In [687]:

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

In [688]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from keras.models import Sequential
from keras.layers import LSTM, Dense
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder
```

In [689]:

```
df = pd.read_excel(r"C:\Users\lenovo\Downloads\Sample - Superstore.xls")
df.head()
```

Out[689]:

Segment	Country	City	State	Postal Code	Region	Product ID	Category	Sub-Category	
Consumer	United States	Henderson	Kentucky	42420	South	FUR-BO-10001798	Furniture	Bookcases	
Consumer	United States	Henderson	Kentucky	42420	South	FUR-CH-10000454	Furniture	Chairs	H Up
Corporate	United States	Los Angeles	California	90036	West	OFF-LA-10000240	Office Supplies	Labels	T
Consumer	United States	Fort Lauderdale	Florida	33311	South	FUR-TA-10000577	Furniture	Tables	S Re
Consumer	United States	Fort Lauderdale	Florida	33311	South	OFF-ST-10000760	Office Supplies	Storage	E 'N

In [690]:

```
df.tail()
```

Out[690]:

	Row ID	Order ID	Order Date	Ship Mode	Customer ID	Customer Name	Segment	Country	C
9989	9990	CA-2014-110422	2017-12-18	Second Class	TB-21400	Tom Boeckenhauer	Consumer	United States	Mi
9990	9991	CA-2017-121258	2017-12-19	Standard Class	DB-13060	Dave Brooks	Consumer	United States	Costa M
9991	9992	CA-2017-121258	2017-12-20	Standard Class	DB-13060	Dave Brooks	Consumer	United States	Costa M
9992	9993	CA-2017-121258	2017-12-21	Standard Class	DB-13060	Dave Brooks	Consumer	United States	Costa M
9993	9994	CA-2017-119914	2017-12-22	Second Class	CC-12220	Chris Cortes	Consumer	United States	Westmins

In [691]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9994 entries, 0 to 9993
Data columns (total 20 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 Mode      9994 non-null   object        
 4   Customer ID    9994 non-null   object        
 5   Customer Name  9994 non-null   object        
 6   Segment        9994 non-null   object        
 7   Country        9994 non-null   object        
 8   City           9994 non-null   object        
 9   State          9994 non-null   object        
 10  Postal Code    9994 non-null   int64         
 11  Region         9994 non-null   object        
 12  Product ID     9994 non-null   object        
 13  Category       9994 non-null   object        
 14  Sub-Category   9994 non-null   object        
 15  Product Name   9994 non-null   object        
 16  Sales          9994 non-null   float64       
 17  Quantity       9994 non-null   int64         
 18  Discount       9994 non-null   float64       
 19  Profit         9994 non-null   float64       
dtypes: datetime64[ns](1), float64(3), int64(3), object(13)
memory usage: 1.5+ MB
```

In [692]:

```
df.shape
```

Out[692]:

```
(9994, 20)
```

In [693]:

```
df.describe()
```

Out[693]:

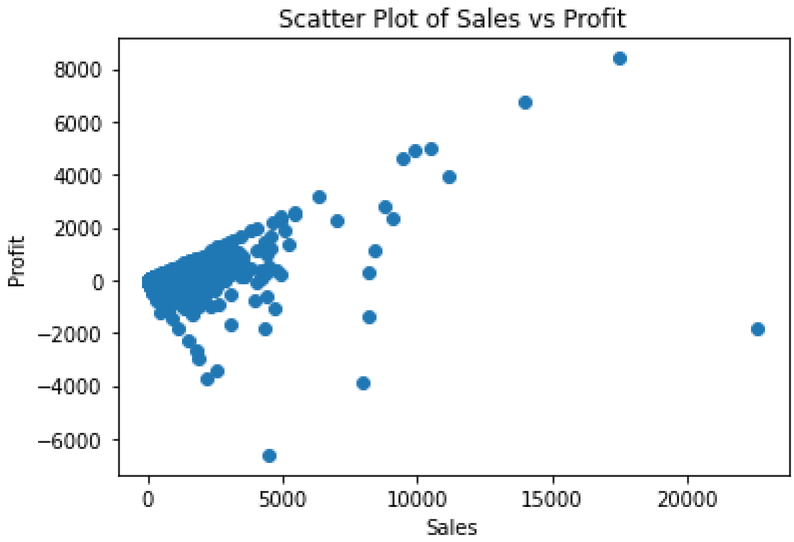
	Row ID	Postal Code	Sales	Quantity	Discount	Profit
count	9994.000000	9994.000000	9994.000000	9994.000000	9994.000000	9994.000000
mean	4997.500000	55190.379428	229.858001	3.789574	0.156203	28.656896
std	2885.163629	32063.693350	623.245101	2.225110	0.206452	234.260108
min	1.000000	1040.000000	0.444000	1.000000	0.000000	-6599.978000
25%	2499.250000	23223.000000	17.280000	2.000000	0.000000	1.728750
50%	4997.500000	56430.500000	54.490000	3.000000	0.200000	8.666500
75%	7495.750000	90008.000000	209.940000	5.000000	0.200000	29.364000
max	9994.000000	99301.000000	22638.480000	14.000000	0.800000	8399.976000

In [694]:

```
# Convert Order Date column to datetime
df['Order Date'] = pd.to_datetime(df['Order Date'])
```

