Phase2(Innovation)

Dataset link: **https://www.kaggle.com/datasets/ksabishek/product-sales-data**

Machine learning algorithms to implement the Design phase :

Machine learning algorithms that can be applied to each stage of our Product-Sales Analysis process:

To train and test dataset:

ML-algorithm: Clustering(K-means)\

STEPS:

1.Data Preprocessing:

Output: Cleaned and preprocessed dataset.

2.Feature Selection:

Output: Subset of relevant features for clustering..

3.Choose a Clustering Algorithm:

Output: Chosen clustering algorithm..

4.Hyperparameter Tuning:

Output: Tuned hyperparameters for the chosen algorithm.

5.Fit the Model:

Output: Trained clustering model.

6.Analyze and Visualize Results:

Output: Cluster labels and visualizations of clustered data.

7.Interpretation and Insights:

Output: Understanding of the clusters and insights into your data.

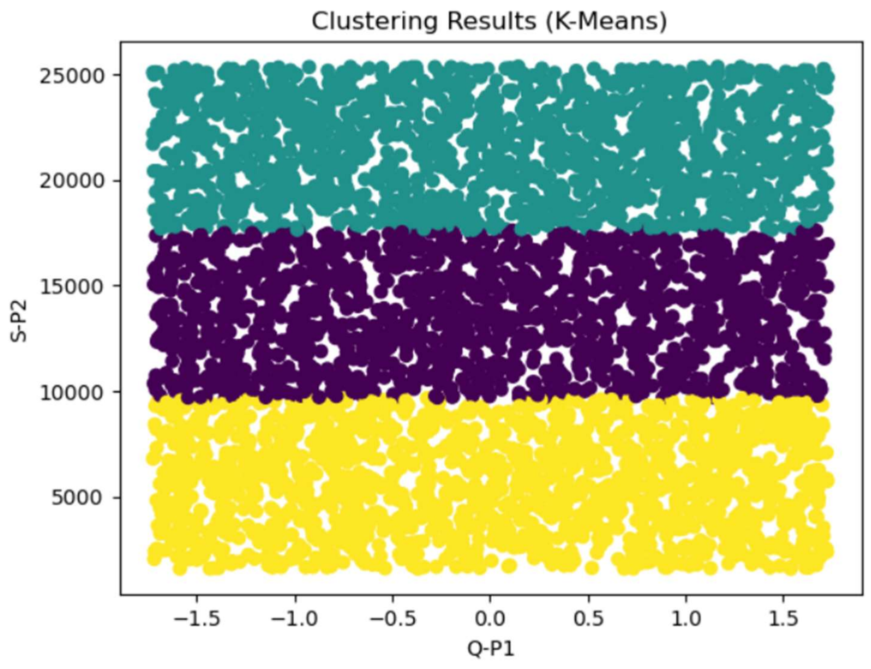
Algorithm :

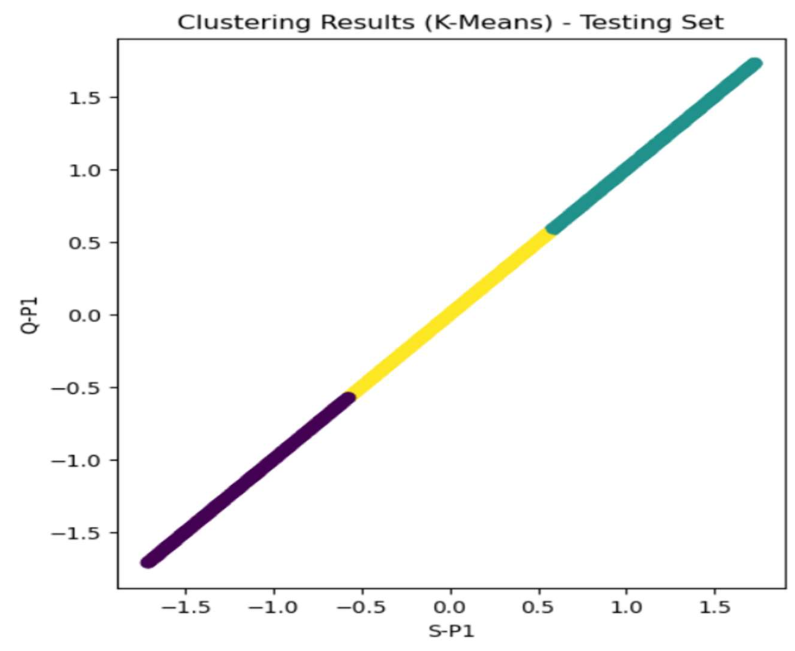
```
import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split
# Load the sales data from a CSV file
data = pd.read_csv('statsfinal.csv')  # Adjust the filename as needed
# Step 1: Data Preprocessing
# Remove rows with missing values or replace them with appropriate values
data.dropna(subset=['S-P1', 'Q-P1'], inplace=True)
# Step 2: Data Transformation
# Convert date column to a datetime object (if needed)
# data['Date'] = pd.to_datetime(data['Date'])
# Step 3: Feature Scaling (Standardization)
scaler = StandardScaler()
data[['S-P1', 'Q-P1']] = scaler.fit_transform(data[['S-P1', 'Q-P1']])
# Step 4: Save Preprocessed Data
# Save the preprocessed data to a new CSV file
data.to_csv('preprocessed_sales_data.csv', index=False)
# Step 2: Feature Selection
features = data[['S-P1', 'Q-P1']]  # Select relevant features for clustering
# Step 3: Choose a Clustering Algorithm (K-Means)
k = 3  # Choose the number of clusters (K)
kmeans = KMeans(n_clusters=k)
# Step 4: Split the data into a training and testing set (70-30 split)
X = features  # Features
X_train, X_test = train_test_split(X, test_size=0.3, random_state=42)
# Step 5: Fit the Model on the Training Set
kmeans.fit(X_train)
# Step 6: Predict on Both Training and Testing Sets
cluster_labels_train = kmeans.predict(X_train)
```

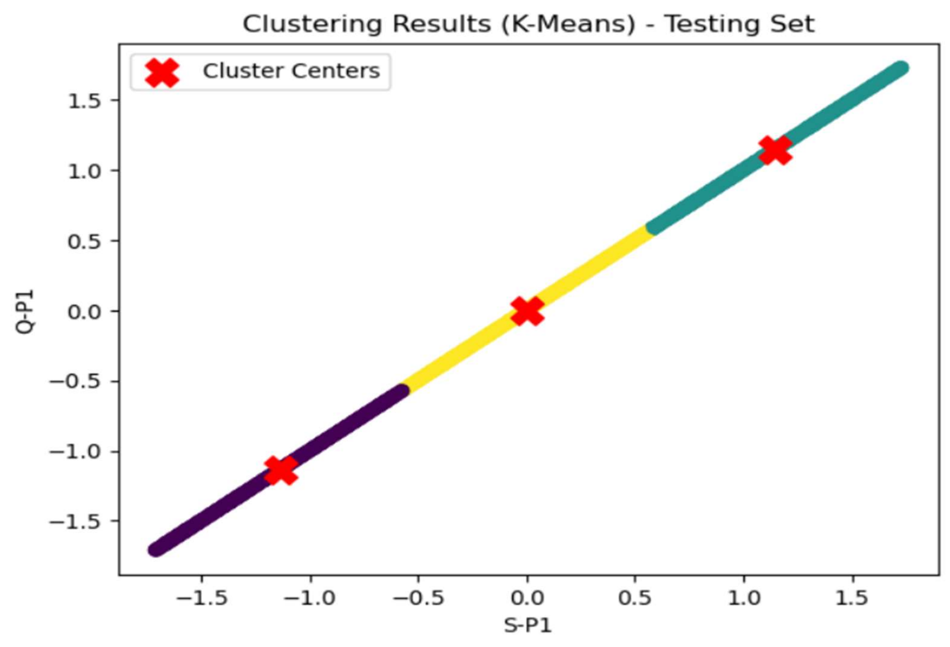
```
cluster_labels_test = kmeans.predict(X_test)
# Visualize the clusters using both training and testing sets
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.scatter(X_train['S-P1'], X_train['Q-P1'], c=cluster_labels_train,
cmap='viridis')
plt.xlabel('S-P1')
plt.ylabel('Q-P1')
plt.title('Clustering Results (K-Means) ')
plt.subplot(1, 2, 2)
plt.scatter(X_test['S-P1'], X_test['Q-P1'], c=cluster_labels_test, cmap='viridis')
plt.xlabel('S-P1')
plt.ylabel('Q-P1')
plt.title('Clustering Results (K-Means) - Testing Set')
plt.show()
# Step 6: Interpretation and Insights
# Get cluster centers (the centroids)
cluster_centers = kmeans.cluster_centers_
# Step 6.1: Cluster Characteristics
# Analyze the characteristics of each cluster in the testing set
for i in range(k):
    cluster_data = X_test[cluster_labels_test == i]
    print(f"Cluster {i + 1}:")
    print(f"Number of data points: {len(cluster_data)}")
    print(f"Cluster Center (S-P1, Q-P1): {cluster_centers[i]}")
    print()
# Step 6.2: Visualization of Cluster Centers in the testing set
plt.scatter(X_test['S-P1'], X_test['Q-P1'], c=cluster_labels_test, cmap='viridis')
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], marker='X', s=200, c='red',
label='Cluster Centers')

plt.xlabel('S-P1')

plt.ylabel('Q-P1')

plt.title('Clustering Results (K-Means) - Testing Set')

plt.legend()

plt.show()

# Output:

# - Cluster characteristics (number of data points, cluster center, average values
of S-P1 and Q-P1).

# - A visualization of cluster centers added to the scatter plot.
```

OUTPUT:





```
Cluster 1:
Number of data points: 462
Cluster Center (S-P1, Q-P1): [-1.1319778 -1.1319778]

Cluster 2:
Number of data points: 459
Cluster Center (S-P1, Q-P1): [1.1438162 1.1438162]

Cluster 3:
Number of data points: 459
Cluster Center (S-P1, Q-P1): [-0.00093619 -0.00093619]
```



To test dataset:

Testing a dataset in a clustering machine learning algorithm isn't a traditional process as in supervised learning, where we have distinct training and testing phases. Clustering is an unsupervised learning technique, and the evaluation is often based on internal or domain-specific metrics rather than traditional testing.

However, here are some steps we can take to assess the quality and effectiveness of our clustering dataset:

- Silhouette Score: The silhouette score is a metric that measures the separation and cohesion of clusters. A higher silhouette score indicates better-defined clusters. You can calculate the silhouette score using libraries like scikit-learn.
- Inertia or Within-Cluster Sum of Squares: Inertia measures the compactness of clusters. Lower inertia is generally better. It can help you determine the optimal number of clusters using the "elbow method," which looks for a point where the inertia starts to level off.

Algorithm:

```
from sklearn.cluster import KMeans

from sklearn.metrics import silhouette_score

import matplotlib.pyplot as plt

# Load and preprocess your data

# The dataset is preprocessed in train methos of design

# Choosing the number of clusters (K)

k = 3

# Create a K-Means clustering model

kmeans = KMeans(n_clusters=k, random_state=42)  # Use a fixed random state for reproducibility

# Fit the model to the data

kmeans.fit(features)

# Test the dataset using Silhouette Score

silhouette_avg = silhouette_score(features, kmeans.labels_)

print(f"Silhouette Score: {silhouette_avg}")
```

# The Silhouette Score ranges from -1 (poor clustering) to 1 (perfect clustering).

# Test the dataset using Inertia (Within-Cluster Sum of Squares)

inertia = kmeans.inertia_

print(f"Inertia (Within-Cluster Sum of Squares): {inertia}")

# You can use the elbow method to determine an optimal K based on the point where inertia starts to level off.

# Optionally, visualize the clusters

plt.scatter(data['S-P1'], data['Q-P1'], c=kmeans.labels_, cmap='viridis')

plt.xlabel('S-P1')

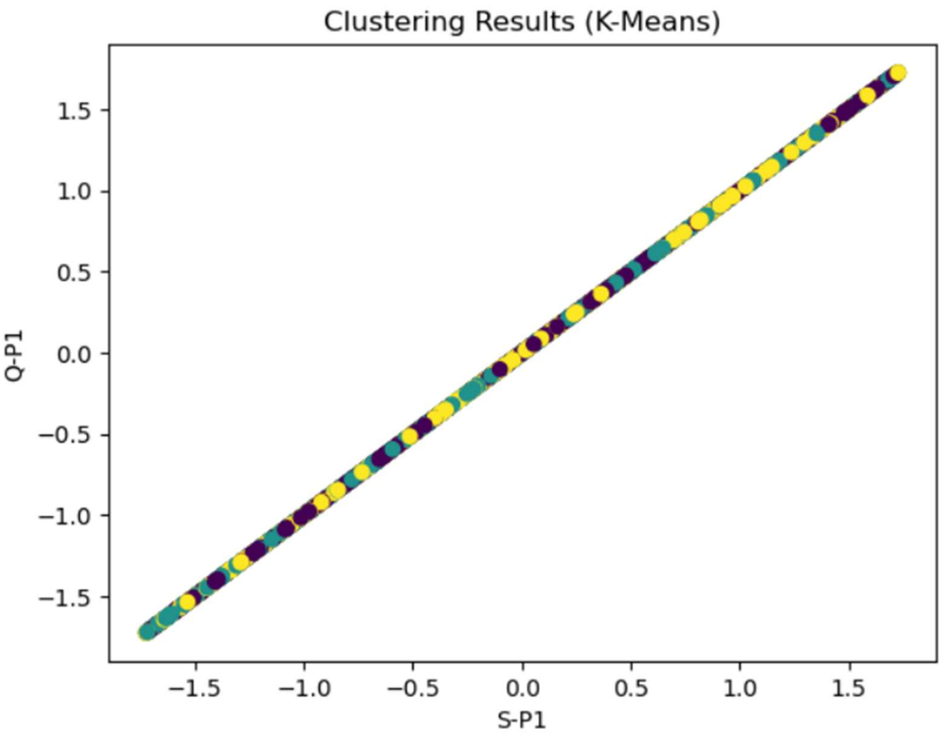
plt.ylabel('Q-P1')

plt.title('Clustering Results (K-Means)')

plt.show().

OUTPUT:

```
Silhouette Score: 0.588055961073017
Inertia (Within-Cluster Sum of Squares): 24256830162.01895
```



A silhouette score of 0.588 is a relatively high value, and it suggests that the clusters in our data are reasonably well-separated and have a good degree of cohesion. The silhouette score is a metric that measures the quality of clusters in your data, and it ranges from -1 to 1:

- A score near 1 indicates that the clusters are well-separated and that each data point is assigned to the correct cluster.

- A score near 0 suggests that the clusters are overlapping or too close to each other.

- A score near -1 indicates that data points may have been assigned to the wrong clusters.

In our case, a silhouette score of 0.588 indicates that the clusters in your dataset are relatively well-defined, and the data points are closer to their own cluster's center compared to other clusters. This suggests that the clustering process has been reasonably successful.

Visualization using Cognos and ML algorithms:

Cognos:

Visualize clustering results using Cognos:

1. Data Preparation:

   - Ensuring our data is well-prepared and includes the necessary features, including cluster labels assigned by the clustering algorithm.

2. Data Import:

   - Import your preprocessed data into Cognos. Cognos typically supports various data sources, such as databases, CSV files, and more. Importing the data source into Cognos.

3. Create a Data Source:

   - In Cognos, create a data source connection to the imported data.

4. Report Creation:

   - Create a new report or dashboard within Cognos.

5. Visualization Component:

   - Add a visualization component (e.g., chart, graph, or table) to your report or dashboard.

6. Select Data Attributes:

- Maping our data attributes to the visualization component. Depending on our data and the clustering results, you can select the attributes related to the clustering, such as the features used for clustering and the assigned cluster labels.

7. Visualization Configuration:

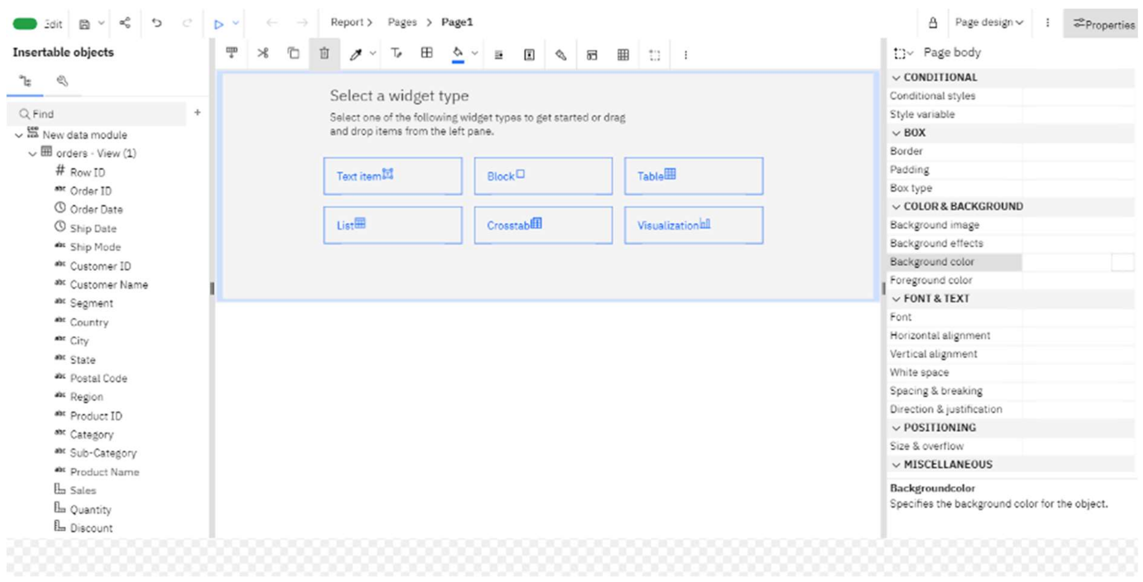
- Configure the visualization to represent the clustering results. This may include setting the X and Y axes (if applicable), color-coding data points by cluster label, and configuring labels or tooltips to display information about each data point.

8. Interactivity:

- Depending on our requirements, you can add interactivity to your visualization. For example, you might allow users to click on a cluster to see detailed information about its data points.

9. Publish and Share:

- Once our report or dashboard is ready, publish it in Cognos and share it with your intended audience.


Select a widget type

Select one of the following widget types to get started or drag and drop items from the left pane.

Backgroundcolor

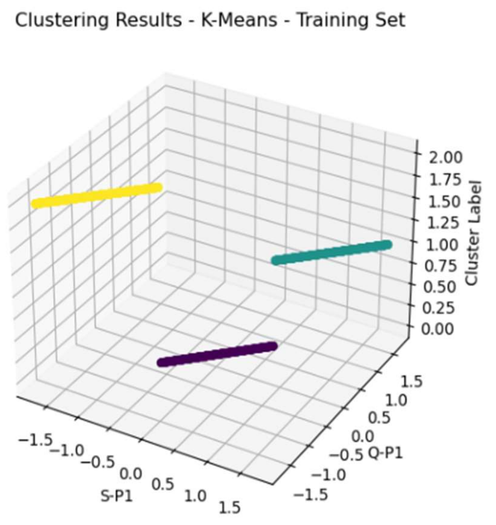
Specifies the background color for the object.

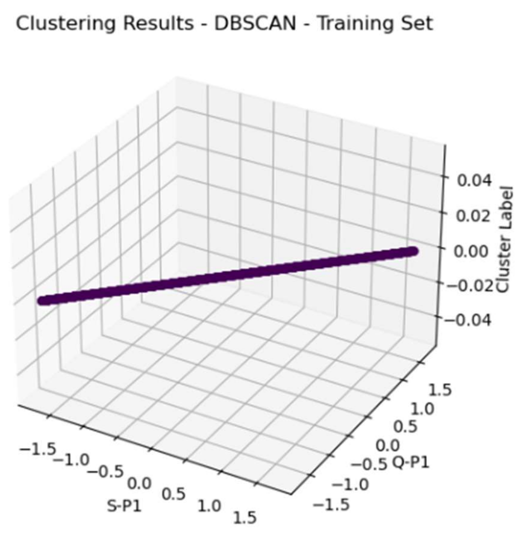
Can be viewed using any one of the Widget type.

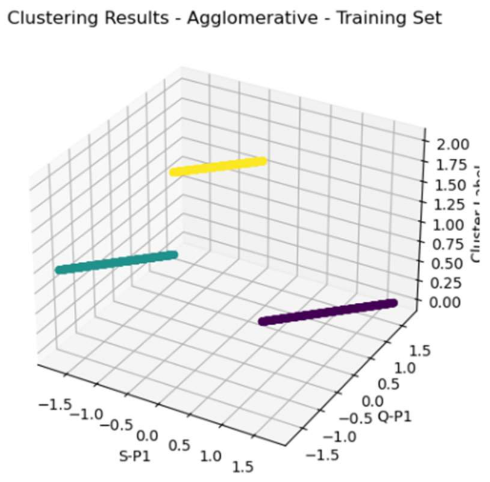
ML algorithms:

```
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.model_selection import train_test_split
# Load the sales data from a CSV file
data = pd.read_csv('statsfinal.csv')  # Adjust the filename as needed
# Step 1: Data Preprocessing
data.dropna(subset=['S-P1', 'Q-P1'], inplace=True)
# Step 2: Data Transformation and Feature Scaling
scaler = StandardScaler()
data[['S-P1', 'Q-P1']] = scaler.fit_transform(data[['S-P1', 'Q-P1']])
# Step 3: Split the data into a training and testing set (70-30 split)
X = data[['S-P1', 'Q-P1']]
X_train, X_test = train_test_split(X, test_size=0.3, random_state=42)
# Clustering Algorithms
kmeans = KMeans(n_clusters=3, random_state=0)
dbscan = DBSCAN(eps=0.3, min_samples=5)
agg_clustering = AgglomerativeClustering(n_clusters=3)
algorithms = [kmeans, dbscan, agg_clustering]
algorithm_names = ['K-Means', 'DBSCAN', 'Agglomerative']
# Create 3D plots for each clustering algorithm
fig = plt.figure(figsize=(18, 6))
for i, algorithm in enumerate(algorithms):
    algorithm.fit(X_train)
    cluster_labels = algorithm.labels_
    ax = fig.add_subplot(131 + i, projection='3d')
    ax.scatter(X_train['S-P1'], X_train['Q-P1'], cluster_labels,
c=cluster_labels, cmap='viridis')
    ax.set_xlabel('S-P1')
    ax.set_ylabel('Q-P1')
    ax.set_zlabel('Cluster Label')
    ax.set_title(f'Clustering Results - {algorithm_names[i]} - Training Set')
 plt.show()
```

OUTPUT:







Model:

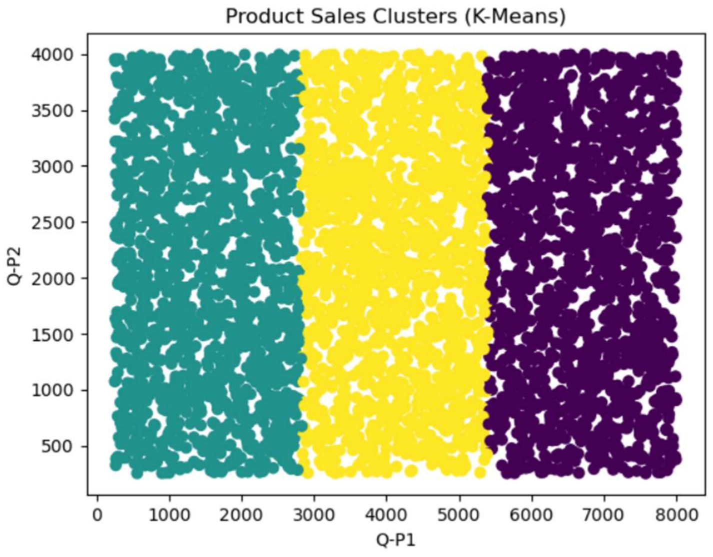
Selecting a machine learning algorithm that is appropriate for our problem. Common algorithms include linear regression, decision trees, random forests, support vector machines, and deep neural networks. The choice of the algorithm depends on the nature of our data and the problem.

1.K-Means Clustering: Segment products into clusters based on features like category or region.

Algorithm:

```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
# Load the sales data from a CSV file
data = pd.read_csv('statsfinal.csv')  # Adjust the filename as needed
# Select the features you want to cluster
features = data[['Q-P1', 'Q-P2']]
# Choose the number of clusters (K) based on your analysis goals
k = 3
# Create a K-Means model
kmeans = KMeans(n_clusters=k)
# Fit the model to the data
kmeans.fit(features)
# Add the cluster labels to the original dataset
data['Cluster'] = kmeans.labels_
# Visualize the clusters
plt.scatter(data['Q-P1'], data['Q-P2'], c=data['Cluster'], cmap='viridis')
plt.xlabel('Q-P1')
plt.ylabel('Q-P2')
plt.title('Product Sales Clusters (K-Means)')
plt.show()
```

Output:

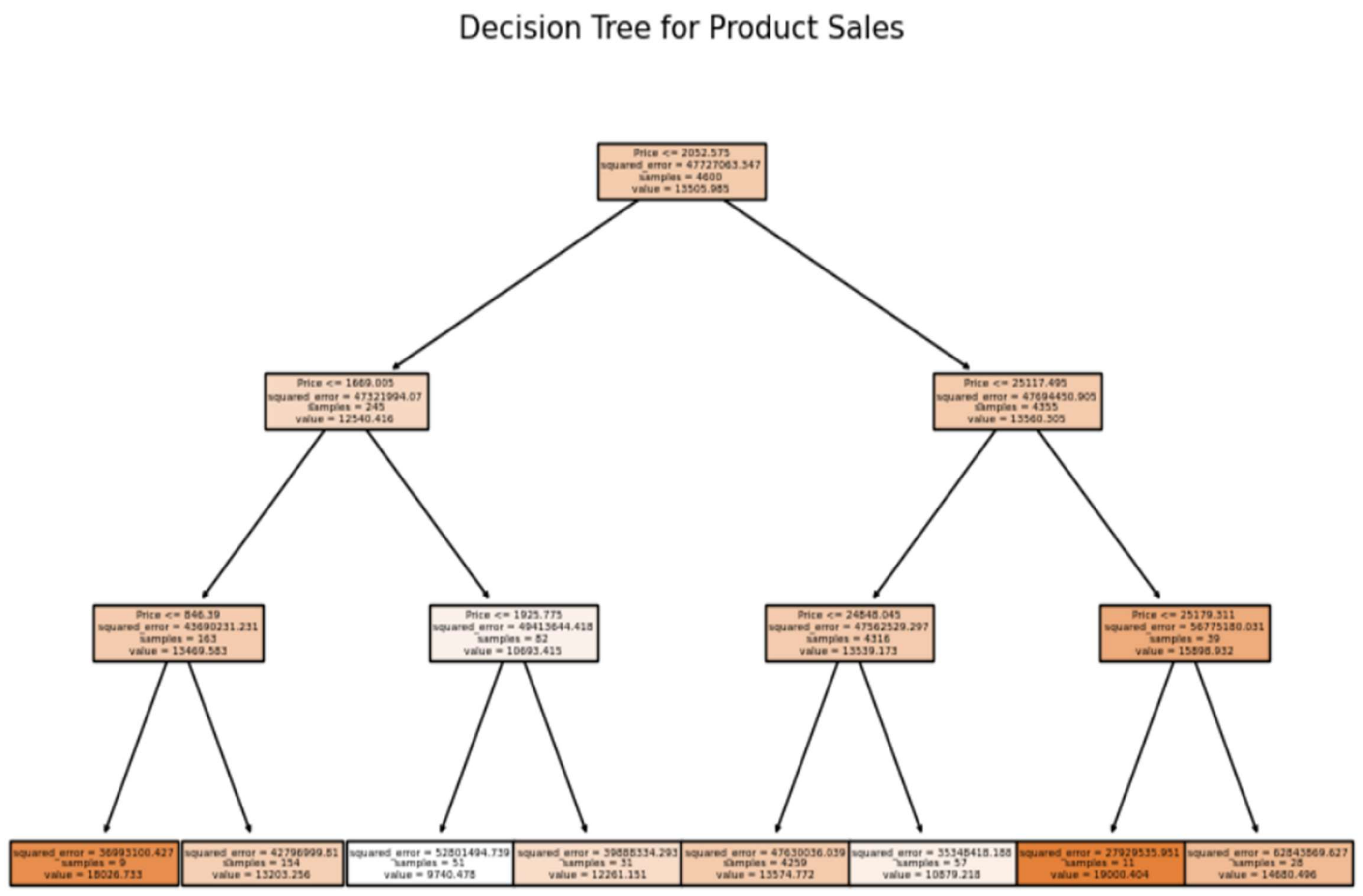


2.Decision Trees: Create product segments by splitting on attributes like price range or product type.

Algorithm:

```
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt
# Load the sales data from a CSV file
data = pd.read_csv('statsfinal.csv')  # Adjust the filename as needed
# Select the feature(s) influencing sales and the target variable
X = data[['S-P1']]
y = data['S-P2']
# Create a Decision Tree regressor model
tree_model = DecisionTreeRegressor(max_depth=3)  # Adjust the
max_depth as needed
# Fit the model to the data
tree_model.fit(X, y)
# Predict sales using the trained model
sales_predictions = tree_model.predict(X)
# Visualize the Decision Tree (Optional)
from sklearn.tree import plot_tree
plt.figure(figsize=(10, 6))
plot_tree(tree_model, filled=True, feature_names=['Price'])
plt.title('Decision Tree for Product Sales')
plt.show()
```

Output:



Recommendations:

Recommendations in product sales analysis typically involve suggesting actions or strategies to optimize sales, improve profitability, and enhance the customer experience. The specific recommendations can vary depending on the nature of business, data, and goals. Here are some common types of recommendations in product sales analysis:

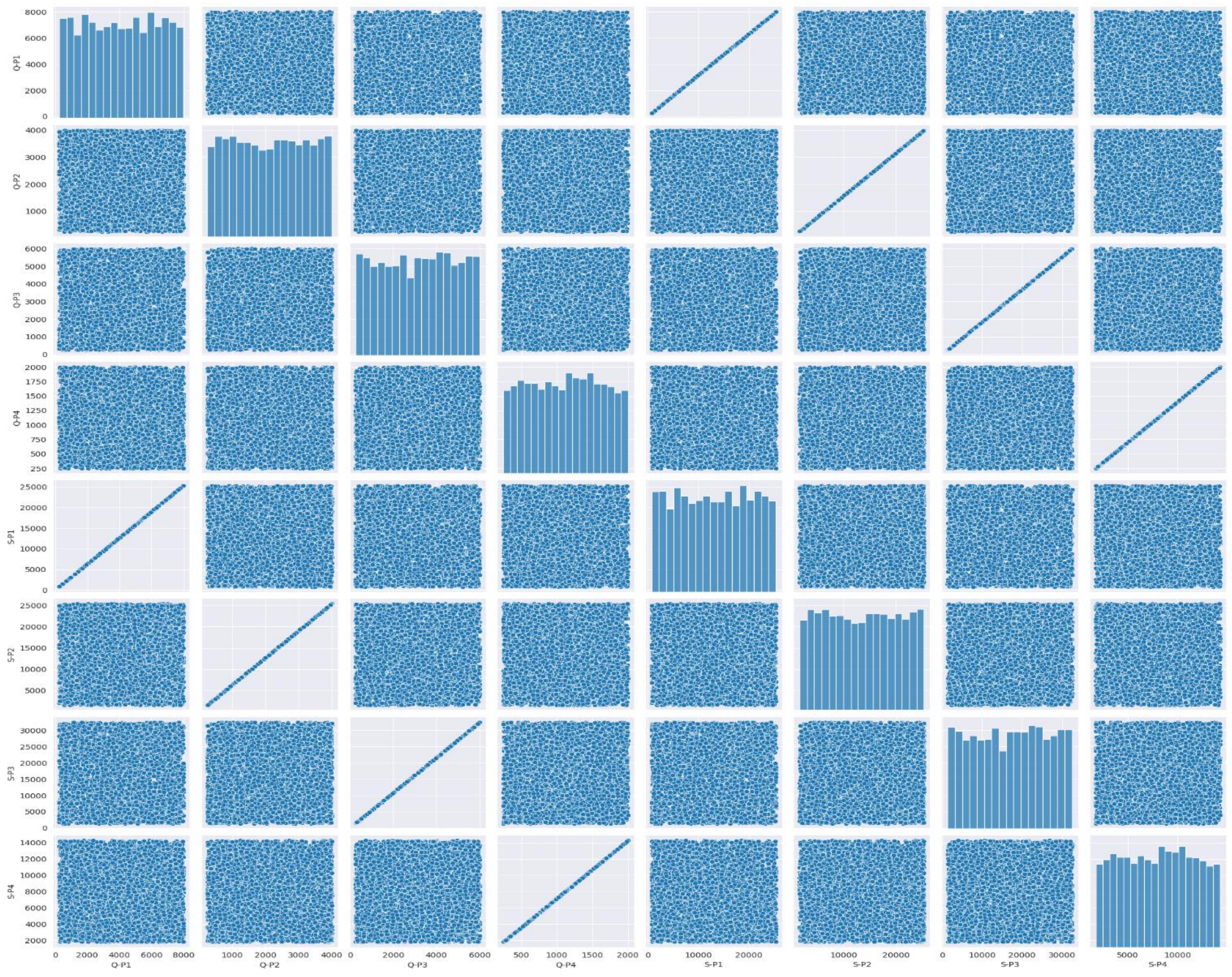
- **Rule-Based Recommendations:**

  Implement simple if-then rules to recommend related products or upsell/cross-sell items.

  Code:

  ```
  p1 = product_sales_df.drop(['Date','Unnamed: 0'], axis=1)
  sns.pairplot(p1)
  ```

  Output:

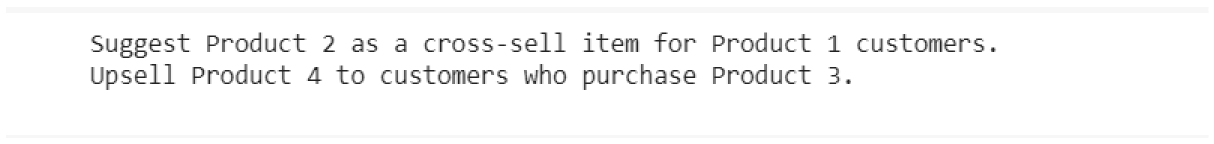


The above code is for visualize the relationships between the columns in the DataFrame. This can be useful for gaining insights into the relationships and correlations between different variables in your data.

Code:

```
product_sales = {
'Q1': product_sales_df['Q-P1'].sum(),
'Q2': product_sales_df['Q-P2'].sum(),
'Q3': product_sales_df['Q-P3'].sum(),
'Q4': product_sales_df['Q-P4'].sum(),
}
product_revenue = {
'S1': product_sales_df['S-P1'].sum(),
'S2': product_sales_df['S-P2'].sum(),
'S3': product_sales_df['S-P3'].sum(),
'S4': product_sales_df['S-P4'].sum(),
}
# Implement if-then rules for recommendations
recommendations = []

# If total unit sales of Product 1 are above a certain threshold, recommend Product 2
if product_sales['Q1'] > 4121: #4121 is threshold value
recommendations.append('Suggest Product 2 as a cross-sell item for Product 1 customers.')

# If total revenue from Product 3 is high, recommend Product 4
if product_revenue['S3'] > 17049:   #17049 is threshold value
recommendations.append('Upsell Product 4 to customers who purchase Product 3.')

# Print the recommendations
for recommendation in recommendations:
print(recommendation)
```

Output:



```
Suggest Product 2 as a cross-sell item for Product 1 customers.
Upsell Product 4 to customers who purchase Product 3.
```

Rule-based recommendations in product sales analysis involve using predefined business rules or conditions to suggest related products, cross-sell items, upsell opportunities, or other actions based on the sales data

- **Basic Collaborative Filtering:**

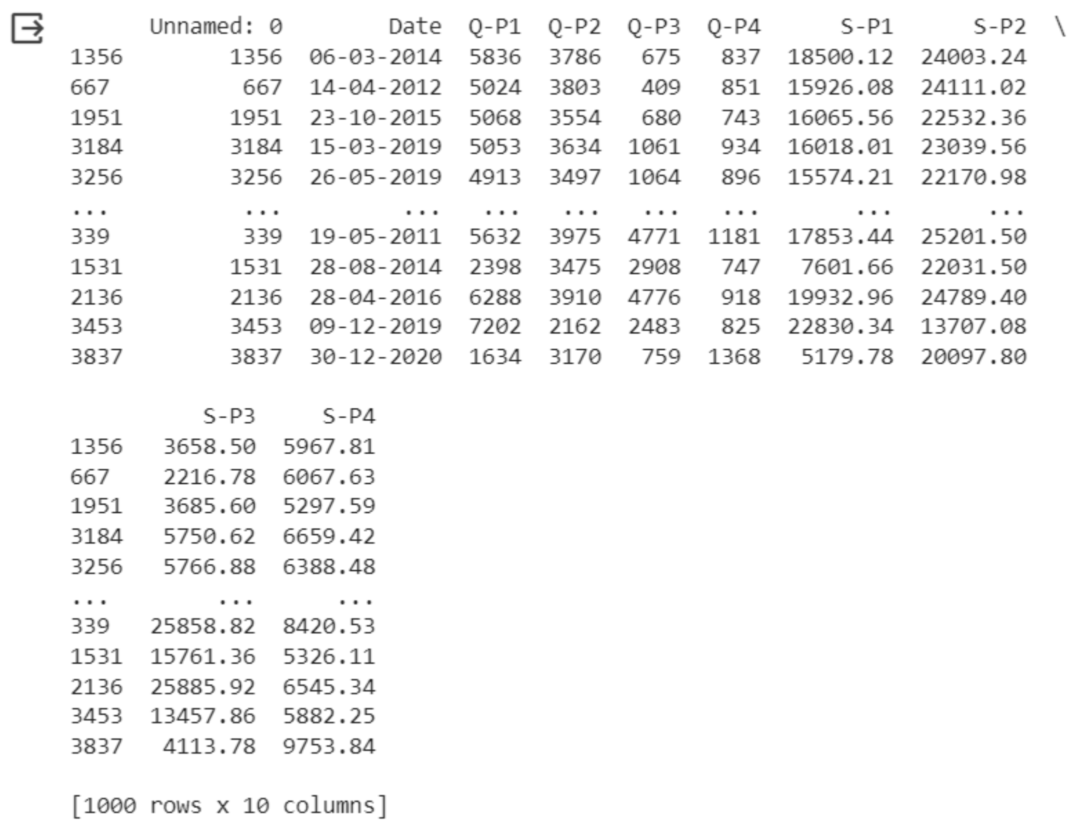
Collaborative Filtering is a popular technique used in recommendation systems to make product recommendations based on user behavior and preferences. In the context of product sales analysis,we can apply basic collaborative filtering to recommend products to customers based on the behavior of other customers.

Use a similarity metric (e.g., cosine similarity) to recommend products based on user behavior.

Code:

```
from sklearn.metrics.pairwise import cosine_similarity


data = product_sales_df[['Q-P1', 'Q-P2' , 'Q-P3' , 'Q-P4' , 'S-P1' , 'S-P2' , 'S-P3' , 'S-P4']]

# Normalize the data
normalized_data = (data - data.min()) / (data.max() - data.min())

# Calculate cosine similarity
cosine_sim = cosine_similarity(normalized_data, normalized_data)

# Create a DataFrame for the cosine similarity matrix
cosine_sim_df = pd.DataFrame(cosine_sim, index=product_sales_df.index,
columns=product_sales_df.index)

# Recommend products based on user behavior
def recommend_products(product_index, n=1000):
    sim_scores = cosine_sim_df[product_index]
    product_indices = sim_scores.nlargest(n + 1).index[1:]  # Exclude the product itself
    recommended_products = product_sales_df.iloc[product_indices]
    return recommended_products
# Example: Recommend products similar to the first product (index 0)
recommended_products = recommend_products(0)
print(recommended_products)
```

Output:



```
      Unnamed: 0        Date  Q-P1  Q-P2  Q-P3  Q-P4      S-P1      S-P2  \
1356        1356  06-03-2014  5836  3786   675   837  18500.12  24003.24
667          667  14-04-2012  5024  3803   409   851  15926.08  24111.02
1951        1951  23-10-2015  5068  3554   680   743  16065.56  22532.36
3184        3184  15-03-2019  5053  3634  1061   934  16018.01  23039.56
3256        3256  26-05-2019  4913  3497  1064   896  15574.21  22170.98
...          ...         ...   ...   ...   ...   ...       ...       ...
339          339  19-05-2011  5632  3975  4771  1181  17853.44  25201.50
1531        1531  28-08-2014  2398  3475  2908   747   7601.66  22031.50
2136        2136  28-04-2016  6288  3910  4776   918  19932.96  24789.40
3453        3453  09-12-2019  7202  2162  2483   825  22830.34  13707.08
3837        3837  30-12-2020  1634  3170   759  1368   5179.78  20097.80

          S-P3     S-P4
1356   3658.50  5967.81
667    2216.78  6067.63
1951   3685.60  5297.59
3184   5750.62  6659.42
3256   5766.88  6388.48
...        ...      ...
339   25858.82  8420.53
1531  15761.36  5326.11
2136  25885.92  6545.34
3453  13457.86  5882.25
3837   4113.78  9753.84

[1000 rows x 10 columns]
```

In essence, this code is a simple implementation of business rules for product recommendations. It suggests specific product recommendations or upselling opportunities to customers based on their past purchase behavior and the predefined conditions and thresholds set in the code.

Continuous Improvement:

Continuous improvement in product sales analysis is a vital process to enhance your business's ability to understand market trends, make informed decisions, and increase profitability.

- **Basic A/B Testing:**

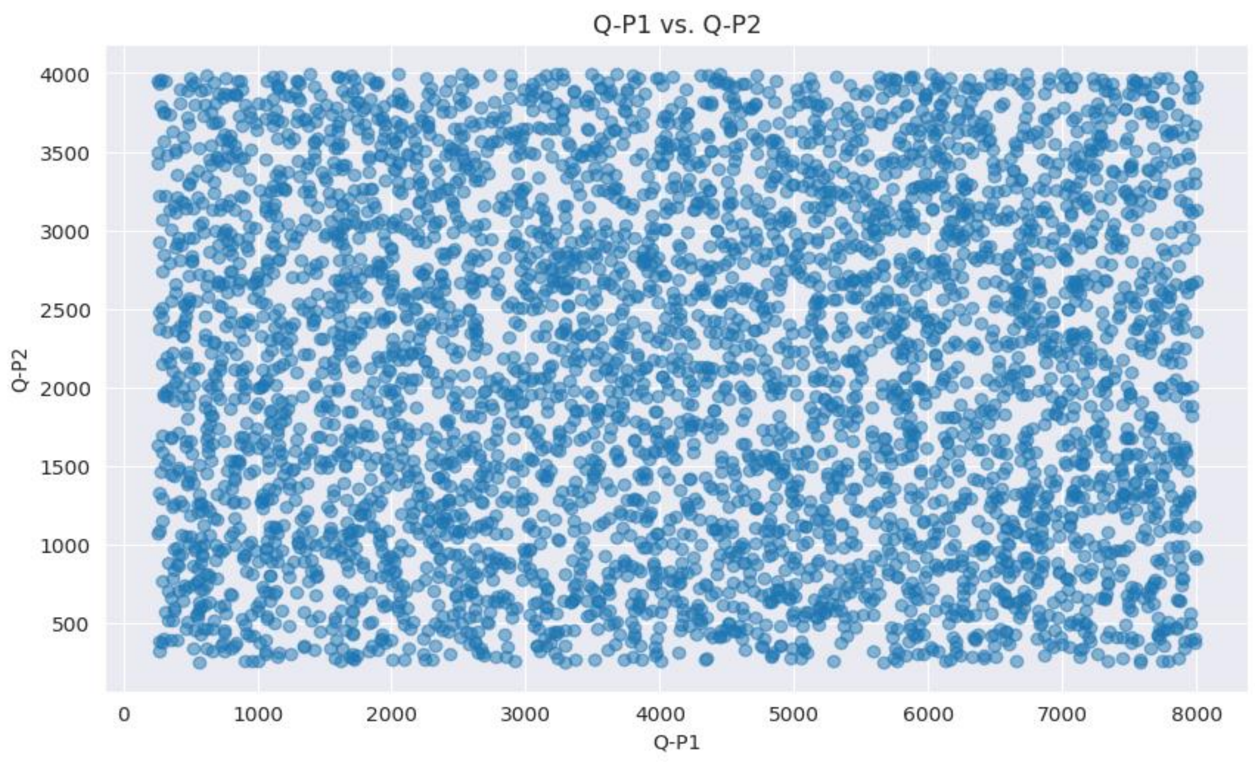
  Conduct A/B tests to compare the effectiveness of different marketing strategies or pricing changes.

  Basic A/B testing in product sales analysis is a statistical method used to compare the performance of two versions (A and B) of a product, feature, or marketing strategy to determine which one is more effective in driving sales. The goal is to understand whether a change or variation leads to a statistically significant difference in sales performance

Code:

```
column1_name = 'Q-P1'
column2_name = 'Q-P2'

# Extract data from the DataFrame
data_column1 = product_sales_df[column1_name]
data_column2 = product_sales_df[column2_name]

# Create a graph to compare the two columns
plt.figure(figsize=(10, 6))  # Adjust the figure size as needed
plt.scatter(data_column1, data_column2, alpha=0.5)  # alpha sets transparency

plt.xlabel(column1_name)
plt.ylabel(column2_name)
plt.title(f'{column1_name} vs. {column2_name}')
plt.grid(True)  # Add gridlines if desired

plt.show()
```

Output:



It create a visual representation of the relationship between the two specified columns, making it easier to analyze and interpret the data. It's a common practice in data analysis to use plots and graphs to gain insights into the relationships and patterns in data.

- **Rule-Based Adjustments:**
  Adjust strategies manually based on observed results.

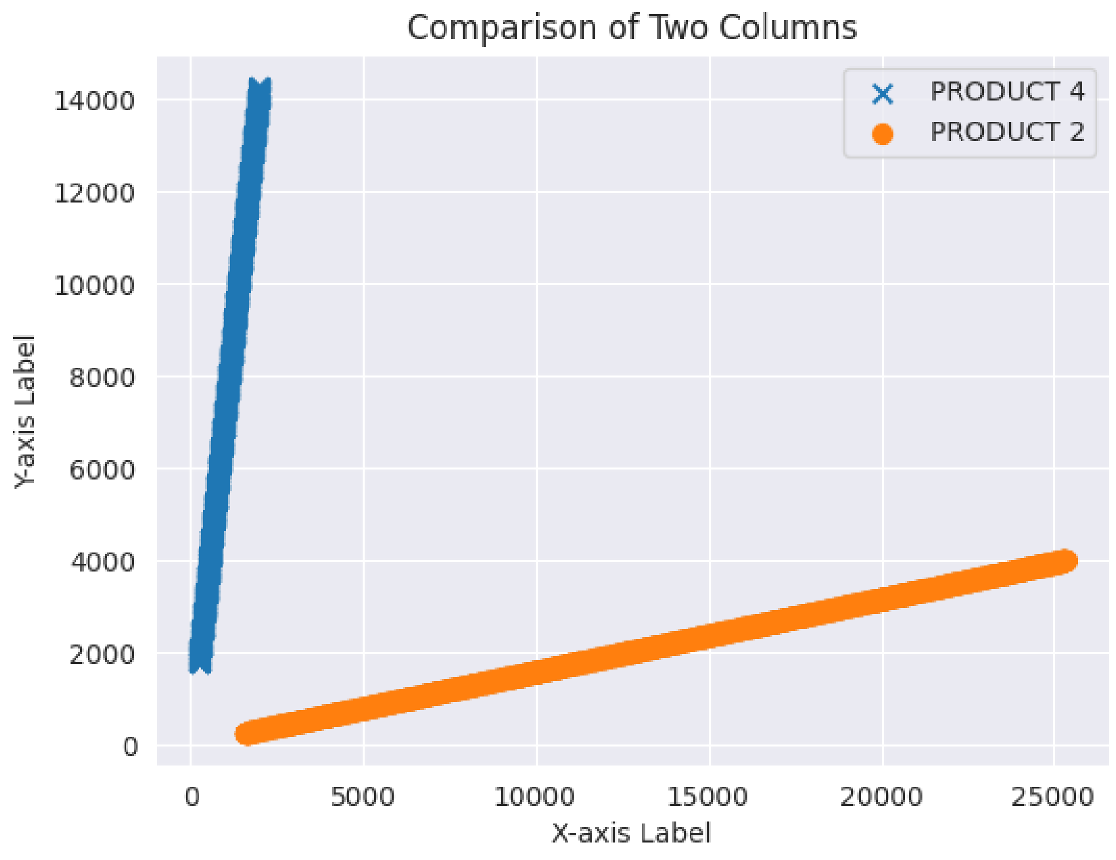
Rule-based adjustments in product sales analysis involve applying specific rules or conditions to modify or enhance your sales analysis based on predefined criteria. These adjustments can help you gain deeper insights into your product sales data and make more informed decisions.

Code:

```
# Extract data from the DataFrame

x1 = product_sales_df['Q-P4']

y1 = product_sales_df['S-P4']

x2 = product_sales_df['S-P2']

y2 = product_sales_df['Q-P2']
```

```
# Create a scatter plot
plt.scatter(x1, y1, label='PRODUCT 4', marker='x', s=50)
plt.scatter(x2, y2, label='PRODUCT 2', marker='o', s=50)

# Add labels and a legend
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')
plt.title('Comparison of Four Columns')
plt.legend()

# Show the graph
plt.show()
```

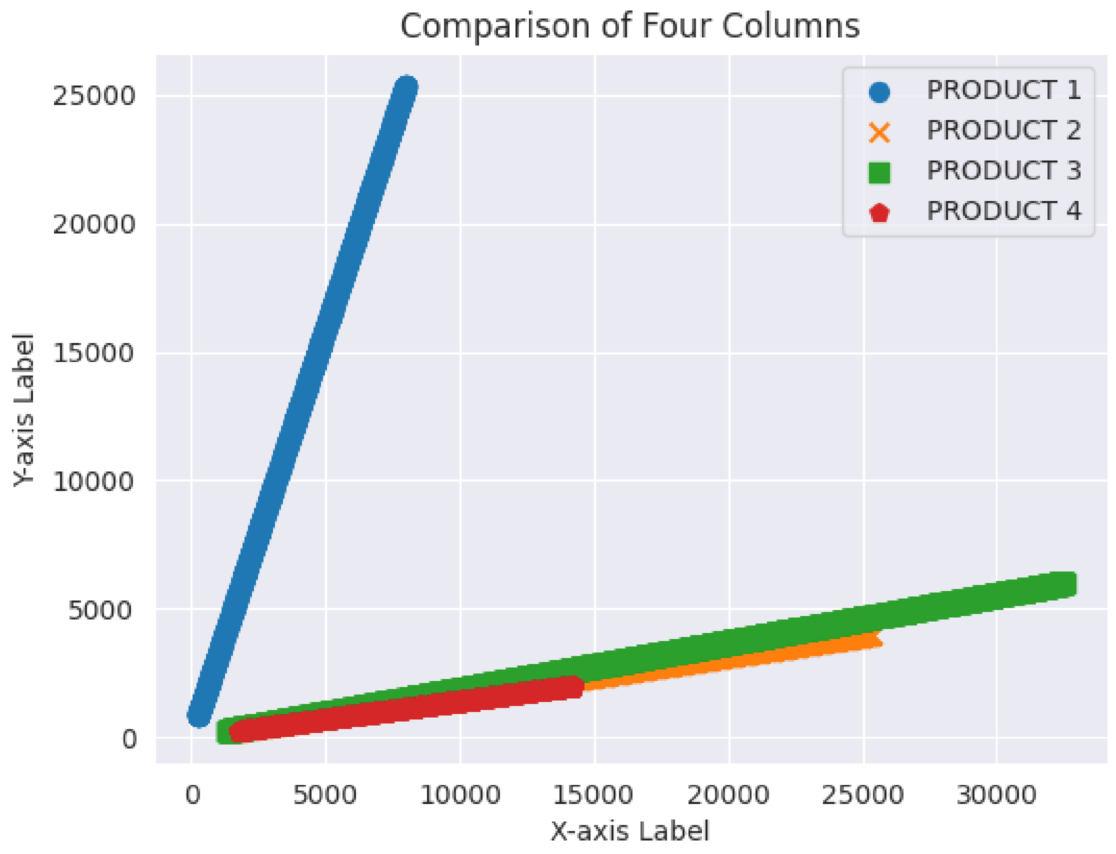
Output:



Code:

```
# Extract data from the DataFrame
x1 = product_sales_df['Q-P1']
y1 = product_sales_df['S-P1']
x2 = product_sales_df['S-P2']
y2 = product_sales_df['Q-P2']
x3 = product_sales_df['S-P3']
y3 = product_sales_df['Q-P3']
x4 = product_sales_df['S-P4']
y4 = product_sales_df['Q-P4']

# Create a scatter plot
plt.scatter(x1, y1, label='PRODUCT 1', marker='o', s=50)
plt.scatter(x2, y2, label='PRODUCT 2', marker='x', s=50)
plt.scatter(x3, y3, label='PRODUCT 3', marker='s', s=50)
plt.scatter(x4, y4, label='PRODUCT 4', marker='p', s=50)

# Add labels and a legend
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')
plt.title('Comparison of Four Columns')
plt.legend()

# Show the graph
plt.show()
```

Output:



The scatter plot created by this code allows you to visually compare the sales and revenue data for the four products. It helps in identifying any patterns or relationships between sales and revenue for each product and can provide insights into how each product is performing in terms of generating revenue.

From the result of above code, we conclude that stronger positive correlation between the quantity (Q) and sales (S) for "PRODUCT 1" compared to the other products. In other words, an increase in the quantity of "PRODUCT 1" is associated with a proportionally larger increase in sales for that product.

This could be a positive sign for "PRODUCT 1" in the sense that increasing the quantity of this product may lead to higher sales and revenue. On the other hand, for the other products (PRODUCT 2, PRODUCT 3, and PRODUCT 4), the relationship between quantity and sales may not be as strong or linear.