Jupyter Notebook Execution Report

Name: Your Name
Date: December 31, 2025

Cell 1: ■ Markdown

Imports and Loading in the data

Cell 2: ■ Code

```
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
```

Cell 3: ■ Code

```
df = pd.read_csv(filepath_or_buffer= 'supply_chain_data.csv')
```

Cell 4: ■ Markdown

Data Cleaning and looking for missng values

Cell 5: ■ Code

```
df.info()
```

Output:

```
&lt;class 'pandas.core.frame.DataFrame'&gt;
RangeIndex: 100 entries, 0 to 99
Data columns (total 24 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Product type             100 non-null    object
 1   SKU                      100 non-null    object
 2   Price                    100 non-null    float64
```

```
 3   Availability            100 non-null    int64
 4   Number of products sold  100 non-null    int64
 5   Revenue generated        100 non-null    float64
 6   Customer demographics    100 non-null    object
 7   Stock levels             100 non-null    int64
 8   Lead times               100 non-null    int64
 9   Order quantities         100 non-null    int64
 10  Shipping times           100 non-null    int64
 11  Shipping carriers        100 non-null    object
 12  Shipping costs           100 non-null    float64
 13  Supplier name            100 non-null    object
 14  Location                 100 non-null    object
 15  Lead time                100 non-null    int64
 16  Production volumes       100 non-null    int64
 17  Manufacturing lead time  100 non-null    int64
 18  Manufacturing costs      100 non-null    float64
 19  Inspection results       100 non-null    object
 20  Defect rates             100 non-null    float64
 21  Transportation modes     100 non-null    object
 22  Routes                   100 non-null    object
 23  Costs                    100 non-null    float64
dtypes: float64(6), int64(9), object(9)
memory usage: 18.9+ KB
```

Cell 6: ■ Code

```
df.describe()
```

Output:

```
            Price  Availability  ...  Defect rates       Costs
count  100.000000    100.000000  ...    100.000000  100.000000
mean    49.462461     48.400000  ...      2.277158  529.245782
std     31.168193     30.743317  ...      1.461366  258.301696
min      1.699976      1.000000  ...      0.018608  103.916248
25%     19.597823     22.750000  ...      1.009650  318.778455
```

```
50%     51.239831     43.500000  ...      2.141863  520.430444
75%     77.198228     75.000000  ...      3.563995  763.078231
max     99.171329    100.000000  ...      4.939255  997.413450

[8 rows x 15 columns]
```

Cell 7: ■ Code

```
# Looking for any missing values
print("Amount of missing values per col")
df.isna().sum()
```

Output:

```
Amount of missing values per col
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
```

Cell 8: ■ Code

```
# Looking for duplicated values
print('Amount of duplicated values are:', int(df.duplicated().sum()))
```

Output:

```
Amount of duplicated values are: 0
```

Cell 9: ■ Code

```
skip_items = ['SKU','Price','Availability','Number of products sold','Revenue generated','Shipping costs','Costs','Manufacturing costs','Defect rates']
for col in df.columns:
if col in skip_items:
continue
else:
print(col,' : ',df[col].unique())
```

Output:

```
Product type  :  ['haircare' 'skincare' 'cosmetics']
Customer demographics  :  ['Non-binary' 'Female' 'Unknown' 'Male']
Stock levels  :  [ 58  53   1  23   5  90  11  93  14  51  46 100  80  54   9   2  45  10
  48  27  69  71  84   4  82  59  47  60   6  89  42  18  25  78  64  22
  36  13  92  30  97  31  96  33  41  32  86  73  57  12   0  95  76  17
  16  38  39  65  15  66  98  63  77  67  55]
Lead times  :  [ 7 30 10 13  3 27 15 17 23  8 29  5 11 12 25  1 26 16  9 20 19 24  4 22
 18  2  6 28 14]
Order quantities  :  [96 37 88 59 56 66 58 11 15 83 80 60 85 48 78 69 46 94 68  7 63 29  2 52
 62 24 67 35 44 64 95 21 28 34 39 38 57 72  6 51  9 82 54 61 26 36 40 10
 75 19 71 27 22 77  1 20 41  8 55 32  4]
Shipping times  :  [ 4  2  6  8  3  1  7  9  5 10]
Shipping carriers  :  ['Carrier B' 'Carrier A' 'Carrier C']
```

```
Supplier name  :  ['Supplier 3' 'Supplier 1' 'Supplier 5' 'Supplier 4' 'Supplier 2']
Location  :  ['Mumbai' 'Kolkata' 'Delhi' 'Bangalore' 'Chennai']
Lead time  :  [29 23 12 24  5 10 14 22 13 18 28  3 25  7 20 19 11 26 16 27 30  1  4  9
 21 17  2  8  6]
Production volumes  :  [215 517 971 937 414 104 314 564 769 963 830 362 563 173 558 580 399 453
 374 694 309 791 780 568 447 934 171 291 329 806 461 737 251 452 367 671
 867 841 793 892 179 206 834 794 870 964 109 177 306 673 727 631 497 918
 826 588 396 176 929 480 751 736 328 358 198 375 862 775 258 152 444 919
 759 985 334 858 228 202 698 955 443 589 211 569 523 953 370 585 207 824
 908 450 648 535 581 921]
Manufacturing lead time  :  [29 30 27 18  3 17 24  1  8 23  5 11 10 14  7 21 16  6  4 28  2 19 15
 25 20  9 26 22 13]
Inspection results  :  ['Pending' 'Fail' 'Pass']
Transportation modes  :  ['Road' 'Air' 'Rail' 'Sea']
Routes  :  ['Route B' 'Route C' 'Route A']
```

Cell 10: ■ Code

```
list(df.columns)
```

Output:

```
['Product type', 'SKU', 'Price', 'Availability', 'Number of products sold', 'Revenue generated',
```

Cell 11: ■ Markdown

Inventory Optimization & Stock-Out Prevention

Ensure the right amount of product is in the right place at the right time.

Cell 12: ■ Code

```
import numpy as np
```

```
# 1. Calculate Usage Value
df['Usage_Value'] = df['Costs'] * df['Number of products sold']
```

```
# 2. Sort and calculate cumulative percentage (standardizes the logic)
df = df.sort_values(by='Usage_Value', ascending=False).reset_index(drop=True)
```

```
df['cum_perc'] = df['Usage_Value'].cumsum() / df['Usage_Value'].sum()
```

```
# 3. Use np.select for vectorized labeling
conditions = [
df['cum_perc'] &lt;= 0.70,
df['cum_perc'] &lt;= 0.90
]
choices = ['High (A)', 'Medium (B)']
```

```
df['ABC_category'] = np.select(conditions, choices, default='Low (C)')
```

Cell 13: ■ Code

```
print(df['ABC_category'].value_counts(normalize=True))
```

Output:

```
ABC_category
Low (C)       0.39
High (A)      0.35
Medium (B)    0.26
Name: proportion, dtype: float64
```

Cell 14: ■ Code

```
df['ABC_category'].isna().sum()
```

Output:

```
np.int64(0)
```

Cell 15: ■ Code

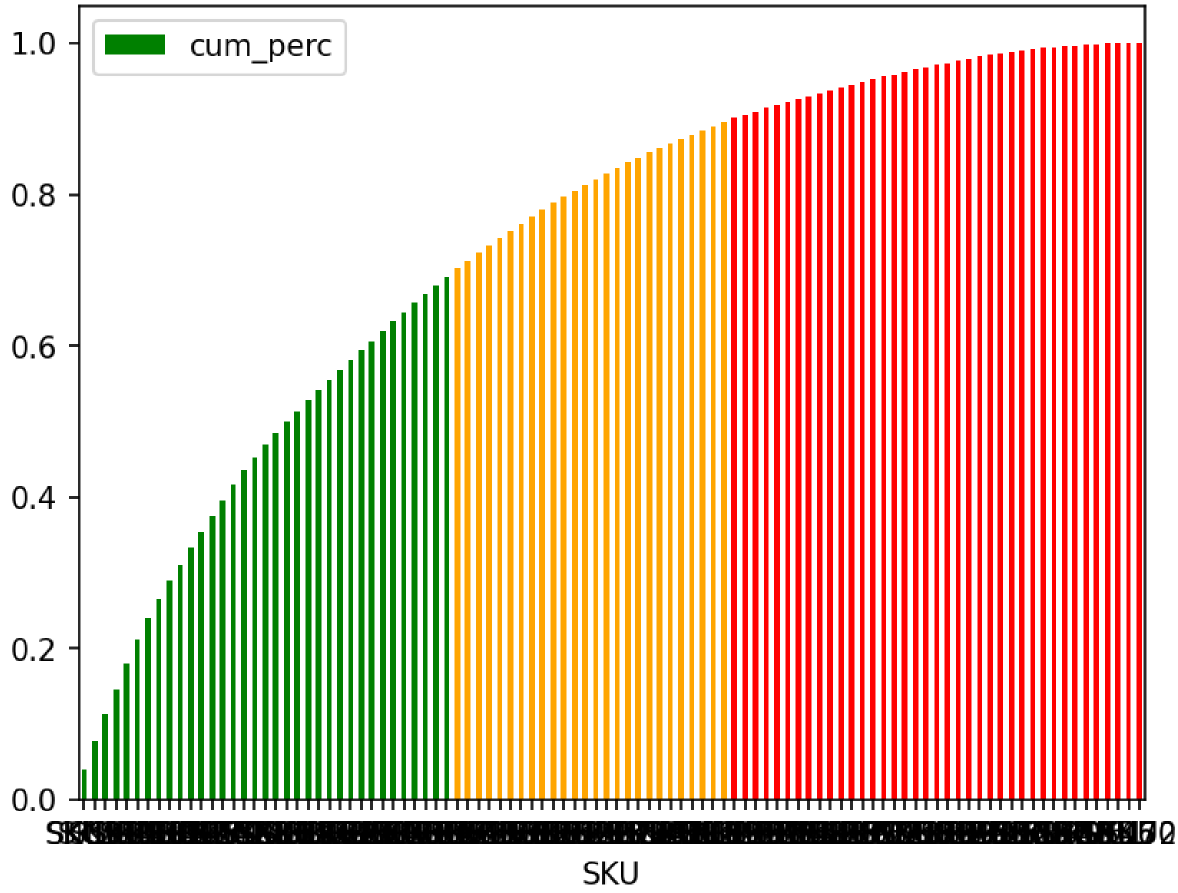
```
# Create a vertical bar plot to show
color_map = {'High (A)': 'green', 'Medium (B)': 'orange', 'Low (C)': 'red'}
```

```
# Create the color list
colors = df['ABC_category'].map(color_map)
```

```
# Plot using the 'color' list
ax = df.plot.bar(x='SKU', y='cum_perc', rot=0, color=colors)
```

```
'''
Green: Class A
Yellow : Class B
Red: Class C
```

```
ABC analysis is an inventory management method that classifies stock into three categories (A, B, C)
based on value and importance, applying the Pareto Principle (80/20 rule) to focus resources where they matter most:
A items are high-value, require tight control (e.g., 70-80% cost, 20% quantity); B items are mid-range (moderate control);
and C items are low-value, numerous, and need minimal control (e.g., 10% cost, 50% quantity). This technique optimizes
inventory control, reduces costs, and improves efficiency by prioritizing management efforts on critical items, leading
to better stock availability and profitability.
'''
```



Cell 16: ■ Code

```
# Show which SKUs are in each class
```

```
print("List of Low value SKUs")
print((pd.DataFrame(df.loc[df['ABC_category'] == 'Low (C)'])))
```

Output:

```
List of Low value SKUs
   Product type    SKU      Price  ...   Usage_Value  cum_perc  ABC_category
61     skincare  SKU39  19.127477  ...  115046.447041  0.900733       Low (C)
62     haircare  SKU57  49.263205  ...  109457.411163  0.905273       Low (C)
63     haircare  SKU51  26.700761  ...  107027.276907  0.909712       Low (C)
64     skincare  SKU69  54.865529  ...  106115.898373  0.914113       Low (C)
65     skincare  SKU64  89.634096  ...  103344.161347  0.918400       Low (C)
66     skincare  SKU31  50.847393  ...  102375.706712  0.922646       Low (C)
67     skincare  SKU58  59.841561  ...   98886.444367  0.926747       Low (C)
68     skincare  SKU42  46.529168  ...   98526.853678  0.930834       Low (C)
69     haircare  SKU93  69.290831  ...   93881.718431  0.934727       Low (C)
70     skincare  SKU75  92.996884  ...   92551.742690  0.938566       Low (C)
71     skincare  SKU19  51.123870  ...   89256.527014  0.942268       Low (C)
72     skincare  SKU67  87.755432  ...   86836.434110  0.945870       Low (C)
73    cosmetics  SKU62  72.796354  ...   86628.559255  0.949462       Low (C)
74    cosmetics  SKU17  81.462534  ...   84537.733240  0.952969       Low (C)
75     skincare  SKU56  20.986386  ...   77395.605210  0.956179       Low (C)
76    cosmetics  SKU23   4.324341  ...   76764.813430  0.959363       Low (C)
77    cosmetics  SKU88  75.270407  ...   76586.274747  0.962539       Low (C)
78    cosmetics   SKU8  68.717597  ...   75833.570134  0.965684       Low (C)
79     haircare  SKU79  57.057031  ...   69597.835428  0.968571       Low (C)
80     haircare  SKU87  80.414037  ...   69469.378201  0.971452       Low (C)
81    cosmetics  SKU89  97.760086  ...   69344.996506  0.974328       Low (C)
82     haircare  SKU25  39.629344  ...   65112.104295  0.977029       Low (C)
83    cosmetics  SKU92  47.714233  ...   63505.607980  0.979663       Low (C)
84    cosmetics  SKU96  24.423131  ...   61152.453732  0.982199       Low (C)
85     skincare  SKU15  36.989245  ...   59967.184201  0.984686       Low (C)
```

```
86     skincare  SKU86  19.998177  ...  58928.840433  0.987130       Low (C)
87     haircare  SKU68  37.931812  ...  48852.127408  0.989156       Low (C)
88    cosmetics  SKU28   2.397275  ...  48634.188840  0.991174       Low (C)
89     haircare   SKU5   1.699976  ...  34612.801800  0.992609       Low (C)
90     haircare  SKU97   3.526111  ...  33488.210218  0.993998       Low (C)
91    cosmetics  SKU49  78.897913  ...  33352.126683  0.995381       Low (C)
92     haircare  SKU48  76.035544  ...  22290.905505  0.996306       Low (C)
93     skincare   SKU3  61.163343  ...  21146.421215  0.997183       Low (C)
94    cosmetics  SKU85  76.962994  ...  21067.170750  0.998057       Low (C)
95     haircare  SKU61  52.028750  ...  19230.883804  0.998854       Low (C)
96     haircare  SKU45  33.784138  ...  11887.336729  0.999347       Low (C)
97     skincare   SKU6   4.078333  ...   8733.991296  0.999710       Low (C)
98     haircare  SKU70  47.914542  ...   5864.732760  0.999953       Low (C)
99     haircare   SKU2  11.319683  ...   1135.362254  1.000000       Low (C)

[39 rows x 27 columns]
```

Cell 17: ■ Markdown

Actionable Insights

The team should look into retiring some of the low category items has it is not moving as well and not producing alot of revenue for the company

The focus should be shifted more toward level A items.

