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 :

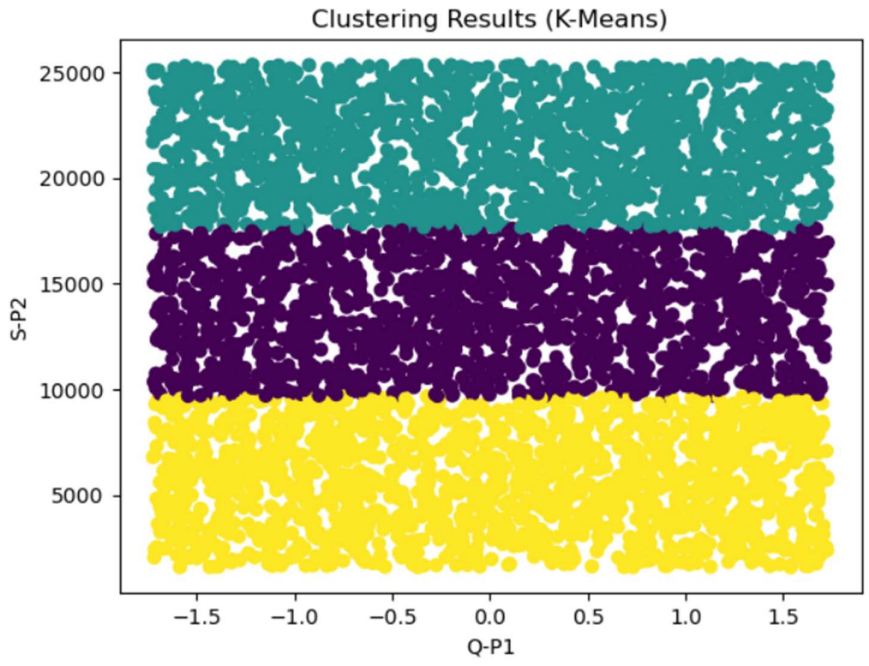
```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

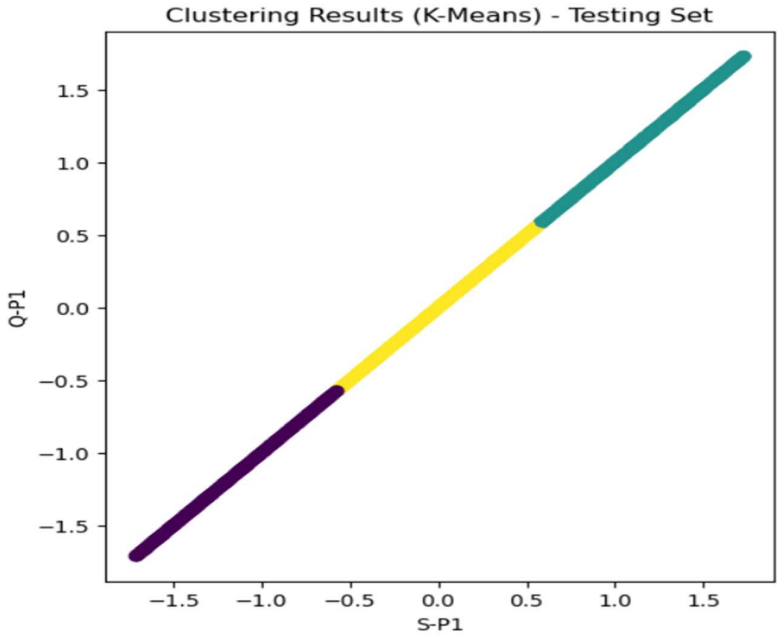
```
from sklearn.model_selection import train_test_split
# Load the sales data from a CSV file
data = pd.read_csv('statsfinal.csv')  # Adjust the filename as needed
# Step 1: Data Preprocessing
# Remove rows with missing values or replace them with appropriate values
data.dropna(subset=['S-P1', 'Q-P1'], 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')
```

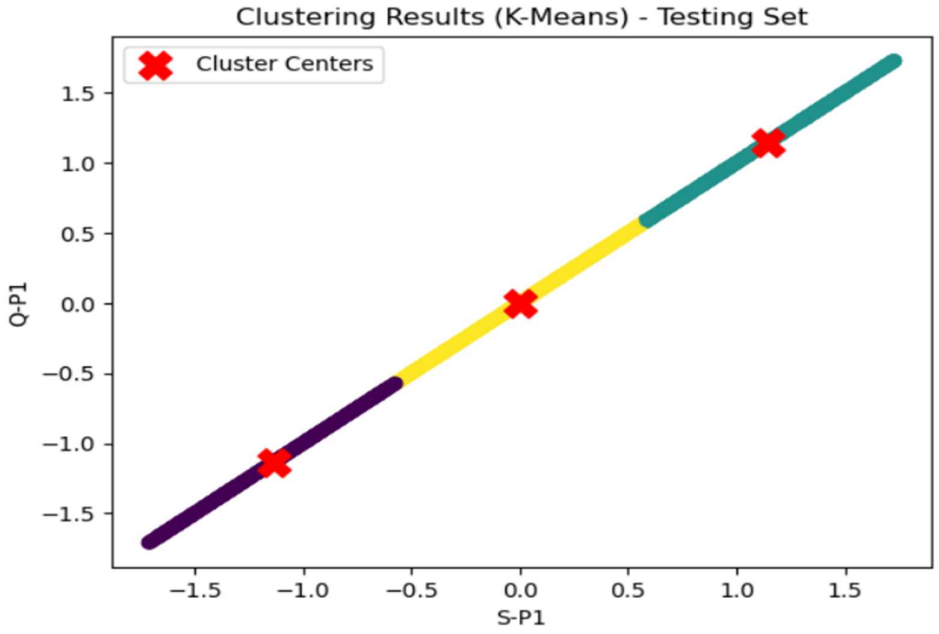
```
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], marker='X', s=200, c='red', label='Cluster Centers')

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')

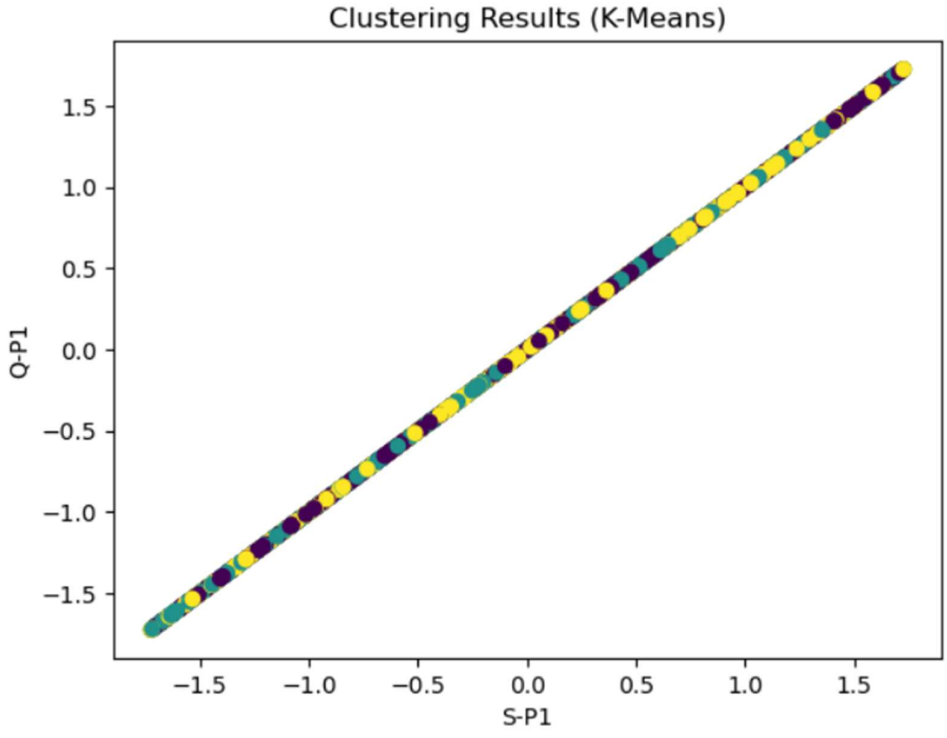
plt.ylabel('Q-P1')

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:

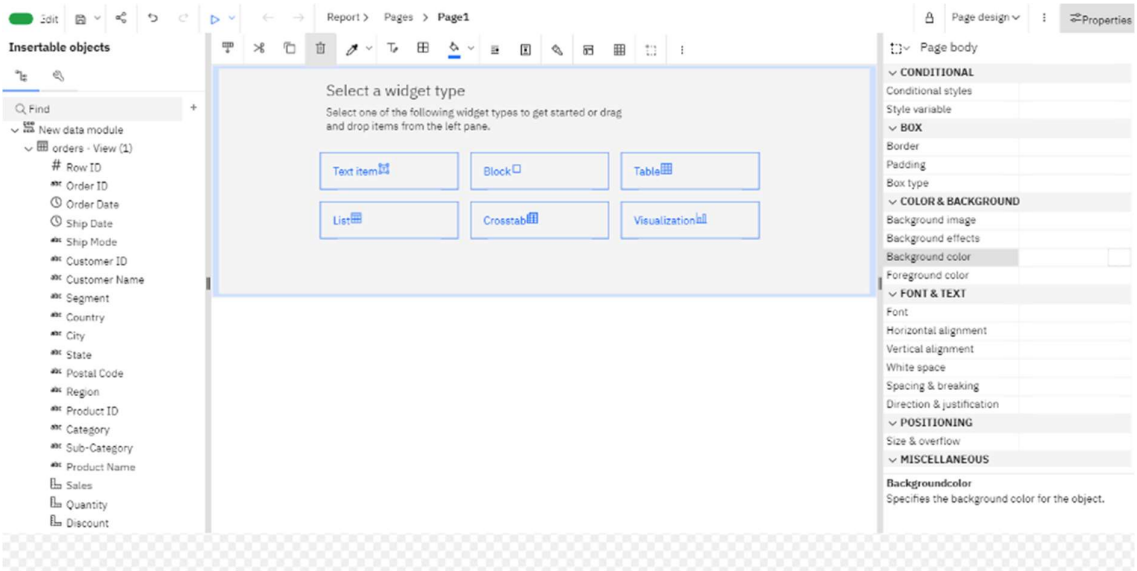
- Configure the visualization to represent the clustering results. This may include setting the X and Y axes (if applicable), color-coding data points by cluster label, and configuring labels or tooltips to display information about each data point.

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.



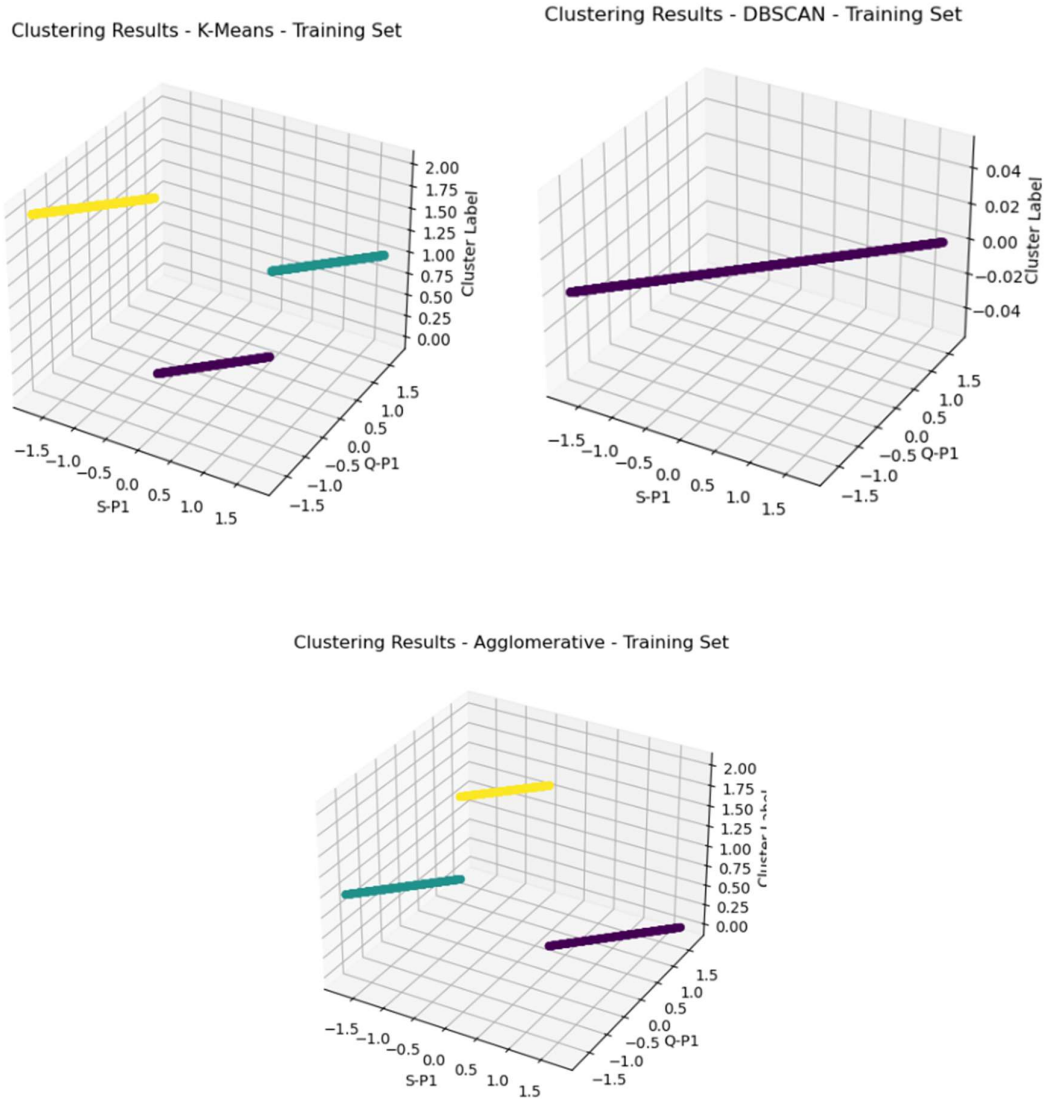
Can be viewed using any one of the Widget type.

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: