```python
import pandas as pd

# Load the dataset
file_path = r"C:\Users\engah\Desktop\Data Analyst project\superstore python\SampleSuperstore.csv"

# Read the CSV file
df = pd.read_csv(file_path)

# Display the first few rows
print(df.head())
```

```
        Ship Mode    Segment        Country             City
State  \
0    Second Class   Consumer  United States        Henderson
Kentucky
1    Second Class   Consumer  United States        Henderson
Kentucky
2    Second Class  Corporate  United States      Los Angeles
California
3  Standard Class   Consumer  United States  Fort Lauderdale
Florida
4  Standard Class   Consumer  United States  Fort Lauderdale
Florida

   Postal Code Region         Category Sub-Category     Sales
Quantity  \
0        42420  South        Furniture    Bookcases  261.9600
2
1        42420  South        Furniture       Chairs  731.9400
3
2        90036   West  Office Supplies       Labels   14.6200
2
3        33311  South        Furniture       Tables  957.5775
5
4        33311  South  Office Supplies      Storage   22.3680
2

   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
```

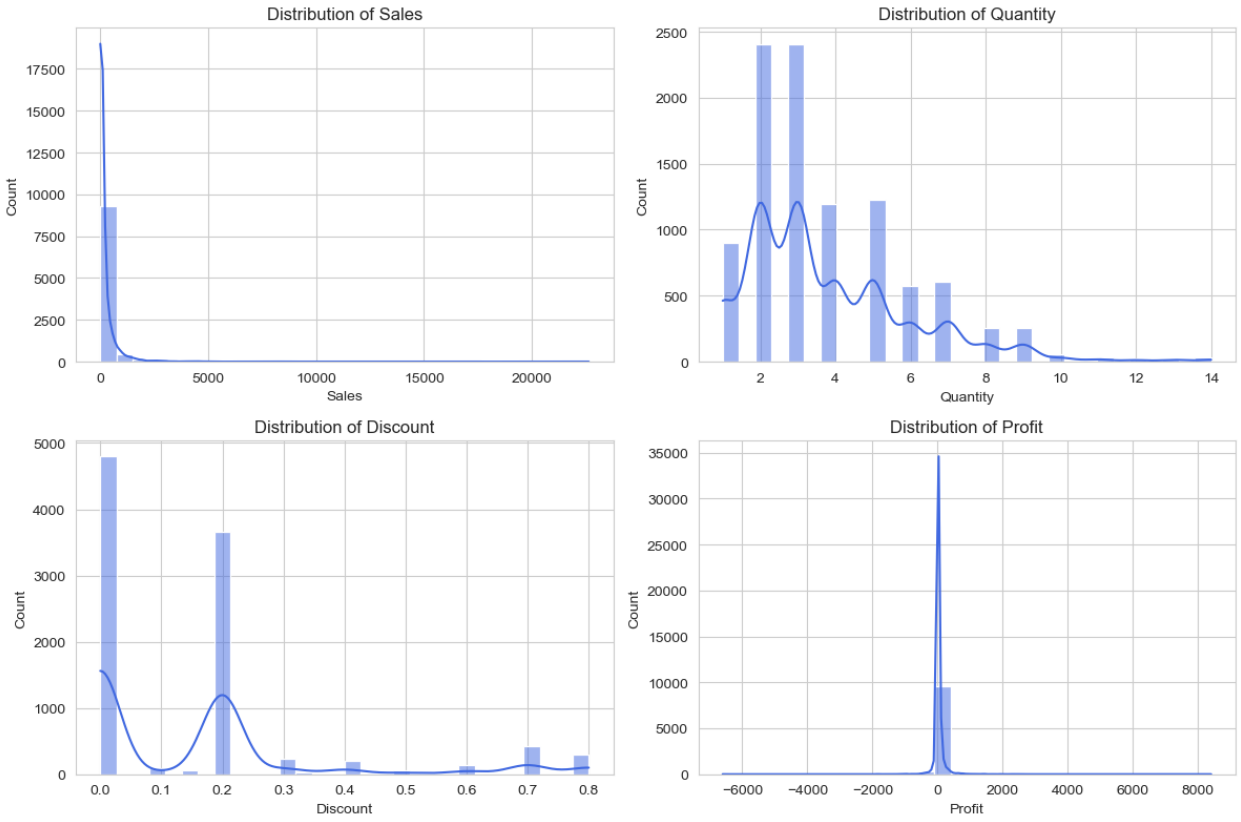
```python
# Check dataset structure
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 13 columns):
```

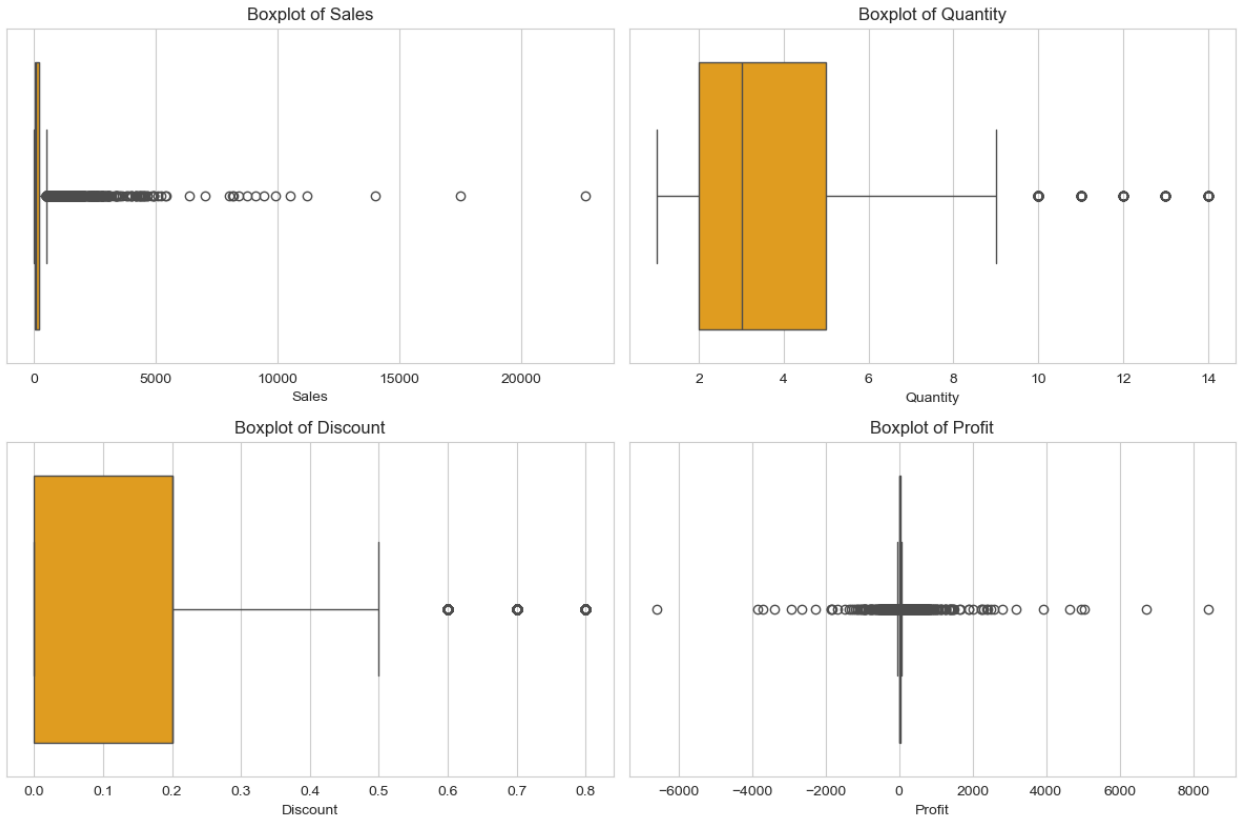
```
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Ship Mode     9994 non-null   object
 1   Segment       9994 non-null   object
 2   Country       9994 non-null   object
 3   City          9994 non-null   object
 4   State         9994 non-null   object
 5   Postal Code   9994 non-null   int64
 6   Region        9994 non-null   object
 7   Category      9994 non-null   object
 8   Sub-Category  9994 non-null   object
 9   Sales         9994 non-null   float64
 10  Quantity      9994 non-null   int64
 11  Discount      9994 non-null   float64
 12  Profit        9994 non-null   float64
dtypes: float64(3), int64(2), object(8)
memory usage: 1015.1+ KB
None

# Check for missing values
print(df.isnull().sum())

Ship Mode       0
Segment         0
Country         0
City            0
State           0
Postal Code     0
Region          0
Category        0
Sub-Category    0
Sales           0
Quantity        0
Discount        0
Profit          0
dtype: int64

# Show summary statistics for numerical columns
print(df.describe())

        Postal Code         Sales     Quantity     Discount
Profit
count   9994.000000   9994.000000  9994.000000  9994.000000
9994.000000
mean   55190.379428    229.858001     3.789574     0.156203
28.656896
std    32063.693350    623.245101     2.225110     0.206452
234.260108
min     1040.000000      0.444000     1.000000     0.000000 -
6599.978000
```

```
25%    23223.000000     17.280000     2.000000     0.000000
1.728750
50%    56430.500000     54.490000     3.000000     0.200000
8.666500
75%    90008.000000    209.940000     5.000000     0.200000
29.364000
max    99301.000000  22638.480000    14.000000     0.800000
8399.976000
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#Code to Visualize Distributions

import matplotlib.pyplot as plt
import seaborn as sns

# List of numerical columns to analyze
num_cols = ['Sales', 'Quantity', 'Discount', 'Profit']

# Set plot style
sns.set_style("whitegrid")

# Create subplots
fig, axes = plt.subplots(2, 2, figsize=(12, 8))

# Flatten axes array for easy iteration
axes = axes.flatten()

# Plot histograms for each numerical column
for i, col in enumerate(num_cols):
    sns.histplot(df[col], bins=30, kde=True, ax=axes[i],
color='royalblue')
    axes[i].set_title(f'Distribution of {col}')
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Count')

# Adjust layout and show plot
plt.tight_layout()
plt.show()
```

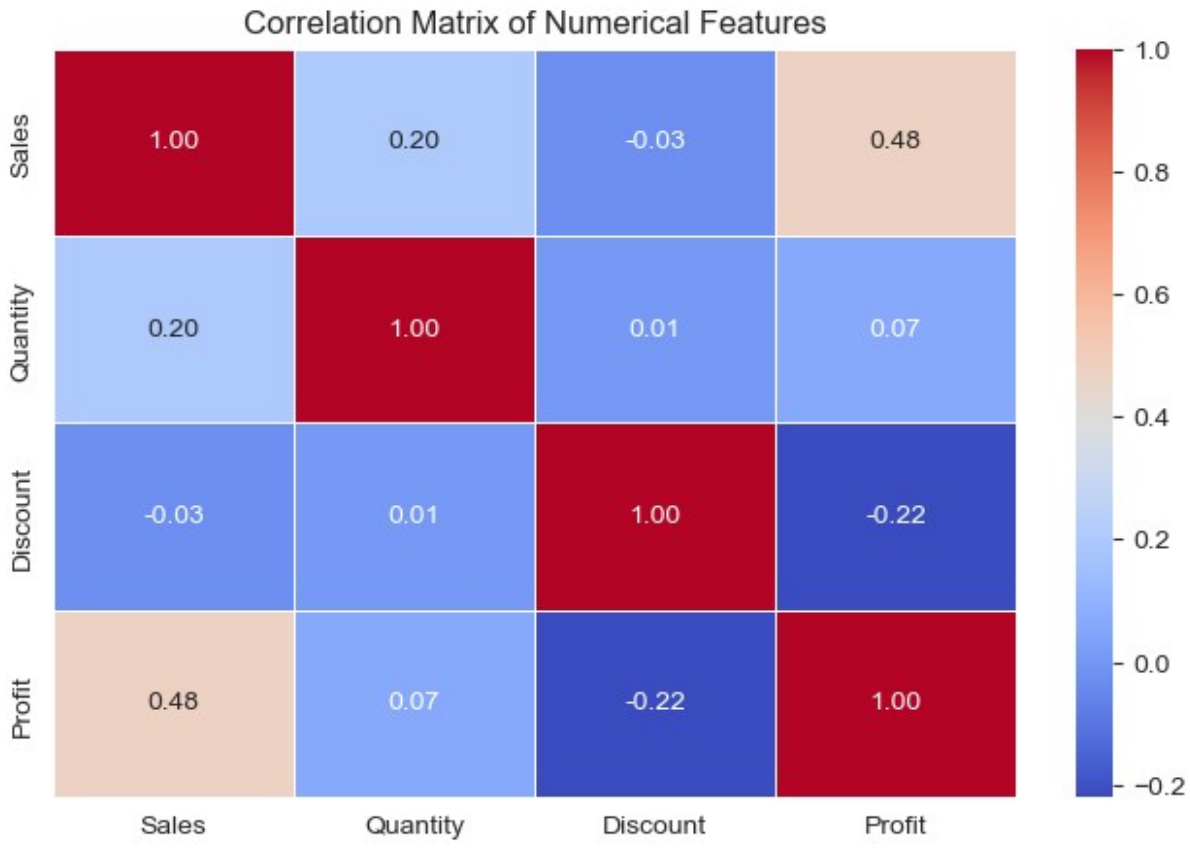



```
#Code to Visualize Boxplots (Detecting Outliers)

# Create subplots
fig, axes = plt.subplots(2, 2, figsize=(12, 8))

# Flatten axes array for easy iteration
axes = axes.flatten()

# Plot boxplots for each numerical column
for i, col in enumerate(num_cols):
    sns.boxplot(x=df[col], ax=axes[i], color='orange')
    axes[i].set_title(f'Boxplot of {col}')

# Adjust layout and show plot
plt.tight_layout()
plt.show()
```




```python
#Code for Correlation Analysis

import seaborn as sns
import matplotlib.pyplot as plt

# Compute the correlation matrix
correlation_matrix = df[['Sales', 'Quantity', 'Discount',
'Profit']].corr()

# Display the correlation matrix
print(correlation_matrix)

# Plot the correlation heatmap
plt.figure(figsize=(8, 5))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f", linewidths=0.5)
plt.title("Correlation Matrix of Numerical Features")
plt.show()
```

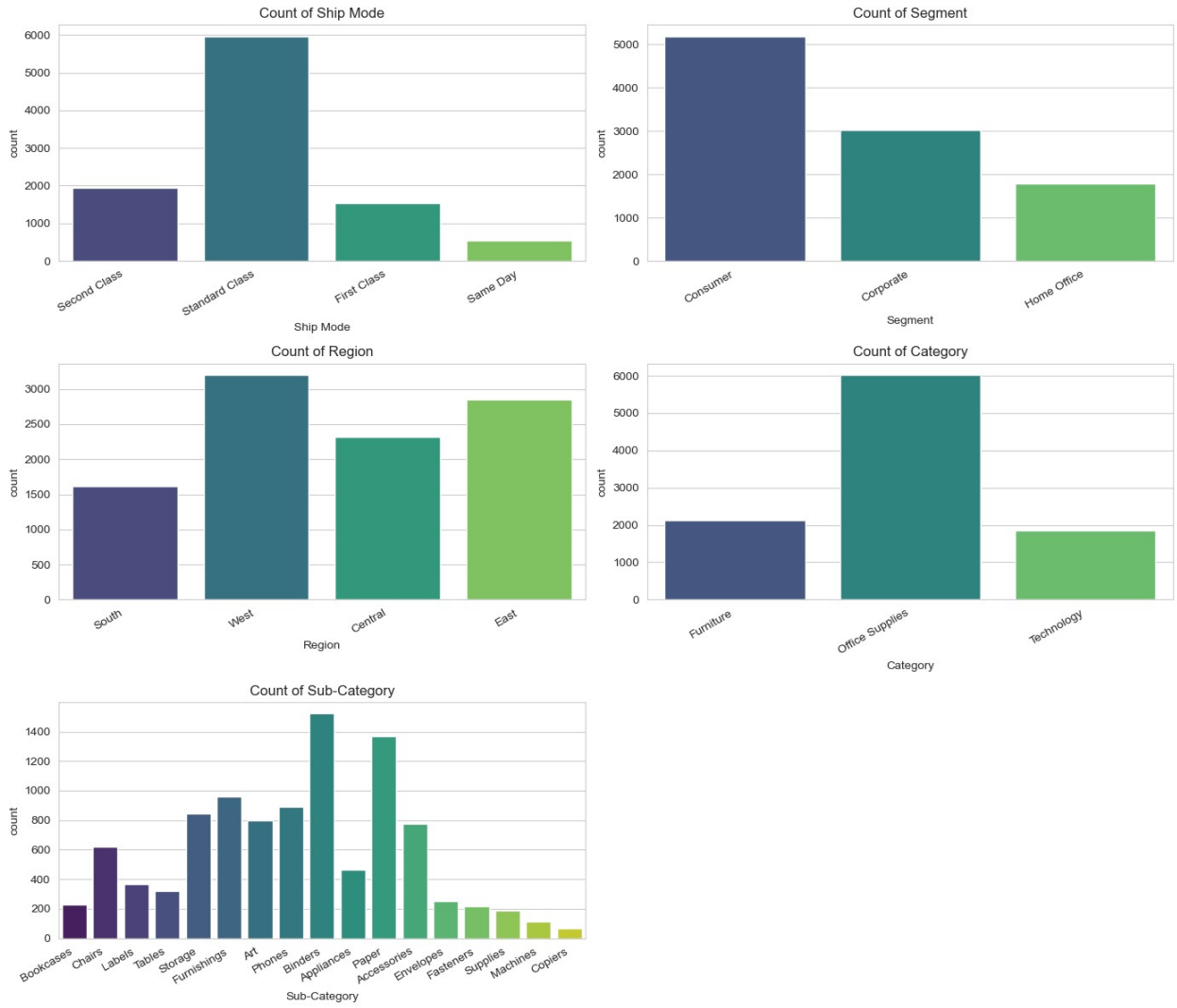
```
              Sales  Quantity  Discount    Profit
Sales      1.000000  0.200795 -0.028190  0.479064
Quantity   0.200795  1.000000  0.008623  0.066253
Discount -0.028190  0.008623  1.000000 -0.219487
Profit     0.479064  0.066253 -0.219487  1.000000
```




```
#Categorical Analysis

# Count Plots for Categorical Variables

import seaborn as sns
import matplotlib.pyplot as plt

# List of categorical columns to analyze
cat_cols = ['Ship Mode', 'Segment', 'Region', 'Category', 'Sub-
Category']

# Set plot style
sns.set_style("whitegrid")

# Create subplots
fig, axes = plt.subplots(3, 2, figsize=(14, 12))
axes = axes.flatten()  # Flatten the array for easy looping

# Loop through categorical columns and create count plots
for i, col in enumerate(cat_cols):
    ax = axes[i]
    sns.countplot(data=df, x=col, hue=col, palette='viridis', ax=ax,
legend=False)  # Fix FutureWarning
    ax.set_title(f'Count of {col}')
```

```
    ax.set_xticklabels(ax.get_xticklabels(), rotation=30, ha='right')

# Remove empty subplot if needed
if len(cat_cols) < len(axes):
    fig.delaxes(axes[-1])

# Adjust layout and show plot
plt.tight_layout()
plt.show()
```

```
C:\Users\engah\AppData\Local\Temp\ipykernel_6416\3286844220.py:19:
UserWarning: set_ticklabels() should only be used with a fixed number
of ticks, i.e. after set_ticks() or using a FixedLocator.
  ax.set_xticklabels(ax.get_xticklabels(), rotation=30, ha='right')
C:\Users\engah\AppData\Local\Temp\ipykernel_6416\3286844220.py:19:
UserWarning: set_ticklabels() should only be used with a fixed number
of ticks, i.e. after set_ticks() or using a FixedLocator.
  ax.set_xticklabels(ax.get_xticklabels(), rotation=30, ha='right')
C:\Users\engah\AppData\Local\Temp\ipykernel_6416\3286844220.py:19:
UserWarning: set_ticklabels() should only be used with a fixed number
of ticks, i.e. after set_ticks() or using a FixedLocator.
  ax.set_xticklabels(ax.get_xticklabels(), rotation=30, ha='right')
C:\Users\engah\AppData\Local\Temp\ipykernel_6416\3286844220.py:19:
UserWarning: set_ticklabels() should only be used with a fixed number
of ticks, i.e. after set_ticks() or using a FixedLocator.
  ax.set_xticklabels(ax.get_xticklabels(), rotation=30, ha='right')
C:\Users\engah\AppData\Local\Temp\ipykernel_6416\3286844220.py:19:
UserWarning: set_ticklabels() should only be used with a fixed number
of ticks, i.e. after set_ticks() or using a FixedLocator.
  ax.set_xticklabels(ax.get_xticklabels(), rotation=30, ha='right')
```

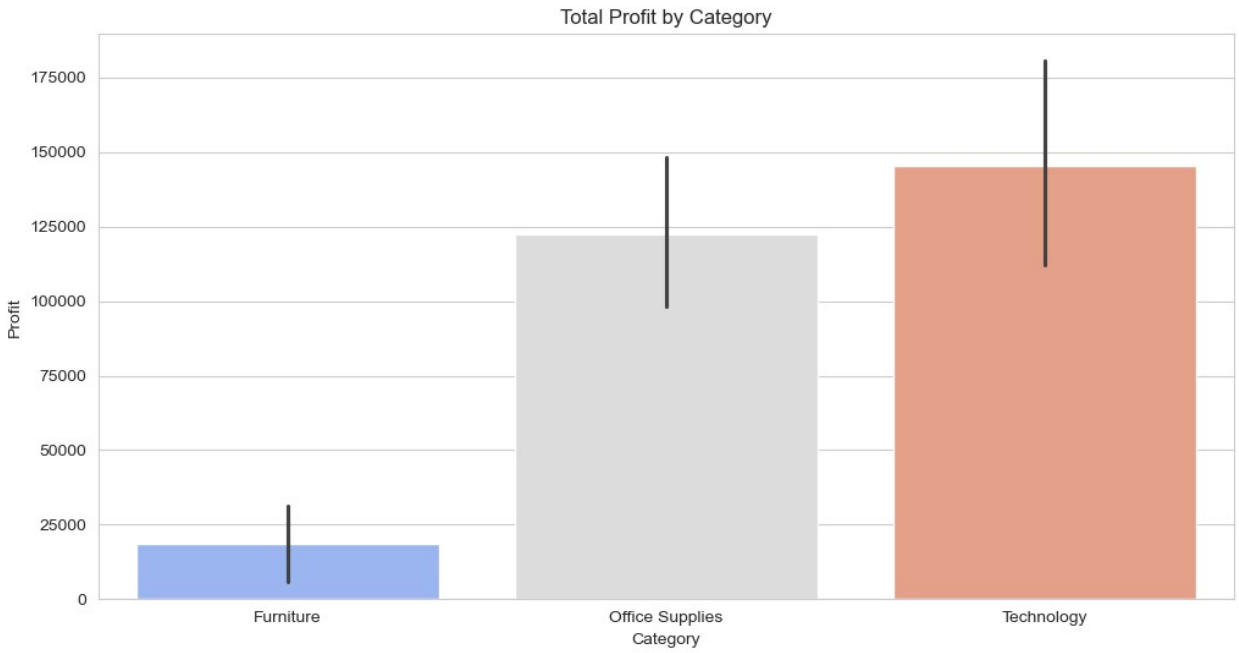



```
#Sales & Profit by Category

# Sales and Profit by Category
plt.figure(figsize=(12, 6))
sns.barplot(data=df, x='Category', y='Sales', estimator=sum,
palette='coolwarm')
plt.title('Total Sales by Category')
plt.show()

plt.figure(figsize=(12, 6))
sns.barplot(data=df, x='Category', y='Profit', estimator=sum,
palette='coolwarm')
plt.title('Total Profit by Category')
plt.show()
```

```
C:\Users\engah\AppData\Local\Temp\ipykernel_6416\1982947027.py:3:
FutureWarning:
```

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

  sns.barplot(data=df, x='Category', y='Sales', estimator=sum,
palette='coolwarm')
```

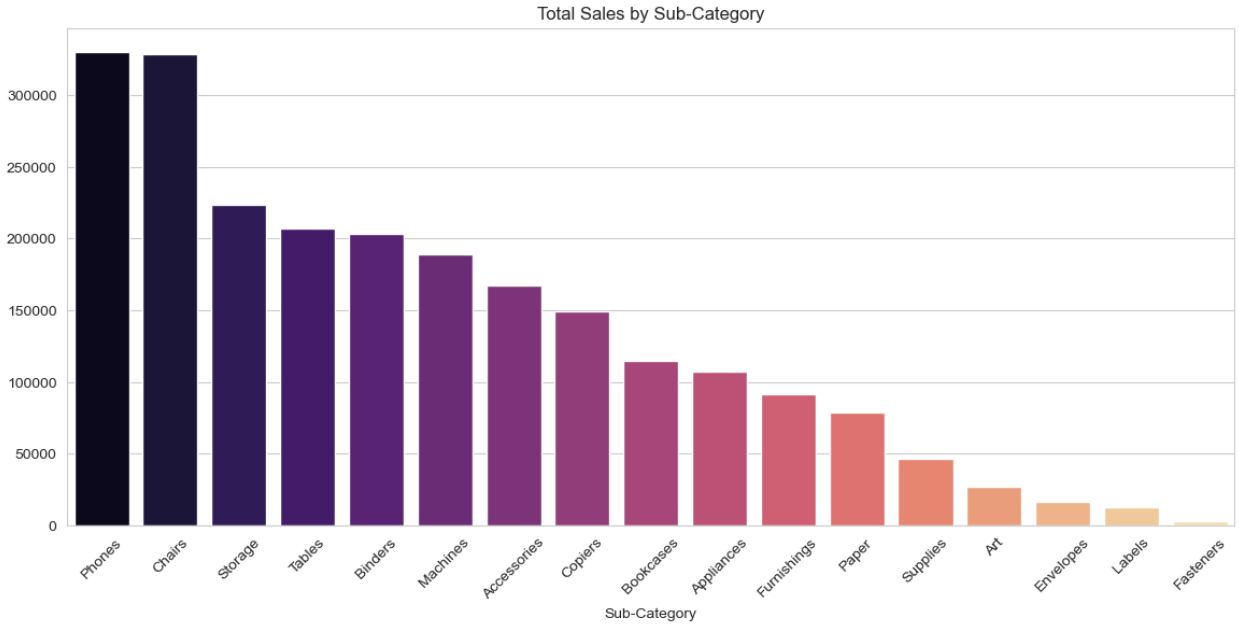

```
C:\Users\engah\AppData\Local\Temp\ipykernel_6416\1982947027.py:8:
FutureWarning:

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

  sns.barplot(data=df, x='Category', y='Profit', estimator=sum,
palette='coolwarm')
```



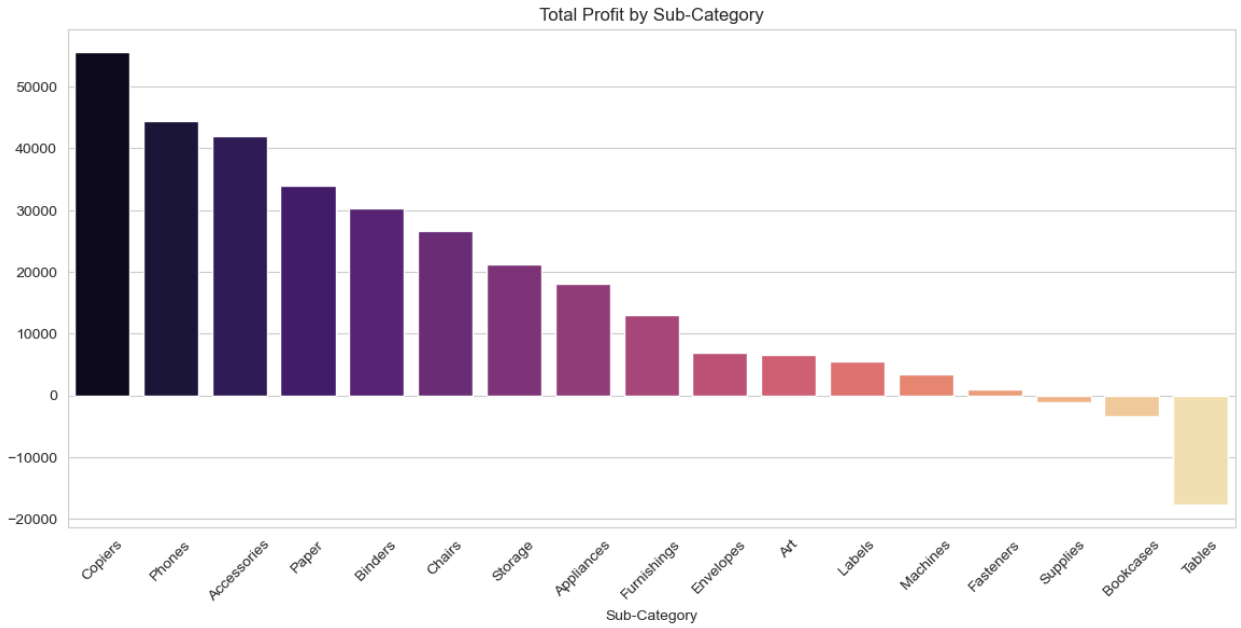

```
#Sales & Profit by Sub-Category
# Sorting sub-categories by sales for better visualization
sub_cat_sales = df.groupby('Sub-Category')
['Sales'].sum().sort_values(ascending=False)

plt.figure(figsize=(14, 6))
sns.barplot(x=sub_cat_sales.index, y=sub_cat_sales.values,
palette='magma')
plt.xticks(rotation=45)
plt.title("Total Sales by Sub-Category")
plt.show()

# Sorting sub-categories by profit
sub_cat_profit = df.groupby('Sub-Category')
['Profit'].sum().sort_values(ascending=False)

plt.figure(figsize=(14, 6))
sns.barplot(x=sub_cat_profit.index, y=sub_cat_profit.values,
palette='magma')
plt.xticks(rotation=45)
plt.title("Total Profit by Sub-Category")
plt.show()
```

```
C:\Users\engah\AppData\Local\Temp\ipykernel_6416\3078729398.py:6:
FutureWarning:

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

```
  sns.barplot(x=sub_cat_sales.index, y=sub_cat_sales.values,
palette='magma')
```

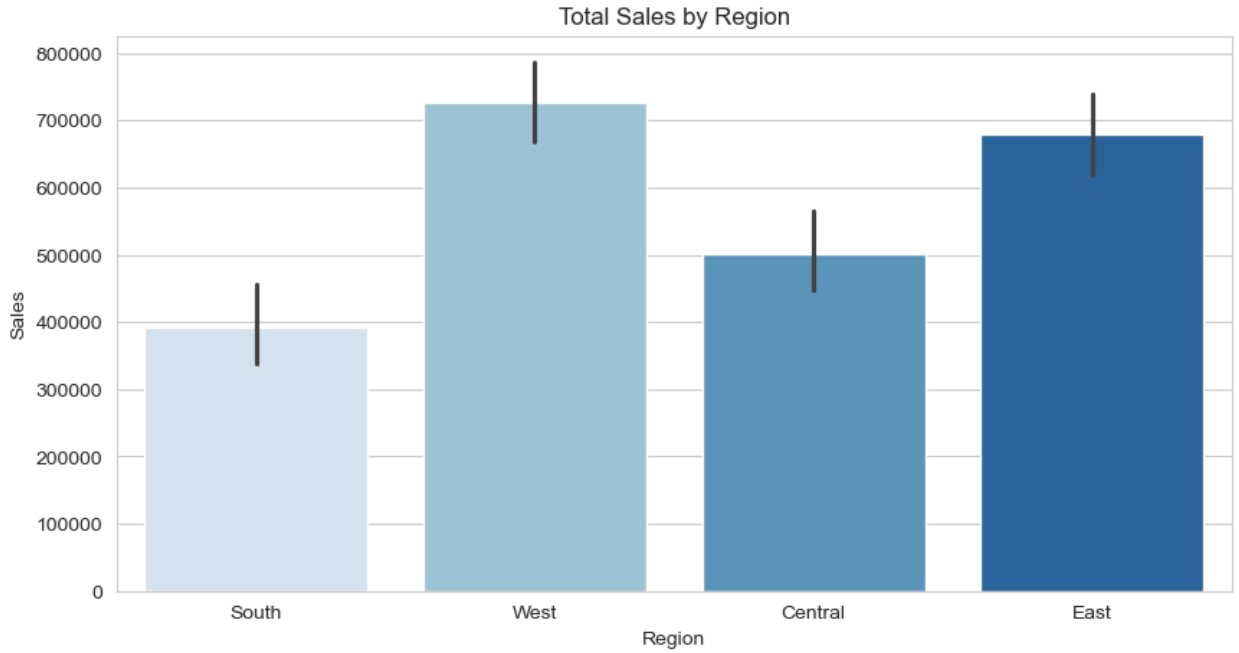



```
C:\Users\engah\AppData\Local\Temp\ipykernel_6416\3078729398.py:15:
FutureWarning:

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

  sns.barplot(x=sub_cat_profit.index, y=sub_cat_profit.values,
palette='magma')
```

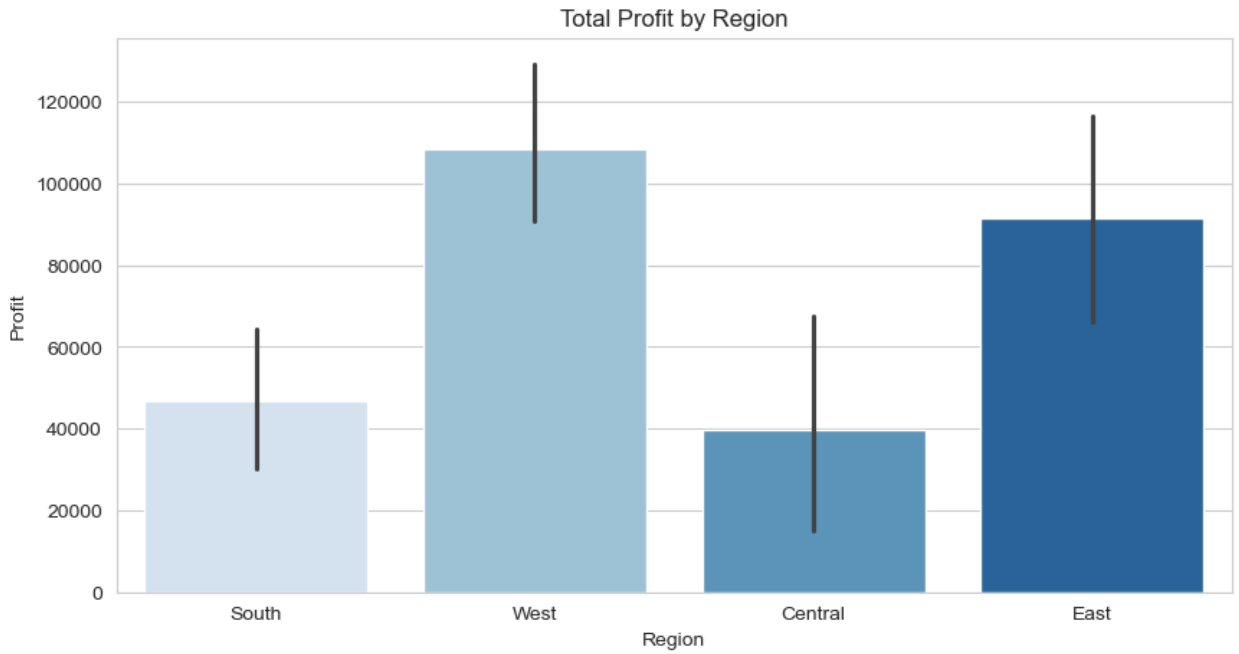



```
#Regional Analysis
# Sales and Profit by Region
plt.figure(figsize=(10, 5))
sns.barplot(data=df, x='Region', y='Sales', estimator=sum,
palette='Blues')
plt.title("Total Sales by Region")
plt.show()

plt.figure(figsize=(10, 5))
sns.barplot(data=df, x='Region', y='Profit', estimator=sum,
palette='Blues')
plt.title("Total Profit by Region")
plt.show()
```

```
C:\Users\engah\AppData\Local\Temp\ipykernel_6416\2926771101.py:4:
FutureWarning:

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

  sns.barplot(data=df, x='Region', y='Sales', estimator=sum,
palette='Blues')
```




```
C:\Users\engah\AppData\Local\Temp\ipykernel_6416\2926771101.py:9:
FutureWarning:

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

  sns.barplot(data=df, x='Region', y='Profit', estimator=sum,
palette='Blues')
```



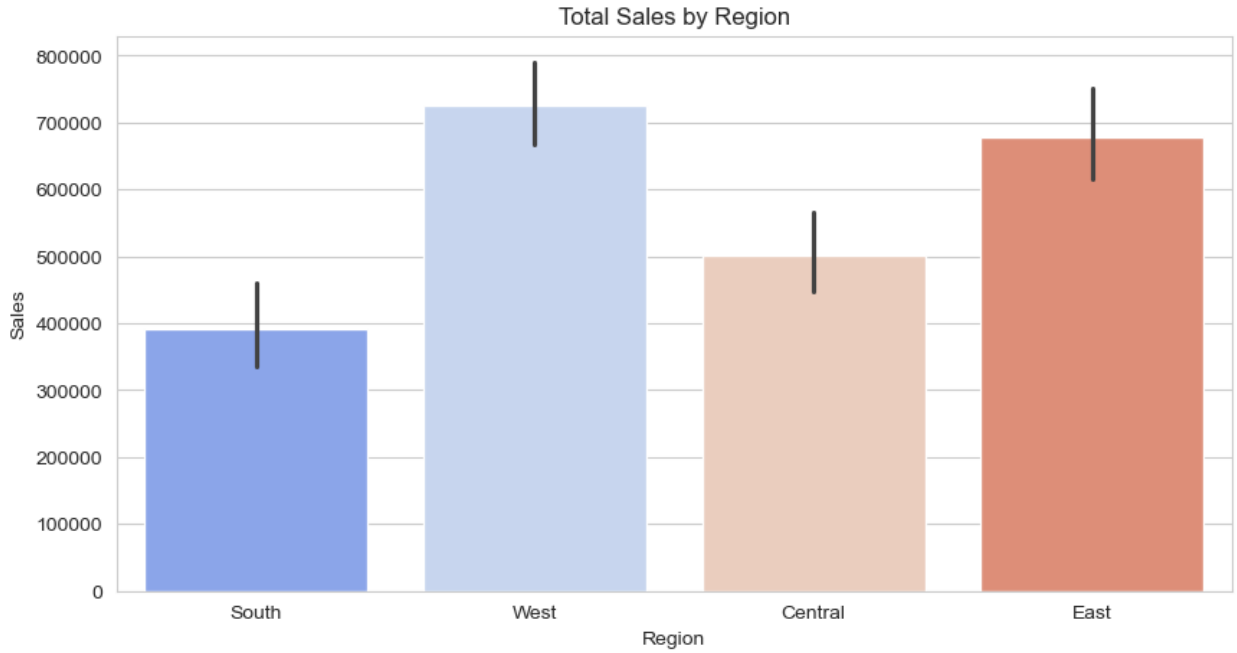
```
print(df.columns)
```

```
Index(['Ship Mode', 'Segment', 'Country', 'City', 'State', 'Postal
Code',
       'Region', 'Category', 'Sub-Category', 'Sales', 'Quantity',
'Discount',
       'Profit'],
      dtype='object')
```

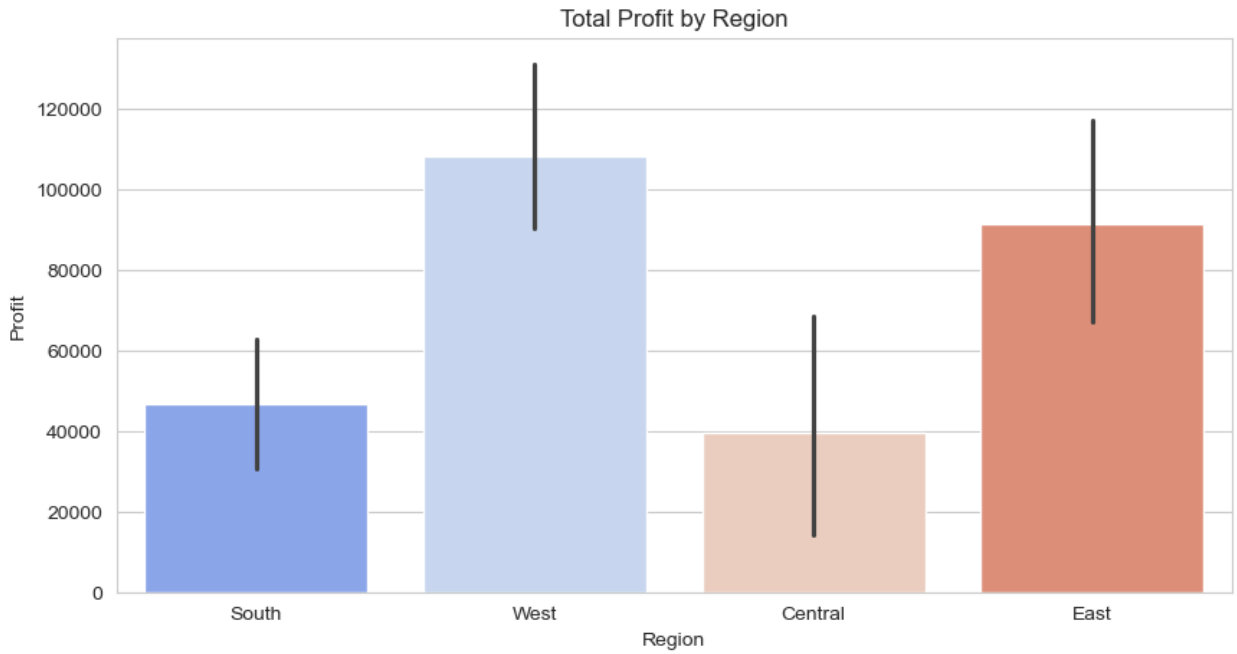
```
print(df.head())  # Check first few rows
print(df.info())  # Show column types
```

```
        Ship Mode    Segment        Country             City
State  \
0    Second Class   Consumer  United States        Henderson
Kentucky
1    Second Class   Consumer  United States        Henderson
Kentucky
2    Second Class  Corporate  United States      Los Angeles
California
3  Standard Class   Consumer  United States  Fort Lauderdale
Florida
4  Standard Class   Consumer  United States  Fort Lauderdale
Florida

   Postal Code Region         Category Sub-Category     Sales
Quantity  \
0        42420  South        Furniture    Bookcases  261.9600
2
1        42420  South        Furniture       Chairs  731.9400
3
2        90036   West  Office Supplies       Labels   14.6200
2
3        33311  South        Furniture       Tables  957.5775
5
4        33311  South  Office Supplies      Storage   22.3680
2

   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
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 13 columns):
 #   Column        Non-Null Count  Dtype
```

```
---  ------         --------------  -----
 0   Ship Mode      9994 non-null   object
 1   Segment        9994 non-null   object
 2   Country        9994 non-null   object
 3   City           9994 non-null   object
 4   State          9994 non-null   object
 5   Postal Code    9994 non-null   int64
 6   Region         9994 non-null   object
 7   Category       9994 non-null   object
 8   Sub-Category   9994 non-null   object
 9   Sales          9994 non-null   float64
 10  Quantity       9994 non-null   int64
 11  Discount       9994 non-null   float64
 12  Profit         9994 non-null   float64
dtypes: float64(3), int64(2), object(8)
memory usage: 1015.1+ KB
None

#Geographical Insights

#Total Sales and Profit by Region
# Total Sales and Profit by Region
plt.figure(figsize=(10, 5))
sns.barplot(data=df, x='Region', y='Sales', estimator=sum,
palette='coolwarm')
plt.title("Total Sales by Region")
plt.show()

plt.figure(figsize=(10, 5))
sns.barplot(data=df, x='Region', y='Profit', estimator=sum,
palette='coolwarm')
plt.title("Total Profit by Region")
plt.show()

C:\Users\engah\AppData\Local\Temp\ipykernel_6416\3386382487.py:4:
FutureWarning:

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

  sns.barplot(data=df, x='Region', y='Sales', estimator=sum,
palette='coolwarm')
```

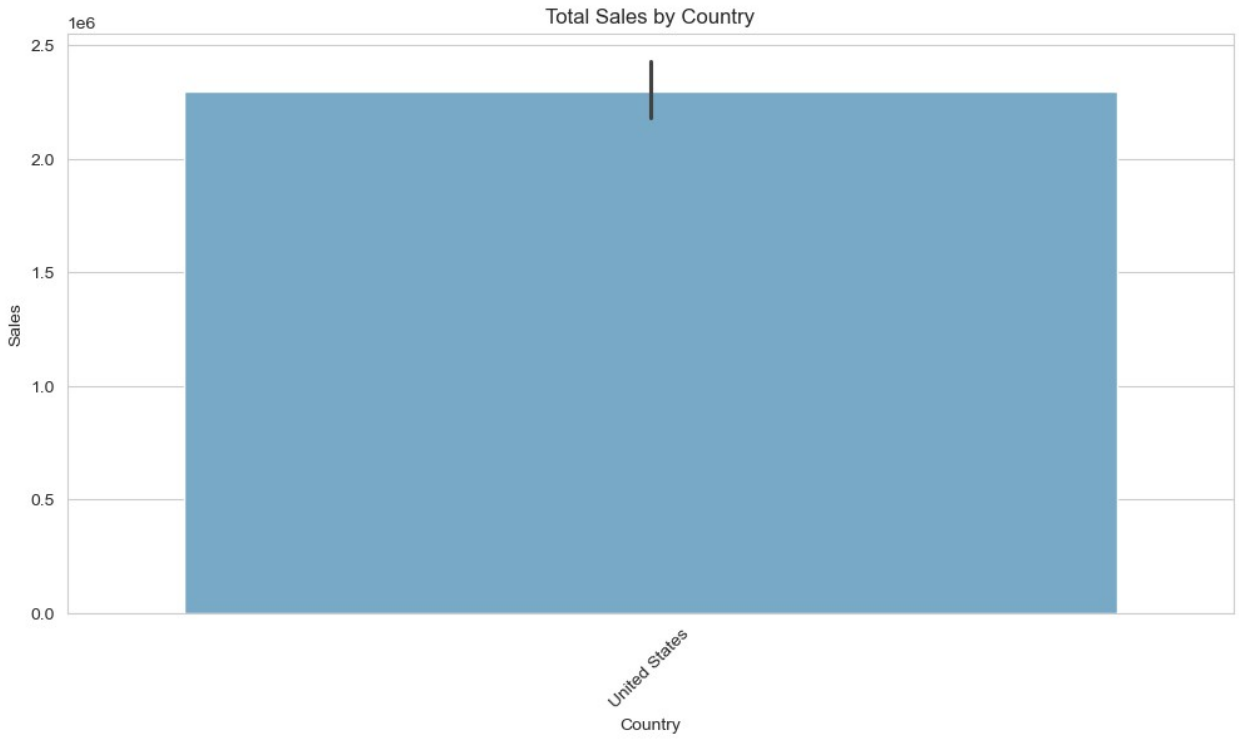



```
C:\Users\engah\AppData\Local\Temp\ipykernel_6416\3386382487.py:9:
FutureWarning:

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

  sns.barplot(data=df, x='Region', y='Profit', estimator=sum,
palette='coolwarm')
```



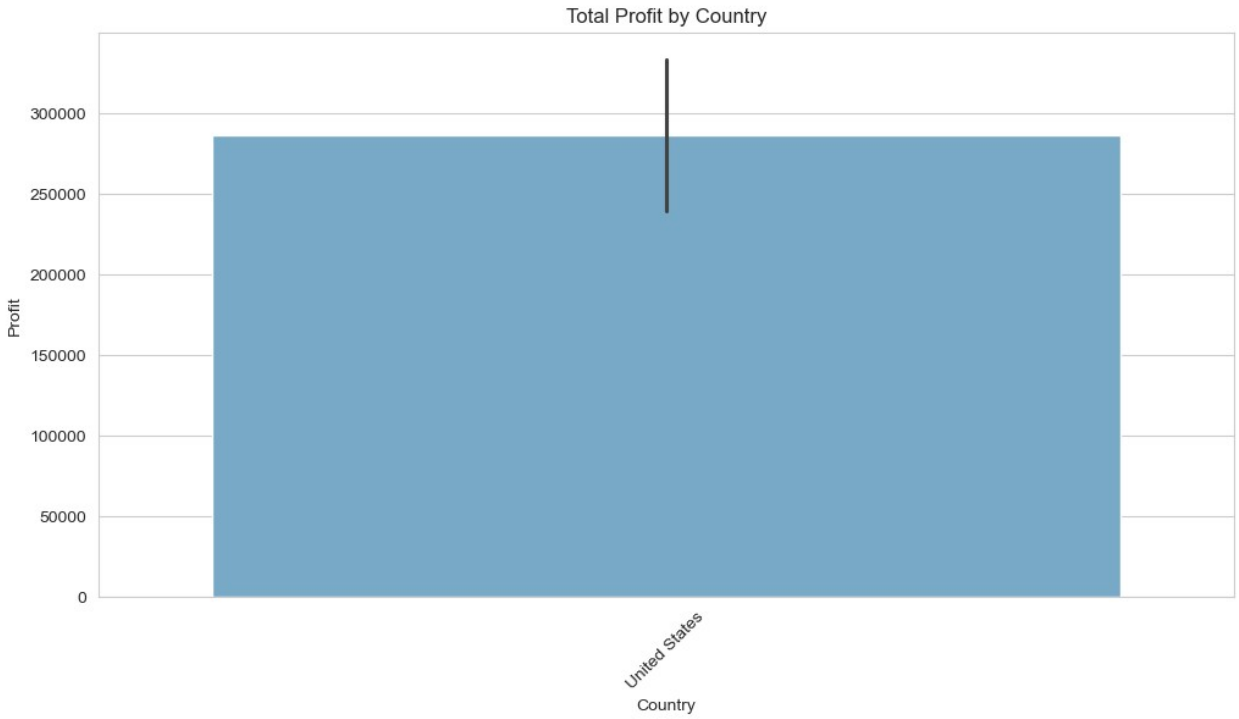
```
# Total Sales and Profit by Country
plt.figure(figsize=(12, 6))
sns.barplot(data=df, x='Country', y='Sales', estimator=sum,
palette='Blues')
plt.title("Total Sales by Country")
plt.xticks(rotation=45)
plt.show()

plt.figure(figsize=(12, 6))
sns.barplot(data=df, x='Country', y='Profit', estimator=sum,
palette='Blues')
plt.title("Total Profit by Country")
plt.xticks(rotation=45)
plt.show()
```

```
C:\Users\engah\AppData\Local\Temp\ipykernel_6416\3967917131.py:3:
FutureWarning:

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

  sns.barplot(data=df, x='Country', y='Sales', estimator=sum,
palette='Blues')
```



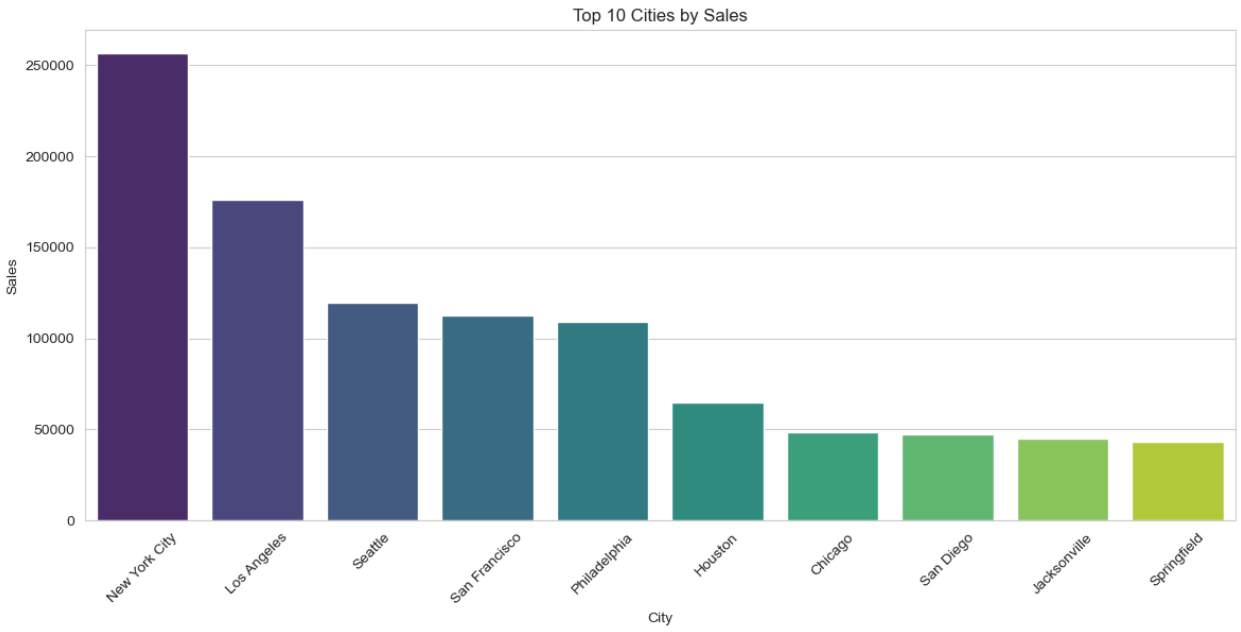

```
C:\Users\engah\AppData\Local\Temp\ipykernel_6416\3967917131.py:9:
FutureWarning:
```

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

  sns.barplot(data=df, x='Country', y='Profit', estimator=sum, palette='Blues')
```

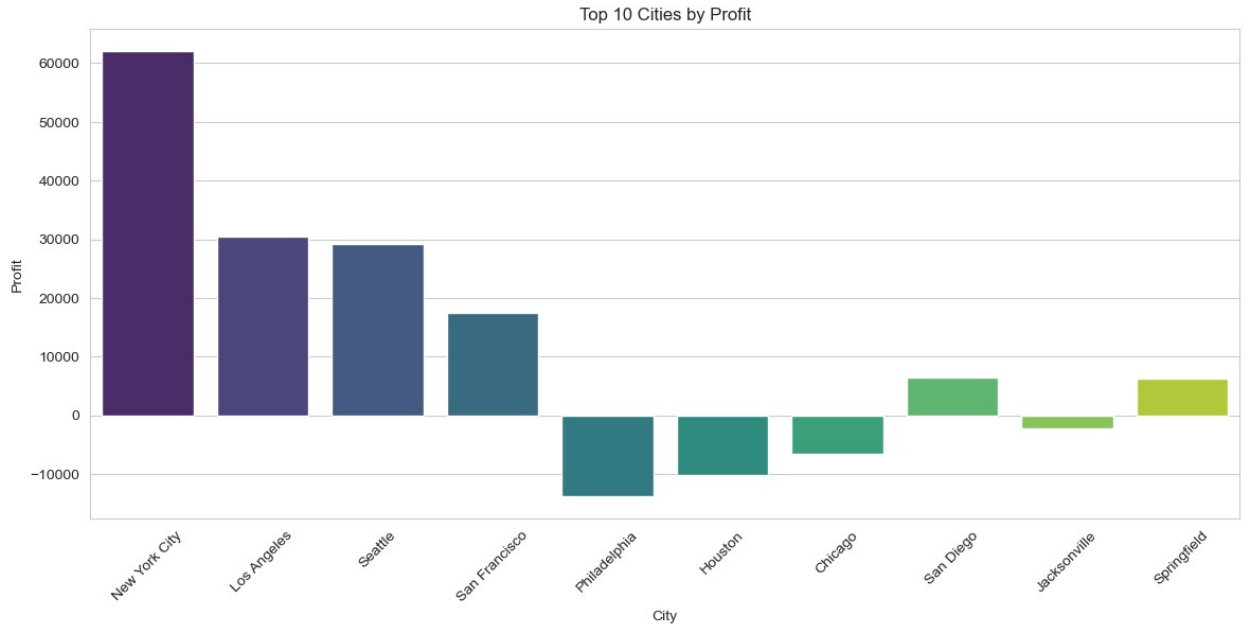



```
# Top 10 Cities by Sales
top_cities = df.groupby('City').agg({'Sales': 'sum', 'Profit': 'sum'}).sort_values(by='Sales', ascending=False).head(10)

# Plot Sales by Top 10 Cities
plt.figure(figsize=(14, 6))
sns.barplot(x=top_cities.index, y=top_cities['Sales'], palette='viridis')
plt.title("Top 10 Cities by Sales")
plt.xticks(rotation=45)
plt.show()

# Plot Profit by Top 10 Cities
plt.figure(figsize=(14, 6))
sns.barplot(x=top_cities.index, y=top_cities['Profit'], palette='viridis')
plt.title("Top 10 Cities by Profit")
plt.xticks(rotation=45)
plt.show()
```

```
C:\Users\engah\AppData\Local\Temp\ipykernel_6416\2684185783.py:6:
FutureWarning:

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

  sns.barplot(x=top_cities.index, y=top_cities['Sales'],
palette='viridis')
```

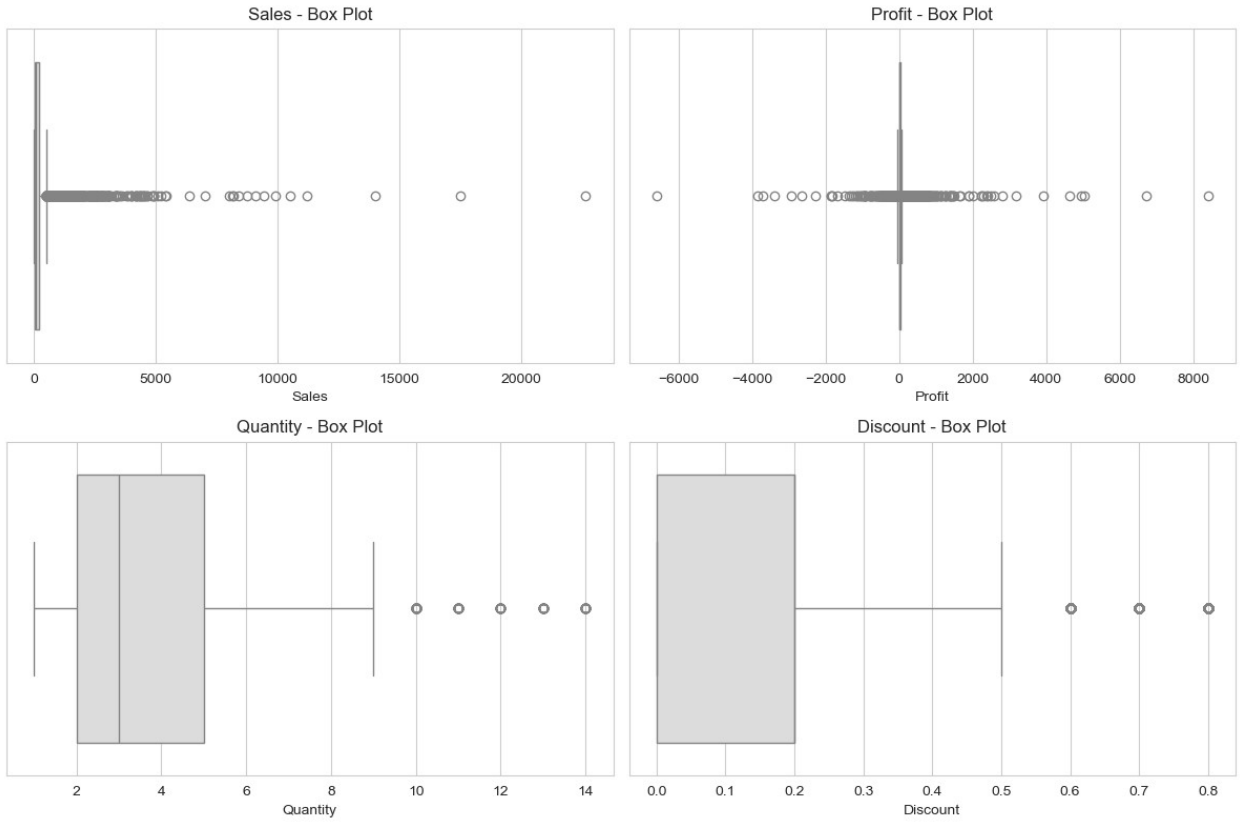



```
C:\Users\engah\AppData\Local\Temp\ipykernel_6416\2684185783.py:13:
FutureWarning:

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

  sns.barplot(x=top_cities.index, y=top_cities['Profit'],
palette='viridis')
```




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

# Set the size of the plots
plt.figure(figsize=(12, 8))

# Sales Box Plot
plt.subplot(2, 2, 1)
sns.boxplot(data=df, x='Sales', palette='coolwarm')
plt.title("Sales - Box Plot")

# Profit Box Plot
plt.subplot(2, 2, 2)
sns.boxplot(data=df, x='Profit', palette='coolwarm')
plt.title("Profit - Box Plot")

# Quantity Box Plot
plt.subplot(2, 2, 3)
sns.boxplot(data=df, x='Quantity', palette='coolwarm')
plt.title("Quantity - Box Plot")

# Discount Box Plot
plt.subplot(2, 2, 4)
sns.boxplot(data=df, x='Discount', palette='coolwarm')
plt.title("Discount - Box Plot")

# Display all the box plots
plt.tight_layout()
plt.show()
```

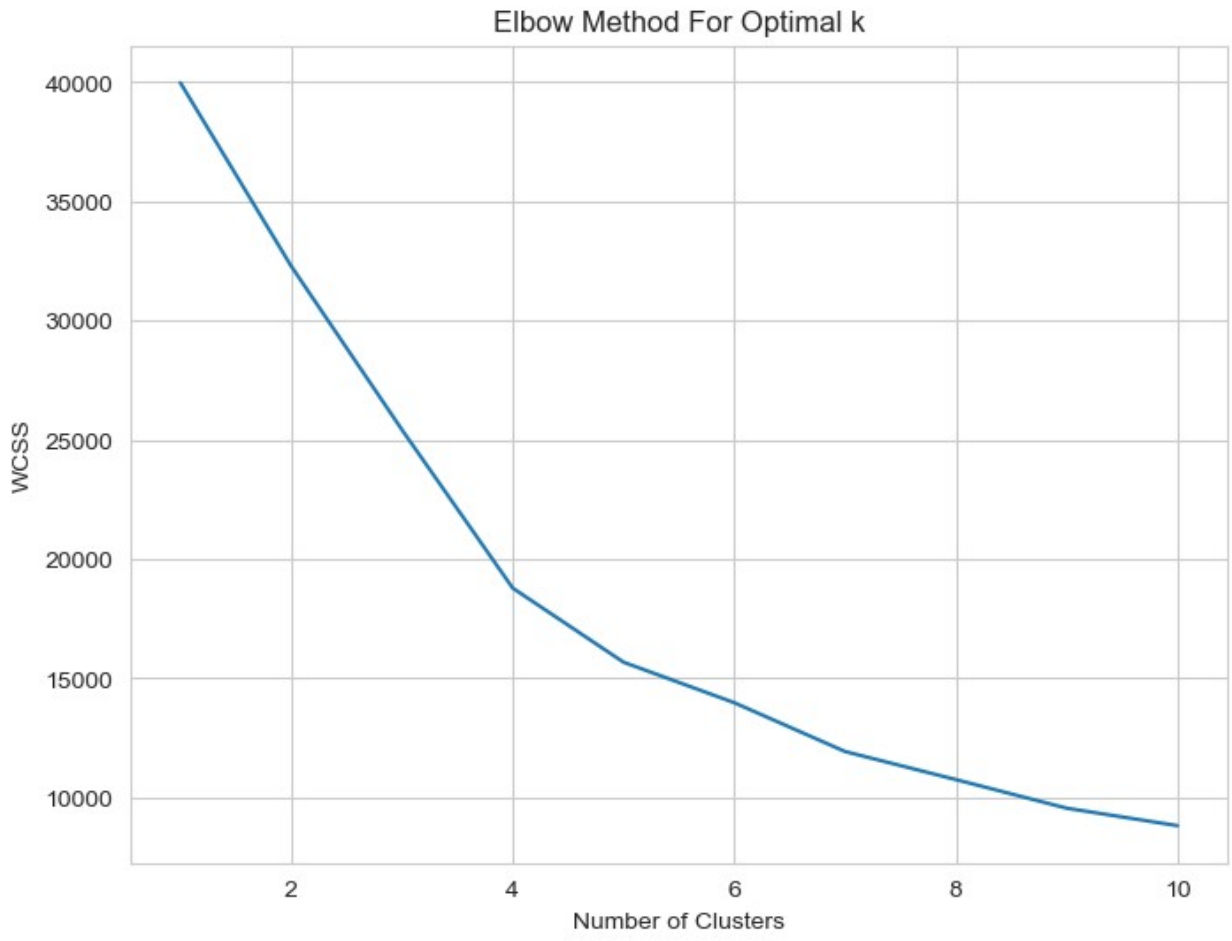
```
C:\Users\engah\AppData\Local\Temp\ipykernel_6416\4188732455.py:9:
FutureWarning:

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.

  sns.boxplot(data=df, x='Sales', palette='coolwarm')
C:\Users\engah\AppData\Local\Temp\ipykernel_6416\4188732455.py:14:
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.

  sns.boxplot(data=df, x='Profit', palette='coolwarm')
C:\Users\engah\AppData\Local\Temp\ipykernel_6416\4188732455.py:19:
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.

  sns.boxplot(data=df, x='Quantity', palette='coolwarm')
C:\Users\engah\AppData\Local\Temp\ipykernel_6416\4188732455.py:24:
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.

  sns.boxplot(data=df, x='Discount', palette='coolwarm')
```




```
#Code for Customer Segmentation Using K-Means
#1. Data Preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns

# Select relevant numerical columns
df_segmentation = df[['Sales', 'Quantity', 'Profit', 'Discount']]

# Handling missing values (if any)
df_segmentation = df_segmentation.dropna()

# Scale the data
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df_segmentation)

# Check the first few rows after scaling
df_scaled[:5]
```

```
array([[ 0.0515104 , -0.8043034 ,  0.05659251, -0.75664349],
       [ 0.80563348, -0.35486486,  0.81505408, -0.75664349],
       [-0.34536777, -0.8043034 , -0.09300169, -0.75664349],
       [ 1.16768814,  0.5440122 , -1.75748444,  1.42314932],
       [-0.33293544, -0.8043034 , -0.11159307,  0.21215332]])
```

```
# Determine the Optimal Number of Clusters (Elbow Method)

# Elbow Method to find optimal k
wcss = []
for i in range(1, 11):  # Trying cluster sizes from 1 to 10
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300,
n_init=10, random_state=42)
    kmeans.fit(df_scaled)
    wcss.append(kmeans.inertia_)

# Plot the Elbow Graph
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method For Optimal k')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```




```
#3. Apply K-Means Clustering
# Fit KMeans with the optimal number of clusters (let's assume k=4)
```

```
kmeans = KMeans(n_clusters=4, init='k-means++', max_iter=300,
n_init=10, random_state=42)
y_kmeans = kmeans.fit_predict(df_scaled)

# Add the cluster label to the original dataframe
df['Cluster'] = y_kmeans

# Check the first few rows of the dataframe with cluster labels
df[['Sales', 'Quantity', 'Profit', 'Discount', 'Cluster']].head()
```

	Sales	Quantity	Profit	Discount	Cluster
0	261.9600	2	41.9136	0.00	3
1	731.9400	3	219.5820	0.00	3
2	14.6200	2	6.8714	0.00	3
3	957.5775	5	-383.0310	0.45	1
4	22.3680	2	2.5164	0.20	3

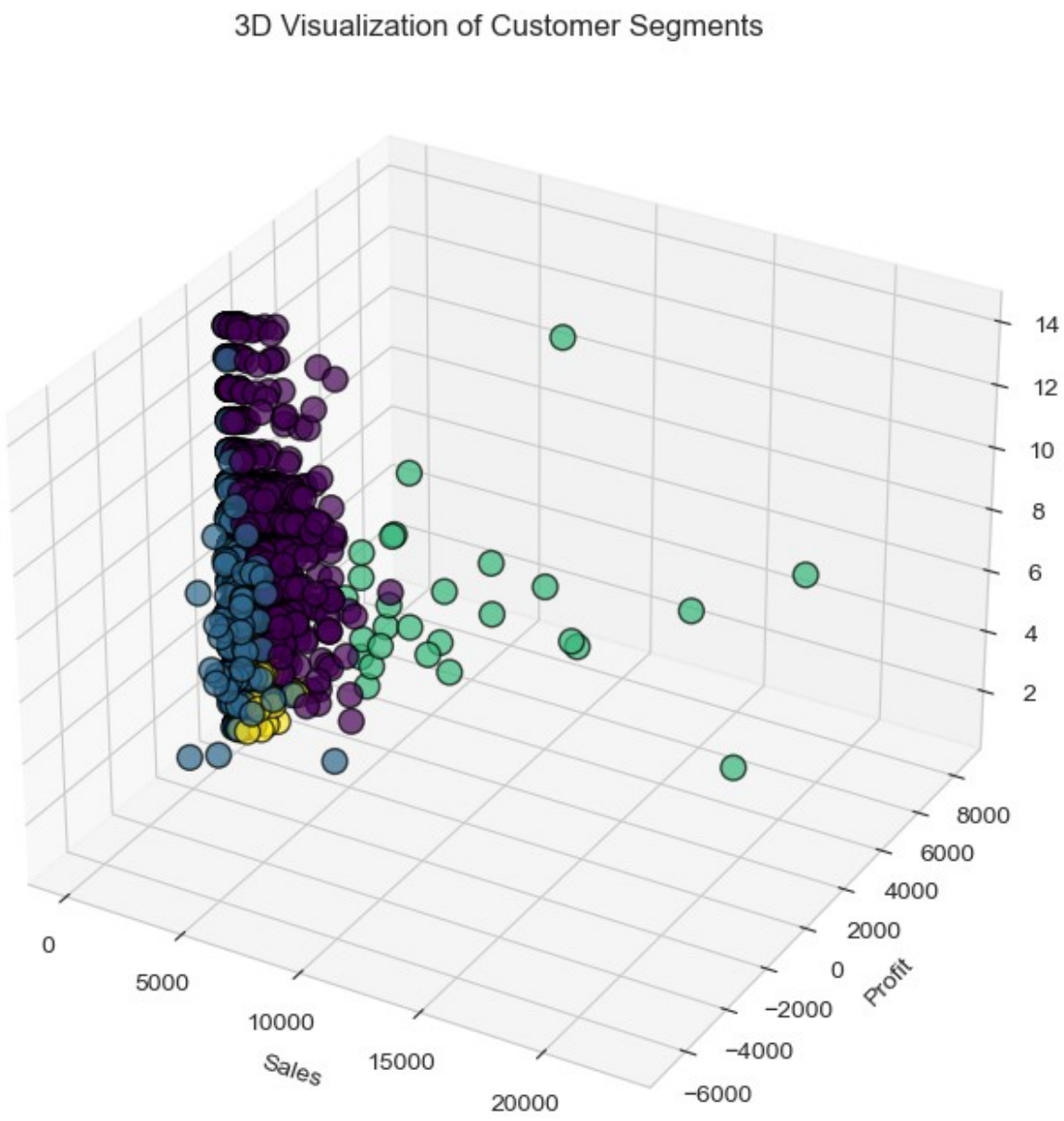
```
#4. Visualize the Segments
# Plotting 2D visualization of the clusters (Sales vs Profit)
plt.figure(figsize=(10, 8))
sns.scatterplot(data=df, x='Sales', y='Profit', hue='Cluster',
palette='viridis', s=100, alpha=0.7, edgecolor='black')
plt.title("Customer Segments - Sales vs Profit")
plt.xlabel("Sales")
plt.ylabel("Profit")
plt.legend(title="Cluster")
plt.show()
```


```
#5. Analyze Each Customer Segment
# Group by cluster and calculate the mean values for each cluster
segment_summary = df.groupby('Cluster')[['Sales', 'Quantity',
'Profit', 'Discount']].mean()
print(segment_summary)
```

```
                Sales  Quantity       Profit  Discount
Cluster
0          407.908020  6.452784    69.790464  0.094655
1          153.637349  3.836207  -108.306796  0.667960
2         7685.179259  5.185185  2610.220085  0.070370
3          126.813585  2.533279    21.516522  0.097487
```

```
#3D Visualization of Clusters
from mpl_toolkits.mplot3d import Axes3D

# 3D scatter plot (Sales, Profit, Quantity)
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
```

```
# Plotting the clusters
ax.scatter(df['Sales'], df['Profit'], df['Quantity'], c=df['Cluster'],
cmap='viridis', s=100, alpha=0.7, edgecolor='black')

ax.set_xlabel('Sales')
ax.set_ylabel('Profit')
ax.set_zlabel('Quantity')

plt.title("3D Visualization of Customer Segments")
plt.show()
```



```
# Additional

# Assuming the dataset has a 'Sales' and 'Cost' column (if not, you
can adjust)
# Feature: Profit Margin
df['Profit Margin'] = df['Profit'] / df['Sales'] * 100

# Feature: Sales per Unit
df['Sales Per Unit'] = df['Sales'] / df['Quantity']

# Feature: Discount Rate (Assuming the original price is available in
a column named 'Original Price')
# If the 'Original Price' column doesn't exist, this step is skipped.
df['Discount Rate'] = df['Discount'] / (df['Sales'] + df['Discount'])
* 100

# Show the new columns
print(df[['Profit Margin', 'Sales Per Unit', 'Discount Rate']].head())
```

```
   Profit Margin  Sales Per Unit  Discount Rate
0          16.00        130.9800       0.000000
1          30.00        243.9800       0.000000
2          47.00          7.3100       0.000000
3         -40.00        191.5155       0.046972
4          11.25         11.1840       0.886211
```

```
#Predictive Modeling
#1. Data Preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer

# Handling missing values
imputer = SimpleImputer(strategy='mean')  # Replace missing values
with mean for numerical columns
df[['Sales', 'Quantity', 'Profit', 'Discount']] =
imputer.fit_transform(df[['Sales', 'Quantity', 'Profit', 'Discount']])

# Encode categorical columns (Ship Mode, Segment, etc.)
label_encoder = LabelEncoder()
df['Ship Mode'] = label_encoder.fit_transform(df['Ship Mode'])
df['Segment'] = label_encoder.fit_transform(df['Segment'])
df['Country'] = label_encoder.fit_transform(df['Country'])
df['City'] = label_encoder.fit_transform(df['City'])
df['State'] = label_encoder.fit_transform(df['State'])
df['Region'] = label_encoder.fit_transform(df['Region'])
df['Category'] = label_encoder.fit_transform(df['Category'])
df['Sub-Category'] = label_encoder.fit_transform(df['Sub-Category'])

# Feature scaling for numerical columns (important for many models)
scaler = StandardScaler()
```

```
df[['Sales', 'Quantity', 'Profit', 'Discount']] =
scaler.fit_transform(df[['Sales', 'Quantity', 'Profit', 'Discount']])

# Check the dataframe after preprocessing
df.head()
```

```
   Ship Mode  Segment  Country  City  State  Postal Code  Region
Category  \
0          2        0        0   194     15        42420       2
0
1          2        0        0   194     15        42420       2
0
2          2        1        0   266      3        90036       3
1
3          3        0        0   153      8        33311       2
0
4          3        0        0   153      8        33311       2
1

   Sub-Category     Sales  Quantity  Discount    Profit  Cluster  \
0             4  0.051510 -0.804303 -0.756643  0.056593        3
1             5  0.805633 -0.354865 -0.756643  0.815054        3
2            10 -0.345368 -0.804303 -0.756643 -0.093002        3
3            16  1.167688  0.544012  1.423149 -1.757484        1
4            14 -0.332935 -0.804303  0.212153 -0.111593        3

   Profit Margin  Sales Per Unit  Discount Rate
0          16.00        130.9800       0.000000
1          30.00        243.9800       0.000000
2          47.00          7.3100       0.000000
3         -40.00        191.5155       0.046972
4          11.25         11.1840       0.886211
```

```
#2. Train-Test Split
# Define features and target variable (Predicting Profit)
X = df[['Sales', 'Quantity', 'Discount', 'Ship Mode', 'Segment',
'Country', 'City', 'State', 'Region', 'Category', 'Sub-Category']]
y = df['Profit']

# Train-test split (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Check the shapes of training and testing sets
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((7995, 11), (1999, 11), (7995,), (1999,))
```

```
#3. Model Selection: Linear Regression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error,
```

```
r2_score

# Initialize the model
lr_model = LinearRegression()

# Fit the model on the training data
lr_model.fit(X_train, y_train)

# Predict on the test data
y_pred = lr_model.predict(X_test)

# Evaluate the model's performance
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)

# Print evaluation metrics
print(f'Mean Absolute Error (MAE): {mae}')
print(f'Mean Squared Error (MSE): {mse}')
print(f'Root Mean Squared Error (RMSE): {rmse}')
print(f'R² (R-squared): {r2}')
```

```
Mean Absolute Error (MAE): 0.30751103926630197
Mean Squared Error (MSE): 1.5124368194330955
Root Mean Squared Error (RMSE): 1.2298117008034586
R² (R-squared): -0.7116774709454254
```

```
C:\Users\engah\anaconda3\Lib\site-packages\sklearn\metrics\
_regression.py:492: FutureWarning: 'squared' is deprecated in version
1.4 and will be removed in 1.6. To calculate the root mean squared
error, use the function'root_mean_squared_error'.
  warnings.warn(
```

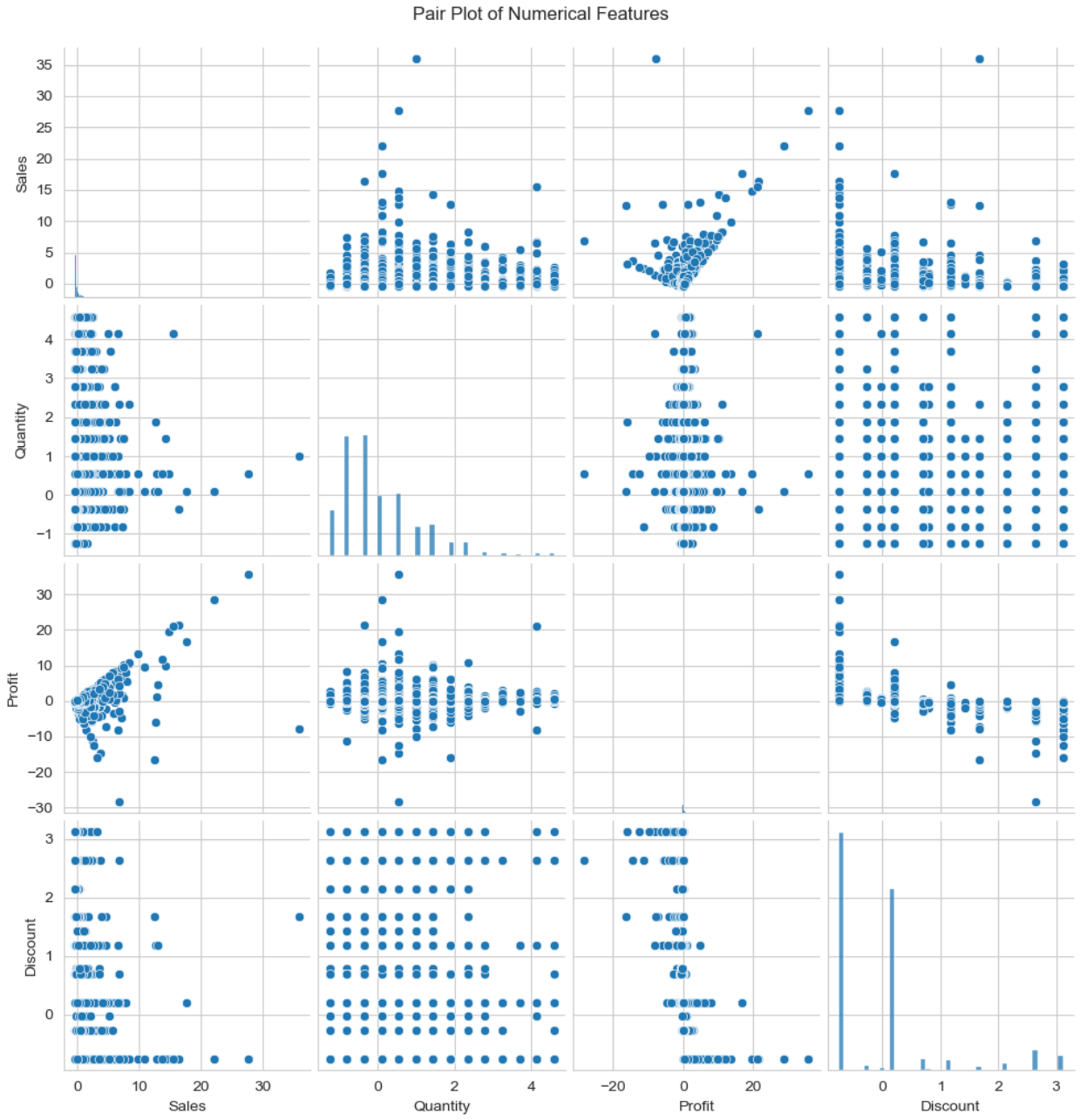
```
#Model Comparison
from sklearn.ensemble import RandomForestRegressor

# Initialize the Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the model on the training data
rf_model.fit(X_train, y_train)

# Predict on the test data
y_pred_rf = rf_model.predict(X_test)

# Evaluate the model's performance
mae_rf = mean_absolute_error(y_test, y_pred_rf)
mse_rf = mean_squared_error(y_test, y_pred_rf)
rmse_rf = mean_squared_error(y_test, y_pred_rf, squared=False)
r2_rf = r2_score(y_test, y_pred_rf)
```

```python
# Print evaluation metrics for Random Forest
print(f'Random Forest - MAE: {mae_rf}')
print(f'Random Forest - MSE: {mse_rf}')
print(f'Random Forest - RMSE: {rmse_rf}')
print(f'Random Forest - R²: {r2_rf}')
```

```
Random Forest - MAE: 0.11793597613276496
Random Forest - MSE: 1.0561515276353972
Random Forest - RMSE: 1.0276923312136748
Random Forest - R²: -0.19528350046100829
```

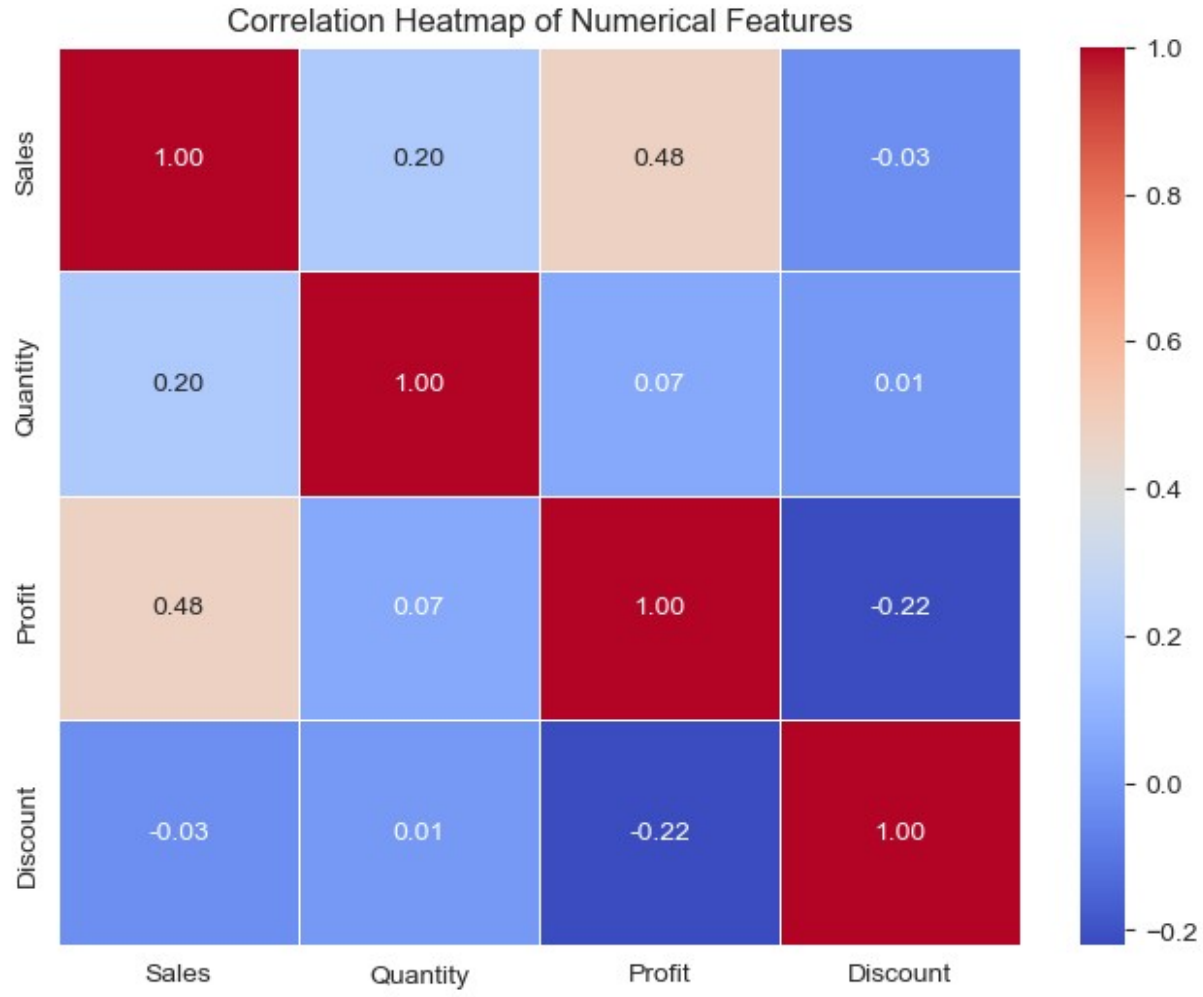
```
C:\Users\engah\anaconda3\Lib\site-packages\sklearn\metrics\
_regression.py:492: FutureWarning: 'squared' is deprecated in version
1.4 and will be removed in 1.6. To calculate the root mean squared
error, use the function'root_mean_squared_error'.
  warnings.warn(
```

```python
# Advanced Visualizations
#1. Pair Plot
import seaborn as sns
import matplotlib.pyplot as plt

# Pair plot to visualize relationships between numerical variables
sns.pairplot(df[['Sales', 'Quantity', 'Profit', 'Discount']])
plt.suptitle('Pair Plot of Numerical Features', y=1.02)
plt.show()
```

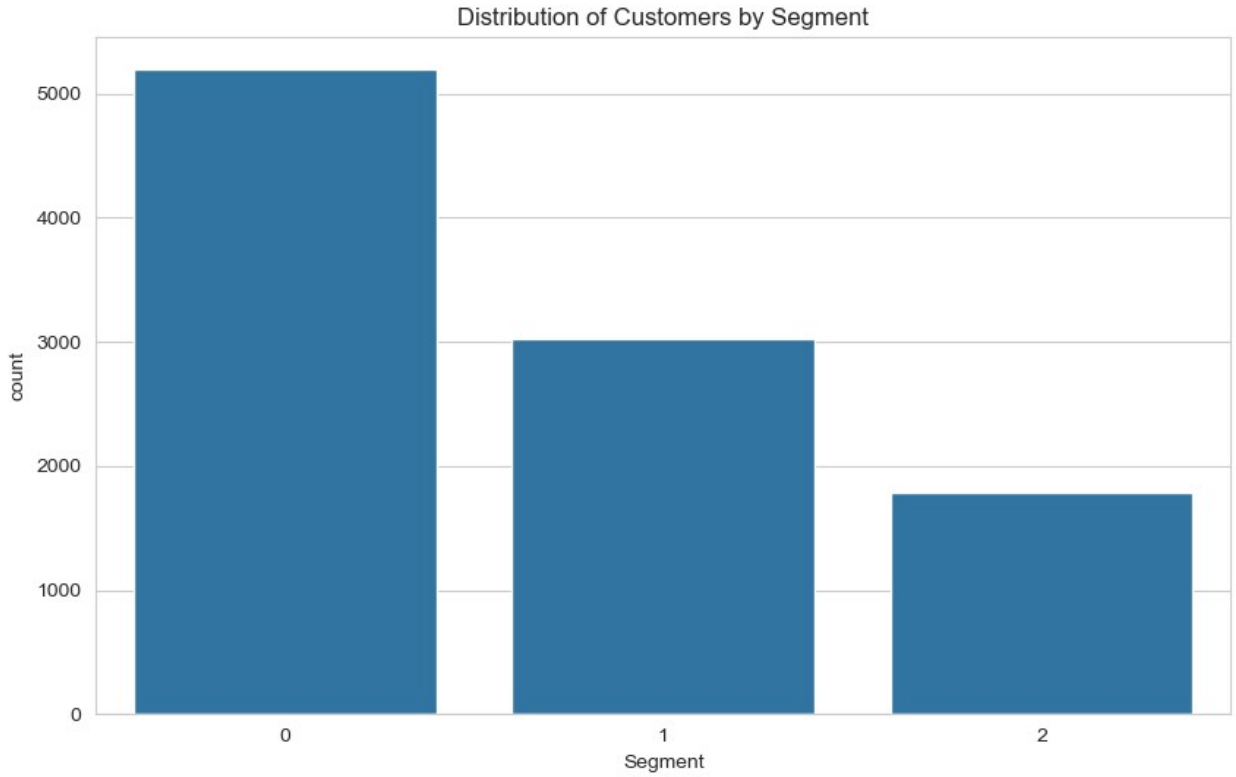



```
#2. Heatmap of Correlations
import seaborn as sns

# Correlation matrix
corr = df[['Sales', 'Quantity', 'Profit', 'Discount']].corr()

# Plot heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f',
linewidths=0.5)
```

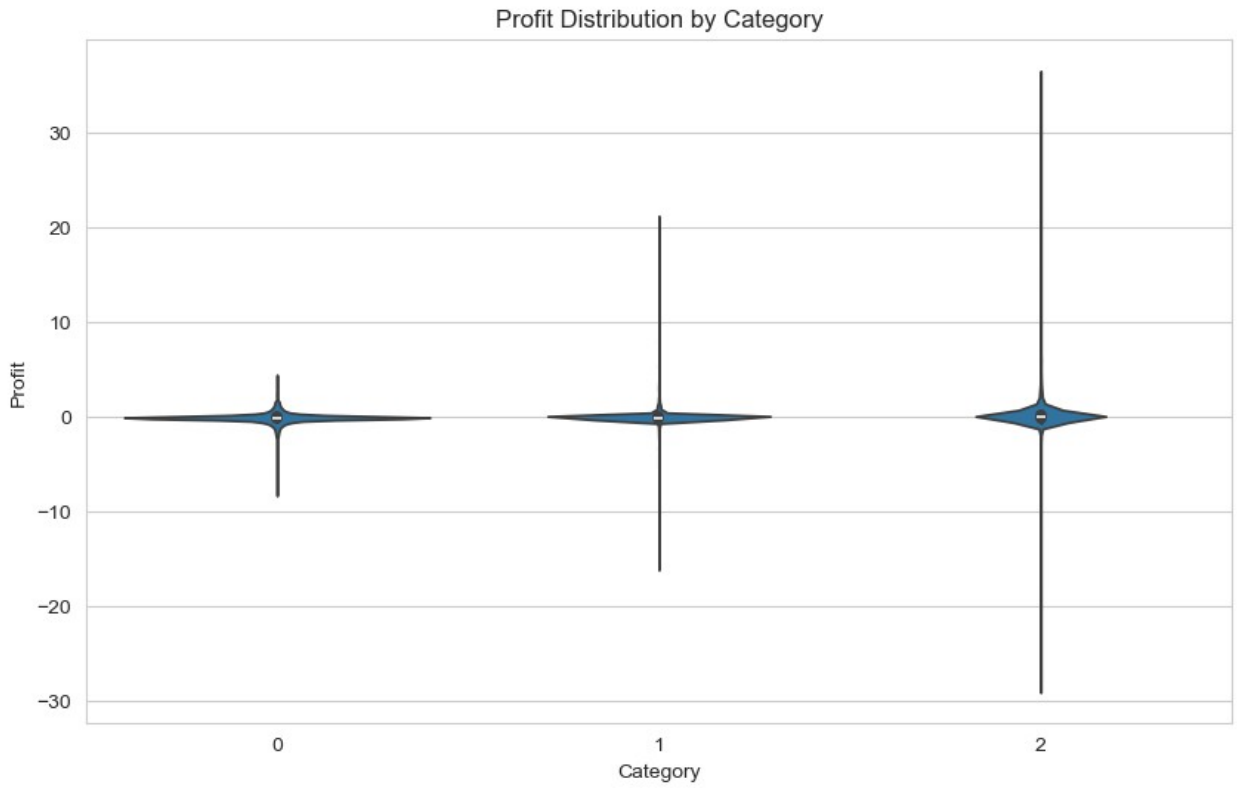
```
plt.title('Correlation Heatmap of Numerical Features')
plt.show()
```



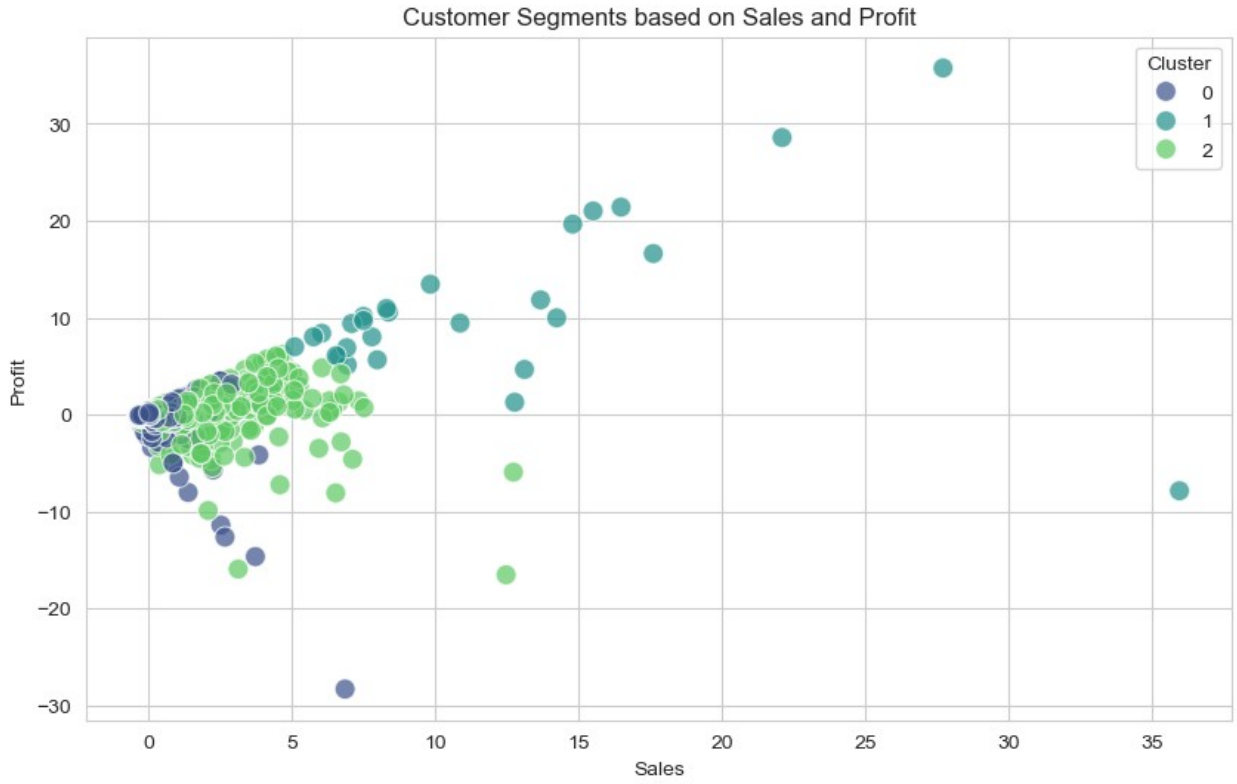

```
#3. Bar Plot for Categorical Distributions
# Bar plot for categorical variable (e.g., Segment)
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='Segment')
plt.title('Distribution of Customers by Segment')
plt.show()
```



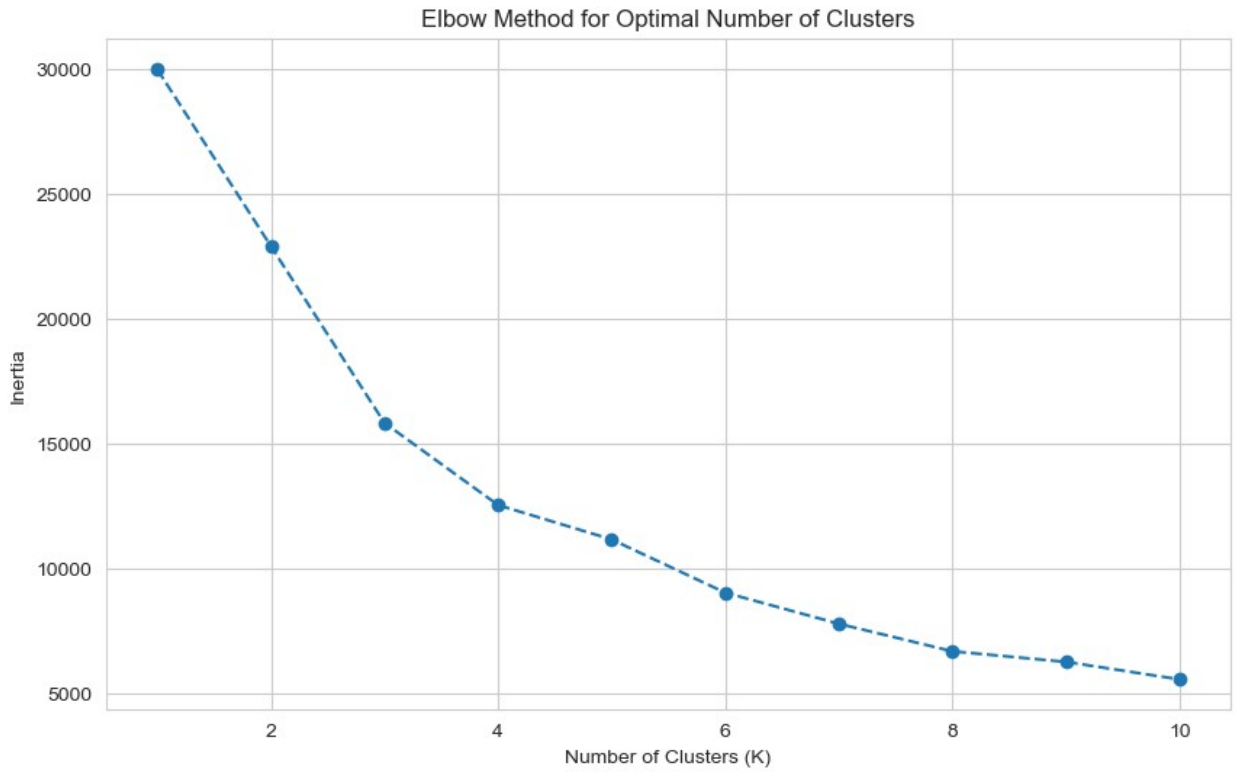

```
#4. Violin Plot
# Violin plot to show the distribution of Profit by Category
plt.figure(figsize=(10, 6))
sns.violinplot(data=df, x='Category', y='Profit')
plt.title('Profit Distribution by Category')
plt.show()
```




```
# Customer Segmentation Using K-Means Clustering
#1. K-Means Clustering Algorithm
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Select features for clustering (e.g., Sales, Quantity, and Profit)
X_segmentation = df[['Sales', 'Quantity', 'Profit']]

# Scaling the data before clustering
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_segmentation)

# Fit KMeans with 3 clusters (You can choose a different value for
'k')
kmeans = KMeans(n_clusters=3, random_state=42)
df['Cluster'] = kmeans.fit_predict(X_scaled)

# Add the cluster labels to the dataframe
df['Cluster'] = df['Cluster'].astype('category')

# Visualize the clusters using a scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Sales', y='Profit', hue='Cluster', data=df,
palette='viridis', s=100, alpha=0.7)
plt.title('Customer Segments based on Sales and Profit')
plt.show()
```




```
#2. Elbow Method to Find Optimal K
# Elbow method to determine the optimal number of clusters
inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Plot inertia vs number of clusters
plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), inertia, marker='o', linestyle='--')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.show()
```




```
#3. Interpreting Clusters
# Grouping by clusters to see average values for each cluster
cluster_summary = df.groupby('Cluster')[['Sales', 'Quantity',
'Profit']].mean()
print(cluster_summary)
```

```
              Sales  Quantity     Profit
Cluster
0        -0.169688 -0.429995  -0.076834
1        11.962700  0.627242  11.020623
2         0.436821  1.462214   0.130724

C:\Users\engah\AppData\Local\Temp\ipykernel_6416\1365968950.py:3:
FutureWarning: The default of observed=False is deprecated and will be
changed to True in a future version of pandas. Pass observed=False to
retain current behavior or observed=True to adopt the future default
and silence this warning.
  cluster_summary = df.groupby('Cluster')[['Sales', 'Quantity',
'Profit']].mean()
```

```
#4. Visualize Cluster Centers
# Plot the cluster centers on the scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Sales', y='Profit', hue='Cluster', data=df,
palette='viridis', s=100, alpha=0.7)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,
```

```
1], s=200, c='red', marker='X', label='Centroids')
plt.title('Customer Segments and Cluster Centers')
plt.legend()
plt.show()
```