documented python file

March 29, 2025

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```python
df=pd.read_csv("supply_chain_data.csv")
# Display the first 5 rows of the dataset
print(df.head())
```

```
  Product type   SKU      Price  Availability  Number of products sold  \
0     haircare  SKU0  69.808006            55                      802
1     skincare  SKU1  14.843523            95                      736
2     haircare  SKU2  11.319683            34                        8
3     skincare  SKU3  61.163343            68                       83
4     skincare  SKU4   4.805496            26                      871

   Revenue generated Customer demographics  Stock levels  Lead times  \
0        8661.996792            Non-binary            58           7
1        7460.900065                Female            53          30
2        9577.749626               Unknown             1          10
3        7766.836426            Non-binary            23          13
4        2686.505152            Non-binary             5           3

   Order quantities  …  Location Lead time  Production volumes  \
0                96  …    Mumbai        29                 215
1                37  …    Mumbai        23                 517
2                88  …    Mumbai        12                 971
3                59  …   Kolkata        24                 937
4                56  …     Delhi         5                 414

  Manufacturing lead time Manufacturing costs  Inspection results  \
0                      29           46.279879             Pending
1                      30           33.616769             Pending
2                      27           30.688019             Pending
3                      18           35.624741                Fail
4                       3           92.065161                Fail

   Defect rates  Transportation modes   Routes       Costs
```

```
0      0.226410                    Road  Route B  187.752075
1      4.854068                    Road  Route B  503.065579
2      4.580593                     Air  Route C  141.920282
3      4.746649                    Rail  Route A  254.776159
4      3.145580                     Air  Route A  923.440632

[5 rows x 24 columns]
```

[3]:
```python
# Check the shape of the dataset (rows, columns)
print("Shape of the dataset:", df.shape)
```

```
Shape of the dataset: (100, 24)
```

[4]:
```python
# Check for missing values
print("Missing values in each column:")
print(df.isnull().sum())
```

```
Missing values in each column:
Product type               0
SKU                        0
Price                      0
Availability               0
Number of products sold    0
Revenue generated          0
Customer demographics      0
Stock levels               0
Lead times                 0
Order quantities           0
Shipping times             0
Shipping carriers          0
Shipping costs             0
Supplier name              0
Location                   0
Lead time                  0
Production volumes         0
Manufacturing lead time    0
Manufacturing costs        0
Inspection results         0
Defect rates               0
Transportation modes       0
Routes                     0
Costs                      0
dtype: int64
```

[5]:
```python
# Get basic statistics of numerical columns
print("Basic statistics:")
print(df.describe())
```

```
Basic statistics:
```

```
          Price  Availability  Number of products sold  Revenue generated  \
count  100.000000  100.000000              100.000000         100.000000
mean    49.462461   48.400000              460.990000        5776.048187
std     31.168193   30.743317              303.780074        2732.841744
min      1.699976    1.000000                8.000000        1061.618523
25%     19.597823   22.750000              184.250000        2812.847151
50%     51.239830   43.500000              392.500000        6006.352023
75%     77.198228   75.000000              704.250000        8253.976920
max     99.171329  100.000000              996.000000        9866.465458

       Stock levels  Lead times  Order quantities  Shipping times  \
count    100.000000  100.000000        100.000000      100.000000
mean      47.770000   15.960000         49.220000        5.750000
std       31.369372    8.785801         26.784429        2.724283
min        0.000000    1.000000          1.000000        1.000000
25%       16.750000    8.000000         26.000000        3.750000
50%       47.500000   17.000000         52.000000        6.000000
75%       73.000000   24.000000         71.250000        8.000000
max      100.000000   30.000000         96.000000       10.000000

       Shipping costs   Lead time  Production volumes  \
count      100.000000  100.000000          100.000000
mean         5.548149   17.080000          567.840000
std          2.651376    8.846251          263.046861
min          1.013487    1.000000          104.000000
25%          3.540248   10.000000          352.000000
50%          5.320534   18.000000          568.500000
75%          7.601695   25.000000          797.000000
max          9.929816   30.000000          985.000000

       Manufacturing lead time  Manufacturing costs  Defect rates       Costs
count                100.00000           100.000000    100.000000  100.000000
mean                  14.77000            47.266693      2.277158  529.245782
std                    8.91243            28.982841      1.461366  258.301696
min                    1.00000             1.085069      0.018608  103.916248
25%                    7.00000            22.983299      1.009650  318.778455
50%                   14.00000            45.905622      2.141863  520.430444
75%                   23.00000            68.621026      3.563995  763.078231
max                   30.00000            99.466109      4.939255  997.413450
```

[6]:
```python
# Check data types of each column
print("Data types:")
print(df.dtypes)
```

```
Data types:
Product type                object
SKU                         object
Price                      float64
```

```
Availability                  int64
Number of products sold       int64
Revenue generated           float64
Customer demographics        object
Stock levels                  int64
Lead times                    int64
Order quantities              int64
Shipping times                int64
Shipping carriers            object
Shipping costs              float64
Supplier name                object
Location                     object
Lead time                     int64
Production volumes            int64
Manufacturing lead time       int64
Manufacturing costs         float64
Inspection results           object
Defect rates                float64
Transportation modes         object
Routes                       object
Costs                       float64
dtype: object
```

[7]:
```
df['Product type'].unique()
```

[7]:
```
array(['haircare', 'skincare', 'cosmetics'], dtype=object)
```

[8]:
```
df['Transportation modes'].unique()
```

[8]:
```
array(['Road', 'Air', 'Rail', 'Sea'], dtype=object)
```

[9]:
```
df['Routes'].unique()
```

[9]:
```
array(['Route B', 'Route C', 'Route A'], dtype=object)
```

[10]:
```
df['Customer demographics'].unique()
```

[10]:
```
array(['Non-binary', 'Female', 'Unknown', 'Male'], dtype=object)
```

[11]:
```
df['Location'].unique()
```

[11]:
```
array(['Mumbai', 'Kolkata', 'Delhi', 'Bangalore', 'Chennai'], dtype=object)
```

[12]:
```
# Frequency distribution for categorical variables
categorical_variables = ['Product type', 'Customer demographics', 'Shipping␣
 ↪carriers', 'Supplier name', 'Location', 'Inspection results',␣
 ↪'Transportation modes', 'Routes']
frequency_distribution = {}
```

```
for col in categorical_variables:
    frequency_distribution[col] = df[col].value_counts()

print("\nFrequency Distribution for Categorical Variables:")
for col, freq_dist in frequency_distribution.items():
    print(f"\n{col}:")
    print(freq_dist)
```

```
Frequency Distribution for Categorical Variables:

Product type:
Product type
skincare     40
haircare     34
cosmetics    26
Name: count, dtype: int64

Customer demographics:
Customer demographics
Unknown       31
Female        25
Non-binary    23
Male          21
Name: count, dtype: int64

Shipping carriers:
Shipping carriers
Carrier B    43
Carrier C    29
Carrier A    28
Name: count, dtype: int64

Supplier name:
Supplier name
Supplier 1    27
Supplier 2    22
Supplier 5    18
Supplier 4    18
Supplier 3    15
Name: count, dtype: int64

Location:
Location
Kolkata      25
Mumbai       22
Chennai      20
```

```
Bangalore    18
Delhi        15
Name: count, dtype: int64

Inspection results:
Inspection results
Pending    41
Fail       36
Pass       23
Name: count, dtype: int64

Transportation modes:
Transportation modes
Road    29
Rail    28
Air     26
Sea     17
Name: count, dtype: int64

Routes:
Routes
Route A    43
Route B    37
Route C    20
Name: count, dtype: int64
```

[13]:
```python
# Drop rows with missing values (if necessary)
df = df.dropna()
```

[14]:
```python
# Check for duplicates
print("Number of duplicate rows:", df.duplicated().sum())
```

```
Number of duplicate rows: 0
```

[15]:
```python
# Convert 'Order_Date' and 'Delivery_Date' to datetime (if applicable)
# df['Order_Date'] = pd.to_datetime(df['Order_Date'])
# df['Delivery_Date'] = pd.to_datetime(df['Delivery_Date'])
# Convert data types
df['Product type'] = df['Product type'].astype(str)
df['SKU'] = df['SKU'].astype(str)
df['Price'] = df['Price'].astype(float)
df['Availability'] = df['Availability']. astype(int)
df['Number of products sold'] = df['Number of products sold'].astype(int)
df['Revenue generated'] = df['Revenue generated'].astype(float)
df['Customer demographics'] = df['Customer demographics'].astype(str)
df['Stock levels'] = df['Stock levels'].astype(int)
df['Lead times'] = df['Lead times'].astype(int)
df['Order quantities'] = df['Order quantities'].astype(int)
```

```
df['Shipping times'] = df['Shipping times'].astype(int)
df['Shipping carriers'] = df['Shipping carriers'].astype(str)
df['Shipping costs'] = df['Shipping costs'].astype(float)
df['Supplier name'] = df['Supplier name'].astype(str)
df['Location'] = df['Location'].astype(str)
df['Lead time'] = df['Lead time'].astype(int)
df['Production volumes'] = df['Production volumes'].astype(int)
df['Manufacturing lead time'] = df['Manufacturing lead time'].astype(int)
df['Manufacturing costs'] = df['Manufacturing costs'].astype(float)
df['Inspection results'] = df['Inspection results'].astype(str)
df['Defect rates'] = df['Defect rates'].astype(float)
df['Transportation modes'] = df['Transportation modes'].astype(str)
df['Routes'] = df['Routes'].astype(str)
df['Costs'] = df['Costs'].astype(float)

# Display the updated dataframe
print(df.head())
```

```
  Product type   SKU      Price  Availability  Number of products sold  \
0     haircare  SKU0  69.808006            55                      802
1     skincare  SKU1  14.843523            95                      736
2     haircare  SKU2  11.319683            34                        8
3     skincare  SKU3  61.163343            68                       83
4     skincare  SKU4   4.805496            26                      871

   Revenue generated Customer demographics  Stock levels  Lead times  \
0        8661.996792            Non-binary            58           7
1        7460.900065                Female            53          30
2        9577.749626               Unknown             1          10
3        7766.836426            Non-binary            23          13
4        2686.505152            Non-binary             5           3

   Order quantities  …  Location Lead time  Production volumes  \
0                96  …    Mumbai        29                 215
1                37  …    Mumbai        23                 517
2                88  …    Mumbai        12                 971
3                59  …   Kolkata        24                 937
4                56  …     Delhi         5                 414

  Manufacturing lead time Manufacturing costs  Inspection results  \
0                      29           46.279879             Pending
1                      30           33.616769             Pending
2                      27           30.688019             Pending
3                      18           35.624741                Fail
4                       3           92.065161                Fail

   Defect rates  Transportation modes   Routes       Costs
0      0.226410                  Road  Route B  187.752075
```

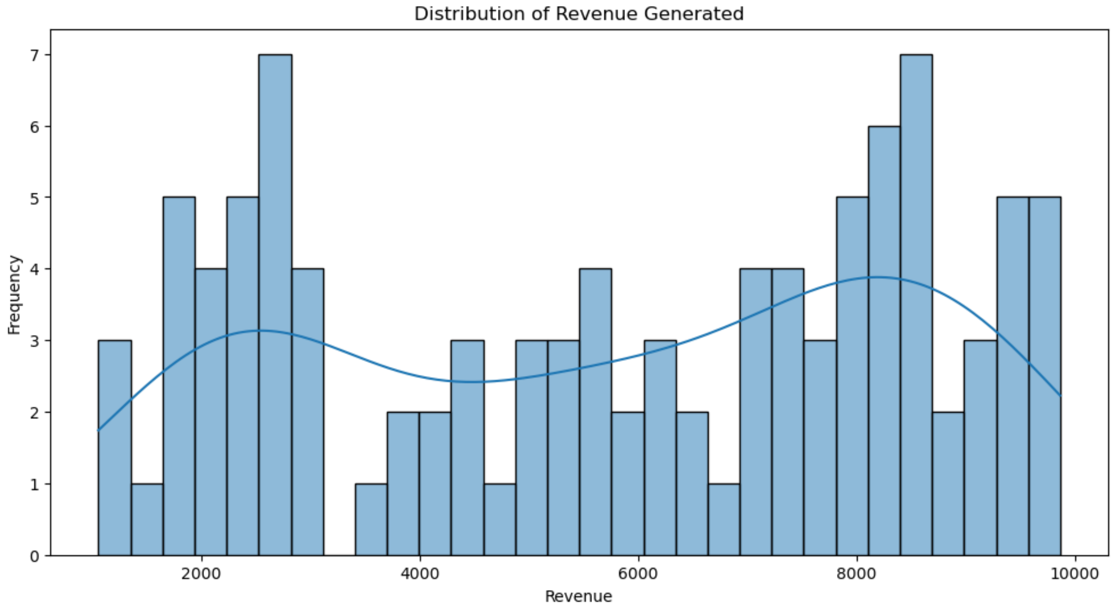
```
1      4.854068                  Road  Route B  503.065579
2      4.580593                   Air  Route C  141.920282
3      4.746649                  Rail  Route A  254.776159
4      3.145580                   Air  Route A  923.440632

[5 rows x 24 columns]
```

[16]:
```
# Save the cleaned dataset to a new CSV file
df.to_csv('cleaned_supply_chain_data final.csv', index=False)
```

[17]:
```
# Distribution of categorical columns
print(df['Product type'].value_counts())
print(df['Shipping carriers'].value_counts())
print(df['Customer demographics'].value_counts())
```

```
Product type
skincare     40
haircare     34
cosmetics    26
Name: count, dtype: int64
Shipping carriers
Carrier B    43
Carrier C    29
Carrier A    28
Name: count, dtype: int64
Customer demographics
Unknown       31
Female        25
Non-binary    23
Male          21
Name: count, dtype: int64
```

[18]:
```
# Visualize distributions
plt.figure(figsize=(12, 6))
sns.histplot(df['Revenue generated'], bins=30, kde=True)
plt.title('Distribution of Revenue Generated')
plt.xlabel('Revenue')
plt.ylabel('Frequency')
plt.show()
```



[19]:
```
# Check if 'Revenue generated' == 'products_sold' * 'Price'
df['Calculated Revenue'] = df['Number of products sold'] * df['Price']
df['Revenue Check'] = df['Revenue generated'] == df['Calculated Revenue']

# Identify rows with discrepancies
discrepancies = df[~df['Revenue Check']]  # ~ means "NOT"

# Print results
print(f"Total rows checked: {len(df)}")
print(f"Rows with discrepancies: {len(discrepancies)}")
print("\nDiscrepant rows:")
print(discrepancies[['Product type', 'Revenue generated', 'Calculated Revenue',␣
↪'Revenue Check']])
```

```
Total rows checked: 100
Rows with discrepancies: 100

Discrepant rows:
   Product type  Revenue generated  Calculated Revenue  Revenue Check
0      haircare        8661.996792        55986.020443          False
1      skincare        7460.900065        10924.833134          False
2      haircare        9577.749626           90.557466          False
3      skincare        7766.836426         5076.557471          False
4      skincare        2686.505152         4185.587047          False
..          …                  …                   …              …
95     haircare        7386.363944        52351.439092          False
96    cosmetics        7698.424766         7913.094580          False
```

```
97     haircare       4370.916580         218.618898          False
98     skincare       8525.952560       18035.954246          False
99     haircare       9185.185829       42960.681103          False

[100 rows x 4 columns]
```

[20]:

```
# Optional: Export discrepancies to a new file
discrepancies.to_csv('revenue_discrepancies.csv', index=False)
```

[21]:

```
df
```

[21]:

```
   Product type    SKU      Price  Availability  Number of products sold  \
0      haircare   SKU0  69.808006            55                      802
1      skincare   SKU1  14.843523            95                      736
2      haircare   SKU2  11.319683            34                        8
3      skincare   SKU3  61.163343            68                       83
4      skincare   SKU4   4.805496            26                      871
..          …      …        …           …                      …
95     haircare  SKU95  77.903927            65                      672
96    cosmetics  SKU96  24.423131            29                      324
97     haircare  SKU97   3.526111            56                       62
98     skincare  SKU98  19.754605            43                      913
99     haircare  SKU99  68.517833            17                      627

    Revenue generated Customer demographics  Stock levels  Lead times  \
0         8661.996792            Non-binary            58           7
1         7460.900065                Female            53          30
2         9577.749626               Unknown             1          10
3         7766.836426            Non-binary            23          13
4         2686.505152            Non-binary             5           3
..                …                   …           …         …
95        7386.363944               Unknown            15          14
96        7698.424766            Non-binary            67           2
97        4370.916580                  Male            46          19
98        8525.952560                Female            53           1
99        9185.185829               Unknown            55           8

    Order quantities  …  Production volumes Manufacturing lead time  \
0                 96  …                 215                      29
1                 37  …                 517                      30
2                 88  …                 971                      27
3                 59  …                 937                      18
4                 56  …                 414                       3
..               …  …                 …                     …
95                26  …                 450                      26
96                32  …                 648                      28
97                 4  …                 535                      13
```

```
98                27  …                  581                         9
99                59  …                  921                         2

    Manufacturing costs Inspection results Defect rates  Transportation modes  \
0             46.279879            Pending     0.226410                  Road
1             33.616769            Pending     4.854068                  Road
2             30.688019            Pending     4.580593                   Air
3             35.624741               Fail     4.746649                  Rail
4             92.065161               Fail     3.145580                   Air
..                  …                  …          …                   …
95            58.890686            Pending     1.210882                   Air
96            17.803756            Pending     3.872048                  Road
97            65.765156               Fail     3.376238                  Road
98             5.604691            Pending     2.908122                  Rail
99            38.072899               Fail     0.346027                  Rail

     Routes       Costs  Calculated Revenue Revenue Check
0   Route B  187.752075        55986.020443         False
1   Route B  503.065579        10924.833134         False
2   Route C  141.920282           90.557466         False
3   Route A  254.776159         5076.557471         False
4   Route A  923.440632         4185.587047         False
..      …          …                 …           …
95  Route A  778.864241        52351.439092         False
96  Route A  188.742141         7913.094580         False
97  Route A  540.132423          218.618898         False
98  Route A  882.198864        18035.954246         False
99  Route B  210.743009        42960.681103         False

[100 rows x 26 columns]
```

[22]:

```
# Revenue Difference = Revenue generated - Actual Revenue
df["Revenue Difference"] = df["Revenue generated"] - df["Calculated Revenue"]
```

[23]:

```
df
```

[23]:

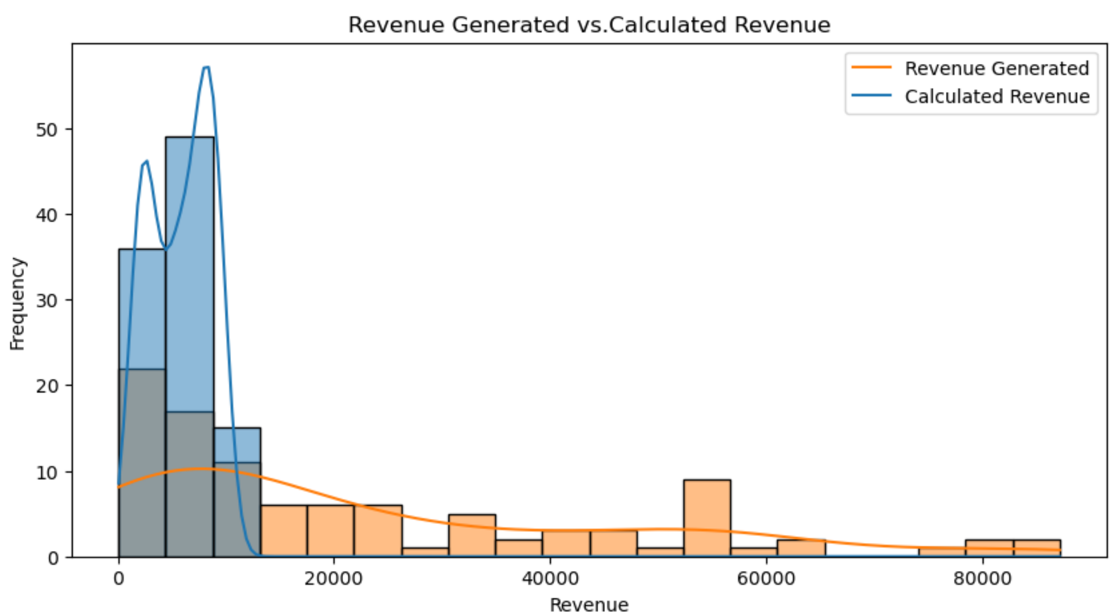
```
   Product type    SKU      Price  Availability  Number of products sold  \
0      haircare   SKU0  69.808006            55                      802
1      skincare   SKU1  14.843523            95                      736
2      haircare   SKU2  11.319683            34                        8
3      skincare   SKU3  61.163343            68                       83
4      skincare   SKU4   4.805496            26                      871
..          …      …          …           …                      …
95     haircare  SKU95  77.903927            65                      672
96    cosmetics  SKU96  24.423131            29                      324
97     haircare  SKU97   3.526111            56                       62
98     skincare  SKU98  19.754605            43                      913
```

```
99    haircare  SKU99  68.517833           17                       627

   Revenue generated Customer demographics  Stock levels  Lead times  \
0        8661.996792            Non-binary            58           7
1        7460.900065                Female            53          30
2        9577.749626               Unknown             1          10
3        7766.836426            Non-binary            23          13
4        2686.505152            Non-binary             5           3
..               …                   …           …          …
95       7386.363944               Unknown            15          14
96       7698.424766            Non-binary            67           2
97       4370.916580                  Male            46          19
98       8525.952560                Female            53           1
99       9185.185829               Unknown            55           8

   Order quantities  …  Manufacturing lead time Manufacturing costs  \
0                96  …                       29           46.279879
1                37  …                       30           33.616769
2                88  …                       27           30.688019
3                59  …                       18           35.624741
4                56  …                        3           92.065161
..              …  …                     …                 …
95               26  …                       26           58.890686
96               32  …                       28           17.803756
97                4  …                       13           65.765156
98               27  …                        9            5.604691
99               59  …                        2           38.072899

   Inspection results Defect rates Transportation modes   Routes       Costs  \
0             Pending     0.226410                 Road  Route B  187.752075
1             Pending     4.854068                 Road  Route B  503.065579
2             Pending     4.580593                  Air  Route C  141.920282
3                Fail     4.746649                 Rail  Route A  254.776159
4                Fail     3.145580                  Air  Route A  923.440632
..                …          …                  …      …         …
95            Pending     1.210882                  Air  Route A  778.864241
96            Pending     3.872048                 Road  Route A  188.742141
97               Fail     3.376238                 Road  Route A  540.132423
98            Pending     2.908122                 Rail  Route A  882.198864
99               Fail     0.346027                 Rail  Route B  210.743009

   Calculated Revenue  Revenue Check Revenue Difference
0        55986.020443          False      -47324.023651
1        10924.833134          False       -3463.933069
2           90.557466          False        9487.192160
3         5076.557471          False        2690.278955
4         4185.587047          False       -1499.081895
```

```
..           …          …          …
95   52351.439092      False   -44965.075148
96    7913.094580      False     -214.669814
97     218.618898      False     4152.297682
98   18035.954246      False    -9510.001686
99   42960.681103      False   -33775.495274

[100 rows x 27 columns]
```

[24]:

```
# Figure 1: Revenue Comparison
plt.figure(figsize=(10, 5))
sns.histplot(df[["Revenue generated", "Calculated Revenue"]], kde=True, bins=20)
plt.title("Revenue Generated vs.Calculated Revenue")
plt.xlabel("Revenue")
plt.ylabel("Frequency")
plt.legend(["Revenue Generated", "Calculated Revenue"])
plt.show()
```

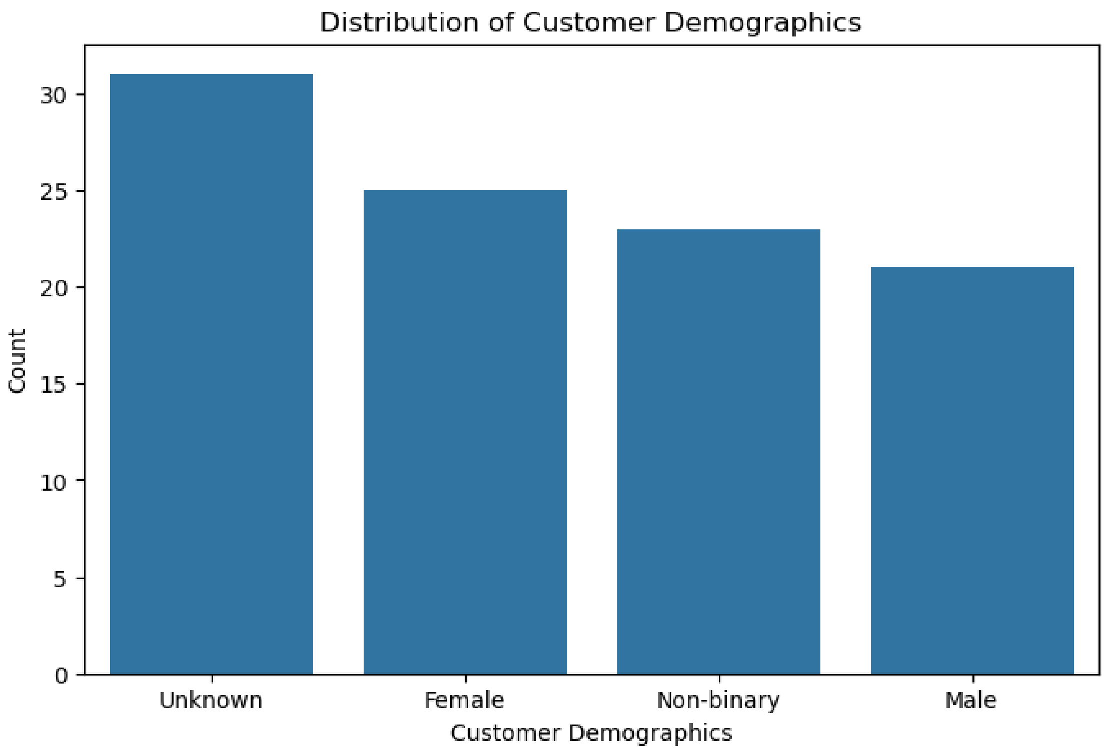


[25]:

```
print(df['Customer demographics'].value_counts())
```

```
Customer demographics
Unknown        31
Female         25
Non-binary     23
Male           21
Name: count, dtype: int64
```

[26]:
```
# Visualize the distribution
plt.figure(figsize=(8, 5))
sns.countplot(data=df, x='Customer demographics', order=df['Customer␣
↪demographics'].value_counts().index)
plt.title('Distribution of Customer Demographics')
plt.xlabel('Customer Demographics')
plt.ylabel('Count')
plt.show()
```

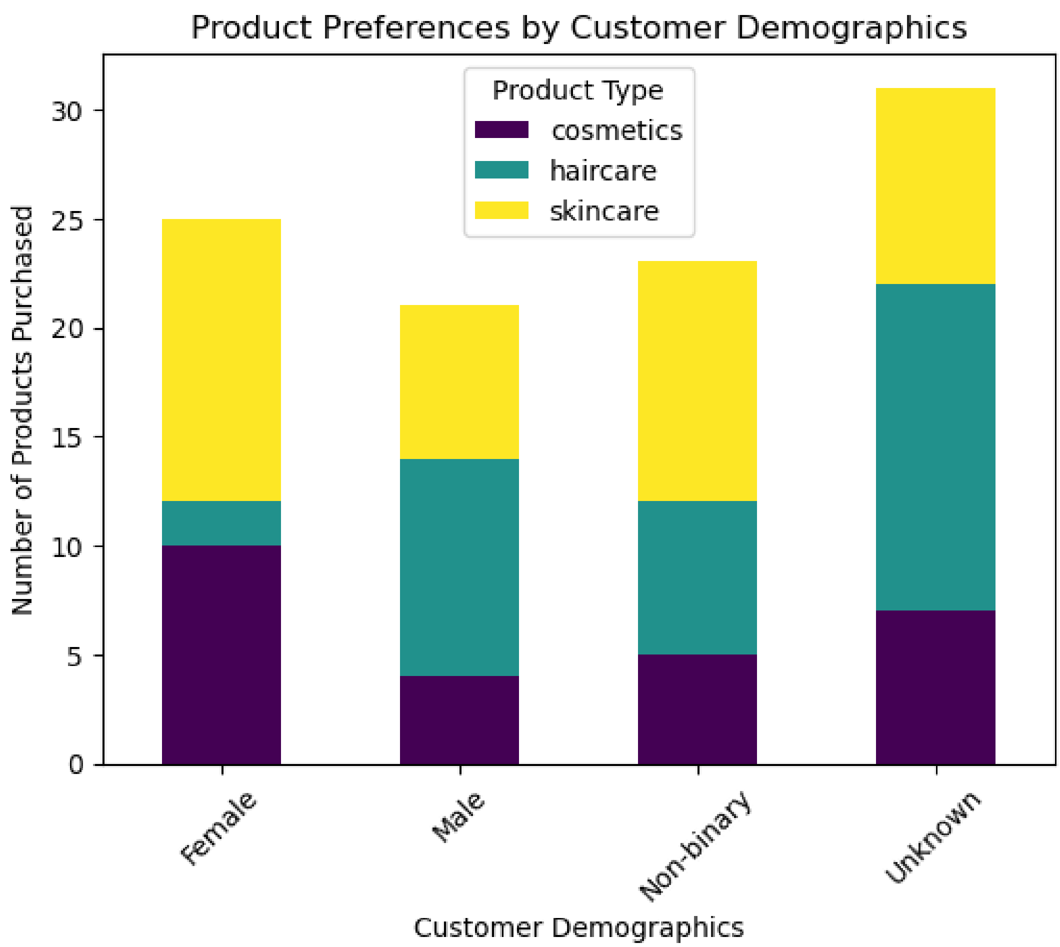


[27]:
```
# Group by 'Customer demographics' and 'Product type' to see product preferences
product_preferences = df.groupby(['Customer demographics', 'Product type']).
↪size().unstack(fill_value=0)

# Display the product preferences
print("Product Preferences by Customer Demographics:")
print(product_preferences)

# Visualize the product preferences
plt.figure(figsize=(12, 6))
product_preferences.plot(kind='bar', stacked=True, colormap='viridis')
plt.title('Product Preferences by Customer Demographics')
plt.xlabel('Customer Demographics')
```

```
plt.ylabel('Number of Products Purchased')
plt.xticks(rotation=45)
plt.legend(title='Product Type')
plt.show()
```

```
Product Preferences by Customer Demographics:
Product type           cosmetics  haircare  skincare
Customer demographics
Female                        10         2        13
Male                           4        10         7
Non-binary                     5         7        11
Unknown                        7        15         9
```

<Figure size 1200x600 with 0 Axes>



[28]:
```
df.rename(columns={"Number of products sold":"products_sold","Customer␣
↪demographics":"Gender"}, inplace=True)
```

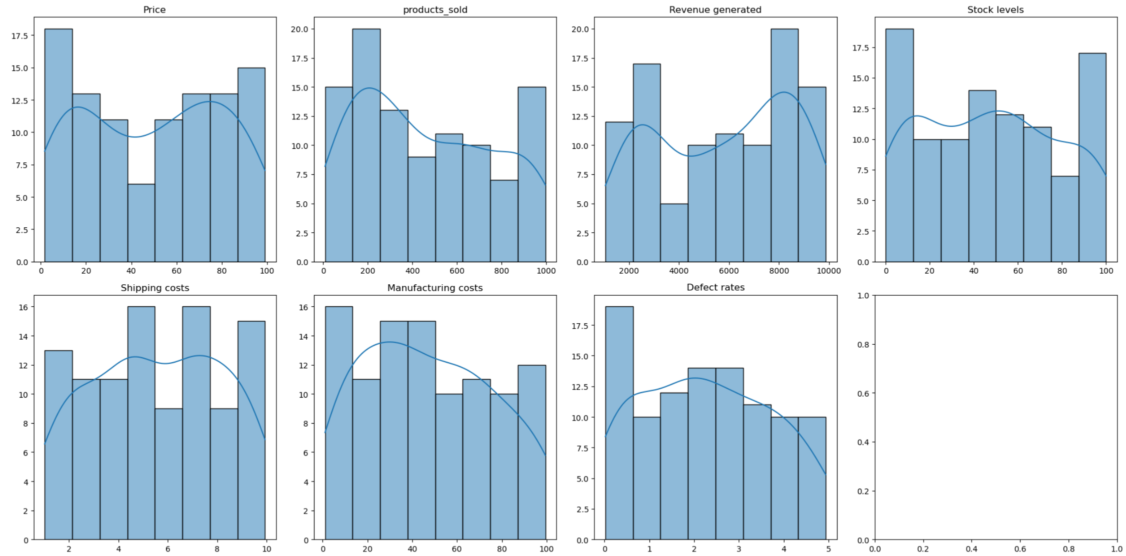
[29]:
```
df
```

[29]:
```
   Product type    SKU      Price  Availability  products_sold  \
0      haircare   SKU0  69.808006            55            802
1      skincare   SKU1  14.843523            95            736
2      haircare   SKU2  11.319683            34              8
3      skincare   SKU3  61.163343            68             83
4      skincare   SKU4   4.805496            26            871
..          …      …          …             …              …
95     haircare  SKU95  77.903927            65            672
96    cosmetics  SKU96  24.423131            29            324
97     haircare  SKU97   3.526111            56             62
98     skincare  SKU98  19.754605            43            913
99     haircare  SKU99  68.517833            17            627

    Revenue generated      Gender  Stock levels  Lead times  Order quantities  \
0         8661.996792  Non-binary            58           7                96
1         7460.900065      Female            53          30                37
2         9577.749626     Unknown             1          10                88
3         7766.836426  Non-binary            23          13                59
4         2686.505152  Non-binary             5           3                56
..                …           …           …          …                …
95        7386.363944     Unknown            15          14                26
96        7698.424766  Non-binary            67           2                32
97        4370.916580        Male            46          19                 4
98        8525.952560      Female            53           1                27
99        9185.185829     Unknown            55           8                59

    …  Manufacturing lead time Manufacturing costs  Inspection results  \
0   …                       29           46.279879             Pending
1   …                       30           33.616769             Pending
2   …                       27           30.688019             Pending
3   …                       18           35.624741                Fail
4   …                        3           92.065161                Fail
..  …                      …                 …                 …
95  …                       26           58.890686             Pending
96  …                       28           17.803756             Pending
97  …                       13           65.765156                Fail
98  …                        9            5.604691             Pending
99  …                        2           38.072899                Fail

   Defect rates Transportation modes   Routes       Costs  Calculated Revenue  \
0      0.226410                 Road  Route B  187.752075        55986.020443
1      4.854068                 Road  Route B  503.065579        10924.833134
2      4.580593                  Air  Route C  141.920282           90.557466
3      4.746649                 Rail  Route A  254.776159         5076.557471
4      3.145580                  Air  Route A  923.440632         4185.587047
```

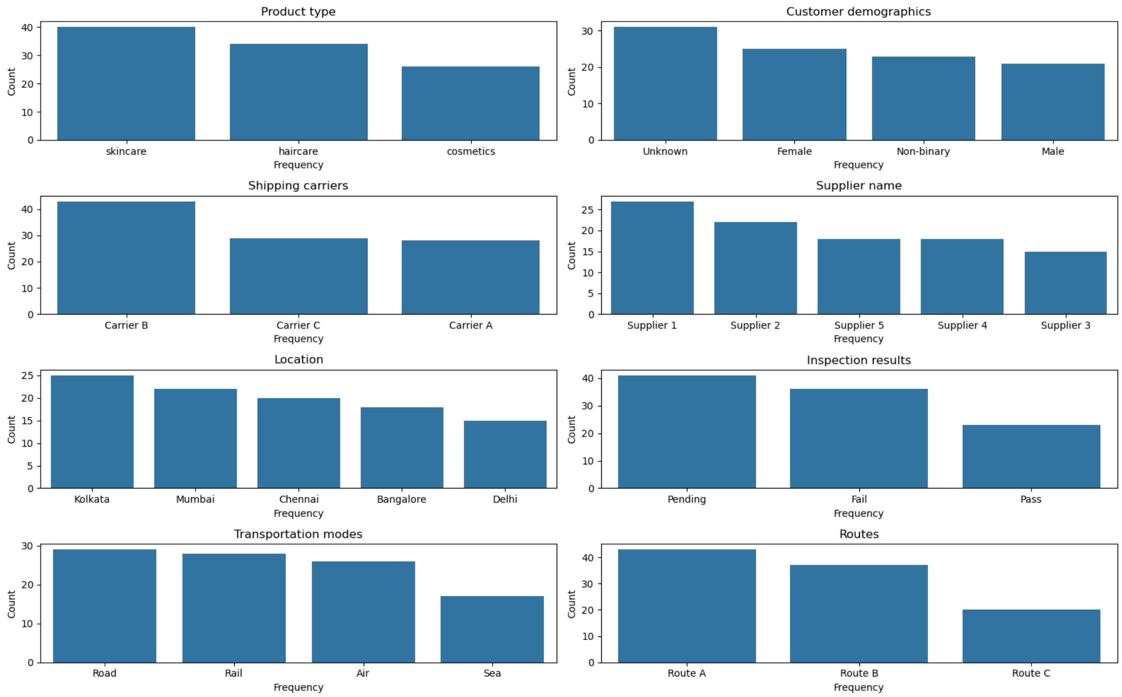
```
..        …                          …      …         …                 …
95      1.210882                   Air  Route A  778.864241      52351.439092
96      3.872048                  Road  Route A  188.742141       7913.094580
97      3.376238                  Road  Route A  540.132423        218.618898
98      2.908122                  Rail  Route A  882.198864      18035.954246
99      0.346027                  Rail  Route B  210.743009      42960.681103

    Revenue Check Revenue Difference
0           False      -47324.023651
1           False       -3463.933069
2           False        9487.192160
3           False        2690.278955
4           False       -1499.081895
..            …                  …
95          False      -44965.075148
96          False        -214.669814
97          False        4152.297682
98          False       -9510.001686
99          False      -33775.495274

[100 rows x 27 columns]
```

[30]:

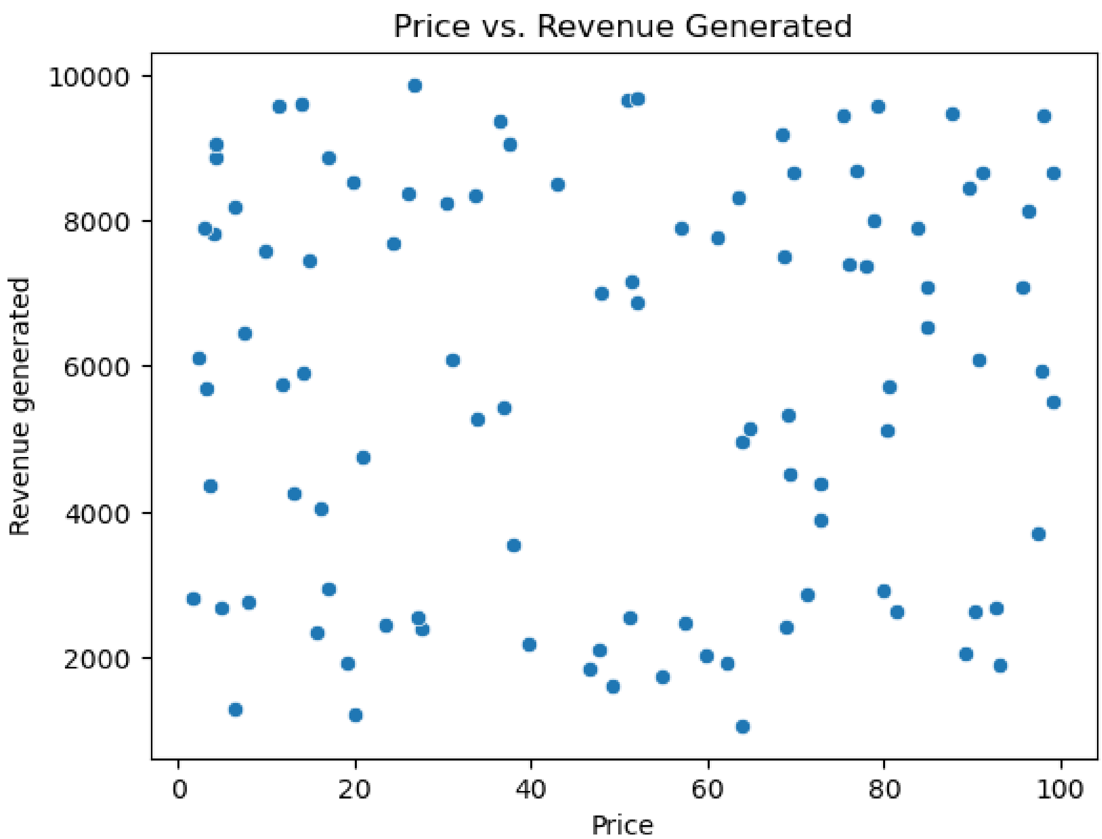
```python
# Selecting columns for visualization
selected_columns = ['Price', 'products_sold', 'Revenue generated', 'Stock␣
↪levels',
                    'Shipping costs', 'Manufacturing costs', 'Defect rates']

# Creating subplots
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(20, 10))

# Flatten the axes array
axes = axes.flatten()

# Plotting each selected column
for i, column in enumerate(selected_columns):
    sns.histplot(df[column], ax=axes[i], kde=True)
    axes[i].set_title(column)
    axes[i].set_xlabel('')
    axes[i].set_ylabel('')

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



[31]:
```python
# Create subplots for each categorical variable
num_plots = len(categorical_variables)
num_cols = 2  # Number of columns in the subplot grid
num_rows = (num_plots + num_cols - 1) // num_cols  # Number of rows in the␣
↪subplot grid
fig, axes = plt.subplots(num_rows, num_cols, figsize=(16, 10))
axes = axes.flatten()

# Plot each categorical variable distribution
for i, (col, freq_dist) in enumerate(frequency_distribution.items()):
    ax = axes[i]
    sns.barplot(x=freq_dist.index, y=freq_dist.values, ax=ax)
    ax.set_title(col)
    ax.set_xlabel("Frequency")
    ax.set_ylabel("Count")
    ax.tick_params(axis='x')  # Rotate x-axis labels for better readability

# Hide empty subplots if there are any
for i in range(num_plots, num_rows * num_cols):
    fig.delaxes(axes[i])

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



[32]:
```
sns.scatterplot(x='Price', y='Revenue generated', data=df)
plt.title('Price vs. Revenue Generated')
plt.show()
```



[33]:
```
import plotly.express as px
import plotly.io as pio
import plotly.graph_objects as go
pio.templates.default = "plotly_white"
```

[34]:
```
fig = px.scatter(df, x='Price',
                 y='Revenue generated',
                 color='Product type',
                 hover_data=['products_sold'],
                 trendline="ols")
fig.show()
```

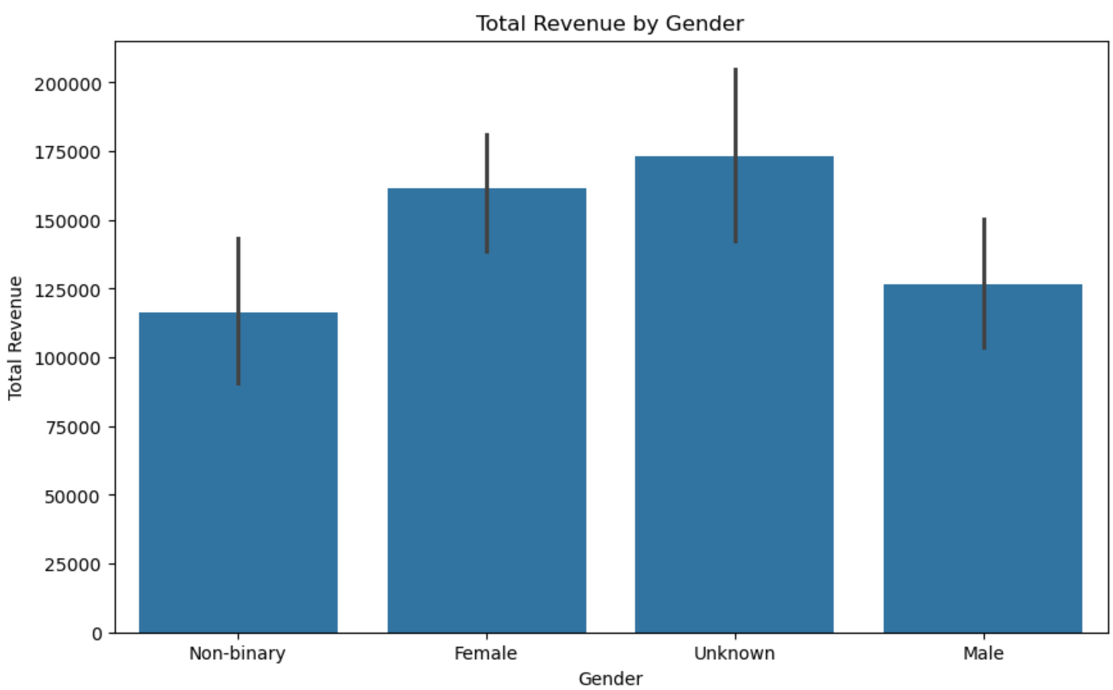
[35]:
```
#What are the top-selling products in terms of revenue or quantity?

df_top_products = df.groupby("Product type").agg(
    Total_Quantity_Sold=("products_sold", "sum"),
    Total_Revenue=("Revenue generated", "sum")
).reset_index()

df_top_products = df_top_products.sort_values(
```

```
    by=["Total_Revenue", "Total_Quantity_Sold"], ascending=[False, False]
)
print(df_top_products)
```

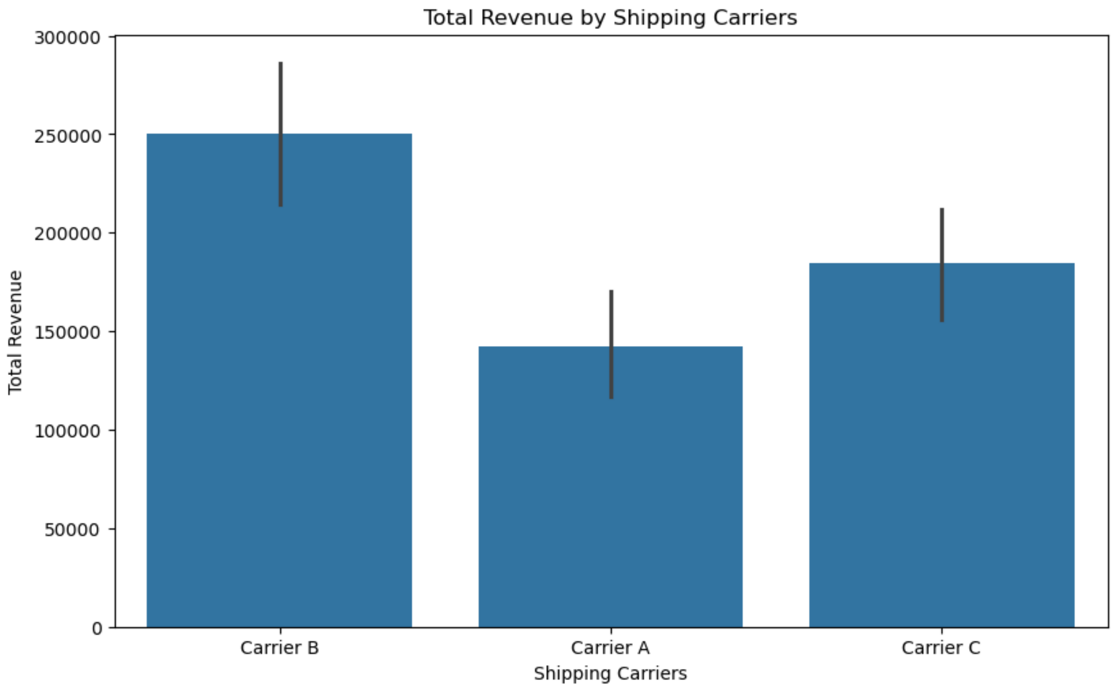
```
  Product type  Total_Quantity_Sold  Total_Revenue
2     skincare                20731  241628.162133
1     haircare                13611  174455.390606
0    cosmetics                11757  161521.266001
```

[36]:
```
sales_data = df.groupby('Product type')['products_sold'].sum().reset_index()

pie_chart = px.pie(sales_data, values='products_sold', names='Product type',
                   title='Sales by Product Type',
                   hover_data=['products_sold'],
                   hole=0.5,
                   color_discrete_sequence=px.colors.qualitative.Pastel)

pie_chart.update_traces(textposition='inside', textinfo='percent+label')
pie_chart.show()
```

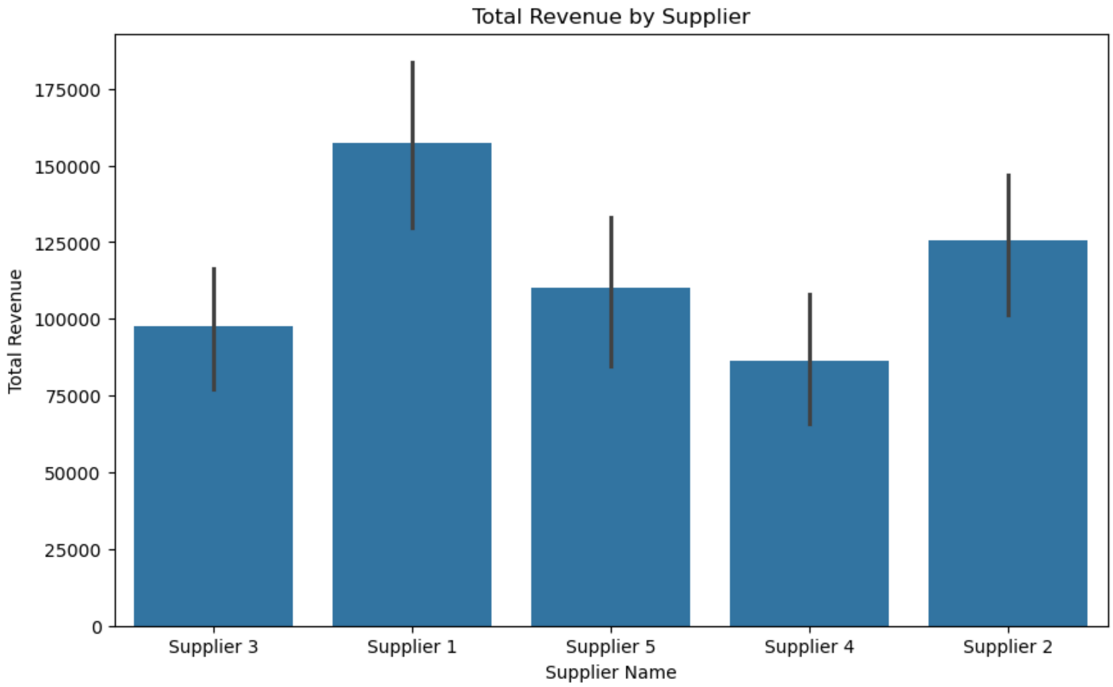
[37]:
```
plt.figure(figsize=(10, 6))
sns.barplot(data=df, x='Gender', y='Revenue generated', estimator=sum)
plt.title('Total Revenue by Gender')
plt.xlabel('Gender')
plt.ylabel('Total Revenue')
plt.show()
```



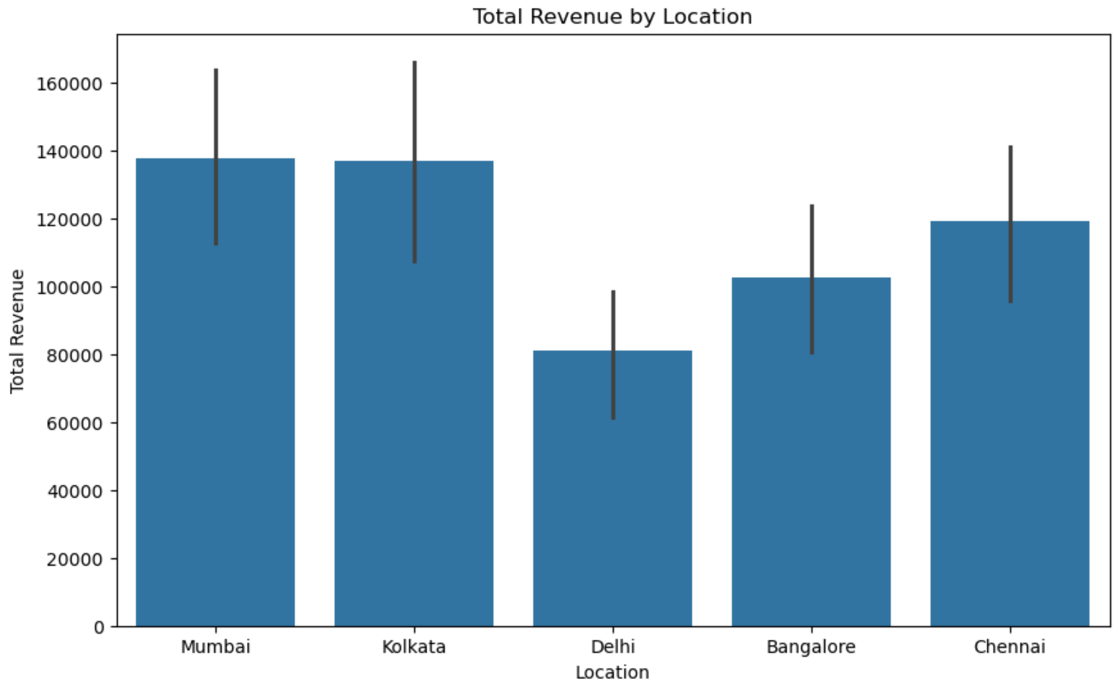
[38]:
```
plt.figure(figsize=(10, 6))
sns.barplot(data=df, x='Shipping carriers', y='Revenue generated',␣
 ↪estimator=sum)
plt.title('Total Revenue by Shipping Carriers')
plt.xlabel('Shipping Carriers')
plt.ylabel('Total Revenue')
plt.show()
```



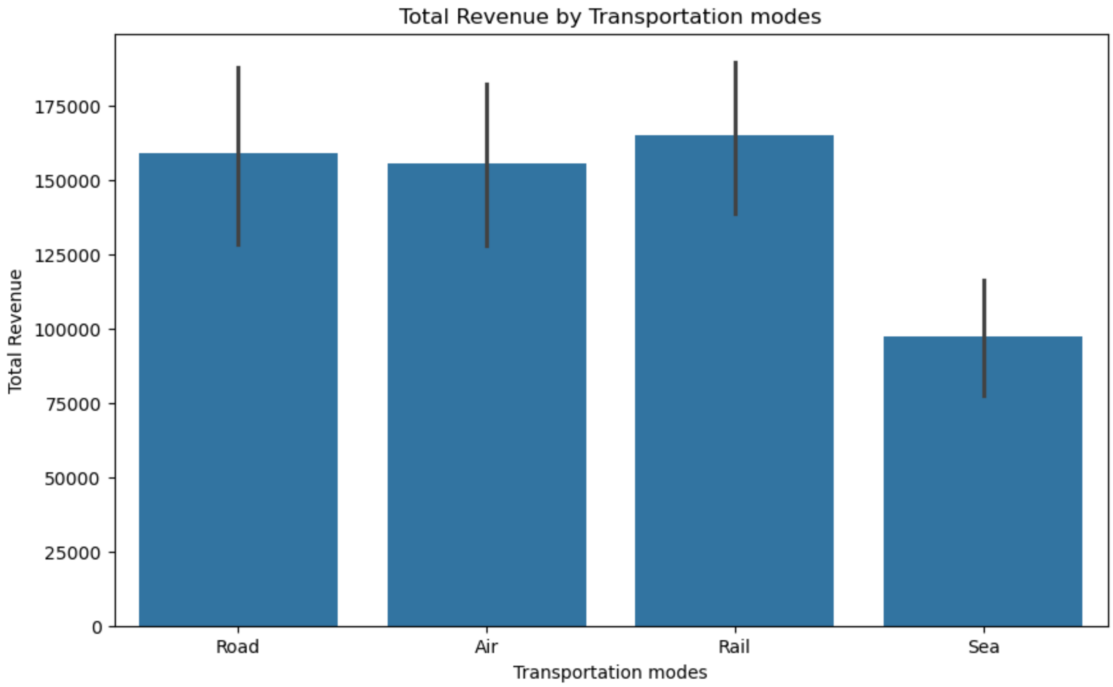
[39]:
```
plt.figure(figsize=(10, 6))
sns.barplot(data=df, x='Supplier name', y='Revenue generated', estimator=sum)
plt.title('Total Revenue by Supplier')
plt.xlabel('Supplier Name')
plt.ylabel('Total Revenue')
plt.show()
```



[40]:
```
plt.figure(figsize=(10, 6))
sns.barplot(data=df, x='Location', y='Revenue generated', estimator=sum)
plt.title('Total Revenue by Location')
plt.xlabel('Location')
plt.ylabel('Total Revenue')
plt.show()
```



[41]:
```
plt.figure(figsize=(10, 6))
sns.barplot(data=df, x='Transportation modes', y='Revenue generated',␣
 ↪estimator=sum)
plt.title('Total Revenue by Transportation modes')
plt.xlabel('Transportation modes')
plt.ylabel('Total Revenue')
plt.show()
```



[74]:

```python
# What are the most common defect rates for each product type and their impact
↪on revenue?

df_grouped = df.groupby(["Product type", "Defect rates"]).agg(
    Defect_Count=("Product type", "count"),
    Avg_Revenue=("Revenue generated", "mean")
).reset_index()
df_grouped = df_grouped.sort_values(
    by=["Product type", "Defect rates"],
    ascending=[True, False])
print(df_grouped)
```

```
   Product type  Defect rates  Defect_Count  Avg_Revenue
25    cosmetics      4.754801             1  5910.885390
24    cosmetics      4.620546             1  7910.886916
23    cosmetics      3.878099             1  8318.903195
22    cosmetics      3.872048             1  7698.424766
21    cosmetics      3.541046             1  5149.998350
..          …           …            …          …
64     skincare      0.159486             1  8458.730878
63     skincare      0.131955             1  9473.798033
62     skincare      0.100683             1  8653.570926
61     skincare      0.045302             1  5521.205259
60     skincare      0.021170             1  6099.944116
```

```
[100 rows x 4 columns]
```

[75]:
```python
# Which Gender are associated with higher purchase volumes and revenues?

df_grouped = df.groupby("Gender").agg(
    Total_Products_Sold=("products_sold", "sum"),
    Total_Revenue=("Revenue generated", "sum")
).reset_index()
df_grouped_sorted = df_grouped.sort_values(by="Total_Revenue", ascending=False)
print(df_grouped_sorted)
```

```
      Gender  Total_Products_Sold  Total_Revenue
3    Unknown                15211  173090.133841
0     Female                12801  161514.489121
1       Male                 7507  126634.394260
2 Non-binary                10580  116365.801518
```

[76]:
```python
#How does customer gender affect product preference and purchasing patterns?

df["Revenue_Per_Product"] = df["Revenue generated"] / df["products_sold"]
df_grouped = df.groupby(["Gender", "Product type"]).agg(
    Total_Products_Sold=("products_sold", "sum"),
    Total_Revenue=("Revenue generated", "sum"),
   Revenue_Per_Product=("Revenue_Per_Product", "mean")
).reset_index()
df_grouped = df_grouped.sort_values(
    by=["Gender", "Total_Revenue"],
    ascending=[True, False]
)
print(df_grouped)
```

```
        Gender Product type  Total_Products_Sold  Total_Revenue  \
2       Female     skincare                 7853   79241.113641
0       Female    cosmetics                 4012   69548.542197
1       Female     haircare                  936   12724.833283
5         Male     skincare                 2911   54643.501453
4         Male     haircare                 2292   50599.927309
3         Male    cosmetics                 2304   21390.965498
8   Non-binary     skincare                 5153   51159.172773
7   Non-binary     haircare                 2820   38971.147085
6   Non-binary    cosmetics                 2607   26235.481660
10     Unknown     haircare                 7563   72159.482929
11     Unknown     skincare                 4814   56584.374266
9      Unknown    cosmetics                 2834   44346.276646

    Revenue_Per_Product
2             15.108744
0             53.125086
1            112.765780
```

```
5           37.957886
4           52.293366
3           10.908163
8           22.549902
7           51.357022
6           10.714195
10          92.359454
11          12.677659
9           27.079134
```

[77]:

```python
#What is Sales Trends Based on Customer Demographics?

df_grouped = df.groupby(["Gender", "Location"]).agg(
    Total_Products_Sold=("products_sold", "sum"),
    Total_Revenue=("Revenue generated", "sum")
).reset_index()
df_grouped_sorted = df_grouped.sort_values(by=["Gender", "Total_Revenue"],␣
 ↪ascending=[True, False])
print(df_grouped_sorted)
```

```
        Gender   Location  Total_Products_Sold  Total_Revenue
2       Female      Delhi                 4002   41346.579416
3       Female    Kolkata                 3989   32862.333872
0       Female  Bangalore                 1530   31984.736279
4       Female     Mumbai                 1660   29959.672241
1       Female    Chennai                 1620   25361.167313
5         Male  Bangalore                 1206   32013.151045
8         Male    Kolkata                 1901   28208.649917
6         Male    Chennai                  646   27967.825528
9         Male     Mumbai                 1895   25351.550977
7         Male      Delhi                 1859   13093.216793
14  Non-binary     Mumbai                 1953   38216.831087
10  Non-binary  Bangalore                 2021   36192.081924
13  Non-binary    Kolkata                 3258   27653.394244
11  Non-binary    Chennai                 1581    9595.839301
12  Non-binary      Delhi                 1767    4707.654962
16     Unknown    Chennai                 4921   56217.983608
18     Unknown    Kolkata                 3622   48353.172972
19     Unknown     Mumbai                 3918   44226.972575
17     Unknown      Delhi                 2087   21880.250054
15     Unknown  Bangalore                  663    2411.754632
```

[78]:

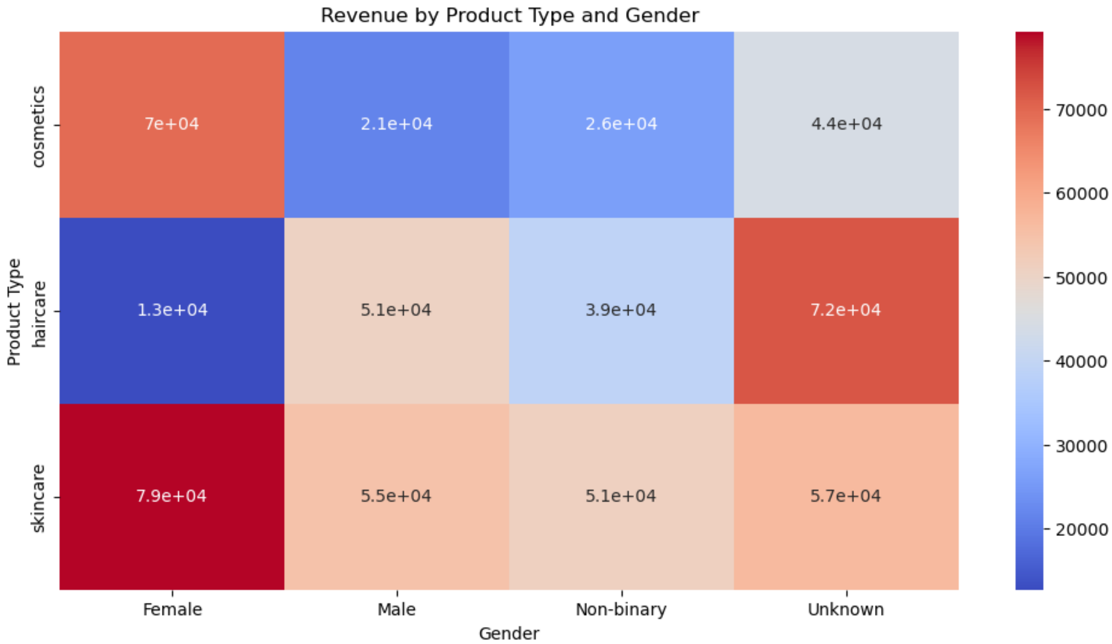
```python
#How do customer Gender influence purchasing behavior?

df_grouped = df.groupby("Gender").agg(
    Purchase_Frequency=("Gender", "count"),
    Total_Products_Sold=("products_sold", "sum"),
    Total_Revenue=("Revenue generated", "sum"),
```

```
    Avg_Revenue_Per_Product=("Revenue generated", lambda x: (x /␣
↪df["products_sold"]).mean())
).reset_index()
df_grouped_sorted = df_grouped.sort_values(by="Total_Revenue", ascending=False)
print(df_grouped_sorted)
```

```
      Gender  Purchase_Frequency  Total_Products_Sold  Total_Revenue  \
3    Unknown                  31                15211  173090.133841
0     Female                  25                12801  161514.489121
1       Male                  21                 7507  126634.394260
2 Non-binary                  23                10580  116365.801518

   Avg_Revenue_Per_Product
3                54.485312
0                38.127844
1                39.631977
2                28.744307
```

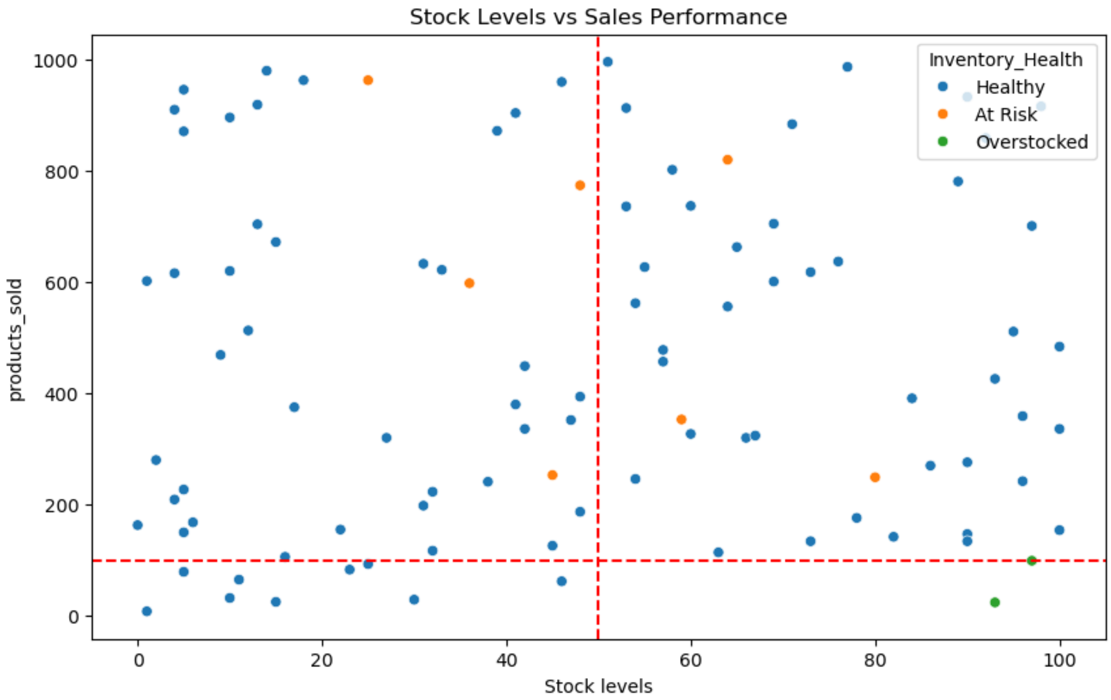
[42]:
```
revenue_chart = px.line(df, x='SKU',
                        y='Revenue generated',
                        title='Revenue Generated by SKU')
revenue_chart.show()
```

[43]:
```
# Group by 'Product type' and 'Customer demographics' and calculate total␣
↪revenue
revenue_by_product_and_demographics = df.groupby(['Product type',␣
↪'Gender'])['Revenue generated'].sum().unstack()

# Visualize revenue by product type and customer demographics
plt.figure(figsize=(12, 6))
sns.heatmap(revenue_by_product_and_demographics, annot=True, cmap='coolwarm')
plt.title('Revenue by Product Type and Gender')
plt.xlabel('Gender')
plt.ylabel('Product Type')
plt.show()
```



[44]:
```python
# Inventory Turnover Analysis
df['Inventory_Turnover'] = df['products_sold'] / df['Stock levels']

# Classify products
df['Inventory_Health'] = np.where(
    (df['Stock levels'] > 50) & (df['products_sold'] < 100),
    'Overstocked',
    np.where(df['Availability'] < 10, 'At Risk', 'Healthy')
)

# Visualize
plt.figure(figsize=(10,6))
sns.scatterplot(data=df, x='Stock levels', y='products_sold',␣
 ↪hue='Inventory_Health')
plt.title('Stock Levels vs Sales Performance')
plt.axhline(y=100, color='r', linestyle='--')
plt.axvline(x=50, color='r', linestyle='--')
```

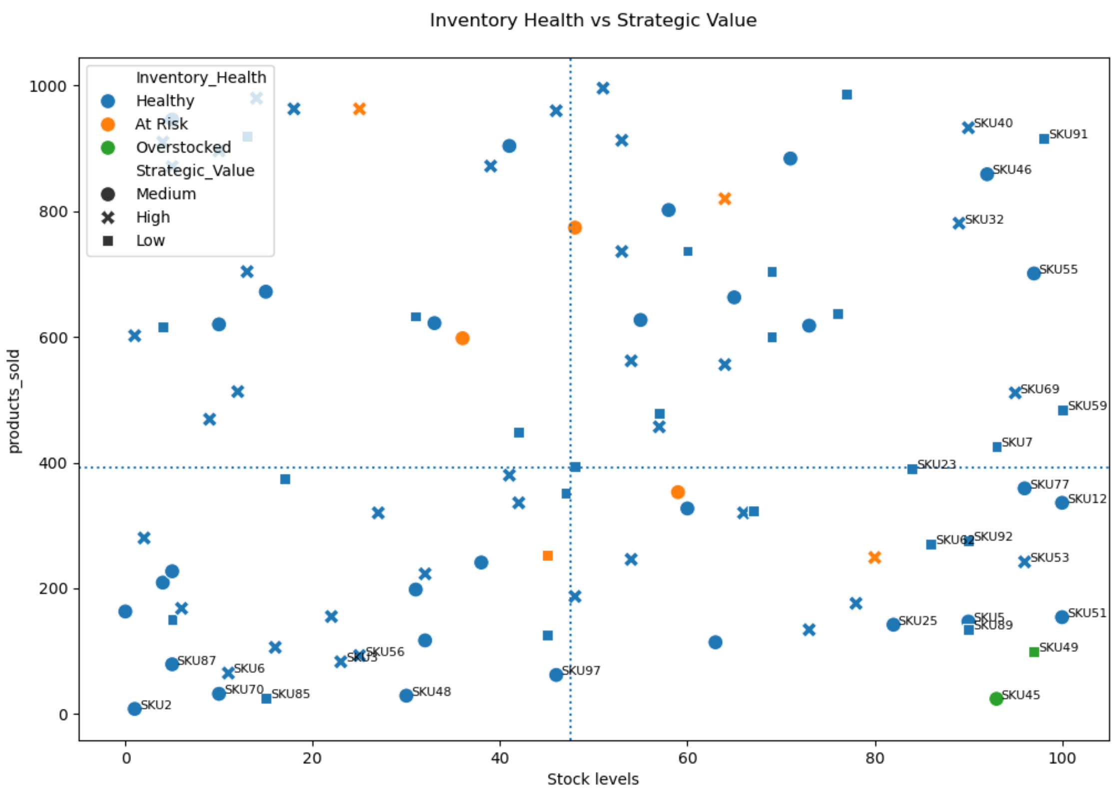
[44]:
```
<matplotlib.lines.Line2D at 0x28794b93650>
```



[45]:

```python
# Add synthetic product categories for practice
product_priority = {
    'skincare': 'High',
    'haircare': 'Medium',
    'cosmetics': 'Low'
}
df['Strategic_Value'] = df['Product type'].map(product_priority)

# Enhanced visualization
plt.figure(figsize=(12,8))
scatter = sns.scatterplot(
    data=df,
    x='Stock levels',
    y='products_sold',
    hue='Inventory_Health',
    style='Strategic_Value',
    s=100
)
plt.title('Inventory Health vs Strategic Value', pad=20)
plt.axvline(x=df['Stock levels'].median(), ls=':')
plt.axhline(y=df['products_sold'].median(), ls=':')

# Add annotations for extreme points
for line in range(0,df.shape[0]):
```

```
    if df['products_sold'][line] < 100 or df['Stock levels'][line] > 80:
        scatter.text(
            df['Stock levels'][line]+0.5,
            df['products_sold'][line],
            df['SKU'][line],
            horizontalalignment='left',
            size=8
        )
```



[46]:
```
# Step 1: Define action rules
# Broaden the "Increase reorder" condition to catch more fast-moving items
conditions = [
    (df['Inventory_Health'] == 'Overstocked') & (df['Strategic_Value'] ==␣
 ↪'High'),
    (df['Inventory_Health'] == 'At Risk') & (df['Strategic_Value'].
 ↪isin(['High', 'Medium'])),  # Expanded priority
    (df['products_sold'] > df['products_sold'].quantile(0.75)) & (df['Stock␣
 ↪levels'] < 50)  # More inclusive
]

actions = [
```

```python
    'Reduce stock gradually',
    'Emergency replenishment',
    'Increase reorder frequency'
]

# Step 2: Apply segmentation
df['Action_Plan'] = np.select(conditions, actions, default='Monitor')

# Step 3: Generate prioritized report
action_report = df[['SKU', 'Product type', 'Stock levels', 'products_sold',␣
↪'Strategic_Value', 'Action_Plan']]
print(action_report.sort_values(['Strategic_Value', 'products_sold'],␣
↪ascending=[False, False]).head(10))
```

```
      SKU Product type  Stock levels  products_sold Strategic_Value  \
78  SKU78     haircare             5            946          Medium
74  SKU74     haircare            41            904          Medium
22  SKU22     haircare            71            884          Medium
46  SKU46     haircare            92            859          Medium
0    SKU0     haircare            58            802          Medium
81  SKU81     haircare            48            774          Medium
55  SKU55     haircare            97            701          Medium
95  SKU95     haircare            15            672          Medium
83  SKU83     haircare            65            663          Medium
99  SKU99     haircare            55            627          Medium

                   Action_Plan
78  Increase reorder frequency
74  Increase reorder frequency
22                     Monitor
46                     Monitor
0                      Monitor
81     Emergency replenishment
55                     Monitor
95                     Monitor
83                     Monitor
99                     Monitor
```

[47]:
```python
# Step 1: Calculate key metrics
inventory_summary = df.groupby(['Product type', 'Inventory_Health']).agg(
    Total_Inventory_Value=('Price', lambda x: (x * df.loc[x.index, 'Stock␣
↪levels']).sum()),
    Avg_Monthly_Sales=('products_sold', 'mean'),
    Coverage_Days=('products_sold', lambda x: (df.loc[x.index, 'Stock levels'] /
↪ (x / 30)).mean())
).reset_index()
```

```
# Step 2: Add optimization targets (example rules)
inventory_summary['Target_Coverage'] = inventory_summary['Product type'].map({
    'skincare': 45,  # High priority → higher safety stock
    'haircare': 30,
    'cosmetics': 20
})

# Step 3: Calculate potential reductions
inventory_summary['Excess_Days'] = np.where(
    inventory_summary['Coverage_Days'] > inventory_summary['Target_Coverage'],
    inventory_summary['Coverage_Days'] - inventory_summary['Target_Coverage'],
    0
)
inventory_summary['Shortage_Days'] = np.where(
    inventory_summary['Coverage_Days'] < inventory_summary['Target_Coverage'],
    inventory_summary['Target_Coverage'] - inventory_summary['Coverage_Days'],
    0
)
print("\nInventory Optimization Potential:")
print(inventory_summary[['Product type', 'Inventory_Health', 'Coverage_Days',␣
↪'Target_Coverage', 'Excess_Days','Shortage_Days']])
```

```
Inventory Optimization Potential:
  Product type Inventory_Health  Coverage_Days  Target_Coverage  Excess_Days  \
0    cosmetics          At Risk       5.335968               20     0.000000
1    cosmetics          Healthy       5.418444               20     0.000000
2    cosmetics      Overstocked      29.393939               20     9.393939
3     haircare          At Risk       2.893550               30     0.000000
4     haircare          Healthy       6.890465               30     0.000000
5     haircare      Overstocked     116.250000               30    86.250000
6     skincare          At Risk       4.252945               45     0.000000
7     skincare          Healthy       3.812350               45     0.000000

   Shortage_Days
0      14.664032
1      14.581556
2       0.000000
3      27.106450
4      23.109535
5       0.000000
6      40.747055
7      41.187650
```

[48]:
```
print(df['Action_Plan'].value_counts())
```

```
Action_Plan
Monitor                       84
```

```
Increase reorder frequency    10
Emergency replenishment        6
Name: count, dtype: int64
```

[49]:
```
print(df[df['Action_Plan'] == 'Emergency replenishment'][['SKU', 'Product␣
↪type', 'Strategic_Value']])
```

```
      SKU Product type Strategic_Value
13  SKU13     skincare            High
26  SKU26     haircare          Medium
37  SKU37     skincare            High
43  SKU43     haircare          Medium
52  SKU52     skincare            High
81  SKU81     haircare          Medium
```

[50]:
```
!pip install ipywidgets
```

```
Requirement already satisfied: ipywidgets in c:\users\mahmo\anaconda3\lib\site-
packages (8.1.5)
Requirement already satisfied: comm>=0.1.3 in c:\users\mahmo\anaconda3\lib\site-
packages (from ipywidgets) (0.2.1)
Requirement already satisfied: ipython>=6.1.0 in
c:\users\mahmo\anaconda3\lib\site-packages (from ipywidgets) (8.27.0)
Requirement already satisfied: traitlets>=4.3.1 in
c:\users\mahmo\anaconda3\lib\site-packages (from ipywidgets) (5.14.3)
Requirement already satisfied: widgetsnbextension~=4.0.12 in
c:\users\mahmo\anaconda3\lib\site-packages (from ipywidgets) (4.0.13)
Requirement already satisfied: jupyterlab_widgets~=3.0.12 in
c:\users\mahmo\anaconda3\lib\site-packages (from ipywidgets) (3.0.13)
Requirement already satisfied: decorator in c:\users\mahmo\anaconda3\lib\site-
packages (from ipython>=6.1.0->ipywidgets) (5.1.1)
Requirement already satisfied: jedi>=0.16 in c:\users\mahmo\anaconda3\lib\site-
packages (from ipython>=6.1.0->ipywidgets) (0.19.1)
Requirement already satisfied: matplotlib-inline in
c:\users\mahmo\anaconda3\lib\site-packages (from ipython>=6.1.0->ipywidgets)
(0.1.6)
Requirement already satisfied: prompt-toolkit<3.1.0,>=3.0.41 in
c:\users\mahmo\anaconda3\lib\site-packages (from ipython>=6.1.0->ipywidgets)
(3.0.43)
Requirement already satisfied: pygments>=2.4.0 in
c:\users\mahmo\anaconda3\lib\site-packages (from ipython>=6.1.0->ipywidgets)
(2.15.1)
Requirement already satisfied: stack-data in c:\users\mahmo\anaconda3\lib\site-
packages (from ipython>=6.1.0->ipywidgets) (0.2.0)
Requirement already satisfied: colorama in c:\users\mahmo\anaconda3\lib\site-
packages (from ipython>=6.1.0->ipywidgets) (0.4.6)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in
c:\users\mahmo\anaconda3\lib\site-packages (from
jedi>=0.16->ipython>=6.1.0->ipywidgets) (0.8.3)
```

```
Requirement already satisfied: wcwidth in c:\users\mahmo\anaconda3\lib\site-
packages (from prompt-toolkit<3.1.0,>=3.0.41->ipython>=6.1.0->ipywidgets)
(0.2.5)
Requirement already satisfied: executing in c:\users\mahmo\anaconda3\lib\site-
packages (from stack-data->ipython>=6.1.0->ipywidgets) (0.8.3)
Requirement already satisfied: asttokens in c:\users\mahmo\anaconda3\lib\site-
packages (from stack-data->ipython>=6.1.0->ipywidgets) (2.0.5)
Requirement already satisfied: pure-eval in c:\users\mahmo\anaconda3\lib\site-
packages (from stack-data->ipython>=6.1.0->ipywidgets) (0.2.2)
Requirement already satisfied: six in c:\users\mahmo\anaconda3\lib\site-packages
(from asttokens->stack-data->ipython>=6.1.0->ipywidgets) (1.16.0)
```

[51]:
```
# Requires ipywidgets: !pip install ipywidgets
from ipywidgets import interact

def inspect_product(SKU=df['SKU'].sort_values().unique()):
    product = df[df['SKU'] == SKU].iloc[0]

    print(f"\n--- {SKU} ---")
    print(f"Type: {product['Product type']} (Priority:␣
 ↪{product['Strategic_Value']})")
    print(f"Stock: {product['Stock levels']} units | Sold:␣
 ↪{product['products_sold']} units")
    print(f"Status: {product['Inventory_Health']} → {product['Action_Plan']}")

    # Mini visualization
    plt.figure(figsize=(4, 3))
    plt.barh(['Stock', 'Sales'], [product['Stock levels'],␣
 ↪product['products_sold']], color=['skyblue', 'salmon'])
    plt.title('Stock vs Demand')
    plt.show()

interact(inspect_product)
```

```
interactive(children=(Dropdown(description='SKU', options=('SKU0', 'SKU1',␣
 ↪'SKU10', 'SKU11', 'SKU12', 'SKU13',…
```

[51]:
```
<function __main__.inspect_product(SKU=array(['SKU0', 'SKU1', 'SKU10', 'SKU11',
'SKU12', 'SKU13', 'SKU14',
        'SKU15', 'SKU16', 'SKU17', 'SKU18', 'SKU19', 'SKU2', 'SKU20',
        'SKU21', 'SKU22', 'SKU23', 'SKU24', 'SKU25', 'SKU26', 'SKU27',
        'SKU28', 'SKU29', 'SKU3', 'SKU30', 'SKU31', 'SKU32', 'SKU33',
        'SKU34', 'SKU35', 'SKU36', 'SKU37', 'SKU38', 'SKU39', 'SKU4',
        'SKU40', 'SKU41', 'SKU42', 'SKU43', 'SKU44', 'SKU45', 'SKU46',
        'SKU47', 'SKU48', 'SKU49', 'SKU5', 'SKU50', 'SKU51', 'SKU52',
        'SKU53', 'SKU54', 'SKU55', 'SKU56', 'SKU57', 'SKU58', 'SKU59',
        'SKU6', 'SKU60', 'SKU61', 'SKU62', 'SKU63', 'SKU64', 'SKU65',
        'SKU66', 'SKU67', 'SKU68', 'SKU69', 'SKU7', 'SKU70', 'SKU71',
```

```
       'SKU72', 'SKU73', 'SKU74', 'SKU75', 'SKU76', 'SKU77', 'SKU78',
       'SKU79', 'SKU8', 'SKU80', 'SKU81', 'SKU82', 'SKU83', 'SKU84',
       'SKU85', 'SKU86', 'SKU87', 'SKU88', 'SKU89', 'SKU9', 'SKU90',
       'SKU91', 'SKU92', 'SKU93', 'SKU94', 'SKU95', 'SKU96', 'SKU97',
       'SKU98', 'SKU99'], dtype=object))>
```

[52]:
```
%matplotlib inline
```

[53]:
```
def inspect_product(SKU=df['SKU'].sort_values().unique()):  # Fixed parentheses
    product = df[df['SKU'] == SKU].iloc[0]

    print(f"\n--- {SKU} ---")  # Proper f-string
    print(f"Type: {product['Product type']} (Priority:␣
↪{product['Strategic_Value']})")  # Case match
    print(f"Stock: {product['Stock levels']} units | Sold:␣
↪{product['products_sold']} units")
    print(f"Status: {product['Inventory_Health']} → {product['Action_Plan']}") ␣
↪# Consistent arrow

    plt.figure(figsize=(4, 3))
    plt.barh(['Stock', 'Sales'], [product['Stock levels'],␣
↪product['products_sold']], color=['skyblue', 'salmon'])
    plt.title('Stock vs Demand')
    plt.show()

interact(inspect_product)
```

```
interactive(children=(Dropdown(description='SKU', options=('SKU0', 'SKU1',␣
↪'SKU10', 'SKU11', 'SKU12', 'SKU13',…
```

[53]:
```
<function __main__.inspect_product(SKU=array(['SKU0', 'SKU1', 'SKU10', 'SKU11',
'SKU12', 'SKU13', 'SKU14',
       'SKU15', 'SKU16', 'SKU17', 'SKU18', 'SKU19', 'SKU2', 'SKU20',
       'SKU21', 'SKU22', 'SKU23', 'SKU24', 'SKU25', 'SKU26', 'SKU27',
       'SKU28', 'SKU29', 'SKU3', 'SKU30', 'SKU31', 'SKU32', 'SKU33',
       'SKU34', 'SKU35', 'SKU36', 'SKU37', 'SKU38', 'SKU39', 'SKU4',
       'SKU40', 'SKU41', 'SKU42', 'SKU43', 'SKU44', 'SKU45', 'SKU46',
       'SKU47', 'SKU48', 'SKU49', 'SKU5', 'SKU50', 'SKU51', 'SKU52',
       'SKU53', 'SKU54', 'SKU55', 'SKU56', 'SKU57', 'SKU58', 'SKU59',
       'SKU6', 'SKU60', 'SKU61', 'SKU62', 'SKU63', 'SKU64', 'SKU65',
       'SKU66', 'SKU67', 'SKU68', 'SKU69', 'SKU7', 'SKU70', 'SKU71',
       'SKU72', 'SKU73', 'SKU74', 'SKU75', 'SKU76', 'SKU77', 'SKU78',
       'SKU79', 'SKU8', 'SKU80', 'SKU81', 'SKU82', 'SKU83', 'SKU84',
       'SKU85', 'SKU86', 'SKU87', 'SKU88', 'SKU89', 'SKU9', 'SKU90',
       'SKU91', 'SKU92', 'SKU93', 'SKU94', 'SKU95', 'SKU96', 'SKU97',
       'SKU98', 'SKU99'], dtype=object))>
```

[54]:
```python
%matplotlib inline
from ipywidgets import interact

def inspect_product(SKU=df['SKU'].sort_values().unique()):  # Fixed parentheses
    product = df[df['SKU'] == SKU].iloc[0]

    print(f"\n--- {SKU} ---")  # Correct f-string
    print(f"Type: {product['Product type']} (Priority:␣
↪{product['Strategic_Value']})")
    print(f"Stock: {product['Stock levels']} units | Sold:␣
↪{product['products_sold']} units")
    print(f"Status: {product['Inventory_Health']} → {product['Action_Plan']}") ␣
↪# Consistent arrow

    plt.figure(figsize=(4, 3))
    plt.barh(['Stock', 'Sales'], [product['Stock levels'],␣
↪product['products_sold']], color=['skyblue', 'salmon'])
    plt.title('Stock vs Demand')
    plt.show()

interact(inspect_product)
```

```
interactive(children=(Dropdown(description='SKU', options=('SKU0', 'SKU1',␣
↪'SKU10', 'SKU11', 'SKU12', 'SKU13',…
```

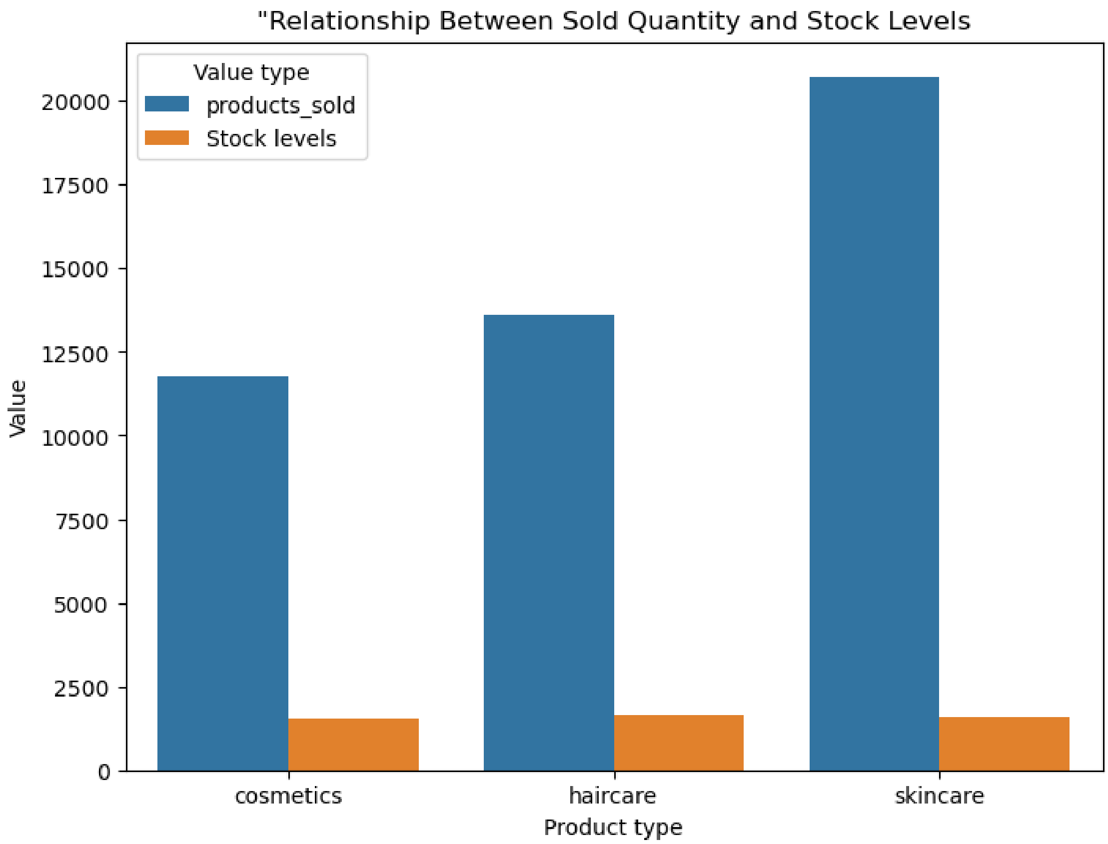
[54]:
```
<function __main__.inspect_product(SKU=array(['SKU0', 'SKU1', 'SKU10', 'SKU11',
'SKU12', 'SKU13', 'SKU14',
       'SKU15', 'SKU16', 'SKU17', 'SKU18', 'SKU19', 'SKU2', 'SKU20',
       'SKU21', 'SKU22', 'SKU23', 'SKU24', 'SKU25', 'SKU26', 'SKU27',
       'SKU28', 'SKU29', 'SKU3', 'SKU30', 'SKU31', 'SKU32', 'SKU33',
       'SKU34', 'SKU35', 'SKU36', 'SKU37', 'SKU38', 'SKU39', 'SKU4',
       'SKU40', 'SKU41', 'SKU42', 'SKU43', 'SKU44', 'SKU45', 'SKU46',
       'SKU47', 'SKU48', 'SKU49', 'SKU5', 'SKU50', 'SKU51', 'SKU52',
       'SKU53', 'SKU54', 'SKU55', 'SKU56', 'SKU57', 'SKU58', 'SKU59',
       'SKU6', 'SKU60', 'SKU61', 'SKU62', 'SKU63', 'SKU64', 'SKU65',
       'SKU66', 'SKU67', 'SKU68', 'SKU69', 'SKU7', 'SKU70', 'SKU71',
       'SKU72', 'SKU73', 'SKU74', 'SKU75', 'SKU76', 'SKU77', 'SKU78',
       'SKU79', 'SKU8', 'SKU80', 'SKU81', 'SKU82', 'SKU83', 'SKU84',
       'SKU85', 'SKU86', 'SKU87', 'SKU88', 'SKU89', 'SKU9', 'SKU90',
       'SKU91', 'SKU92', 'SKU93', 'SKU94', 'SKU95', 'SKU96', 'SKU97',
       'SKU98', 'SKU99'], dtype=object))>
```

[70]:
```python
#What is the relationship between the number of products sold and stock levels␣
↪for each product type?

df_grouped = df.groupby("Product type")[["products_sold", "Stock levels"]].
↪sum().reset_index()
```

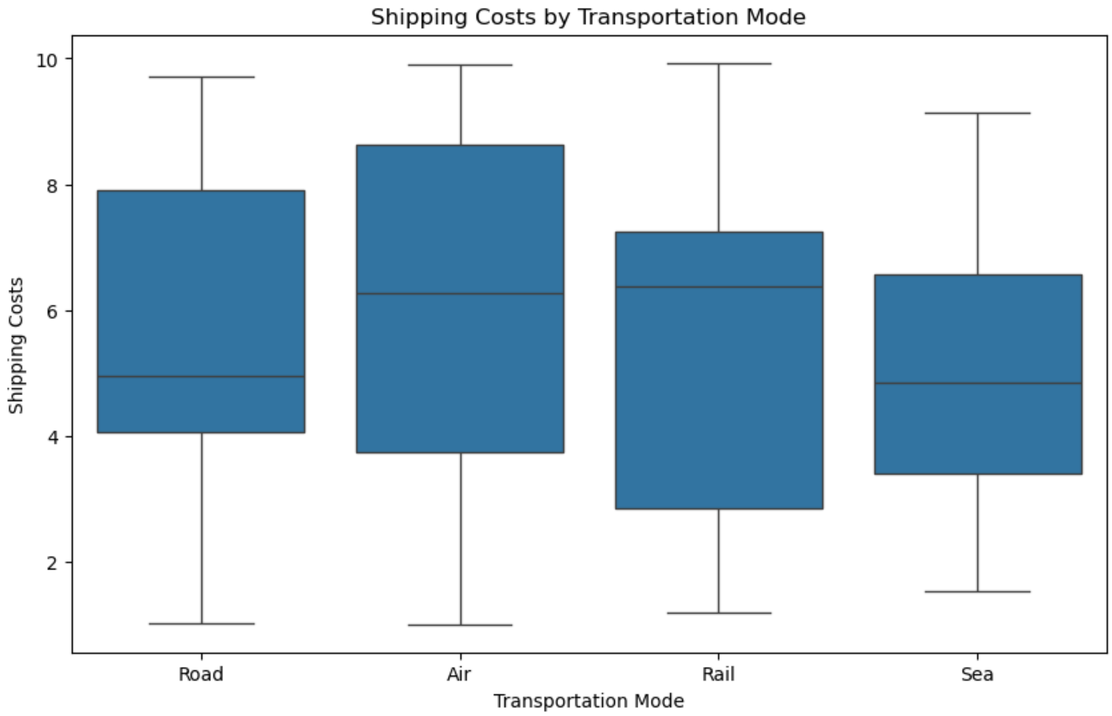
```python
df_grouped = df_grouped.sort_values(by="Product type")
print(df_grouped)
```

```
  Product type  products_sold  Stock levels
0    cosmetics          11757          1525
1     haircare          13611          1644
2     skincare          20731          1608
```

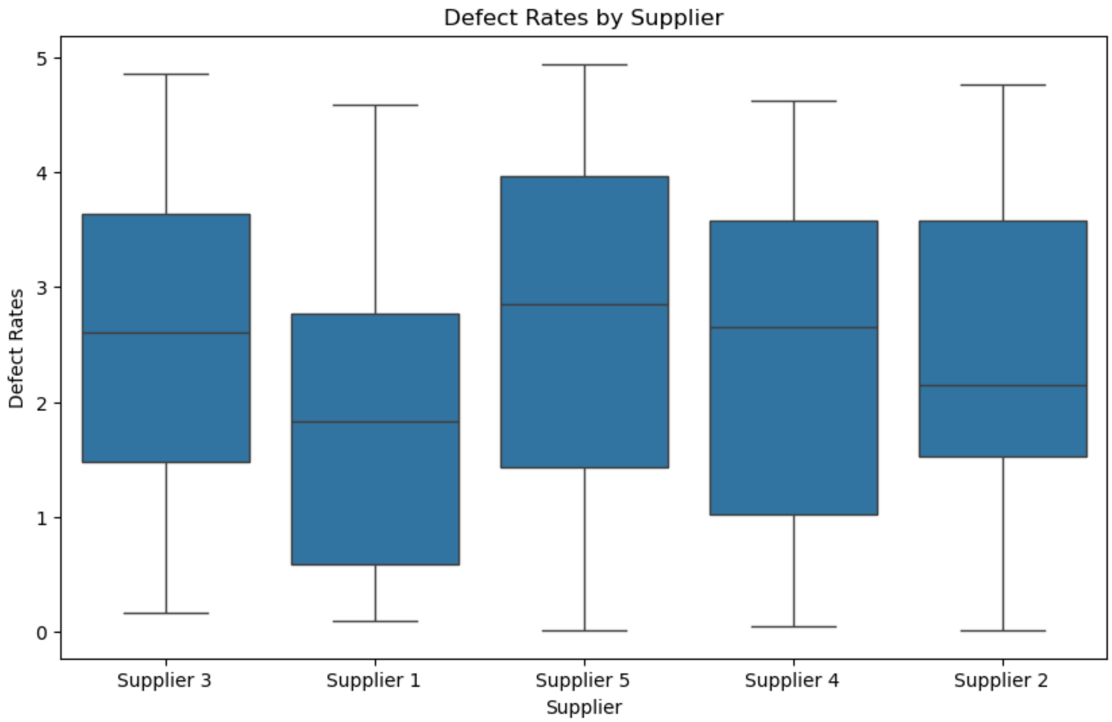
[71]:
```python
df_melted = df_grouped.melt(
    id_vars='Product type',
    value_vars=['products_sold', 'Stock levels'],
    var_name='Metric',
    value_name='Value'
)
plt.figure(figsize=(8, 6))
sns.barplot(data=df_melted, x='Product type', y='Value', hue='Metric')

plt.title('"Relationship Between Sold Quantity and Stock Levels')
plt.xlabel('Product type')
plt.ylabel('Value')
plt.legend(title='Value type')
plt.xticks(rotation=0)

plt.show()
```



[55]:
```python
# Shipping Costs by Transportation Mode
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='Transportation modes', y='Shipping costs')
plt.title('Shipping Costs by Transportation Mode')
plt.xlabel('Transportation Mode')
plt.ylabel('Shipping Costs')
plt.show()
```



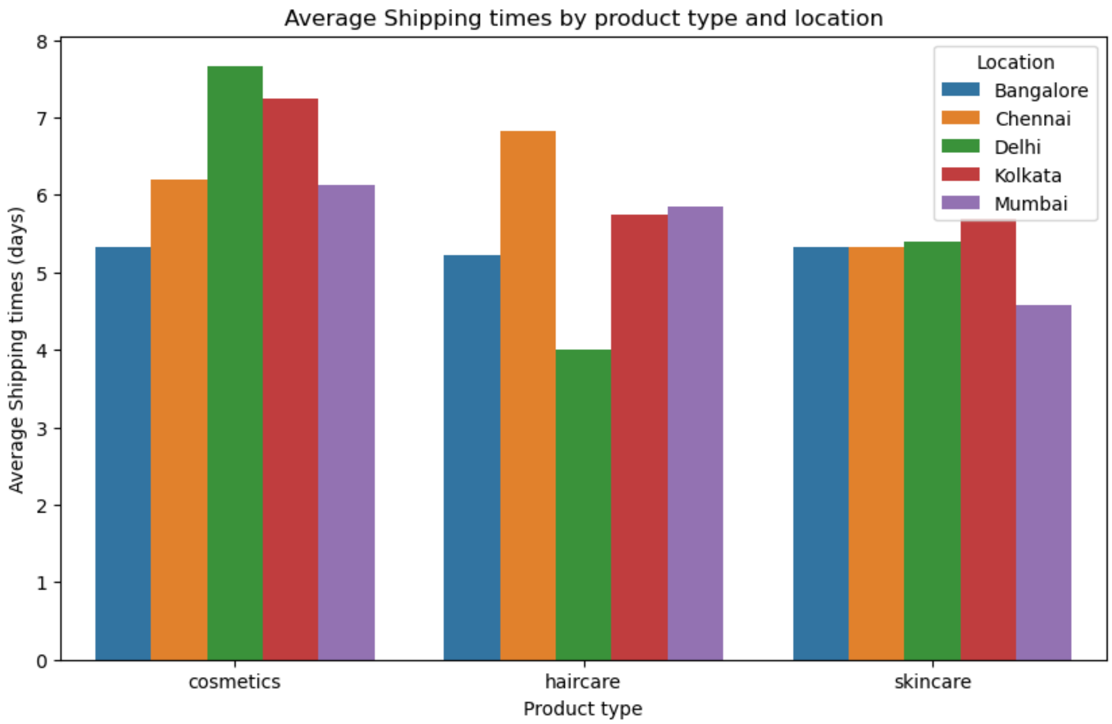
[56]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df, x='Supplier name', y='Defect rates')
plt.title('Defect Rates by Supplier')
plt.xlabel('Supplier')
plt.ylabel('Defect Rates')
plt.show()
```



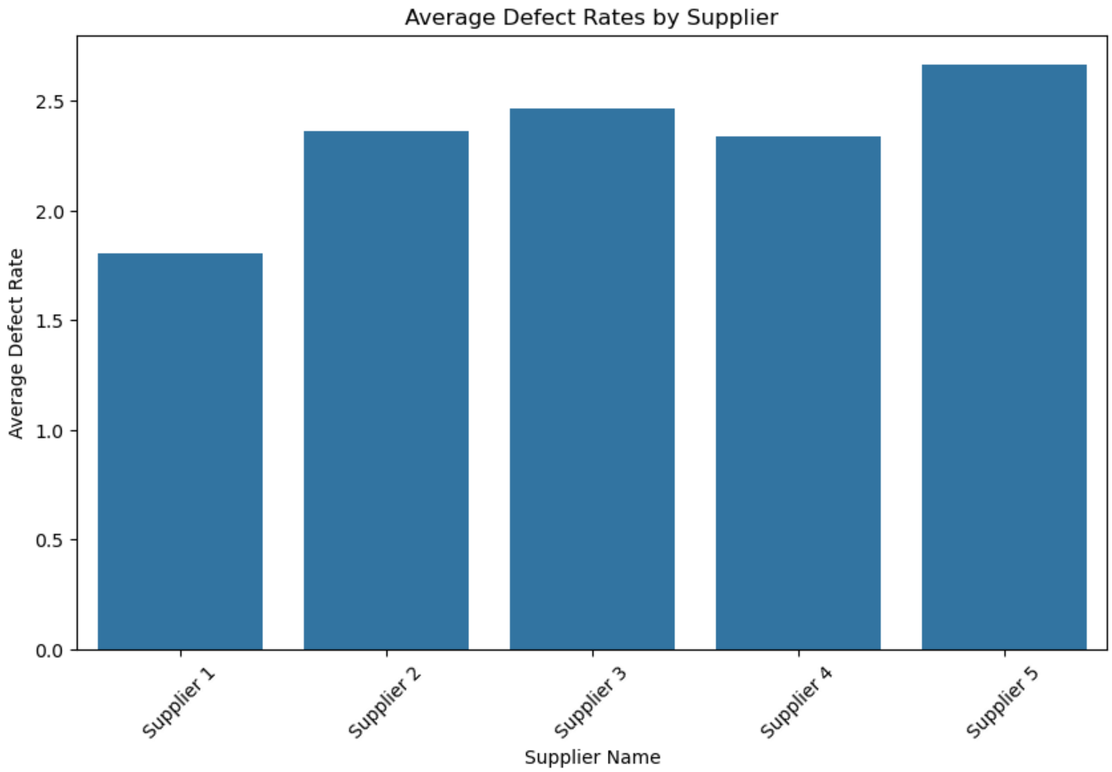
[72]:
```
#How do shipping times and costs vary by product type and location?

df_grouped = df.groupby(["Product type", "Location"])[["Shipping times",␣
↪"Shipping costs"]].mean().reset_index()
print(df_grouped)
```

```
   Product type   Location  Shipping times  Shipping costs
0     cosmetics  Bangalore        5.333333        6.058714
1     cosmetics    Chennai        6.200000        4.848736
2     cosmetics      Delhi        7.666667        6.005597
3     cosmetics    Kolkata        7.250000        7.062342
4     cosmetics     Mumbai        6.125000        6.357611
5      haircare  Bangalore        5.222222        5.720735
6      haircare    Chennai        6.833333        5.426831
7      haircare      Delhi        4.000000        5.869714
8      haircare    Kolkata        5.750000        5.816768
9      haircare     Mumbai        5.857143        6.686163
10     skincare  Bangalore        5.333333        5.634877
11     skincare    Chennai        5.333333        4.108546
12     skincare      Delhi        5.400000        3.307156
13     skincare    Kolkata        5.692308        5.327064
14     skincare     Mumbai        4.571429        5.687675
```

[73]:
```
plt.figure(figsize=(10, 6))
sns.barplot(data=df_grouped, x="Product type", y="Shipping times",␣
↪hue="Location")

plt.title("Average Shipping times by product type and location")
plt.xlabel("Product type")
plt.ylabel("Average Shipping times (days)")
plt.legend(title="Location")
plt.show()
```



[57]:
```
# Group by 'Supplier name' and calculate average defect rates
defect_rates_by_supplier = df.groupby('Supplier name')['Defect rates'].mean()

# Visualize defect rates by supplier
plt.figure(figsize=(10, 6))
sns.barplot(x=defect_rates_by_supplier.index, y=defect_rates_by_supplier.values)
plt.title('Average Defect Rates by Supplier')
plt.xlabel('Supplier Name')
plt.ylabel('Average Defect Rate')
plt.xticks(rotation=45)
plt.show()
```



[79]:
```
#Which suppliers have the shortest lead times and lowest manufacturing costs?

df_suppliers = df.groupby("Supplier name").agg(
    Avg_Lead_Time=("Lead time", "mean"),
    Avg_Cost_Per_Unit=("Manufacturing costs", "mean")
).reset_index()
df_suppliers = df_suppliers.sort_values(by=["Avg_Lead_Time",␣
 ↪"Avg_Cost_Per_Unit"], ascending=[True, True])
print(df_suppliers)
```

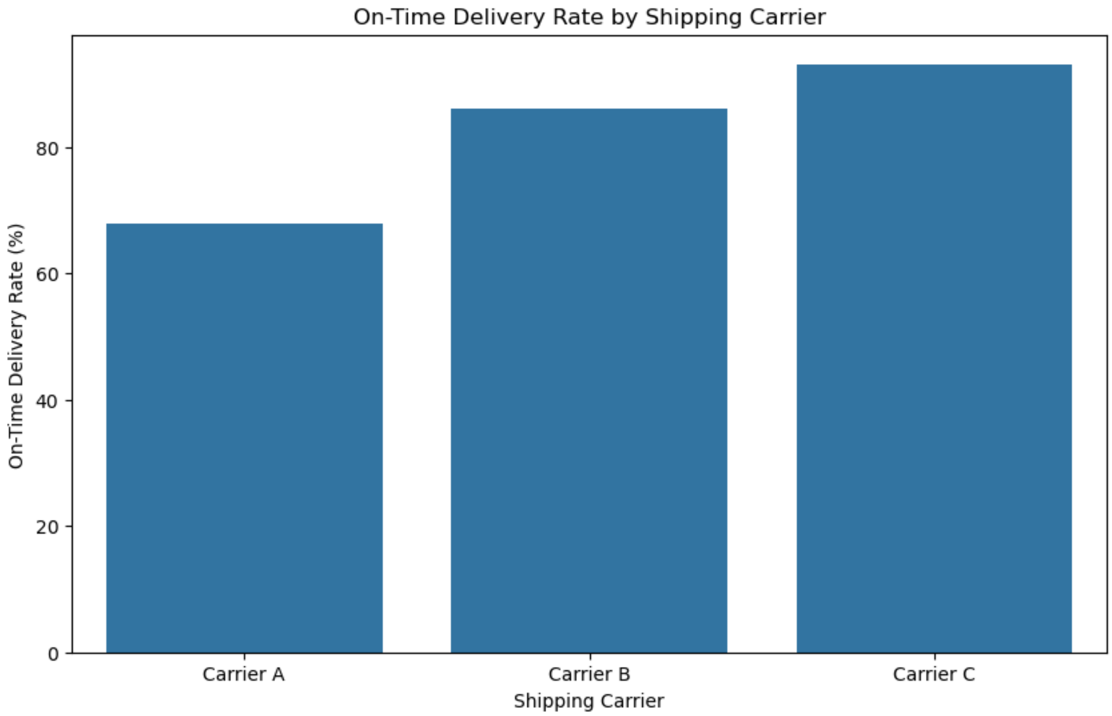
```
  Supplier name  Avg_Lead_Time  Avg_Cost_Per_Unit
0    Supplier 1      14.777778          45.254027
3    Supplier 4      15.222222          62.709727
4    Supplier 5      18.055556          44.768243
1    Supplier 2      18.545455          41.622514
2    Supplier 3      20.133333          43.634121
```

[80]:
```
#How do production volumes vary by supplier and location?

df_production = df.groupby(["Supplier name", "Location"]).agg(
    Total_Production=("Production volumes", "sum")
).reset_index()
```

```
df_production = df_production.sort_values(by="Total_Production",␣
  ↪ascending=False)
print(df_production)
```

```
   Supplier name   Location  Total_Production
18    Supplier 4    Kolkata              4425
7     Supplier 2      Delhi              4225
3     Supplier 1    Kolkata              4021
23    Supplier 5    Kolkata              3560
4     Supplier 1     Mumbai              3462
9     Supplier 2     Mumbai              3439
16    Supplier 4    Chennai              2854
19    Supplier 4     Mumbai              2683
6     Supplier 2    Chennai              2678
21    Supplier 5    Chennai              2423
0     Supplier 1  Bangalore              2349
5     Supplier 2  Bangalore              2236
24    Supplier 5     Mumbai              2196
11    Supplier 3    Chennai              2056
1     Supplier 1    Chennai              1973
13    Supplier 3    Kolkata              1918
2     Supplier 1      Delhi              1740
10    Supplier 3  Bangalore              1549
8     Supplier 2    Kolkata              1527
14    Supplier 3     Mumbai              1380
17    Supplier 4      Delhi              1126
12    Supplier 3      Delhi              1094
20    Supplier 5  Bangalore              1025
15    Supplier 4  Bangalore               668
22    Supplier 5      Delhi               177
```

[58]:
```
# Calculate on-time delivery rate by shipping carrier
df['On_Time_Delivery'] = df['Shipping times'] <= df['Lead times']
on_time_delivery_by_carrier = df.groupby('Shipping␣
  ↪carriers')['On_Time_Delivery'].mean() * 100

# Visualize on-time delivery rates by carrier
plt.figure(figsize=(10, 6))
sns.barplot(x=on_time_delivery_by_carrier.index, y=on_time_delivery_by_carrier.
  ↪values)
plt.title('On-Time Delivery Rate by Shipping Carrier')
plt.xlabel('Shipping Carrier')
plt.ylabel('On-Time Delivery Rate (%)')
plt.show()
```



[59]:
```
# Calculate cost efficiency (Revenue/Manufacturing Cost) and defect rates
supplier_metrics = df.groupby('Supplier name').agg({
    'Manufacturing costs': 'mean',
    'Defect rates': 'mean',
    'Revenue generated': 'sum',
    'Lead time': 'median'
}).reset_index()

supplier_metrics['Cost_Efficiency'] = supplier_metrics['Revenue generated'] /
 ↪supplier_metrics['Manufacturing costs']

# Visualize
fig = px.scatter(
    supplier_metrics,
    x='Defect rates',
    y='Cost_Efficiency',
    size='Revenue generated',
    color='Lead time',
    hover_name='Supplier name',
    title='Supplier Performance: Quality vs. Cost Efficiency'
)
fig.show()
```

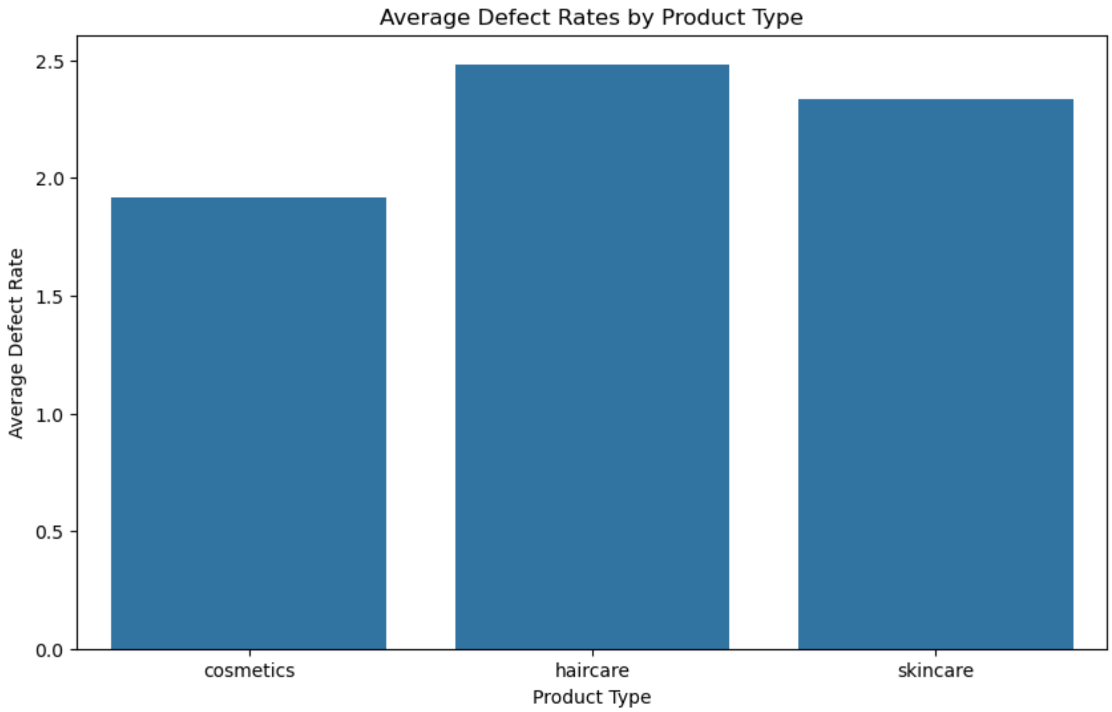
[60]:

```python
import plotly.express as px
import pandas as pd

# Calculate metrics
supplier_metrics = df.groupby('Supplier name').agg({
    'Manufacturing costs': 'mean',
    'Defect rates': 'mean',
    'Revenue generated': 'sum',
    'Lead time': 'median'
}).reset_index()

supplier_metrics['Cost_Efficiency'] = (
    supplier_metrics['Revenue generated'] / supplier_metrics['Manufacturing␣
 ↪costs']
)

# Add a composite score (weighted example: 60% cost efficiency, 40% quality)
supplier_metrics['Cost_Quality_Score'] = (
    0.6 * (supplier_metrics['Cost_Efficiency'] /␣
 ↪supplier_metrics['Cost_Efficiency'].max()) +
    0.4 * (1 - supplier_metrics['Defect rates'] / supplier_metrics['Defect␣
 ↪rates'].max())  # Lower defects = better
)

# Visualize with enhancements
fig = px.scatter(
    supplier_metrics,
    x='Defect rates',
    y='Cost_Efficiency',
    size='Revenue generated',
    color='Lead time',
    hover_name='Supplier name',
    hover_data={
        'Lead time': ':.1f days',  # Clarify units
        'Cost_Efficiency': ':.1f (Revenue/$Cost)',
        'Defect rates': ':.2%',
        'Cost_Quality_Score': ':.2f'  # Show score in hover
    },
    title='Supplier Performance: Quality (Defect Rates) vs. Cost Efficiency',
    labels={
        'Defect rates': 'Defect Rate (%)',
        'Cost_Efficiency': 'Cost Efficiency (Revenue per $1 Manufacturing␣
 ↪Cost)',
        'Lead time': 'Lead Time (days)'
    }
)
```

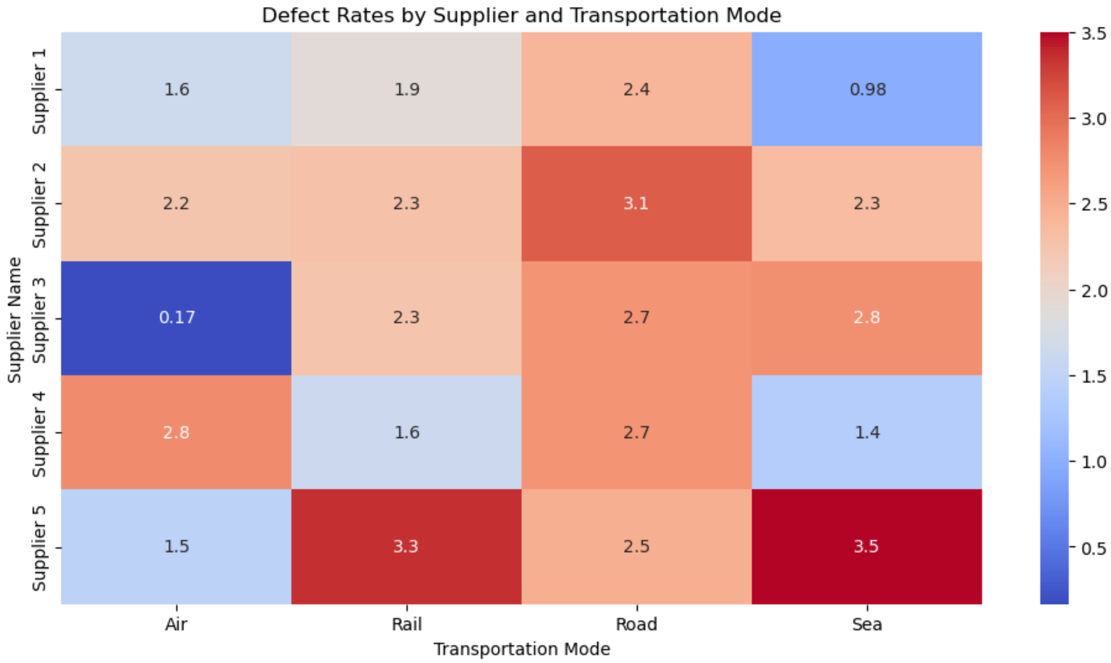
```
# Highlight Pareto frontier (optional)
fig.update_traces(
    marker=dict(line=dict(width=1, color='DarkSlateGrey')),
    selector=dict(mode='markers')
)

# Add reference lines/annotations for clarity
fig.add_hline(
    y=supplier_metrics['Cost_Efficiency'].median(),
    line_dash="dot", line_color="grey",
    annotation_text="Median Cost Efficiency"
)
fig.add_vline(
    x=supplier_metrics['Defect rates'].median(),
    line_dash="dot", line_color="grey",
    annotation_text="Median Defect Rate"
)

fig.show()

# Optional: Print top suppliers by composite score
print("Top Suppliers by Cost-Quality Score:")
print(
    supplier_metrics.sort_values('Cost_Quality_Score', ascending=False)
    [['Supplier name', 'Cost_Quality_Score', 'Defect rates', 'Cost_Efficiency']]
    .head(5)
)
```

```
Top Suppliers by Cost-Quality Score:
  Supplier name  Cost_Quality_Score  Defect rates  Cost_Efficiency
0    Supplier 1            0.729328      1.803630      3480.993954
1    Supplier 2            0.564998      2.362750      3014.412297
4    Supplier 5            0.424839      2.665408      2464.770923
2    Supplier 3            0.416274      2.465786      2241.273038
3    Supplier 4            0.286894      2.337397      1378.876390
```

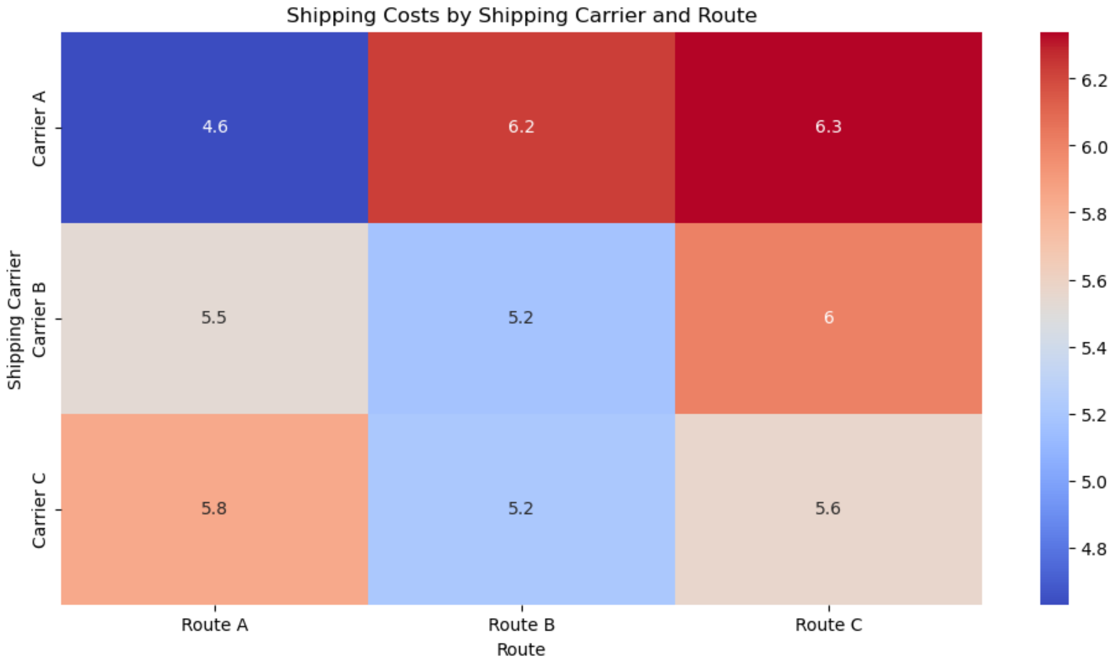
[61]:
```
# Group by 'Product type' and calculate average defect rates
defect_rates_by_product = df.groupby('Product type')['Defect rates'].mean()

# Visualize defect rates by product type
plt.figure(figsize=(10, 6))
sns.barplot(x=defect_rates_by_product.index, y=defect_rates_by_product.values)
plt.title('Average Defect Rates by Product Type')
plt.xlabel('Product Type')
plt.ylabel('Average Defect Rate')
plt.show()
```



[62]:
```
# Group by 'Supplier name' and 'Transportation modes' and calculate average␣
↪defect rates
defect_rates_by_supplier_and_transport = df.groupby(['Supplier name',␣
↪'Transportation modes'])['Defect rates'].mean().unstack()

# Visualize defect rates by supplier and transportation mode
plt.figure(figsize=(12, 6))
sns.heatmap(defect_rates_by_supplier_and_transport, annot=True, cmap='coolwarm')
plt.title('Defect Rates by Supplier and Transportation Mode')
plt.xlabel('Transportation Mode')
plt.ylabel('Supplier Name')
plt.show()
```

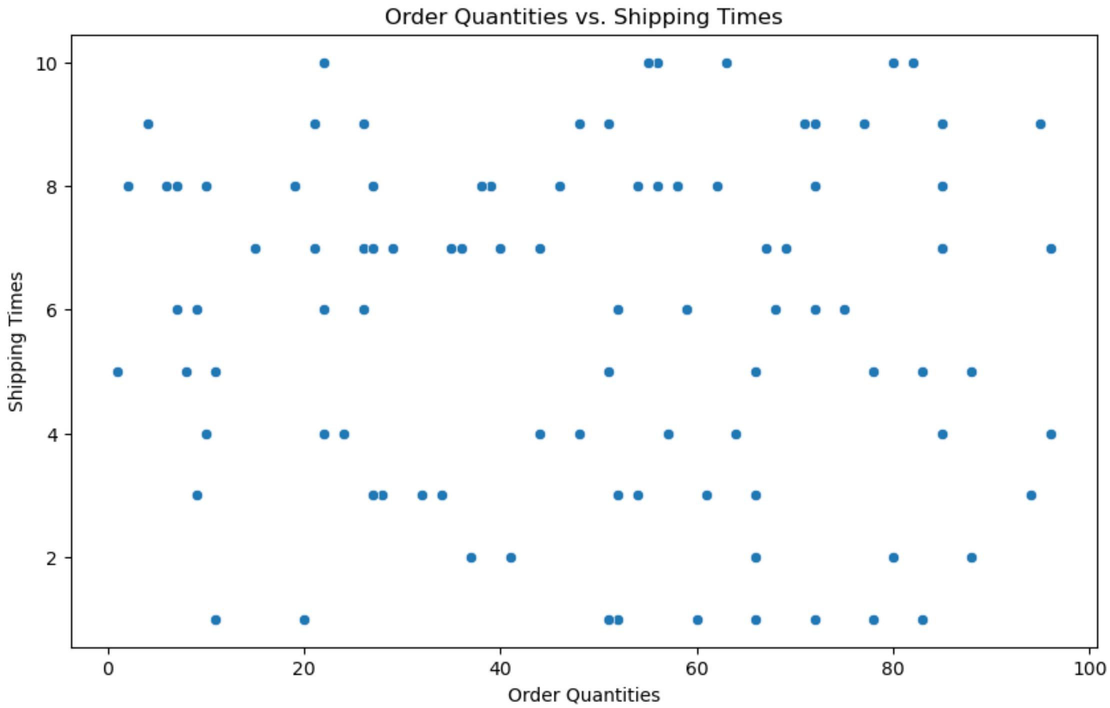


[63]:
```
# Group by 'Shipping carriers' and 'Routes' and calculate average shipping costs
shipping_costs_by_carrier_and_route = df.groupby(['Shipping carriers',␣
↪'Routes'])['Shipping costs'].mean().unstack()

# Visualize shipping costs by carrier and route
plt.figure(figsize=(12, 6))
sns.heatmap(shipping_costs_by_carrier_and_route, annot=True, cmap='coolwarm')
plt.title('Shipping Costs by Shipping Carrier and Route')
plt.xlabel('Route')
plt.ylabel('Shipping Carrier')
plt.show()
```



[64]:

```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='Order quantities', y='Shipping times')
plt.title('Order Quantities vs. Shipping Times')
plt.xlabel('Order Quantities')
plt.ylabel('Shipping Times')
plt.show()
```



[65]:
```
# Calculate cost per unit shipped
transport_stats = df.groupby('Transportation modes').agg({
    'Shipping costs': 'mean',
    'Defect rates': 'mean',
    'Shipping times': 'median'
}).reset_index()

# Visualize tradeoffs
fig = px.parallel_coordinates(
    transport_stats,
    color='Shipping costs',
    dimensions=['Shipping costs', 'Defect rates', 'Shipping times'],
    title='Transportation Mode Tradeoffs'
)
fig.show()
```

[66]:
```
import plotly.express as px
import pandas as pd

# --- Data Preparation ---
transport_stats = df.groupby('Transportation modes').agg({
    'Shipping costs': 'mean',
    'Defect rates': 'mean',
    'Shipping times': 'median',
```

```
    'Shipping carriers': 'first',  # Use existing column
    'Routes': 'first'  # Optional: Include routes
}).reset_index()

# Calculate "Total Cost of Defects" (example: $10 per defect)
transport_stats['Defect_cost_impact'] = transport_stats['Defect rates'] * 10

# --- Visualization: Parallel Coordinates ---
fig = px.parallel_coordinates(
    transport_stats,
    color='Shipping costs',
    dimensions=[
        'Transportation modes',  # Include this first for identification
        'Shipping costs',
        'Defect rates',
        'Shipping times',
        'Defect_cost_impact',
        'Shipping carriers',  # Show carrier info
        'Routes'  # Show route info
    ],
    labels={
        'Shipping costs': 'Cost ($/unit)',
        'Defect rates': 'Defect Rate (%)',
        'Shipping times': 'Time (days)',
        'Defect_cost_impact': 'Defect Cost ($/unit)',
        'Shipping carriers': 'Carrier',
        'Routes': 'Route',
        'Transportation modes': 'Transport Mode'
    },
    title='Transportation Mode Tradeoffs: Cost, Defects, Speed, and Defect Cost␣
 ↪Impact',
    color_continuous_scale=px.colors.sequential.Viridis
)

# Customize the display
fig.update_layout(
    hovermode='closest',
    dragmode='select'
)

# Manually set colors for specific transportation modes
# (Parallel coordinates doesn't support direct coloring by category, so we'll␣
 ↪use annotations)
fig.update_layout(
    annotations=[
        dict(
            x=0.05, y=1.05,
```

```
            xref='paper', yref='paper',
            text="Color Key: Warmer = Higher Shipping Costs",
            showarrow=False
        )
    ]
)

fig.show()

# --- Print Decision Matrix ---
print("\n**Transportation Mode Performance:**")
print(transport_stats[['Transportation modes', 'Shipping carriers', 'Routes',
                      'Shipping costs', 'Defect rates', 'Shipping times']].
 ↪sort_values('Shipping costs'))
```

```
**Transportation Mode Performance:**
  Transportation modes Shipping carriers   Routes  Shipping costs  \
3                  Sea         Carrier C  Route A        4.970294
1                 Rail         Carrier C  Route A        5.469098
2                 Road         Carrier B  Route B        5.542115
0                  Air         Carrier B  Route C        6.017839

   Defect rates  Shipping times
3      2.315281             8.0
1      2.318814             7.0
2      2.620938             4.0
0      1.823924             5.0
```

[81]:
```
#What are the average shipping costs and times for each transportation mode?

df_shipping = df.groupby("Transportation modes").agg(
    Avg_Shipping_Cost=("Shipping costs", "mean"),
    Avg_Shipping_Time=("Shipping times", "mean")
).reset_index()
df_shipping = df_shipping.sort_values(by="Avg_Shipping_Cost", ascending=False)
print(df_shipping)
```

```
  Transportation modes  Avg_Shipping_Cost  Avg_Shipping_Time
0                  Air           6.017839           5.115385
2                 Road           5.542115           4.724138
1                 Rail           5.469098           6.571429
3                  Sea           4.970294           7.117647
```

[82]:
```
#--How do shipping carriers affect shipping times and costs?

df_carrier_analysis = df.groupby("Shipping carriers").agg(
    Avg_Shipping_Cost=("Shipping costs", "mean"),
```

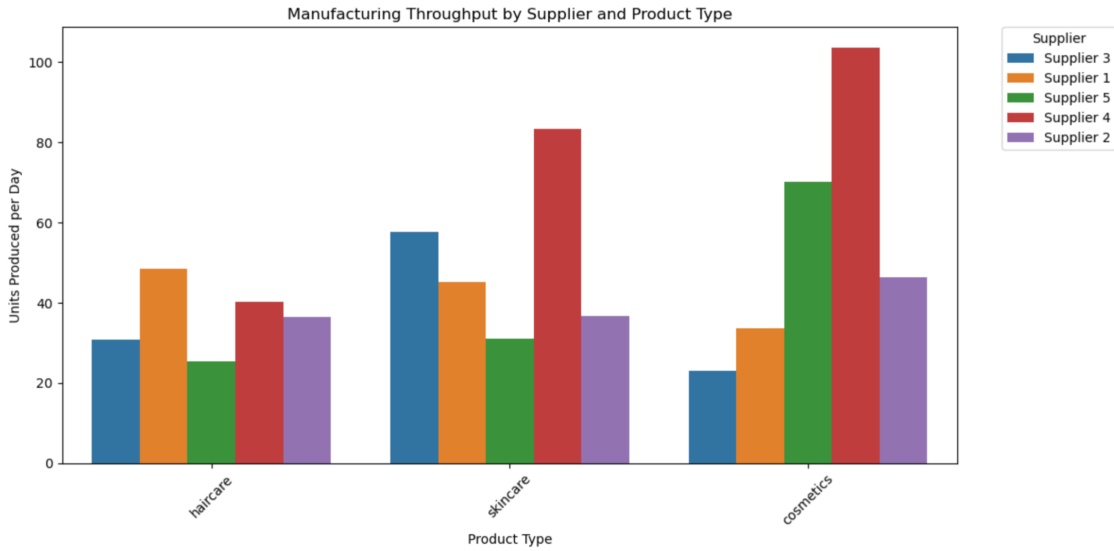
```
    Avg_Shipping_Time=("Shipping times", "mean")
).reset_index()
df_carrier_analysis = df_carrier_analysis.sort_values(by="Avg_Shipping_Cost",␣
↪ascending=False)
print(df_carrier_analysis)
```

```
  Shipping carriers  Avg_Shipping_Cost  Avg_Shipping_Time
2         Carrier C           5.599292           6.034483
0         Carrier A           5.554923           6.142857
1         Carrier B           5.509247           5.302326
```

[67]:

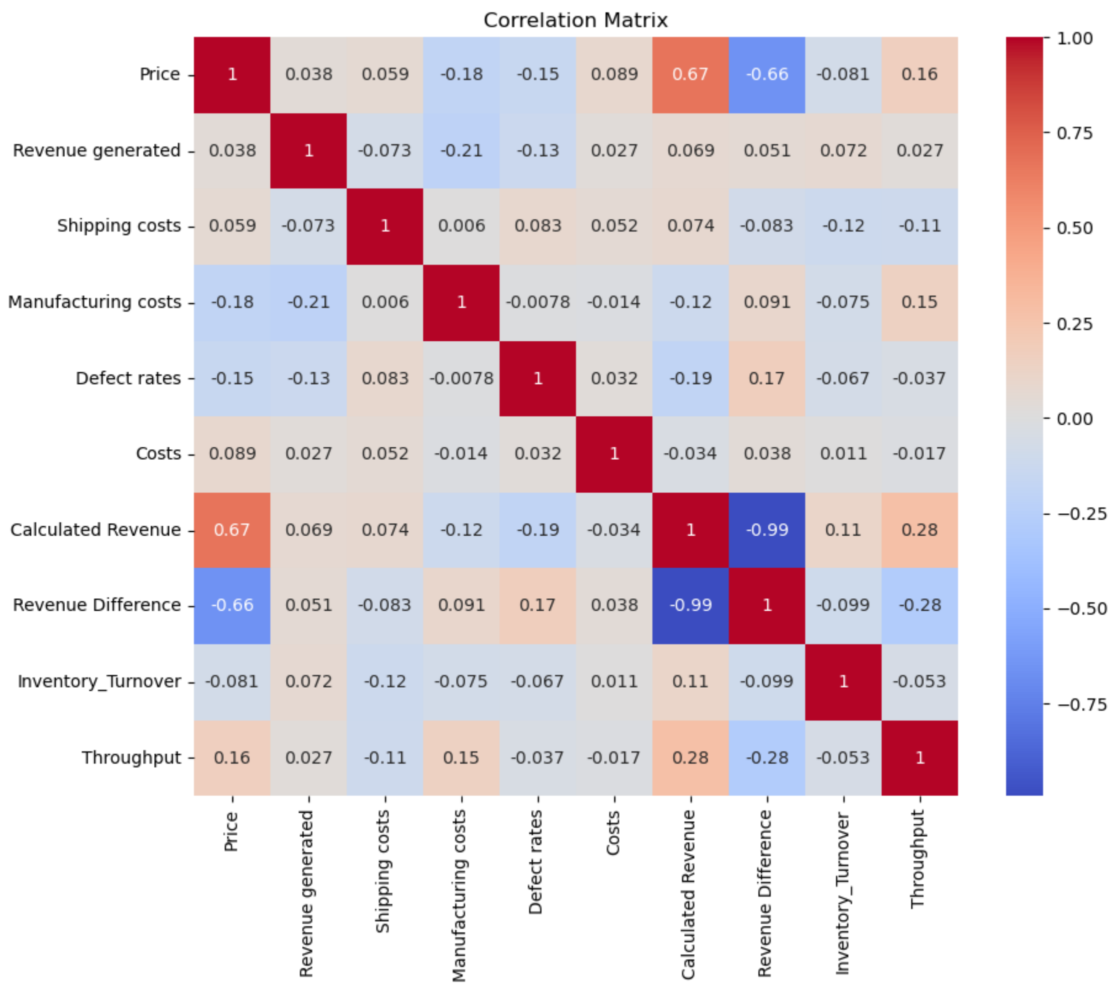
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Ensure your DataFrame is properly loaded
# df = pd.read_csv('your_data.csv')  # Uncomment if needed

# Calculate throughput
df['Throughput'] = df['Production volumes'] / df['Manufacturing lead time']

# Create the visualization
plt.figure(figsize=(12,6))
ax = sns.barplot(
    data=df,
    x='Product type',
    y='Throughput',
    hue='Supplier name',
    estimator=np.median,
    errorbar=None  # Removes confidence intervals for cleaner view
)

plt.title('Manufacturing Throughput by Supplier and Product Type')
plt.ylabel('Units Produced per Day')
plt.xlabel('Product Type')

# Improve legend placement and formatting
plt.legend(
    title='Supplier',
    bbox_to_anchor=(1.05, 1),
    loc='upper left',
    borderaxespad=0
)

# Rotate x-axis labels if needed
plt.xticks(rotation=45)
```

```
# Show plot
plt.tight_layout()
plt.show()
```



[69]:
```
# Select only numerical columns
numerical_df = df.select_dtypes(include=['float64', 'int64'])

# Generate the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(numerical_df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```


Correlation Matrix
Price
Revenue generated
Shipping costs
Manufacturing costs
Defect rates
Costs
Calculated Revenue
Revenue Difference
Inventory_Turnover
Throughput
1 0.038 0.059 -0.18 -0.15 0.089 0.67 -0.66 -0.081 0.16
0.038 1 -0.073 -0.21 -0.13 0.027 0.069 0.051 0.072 0.027
0.059 -0.073 1 0.006 0.083 0.052 0.074 -0.083 -0.12 -0.11
-0.18 -0.21 0.006 1 -0.0078 -0.014 -0.12 0.091 -0.075 0.15
-0.15 -0.13 0.083 -0.0078 1 0.032 -0.19 0.17 -0.067 -0.037
0.089 0.027 0.052 -0.014 0.032 1 -0.034 0.038 0.011 -0.017
0.67 0.069 0.074 -0.12 -0.19 -0.034 1 -0.99 0.11 0.28
-0.66 0.051 -0.083 0.091 0.17 0.038 -0.99 1 -0.099 -0.28
-0.081 0.072 -0.12 -0.075 -0.067 0.011 0.11 -0.099 1 -0.053
0.16 0.027 -0.11 0.15 -0.037 -0.017 0.28 -0.28 -0.053 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

```
[ ]:
```