Understand Dataset

```
import pandas as pd
df = pd.read_csv("Sales.csv",encoding="ISO-8859-1")
df.head()
```

{"type":"dataframe","variable_name":"df"}

```
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   object 
 3   Ship Date      9994 non-null   object 
 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
dtypes: float64(3), int64(3), object(15)
memory usage: 1.6+ MB
```

```
df.describe()
```

{"summary":"{\n \"name\": \"df\",\n \"rows\": 8,\n \"fields\": [\n
{\n   \"column\": \"Row ID\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 3601.5811575098865,\n
\"min\": 1.0,\n   \"max\": 9994.0,\n
\"num_unique_values\": 6,\n   \"samples\": [\n   9994.0,\n
4997.5,\n   7495.75\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Postal Code\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 35860.31406157157,\n

```
\"min\": 1040.0,\n        \"max\": 99301.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
55190.3794276566,\n          56430.5,\n          9994.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Sales\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 8197.010918685499,\n
\"min\": 0.444,\n        \"max\": 22638.48,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
229.85800083049833,\n          54.489999999999995,\n          9994.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Quantity\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3531.848471644344,\n        \"min\": 1.0,\n        \"max\": 9994.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
3.789573744246548,\n          3.0,\n          9994.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Discount\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
3533.3336684667293,\n        \"min\": 0.0,\n        \"max\": 9994.0,\n
\"num_unique_values\": 6,\n        \"samples\": [\n          9994.0,\n
0.1562027216329797 7,\n          0.8\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Profit\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
5288.326642672474,\n        \"min\": -6599.978,\n        \"max\":
9994.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
28.65689630778467,\n          8.6665,\n          9994.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```

```python
# Check unique ship modes

unique_ship_modes_count = df['Ship Mode'].nunique()
print(f"Number of unique Ship Modes: {unique_ship_modes_count}")
print(df['Ship Mode'].unique())
```

```
Number of unique Ship Modes: 4
['Second Class' 'Standard Class' 'First Class' 'Same Day']
```

```python
# Check unique Customers

unique_customer_name_count = df['Customer Name'].nunique()
print(f"Number of unique Scustomer Name:
{unique_customer_name_count}")
```

```
Number of unique Scustomer Name: 793
```

```python
# Check unique Segment

unique_Segment_count = df['Segment'].nunique()
```

```python
print(f"Number of unique Segment: {unique_Segment_count}")
print(df['Segment'].unique())
```

```
Number of unique Segment: 3
['Consumer' 'Corporate' 'Home Office']
```

```python
# Check unique Country

unique_Country_count = df['Country'].nunique()
print(f"Number of unique Country: {unique_Country_count}")
print(df['Country'].unique())
```

```
Number of unique Country: 1
['United States']
```

```python
# Check unique State

unique_State_count = df['State'].nunique()
print(f"Number of unique State: {unique_State_count}")
print(df['State'].unique())
```

```
Number of unique State: 49
['Kentucky' 'California' 'Florida' 'North Carolina' 'Washington'
'Texas'
 'Wisconsin' 'Utah' 'Nebraska' 'Pennsylvania' 'Illinois' 'Minnesota'
 'Michigan' 'Delaware' 'Indiana' 'New York' 'Arizona' 'Virginia'
 'Tennessee' 'Alabama' 'South Carolina' 'Oregon' 'Colorado' 'Iowa'
'Ohio'
 'Missouri' 'Oklahoma' 'New Mexico' 'Louisiana' 'Connecticut' 'New
Jersey'
 'Massachusetts' 'Georgia' 'Nevada' 'Rhode Island' 'Mississippi'
 'Arkansas' 'Montana' 'New Hampshire' 'Maryland' 'District of
Columbia'
 'Kansas' 'Vermont' 'Maine' 'South Dakota' 'Idaho' 'North Dakota'
 'Wyoming' 'West Virginia']
```

```python
# Check unique City

unique_City_count = df['City'].nunique()
print(f"Number of unique City: {unique_City_count}")
print(df['City'].unique())
```

```
Number of unique City: 531
['Henderson' 'Los Angeles' 'Fort Lauderdale' 'Concord' 'Seattle'
 'Fort Worth' 'Madison' 'West Jordan' 'San Francisco' 'Fremont'
 'Philadelphia' 'Orem' 'Houston' 'Richardson' 'Naperville' 'Melbourne'
 'Eagan' 'Westland' 'Dover' 'New Albany' 'New York City' 'Troy'
'Chicago'
 'Gilbert' 'Springfield' 'Jackson' 'Memphis' 'Decatur' 'Durham'
'Columbia'
 'Rochester' 'Minneapolis' 'Portland' 'Saint Paul' 'Aurora'
```

```
'Charlotte'
 'Orland Park' 'Urbandale' 'Columbus' 'Bristol' 'Wilmington'
'Bloomington'
 'Phoenix' 'Roseville' 'Independence' 'Pasadena' 'Newark' 'Franklin'
 'Scottsdale' 'San Jose' 'Edmond' 'Carlsbad' 'San Antonio' 'Monroe'
 'Fairfield' 'Grand Prairie' 'Redlands' 'Hamilton' 'Westfield' 'Akron'
 'Denver' 'Dallas' 'Whittier' 'Saginaw' 'Medina' 'Dublin' 'Detroit'
 'Tampa' 'Santa Clara' 'Lakeville' 'San Diego' 'Brentwood' 'Chapel
Hill'
 'Morristown' 'Cincinnati' 'Inglewood' 'Tamarac' 'Colorado Springs'
 'Belleville' 'Taylor' 'Lakewood' 'Arlington' 'Arvada' 'Hackensack'
 'Saint Petersburg' 'Long Beach' 'Hesperia' 'Murfreesboro' 'Layton'
 'Austin' 'Lowell' 'Manchester' 'Harlingen' 'Tucson' 'Quincy'
 'Pembroke Pines' 'Des Moines' 'Peoria' 'Las Vegas' 'Warwick' 'Miami'
 'Huntington Beach' 'Richmond' 'Louisville' 'Lawrence' 'Canton'
 'New Rochelle' 'Gastonia' 'Jacksonville' 'Auburn' 'Norman' 'Park
Ridge'
 'Amarillo' 'Lindenhurst' 'Huntsville' 'Fayetteville' 'Costa Mesa'
 'Parker' 'Atlanta' 'Gladstone' 'Great Falls' 'Lakeland' 'Montgomery'
 'Mesa' 'Green Bay' 'Anaheim' 'Marysville' 'Salem' 'Laredo' 'Grove
City'
 'Dearborn' 'Warner Robins' 'Vallejo' 'Mission Viejo' 'Rochester
Hills'
 'Plainfield' 'Sierra Vista' 'Vancouver' 'Cleveland' 'Tyler'
'Burlington'
 'Waynesboro' 'Chester' 'Cary' 'Palm Coast' 'Mount Vernon' 'Hialeah'
 'Oceanside' 'Evanston' 'Trenton' 'Cottage Grove' 'Bossier City'
 'Lancaster' 'Asheville' 'Lake Elsinore' 'Omaha' 'Edmonds' 'Santa Ana'
 'Milwaukee' 'Florence' 'Lorain' 'Linden' 'Salinas' 'New Brunswick'
 'Garland' 'Norwich' 'Alexandria' 'Toledo' 'Farmington' 'Riverside'
 'Torrance' 'Round Rock' 'Boca Raton' 'Virginia Beach' 'Murrieta'
 'Olympia' 'Washington' 'Jefferson City' 'Saint Peters' 'Rockford'
 'Brownsville' 'Yonkers' 'Oakland' 'Clinton' 'Encinitas' 'Roswell'
 'Jonesboro' 'Antioch' 'Homestead' 'La Porte' 'Lansing' 'Cuyahoga
Falls'
 'Reno' 'Harrisonburg' 'Escondido' 'Royal Oak' 'Rockville' 'Coral
Springs'
 'Buffalo' 'Boynton Beach' 'Gulfport' 'Fresno' 'Greenville' 'Macon'
 'Cedar Rapids' 'Providence' 'Pueblo' 'Deltona' 'Murray' 'Middletown'
 'Freeport' 'Pico Rivera' 'Provo' 'Pleasant Grove' 'Smyrna' 'Parma'
 'Mobile' 'New Bedford' 'Irving' 'Vineland' 'Glendale' 'Niagara Falls'
 'Thomasville' 'Westminster' 'Coppell' 'Pomona' 'North Las Vegas'
 'Allentown' 'Tempe' 'Laguna Niguel' 'Bridgeton' 'Everett' 'Watertown'
 'Appleton' 'Bellevue' 'Allen' 'El Paso' 'Grapevine' 'Carrollton'
'Kent'
 'Lafayette' 'Tigard' 'Skokie' 'Plano' 'Suffolk' 'Indianapolis'
'Bayonne'
 'Greensboro' 'Baltimore' 'Kenosha' 'Olathe' 'Tulsa' 'Redmond'
'Raleigh'
```

```
 'Muskogee' 'Meriden' 'Bowling Green' 'South Bend' 'Spokane' 'Keller'
 'Port Orange' 'Medford' 'Charlottesville' 'Missoula' 'Apopka'
'Reading'
 'Broomfield' 'Paterson' 'Oklahoma City' 'Chesapeake' 'Lubbock'
 'Johnson City' 'San Bernardino' 'Leominster' 'Bozeman' 'Perth Amboy'
 'Ontario' 'Rancho Cucamonga' 'Moorhead' 'Mesquite' 'Stockton'
 'Ormond Beach' 'Sunnyvale' 'York' 'College Station' 'Saint Louis'
 'Manteca' 'San Angelo' 'Salt Lake City' 'Knoxville' 'Little Rock'
 'Lincoln Park' 'Marion' 'Littleton' 'Bangor' 'Southaven' 'New Castle'
 'Midland' 'Sioux Falls' 'Fort Collins' 'Clarksville' 'Sacramento'
 'Thousand Oaks' 'Malden' 'Holyoke' 'Albuquerque' 'Sparks' 'Coachella'
 'Elmhurst' 'Passaic' 'North Charleston' 'Newport News' 'Jamestown'
 'Mishawaka' 'La Quinta' 'Tallahassee' 'Nashville' 'Bellingham'
 'Woodstock' 'Haltom City' 'Wheeling' 'Summerville' 'Hot Springs'
 'Englewood' 'Las Cruces' 'Hoover' 'Frisco' 'Vacaville' 'Waukesha'
 'Bakersfield' 'Pompano Beach' 'Corpus Christi' 'Redondo Beach'
'Orlando'
 'Orange' 'Lake Charles' 'Highland Park' 'Hempstead' 'Noblesville'
 'Apple Valley' 'Mount Pleasant' 'Sterling Heights' 'Eau Claire'
'Pharr'
 'Billings' 'Gresham' 'Chattanooga' 'Meridian' 'Bolingbrook' 'Maple
Grove'
 'Woodland' 'Missouri City' 'Pearland' 'San Mateo' 'Grand Rapids'
 'Visalia' 'Overland Park' 'Temecula' 'Yucaipa' 'Revere' 'Conroe'
 'Tinley Park' 'Dubuque' 'Dearborn Heights' 'Santa Fe' 'Hickory'
 'Carol Stream' 'Saint Cloud' 'North Miami' 'Plantation'
 'Port Saint Lucie' 'Rock Hill' 'Odessa' 'West Allis' 'Chula Vista'
 'Manhattan' 'Altoona' 'Thornton' 'Champaign' 'Texarkana' 'Edinburg'
 'Baytown' 'Greenwood' 'Woonsocket' 'Superior' 'Bedford' 'Covington'
 'Broken Arrow' 'Miramar' 'Hollywood' 'Deer Park' 'Wichita' 'Mcallen'
 'Iowa City' 'Boise' 'Cranston' 'Port Arthur' 'Citrus Heights'
 'The Colony' 'Daytona Beach' 'Bullhead City' 'Portage' 'Fargo'
'Elkhart'
 'San Gabriel' 'Margate' 'Sandy Springs' 'Mentor' 'Lawton' 'Hampton'
 'Rome' 'La Crosse' 'Lewiston' 'Hattiesburg' 'Danville' 'Logan'
 'Waterbury' 'Athens' 'Avondale' 'Marietta' 'Yuma' 'Wausau' 'Pasco'
 'Oak Park' 'Pensacola' 'League City' 'Gaithersburg' 'Lehi'
'Tuscaloosa'
 'Moreno Valley' 'Georgetown' 'Loveland' 'Chandler' 'Helena'
'Kirkwood'
 'Waco' 'Frankfort' 'Bethlehem' 'Grand Island' 'Woodbury' 'Rogers'
 'Clovis' 'Jupiter' 'Santa Barbara' 'Cedar Hill' 'Norfolk' 'Draper'
 'Ann Arbor' 'La Mesa' 'Pocatello' 'Holland' 'Milford' 'Buffalo Grove'
 'Lake Forest' 'Redding' 'Chico' 'Utica' 'Conway' 'Cheyenne'
'Owensboro'
 'Caldwell' 'Kenner' 'Nashua' 'Bartlett' 'Redwood City' 'Lebanon'
 'Santa Maria' 'Des Plaines' 'Longview' 'Hendersonville' 'Waterloo'
 'Cambridge' 'Palatine' 'Beverly' 'Eugene' 'Oxnard' 'Renton'
'Glenview'
```

```
 'Delray Beach' 'Commerce City' 'Texas City' 'Wilson' 'Rio Rancho'
 'Goldsboro' 'Montebello' 'El Cajon' 'Beaumont' 'West Palm Beach'
 'Abilene' 'Normal' 'Saint Charles' 'Camarillo' 'Hillsboro' 'Burbank'
 'Modesto' 'Garden City' 'Atlantic City' 'Longmont' 'Davis' 'Morgan
Hill'
 'Clifton' 'Sheboygan' 'East Point' 'Rapid City' 'Andover' 'Kissimmee'
 'Shelton' 'Danbury' 'Sanford' 'San Marcos' 'Greeley' 'Mansfield'
'Elyria'
 'Twin Falls' 'Coral Gables' 'Romeoville' 'Marlborough' 'Laurel'
'Bryan'
 'Pine Bluff' 'Aberdeen' 'Hagerstown' 'East Orange' 'Arlington
Heights'
 'Oswego' 'Coon Rapids' 'San Clemente' 'San Luis Obispo' 'Springdale'
 'Lodi' 'Mason']
```

```python
# Check unique Region

unique_Region_count = df['Region'].nunique()
print(f"Number of unique City: {unique_Region_count}")
print(df['Region'].unique())
```

```
Number of unique City: 4
['South' 'West' 'Central' 'East']
```

```python
# Check unique Category

unique_Category_count = df['Category'].nunique()
print(f"Number of unique Category: {unique_Category_count}")
print(df['Category'].unique())
```

```
Number of unique Category: 3
['Furniture' 'Office Supplies' 'Technology']
```

```python
# Check unique Sub-Category

unique_Sub_Category_count = df['Sub-Category'].nunique()
print(f"Number of unique Sub-Category: {unique_Sub_Category_count}")
print(df['Sub-Category'].unique())
```

```
Number of unique Sub-Category: 17
['Bookcases' 'Chairs' 'Labels' 'Tables' 'Storage' 'Furnishings' 'Art'
 'Phones' 'Binders' 'Appliances' 'Paper' 'Accessories' 'Envelopes'
 'Fasteners' 'Supplies' 'Machines' 'Copiers']
```

```python
# Find numerical columns
numerical_columns =
df.select_dtypes(include=['number']).columns.tolist()

print("Numerical Columns:", numerical_columns)
```

```
Numerical Columns: ['Row ID', 'Postal Code', 'Sales', 'Quantity',
'Discount', 'Profit']
```

```python
# Find non-numerical columns
non_numerical_columns =
df.select_dtypes(exclude=['number']).columns.tolist()

print("Non-Numerical Columns:", non_numerical_columns)
```

```
Non-Numerical Columns: ['Order ID', 'Order Date', 'Ship Date', 'Ship
Mode', 'Customer ID', 'Customer Name', 'Segment', 'Country', 'City',
'State', 'Region', 'Product ID', 'Category', 'Sub-Category', 'Product
Name']
```

Data preprocessing

```python
# Check for missing values in each column
missing_values = df.isnull().sum()
print("Missing Values:")
print(missing_values)
```

```
Missing Values:
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            0
Quantity         0
Discount         0
Profit           0
dtype: int64
```

```python
# Select numerical columns
numerical_columns = df.select_dtypes(include=['number']).columns

# Calculate IQR for each numerical column
for column in numerical_columns:
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Identify outliers
    outliers = df[(df[column] < lower_bound) | (df[column] >
upper_bound)]
    print(f"Outliers in {column}: {len(outliers)}")
```

```
Outliers in Row ID: 0
Outliers in Postal Code: 0
Outliers in Sales: 1167
Outliers in Quantity: 170
Outliers in Discount: 856
Outliers in Profit: 1881
```

```
# Check for duplicate rows
duplicates = df.duplicated()
print("Number of duplicates:", duplicates.sum())
```

```
Number of duplicates: 0
```

```
df['Order Date'] = pd.to_datetime(df['Order Date'])
df['Ship Date'] = pd.to_datetime(df['Ship Date'])
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
dtypes: datetime64[ns](2), float64(3), int64(3), object(13)
memory usage: 1.6+ MB
```

```
df.duplicated(subset=["Order ID","Product ID","Order Date","Ship
Date"]).sum()
```

```
8
```

```
df[df.duplicated(subset=["Order ID", "Order Date"], keep=False)]
```

```
{"type":"dataframe"}
```

```
df = df.drop_duplicates(subset=["Order ID","Product ID","Order
Date"],keep=False)
```

```
print(df.shape)
```

```
(9978, 21)
```

```
df['Lead Time'] = (df['Ship Date'] - df['Order Date']).dt.days
```

```
<ipython-input-217-dfc6d2bf1820>:1: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
```

```
df[['Order Date', 'Ship Date', 'Lead Time']].head()
```

```
{"summary":"{\n  \"name\": \"df[['Order Date', 'Ship Date', 'Lead
Time']]\",\n  \"rows\": 5,\n  \"fields\": [\n    {\n      \"column\":
\"Order Date\",\n      \"properties\": {\n        \"dtype\":
\"date\",\n        \"min\": \"2015-10-11 00:00:00\",\n        \"max\":
\"2016-11-08 00:00:00\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"2016-11-08 00:00:00\",\n          \"2016-
06-12 00:00:00\",\n          \"2015-10-11 00:00:00\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Ship Date\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2015-10-18 00:00:00\",\n        \"max\": \"2016-11-11 00:00:00\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"2016-
11-11 00:00:00\",\n          \"2016-06-16 00:00:00\",\n
\"2015-10-18 00:00:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Lead Time\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 2,\n        \"min\": 3,\n
\"max\": 7,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          3,\n          4,\n          7\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```

```
df["Order_year"]=df.loc[:,"Order Date"].dt.year
df['Order_year_month'] = df['Order Date'].dt.strftime('%Y-%m')
```

```
df.head()
```

```
{"type":"dataframe","variable_name":"df"}
```

```
df = df.drop(columns=["Product ID","Customer Name","Row ID","Postal
Code","Country","Ship Date"])
```

```
df.head()
```

```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 9978,\n  \"fields\":
[\n    {\n      \"column\": \"Order ID\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 5007,\n
\"samples\": [\n          \"CA-2015-153738\",\n          \"CA-2014-
103590\",\n          \"CA-2014-103807\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Order Date\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2014-01-03 00:00:00\",\n        \"max\": \"2017-12-30 00:00:00\",\n
\"num_unique_values\": 1237,\n        \"samples\": [\n
\"2017-03-28 00:00:00\",\n          \"2014-12-19 00:00:00\",\n
\"2016-03-25 00:00:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Ship Mode\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 4,\n
\"samples\": [\n          \"Standard Class\",\n          \"Same
Day\",\n          \"Second Class\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Customer ID\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 793,\n        \"samples\": [\n          \"DJ-
13510\",\n          \"MD-17350\",\n          \"NF-18475\"\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Segment\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n
\"Consumer\",\n          \"Corporate\",\n          \"Home Office\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"City\",\n      \"properties\":
```

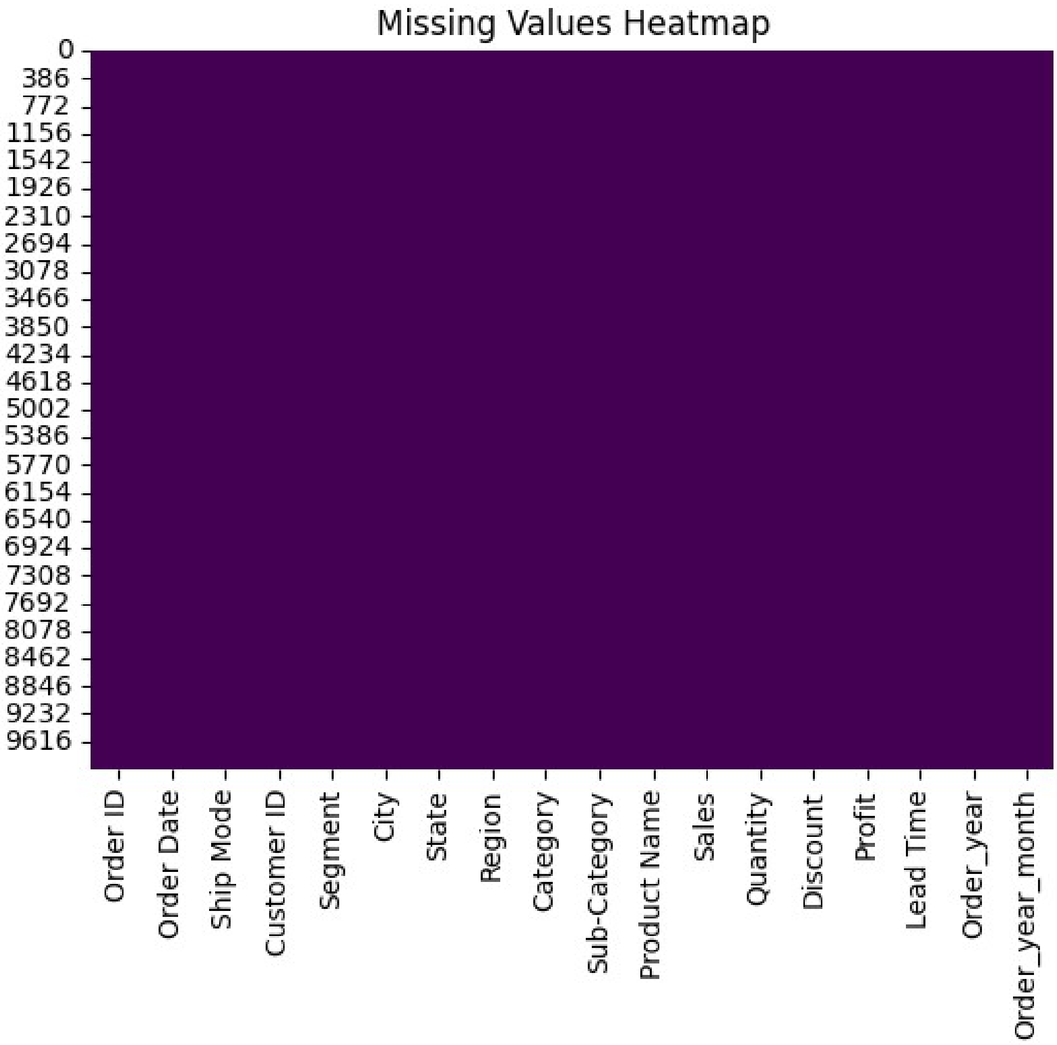
```
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
531,\n        \"samples\": [\n          \"Laurel\",\n
\"Madison\",\n          \"Hot Springs\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"State\",\n      \"properties\": {\
n        \"dtype\": \"category\",\n        \"num_unique_values\": 49,\
n        \"samples\": [\n          \"Delaware\",\n
\"Idaho\",\n          \"Wyoming\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Region\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
4,\n        \"samples\": [\n          \"West\",\n          \"East\",\n
\"South\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Category\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"Furniture\",\n          \"Office Supplies\",\n
\"Technology\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Sub-Category\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 17,\n
\"samples\": [\n          \"Bookcases\",\n          \"Chairs\",\n
\"Furnishings\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Product Name\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 1850,\n
\"samples\": [\n          \"Belkin 19\\\" Vented Equipment Shelf,
Black\",\n          \"Verbatim Slim CD and DVD Storage Cases,
50/Pack\",\n          \"Zebra Zazzle Fluorescent Highlighters\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Sales\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
623.5505543224153,\n        \"min\": 0.444,\n        \"max\":
22638.48,\n        \"num_unique_values\": 5818,\n        \"samples\":
[\n          89.98,\n          832.93,\n          192.8\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Quantity\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 2,\n
\"min\": 1,\n        \"max\": 14,\n        \"num_unique_values\": 14,\
n        \"samples\": [\n          14,\n          13,\n          2\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Discount\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.2065463296811589,\n        \"min\": 0.0,\n        \"max\": 0.8,\n
\"num_unique_values\": 12,\n        \"samples\": [\n          0.4,\n
0.1,\n          0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Profit\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 234.39247362956132,\n        \"min\": -6599.978,\n
\"max\": 8399.976,\n        \"num_unique_values\": 7277,\n
\"samples\": [\n          30.6054,\n          14.5728,\n
37.534\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Lead Time\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1,\n        \"min\": 0,\n
\"max\": 7,\n        \"num_unique_values\": 8,\n        \"samples\":
[\n          4,\n          6,\n          3\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Order_year\",\n
\"properties\": {\n        \"dtype\": \"int32\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n          2015,\n
2017,\n          2016\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Order_year_month\",\n      \"properties\": {\n        \"dtype\":
\"object\",\n        \"num_unique_values\": 48,\n        \"samples\":
[\n          \"2017-06\",\n          \"2015-06\",\n          \"2015-
01\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```

```
print(df.shape)
```

```
(9978, 18)
```

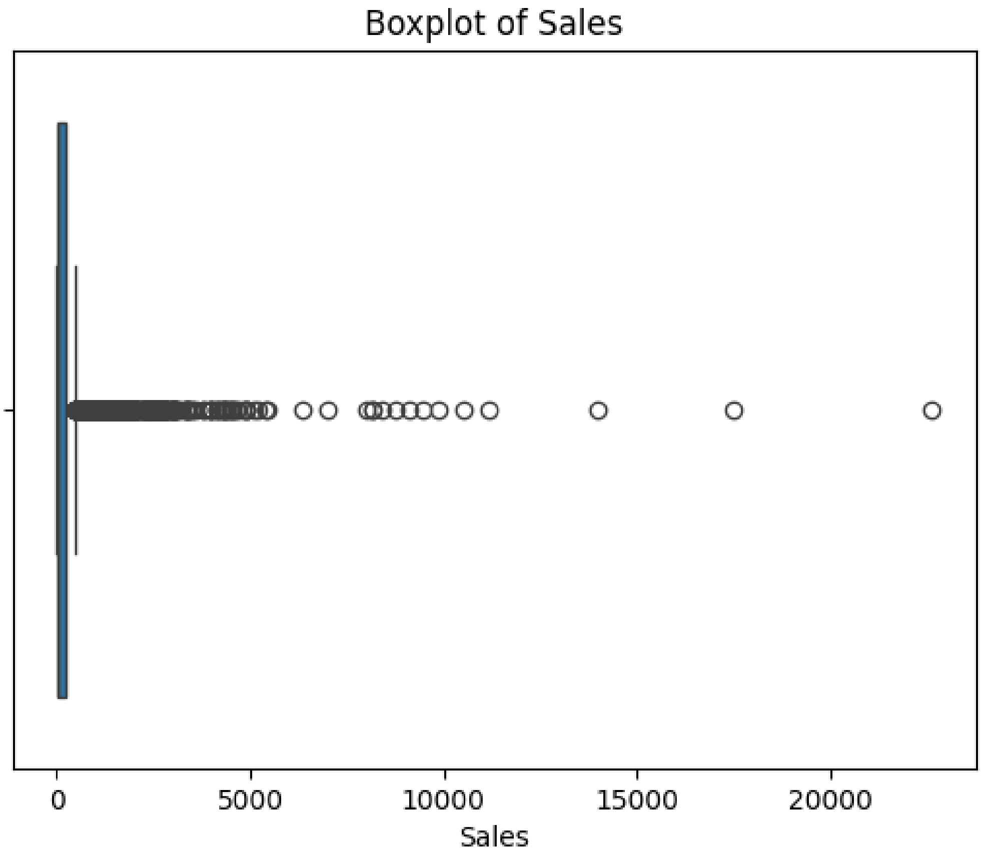
Data Vizualization

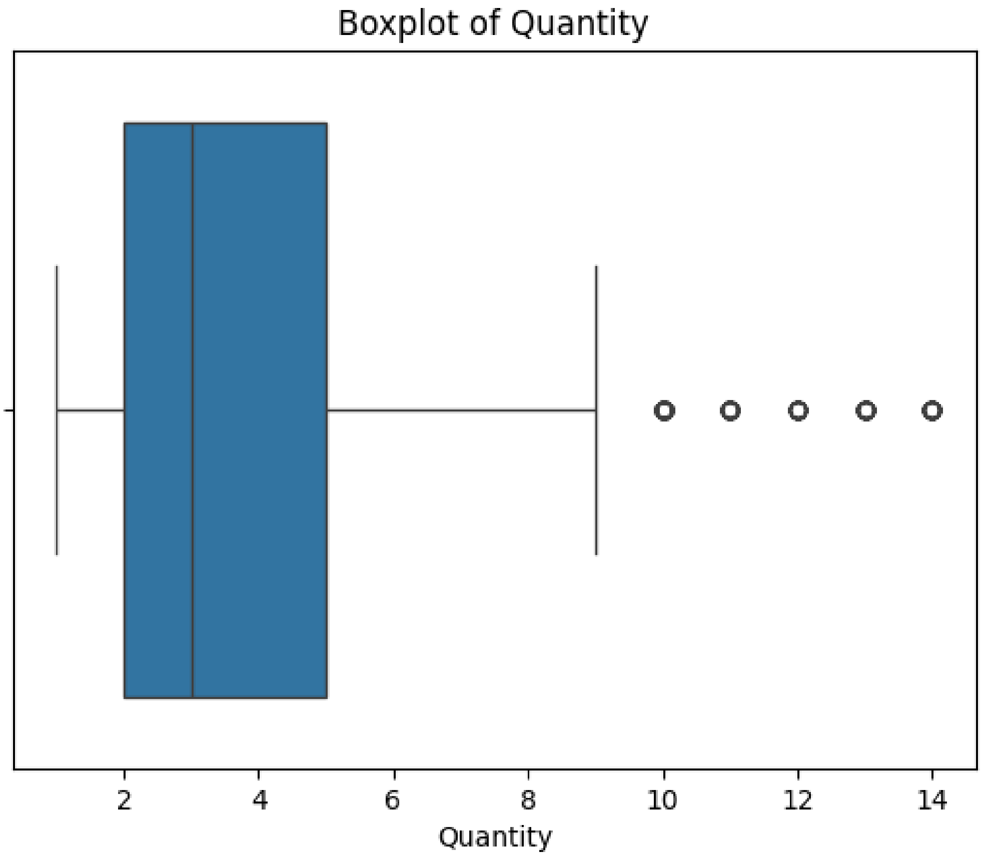
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title("Missing Values Heatmap")
plt.show()
```

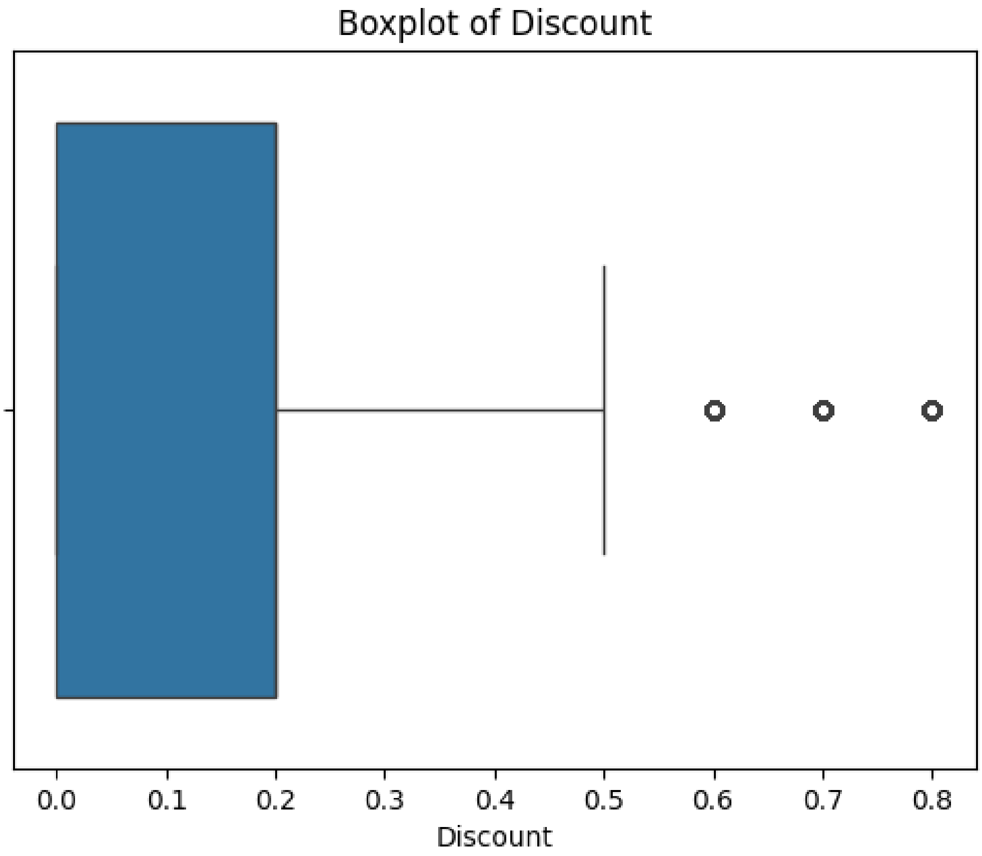


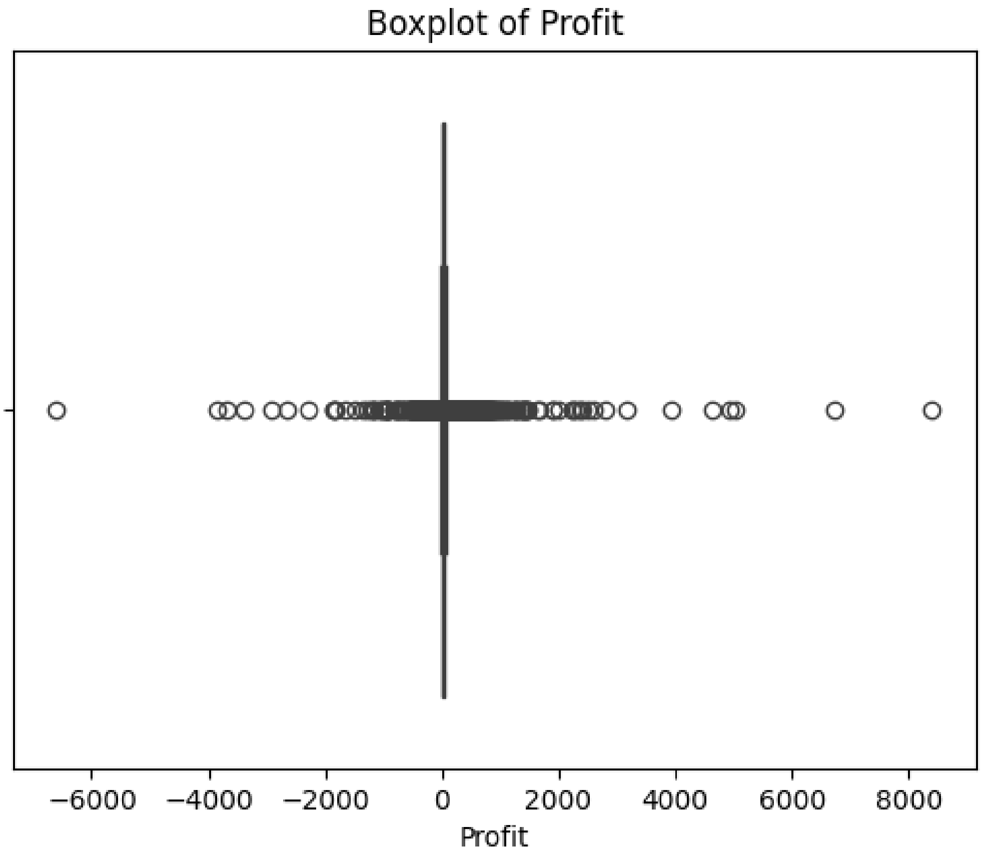
```
# Find numerical columns
numerical_columns =
df.select_dtypes(include=['number']).columns.tolist()

print("Numerical Columns:", numerical_columns)

# ... (rest of the code)

# Plot boxplots for numerical columns
for column in numerical_columns:
    sns.boxplot(x=df[column])
    plt.title(f"Boxplot of {column}")
    plt.show()
```

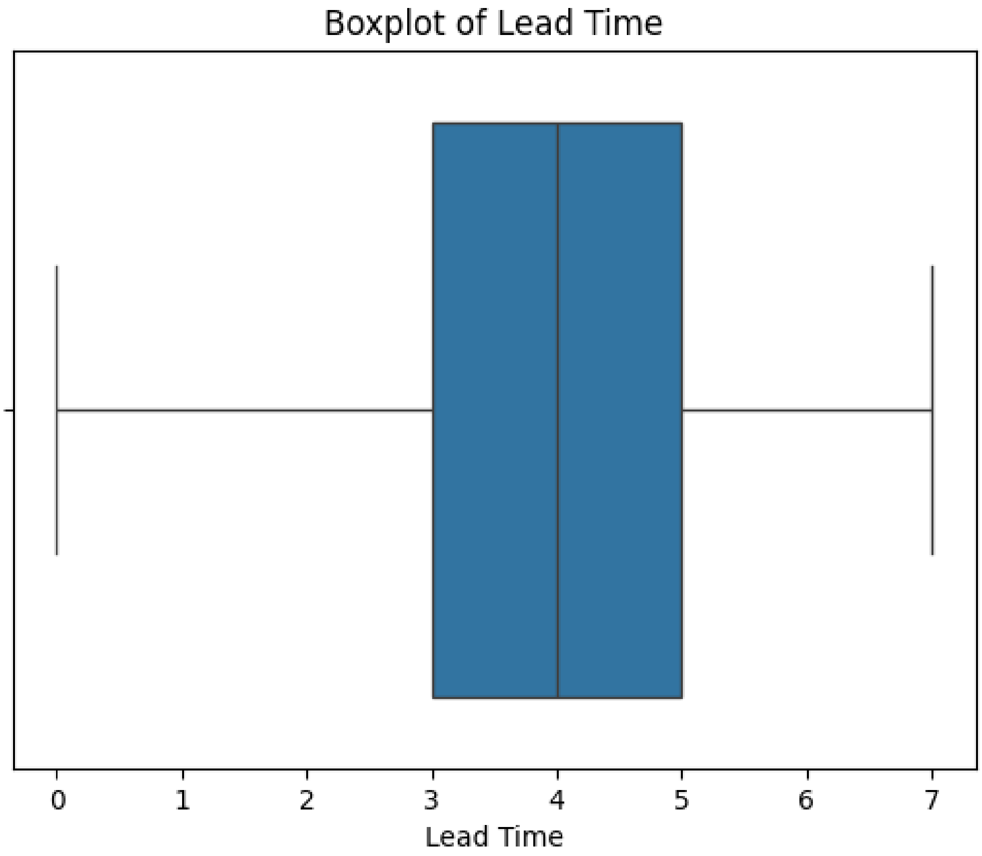
```
Numerical Columns: ['Sales', 'Quantity', 'Discount', 'Profit', 'Lead
Time', 'Order_year']
```

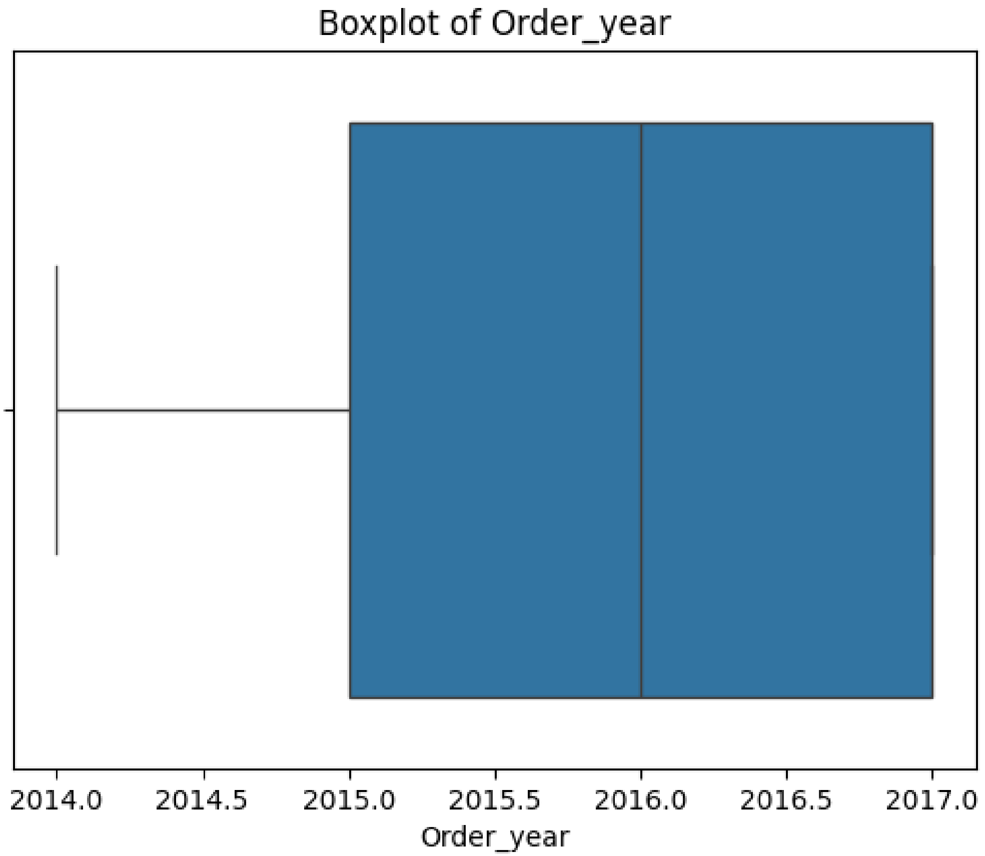

Boxplot of Sales
0
5000
10000
15000
20000
Sales


Boxplot of Quantity
2
4
6
8
10
12
14
Quantity


Boxplot of Discount
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Discount


Boxplot of Profit
−6000
−4000
−2000
0
2000
4000
6000
8000
Profit


Boxplot of Lead Time
0
1
2
3
4
5
6
7
Lead Time

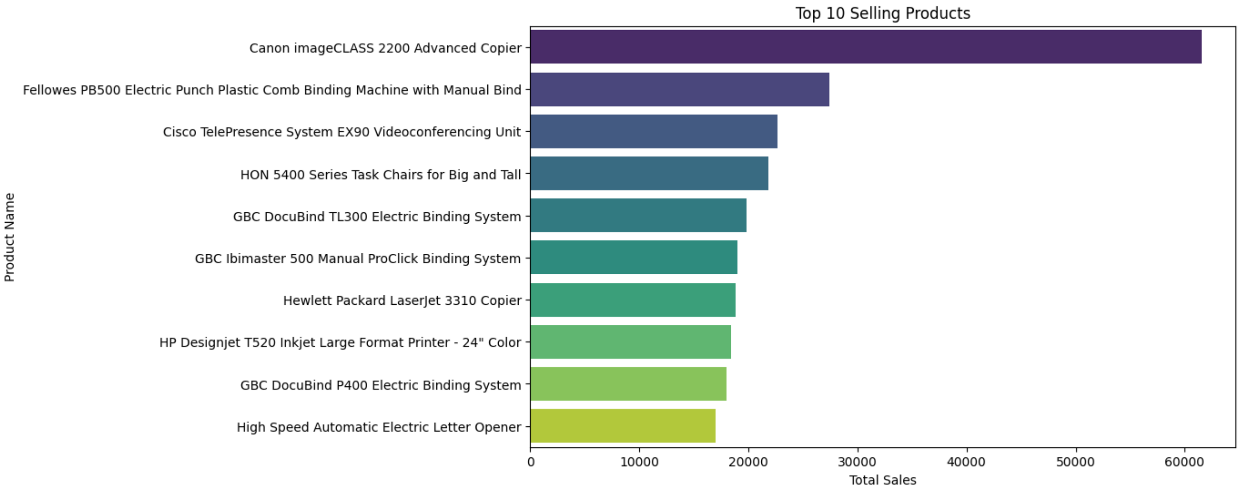


```
top_selling_products = df.groupby('Product Name')
['Sales'].sum().reset_index()

top_selling_products = top_selling_products.sort_values(by='Sales',
ascending=False)

top_selling_products.head(10)
```

```
{"summary":"{\n  \"name\": \"top_selling_products\",\n  \"rows\":
1850,\n  \"fields\": [\n    {\n      \"column\": \"Product Name\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 1850,\n        \"samples\": [\n
\"Bevis Round Conference Table Top, X-Base\",\n          \"Cush Cases
Heavy Duty Rugged Cover Case for Samsung Galaxy S5 - Purple\",\n
\"Ibico Plastic Spiral Binding Combs\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Sales\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 2792.010128129258,\n
\"min\": 1.624,\n        \"max\": 61599.824,\n
\"num_unique_values\": 1825,\n        \"samples\": [\n
881.192,\n          26.688000000000002,\n          2033.496\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"top_selling_products"}
```

```
top_10_products = top_selling_products.head(10)

# Plot the top 10 selling products
plt.figure(figsize=(10, 6))
sns.barplot(x='Sales', y='Product Name', data=top_10_products,
palette='viridis')
plt.title('Top 10 Selling Products')
plt.xlabel('Total Sales')
plt.ylabel('Product Name')
plt.show()
```

```
<ipython-input-229-80bf2dcc1207>:5: FutureWarning:



Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.
```

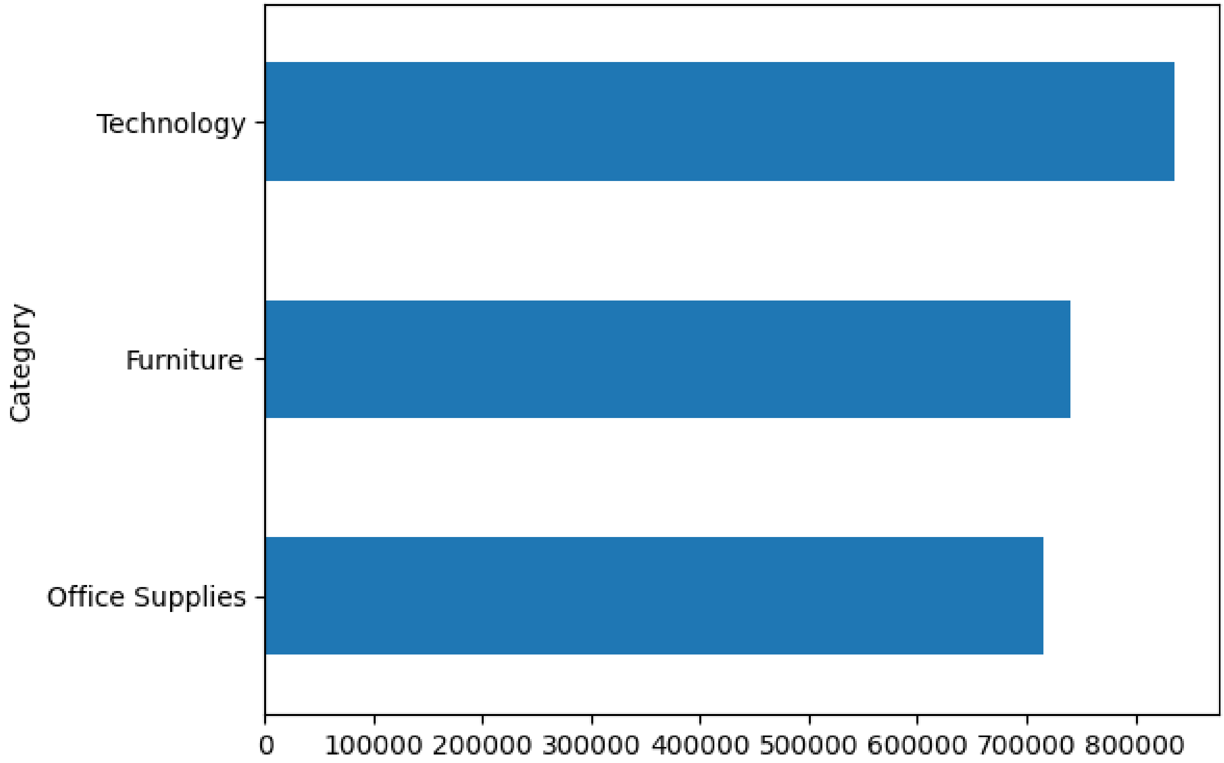


```
yearly_sales = df.groupby('Order_year')['Sales'].sum().reset_index()

yearly_sales
```

```
{"summary":"{\n  \"name\": \"yearly_sales\",\n  \"rows\": 4,\n
\"fields\": [\n    {\n      \"column\": \"Order_year\",\n
\"properties\": {\n        \"dtype\": \"int32\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n          2015,\n
2017,\n          2014\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Sales\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 121945.94041026258,\n        \"min\": 470307.359,\n
\"max\": 730858.6472,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          470307.359,\n          730858.6472,\n
483684.7541\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"yearly_sales"}
```

```
df.groupby('Category')
['Sales'].sum().sort_values(ascending=True).plot.barh()
```

```
<Axes: ylabel='Category'>
```

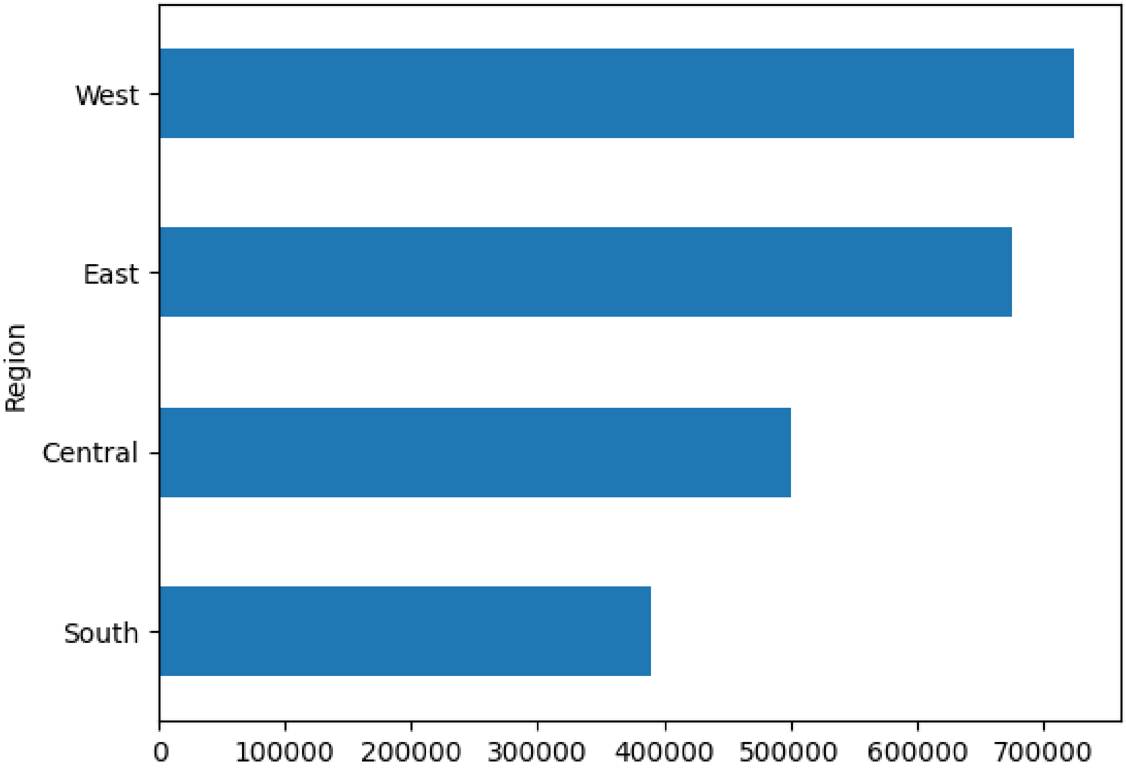


```
category_revenue_profit = df.groupby('Category').agg({'Sales': 'sum',
'Profit': 'sum'}).reset_index()

highest_revenue_profit =
category_revenue_profit.sort_values(by='Profit',
ascending=False).iloc[0]

print("Category generating the highest revenue and profit:")
print(highest_revenue_profit)
```

```
Category generating the highest revenue and profit:
Category     Technology
Sales        835274.241
```

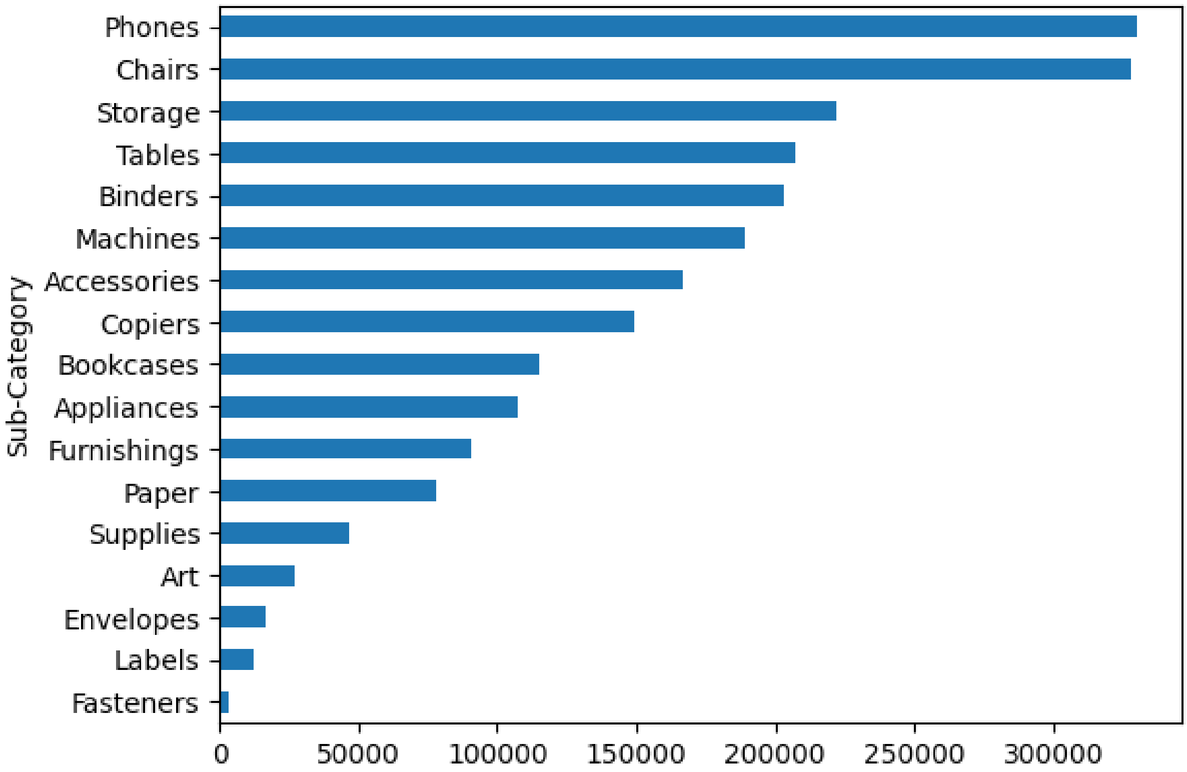
```
Profit        145298.9415
Name: 2, dtype: object
```

```
sales_by_region = df.groupby('Region')
['Sales'].sum().sort_values(ascending = True).plot.barh()
```

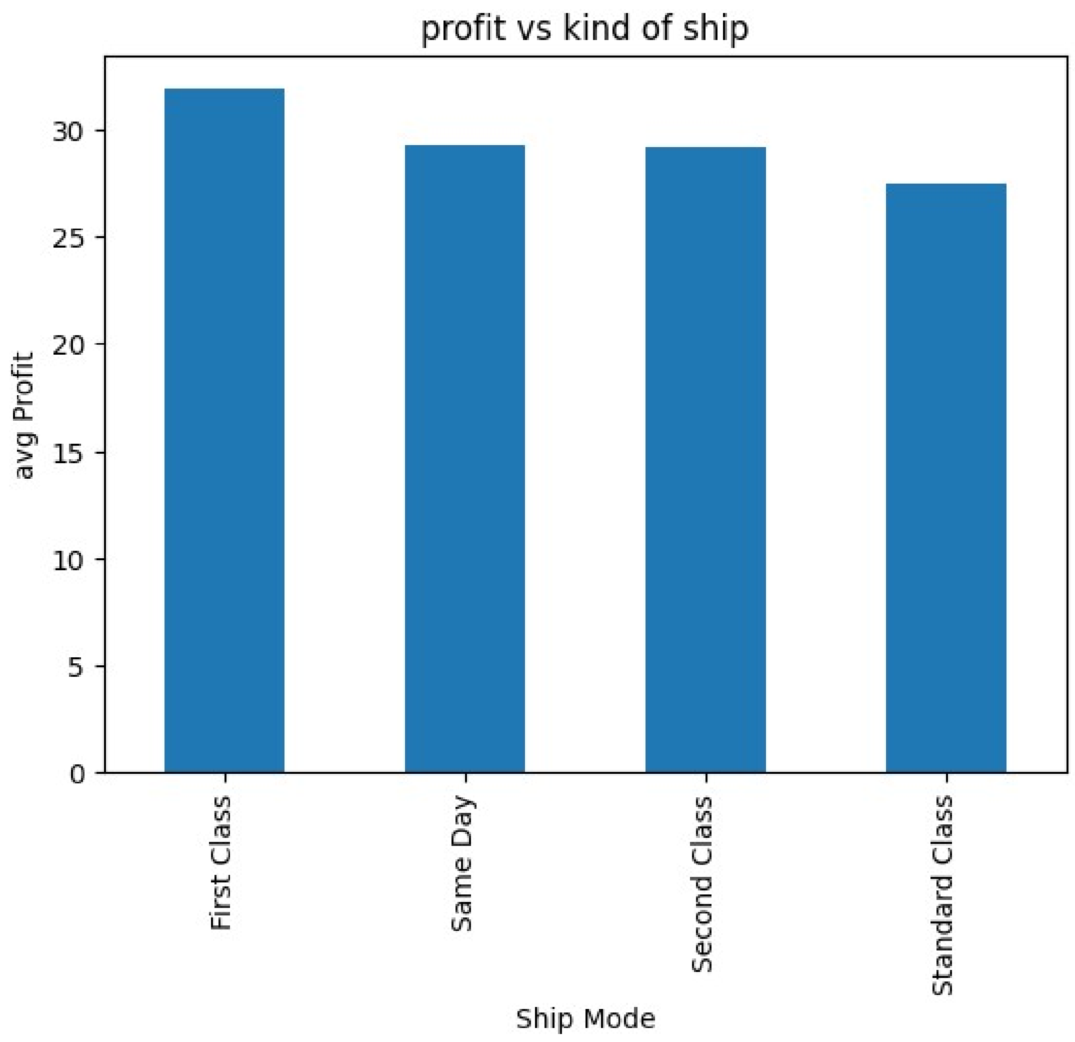


```
df.groupby('Sub-Category')['Sales'].sum().sort_values(ascending =
True).plot.barh()
```

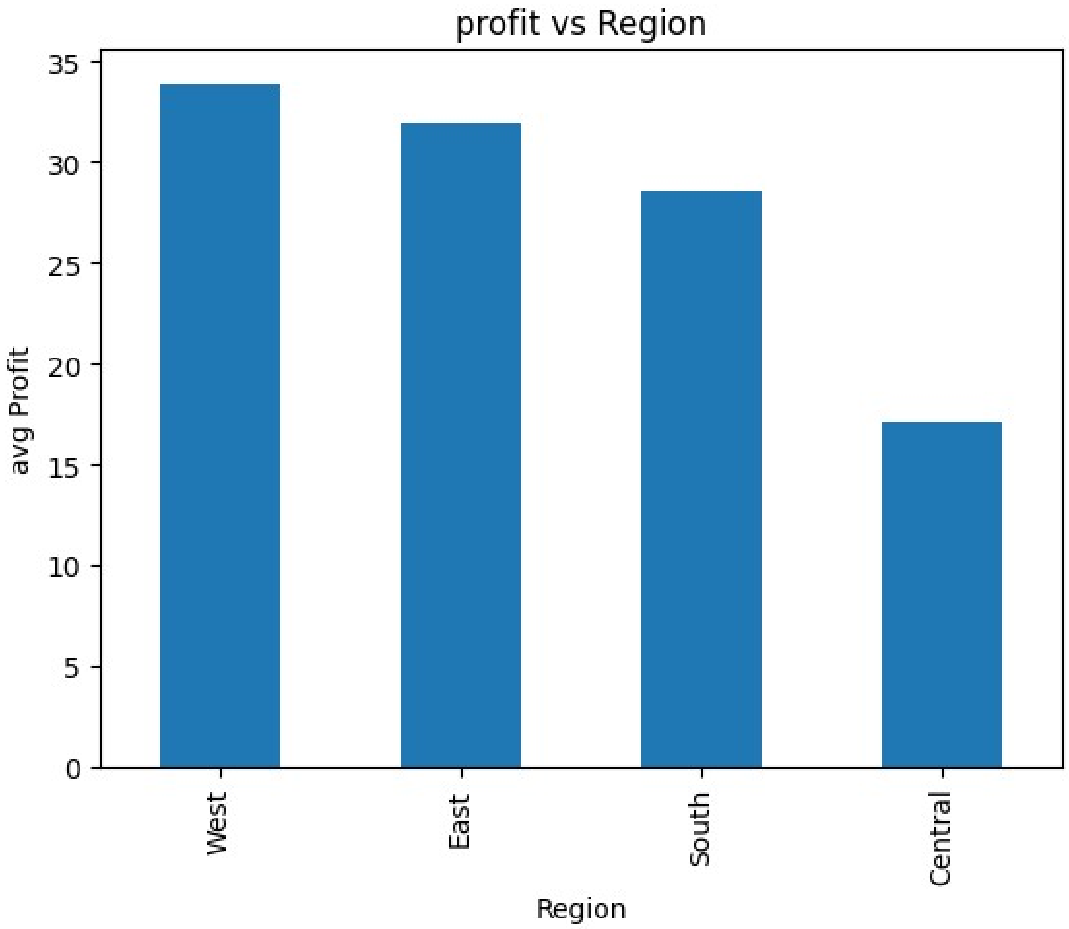
```
<Axes: ylabel='Sub-Category'>
```



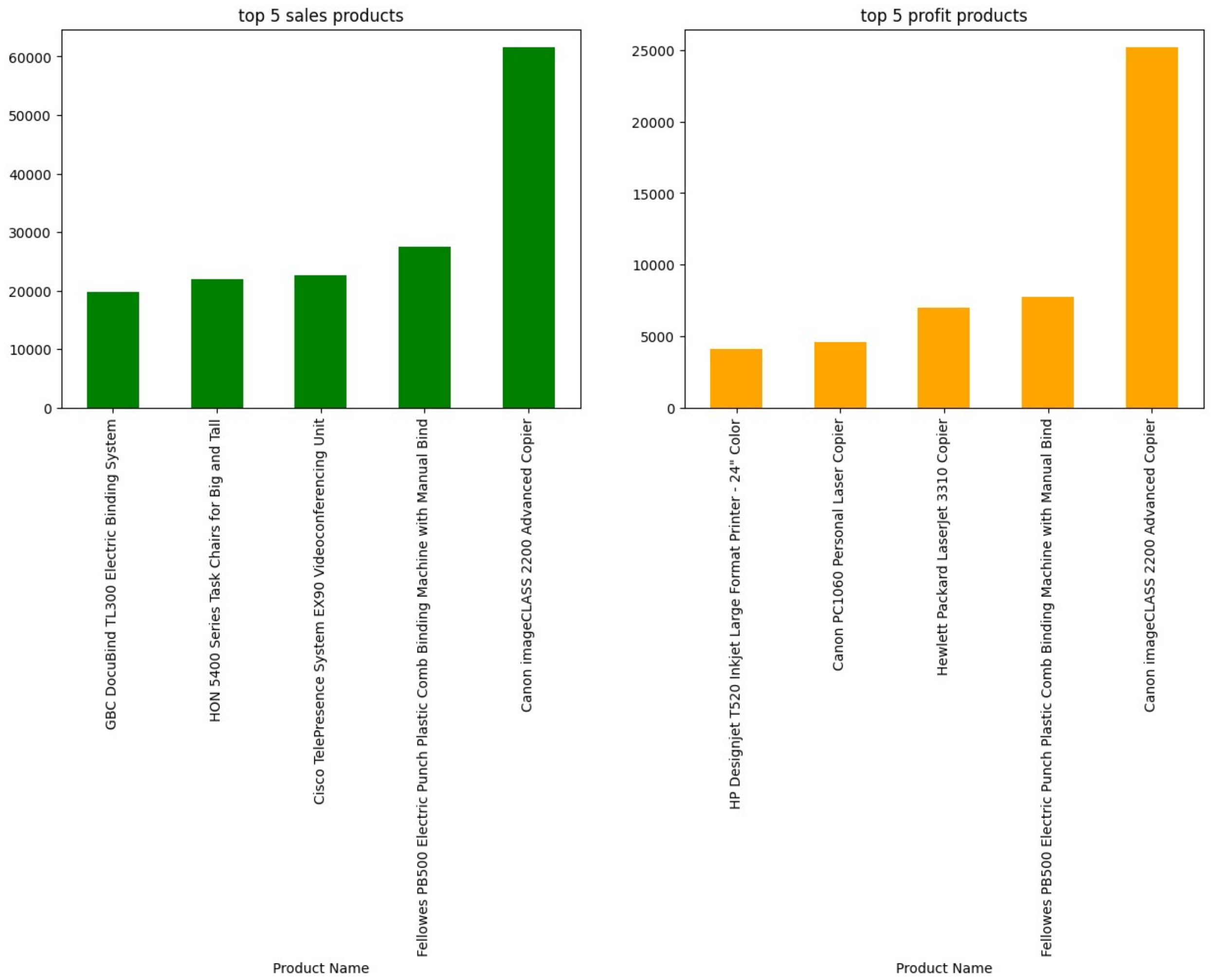
```
ship_type_profit=df[["Ship Mode","Profit"]]
ship_type_profit.groupby('Ship Mode')
['Profit'].mean().sort_values(ascending=False).plot(kind="bar",title="
profit vs kind of ship")
plt.xlabel("Ship Mode")
plt.ylabel("avg Profit");
```



```
df.groupby('Region')
['Profit'].mean().sort_values(ascending=False).plot(kind="bar",title="
profit vs Region")
plt.xlabel("Region")
plt.ylabel("avg Profit");
```

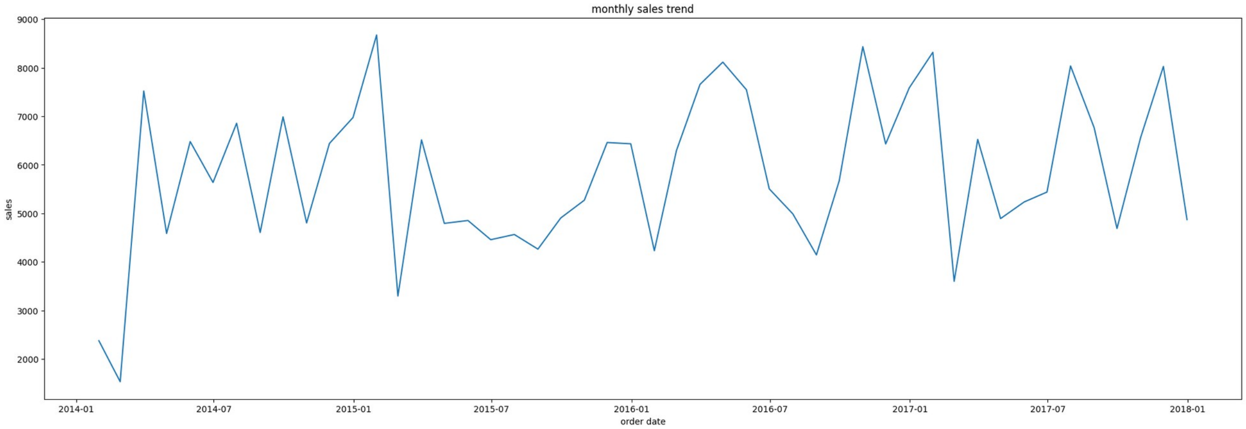


```
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(15,5))
product_info=df[['Product Name','Sales',"Profit"]]
sales_info=product_info.groupby('Product Name')
['Sales'].sum().sort_values().tail().plot(kind="bar",ax=ax1,color="green")
profit_info=product_info.groupby('Product Name')
['Profit'].sum().sort_values().tail().plot(kind="bar",ax=ax2,color="orange")
ax1.set_title("top 5 sales products")
ax2.set_title("top 5 profit products")
plt.show();
```



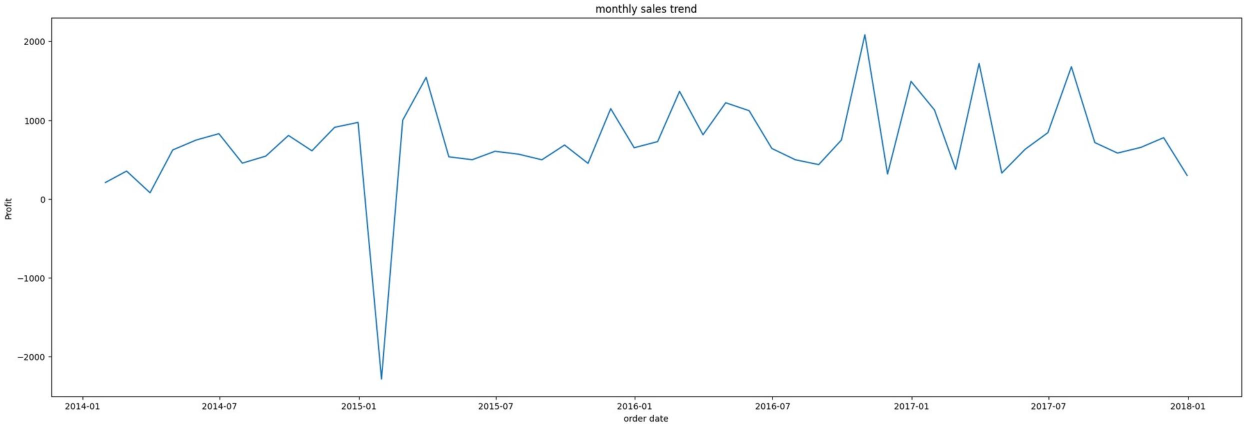
```
montly_sales=df.groupby("Order Date")
montly_sales=montly_sales['Sales'].mean()
montly_sales=montly_sales.resample("M").sum()
plt.figure(figsize=(25,8))
plt.plot(montly_sales)
plt.xlabel("order date")
plt.ylabel("sales")
plt.title("monthly sales trend")
plt.show();
```

```
<ipython-input-241-3fa6707e3674>:3: FutureWarning:

'M' is deprecated and will be removed in a future version, please use
'ME' instead.
```

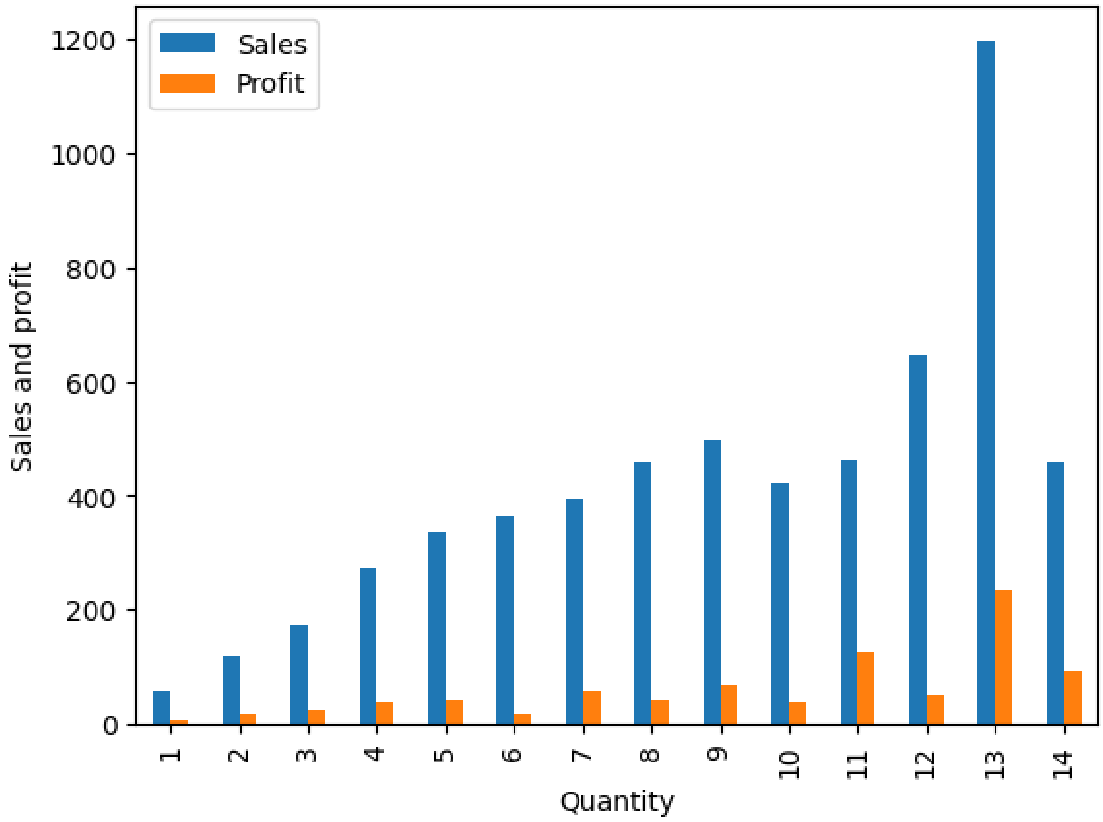


```
montly_sales=df.groupby("Order Date")
montly_sales=montly_sales['Profit'].mean()
montly_sales=montly_sales.resample("M").sum()
plt.figure(figsize=(25,8))
plt.plot(montly_sales)
plt.xlabel("order date")
plt.ylabel("Profit")
plt.title("monthly sales trend")
plt.show();
```

```
<ipython-input-242-463718e16ac2>:3: FutureWarning:

'M' is deprecated and will be removed in a future version, please use
'ME' instead.
```



```
discount_group=df.groupby(['Quantity'])
discount_group=discount_group[['Sales','Profit']].mean()
ax=discount_group.plot(kind="bar")
ax.set_ylabel("Sales and profit")
plt.show();
```



```python
import plotly.express as px
categories = df['Category'].value_counts().reset_index(name='Orders')
px.pie(data_frame=categories,values='Orders',names='Category',title='Number of Orders for each Category')
```

Feature Engineering

```python
# Assuming df is your original DataFrame
df["Profit_Margin"] = df["Profit"] / df["Sales"]
df["Discount_Impact"] = df["Discount"] * df["Sales"]
df["Log_Sales"] = np.log1p(df["Sales"])
df["Log_Profit"] = np.log1p(df["Profit"])
```

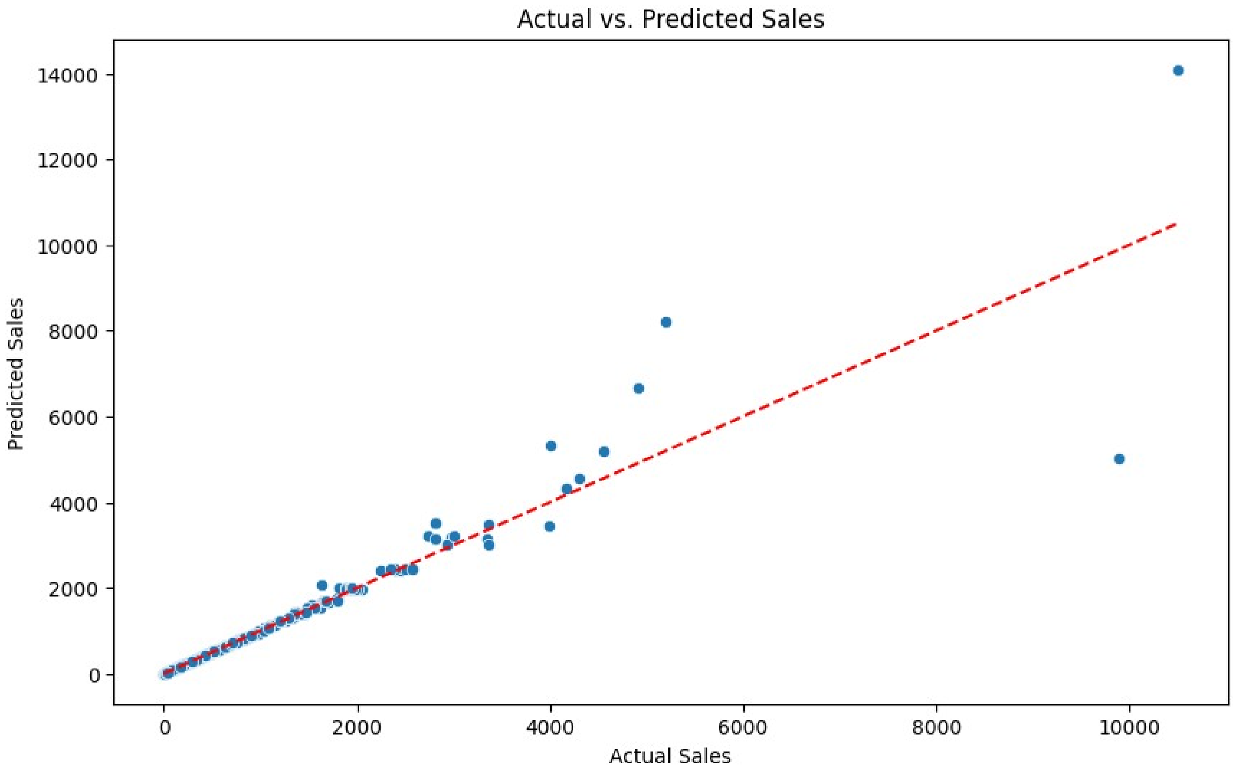
```
/usr/local/lib/python3.11/dist-packages/pandas/core/arraylike.py:399:
RuntimeWarning:

invalid value encountered in log1p
```

```python
df["Winsorized_Profit"] = winsorize(df["Profit"], limits=[0.05, 0.05])

# Encode categorical variables
df_encoded = pd.get_dummies(df, columns=["Category", "Segment", "Ship Mode"], drop_first=True)
```

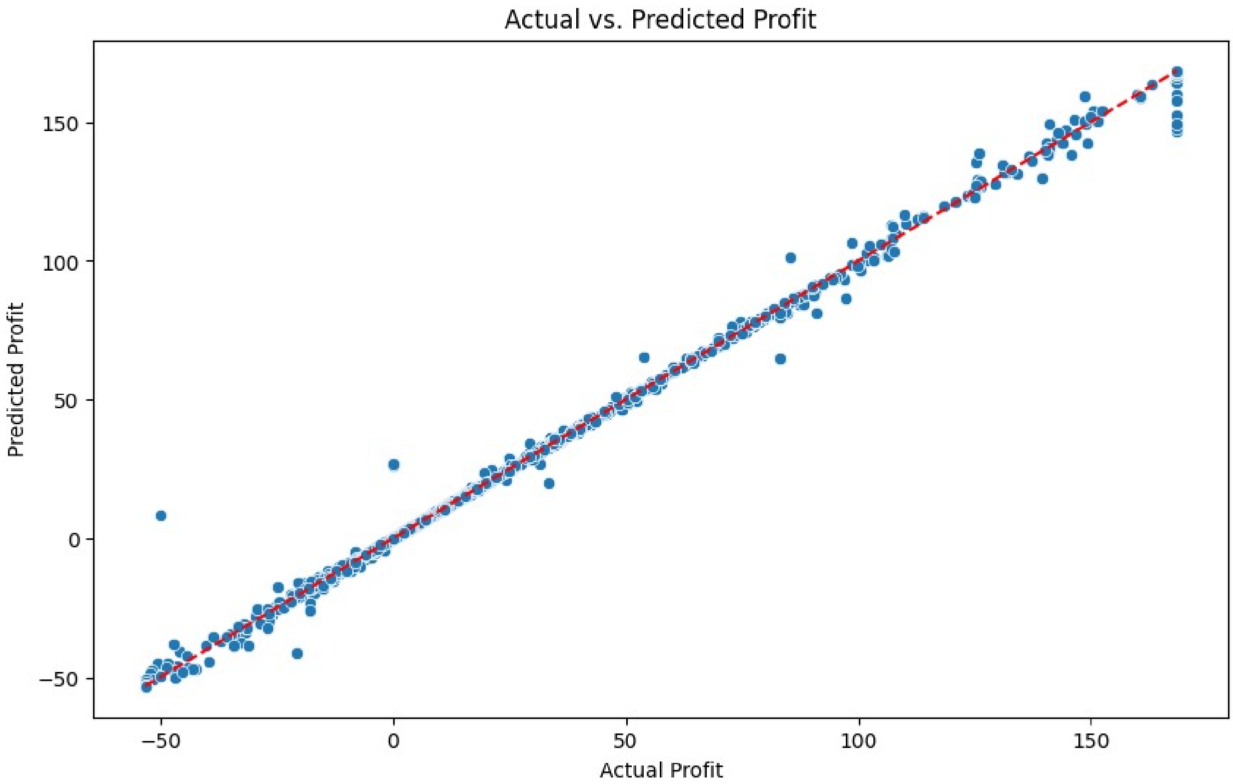
```python
# Define features (excluding Profit and Sales)
features = ["Quantity", "Discount", "Profit_Margin",
"Discount_Impact", "Log_Sales"] + list(df_encoded.columns[-6:])

# Define targets
target_profit = "Winsorized_Profit"
target_sales = "Sales"

# Split data into features and targets
X = df_encoded[features]
y_profit = df_encoded[target_profit]
y_sales = df_encoded[target_sales]

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score

# Split data into train and test sets
X_train, X_test, y_train_profit, y_test_profit = train_test_split(X,
y_profit, test_size=0.2, random_state=42)
X_train, X_test, y_train_sales, y_test_sales = train_test_split(X,
y_sales, test_size=0.2, random_state=42)

# Train Random Forest model for Profit
rf_profit = RandomForestRegressor(random_state=42)
rf_profit.fit(X_train, y_train_profit)

# Train XGBoost model for Sales
xgb_sales = XGBRegressor(n_estimators=100, learning_rate=0.1,
random_state=42)
xgb_sales.fit(X_train, y_train_sales)

# Evaluate Profit model
y_pred_profit = rf_profit.predict(X_test)
print("Profit Model Evaluation:")
print(f"MAE: {mean_absolute_error(y_test_profit, y_pred_profit)}")
print(f"MSE: {mean_squared_error(y_test_profit, y_pred_profit)}")
print(f"R²: {r2_score(y_test_profit, y_pred_profit)}")

# Evaluate Sales model
y_pred_sales = xgb_sales.predict(X_test)
print("\nSales Model Evaluation:")
print(f"MAE: {mean_absolute_error(y_test_sales, y_pred_sales)}")
print(f"MSE: {mean_squared_error(y_test_sales, y_pred_sales)}")
print(f"R²: {r2_score(y_test_sales, y_pred_sales)}")
```

```
Profit Model Evaluation:
MAE: 0.5859414699398866
```

```
MSE: 5.290069954997806
R²: 0.9977833966442337

Sales Model Evaluation:
MAE: 12.88756082030623
MSE: 26708.04037563255
R²: 0.9179105269209058

# Create a DataFrame for comparison
comparison_df = pd.DataFrame({
    "Actual_Sales": y_test_sales,
    "Predicted_Sales": y_pred_sales,
    "Actual_Profit": y_test_profit,
    "Predicted_Profit": y_pred_profit
})

# Display the first few rows of the comparison DataFrame
print(comparison_df.head())

      Actual_Sales  Predicted_Sales  Actual_Profit  Predicted_Profit
5763      1448.820      1404.369507       168.4704        160.094872
7635       300.980       302.737152        87.2842         87.411124
6403         8.010         8.224954         3.0438          3.179979
107         27.992        28.523520         2.0994          2.238129
3432        60.120        60.348942        28.8576         28.817681

import matplotlib.pyplot as plt
import seaborn as sns

# Plot Actual vs. Predicted Sales
plt.figure(figsize=(10, 6))
sns.scatterplot(x="Actual_Sales", y="Predicted_Sales",
data=comparison_df)
plt.plot([comparison_df["Actual_Sales"].min(),
comparison_df["Actual_Sales"].max()],
         [comparison_df["Actual_Sales"].min(),
comparison_df["Actual_Sales"].max()],
         color='red', linestyle='--')  # Diagonal line for perfect
predictions
plt.title("Actual vs. Predicted Sales")
plt.xlabel("Actual Sales")
plt.ylabel("Predicted Sales")
plt.show()
```



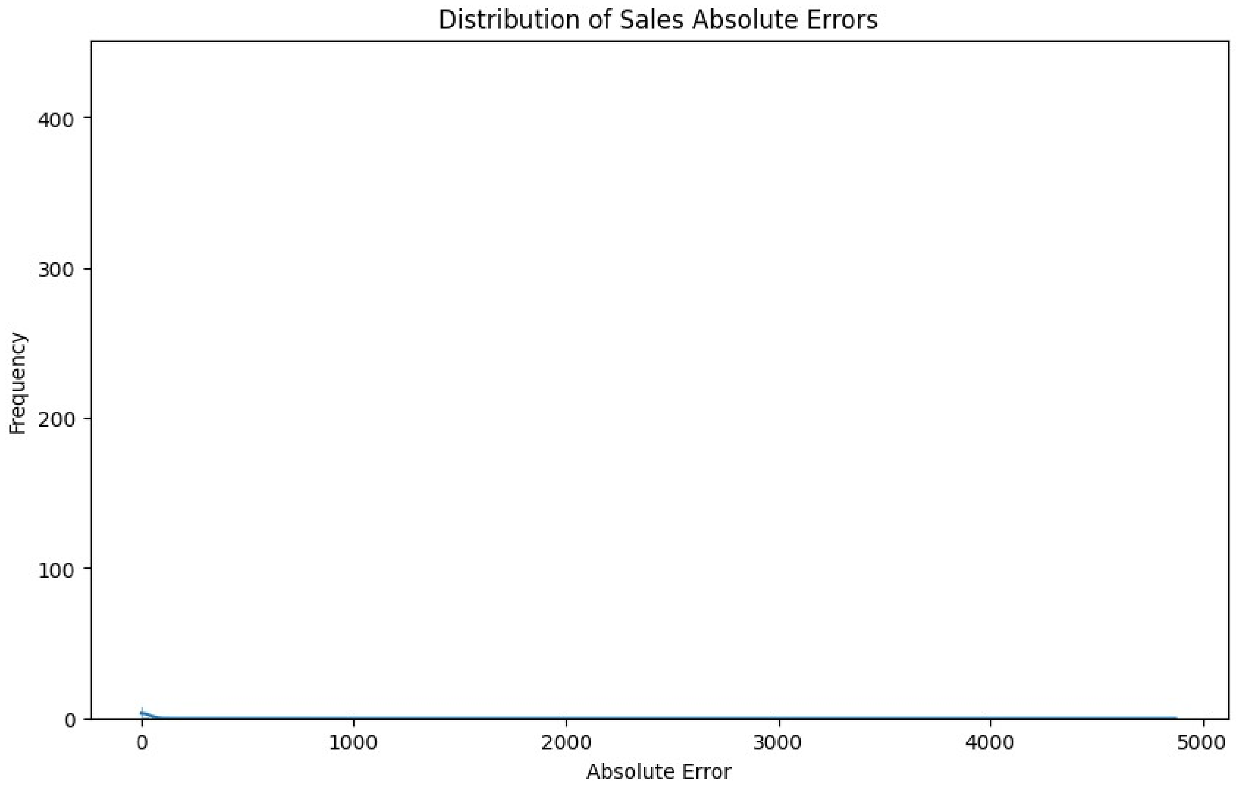
```
# Plot Actual vs. Predicted Profit
plt.figure(figsize=(10, 6))
sns.scatterplot(x="Actual_Profit", y="Predicted_Profit",
data=comparison_df)
plt.plot([comparison_df["Actual_Profit"].min(),
comparison_df["Actual_Profit"].max()],
         [comparison_df["Actual_Profit"].min(),
comparison_df["Actual_Profit"].max()],
         color='red', linestyle='--')  # Diagonal line for perfect
predictions
plt.title("Actual vs. Predicted Profit")
plt.xlabel("Actual Profit")
plt.ylabel("Predicted Profit")
plt.show()
```

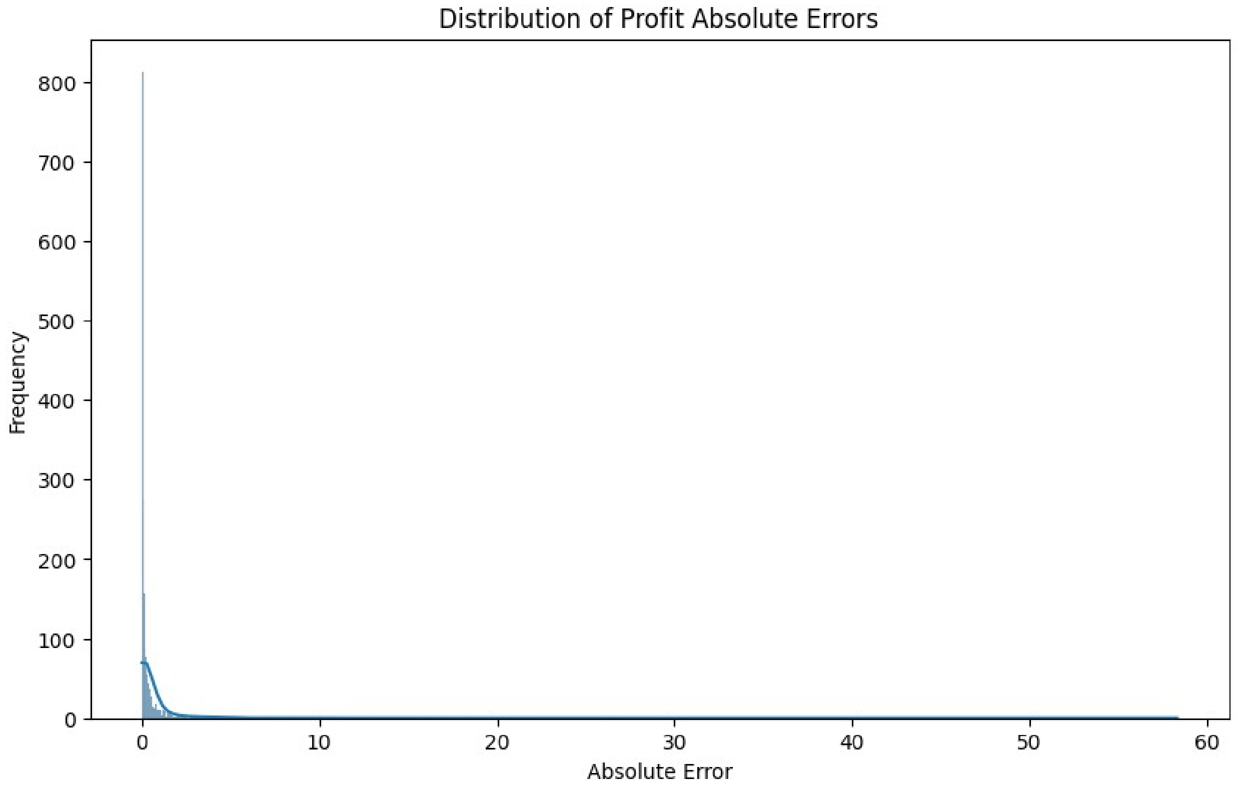


```
# Calculate absolute and percentage errors
comparison_df["Sales_Absolute_Error"] =
abs(comparison_df["Actual_Sales"] - comparison_df["Predicted_Sales"])
comparison_df["Sales_Percentage_Error"] =
(comparison_df["Sales_Absolute_Error"] /
comparison_df["Actual_Sales"]) * 100

comparison_df["Profit_Absolute_Error"] =
abs(comparison_df["Actual_Profit"] -
comparison_df["Predicted_Profit"])
comparison_df["Profit_Percentage_Error"] =
(comparison_df["Profit_Absolute_Error"] /
comparison_df["Actual_Profit"]) * 100

# Display the first few rows with errors
print(comparison_df.head())
```

```
      Actual_Sales  Predicted_Sales  Actual_Profit
Predicted_Profit  \
5763      1448.820      1404.369507       168.4704         160.094872

7635       300.980       302.737152        87.2842          87.411124

6403         8.010         8.224954         3.0438           3.179979

107         27.992        28.523520         2.0994           2.238129
```

```
3432        60.120        60.348942        28.8576         28.817681


      Sales_Absolute_Error  Sales_Percentage_Error
Profit_Absolute_Error  \
5763             44.450493                3.068048
8.375528
7635              1.757152                0.583810
0.126924
6403              0.214954                2.683566
0.136179
107               0.531520                1.898827
0.138729
3432              0.228942                0.380808
0.039919

      Profit_Percentage_Error
5763                 4.971513
7635                 0.145415
6403                 4.473980
107                  6.608031
3432                 0.138331

# Plot distribution of Sales Absolute Errors
plt.figure(figsize=(10, 6))
sns.histplot(comparison_df["Sales_Absolute_Error"], kde=True)
plt.title("Distribution of Sales Absolute Errors")
plt.xlabel("Absolute Error")
plt.ylabel("Frequency")
plt.show()

# Plot distribution of Profit Absolute Errors
plt.figure(figsize=(10, 6))
sns.histplot(comparison_df["Profit_Absolute_Error"], kde=True)
plt.title("Distribution of Profit Absolute Errors")
plt.xlabel("Absolute Error")
plt.ylabel("Frequency")
plt.show()
```


Distribution of Sales Absolute Errors
Frequency
0
100
200
300
400
0
1000
2000
3000
4000
5000
Absolute Error


Distribution of Profit Absolute Errors
Frequency
0
100
200
300
400
500
600
700
800
0
10
20
30
40
50
60
Absolute Error

```python
import pandas as pd

# Function to collect user input
def get_user_input():
    print("Please enter the following details:")
    quantity = float(input("Quantity: "))
    discount = float(input("Discount (e.g., 0.1 for 10%): "))
    category = input("Category (e.g., Furniture, Office Supplies, Technology): ")
    segment = input("Segment (e.g., Consumer, Corporate, Home Office): ")
    ship_mode = input("Ship Mode (e.g., Standard Class, Second Class, First Class, Same Day): ")

    # Return input as a dictionary
    return {
        "Quantity": quantity,
        "Discount": discount,
        "Category": category,
        "Segment": segment,
        "Ship Mode": ship_mode
    }

# Get user input
input_data = get_user_input()

# Convert to DataFrame
input_df = pd.DataFrame([input_data])

# Apply feature engineering (same as training data)
input_df["Profit_Margin"] = 0  # Placeholder, since Profit is not available
input_df["Discount_Impact"] = input_df["Discount"] * 0  # Placeholder, since Sales is not available
input_df["Log_Sales"] = 0  # Placeholder, since Sales is not available

# Encode categorical variables
input_df_encoded = pd.get_dummies(input_df, columns=["Category", "Segment", "Ship Mode"], drop_first=True)

# Ensure all columns are present (in case some categories were not in the input)
missing_cols = set(features) - set(input_df_encoded.columns)
for col in missing_cols:
    input_df_encoded[col] = 0

# Reorder columns to match the training data
input_df_encoded = input_df_encoded[features]

# Predict Profit and Sales
```

```
predicted_profit = rf_profit.predict(input_df_encoded)
predicted_sales = xgb_sales.predict(input_df_encoded)

print(f"Predicted Profit: {predicted_profit[0]}")
print(f"Predicted Sales: {predicted_sales[0]}")
```

```
Please enter the following details:
Quantity: 4
Discount (e.g., 0.1 for 10%): 0.2
Category (e.g., Furniture, Office Supplies, Technology): Furniture
Segment (e.g., Consumer, Corporate, Home Office): Consumer
Ship Mode (e.g., Standard Class, Second Class, First Class, Same Day):
First Class
Predicted Profit: -0.043880999999999996
Predicted Sales: 2.575498580932617
```

```
import joblib

# Save the Random Forest model for profit prediction
joblib.dump(rf_profit, "rf_profit_model.pkl")

# Save the XGBoost model for sales prediction
joblib.dump(xgb_sales, "xgb_sales_model.pkl")

print("Models saved successfully!")
```

```
Models saved successfully!
```