Cell 18: ■ Markdown

Supplier Performance & Quality Control

Cell 19: ■ Code

```
''''

What you can do:

Supplier Scorecard: Rank suppliers by a weighted score of Defect rates (quality)
and Manufacturing lead time (speed).

Correlation Analysis: Check if higher Manufacturing costs actually lead to lower
Defect rates.

'''
```

Cell 20: ■ Code

```
supplierScoreCard = pd.DataFrame(df[['SKU', 'Product type','Price','Lead
times','Defect rates','Supplier name'

]])
```

Cell 21: ■ Code

```
lead_time = 29

defect_rate = 0.22641036084992516

targetLeadTime = 1
```

```
supplierScoreCard['Supplier Score'] = (targetLeadTime/supplierScoreCard['Lead
times']) * 100 * 0.4 + (1-supplierScoreCard['Defect rates']/100) * 100 * 0.6
```

Cell 22: ■ Code

```
print((10/1) * 100 * 0.4 + (1-0.613327/100) * 100 * 0.6)

print((1) * 100 * 0.4 + (1-0.613327/100) * 100 * 0.6)
```

```
print(supplierScoreCard['Defect rates'].max())
```

Output:

```
459.6320038

99.6320038

4.939255288620948
```

Cell 23: ■ Code

```
supplierScoreCard.groupby(by = 'Supplier name')['Supplier
Score'].mean().sort_values()
```

```
'''

Insights: Supplier 3 is preforming significantly worse than the best supplier and
better KPIs need to be set inorder to keep the supplier on track.

'''
```

Cell 24: ■ Code

```python
#standard deviation and sample size
```

```python
#print(list(df['Manufacturing costs']))
#list(df['Defect rates'])
```

```python
x = df[['Manufacturing costs']]
y = df[['Defect rates']]
```

```python
person_corr = df[['Manufacturing costs','Defect
rates']].corr(method='pearson').loc['Manufacturing costs','Defect rates']
covariance = df[['Manufacturing costs','Defect rates']].cov().loc['Manufacturing
costs','Defect rates']
standardDevMC = df[['Manufacturing costs']].std().item()
standardDevDR = df[['Defect rates']].std().item()
sampleSize = float(len(df))
```

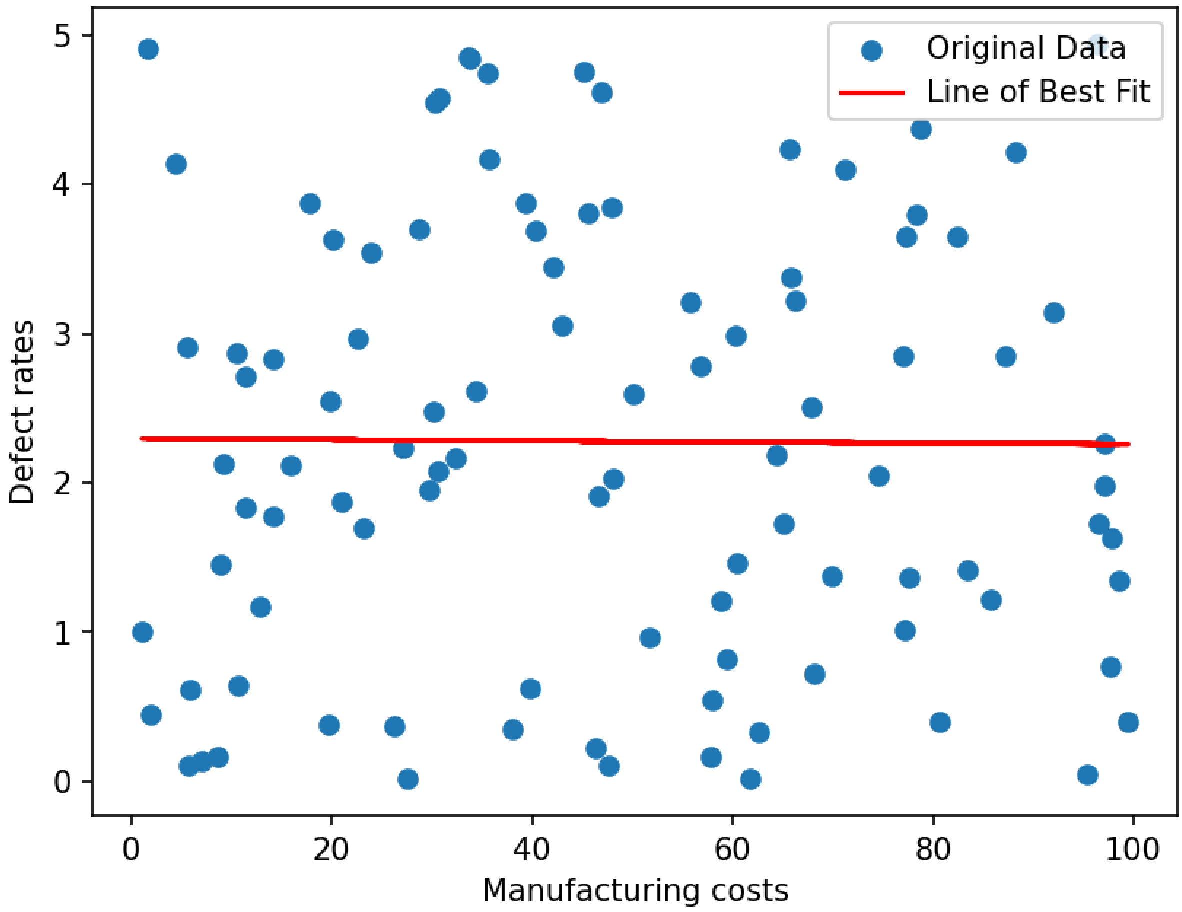
```python
slope, intercept, r_value, p_value, std_err = stats.linregress(df['Manufacturing
costs'], df['Defect rates'])
```

```python
# 2. Create the "Line" equation: y = mx + b
# We use this to create the points for the line on a graph
line = slope * df['Manufacturing costs'] + intercept
```

```python
# 3. Visualize it
plt.scatter(df['Manufacturing costs'], df['Defect rates'], label='Original Data')
plt.plot(df['Manufacturing costs'], line, color='red', label='Line of Best Fit')
plt.xlabel('Manufacturing costs')
plt.ylabel('Defect rates')
plt.legend()
plt.show()
```

Output:

```
[STDERR]
&lt;string&gt;:1: UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be shown
```



Cell 25: ■ Markdown

This shows there is not a statistically significant corelation between Manufacuring costs and the defect rates.

Cell 26: ■ Markdown

3. Shipping & Logistics Efficiency

The Goal: Reduce the "last mile" cost and time.

Target Variables: Shipping times, Shipping carriers, Shipping costs, Transportation modes, Routes.

What you can do:

Cost-Benefit Map: Compare Shipping costs against Shipping times for different Transportation modes. Is the Air route actually significantly faster than the Sea route for the price difference?

Carrier Benchmarking: Which Shipping carriers have the highest variance in Shipping times? This helps in choosing the most reliable partner for specific Routes.

Cell 27: ■ Code

```
shipping_df = pd.DataFrame(df[['SKU', 'Product type','Price','Shipping times','Shipping carriers','Shipping costs','Supplier name','Location','Lead time','Production volumes','Transportation modes','Routes','Costs']])
```

Cell 28: ■ Code

```
print(shipping_df.groupby(by='Transportation modes')['Shipping costs'].mean())

print('\n')

print(shipping_df.groupby(by='Transportation modes')['Shipping times'].mean())
```

```
airByDays = shipping_df.groupby(by='Transportation modes')['Shipping costs'].mean()["Air"] / shipping_df.groupby(by='Transportation modes')['Shipping times'].mean()["Air"]

seaByDays = shipping_df.groupby(by='Transportation modes')['Shipping costs'].mean()["Sea"] / shipping_df.groupby(by='Transportation modes')['Shipping times'].mean()["Sea"]
```

```
print('Cost per day Air: ', airByDays)

print('Cost per day Sea: ', seaByDays)

print("Percent Increase: ", round(((airByDays - seaByDays) / seaByDays) * 100,ndigits=2),'%')
```

```
# The company is paying about 68% per shipping day when using Air vs sea. It would be interesting to investiage to see if opperations can be moved to sea to save on shipping costs, as the cost per day is not that influential.
```

Output:

```
Transportation modes
Air     6.017839
Rail    5.469098
Road    5.542115
Sea     4.970294
Name: Shipping costs, dtype: float64

Transportation modes
Air     5.115385
Rail    6.571429
Road    4.724138
Sea     7.117647
```

```
Name: Shipping times, dtype: float64
Cost per day Air:  1.1764197279770232
Cost per day Sea:  0.6983057426347634
Percent Increase:  68.47 %
Transportation modes
Air     6.017839
Rail    5.469098
Road    5.542115
Sea     4.970294
Name: Shipping costs, dtype: float64

Transportation modes
Air     5.115385
Rail    6.571429
Road    4.724138
Sea     7.117647
Name: Shipping times, dtype: float64
Cost per day Air:  1.1764197279770232
Cost per day Sea:  0.6983057426347634
Percent Increase:  68.47 %
```

Cell 29: ■ Code

```
print(shipping_df.groupby(by='Routes')['Shipping times'].mean())
```

Output:

```
Routes
Route A    6.023256
Route B    5.702703
Route C    5.250000
Name: Shipping times, dtype: float64
```

Cell 30: ■ Code

```
print("Shipping Carriers means:")
print(shipping_df.groupby(by=['Shipping carriers','Routes'])['Shipping
times'].mean())
```

```
print("Variance:")
print(shipping_df.groupby(by=['Shipping carriers','Routes'])['Shipping times'].var())
```

Output:

```
Shipping Carriers means:
Shipping carriers  Routes
Carrier A          Route A    6.250000
                   Route B    6.384615
                   Route C    4.666667
Carrier B          Route A    5.764706
                   Route B    6.000000
                   Route C    3.400000
Carrier C          Route A    6.142857
                   Route B    4.000000
                   Route C    8.142857
Name: Shipping times, dtype: float64
Variance:
Shipping carriers  Routes
Carrier A          Route A     8.931818
                   Route B    10.089744
                   Route C    22.333333
Carrier B          Route A     7.316176
                   Route B     3.600000
                   Route C     4.711111
Carrier C          Route A     5.516484
                   Route B     5.714286
                   Route C     1.809524
Name: Shipping times, dtype: float64
```

Cell 31: ■ Code

```
print(shipping_df.groupby(by=['Shipping carriers','Transportation modes','Routes'])['Shipping times'].mean())
```

Output:

```
Shipping carriers  Transportation modes  Routes
```

```
Carrier A   Air    Route A     4.500000
                   Route B     6.333333
            Rail   Route A     7.333333
                   Route B     6.666667
                   Route C    10.000000
            Road   Route A     6.000000
                   Route B     5.800000
                   Route C     2.000000
            Sea    Route A     6.666667
                   Route B     7.500000
Carrier B   Air    Route A     5.500000
                   Route B     4.333333
                   Route C     2.600000
            Rail   Route A     5.857143
                   Route B     7.142857
                   Route C     4.000000
            Road   Route A     5.333333
                   Route B     5.200000
                   Route C     4.000000
            Sea    Route A     8.000000
                   Route B     7.000000
                   Route C     5.000000
Carrier C   Air    Route A     5.333333
                   Route B     5.000000
                   Route C     8.333333
            Rail   Route A     6.500000
                   Route B     3.000000
                   Route C     8.000000
            Road   Route A     4.750000
                   Route B     2.333333
            Sea    Route A     8.333333
                   Route B     5.666667
                   Route C     8.000000
Name: Shipping times, dtype: float64
```

Cell 32: ■ Code

```python
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt
```

```python
df = shipping_df.groupby(by=['Shipping carriers','Transportation
modes','Routes'])['Shipping times'].mean()
```

```python
# 2. Reset the index to make the data 'flat' for plotting

plot_df = df.reset_index()
```

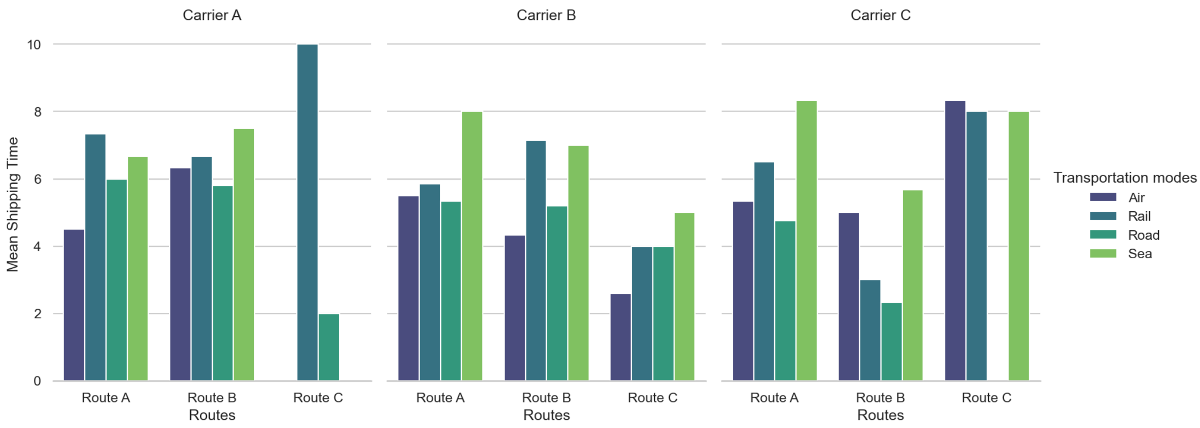
```python
# 3. Create the faceted bar chart

sns.set_theme(style="whitegrid")

g = sns.catplot(

data=plot_df,

kind="bar",

x="Routes",

y="Shipping times",

hue="Transportation modes",

col="Shipping carriers",

palette="viridis",

height=5,

aspect=0.8

)
```

```python
# 4. Final touches for readability

g.set_axis_labels("Routes", "Mean Shipping Time")

g.set_titles("{col_name}")

g.despine(left=True)
```

```python
plt.show()
```

Output:

```
[STDERR]

&lt;string&gt;:1: UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be shown
```



Cell 33: ■ Code

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

```
df = shipping_df.groupby(by=['Shipping carriers','Transportation modes','Routes'])['Shipping times'].var()
```

```
# 2. Reset the index to make the data 'flat' for plotting
plot_df = df.reset_index()
```

```
# 3. Create the faceted bar chart
sns.set_theme(style="whitegrid")
g = sns.catplot(
data=plot_df,
kind="bar",
x="Routes",
y="Shipping times",
hue="Transportation modes",
col="Shipping carriers",
palette="viridis",
height=5,
aspect=0.8
)
```

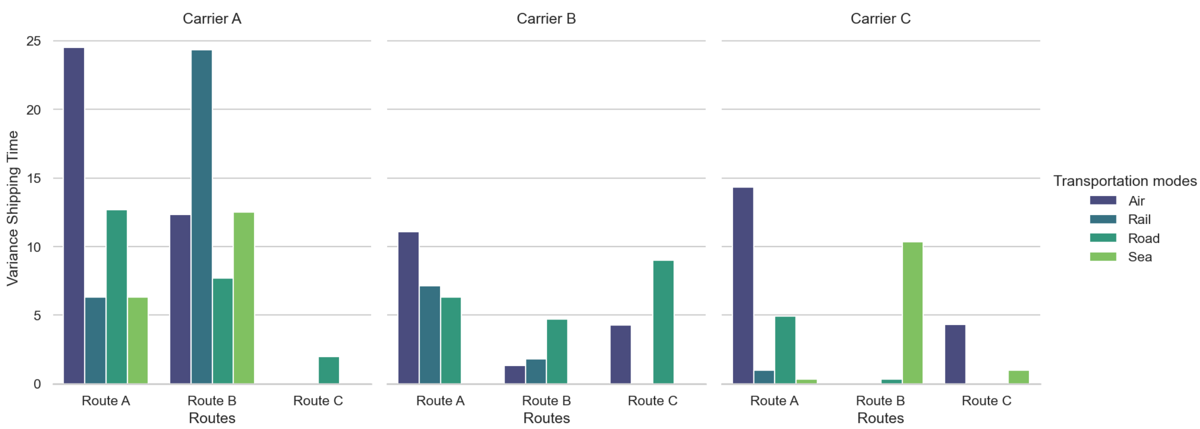
```
# 4. Final touches for readability
g.set_axis_labels("Routes", "Variance Shipping Time")
```

```
g.set_titles("{col_name}")

g.despine(left=True)
```

```
plt.show()
```

Output:

```
[STDERR]

&lt;string&gt;:1: UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be shown
```



Cell 34: ■ Code

```
'''

Key takeaways:

For any sucessful business, both a low variance and mean is important, thus I
created a Reliability score, to see which Transportation methods, Routes and
Carriers we should be using.

Reliability score (Coefficient of Variation) = Sigma / Mu

sigma = is the Standard Deviation (the square root of the Variance)

mu = mean

A good starting point would to look at the top 4 [carrieers + route] and call them
to figure out why these routes have such high scores. A high score indicates there
is something broken

in the process, as either the mean time or variance is really high.
'''
```

```
## This creates a table with both metrics side-by-side
```

```
analysis_df = shipping_df.groupby(['Shipping carriers', 'Routes']).agg({'Shipping times': ['mean', 'var', 'std']})

analysis_df.columns = ['mean', 'var', 'std']

analysis_df['Reliability score'] = analysis_df['mean'] / analysis_df['std']
```

```
print(analysis_df.sort_values(by='Reliability score', ascending=False))
```

Output:

```
                                 mean        var       std  Reliability score
Shipping carriers Routes
Carrier C         Route C   8.142857   1.809524  1.345185           6.053334
Carrier B         Route B   6.000000   3.600000  1.897367           3.162278
Carrier C         Route A   6.142857   5.516484  2.348720           2.615407
Carrier B         Route A   5.764706   7.316176  2.704843           2.131253
Carrier A         Route A   6.250000   8.931818  2.988615           2.091270
                  Route B   6.384615  10.089744  3.176436           2.009994
Carrier C         Route B   4.000000   5.714286  2.390457           1.673320
Carrier B         Route C   3.400000   4.711111  2.170509           1.566453
Carrier A         Route C   4.666667  22.333333  4.725816           0.987484
```

Cell 35: ■ Code

```
# Looking how to catorgize each of the [Shipping carriers, Routes] combinations. I am catorgizing it into 3 tiers called ["Gold", "Silver", "Avoid"]

GLOBAL_MEAN_THRESHOLD_GOLD = analysis_df['mean'].quantile(0.33)

GLOBAL_RS_THRESHOLD_GOLD = analysis_df['Reliability score'].quantile(0.66)
```

```
GLOBAL_MEAN_THRESHOLD_AVOID = analysis_df['mean'].quantile(0.33)

GLOBAL_RS_THRESHOLD_AVOID = analysis_df['Reliability score'].quantile(0.66)
```

```
def categorize_route(row):
```

```
avg = row['mean']

rel = row['Reliability score']
```

```
if avg &lt; GLOBAL_MEAN_THRESHOLD_GOLD and rel &lt; GLOBAL_RS_THRESHOLD_GOLD:

return 'Gold'
```

```
elif avg &gt; GLOBAL_MEAN_THRESHOLD_AVOID and rel &gt; GLOBAL_RS_THRESHOLD_AVOID:
```

```
return 'Avoid'
else:
return 'Silver'
```

```
analysis_df['Tier'] = analysis_df.apply(categorize_route, axis=1)
print(analysis_df)
```

Output:

```
                              mean        var  ...  Reliability score    Tier
Shipping carriers Routes                       ...
Carrier A         Route A  6.250000   8.931818  ...           2.091270  Silver
                  Route B  6.384615  10.089744  ...           2.009994  Silver
                  Route C  4.666667  22.333333  ...           0.987484    Gold
Carrier B         Route A  5.764706   7.316176  ...           2.131253  Silver
                  Route B  6.000000   3.600000  ...           3.162278   Avoid
                  Route C  3.400000   4.711111  ...           1.566453    Gold
Carrier C         Route A  6.142857   5.516484  ...           2.615407   Avoid
                  Route B  4.000000   5.714286  ...           1.673320    Gold
                  Route C  8.142857   1.809524  ...           6.053334   Avoid

[9 rows x 5 columns]
```

Cell 36: ■ Code

```
''' Visualization of the chart above'''
sns.set_theme(style="whitegrid")
plt.figure(figsize=(10, 6))
```

```
plot = sns.scatterplot(
data=analysis_df,
x='mean',
y='Reliability score',
hue='Tier',
s=100,
palette={'Gold': 'gold', 'Silver': 'silver', 'Avoid': 'salmon'}
)
```

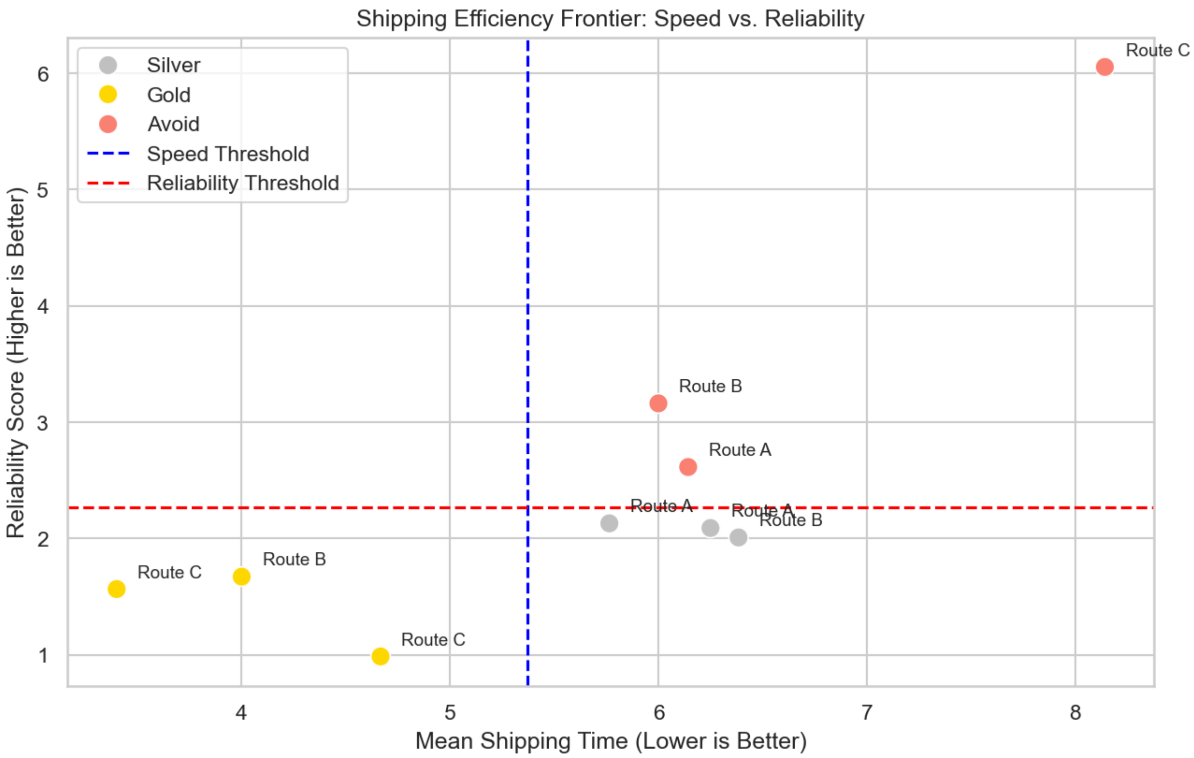
```python
plt.axvline(x=GLOBAL_MEAN_THRESHOLD_GOLD, color='blue', linestyle='--',
label='Speed Threshold')

plt.axhline(y=GLOBAL_RS_THRESHOLD_GOLD, color='red', linestyle='--',
label='Reliability Threshold')
```

```python
for i in range(analysis_df.shape[0]):

plt.text(

x=analysis_df['mean'].iloc[i] + 0.1,

y=analysis_df['Reliability score'].iloc[i] + 0.1,

s=analysis_df.index[i][1],

fontsize=9

)
```

```python
plt.title('Shipping Efficiency Frontier: Speed vs. Reliability')

plt.xlabel('Mean Shipping Time (Lower is Better)')

plt.ylabel('Reliability Score (Higher is Better)')

plt.legend()

plt.show()
```

Output:

```
[STDERR]

&lt;string&gt;:1: UserWarning: FigureCanvasAgg is non-interactive, and thus cannot be shown
```


Shipping Efficiency Frontier: Speed vs. Reliability
Silver
Gold
Avoid
Speed Threshold
Reliability Threshold
Route C
Route B
Route A
Route A
Route A
Route B
Route C
Route B
Route C
6
5
4
3
2
1
4
5
6
7
8
Reliability Score (Higher is Better)
Mean Shipping Time (Lower is Better)

Cell 37: ■ Code