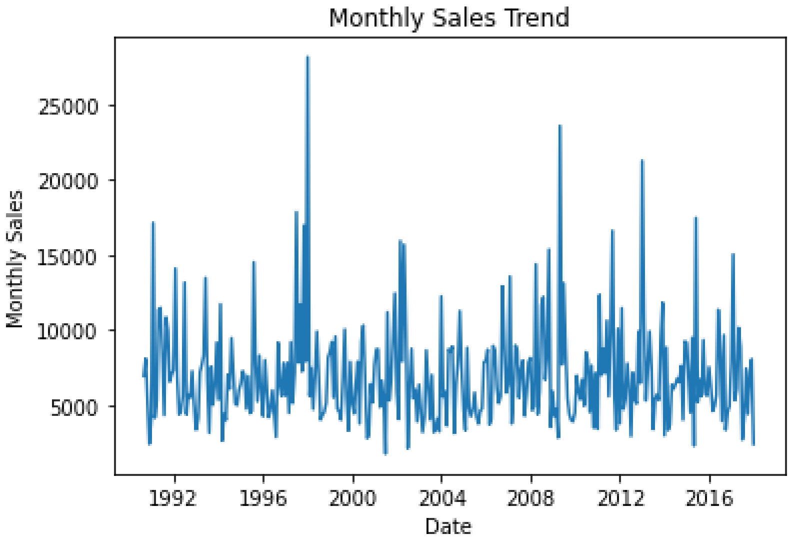
In [695]:

```
# Create a scatter plot of Sales vs Profit
plt.scatter(df['Sales'], df['Profit'])

# Set the x-axis label
plt.xlabel('Sales')

# Set the y-axis label
plt.ylabel('Profit')

# Set the title of the plot
plt.title('Scatter Plot of Sales vs Profit')

# Display the plot
plt.show()
```



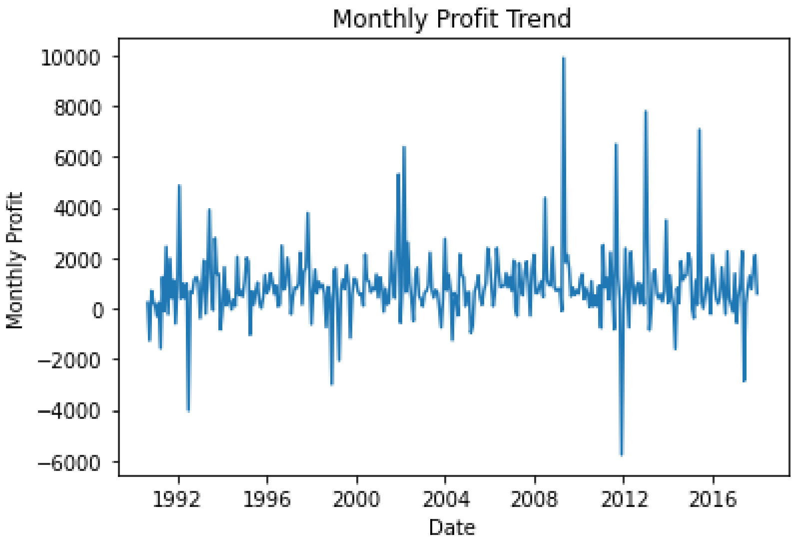
In [696]:

```
# Set the 'Order Date' column as the index of the DataFrame
df_time = df.set_index('Order Date')

# Calculate the monthly sum of sales using resample
monthly_sales = df_time['Sales'].resample('M').sum()

# Calculate the monthly sum of profit using resample
monthly_profit = df_time['Profit'].resample('M').sum()

# Plot the monthly sales trend
plt.plot(monthly_sales.index, monthly_sales.values)

# Set the x-axis label
plt.xlabel('Date')

# Set the y-axis label
plt.ylabel('Monthly Sales')

# Set the title of the plot
plt.title('Monthly Sales Trend')

# Display the plot
plt.show()
```



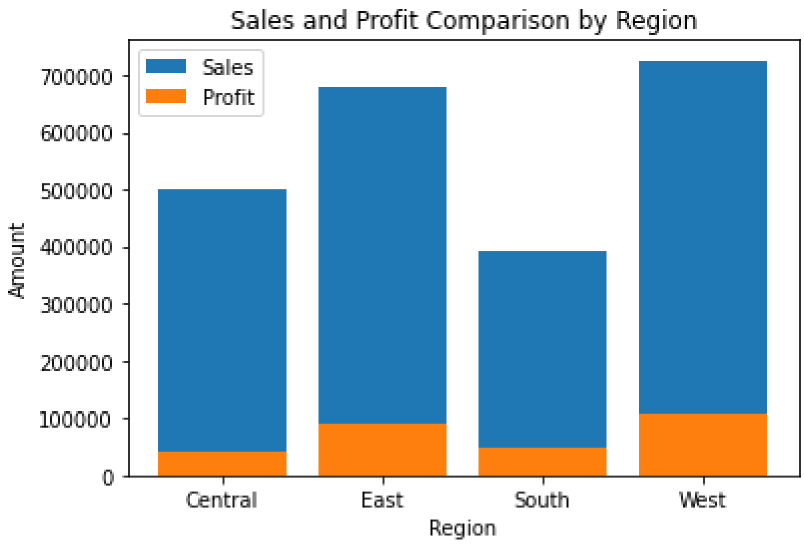
In [697]:

```
# Plot the monthly profit trend
plt.plot(monthly_profit.index, monthly_profit.values)

# Set the x-axis label
plt.xlabel('Date')

# Set the y-axis label
plt.ylabel('Monthly Profit')

# Set the title of the plot
plt.title('Monthly Profit Trend')

# Display the plot
plt.show()
```



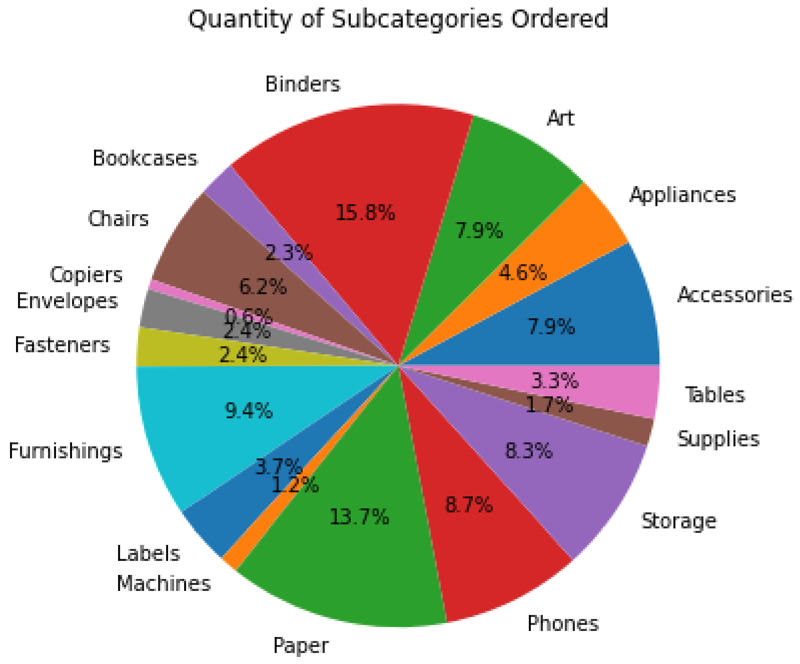
In [698]:

```
# Calculate the total sales by region using groupby
region_sales = df.groupby('Region')['Sales'].sum()

# Calculate the total profit by region using groupby
region_profit = df.groupby('Region')['Profit'].sum()

# Create a bar plot comparing sales and profit by region
plt.bar(region_sales.index, region_sales.values, label='Sales')
plt.bar(region_profit.index, region_profit.values, label='Profit')

# Set the x-axis label as 'Region'
plt.xlabel('Region')

# Set the y-axis label as 'Amount'
plt.ylabel('Amount')

# Set the title of the plot as 'Sales and Profit Comparison by Region'
plt.title('Sales and Profit Comparison by Region')

# Display a legend for the sales and profit bars
plt.legend()

# Display the plot
plt.show()
```



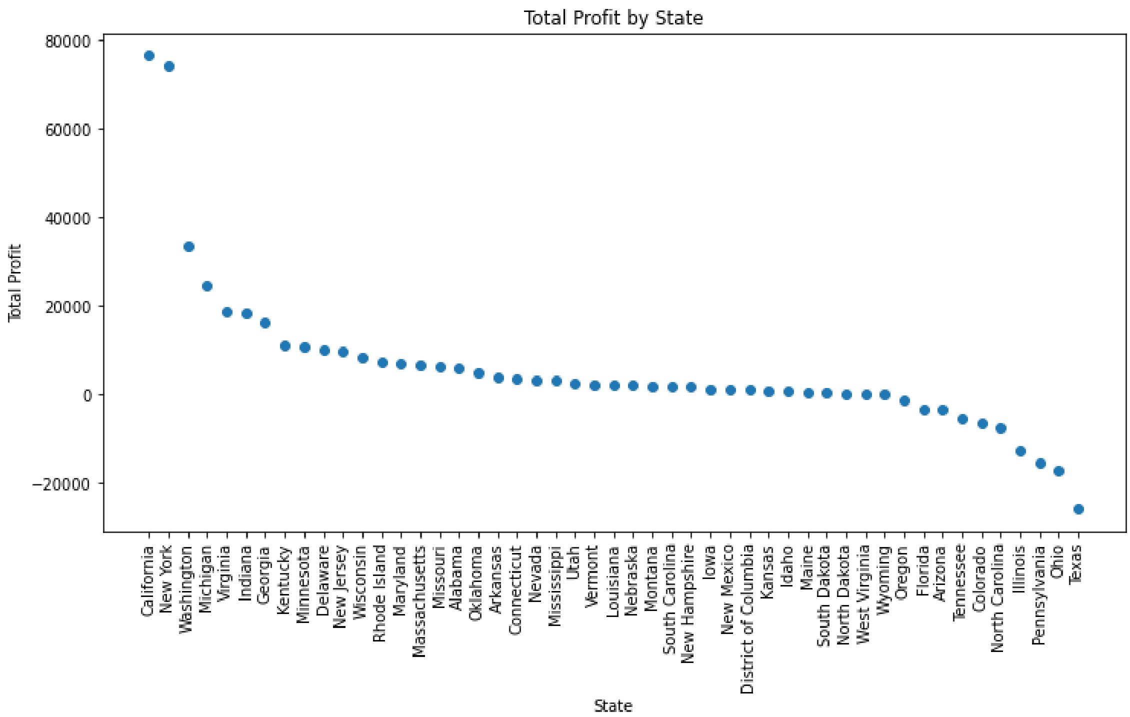
In [699]:

```
# Group the data by Region and Sub-Category and calculate the sum of Quantity
region_subcategory_quantity = df.groupby(['Sub-Category'])['Quantity'].sum()

# Create a new figure with adjusted size
fig = plt.figure(figsize=(6,6))

# Create the pie chart
plt.pie(region_subcategory_quantity, labels=region_subcategory_quantity.index, autopct='

# Set the title
plt.title('Quantity of Subcategories Ordered')

# Display the plot
plt.show()
```



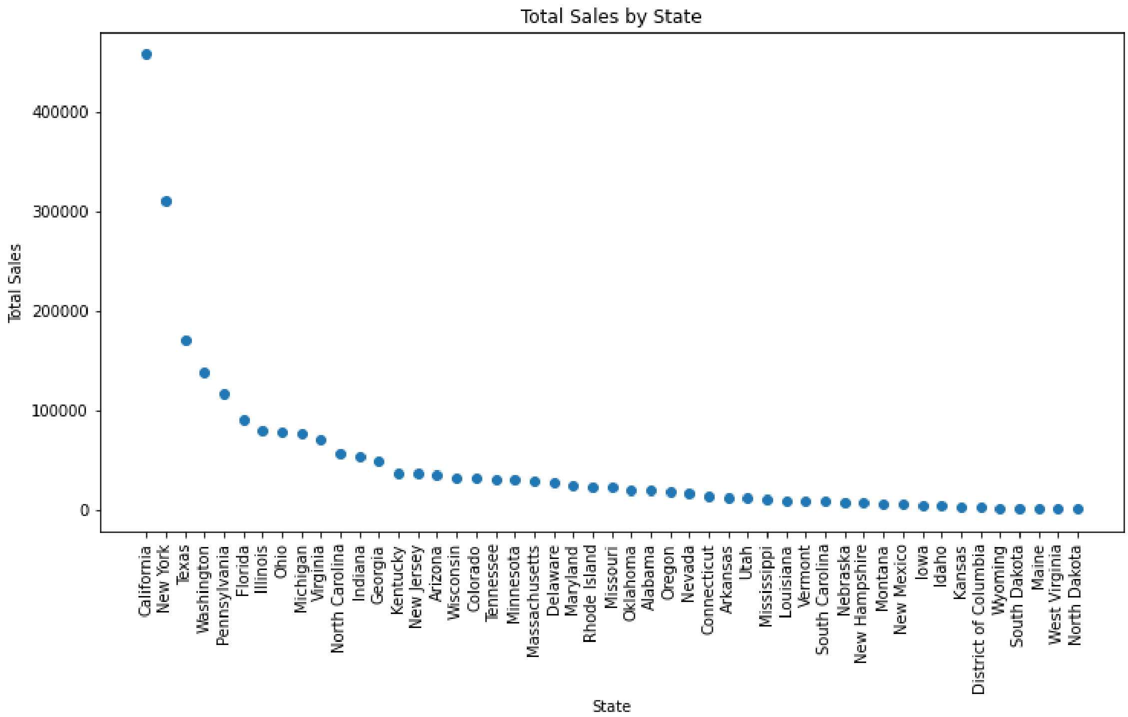
In [700]:

```
city_profit = df.groupby('State')['Profit'].sum()

# Sort the cities based on profit in descending order
city_profit = city_profit.sort_values(ascending=False)

# Create a scatter plot
plt.figure(figsize=(12, 6))
plt.scatter(city_profit.index, city_profit.values)
plt.xlabel('State')
plt.ylabel('Total Profit')
plt.title('Total Profit by State')
plt.xticks(rotation=90)
plt.show()
```



In [703]:

```
# Calculate the total sales by state using groupby
state_sales = df.groupby('State')['Sales'].sum()

# Sort the states based on sales in descending order
state_sales = state_sales.sort_values(ascending=False)

# Create a scatter plot of total sales by state
plt.figure(figsize=(12, 6))
plt.scatter(state_sales.index, state_sales.values)

# Set the x-axis label as 'State'
plt.xlabel('State')

# Set the y-axis label as 'Total Sales'
plt.ylabel('Total Sales')

# Set the title of the plot as 'Total Sales by State'
plt.title('Total Sales by State')

# Rotate the x-axis tick labels for better readability
plt.xticks(rotation=90)

# Display the plot
plt.show()
```



Linear Regression

In [704]:

```
# remove unnecessary columns
df.drop(['Row ID', 'Order ID', 'Customer ID', 'Customer Name', 'Postal Code', 'Product I
```

In [705]:

```
# convert categorical columns into numerical columns using one-hot encoding
cat_cols = ['Ship Mode', 'Segment', 'Country', 'City', 'State', 'Region', 'Category', 'S
df = pd.get_dummies(df, columns=cat_cols)
```

In [706]:

```
# split the data into training and testing sets
X = df.drop(['Sales', 'Profit'], axis=1)
y_sales = df['Sales']
y_profit = df['Profit']
X_train, X_test, y_sales_train, y_sales_test, y_profit_train, y_profit_test = train_test
```

In [707]:

```
# filter out the negative values from the 'Profit' column
df = df[df['Profit'] > 0]
```

In [708]:

```
# Convert Order Date column to numerical values
X_train['date_num'] = pd.to_numeric(X_train['Order Date'])
X_test['date_num'] = pd.to_numeric(X_test['Order Date'])

# Drop the original Order Date column
X_train.drop('Order Date', axis=1, inplace=True)
X_test.drop('Order Date', axis=1, inplace=True)
```

In [709]:

```
# create a linear regression model for Sales
sales_reg = LinearRegression()
sales_reg.fit(X_train, y_sales_train)
```

Out[709]:

```
▾ LinearRegression
LinearRegression()
```

In [710]:

```
# predict sales for test set
y_sales_pred = sales_reg.predict(X_test)
```

In [711]:

```
# create a new DataFrame to store the actual and predicted values
sales_df = pd.DataFrame({'Actual Sales': y_sales_test, 'Predicted Sales': y_sales_pred})

# print the first 10 rows of the DataFrame
print(sales_df)
```

```
      Actual Sales  Predicted Sales
3125       563.808       222.101235
1441        36.672       218.933302
4510        37.300       224.706690
39         212.058       216.295866
4509       171.288       224.704809
...            ...              ...
9956        46.350       234.951680
1561         2.780       219.159045
1670        16.680       219.364096
6951       479.988       229.298688
3910       352.450       223.577973

[1999 rows x 2 columns]
```

In [712]:

```
# create a linear regression model for Profit
profit_reg = LinearRegression()
profit_reg.fit(X_train, y_profit_train)
```

Out[712]:

```
▾ LinearRegression
LinearRegression()
```

In [713]:

```
# make predictions on the testing set
y_profit_pred = profit_reg.predict(X_test)
```

In [714]:

```
# create a new DataFrame to store the actual and predicted values
profit_df = pd.DataFrame({'Actual Profit': y_profit_test, 'Predicted Profit': y_profit_p

# print the first 10 rows of the DataFrame
print(profit_df)
```

```
      Actual Profit  Predicted Profit
3125        21.1428         28.638814
1441        11.4600         26.794847
4510        17.1580         30.155378
39         -15.1470         25.259667
4509        -6.4233         30.154283
...             ...               ...
9956        21.7845         36.118706
1561         0.7228         26.926246
1670         5.2125         27.045600
6951        55.9986         32.828254
3910      -211.4700         29.498383

[1999 rows x 2 columns]
```

In [715]:

```
# calculate MSE for sales prediction
mse_sales = mean_squared_error(y_sales_test, y_sales_pred)
print("MSE for sales prediction:", mse_sales)
```

```
MSE for sales prediction: 591455.7154986125
```

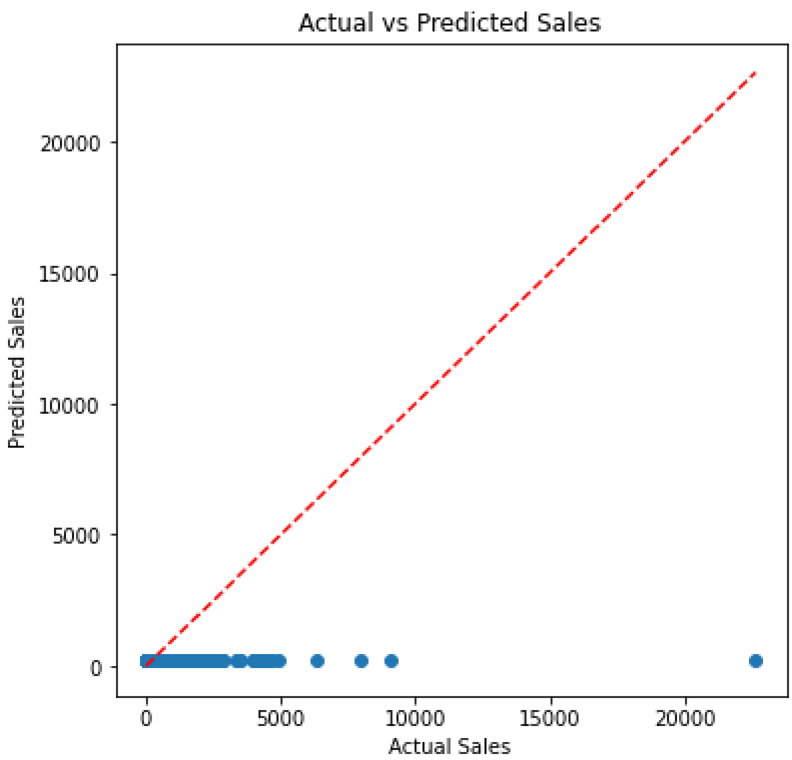
In [716]:

```
# calculate MSE for profit prediction
mse_profit = mean_squared_error(y_profit_test, y_profit_pred)
print("MSE for profit prediction:", mse_profit)
```

```
MSE for profit prediction: 48586.25769997954
```

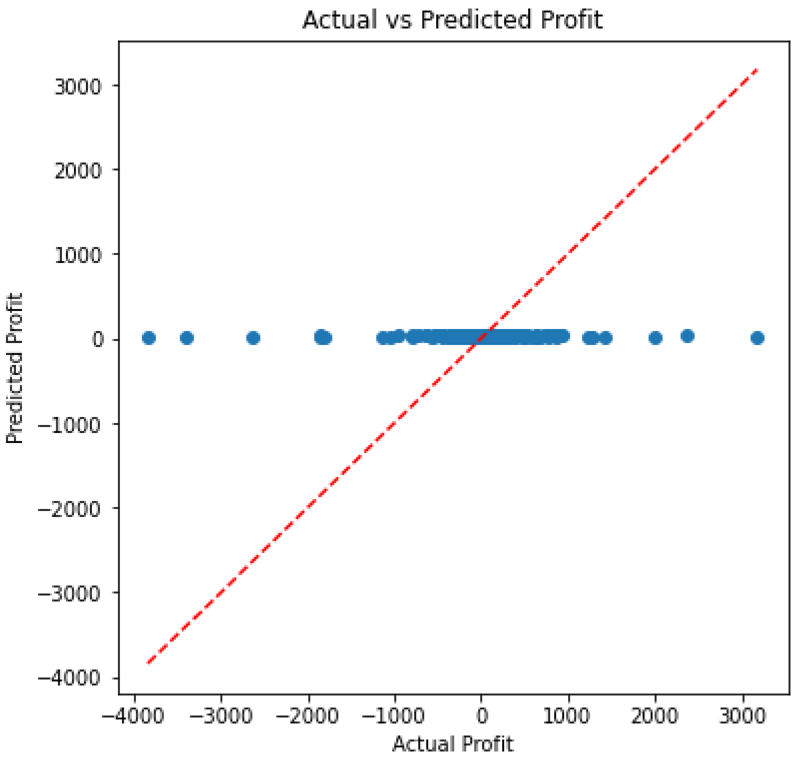
In [717]:

```
# Create a scatter plot for actual vs predicted sales
plt.figure(figsize=(6, 6))
plt.scatter(y_sales_test, y_sales_pred)

# Add a red dashed line to represent perfect prediction
plt.plot([y_sales_test.min(), y_sales_test.max()], [y_sales_test.min(), y_sales_test.max

# Set the x-axis label as 'Actual Sales'
plt.xlabel('Actual Sales')

# Set the y-axis label as 'Predicted Sales'
plt.ylabel('Predicted Sales')

# Set the title of the plot as 'Actual vs Predicted Sales'
plt.title('Actual vs Predicted Sales')

# Display the plot
plt.show()
```



In [718]:

```
# Create a scatter plot for actual vs predicted profit
plt.figure(figsize=(6, 6))
plt.scatter(y_profit_test, y_profit_pred)

# Add a red dashed line to represent perfect prediction
plt.plot([y_profit_test.min(), y_profit_test.max()], [y_profit_test.min(), y_profit_test

# Set the x-axis label as 'Actual Profit'
plt.xlabel('Actual Profit')

# Set the y-axis label as 'Predicted Profit'
plt.ylabel('Predicted Profit')

# Set the title of the plot as 'Actual vs Predicted Profit'
plt.title('Actual vs Predicted Profit')

# Display the plot
plt.show()
```



In [719]:

```
# Calculate the minimum and maximum profit values
min_profit = df['Profit'].min()
max_profit = df['Profit'].max()

# Calculate the range of profit values
profit_range = max_profit - min_profit

# Print the range of profit values
print(profit_range)
```

```
8399.9132
```

Support Vector Regressor

In [720]:

```
# Filter the DataFrame to include only floating-point columns
df_float = df.select_dtypes(include=['float64'])
```

In [721]:

```
# Extract features (input variables) and target variables (sales and profit)
features = df_float.drop(['Sales', 'Profit'], axis=1)  # Exclude the 'Sales' and 'Profit
target_sales = df_float['Sales']
target_profit = df_float['Profit']
```

In [722]:

```
# Split the data into training and testing sets
X_train, X_test, y_sales_train, y_sales_test, y_profit_train, y_profit_test = train_test
    features, target_sales, target_profit, test_size=0.2, random_state=42
)
```

In [723]:

```
# Build the SVM model for sales
svm_sales = SVR(kernel='linear')
svm_sales.fit(X_train, y_sales_train)
```

Out[723]:

```
▾          SVR
SVR(kernel='linear')
```

In [724]:

```
y_sales_predict = svm_sales.predict(X_test)
y_sales_predict
```

Out[724]:

```
array([49.79, 52.07, 52.07, ..., 49.79, 52.07, 49.79])
```

In [725]:

```
data = pd.DataFrame({
    'Sales actual Values': y_sales_test,
    'Sales predicted Values': y_sales_predict
})
data
```

Out[725]:

	Sales actual Values	Sales predicted Values
4715	201.584	49.79
5911	46.530	52.07
2505	8187.650	52.07
7259	45.360	52.07
4110	160.980	52.07
...	...	...
5380	105.552	49.79
6171	3.520	52.07
7767	7.152	49.79
7665	30.400	52.07
5065	383.976	49.79

1612 rows × 2 columns

In [726]:

```
# Build the SVM model for profit
svm_profit = SVR(kernel='linear')
svm_profit.fit(X_train, y_profit_train)
```

Out[726]:

```
SVR
SVR(kernel='linear')
```

In [727]:

```
y_profits_predict = svm_profit.predict(X_test)
y_profits_predict
```

Out[727]:

```
array([10.54699989, 15.526     , 15.526     , ..., 10.54699989,
       15.526     , 10.54699989])
```

In [728]:

```
data = pd.DataFrame({
    'Profit Actual Values': y_profit_test,
    'Profit Predicted Values': y_profits_predict
})
data
```

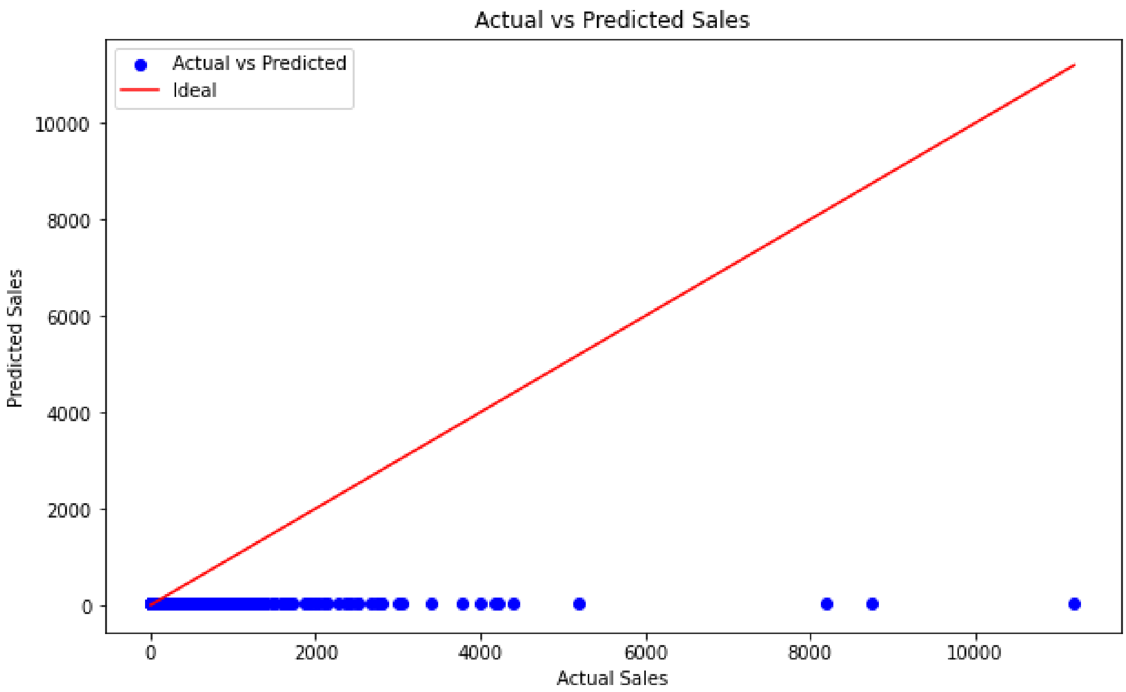
Out[728]:

	Profit Actual Values	Profit Predicted Values
4715	20.1584	10.547
5911	13.0284	15.526
2505	327.5060	15.526
7259	21.7728	15.526
4110	20.9274	15.526
...	...	...
5380	35.6238	10.547
6171	1.0208	15.526
7767	0.7152	10.547
7665	13.9840	15.526
5065	81.5949	10.547

1612 rows × 2 columns

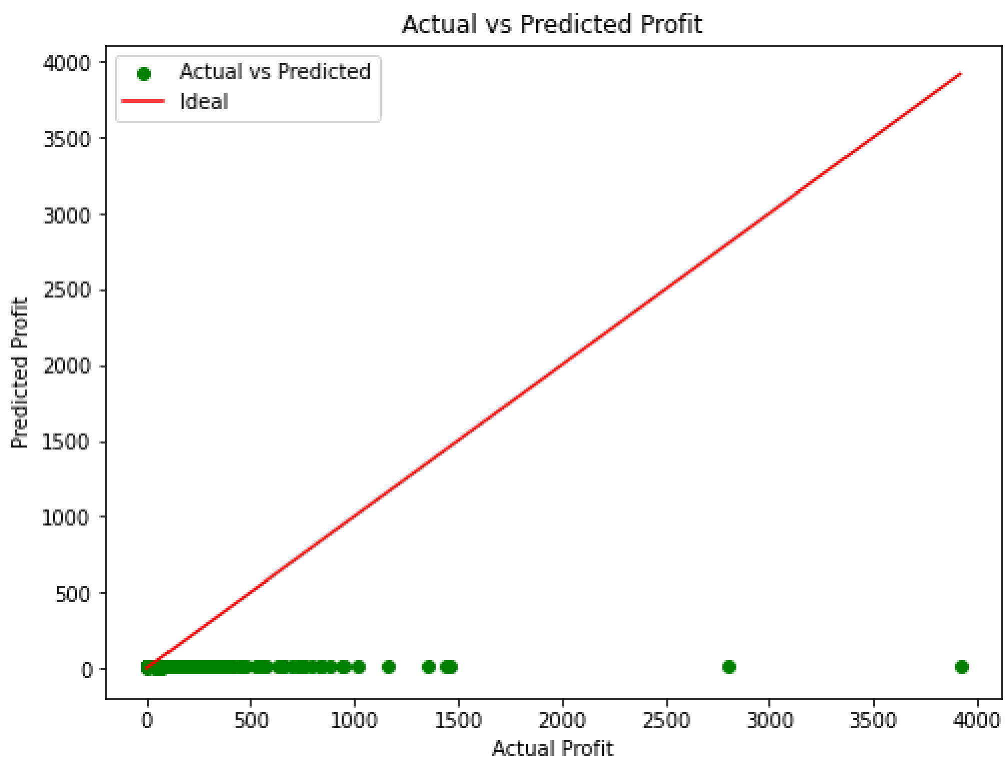
In [730]:

```
# Create a scatter plot for actual vs predicted sales
plt.figure(figsize=(10, 6))
plt.scatter(y_sales_test, y_sales_predict, color='blue', label='Actual vs Predicted')

# Add a red line to represent the ideal prediction
plt.plot([min(y_sales_test), max(y_sales_test)], [min(y_sales_test), max(y_sales_test)],

# Set the x-axis label as 'Actual Sales'
plt.xlabel('Actual Sales')

# Set the y-axis label as 'Predicted Sales'
plt.ylabel('Predicted Sales')

# Set the title of the plot as 'Actual vs Predicted Sales'
plt.title('Actual vs Predicted Sales')

# Add a legend to the plot
plt.legend()

# Display the plot
plt.show()
```



In [731]:

```
# Create a scatter plot for actual vs predicted profit
plt.figure(figsize=(8, 6))
plt.scatter(y_profit_test, y_profits_predict, color='green', label='Actual vs Predicted'

# Add a red line to represent the ideal prediction
plt.plot([min(y_profit_test), max(y_profit_test)], [min(y_profit_test), max(y_profit_tes

# Set the x-axis label as 'Actual Profit'
plt.xlabel('Actual Profit')

# Set the y-axis label as 'Predicted Profit'
plt.ylabel('Predicted Profit')

# Set the title of the plot as 'Actual vs Predicted Profit'
plt.title('Actual vs Predicted Profit')

# Add a legend to the plot
plt.legend()

# Display the plot
plt.show()
```



In [732]:

```
# Calculate the mean squared error for sales
sales_mse = mean_squared_error(y_sales_test, y_sales_predict)

# Print the mean squared error for sales and profit
print('Mean Squared Error for Sales:', sales_mse)
```

```
Mean Squared Error for Sales: 398683.5967014188
```

In [733]:

```
# Calculate the mean squared error for profit
profit_mse = mean_squared_error(y_profit_test, y_profits_predict)

print('Mean Squared Error for Profit:', profit_mse)
```

```
Mean Squared Error for Profit: 29972.93206388223
```

Feedforward Neural Network Model

In [734]:

```
# Select the relevant features and target variables
features = df[['Sales', 'Quantity', 'Discount']]  # Adjust the features accordingly
target_sales = df['Sales']
target_profit = df['Profit']
```

In [735]:

```
# Split the data into training and testing sets
X_train, X_test, y_sales_train, y_sales_test, y_profit_train, y_profit_test = train_test
    features, target_sales, target_profit, test_size=0.2, random_state=42
)
```

In [738]:

```
# Scale the numerical features

# Convert categorical variables into numerical representations
# If you have categorical columns, apply label encoding or one-hot encoding

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [674]:

```
# Construct the deep learning model
model = Sequential()
model.add(Dense(64, activation='relu', input_dim=X_train_scaled.shape[1]))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='linear'))
```

In [675]:

```
# Compile the model
model.compile(loss='mean_squared_error', optimizer='adam')
```

In [676]:

```
# Train the model for sales prediction
model.fit(X_train_scaled, y_sales_train, epochs=10, batch_size=32)
```

```
Epoch 1/10
202/202 [==============================] - 3s 4ms/step - loss: 401977.781
2
Epoch 2/10
202/202 [==============================] - 1s 4ms/step - loss: 360333.187
5
Epoch 3/10
202/202 [==============================] - 1s 4ms/step - loss: 348521.812
5
Epoch 4/10
202/202 [==============================] - 1s 4ms/step - loss: 345487.718
8
Epoch 5/10
202/202 [==============================] - 1s 4ms/step - loss: 342351.000
0
Epoch 6/10
202/202 [==============================] - 1s 4ms/step - loss: 338988.156
2
Epoch 7/10
202/202 [==============================] - 1s 4ms/step - loss: 335310.562
5
Epoch 8/10
202/202 [==============================] - 1s 4ms/step - loss: 331309.125
0
Epoch 9/10
202/202 [==============================] - 1s 4ms/step - loss: 326629.250
0
Epoch 10/10
202/202 [==============================] - 1s 5ms/step - loss: 321920.156
2
```

Out[676]:

```
<keras.callbacks.History at 0x1c4a1f6cd30>
```

In [677]:

```
# Make predictions for sales
sales_predictions = model.predict(X_test_scaled)
```

```
51/51 [==============================] - 0s 4ms/step
```

In [678]:

```
sales_predictions
```

Out[678]:

```
array([[151.3274 ],
       [209.87035],
       [604.0637 ],
       ...,
       [187.45558],
       [122.32873],
       [201.67761]], dtype=float32)
```

In [679]:

```
# Train the model for profit prediction
model.fit(X_train_scaled, y_profit_train, epochs=10, batch_size=32)
```

```
Epoch 1/10
202/202 [==============================] - 1s 6ms/step - loss: 51922.4805
Epoch 2/10
202/202 [==============================] - 2s 9ms/step - loss: 42813.8867
Epoch 3/10
202/202 [==============================] - 2s 11ms/step - loss: 42086.210
9
Epoch 4/10
202/202 [==============================] - 3s 13ms/step - loss: 41518.171
9
Epoch 5/10
202/202 [==============================] - 1s 5ms/step - loss: 40951.9180
Epoch 6/10
202/202 [==============================] - 1s 6ms/step - loss: 40198.8711
Epoch 7/10
202/202 [==============================] - 1s 6ms/step - loss: 39403.6289
Epoch 8/10
202/202 [==============================] - 2s 8ms/step - loss: 38432.2344
Epoch 9/10
202/202 [==============================] - 1s 6ms/step - loss: 37366.1602
Epoch 10/10
202/202 [==============================] - 1s 6ms/step - loss: 36081.3750
```

Out[679]:

```
<keras.callbacks.History at 0x1c4a20c3eb0>
```

In [680]:

```
# Make predictions for profit
profit_predictions = model.predict(X_test_scaled)
```

```
51/51 [==============================] - 1s 5ms/step
```

In [681]:

```
profit_predictions
```

Out[681]:

```
array([[ 15.14651  ],
       [ 35.900803 ],
       [638.9532   ],
       ...,
       [ 16.186436 ],
       [  7.9009047],
       [ 25.386944 ]], dtype=float32)
```

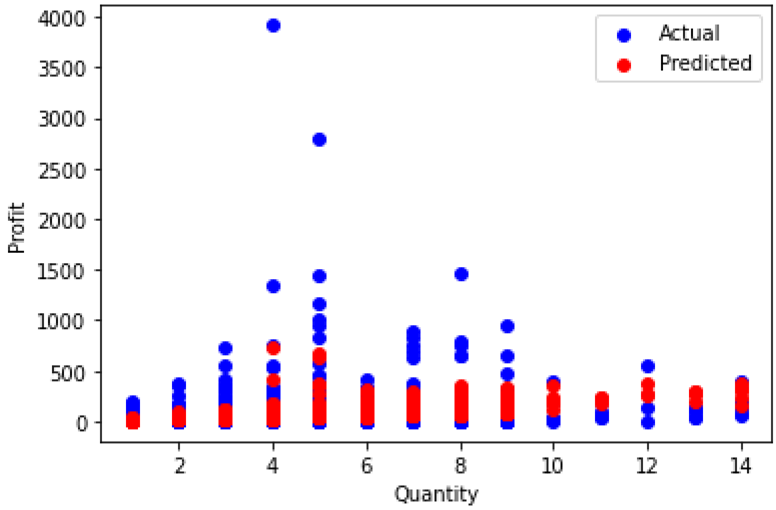
In [682]:

```
# Calculate MSE for sales predictions
sales_mse = mean_squared_error(y_sales_test, sales_predictions)
print("MSE for sales predictions:", sales_mse)

# Calculate MSE for profit predictions
profit_mse = mean_squared_error(y_profit_test, profit_predictions)
print("MSE for profit predictions:", profit_mse)
```

```
MSE for sales predictions: 327821.3735818017
MSE for profit predictions: 18317.05024360379
```

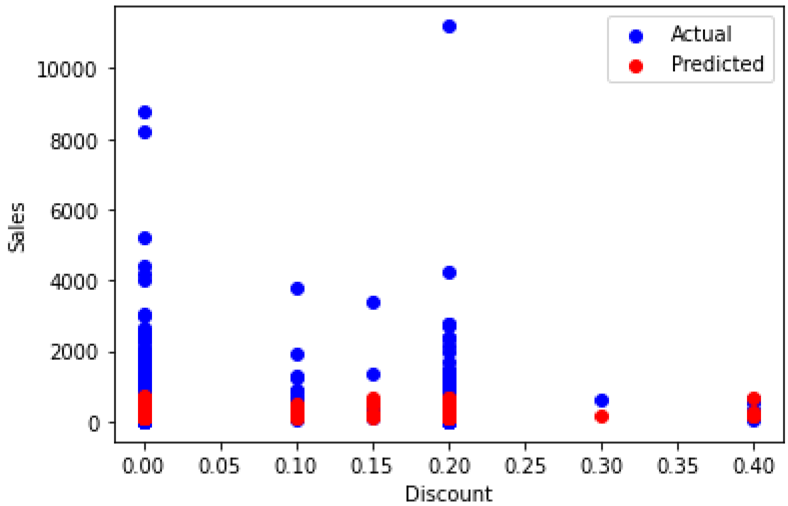
In [683]:

```
# Plot the scatter plot for Quantity vs. Profit
plt.scatter(X_test['Quantity'], y_profit_test, color='blue', label='Actual')
plt.scatter(X_test['Quantity'], profit_predictions, color='red', label='Predicted')
plt.xlabel('Quantity')
plt.ylabel('Profit')
plt.legend()
plt.show()
```



In [684]:

```
# Plot the scatter plot for Discount vs. Sales
plt.scatter(X_test['Discount'], y_sales_test, color='blue', label='Actual')
plt.scatter(X_test['Discount'], sales_predictions, color='red', label='Predicted')
plt.xlabel('Discount')
plt.ylabel('Sales')
plt.legend()
plt.show()
```



In [ ]: