and outliers.

```
!pip install openpyxl
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from google.colab import drive
drive.mount('/content/drive')
path1='/content/drive/MyDrive/DEPI Final Super store /Orders.xlsx'
orders_df=pd.read_excel(path1, engine='openpyxl') # Changed to use read_excel and specify engine
path2='/content/drive/MyDrive/DEPI Final Super store /People.xlsx'
people_df=pd.read_excel(path2, engine='openpyxl') # Changed to use read_excel and specify engine
path3='/content/drive/MyDrive/DEPI Final Super store /Returns.xlsx'
returns_df=pd.read_excel(path3, engine='openpyxl') # Changed to use read_excel and specify engine
```

```
orders_df
```

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	...	Postal Code
0	1	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	42420
1	2	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	42420
2	3	CA-2016-138688	2016-06-12	2016-06-16	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	...	90036
3	4	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	33311

```
people_df
```

	Person	Region
0	Anna Andreadi	West
1	Chuck Magee	East
2	Kelly Williams	Central
3	Cassandra Brandow	South

```
returns_df
```

	Returned	Order ID
0	Yes	CA-2017-153822
1	Yes	CA-2017-129707
2	Yes	CA-2014-152345
3	Yes	CA-2015-156440
4	Yes	US-2017-155999
...	...	...

- Inspect the Data:

```
orders_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Row ID         9994 non-null   int64
 1   Order ID       9994 non-null   object
 2   Order Date     9994 non-null   datetime64[ns]
 3   Ship Date      9994 non-null   datetime64[ns]
 4   Ship Mode      9994 non-null   object
 5   Customer ID    9994 non-null   object
 6   Customer Name  9994 non-null   object
 7   Segment        9994 non-null   object
 8   Country        9994 non-null   object
 9   City           9994 non-null   object
 10  State          9994 non-null   object
 11  Postal Code    9994 non-null   int64
 12  Region         9994 non-null   object
 13  Product ID     9994 non-null   object
 14  Category       9994 non-null   object
 15  Sub-Category   9994 non-null   object
 16  Product Name   9994 non-null   object
 17  Sales          9994 non-null   float64
 18  Quantity       9994 non-null   int64
 19  Discount       9994 non-null   float64
 20  Profit         9994 non-null   float64
```

```
people_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Person  4 non-null      object
 1   Region  4 non-null      object
dtypes: object(2)
memory usage: 196.0+ bytes
```

```
returns_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 296 entries, 0 to 295
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Returned  296 non-null    object
 1   Order ID  296 non-null    object
dtypes: object(2)
memory usage: 4.8+ KB
```

Data Cleaning

Handle Missing Values

```
# Inspect missing values
print(orders_df.isnull().sum())
# Clean 'Returns'
duplicate_order_ids = returns_df[returns_df['Order ID'].duplicated(keep=False)]
if not duplicate_order_ids.empty:
    print("Duplicate Order IDs found:")
    display(duplicate_order_ids)
# Handle missing values (People)
if people_df.isnull().values.any():
    print("There are missing values in the DataFrame.")
else:
    print("There are no missing values in the DataFrame.")
# Handle missing values
for col in ['Sales', 'Quantity', 'Discount', 'Profit']:
    if orders_df[col].isnull().any():
```

```
        orders_df[col] = orders_df[col].fillna(orders_df[col].mean())
# Inspect missing values
print(orders_df.isnull().sum())
# Clean 'Returns'
duplicate_order_ids = returns_df[returns_df['Order ID'].duplicated(keep=False)]
if not duplicate_order_ids.empty:
    print("Duplicate Order IDs found:")
    display(duplicate_order_ids)
# Handle missing values (People)
if people_df.isnull().values.any():
    print("There are missing values in the DataFrame.")
else:
    print("There are no missing values in the DataFrame.")
# Handle missing values
for col in ['Sales', 'Quantity', 'Discount', 'Profit']:
    if orders_df[col].isnull().any():
        orders_df[col] = orders_df[col].fillna(orders_df[col].mean())
```

Handle inconsistencies in categorical variables and Standardize Columns

```
# Handle inconsistencies in categorical variables (example: Standardize
capitalization)
for col in ['Ship Mode', 'Segment', 'Country', 'City', 'State', 'Region',
'Category', 'Sub-Category']:
    if orders_df[col].dtype == 'object':
        orders_df[col] = orders_df[col].str.title()
# Standardize Columns
people_df.columns = [col.lower().replace(' ', '_') for col in
people_df.columns]
returns_df.columns = [col.lower().replace(' ', '_') for col in
returns_df.columns]
```

Identify and handle outliers and Convert Data type

```
# Identify and handle outliers (example: Winsorization for numerical columns)
for col in ['Sales', 'Quantity', 'Discount', 'Profit']:
    # Convert 'Discount' to numeric if it's not already
   if orders_df[col].dtype == 'object' and col == 'Discount':
```

```
  orders_df[col] = pd.to_numeric(orders_df[col].str.rstrip('%'),
errors='coerce') / 100

    q1 = orders_df[col].quantile(0.25)
    q3 = orders_df[col].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    orders_df[col] = orders_df[col].clip(lower=lower_bound, upper=upper_bound)
# Check and convert data types if necessary (example: converting 'order_id' to
string)
if not pd.api.types.is_string_dtype(returns_df['order_id']):
    returns_df['order_id'] = returns_df['order_id'].astype(str)
    print("'order_id' column in df_returns converted to string.")
display(people_df.head())
display(returns_df.head())
```

Remove duplicate rows

```
# Remove duplicate rows
num_duplicates = orders_df.duplicated().sum()
orders_df_cleaned = orders_df.drop_duplicates()
print(f"Number of duplicate rows removed: {num_duplicates}")
# Iterate through rows instead of using apply
for index, row in orders_df.iterrows():
    discount = row['Discount']
    sales = row['Sales']
    if not (0 <= discount <= 1):
        orders_df.loc[index, 'Discount'] = discount / sales if sales != 0 else
0

orders_df['Discount'] = (orders_df['Discount'] * 100).astype(int)  # Convert to
percentage numeric
# Add 'Cost' column (using estimation if cost data is unavailable)
orders_df['Cost'] = orders_df['Sales'] - orders_df['Profit']
# Add 'Original Price' column
# Instead of using .str.rstrip, directly use the numeric 'Discount' column
orders_df['Original Price'] = orders_df['Sales'] / (1 - orders_df['Discount'])

display(orders_df_cleaned.head())
```

Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	City	...	Product ID	Category	Sub-Category	Product Name	Sales	Quantity	Discount	Profit	Cost	Original Price
2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	FUR-BO-10001798	Furniture	Bookcases	Bush Somerset Collection Bookcase	261.960	2.0	0	41.913600	220.046400	261.960000
2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer	United States	Henderson	...	FUR-CH-10000454	Furniture	Chairs	Hon Deluxe Fabric Upholstered Stacking Chairs,...	498.930	3.0	0	70.816875	428.113125	498.930000
2016-06-12	2016-06-16	Second Class	DV-13045	Darrin Van Huff	Corporate	United States	Los Angeles	...	OFF-LA-10000240	Office Supplies	Labels	Self-Adhesive Address Labels for Typewriters b...	14.620	2.0	0	6.871400	7.748600	14.620000
2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	FUR-TA-10000577	Furniture	Tables	Bretford CR4500 Series Slim Rectangular Table	498.930	5.0	45	-39.724125	538.654125	-11.339318
2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States	Fort Lauderdale	...	OFF-ST-10000760	Office Supplies	Storage	Eldon Fold 'N Roll Cart System	22.368	2.0	20	2.516400	19.851600	-1.177263

```
# Identify and remove duplicate rows.
df_people = people_df.drop_duplicates()
# Examine unique values in each column.
print(people_df['person'].unique())
print(people_df['region'].unique())
display(people_df)
```

```
['Anna Andreadi' 'Chuck Magee' 'Kelly Williams' 'Cassandra Brandow']
['West' 'East' 'Central' 'South']
```

	person	region
0	Anna Andreadi	West
1	Chuck Magee	East
2	Kelly Williams	Central
3	Cassandra Brandow	South

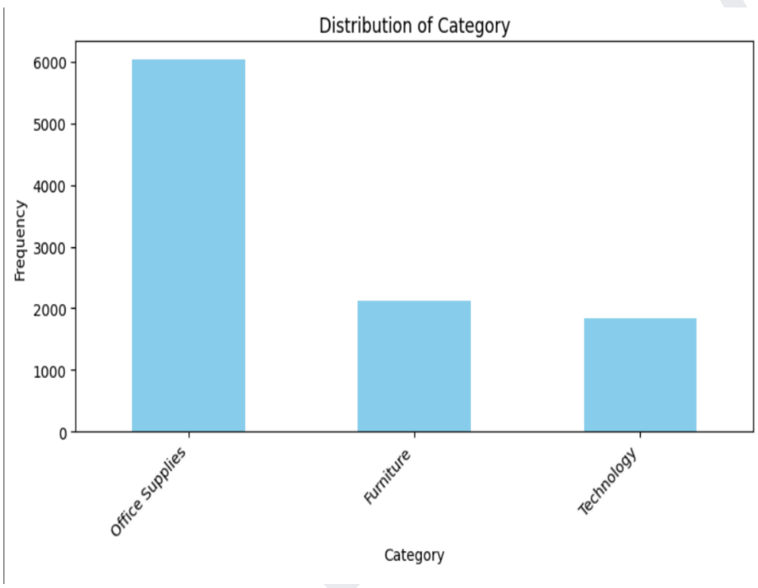
Data exploration

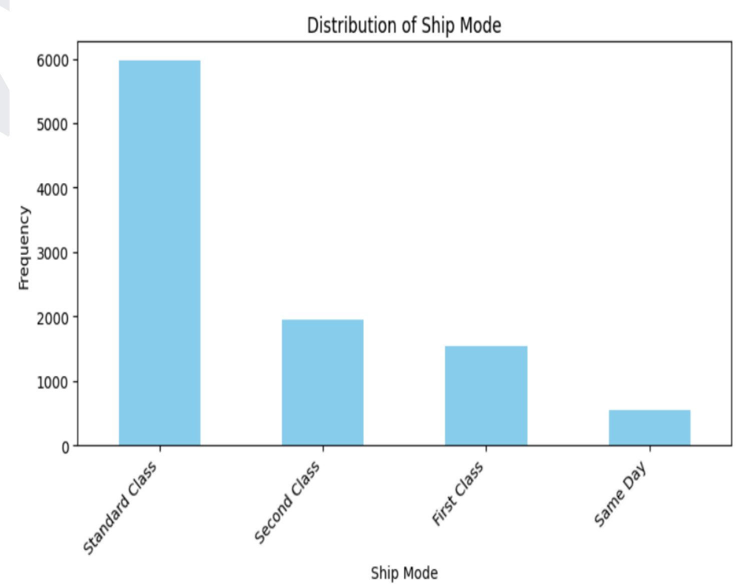
To understand its characteristics by examining its shape, summary statistics, categorical features, and relationships between variables.

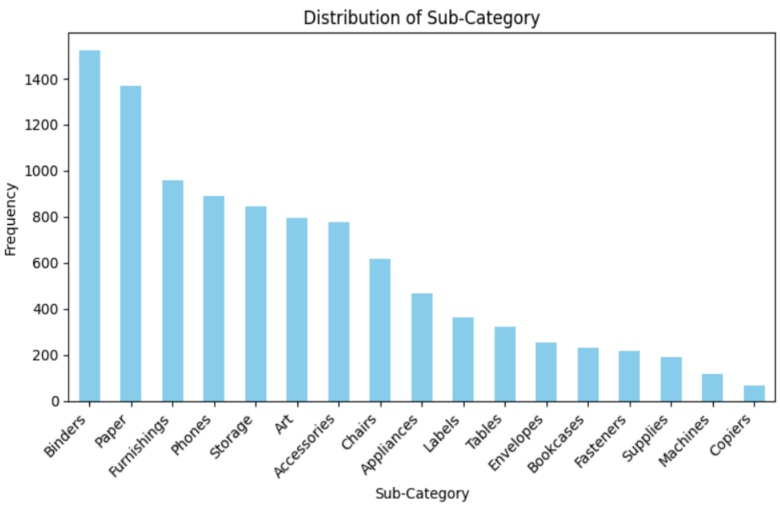
```
# 1. Data Shape
print(f"Number of rows: {orders_df_cleaned.shape[0]}")
print(f"Number of columns: {orders_df_cleaned.shape[1]}")
# 2. Summary Statistics
numerical_features = ['Sales', 'Quantity', 'Discount', 'Profit']
display(orders_df_cleaned[numerical_features].describe())
# 3. Categorical Feature Analysis
categorical_features = ['Category', 'Sub-Category', 'Ship Mode', 'Segment',
'Region', 'State']
import matplotlib.pyplot as plt
for col in categorical_features:
    plt.figure(figsize=(10, 6))  # Adjust figure size
    orders_df_cleaned[col].value_counts().plot(kind='bar', color='skyblue')
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.xticks(rotation=45, ha='right') # Rotate x-axis labels for better
readability
    plt.tight_layout() # Adjust layout to prevent labels from overlapping
    plt.show()
# 4. Relationships Between Variables
display(orders_df_cleaned[numerical_features].corr())
plt.figure(figsize=(8, 6))
import seaborn as sns
sns.heatmap(orders_df_cleaned[numerical_features].corr(), annot=True,
cmap='coolwarm')
plt.title('Correlation Matrix of Numerical Features')
plt.show()
# Additional relationship exploration (example: Average sales per category)
display(orders_df_cleaned.groupby('Category')['Sales'].mean())
```

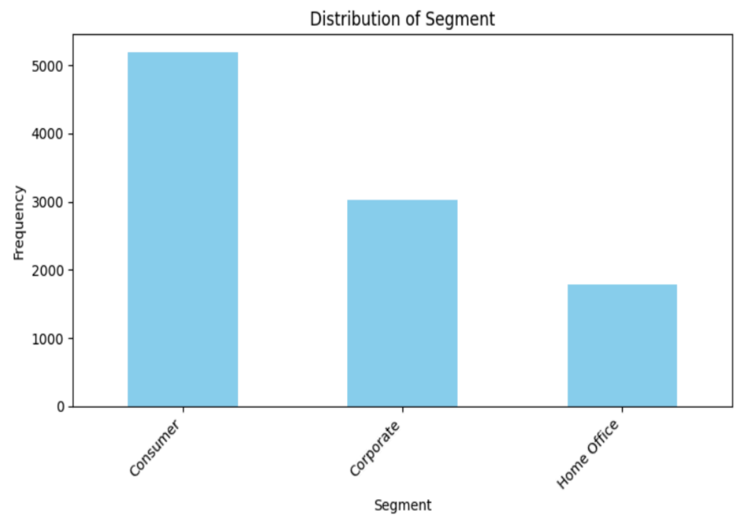
```
Number of rows: 9994
Number of columns: 23
```

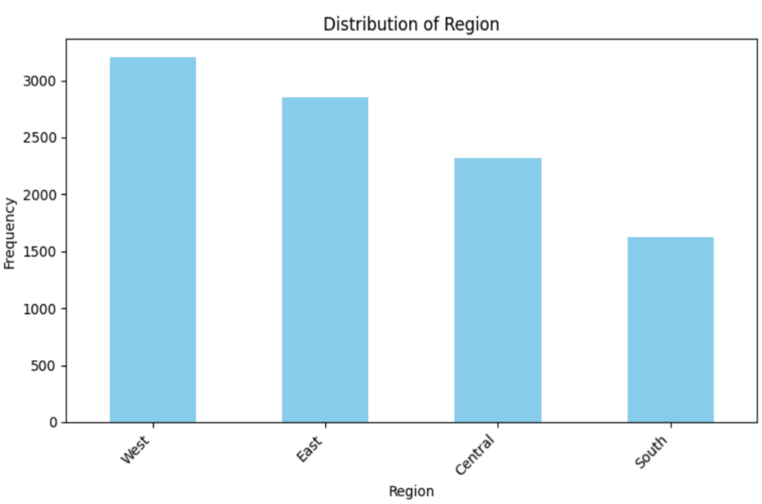
	Sales	Quantity	Discount	Profit
count	9994.000000	9994.000000	9994.000000	9994.000000
mean	140.281105	3.753352	13.745147	16.068014
std	168.804241	2.102557	15.768026	29.486488
min	0.444000	1.000000	0.000000	-39.724125
25%	17.280000	2.000000	0.000000	1.728750
50%	54.490000	3.000000	20.000000	8.666500
75%	209.940000	5.000000	20.000000	29.364000
max	498.930000	9.500000	50.000000	70.816875

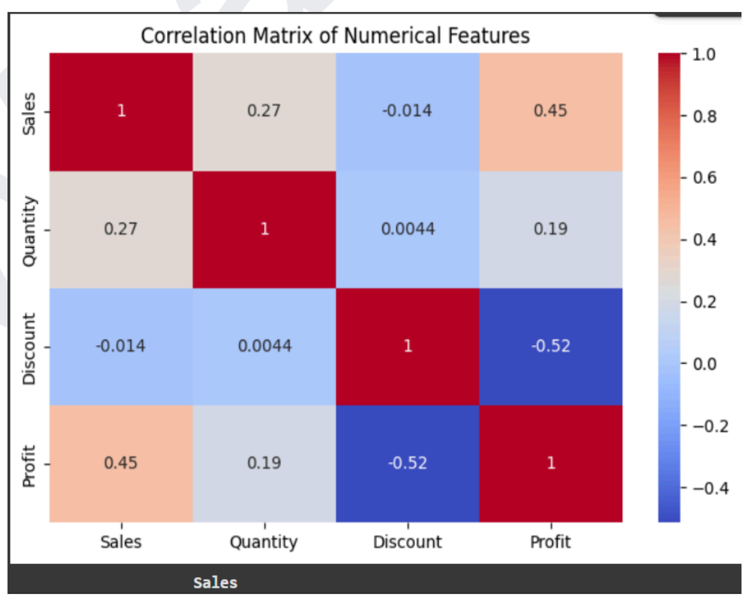












Data Wrangling

To support further analysis and forecasting.

```
# Calculate total revenue
orders_df_wrangled = orders_df_cleaned.copy()
orders_df_wrangled['TotalRevenue'] = orders_df_wrangled['Sales'] *
orders_df_wrangled['Quantity']

# Calculate profit margin, handling potential division by zero
orders_df_wrangled['ProfitMargin'] = orders_df_wrangled['Profit'] /
orders_df_wrangled['Sales']
```

```python
orders_df_wrangled['ProfitMargin'] =
orders_df_wrangled['ProfitMargin'].fillna(0)

# Convert 'Order Date' to datetime and extract components
orders_df_wrangled['Order Date'] = pd.to_datetime(orders_df_wrangled['Order
Date'])
orders_df_wrangled['OrderMonth'] = orders_df_wrangled['Order Date'].dt.month
orders_df_wrangled['OrderYear'] = orders_df_wrangled['Order Date'].dt.year
orders_df_wrangled['OrderQuarter'] = orders_df_wrangled['Order
Date'].dt.quarter
# Create combined location feature
orders_df_wrangled['CombinedLocation'] = orders_df_wrangled['City'] + ', ' +
orders_df_wrangled['State'] + ', ' + orders_df_wrangled['Region']
display(orders_df_wrangled.head())
```

Quantity	Discount	Profit	TotalRevenue	ProfitMargin	OrderMonth	OrderYear	OrderQuarter	CombinedLocation
2.0	0.00	41.913600	523.920	0.160000	11	2016	4	Henderson, Kentucky, South
3.0	0.00	70.816875	1496.790	0.141937	11	2016	4	Henderson, Kentucky, South
2.0	0.00	6.871400	29.240	0.470000	6	2016	2	Los Angeles, California, West
5.0	0.45	-39.724125	2494.650	-0.079619	10	2015	4	Fort Lauderdale, Florida, South
2.0	0.20	2.516400	44.736	0.112500	10	2015	4	Fort Lauderdale, Florida, South

Data analysis

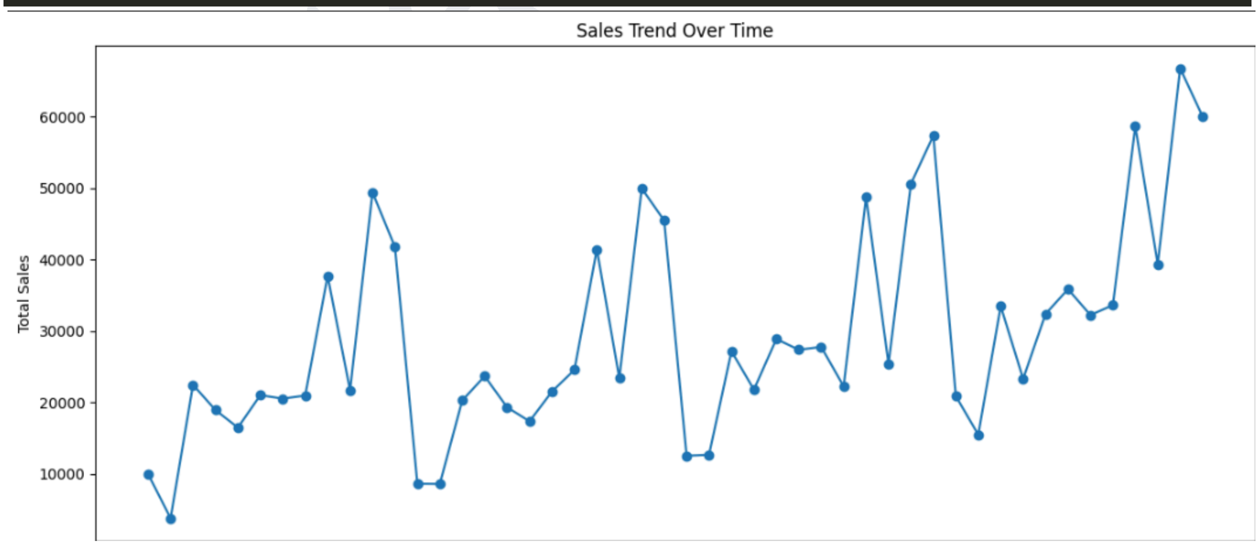
Analyze the data to answer the initial set of analysis questions.

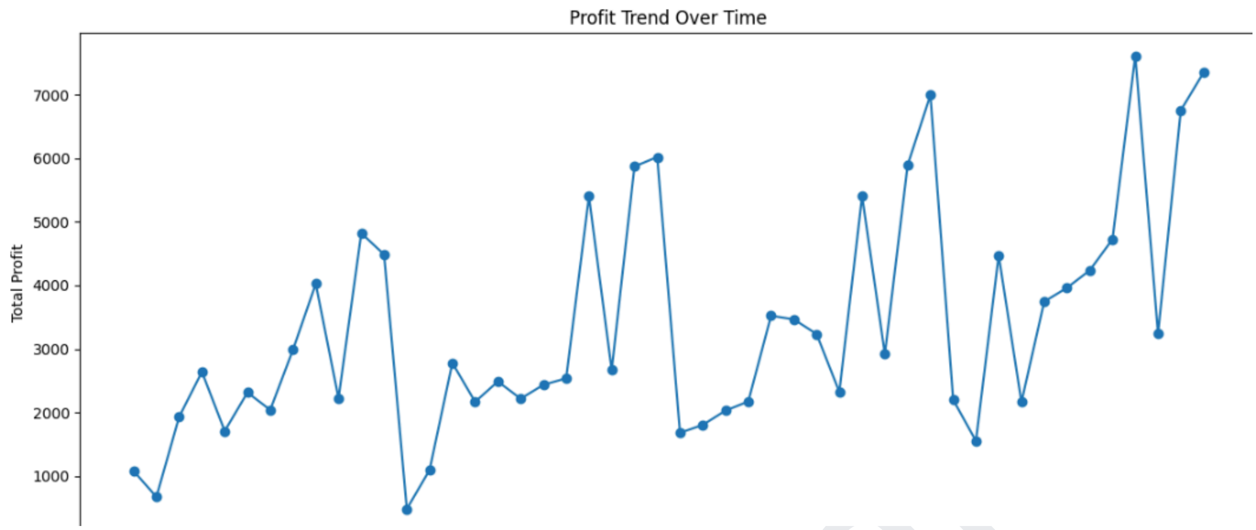
Sales and Profit Trends Over Time

```python
# 1. Sales and Profit Trends Over Time
sales_profit_by_time = orders_df_wrangled.groupby(['OrderYear',
'OrderMonth'])[['Sales', 'Profit']].sum().reset_index()
# Plot sales trend
plt.figure(figsize=(12, 6))
```

```
# Changed 'Order Year' and 'Order Month' to 'OrderYear' and 'OrderMonth' in the plot as well
```

```
plt.plot(sales_profit_by_time['OrderYear'].astype(str) + '-' + sales_profit_by_time['OrderMonth'].astype(str),
         sales_profit_by_time['Sales'], marker='o', linestyle='-')
plt.title('Sales Trend Over Time')
plt.xlabel('Year-Month')
plt.ylabel('Total Sales')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
# Plot profit trend (similar to sales trend)
plt.figure(figsize=(12, 6))
plt.plot(sales_profit_by_time['OrderYear'].astype(str) + '-' + sales_profit_by_time['OrderMonth'].astype(str),
         sales_profit_by_time['Profit'], marker='o', linestyle='-') # Changed to 'Profit' column
plt.title('Profit Trend Over Time')
plt.xlabel('Year-Month')
plt.ylabel('Total Profit')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



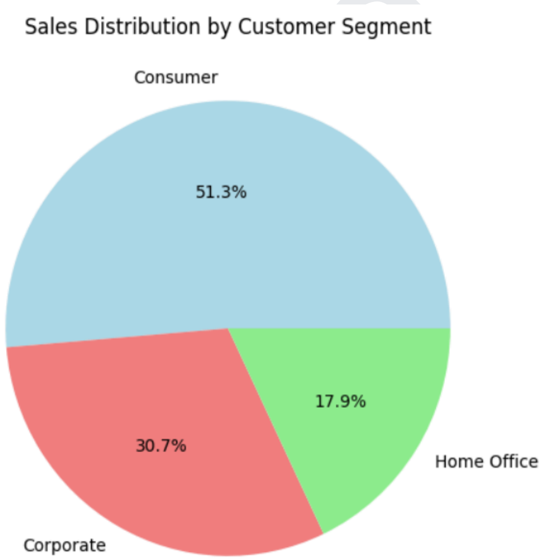


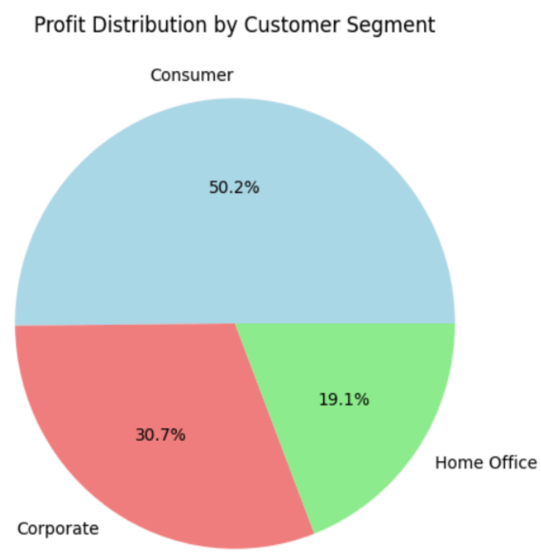
Product Performance

```
 2. Product Performance
# Sales by Category
sales_by_category =
orders_df.groupby('Category')['Sales'].sum().sort_values(ascending=False)
plt.figure(figsize=(8, 6))
sales_by_category.plot(kind='bar', color='skyblue')
plt.title('Sales by Category')
plt.xlabel('Category')
plt.ylabel('Total Sales')
plt.xticks(rotation=0)
plt.show()
 # Sales by Sub-Category (similar to sales by category)
sales_by_category =
orders_df.groupby('Sub-Category')['Sales'].sum().sort_values(ascending=False)
plt.figure(figsize=(15, 10))
sales_by_category.plot(kind='bar', color='skyblue')
plt.title('Sales by Sub-Category')
plt.xlabel('Sub-Category')
plt.ylabel('Total Sales')
plt.xticks(rotation=0)
plt.show()
```

Customer Segmentation and Profitability

```
# 3. Customer Segmentation and Profitability
# Sales by Segment
sales_by_segment =
orders_df.groupby('Segment')['Sales'].sum().sort_values(ascending=False)
plt.figure(figsize=(8, 6))
sales_by_segment.plot(kind='pie', autopct='%1.1f%%', colors=['lightblue',
'lightcoral', 'lightgreen'])
plt.title('Sales Distribution by Customer Segment')
plt.ylabel('')
plt.show()

# Profit by Segment (similar to sales by segment)
Profit_by_segment =
orders_df.groupby('Segment')['Profit'].sum().sort_values(ascending=False)
plt.figure(figsize=(8, 6))
Profit_by_segment.plot(kind='pie', autopct='%1.1f%%', colors=['lightblue',
'lightcoral', 'lightgreen'])
plt.title('Profit Distribution by Customer Segment')
plt.ylabel('')
plt.show()
```





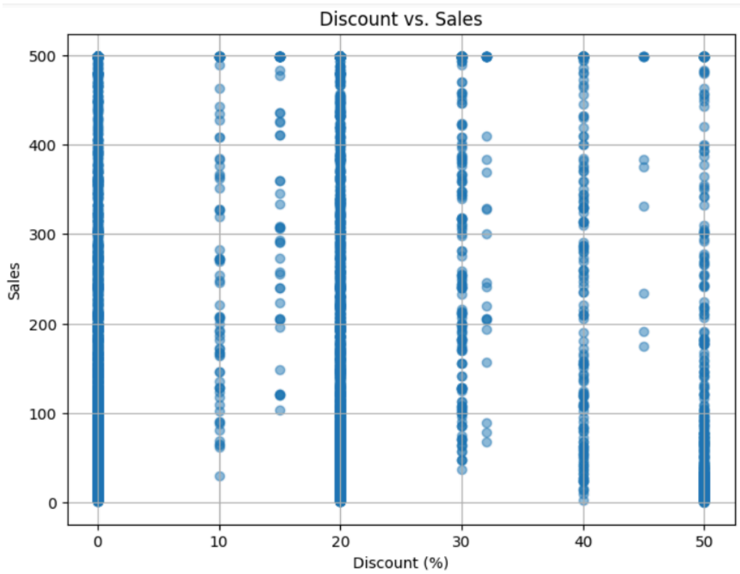
Discount Impact

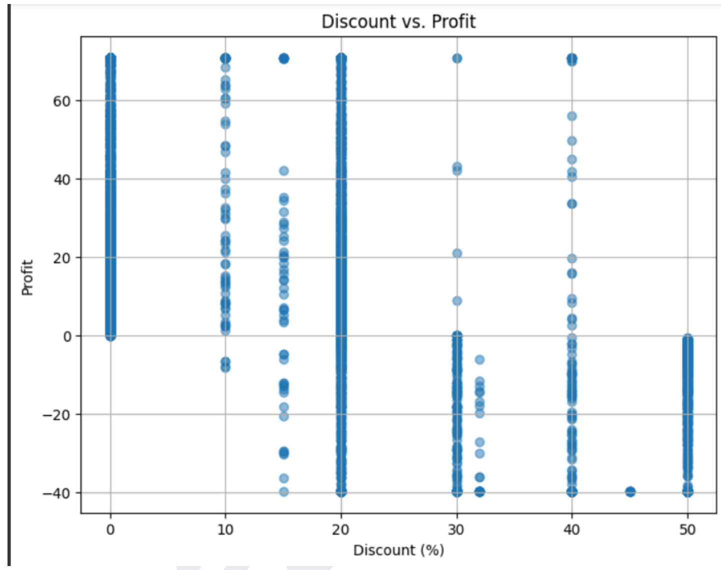
```
# 1. Relationship between Discount and Sales:
# Scatter plot to visualize the relationship
plt.figure(figsize=(8, 6))
plt.scatter(orders_df['Discount'], orders_df['Sales'], alpha=0.5)  # alpha for
transparency
plt.title('Discount vs. Sales')
plt.xlabel('Discount (%)')
plt.ylabel('Sales')
plt.grid(True)
plt.show()

# Calculate correlation
correlation = orders_df['Discount'].corr(orders_df['Sales'])
print(f"Correlation between Discount and Sales: {correlation}")

# 2. Relationship between Discount and Profit:
# Scatter plot to visualize the relationship
plt.figure(figsize=(8, 6))
plt.scatter(orders_df['Discount'], orders_df['Profit'], alpha=0.5)
plt.title('Discount vs. Profit')
plt.xlabel('Discount (%)')
plt.ylabel('Profit')
plt.grid(True)
plt.show()

# Calculate correlation
correlation = orders_df['Discount'].corr(orders_df['Profit'])
print(f"Correlation between Discount and Profit: {correlation}")

# 3. Discount Impact on Sales by Category/Sub-Category:
# Group by category/sub-category and calculate average discount and sales
discount_impact_by_category = orders_df.groupby('Category').agg({'Discount':
'mean', 'Sales': 'mean'})
discount_impact_by_subcategory =
orders_df.groupby('Sub-Category').agg({'Discount': 'mean', 'Sales': 'mean'})

# Visualize with bar charts
discount_impact_by_category.plot(kind='bar', secondary_y='Discount',
figsize=(10, 6))
plt.title('Average Discount and Sales by Category')
plt.xlabel('Category')
plt.ylabel('Average Sales / Discount')
```

```
plt.show()

discount_impact_by_subcategory.plot(kind='bar', secondary_y='Discount',
figsize=(12, 6))
plt.title('Average Discount and Sales by Sub-Category')
plt.xlabel('Sub-Category')
plt.ylabel('Average Sales / Discount')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

# 4. Discount Effectiveness:
# Compare sales with and without discounts
sales_with_discount = orders_df[orders_df['Discount'] > 0]['Sales'].mean()
sales_without_discount = orders_df[orders_df['Discount'] == 0]['Sales'].mean()

print(f"Average Sales with Discount: {sales_with_discount}")
print(f"Average Sales without Discount: {sales_without_discount}")

# Box plot to compare distributions
plt.figure(figsize=(6, 4))
sns.boxplot(x=orders_df['Discount'] > 0, y=orders_df['Sales'])  # Discount > 0
as True/False
plt.title('Sales Distribution with and without Discount')
plt.xlabel('Discount Applied')
plt.ylabel('Sales')
plt.xticks([0, 1], ['No Discount', 'Discount'])
plt.show()
```

Correlation between Discount and Sales: -0.01441869225559738

Correlation between Discount and Profit: -0.5152467760290773

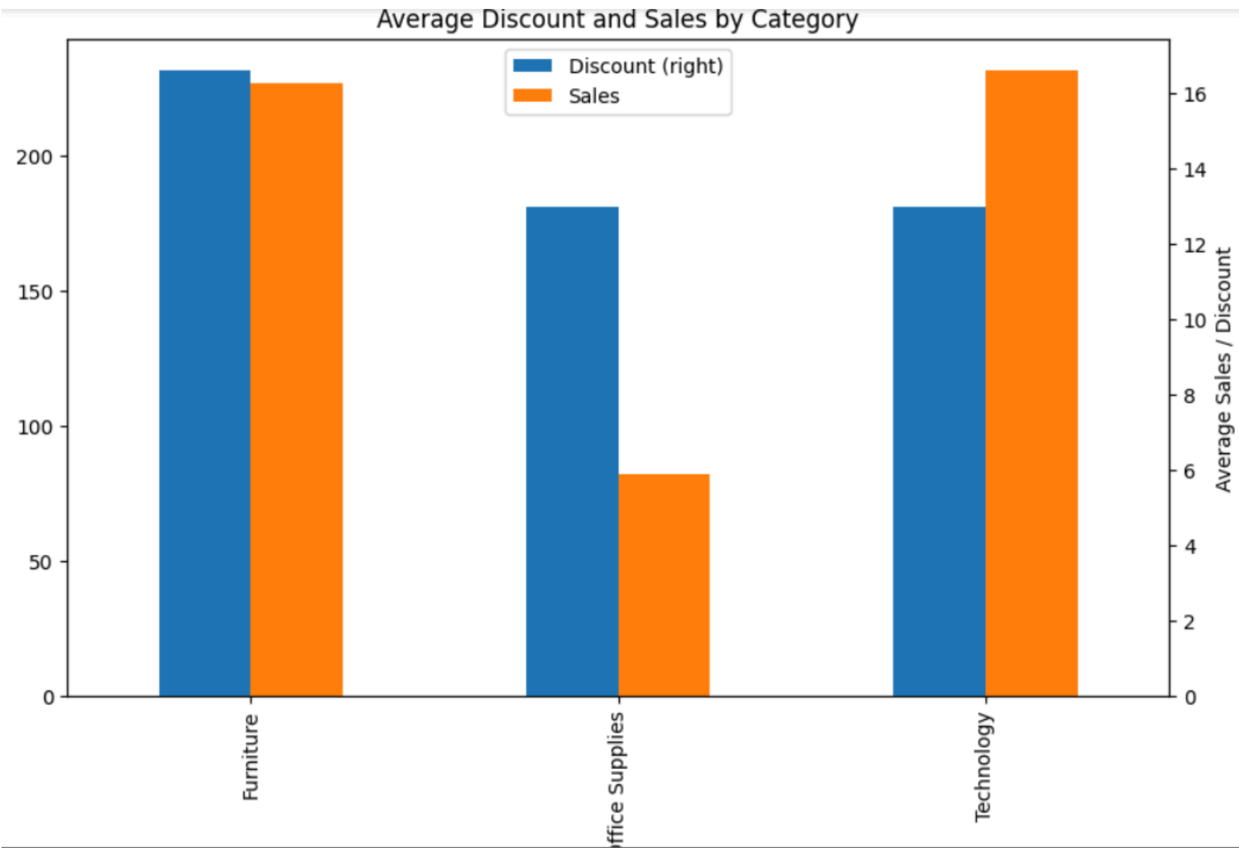


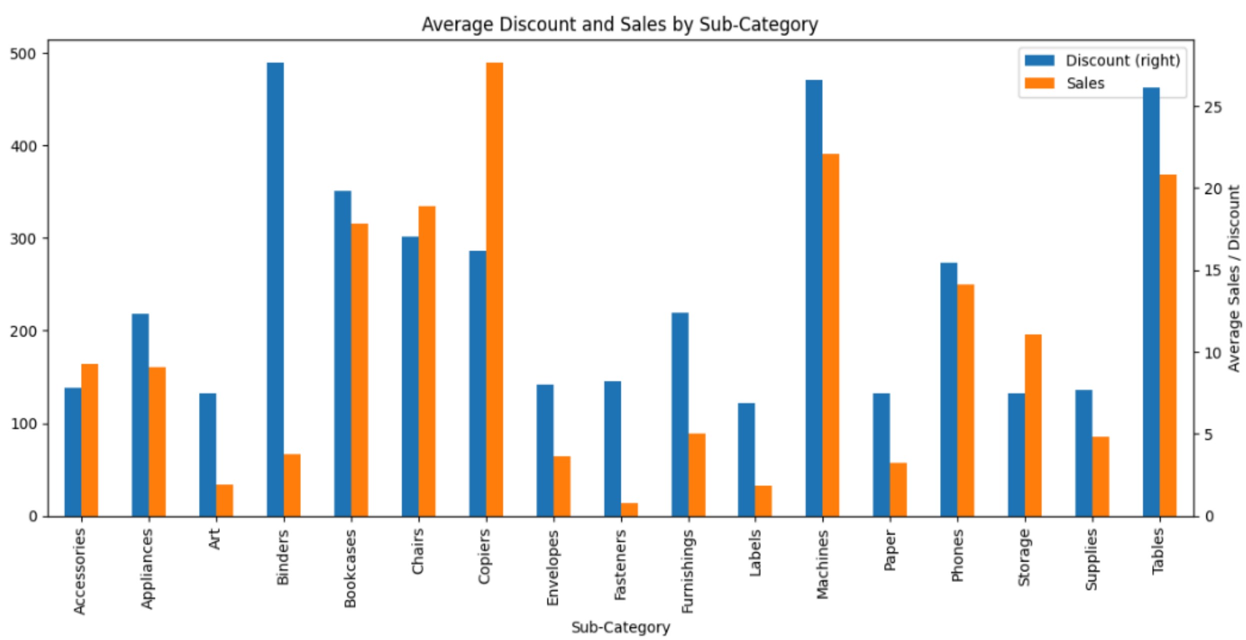


Average Sales with Discount:
146.25859226327944

Average Sales without Discount:
133.80777824093371

Digital Egypt Pioneers





Customer Analysis

Top Customers by Sales Volume

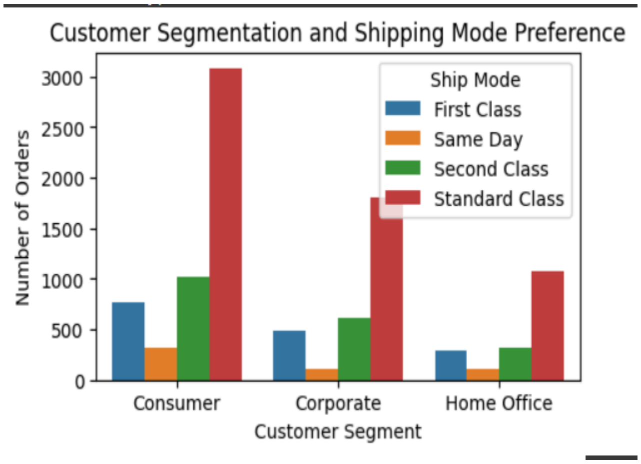
```
top_customers = orders_df.groupby(['Customer ID','Customer
Name'])['Sales'].sum().sort_values(ascending=False).head(10)
print("Top 10 Customers by Sales Volume:")
print(top_customers)


# 1. Customer Segmentation and Shipping Mode Preference:
segment_shipmode_preference = orders_df.groupby(['Segment', 'Ship
Mode'])['Order ID'].count().reset_index()
segment_shipmode_preference.rename(columns={'Order ID': 'OrderCount'},
inplace=True)

plt.figure(figsize=(5, 3))
sns.barplot(data=segment_shipmode_preference, x='Segment', y='OrderCount',
hue='Ship Mode')
plt.title('Customer Segmentation and Shipping Mode Preference')
plt.xlabel('Customer Segment')
plt.ylabel('Number of Orders')
plt.show()

# 2. Average Order Value by Shipping Mode:
avg_order_value_by_shipmode = orders_df.groupby('Ship
Mode')['Sales'].mean().reset_index()
avg_order_value_by_shipmode.rename(columns={'Sales': 'AvgOrderValue'},
inplace=True)

plt.figure(figsize=(5, 3))
sns.barplot(data=avg_order_value_by_shipmode, x='Ship Mode', y='AvgOrderValue')
plt.title('Average Order Value by Shipping Mode')
plt.xlabel('Shipping Mode')
plt.ylabel('Average Order Value')
plt.show()

# 3. Profitability by Shipping Mode:
profit_by_shipmode = orders_df.groupby('Ship
Mode')['Profit'].sum().reset_index()
```

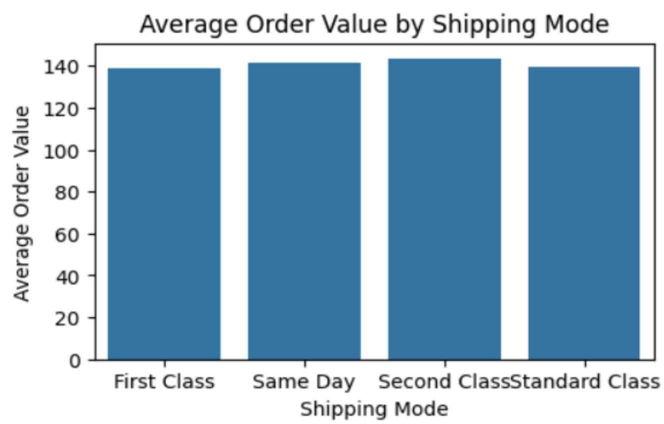
```
plt.figure(figsize=(5, 3))
sns.barplot(data=profit_by_shipmode, x='Ship Mode', y='Profit')
plt.title('Profitability by Shipping Mode')
plt.xlabel('Shipping Mode')
plt.ylabel('Total Profit')
```

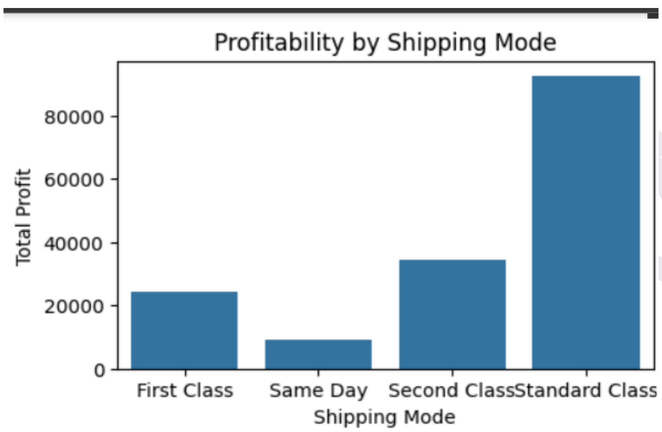
```python
plt.show()

# 4. Return Rate by Shipping Mode:
# Merge orders and returns data to calculate return rate
merged_df = pd.merge(orders_df, returns_df, left_on='Order ID',
right_on='order_id', how='left')
merged_df['Returned'] =
merged_df['order_id'].isin(returns_df['order_id']).astype(int) # 1 if returned,
0 otherwise
return_rate_by_shipmode = merged_df.groupby('Ship
Mode')['Returned'].mean().reset_index()
return_rate_by_shipmode.rename(columns={'Returned': 'ReturnRate'},
inplace=True)

plt.figure(figsize=(5, 3))
sns.barplot(data=return_rate_by_shipmode, x='Ship Mode', y='ReturnRate')
plt.title('Return Rate by Shipping Mode')
plt.xlabel('Shipping Mode')
plt.ylabel('Return Rate')
plt.show()
```

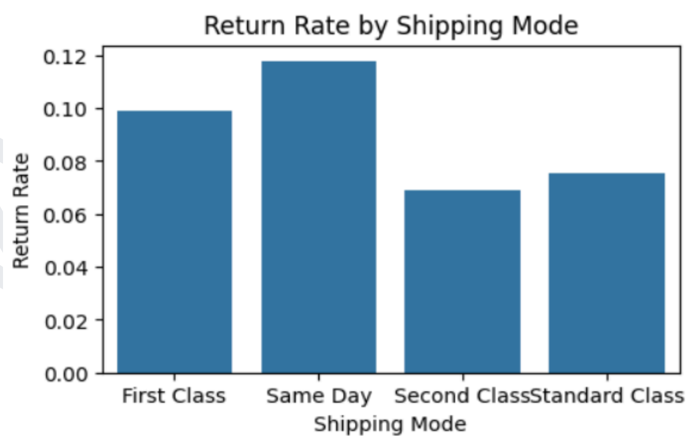
Top 10 Customers by Sales Volume:

Customer ID	Customer Name	
CL-12565	Clay Ludtke	5794.622
SV-20365	Seth Vernon	5775.551
JL-15835	John Lee	5334.072
LA-16780	Laura Armstrong	5248.588
PP-18955	Paul Prost	5194.788
BM-11650	Brian Moss	5129.325
WB-21850	William Brown	5127.694
DR-12880	Dan Reichenbach	5114.498
ME-17320	Maria Etezadi	5036.768
KL-16645	Ken Lonsdale	4868.481









Data preparation

Prepare the data for time series forecasting.Create a new dataframe with relevant columns for forecasting, set 'Order Date' as DateTimeIndex, aggregate sales by month, and handle missing values or outliers.

```
# Select relevant columns
forecast_df = orders_df_wrangled[['Order Date', 'Sales', 'OrderYear',
'OrderMonth', 'OrderQuarter', 'Category', 'Sub-Category', 'Region',
'CombinedLocation']].copy()
```

```
# Set 'Order Date' as DateTimeIndex
```

```
forecast_df['Order Date'] = pd.to_datetime(forecast_df['Order Date'])
forecast_df = forecast_df.set_index('Order Date')
# Aggregate to monthly data
# Create the monthly_sales DataFrame by grouping and summing sales
monthly_sales = forecast_df.groupby(['OrderYear',
'OrderMonth'])['Sales'].sum().reset_index()
#The rest of your code remains the same
monthly_sales['Order Date'] = pd.to_datetime(monthly_sales.apply(lambda x:
f'{int(x.OrderYear)}-{int(x.OrderMonth):02d}-01', axis=1))
monthly_sales = monthly_sales.set_index('Order Date')

monthly_sales.columns = ['OrderYear', 'OrderMonth', 'Sales']
# Create 'Order Date' column using the 'OrderYear' and 'OrderMonth' columns
monthly_sales['Order Date'] = pd.to_datetime(monthly_sales.apply(lambda x:
f'{int(x.OrderYear)}-{int(x.OrderMonth):02d}-01', axis=1))
# Instead of setting 'Order Date' as the index again, keep it as a column
#monthly_sales = monthly_sales.set_index('Order Date') #Comment out this line
# Handle missing values or outliers (if any) - check for missing values
print(monthly_sales.isnull().sum())

display(monthly_sales.head())
```

	OrderYear	OrderMonth	Sales	Order Date
Order Date				
2014-01-01	2014	1	9928.525	2014-01-01
2014-02-01	2014	2	3762.602	2014-02-01
2014-03-01	2014	3	22431.136	2014-03-01
2014-04-01	2014	4	18905.141	2014-04-01
2014-05-01	2014	5	16443.875	2014-05-01

Model training

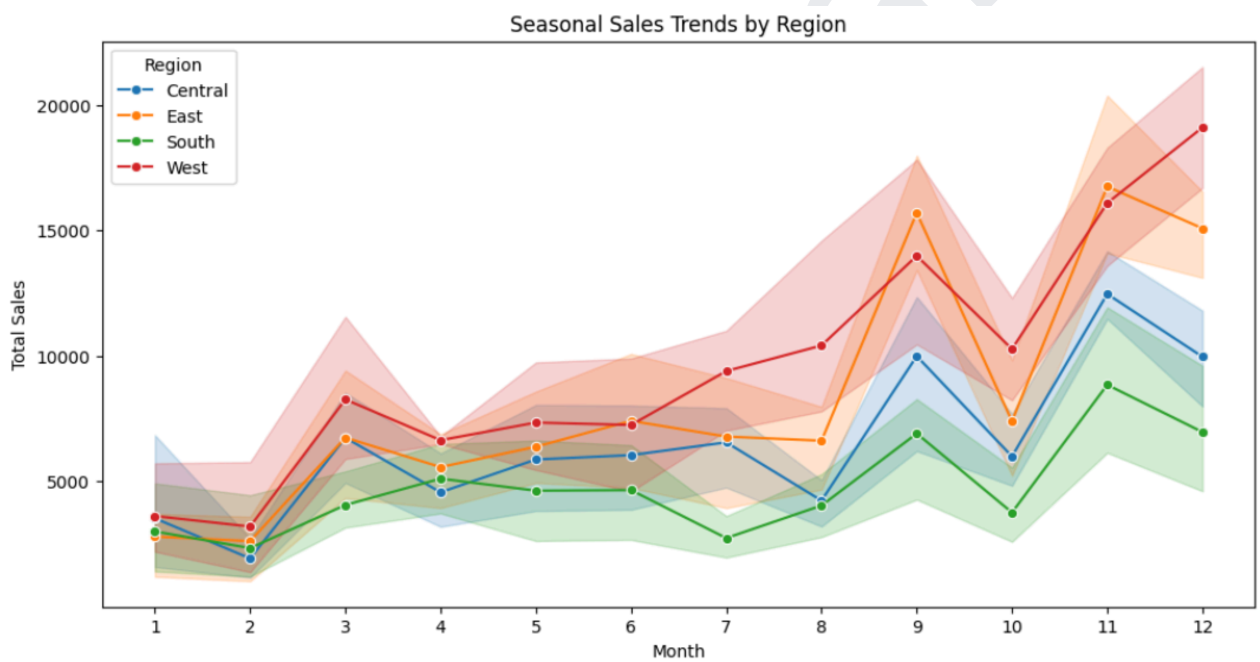
Train a time series forecasting model to predict future sales.Train a time series forecasting model on the monthly_sales data. Split the data into training and testing sets, fit the model, and generate predictions. I will use the Prophet model.

```python
from prophet import Prophet
import pandas as pd
# Prepare the data for Prophet
monthly_sales_prophet = monthly_sales.rename(columns={'Order Date': 'ds',
'Sales': 'y'})[['ds', 'y']]
monthly_sales_prophet.columns = ['ds', 'y']
# Split the data into training and testing sets
train_size = int(len(monthly_sales_prophet) * 0.8)
train_data = monthly_sales_prophet[:train_size]
test_data = monthly_sales_prophet[train_size:]
# Initialize and fit the Prophet model
model = Prophet()
model.fit(train_data)
# Generate future dates for forecasting
future = model.make_future_dataframe(periods=len(test_data), freq='MS')
# Make predictions
predictions = model.predict(future)

# Extract the predicted sales for the test period
predictions = predictions[['ds', 'yhat']].tail(len(test_data))
predictions.columns = ['ds', 'yhat']
```

```python
print(predictions.head())
```

```
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmpsncwpqsf/4enkxouw.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmpsncwpqsf/b23h5poz.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/stan_model/prophet_model.bin',
06:41:45 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
06:41:45 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
           ds          yhat
38 2017-03-01  31479.753222
39 2017-04-01  28104.086321
40 2017-05-01  31021.845327
41 2017-06-01  31342.045606
42 2017-07-01  32186.977770
```

```
# Aggregate sales data by region, year, and month
regional_sales = orders_df_wrangled.groupby(['Region', 'OrderYear',
'OrderMonth'])['Sales'].sum().reset_index()
# Create a line chart with separate lines for each region
plt.figure(figsize=(12, 6))
sns.lineplot(data=regional_sales, x='OrderMonth', y='Sales', hue='Region',
marker='o')
plt.title('Seasonal Sales Trends by Region')
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.xticks(range(1, 13))  # Set x-axis ticks for months
plt.legend(title='Region')
plt.show()
```



Data visualization

Create visualizations to communicate insights from the data analysis and forecasting steps.Visualize the sales trend, sales by category and sub-category, sales by customer segment, and overlay actual vs. predicted sales.

```
!pip install prophet
from prophet import Prophet
```

```python
import pandas as pd
# Select relevant columns
forecast_df = orders_df_wrangled[['Order Date', 'Sales', 'OrderYear',
'OrderMonth', 'OrderQuarter', 'Category', 'Sub-Category', 'Region',
'CombinedLocation']].copy()
# Set 'Order Date' as DateTimeIndex
forecast_df['Order Date'] = pd.to_datetime(forecast_df['Order Date'])
forecast_df = forecast_df.set_index('Order Date')
# Aggregate to monthly data
# Create the monthly_sales DataFrame by grouping and summing sales
monthly_sales = forecast_df.groupby(['OrderYear',
'OrderMonth'])['Sales'].sum().reset_index()
#The rest of your code remains the same
monthly_sales['Order Date'] = pd.to_datetime(monthly_sales.apply(lambda x:
f'{int(x.OrderYear)}-{int(x.OrderMonth):02d}-01', axis=1))
monthly_sales = monthly_sales.set_index('Order Date')
monthly_sales.columns = ['OrderYear', 'OrderMonth', 'Sales']
# Create 'Order Date' column using the 'OrderYear' and 'OrderMonth' columns
monthly_sales['Order Date'] = pd.to_datetime(monthly_sales.apply(lambda x:
f'{int(x.OrderYear)}-{int(x.OrderMonth):02d}-01', axis=1))
# Instead of setting 'Order Date' as the index again, keep it as a column
#monthly_sales = monthly_sales.set_index('Order Date') #Comment out this line
# Handle missing values or outliers (if any) - check for missing values
print(monthly_sales.isnull().sum())
```

```python
display(monthly_sales.head())
# Prepare the data for Prophet
monthly_sales_prophet = monthly_sales.rename(columns={'Order Date': 'ds',
'Sales': 'y'})[['ds', 'y']]
monthly_sales_prophet.columns = ['ds', 'y']
# Split the data into training and testing sets
train_size = int(len(monthly_sales_prophet) * 0.8)
train_data = monthly_sales_prophet[:train_size]
test_data = monthly_sales_prophet[train_size:]
# Initialize and fit the Prophet model
model = Prophet()
```

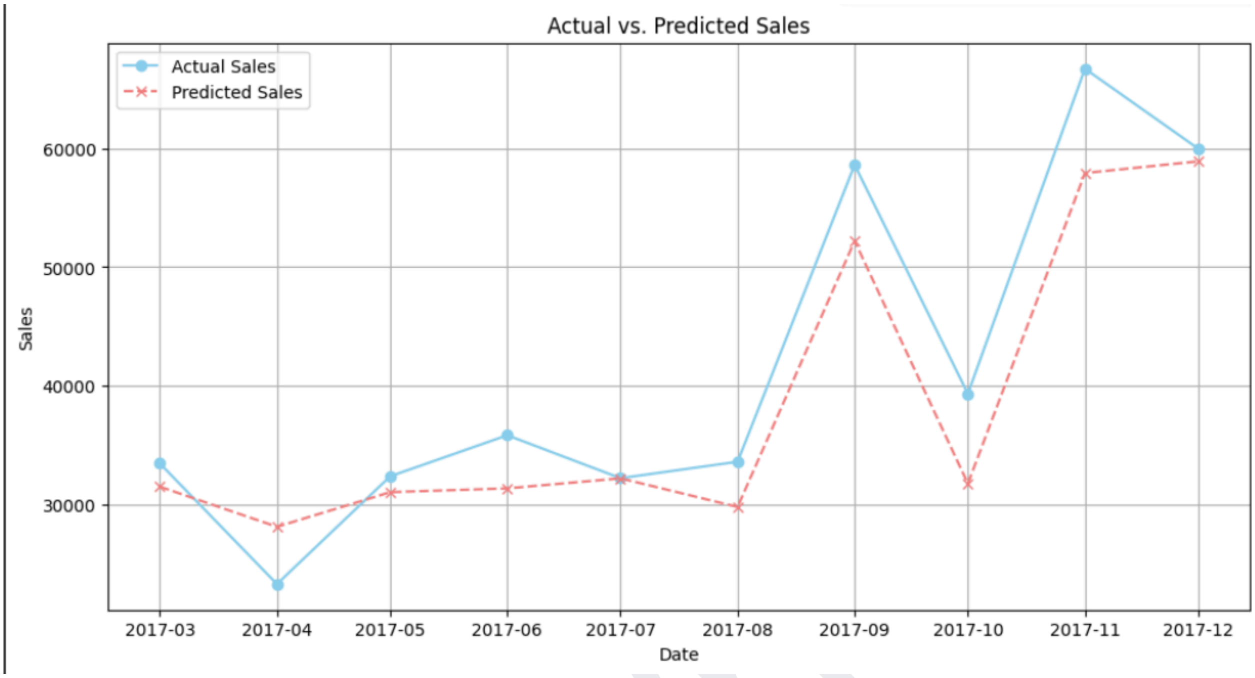
```python
model.fit(train_data)
# Generate future dates for forecasting
future = model.make_future_dataframe(periods=len(test_data), freq='MS')
# Make predictions
```

```python
predictions = model.predict(future)
```

```python
# Extract the predicted sales for the test period
predictions = predictions[['ds', 'yhat']].tail(len(test_data))
predictions.columns = ['ds', 'yhat']
# Forecast Visualization: Actual vs. Predicted Sales
plt.figure(figsize=(12, 6))
plt.plot(test_data['ds'], test_data['y'], label='Actual Sales', marker='o',
linestyle='-', color='skyblue')
plt.plot(predictions['ds'], predictions['yhat'], label='Predicted Sales',
marker='x', linestyle='--', color='lightcoral') # This line was causing the
error
plt.title('Actual vs. Predicted Sales')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.legend()
plt.grid(True)
plt.show()
```

	OrderYear	OrderMonth	Sales	Order Date
Order Date				
2014-01-01	2014	1	9928.525	2014-01-01
2014-02-01	2014	2	3762.602	2014-02-01
2014-03-01	2014	3	22431.136	2014-03-01
2014-04-01	2014	4	18905.141	2014-04-01
2014-05-01	2014	5	16443.875	2014-05-01

```
INFO:prophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmp1jm_by9w/daz16roq.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmp1jm_by9w/8zpqoydn.json
DEBUG:cmdstanpy:idx 0
```



Summary

This analysis of Superstore's sales, profitability, customer behavior, and operations provides valuable insights to drive data-driven decision-making and business growth. Below are the key findings and recommendations:

1- Sales & Profitability Trends Sales and profits are growing, but fluctuations highlight the need to track market trends and adjust strategies accordingly.

Seasonal patterns impact sales, requiring proactive inventory management and targeted marketing campaigns during peak periods.

Regional variations in sales suggest the need for localized strategies—customized marketing, product selection, and resource allocation.

2- Product Performance & Revenue Optimization Top-performing product categories significantly drive revenue, making it essential to focus on high-impact products.

Low-performing products should be reviewed for optimization, promotional strategies, or discontinuation.

Discount strategies influence sales differently, so pricing adjustments should be tailored based on product performance and demand.

3-Customer Insights & Personalization Customer segmentation (Consumer, Corporate, Home Office) reveals different buying behaviors. Personalized marketing and recommendations can improve engagement and sales.

Top customers contribute significantly to sales, making it crucial to develop customer retention programs and personalized loyalty incentives.

Geographic analysis enables better regional marketing strategies and product assortment optimization based on demand.

4- Shipping, Returns & Customer Experience Shipping preferences vary by segment and order value, requiring flexible shipping options to enhance the buying experience.

High return rates for certain products/shipping modes indicate the need for packaging, fulfillment, and product quality improvements.

Reducing return rates by addressing root causes (product descriptions, shipping damage) can lower costs and improve customer satisfaction.

5-Seasonality and Regional Sales:

The analysis revealed that seasonality has varying effects on sales in different regions. Some regions exhibit similar seasonal patterns, while others show significant differences. This indicates that factors other than just seasonality, such as weather, local events, or cultural differences, might influence sales.

Key Business Recommendations:

-Focus on best-selling products while managing underperforming items strategically.

-Use customer data for personalized, targeted campaigns based on region, seasonality, and buying behavior.

-Strengthen customer service, product quality, and website usability to boost satisfaction and loyalty.

-Align inventory with sales forecasts and demand trends to prevent stock issues.

Reference:

Link of Project in google colab:

https://colab.research.google.com/drive/16jgjzsfvGYximrkfajqd_U9EK-nTKIx-?usp=sharing

Link of the Administration Data Set :

https://drive.google.com/file/d/1c-5xFgP2SIgE-SJOCK9zpQ26WvTQB-tf/view?usp=drive_link

Link of the Alternative Data Set :

https://docs.google.com/spreadsheets/d/1xehH4_9_wdO2hTqq6rHUSCH-P10vB9mU/edit?rtpof=true&gid=1612851729#gid=1612851729

Link Git hub of the Project:

https://github.com/Asmaahk/Super-Store-Data-Whales-