phase3

October 15, 2023

[4]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

[7]:
```
df = pd.read_csv('C:/Users/Senthil/Documents/product/statsfinal.csv')  #␣
 ↪Replace 'your_file_path.csv' with the actual file path

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

[7]:

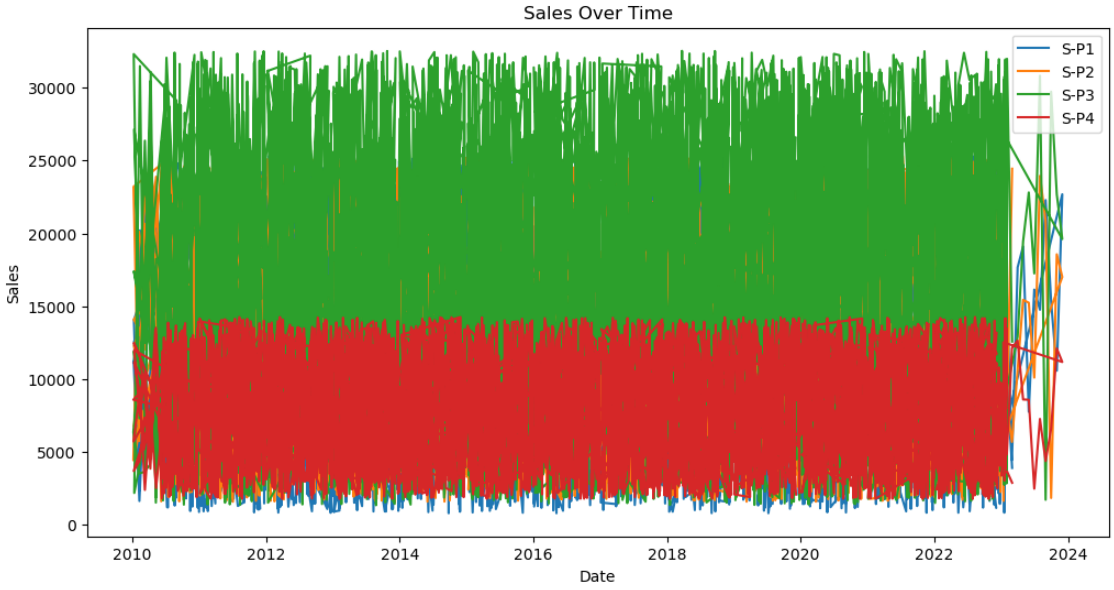
	Unnamed: 0	Date	Q-P1	Q-P2	Q-P3	Q-P4	S-P1	S-P2	S-P3	S-P4
0	0	13-06-2010	5422	3725	576	907	17187.74	23616.50	3121.92	6466.91
1	1	14-06-2010	7047	779	3578	1574	22338.99	4938.86	19392.76	11222.62
2	2	15-06-2010	1572	2082	595	1145	4983.24	13199.88	3224.90	8163.85
3	3	16-06-2010	5657	2399	3140	1672	17932.69	15209.66	17018.80	11921.36
4	4	17-06-2010	3668	3207	2184	708	11627.56	20332.38	11837.28	5048.04

[11]:
```
def validate_date(date_string):
    try:
        return pd.to_datetime(date_string)
    except ValueError:
        return pd.NaT  # Returns a 'Not a Time' placeholder for invalid dates

df['Date'] = df['Date'].apply(validate_date)
df = df.dropna(subset=['Date'])  # Remove rows with invalid dates
df.set_index('Date', inplace=True)
```

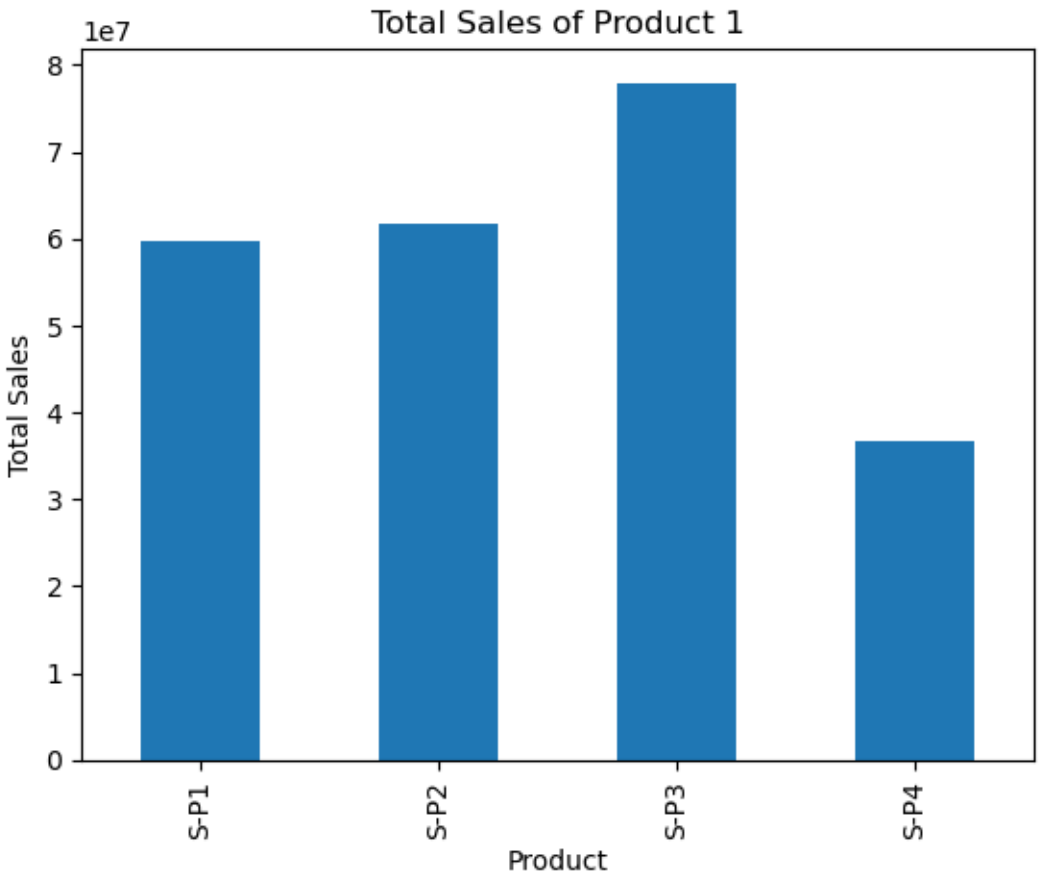
```
C:\Users\Senthil\AppData\Local\Temp\ipykernel_3676\565022283.py:3: UserWarning:
Parsing dates in DD/MM/YYYY format when dayfirst=False (the default) was
specified. This may lead to inconsistently parsed dates! Specify a format to
ensure consistent parsing.
  return pd.to_datetime(date_string)
```

[12]:
```
# Summary Statistics
print(df.describe())

# Visualize Sales over Time
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['S-P1'], label='S-P1')
plt.plot(df.index, df['S-P2'], label='S-P2')
plt.plot(df.index, df['S-P3'], label='S-P3')
plt.plot(df.index, df['S-P4'], label='S-P4')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.title('Sales Over Time')
plt.legend()
plt.show()
```

```
       Unnamed: 0         Q-P1         Q-P2         Q-P3         Q-P4  \
count  4574.000000  4574.000000  4574.000000  4574.000000  4574.000000
mean   2299.372759  4123.342589  2129.705072  3143.769786  1123.738303
std    1327.857219  2243.691134  1089.503315  1671.052866   497.813557
min       0.000000   254.000000   251.000000   250.000000   250.000000
25%    1149.250000  2149.500000  1167.250000  1695.250000   696.000000
50%    2299.500000  4138.000000  2133.500000  3196.500000  1137.000000
75%    3449.750000  6072.000000  3069.750000  4564.750000  1545.750000
max    4599.000000  7998.000000  3998.000000  6000.000000  2000.000000

              S-P1          S-P2          S-P3          S-P4
count  4574.000000   4574.000000   4574.000000   4574.000000
mean  13070.996006  13502.330157  17039.232239   8012.254104
std    7112.500894   6907.451018   9057.106532   3549.410662
min     805.180000   1591.340000   1355.000000   1782.500000
25%    6813.915000   7400.365000   9188.255000   4962.480000
50%   13117.460000  13526.390000  17325.030000   8106.810000
75%   19248.240000  19462.215000  24740.945000  11021.197500
max   25353.660000  25347.320000  32520.000000  14260.000000
```



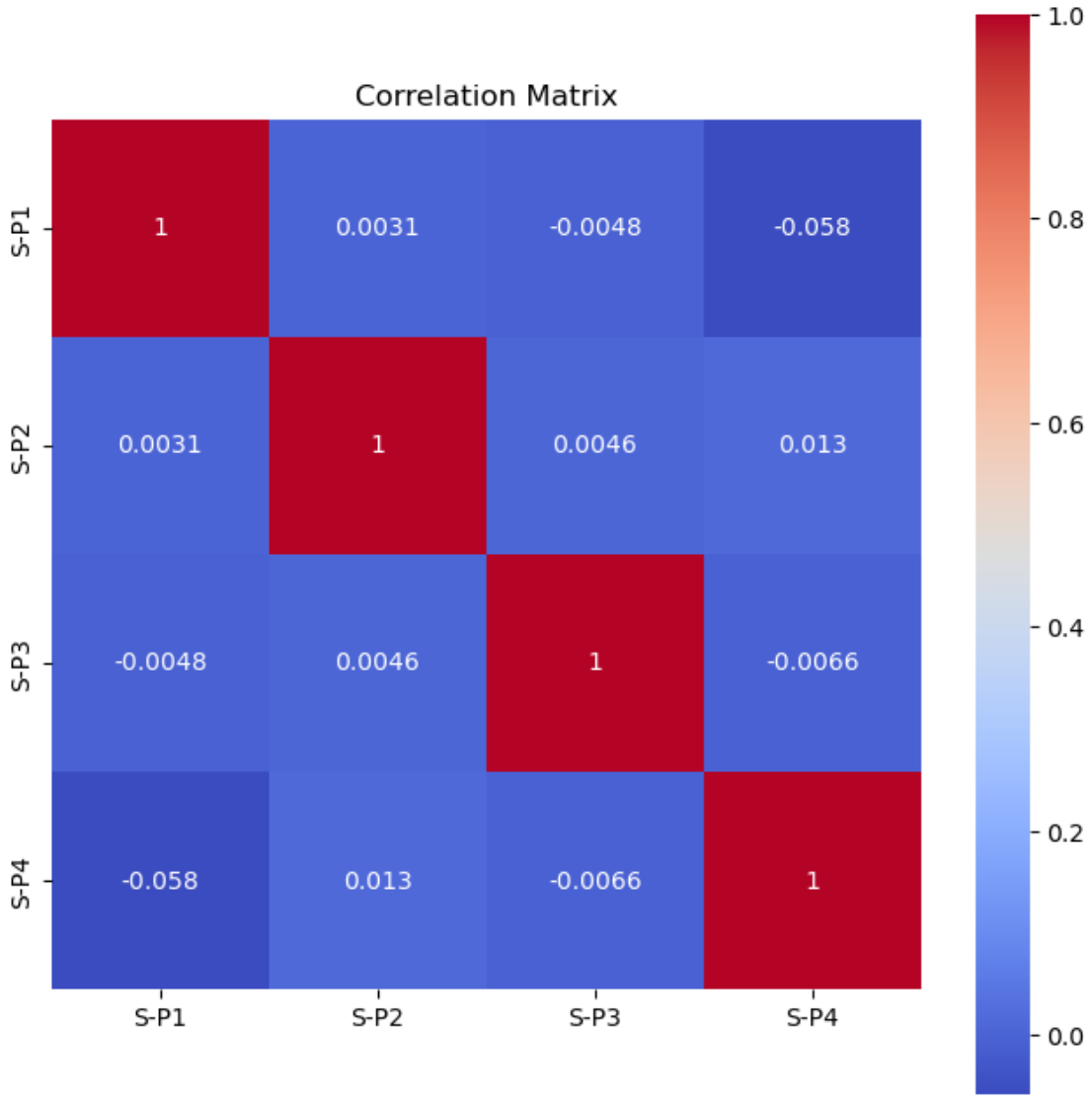
[13]:
```
# Compare Sales of Product 1
product_1_sales = df[['S-P1', 'S-P2', 'S-P3', 'S-P4']].sum()
product_1_sales.plot(kind='bar')
plt.xlabel('Product')
plt.ylabel('Total Sales')
plt.title('Total Sales of Product 1')
plt.show()
```



[ ]:

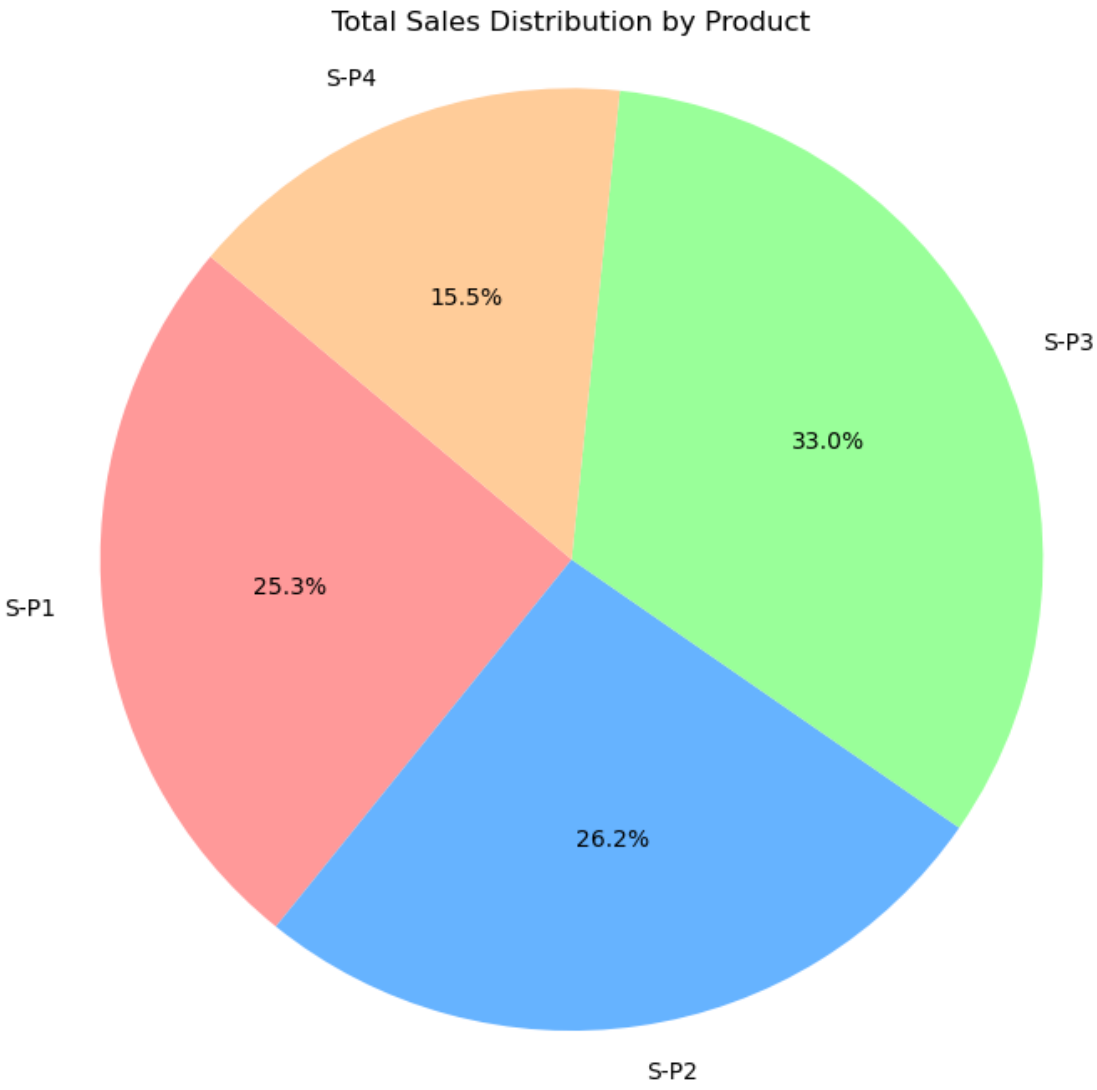
[16]:
```
import seaborn as sns
```

[17]:
```
# Calculate the correlation matrix
correlation_matrix = df[['S-P1', 'S-P2', 'S-P3', 'S-P4']].corr()

# Plot the correlation matrix as a heatmap
plt.figure(figsize=(8, 8))
plt.title('Correlation Matrix')
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', square=True)
plt.show()
```

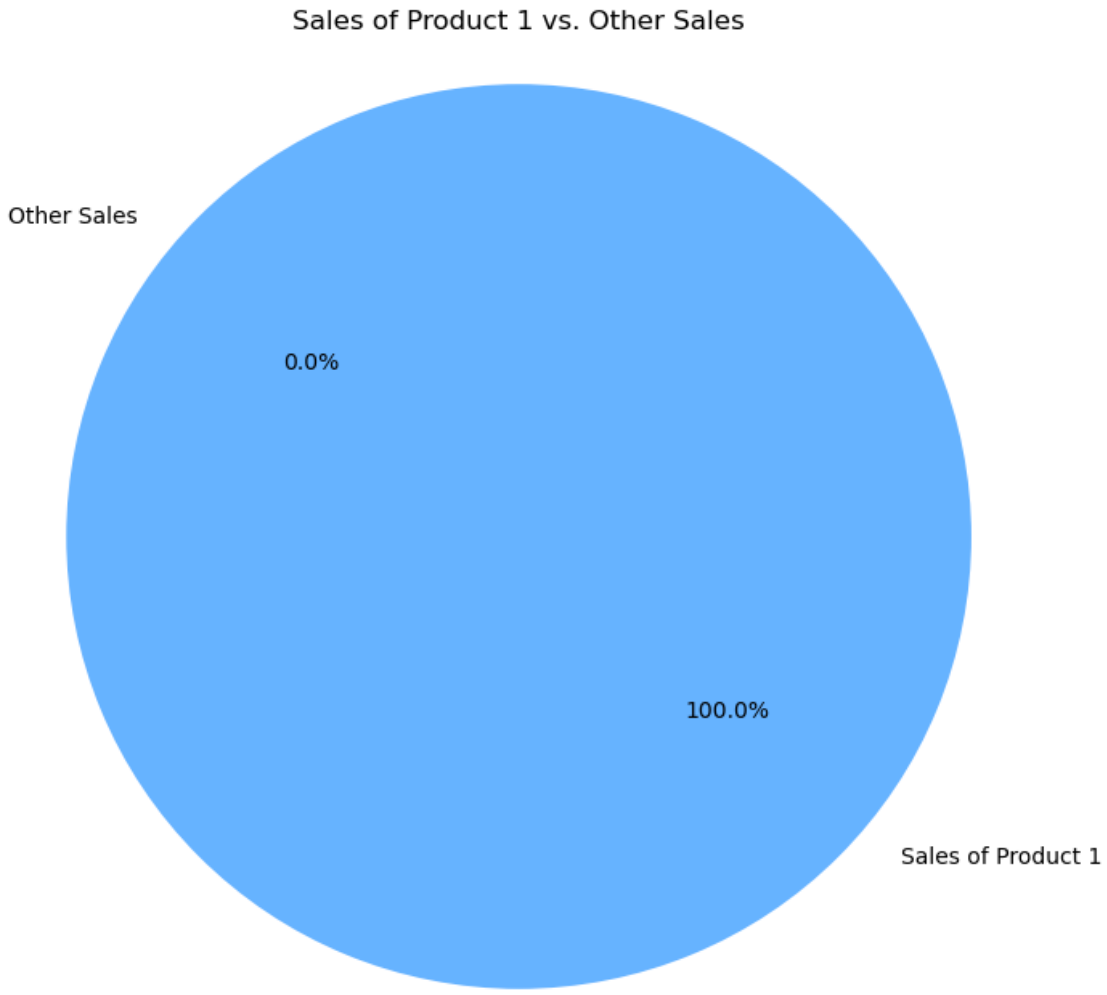


[ ]:

[19]:
```
# Calculate total sales for each product
total_sales = df[['S-P1', 'S-P2', 'S-P3', 'S-P4']].sum()

# Create a pie chart
plt.figure(figsize=(8, 8))
plt.pie(total_sales, labels=total_sales.index, autopct='%1.1f%%',␣
  ↪startangle=140, colors=['#ff9999','#66b3ff','#99ff99','#ffcc99'])
plt.title('Total Sales Distribution by Product')
plt.axis('equal')  # Equal aspect ratio ensures the pie chart is circular
plt.show()
```



[20]:
```
# Calculate total sales for Product 1
total_sales_p1 = df['S-P1'].sum()

# Create a pie chart
plt.figure(figsize=(8, 8))
plt.pie([total_sales_p1, df['S-P1'].sum()-total_sales_p1], labels=['Sales of␣
 ↪Product 1', 'Other Sales'], autopct='%1.1f%%', startangle=140,␣
 ↪colors=['#66b3ff','#99ff99'])
plt.title('Sales of Product 1 vs. Other Sales')
plt.axis('equal')
plt.show()
```



[21]:
```
total_sales_p1 = df['Q-P1'].sum()
print(f"Total unit sales of product 1: {total_sales_p1}")
```

```
Total unit sales of product 1: 18860169
```

[22]:
```
total_sales_p2 = df['Q-P2'].sum()
print(f"Total unit sales of product 2: {total_sales_p2}")
```

```
Total unit sales of product 2: 9741271
```

[23]:
```
total_sales_p3 = df['Q-P3'].sum()
print(f"Total unit sales of product 3: {total_sales_p3}")
```

```
Total unit sales of product 3: 14379603
```

[24]:
```
total_sales_p4 = df['Q-P4'].sum()
print(f"Total unit sales of product 4: {total_sales_p4}")
```

```
Total unit sales of product 4: 5139979
```

[25]:
```
total_revenue_p1 = df['S-P1'].sum()
print(f"Total revenue from product 1: {total_revenue_p1}")
```

```
Total revenue from product 1: 59786735.730000004
```

[26]:
```
total_revenue_p2 = df['S-P2'].sum()
print(f"Total revenue from product 2: {total_revenue_p2}")
```

```
Total revenue from product 2: 61759658.14
```

[27]:
```
total_revenue_p3 = df['S-P3'].sum()
print(f"Total revenue from product 3: {total_revenue_p3}")
```

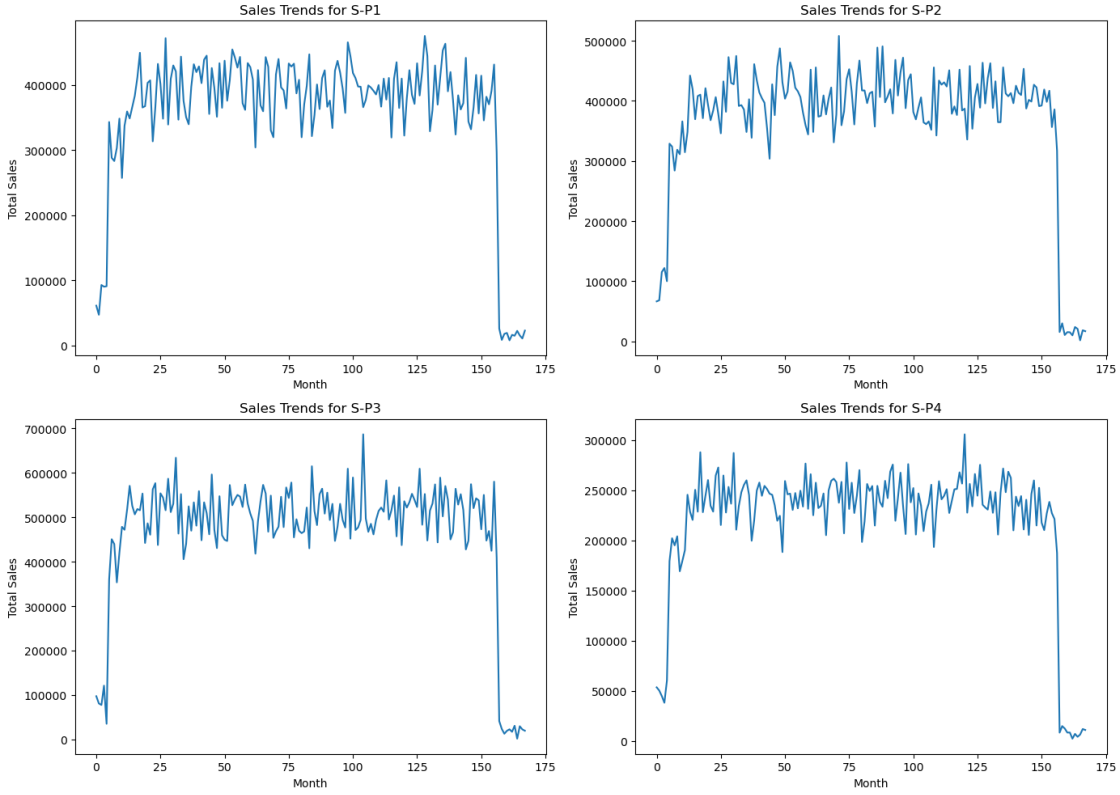
```
Total revenue from product 3: 77937448.26
```

[28]:
```
total_revenue_p4 = df['S-P4'].sum()
print(f"Total revenue from product 4: {total_revenue_p4}")
```

```
Total revenue from product 4: 36648050.269999996
```

[31]:
```
import matplotlib.pyplot as plt

# Assuming 'df' contains your data with dates set as index

# Extract the month and year from the date
df['Month'] = df.index.month
df['Year'] = df.index.year

# Group by month and calculate total sales for each product
monthly_sales = df.groupby(['Year', 'Month'])[['S-P1', 'S-P2', 'S-P3', 'S-P4']].
 ↪sum()

# Reset the index for plotting
monthly_sales = monthly_sales.reset_index()

# Create separate subplots for each product
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

products = ['S-P1', 'S-P2', 'S-P3', 'S-P4']

for i, ax in enumerate(axes.flat):
    col = products[i]
    ax.plot(monthly_sales.index, monthly_sales[col], label=col)
```

```
    ax.set_xlabel('Month')
    ax.set_ylabel('Total Sales')
    ax.set_title(f'Sales Trends for {col}')

# Adjust layout
plt.tight_layout()

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



[32]:
```
# Calculate total sales for each product
total_sales_p1 = df['S-P1'].sum()
total_sales_p2 = df['S-P2'].sum()
total_sales_p3 = df['S-P3'].sum()
total_sales_p4 = df['S-P4'].sum()

# Create a dictionary to store the total sales for each product
total_sales_dict = {
    'Product 1': total_sales_p1,
    'Product 2': total_sales_p2,
    'Product 3': total_sales_p3,
    'Product 4': total_sales_p4
```

```
}

# Find the product with the highest total sales
best_selling_product = max(total_sales_dict, key=total_sales_dict.get)

print(f"The product with the highest total sales is: {best_selling_product}")
```

```
The product with the highest total sales is: Product 3
```

[34]:
```
# Filter data for December 31st for all years
dec_31_sales = df[(df.index.month == 12) & (df.index.day == 31)]

# Calculate the average units sold for each product on December 31st
avg_units_p1 = dec_31_sales['Q-P1'].mean()
avg_units_p2 = dec_31_sales['Q-P2'].mean()
avg_units_p3 = dec_31_sales['Q-P3'].mean()
avg_units_p4 = dec_31_sales['Q-P4'].mean()

print(f"Estimated average units sold on December 31st:")
print(f"Product 1: {avg_units_p1}")
print(f"Product 2: {avg_units_p2}")
print(f"Product 3: {avg_units_p3}")
print(f"Product 4: {avg_units_p4}")
```

```
Estimated average units sold on December 31st:
Product 1: nan
Product 2: nan
Product 3: nan
Product 4: nan
```

[39]:
```
# Calculate total sales and revenue without dropping any product
total_sales_without_dropping = df[['S-P1', 'S-P2', 'S-P3', 'S-P4']].sum().sum()
total_revenue_without_dropping = df[['S-P1', 'S-P2', 'S-P3', 'S-P4']].sum().
 ↪sum()

# Calculate total sales and revenue if each product is dropped
for i in range(1, 5):
    product = f'Q-P{i}'
    total_sales_dropped = total_sales_without_dropping - df[product].sum()
    total_revenue_dropped = total_revenue_without_dropping - (df[product] *␣
 ↪df[f'S-P{i}']).sum()

    # Assess the impact
    impact_on_sales = ((total_sales_dropped - total_sales_without_dropping) /␣
 ↪total_sales_without_dropping) * 100
    impact_on_revenue = ((total_revenue_dropped -␣
 ↪total_revenue_without_dropping) / total_revenue_without_dropping) * 100
```

```
    print(f"If Product {i} is dropped:")
    print(f"Total Sales: {total_sales_dropped}")
    print(f"Total Revenue: {total_revenue_dropped}")
    print(f"Change in Total Sales: {impact_on_sales:.2f}%")
    print(f"Change in Total Revenue: {impact_on_revenue:.2f}%")
    print("\n")
```

```
If Product 1 is dropped:
Total Sales: 217271723.39999998
Total Revenue: -319262162780.35
Change in Total Sales: -7.99%
Change in Total Revenue: -135305.02%


If Product 2 is dropped:
Total Sales: 226390621.39999998
Total Revenue: -165708709253.7
Change in Total Sales: -4.13%
Change in Total Revenue: -70276.34%


If Product 3 is dropped:
Total Sales: 221752289.39999998
Total Revenue: -313993178319.66
Change in Total Sales: -6.09%
Change in Total Revenue: -133073.64%


If Product 4 is dropped:
Total Sales: 230991913.39999998
Total Revenue: -49026924270.67
Change in Total Sales: -2.18%
Change in Total Revenue: -20862.52%
```

[ ]:

[ ]: