In [54]:
```
# Importing All Library's

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

print('All library imported')
```

```
All library imported
```

In [55]:
```
# Read CSV file into dataframe
orders_df = pd.read_csv('Orders.csv', encoding='latin1')
returns_df = pd.read_csv('Returns.csv', encoding='latin1')
people_df = pd.read_csv('People.csv', encoding='latin1')
print('Data Imported')
```

```
Data Imported
```

In [56]:
```
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   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
```

In [57]:
```
returns_df
```

Out[57]:

	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
...	...	...
291	Yes	CA-2015-101910
292	Yes	CA-2017-156958
293	Yes	CA-2016-105585
294	Yes	CA-2016-148796
295	Yes	CA-2015-149636

296 rows × 2 columns

In [58]:
```
orders_df
```

Out[58]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country
0	1	CA-2016-152156	08-11-2016	11-11-2016	Second Class	CG-12520	Claire Gute	Consumer	United States
1	2	CA-2016-152156	08-11-2016	11-11-2016	Second Class	CG-12520	Claire Gute	Consumer	United States
2	3	CA-2016-138688	12-06-2016	16-06-2016	Second Class	DV-13045	Darrin Van Huff	Corporate	United States
3	4	US-2015-108966	11-10-2015	18-10-2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States
4	5	US-2015-108966	11-10-2015	18-10-2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	United States
...	...	...	...	...	...	...	...	...	...
9989	9990	CA-2014-110422	21-01-2014	23-01-2014	Second Class	TB-21400	Tom Boeckenhauer	Consumer	United States
9990	9991	CA-2017-121258	26-02-2017	03-03-2017	Standard Class	DB-13060	Dave Brooks	Consumer	United States
9991	9992	CA-2017-121258	26-02-2017	03-03-2017	Standard Class	DB-13060	Dave Brooks	Consumer	United States
9992	9993	CA-2017-121258	26-02-2017	03-03-2017	Standard Class	DB-13060	Dave Brooks	Consumer	United States
9993	9994	CA-2017-119914	04-05-2017	09-05-2017	Second Class	CC-12220	Chris Cortes	Consumer	United States

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country

9994 rows × 21 columns

In [59]:
```
# Convert date column from text to a datetime format
orders_df['Order Date'] = pd.to_datetime(orders_df['Order Date'], dayfirst = True)
orders_df['Ship Date'] = pd.to_datetime(orders_df['Ship Date'], dayfirst = True)

print("\nDate columns converted to datetime objects.")
```

```
Date columns converted to datetime objects.
```

In [60]:
```
# Check Missing Value
orders_df.isnull().sum()
```

Out[60]:
```
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
```

In [61]:
```
# Select all object columns
object_cols = orders_df.select_dtypes(include='object').columns

# Use a dictionary comprehensions to count blank values
blank_counts = {
    col: (orders_df[col].astype(str).str.strip() == '').sum()
    for col in object_cols
}
print("\nBlank Values (empty string) per object column (Alternative Method):\n", bl
```

```
Blank Values (empty string) per object column (Alternative Method):
 {'Order ID': 0, 'Ship Mode': 0, 'Customer ID': 0, 'Customer Name': 0, 'Segment': 0,
'Country': 0, 'City': 0, 'State': 0, 'Region': 0, 'Product ID': 0, 'Category': 0, 'S
ub-Category': 0, 'Product Name': 0}
```

In [62]:
```
print(orders_df.shape)
```

```
(9994, 21)
```

In [63]:
```
# Check Duplicate rows & entries and aggrgate
duplicate_rows = orders_df[orders_df.duplicated(subset=['Order ID','Product ID'], k
print("Found", len(duplicate_rows), "rows that are duplicates based on Order ID and
```

```
Found 16 rows that are duplicates based on Order ID and Product ID.
```

In [64]:
```
# Aggregate duplicate entries and store the result in new dataframe
agg_dict = {
    'Row ID': 'first',
    'Order ID': 'first',
    'Order Date': 'first',
    'Ship Date': 'first',
    'Ship Mode': 'first',
    'Customer ID': 'first',
    'Customer Name': 'first',
    'Segment': 'first',
    'Country': 'first',
    'City': 'first',
    'State': 'first',
    'Postal Code': 'first',
    'Region': 'first',
    'Product ID': 'first',
    'Category': 'first',
    'Sub-Category': 'first',
    'Product Name': 'first',
    'Sales': 'sum',
    'Quantity': 'sum',
    'Discount': 'first', # Often, discount is applied per order, not summed
    'Profit': 'sum'
}
# Perform Aggrgation
orders_data = orders_df.groupby(['Order ID', 'Product ID']).agg(agg_dict).reset_ind

print("\nOrders Data aggregated successfully with all columns.")
print("Shape of aggregated orders data:", orders_data.shape)
print("Columns in the new dataframe:", list(orders_data.columns))
```

```
Orders Data aggregated successfully with all columns.
Shape of aggregated orders data: (9986, 21)
Columns in the new dataframe: ['Row ID', 'Order ID', 'Order Date', 'Ship Date', 'Shi
p Mode', 'Customer ID', 'Customer Name', 'Segment', 'Country', 'City', 'State', 'Pos
tal Code', 'Region', 'Product ID', 'Category', 'Sub-Category', 'Product Name', 'Sale
s', 'Quantity', 'Discount', 'Profit']
```

In [65]:
```
orders_data
```

Out[65]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	
0	2718	CA-2014-100006	2014-09-07	2014-09-13	Standard Class	DK-13375	Dennis Kane	Consumer	United States	
1	6288	CA-2014-100090	2014-07-08	2014-07-12	Standard Class	EB-13705	Ed Braxton	Corporate	United States	
2	6289	CA-2014-100090	2014-07-08	2014-07-12	Standard Class	EB-13705	Ed Braxton	Corporate	United States	
3	9515	CA-2014-100293	2014-03-14	2014-03-18	Standard Class	NF-18475	Neil Französisch	Home Office	United States	J
4	3084	CA-2014-100328	2014-01-28	2014-02-03	Standard Class	JC-15340	Jasper Cacioppo	Consumer	United States	
...	...	...	...	...	...	...	...	...	...	
9981	5931	US-2017-169551	2017-07-07	2017-07-09	First Class	RL-19615	Rob Lucas	Consumer	United States	Pl
9982	5933	US-2017-169551	2017-07-07	2017-07-09	First Class	RL-19615	Rob Lucas	Consumer	United States	Pl
9983	5934	US-2017-169551	2017-07-07	2017-07-09	First Class	RL-19615	Rob Lucas	Consumer	United States	Pl
9984	5935	US-2017-169551	2017-07-07	2017-07-09	First Class	RL-19615	Rob Lucas	Consumer	United States	Pl
9985	5932	US-2017-169551	2017-07-07	2017-07-09	First Class	RL-19615	Rob Lucas	Consumer	United States	Pl

9986 rows × 21 columns

In [66]:
```
# First Merge: Join orders with return on 'Order ID'
merged_df = pd.merge(orders_data, returns_df, on = 'Order ID', how='left')
```

```
merged_df['Returned'] = merged_df['Returned'].fillna('No')
print("Orders & Returns data merged.")

#Second Merge: Join combined first joined data with people on 'Region'
superstore_df = pd.merge(merged_df, people_df, on='Region', how='left')

print("\nAll data merged successfully.")
print("Final DataFrame shape:", superstore_df.shape)
print("\nFinal DataFrame Info:")
superstore_df.info()
```

```
Orders & Returns data merged.

All data merged successfully.
Final DataFrame shape: (9986, 23)

Final DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9986 entries, 0 to 9985
Data columns (total 23 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Row ID         9986 non-null   int64
 1   Order ID       9986 non-null   object
 2   Order Date     9986 non-null   datetime64[ns]
 3   Ship Date      9986 non-null   datetime64[ns]
 4   Ship Mode      9986 non-null   object
 5   Customer ID    9986 non-null   object
 6   Customer Name  9986 non-null   object
 7   Segment        9986 non-null   object
 8   Country        9986 non-null   object
 9   City           9986 non-null   object
 10  State          9986 non-null   object
 11  Postal Code    9986 non-null   int64
 12  Region         9986 non-null   object
 13  Product ID     9986 non-null   object
 14  Category       9986 non-null   object
 15  Sub-Category   9986 non-null   object
 16  Product Name   9986 non-null   object
 17  Sales          9986 non-null   float64
 18  Quantity       9986 non-null   int64
 19  Discount       9986 non-null   float64
 20  Profit         9986 non-null   float64
 21  Returned       9986 non-null   object
 22  Person         9986 non-null   object
dtypes: datetime64[ns](2), float64(3), int64(3), object(15)
memory usage: 1.8+ MB
```

In [67]:
```
superstore_df
```

Out[67]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Country	
0	2718	CA-2014-100006	2014-09-07	2014-09-13	Standard Class	DK-13375	Dennis Kane	Consumer	United States	
1	6288	CA-2014-100090	2014-07-08	2014-07-12	Standard Class	EB-13705	Ed Braxton	Corporate	United States	
2	6289	CA-2014-100090	2014-07-08	2014-07-12	Standard Class	EB-13705	Ed Braxton	Corporate	United States	
3	9515	CA-2014-100293	2014-03-14	2014-03-18	Standard Class	NF-18475	Neil Französisch	Home Office	United States	J
4	3084	CA-2014-100328	2014-01-28	2014-02-03	Standard Class	JC-15340	Jasper Cacioppo	Consumer	United States	
...	...	...	...	...	...	...	...	...	...	
9981	5931	US-2017-169551	2017-07-07	2017-07-09	First Class	RL-19615	Rob Lucas	Consumer	United States	Pl
9982	5933	US-2017-169551	2017-07-07	2017-07-09	First Class	RL-19615	Rob Lucas	Consumer	United States	Pl
9983	5934	US-2017-169551	2017-07-07	2017-07-09	First Class	RL-19615	Rob Lucas	Consumer	United States	Pl
9984	5935	US-2017-169551	2017-07-07	2017-07-09	First Class	RL-19615	Rob Lucas	Consumer	United States	Pl
9985	5932	US-2017-169551	2017-07-07	2017-07-09	First Class	RL-19615	Rob Lucas	Consumer	United States	Pl

9986 rows × 23 columns

Exploratory Data Analysis (EDA)

In [68]:
```
# Total Sales and Profit by Region
regional_summary = superstore_df.groupby('Region').agg(
    TotalSales = ('Sales', 'sum'),
    TotalProfit = ('Profit','sum')
).sort_values(by='TotalSales', ascending=False)
print("Total Sales and Profit by Region:")
print(regional_summary.to_markdown(numalign="left", stralign="left"))
```

```
Total Sales and Profit by Region:
| Region   | TotalSales   | TotalProfit   |
|:---------|:-------------|:--------------|
| West     | 725458       | 108418        |
| East     | 678781       | 91522.8       |
| Central  | 501240       | 39706.4       |
| South    | 391722       | 46749.4       |
```

In [69]:
```
# Top 10 Most Profitable Products
top_profitable_products = superstore_df.groupby(['Product Name', 'Category']).agg(
    TotalProfit = ('Profit','sum')
).sort_values(by='TotalProfit', ascending=False).head(10)
print("\nTop 10 Most Profitable Products:")
print(top_profitable_products.to_markdown(numalign="left", stralign="left"))
```

```
Top 10 Most Profitable Products:
|
| TotalProfit   |
|:-------------------------------------------------------------------------------------------------------------------------------|:--------------|
| ('Canon imageCLASS 2200 Advanced Copier', 'Technology')
| 25199.9       |
| ('Fellowes PB500 Electric Punch Plastic Comb Binding Machine with Manual Bind', 'Office Supplies') | 7753.04       |
| ('Hewlett Packard LaserJet 3310 Copier', 'Technology')
| 6983.88       |
| ('Canon PC1060 Personal Laser Copier', 'Technology')
| 4570.93       |
| ('HP Designjet T520 Inkjet Large Format Printer - 24" Color', 'Technology')
| 4094.98       |
| ('Ativa V4110MDD Micro-Cut Shredder', 'Technology')
| 3772.95       |
| ('3D Systems Cube Printer, 2nd Generation, Magenta', 'Technology')
| 3717.97       |
| ('Plantronics Savi W720 Multi-Device Wireless Headset System', 'Technology')
| 3696.28       |
| ('Ibico EPK-21 Electric Binding System', 'Office Supplies')
| 3345.28       |
| ('Zebra ZM400 Thermal Label Printer', 'Technology')
| 3343.54       |
```

In [70]:
```
# Sales and Profit by Customer Segment
seg_summ = superstore_df.groupby('Segment').agg(
    TotalSales = ('Sales', 'sum'),
    TotalProfit = ('Profit','sum')
).sort_values(by='TotalSales', ascending=False)
```

```
print("Total Sales and Profit by Customer Segement:")
print(seg_summ.to_markdown(numalign="left", stralign="left"))
```

Total Sales and Profit by Customer Segement:

Segment	TotalSales	TotalProfit
Consumer	1.1614e+06	134119
Corporate	706146	91979.1
Home Office	429653	60298.7

In [71]:
```
# Aggregate Sales by Month
monthly_sales = superstore_df.set_index('Order Date').resample('ME')['Sales'].sum()
monthly_sales['YearMonth'] = monthly_sales['Order Date'].dt.to_period('M')
monthly_sales['YearMonth'] = monthly_sales['YearMonth'].astype(str)

print("Monthly sales data aggregated successfully.")
```

Monthly sales data aggregated successfully.

In [72]:
```
# 1. Aggregate Sales by Year using superstore_df
# We use 'YE' for Year End and reset the index
yearly_sales = superstore_df.set_index('Order Date').resample('YE')['Sales'].sum().

# FIX 1: Convert the 'Year' to a string for simple plotting labels
yearly_sales['Year'] = yearly_sales['Order Date'].dt.to_period('Y').astype(str)
```

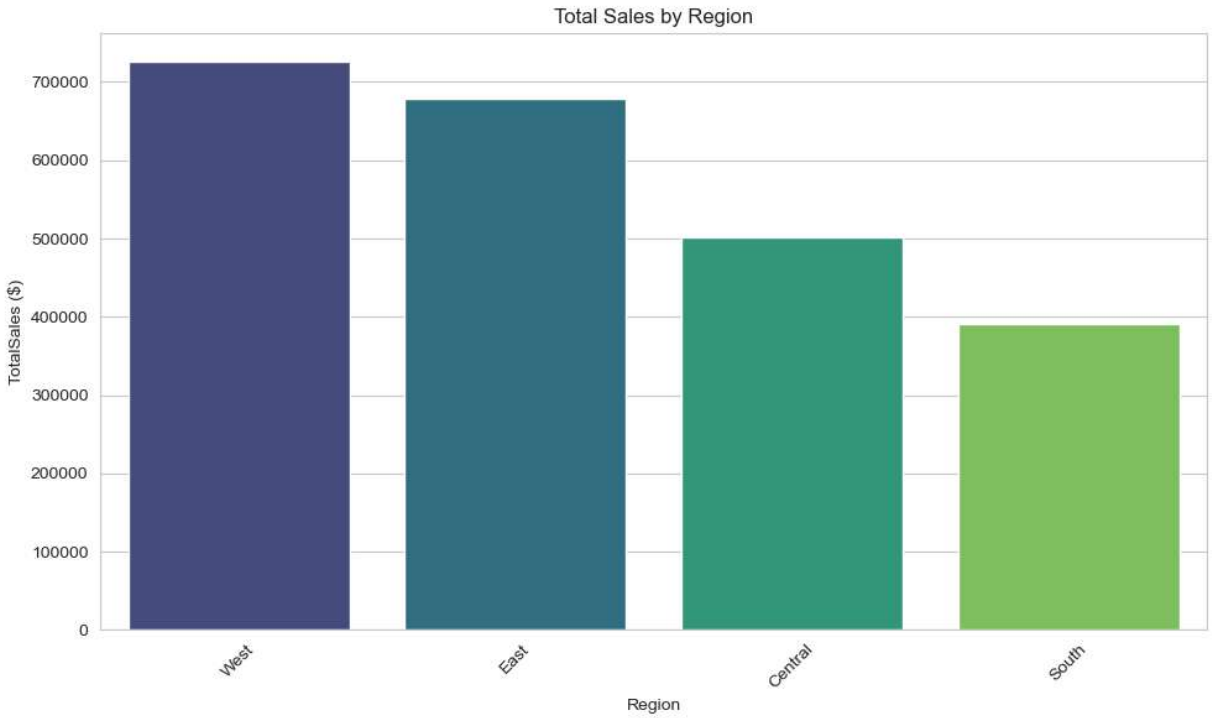
Data Visualization

In [73]:
```
# Set a style for the plots
sns.set_style("whitegrid")
plt.figure(figsize=(12,6))
```

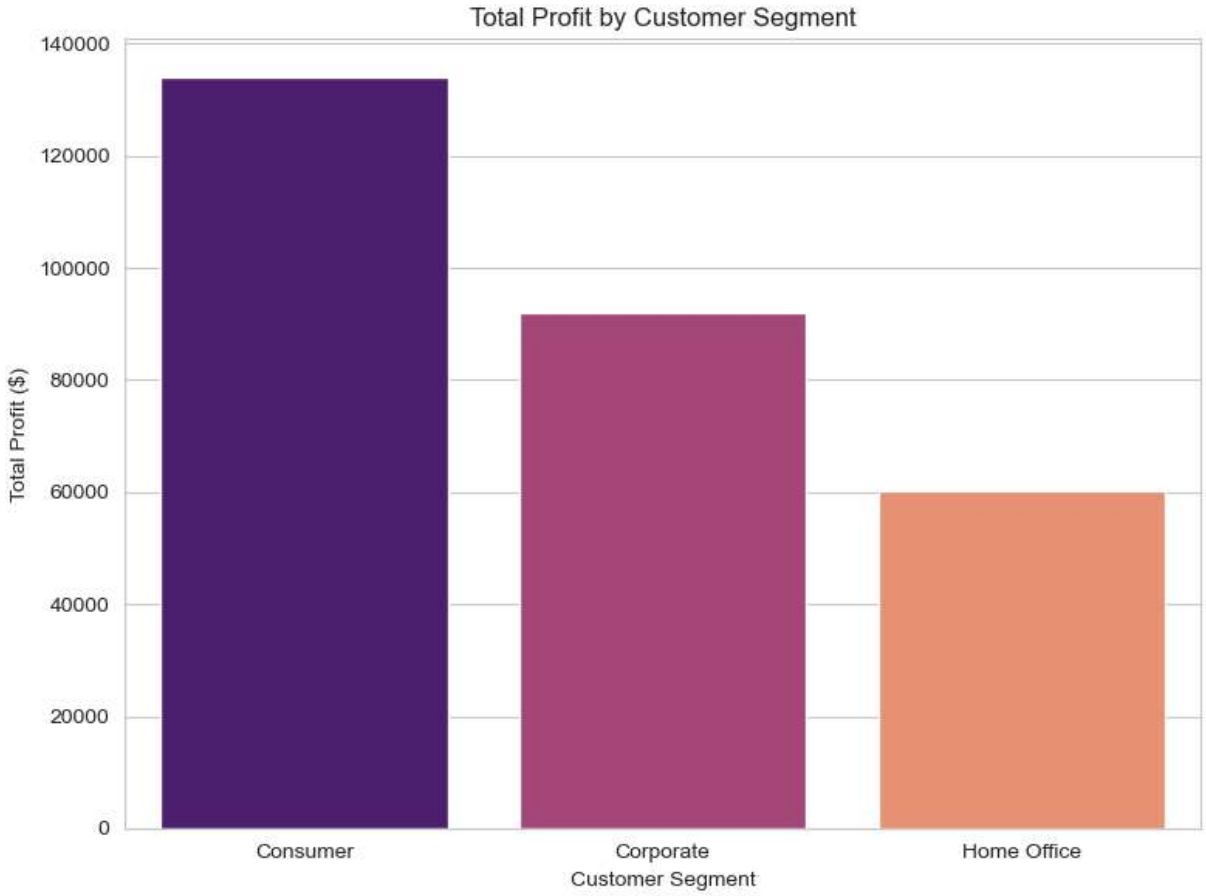
Out[73]: <Figure size 1200x600 with 0 Axes>

<Figure size 1200x600 with 0 Axes>

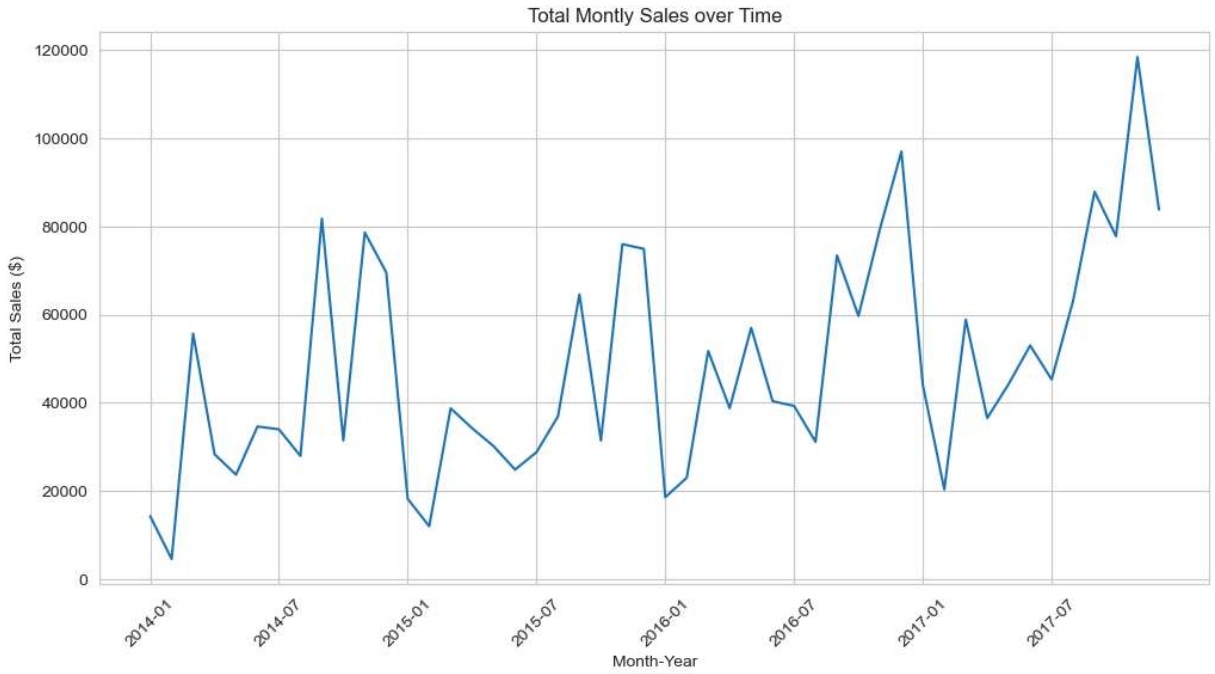
In [74]:
```
# Plot 1: Bar plot of Total Sales by Region
plt.figure(figsize=(10,6))
sns.barplot(
    x=regional_summary.index,
    y=regional_summary['TotalSales'],
    hue=regional_summary.index,
    palette='viridis',
    legend=False
)
plt.title('Total Sales by Region')
plt.xlabel('Region')
plt.ylabel('TotalSales ($)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig('sales_by_region.png', bbox_inches='tight')
```



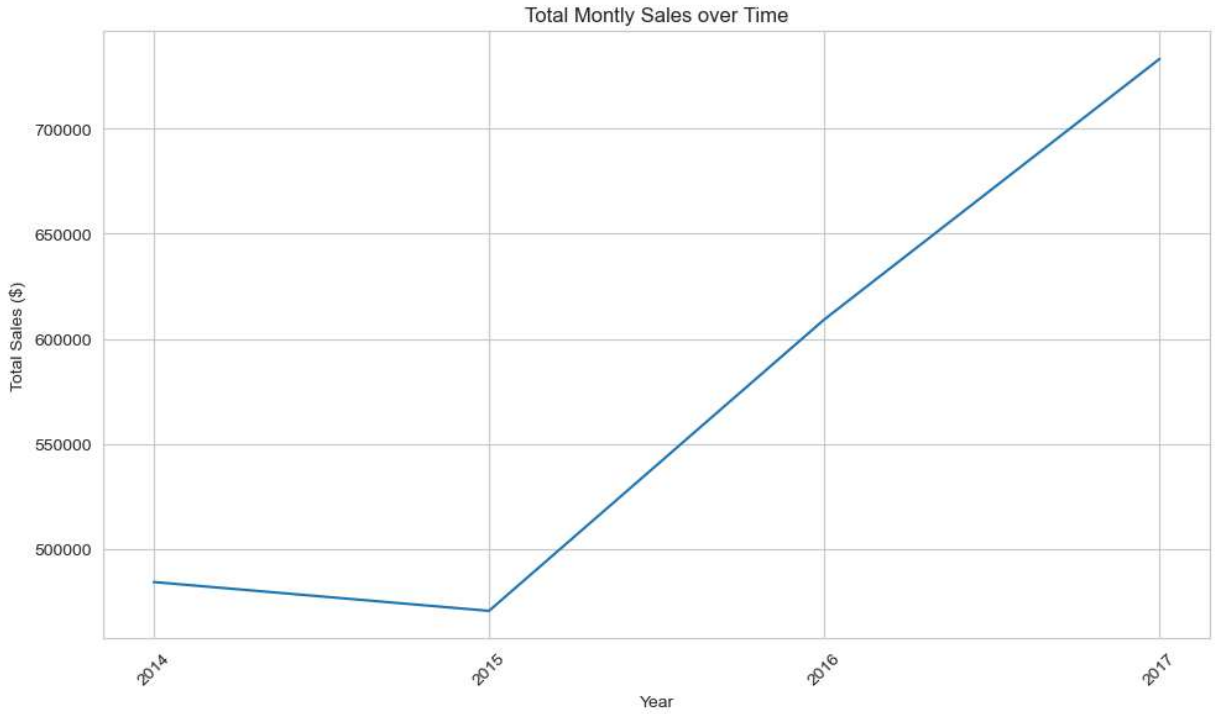
In [76]:
```
# Plot 2: Bar plot of Profit by Customer Segment
plt.figure(figsize=(8, 6))
sns.barplot(
    x=seg_summ.index,
    y=seg_summ['TotalProfit'],
    hue=seg_summ.index,
    palette='magma',
    legend=False
)
plt.title('Total Profit by Customer Segment')
plt.xlabel('Customer Segment')
plt.ylabel('Total Profit ($)')
plt.tight_layout()
plt.savefig('profit_by_segment.png')
```



In [78]:
```
# Plot Month Sales over Time
plt.figure(figsize=(12, 6))

# Plotting as per Month
ax=sns.lineplot(x='YearMonth',  y='Sales', data=monthly_sales)

plt.title('Total Montly Sales over Time')
plt.xlabel('Month-Year')
plt.ylabel('Total Sales ($)')
ax.set_xticks(ax.get_xticks()[::6])
plt.xticks(rotation=45)
plt.savefig('monthly_sales_trend.png')
```



In [80]:
```
# Plot Year Sales over Time
plt.figure(figsize=(10, 6))

# Plotting as per Year
sns.lineplot(x='Year',  y='Sales', data=yearly_sales)
plt.title('Total Montly Sales over Time')
plt.xlabel('Year')
plt.ylabel('Total Sales ($)')
ax.set_xticks(ax.get_xticks()[::6])
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig('yearly_sales_trend.png')
```



Machine Learning Model

1. Linear Regression Model - (Predicting Profit amount)

In [82]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score

print('Library Imported')
```

```
Library Imported
```

In [83]:
```
# New DataFrame 'X_encoded' by conveting the 'State' column into dummy variables.
X_encoded = pd.get_dummies(superstore_df['State'], prefix='State', drop_first=True)
```

In [84]:
```
# Prepare the data for the model
# Add the numerical features ('Quantity' and 'Discount') back to the encoded DataFr
X_encoded['Quantity'] = superstore_df['Quantity']
X_encoded['Discount'] = superstore_df['Discount']

target = 'Profit'
y = superstore_df[target] #defining target variable 'y'
```

In [85]:
```
# Split the data into Training and Testing Sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, ra

# Create and trai the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make prediction on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mae = metrics.mean_absolute_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2_score = metrics.r2_score(y_test, y_pred)

print("\nMachine Learning Model Evaluation:")
print(f"Mean Absolute Error (MAE): {mae: 2f}")
print(f"Mean Squared Error (MSE): {mse: 2f}")
print(f"R squared (R2): {r2_score: 2f}")

# Analyze the Coefficients of the Model
print("\nModel Coefficients:")
for feature, coef in zip(X_encoded.columns, model.coef_):
    print(f"{feature}: {coef: 2f}")
```

```
Machine Learning Model Evaluation:
Mean Absolute Error (MAE):  71.438768
Mean Squared Error (MSE):  75835.261849
R squared (R2):  0.022846

Model Coefficients:
State_Arizona: -61.162452
State_Arkansas: -54.878272
State_California: -53.536440
State_Colorado: -84.481477
State_Connecticut: -58.469557
State_Delaware:  9.355604
State_District of Columbia:  16.602901
State_Florida: -53.892785
State_Georgia: -14.021721
State_Idaho: -38.961104
State_Illinois: -51.589207
State_Indiana: -48.617021
State_Iowa: -80.503100
State_Kansas: -67.218552
State_Kentucky: -24.128249
State_Louisiana: -53.176144
State_Maine: -54.884861
State_Maryland: -50.590978
State_Massachusetts: -47.151147
State_Michigan: -0.605374
State_Minnesota:  38.673933
State_Mississippi: -46.204574
State_Missouri:  4.150730
State_Montana:  55.226214
State_Nebraska: -41.289616
State_Nevada:  1.584227
State_New Hampshire: -39.741140
State_New Jersey: -24.745047
State_New Mexico: -68.183492
State_New York: -27.899953
State_North Carolina: -69.299697
State_North Dakota: -78.128889
State_Ohio: -79.041939
State_Oklahoma: -33.867546
State_Oregon: -54.640269
State_Pennsylvania: -65.614673
State_Rhode Island:  36.974066
State_South Carolina: -60.010567
State_South Dakota: -72.257516
State_Tennessee: -79.988820
State_Texas: -56.118413
State_Utah: -41.742106
State_Vermont:  77.333589
State_Virginia: -15.390163
State_Washington: -36.785236
State_West Virginia: -50.378517
State_Wisconsin: -39.448854
State_Wyoming: -0.000000
Quantity:  7.708052
Discount: -210.669673
```

2. Logistic Regression Model - (Predicting High/Low Profit class)

1. Feature Engineering (Creating the Target)

In [86]:
```
# Create the Target Variable (High/Low Profit)
median_profit = superstore_df['Profit'].median()

# Create the new Target column 'Profit_Class'
# 1 = High Profit (Profit > Median)
# 0 = Low Profit (Profit <= Median)
superstore_df['Profit_Class'] = np.where(superstore_df['Profit'] > median_profit, 1

print("Median Profit used as threshold:", median_profit)
print("\nNew Target Variable Balance:")
print(superstore_df['Profit_Class'].value_counts())
```

```
Median Profit used as threshold: 8.64135

New Target Variable Balance:
Profit_Class
1    4993
0    4993
Name: count, dtype: int64
```

2. Data Splitting

In [87]:
```
# Define new target 'y'
y_class = superstore_df['Profit_Class']

# Split the data
X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(
    X_encoded, y_class, test_size=0.2, random_state=42
)
```

3. Model Training and Evaluation

In [88]:
```
# Create and train the Logistic Regression model
clf = LogisticRegression(max_iter=1000)
clf.fit(X_train_class, y_train_class)

y_pred_class = clf.predict(X_test_class)

# Evaluate the model
accuracy = accuracy_score(y_test_class, y_pred_class)

print("\n--- Logistic Regression Classification Model ---")
```

```
print(f"Accuracy: {accuracy: 2f}")
print("\nClassification Report (High/Low Profit):")
print(classification_report(y_test_class, y_pred_class))
```

```
--- Logistic Regression Classification Model ---
Accuracy:  0.696697

Classification Report (High/Low Profit):
              precision    recall  f1-score   support

           0       0.72      0.64      0.68      1005
           1       0.67      0.75      0.71       993

    accuracy                           0.70      1998
   macro avg       0.70      0.70      0.70      1998
weighted avg       0.70      0.70      0.70      1998
```

Imports to SQL

In [89]:
```
# Prepare the Returns Table

print("--- Creating the Correct RETURNS Lookup Table (Target: ~296 Rows) ---")

# Filter the main data from 'Returned' column
returns_lookup_df = superstore_df[superstore_df['Returned'] == 'Yes'].copy()

# Select only the unique columns and remove duplicates
returns_lookup_df = returns_lookup_df[['Order ID', 'Returned']].drop_duplicates()

# 3. Reset the index
returns_lookup_df.reset_index(drop=True, inplace=True)

print("Returns Table Corrected.")
print(f"Returns Table Rows: {returns_lookup_df.shape[0]}")
print("First 5 rows of the clean Returns table:")
print(returns_lookup_df.head())
```

```
--- Creating the Correct RETURNS Lookup Table (Target: ~296 Rows) ---
Returns Table Corrected.
Returns Table Rows: 296
First 5 rows of the clean Returns table:
         Order ID Returned
0  CA-2014-100762      Yes
1  CA-2014-100867      Yes
2  CA-2014-102652      Yes
3  CA-2014-103373      Yes
4  CA-2014-103744      Yes
```

In [90]:
```
# Prepare the People Table
print("\n--- Preparing People Table ---")

people_df = superstore_df[['Person', 'Region']].drop_duplicates().reset_index(drop=
```

```
print(f"People Table Size: {people_df.shape}")
print(people_df.head())
```

```
--- Preparing People Table ---
People Table Size: (4, 2)
              Person   Region
0        Chuck Magee     East
1      Anna Andreadi     West
2  Cassandra Brandow    South
3     Kelly Williams  Central
```

In [91]:
```
from sqlalchemy import create_engine

print("--- Exporting Final Data to SQL ---")

# Create the database connection engine -- named : 'superstore_project.sqlite'
engine = create_engine('sqlite:///superstore_project.sqlite')

# Export the three tables
orders_data.to_sql('orders', con=engine, if_exists='replace', index=False)
returns_lookup_df.to_sql('returns', con=engine, if_exists='replace', index=False)
people_df.to_sql('people', con=engine, if_exists='replace', index=False)

print("\nSuccess: SQL Database is fully populated!")
```

```
--- Exporting Final Data to SQL ---

Success: SQL Database is fully populated!
```

In [92]:
```
returns_lookup_df
```

Out[92]:

	Order ID	Returned
0	CA-2014-100762	Yes
1	CA-2014-100867	Yes
2	CA-2014-102652	Yes
3	CA-2014-103373	Yes
4	CA-2014-103744	Yes
...	...	...
291	US-2017-136679	Yes
292	US-2017-147886	Yes
293	US-2017-147998	Yes
294	US-2017-151127	Yes
295	US-2017-155999	Yes

296 rows × 2 columns

In [ ]: