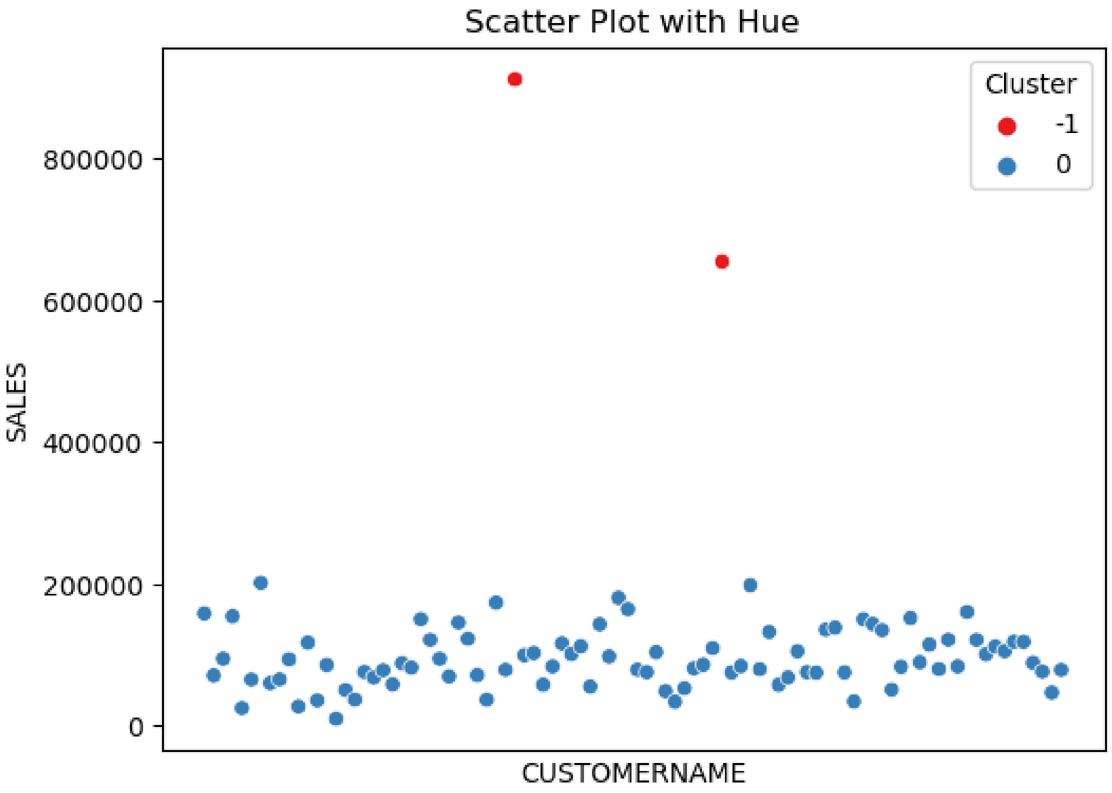
In [16]:

```
import seaborn as sns

# Scatter plot with hue
sns.scatterplot(x='CUSTOMERNAME', y='SALES', hue='Cluster',palette='Set1',
# Remove X-axis labels
plt.xticks([])
plt.title('Scatter Plot with Hue')
plt.show()
```



FILTER THE OUTLIERS

In [17]:

```
# Remove rows with 'Cluster' column value equal to -1
df_filtered = df[df['Cluster'] != -1]
```

In [18]:

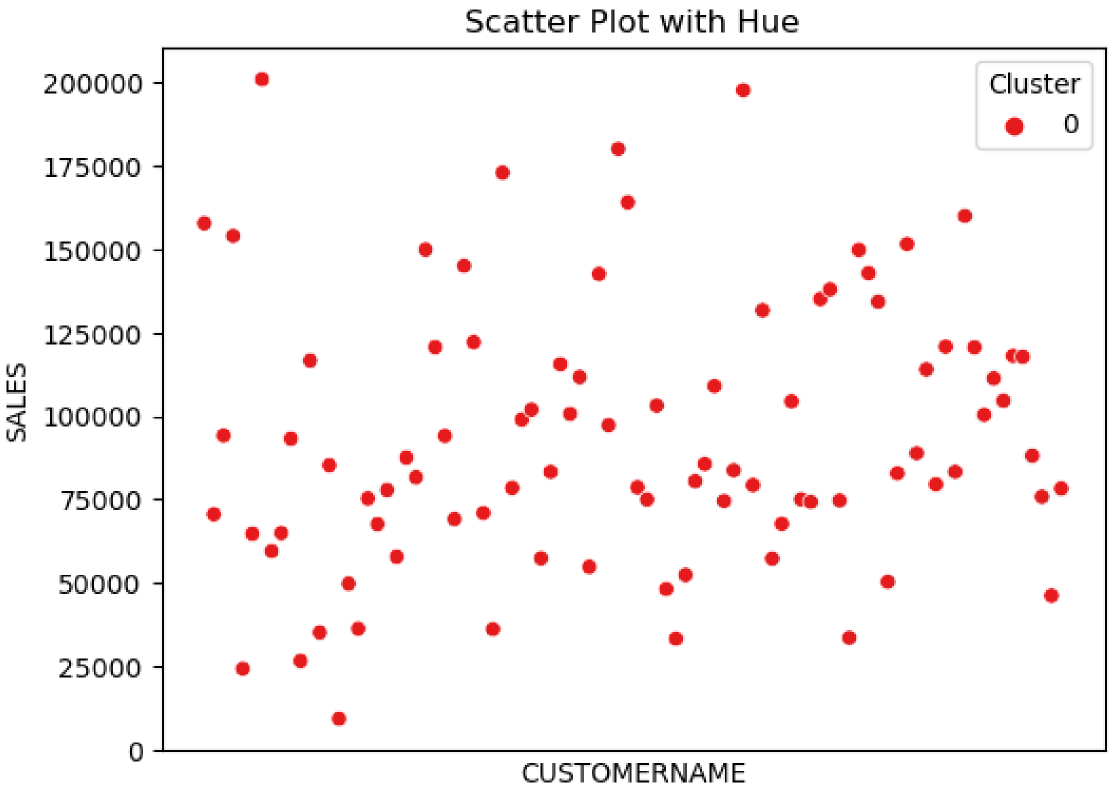
```
df = df_filtered
df
```

Out[18]:

```
                             QUANTITYORDERED  PRICEEACH      SALES  Cluster
                CUSTOMERNAME
              AV Stores, Co.            1778    3975.33  157807.81        0
               Alpha Cognac              687    1701.95   70488.44        0
          Amica Models & Co.             843    2218.41   94117.26        0
   Anna's Decorations, Ltd              1469    3843.67  153996.13        0
           Atelier graphique             270     558.43   24179.96        0
                         ...             ...        ...        ...      ...
             Vida Sport, Ltd            1078    2713.09  117713.56        0
            Vitachrome Inc.              787    2108.11   88041.26        0
   Volvo Model Replicas, Co              647    1720.14   75754.88        0
West Coast Collectables Co.              511    1030.99   46084.64        0
           giftsbymail.co.uk             895    2131.78   78240.84        0

90 rows × 4 columns
```

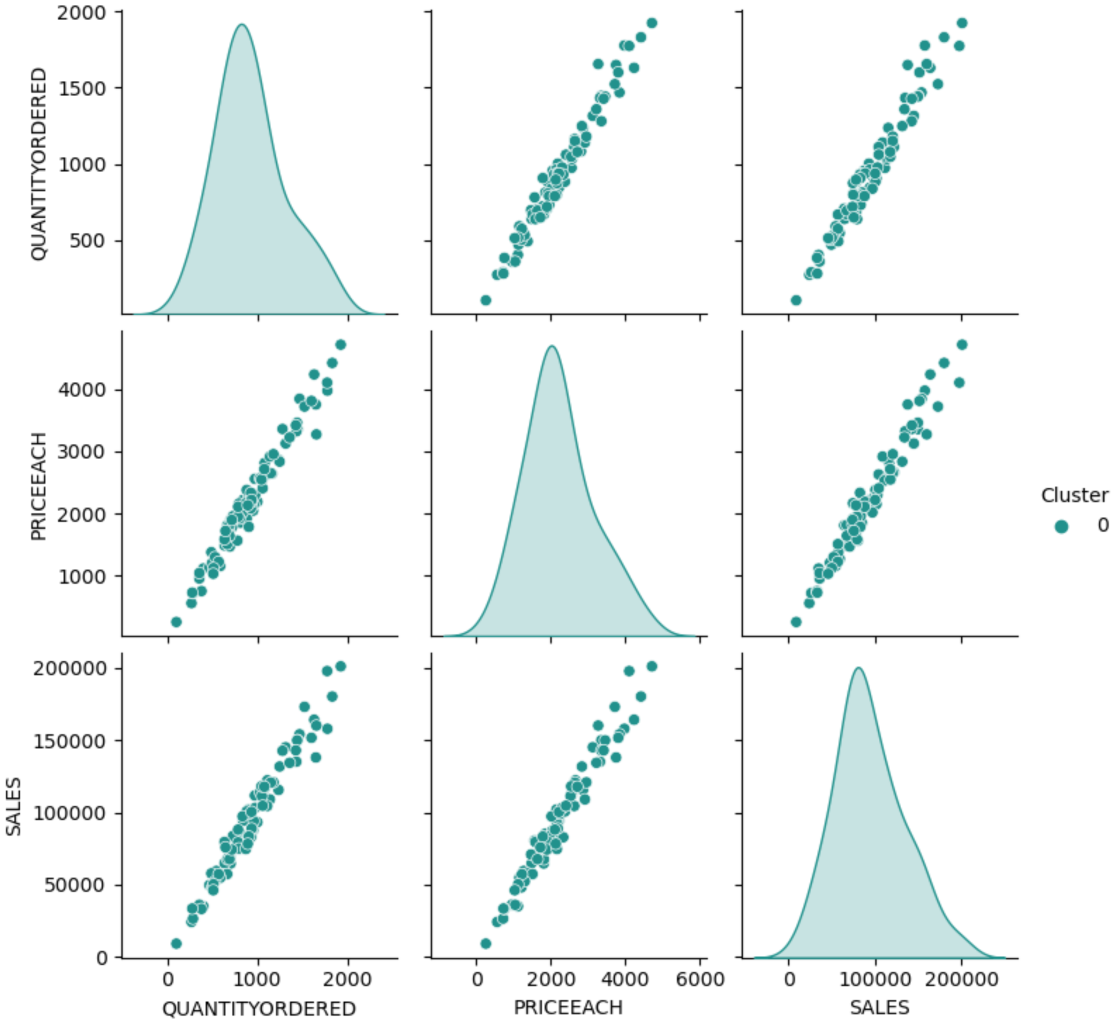
In [19]:

```
import seaborn as sns

# Scatter plot with hue
sns.scatterplot(x='CUSTOMERNAME', y='SALES', hue='Cluster',palette='Set1',
# Remove X-axis labels
plt.xticks([])
plt.title('Scatter Plot with Hue')
plt.show()
```



In [20]:

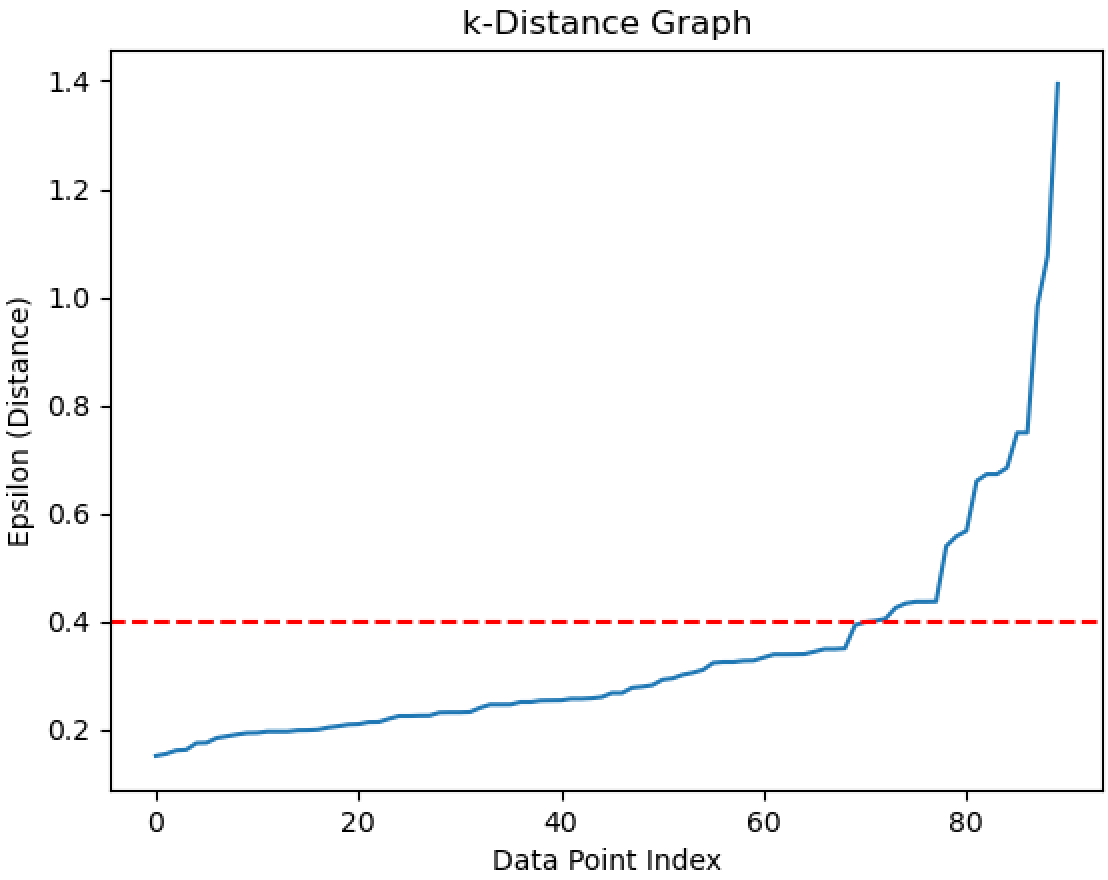
```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'hue_column' is the column you want to use as the hue
sns.pairplot(df, hue='Cluster', palette='viridis')
plt.show()
```



ONCE THE OUTLIERS ARE OUT RE DO THE EXPERIMENT

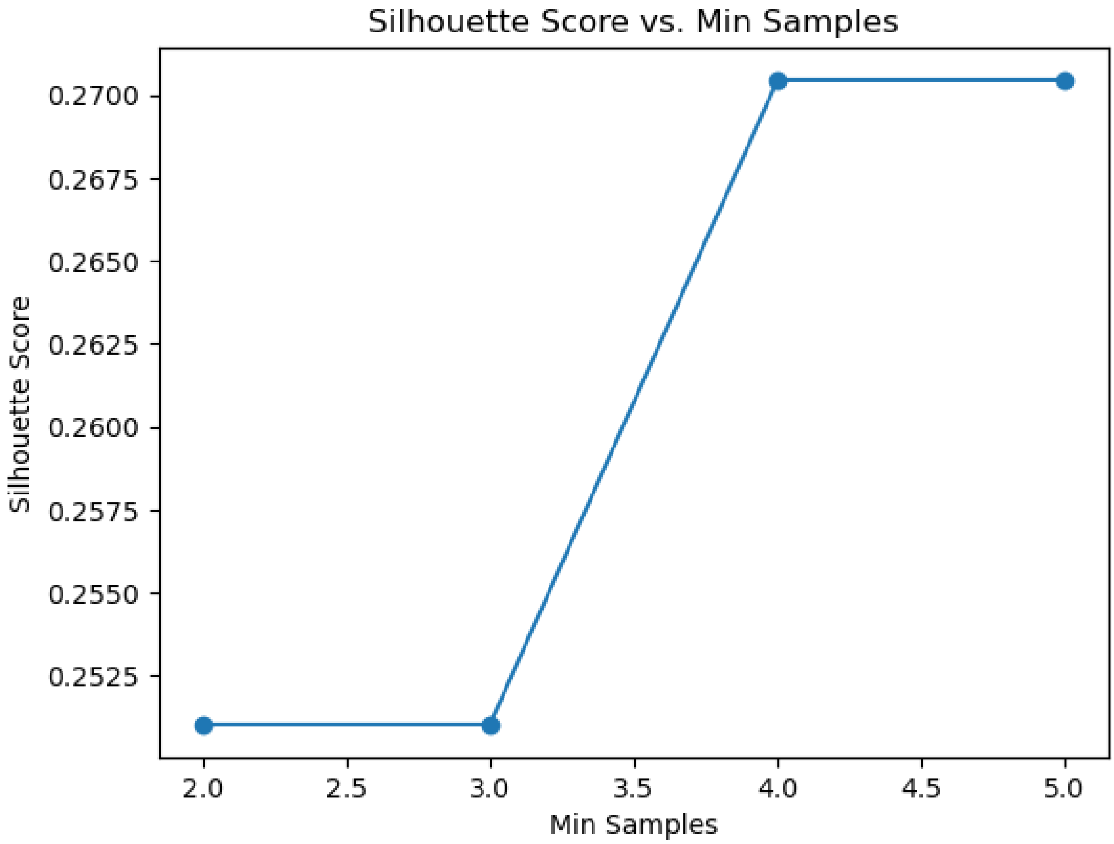
In [21]:

```
from sklearn.neighbors import NearestNeighbors
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler



# Assuming you have 'numerical_data' from the previous code
# Extract numerical columns from the DataFrame
numerical_columns = df.select_dtypes(include=['float64', 'int64']).columns

# Create a DataFrame containing only numerical data
numerical_data = df[numerical_columns]

# Standardize the data (mean=0 and variance=1)
scaler = StandardScaler()
scaled_data = scaler.fit_transform(numerical_data)

# Fit a Nearest Neighbors model to compute distances
neighbors_model = NearestNeighbors(n_neighbors=5)  # You can adjust the num
neighbors_model.fit(scaled_data)
distances, _ = neighbors_model.kneighbors(scaled_data)

# Sort the distances
sorted_distances = np.sort(distances[:, -1])

# Plot the k-distance graph
plt.plot(sorted_distances)
plt.axhline(y=0.4, color='red', linestyle='--', label='Threshold at 0.5')
plt.xlabel('Data Point Index')
plt.ylabel('Epsilon (Distance)')
plt.title('k-Distance Graph')
plt.show()
```


k-Distance Graph
Epsilon (Distance)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
20
40
60
80
Data Point Index

In [22]:

```
from sklearn.metrics import silhouette_score
from sklearn.cluster import DBSCAN


silhouette_scores = []
min_samples_values = range(2, 6)  # Adjust the range based on your data

for min_samples_val in min_samples_values:
    dbscan = DBSCAN(eps=0.4, min_samples=min_samples_val)
    labels = dbscan.fit_predict(scaled_data)
    silhouette_scores.append(silhouette_score(scaled_data, labels))

# Plot silhouette scores
plt.plot(min_samples_values, silhouette_scores, marker='o')
plt.xlabel('Min Samples')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score vs. Min Samples')
plt.show()

# Choose the min_samples with the highest silhouette score
optimal_min_samples = min_samples_values[np.argmax(silhouette_scores)]
print(f"Optimal Min Samples: {optimal_min_samples}")
```



```
Optimal Min Samples: 4
```

In [23]:

```
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Assuming you have a DataFrame named 'df' with numerical data
# If your data contains non-numerical columns, you may need to preprocess t

# Extract numerical columns from the DataFrame
numerical_columns = df.select_dtypes(include=['float64', 'int64']).columns

# Create a DataFrame containing only numerical data
numerical_data = df[numerical_columns]

# Standardize the data (mean=0 and variance=1)
scaler = StandardScaler()
scaled_data = scaler.fit_transform(numerical_data)

# Choose the epsilon and min_samples based on your analysis
epsilon = 0.4 # Adjust based on your data
min_samples = 4  # Adjust based on your data

# Create a DBSCAN object
dbscan = DBSCAN(eps=epsilon, min_samples=min_samples)

# Fit and predict clusters
labels = dbscan.fit_predict(scaled_data)

# Add the cluster labels to the original DataFrame
df['Cluster'] = labels

# Display the clusters
print("Clusters:")
print(df['Cluster'].value_counts())

# Plot the clusters (assuming 2D or 3D data)
if numerical_data.shape[1] == 2:
    plt.scatter(numerical_data.iloc[:, 0], numerical_data.iloc[:, 1], c=lab
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('DBSCAN Clustering')
    plt.show()
elif numerical_data.shape[1] == 3:
    from mpl_toolkits.mplot3d import Axes3D
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(numerical_data.iloc[:, 0], numerical_data.iloc[:, 1], numeri
    ax.set_xlabel('Feature 1')
    ax.set_ylabel('Feature 2')
    ax.set_zlabel('Feature 3')
    ax.set_title('DBSCAN Clustering')
    plt.show()
else:
    print("Can't visualize clusters for more than 3 dimensions.")
```

```
Clusters:
 0    65
-1    11
 1     7
 2     7
Name: Cluster, dtype: int64
Can't visualize clusters for more than 3 dimensions.

C:\Users\castr\AppData\Local\Temp\ipykernel_20392\1299169578.py:29: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df['Cluster'] = labels
```

In [24]:
```
df
```

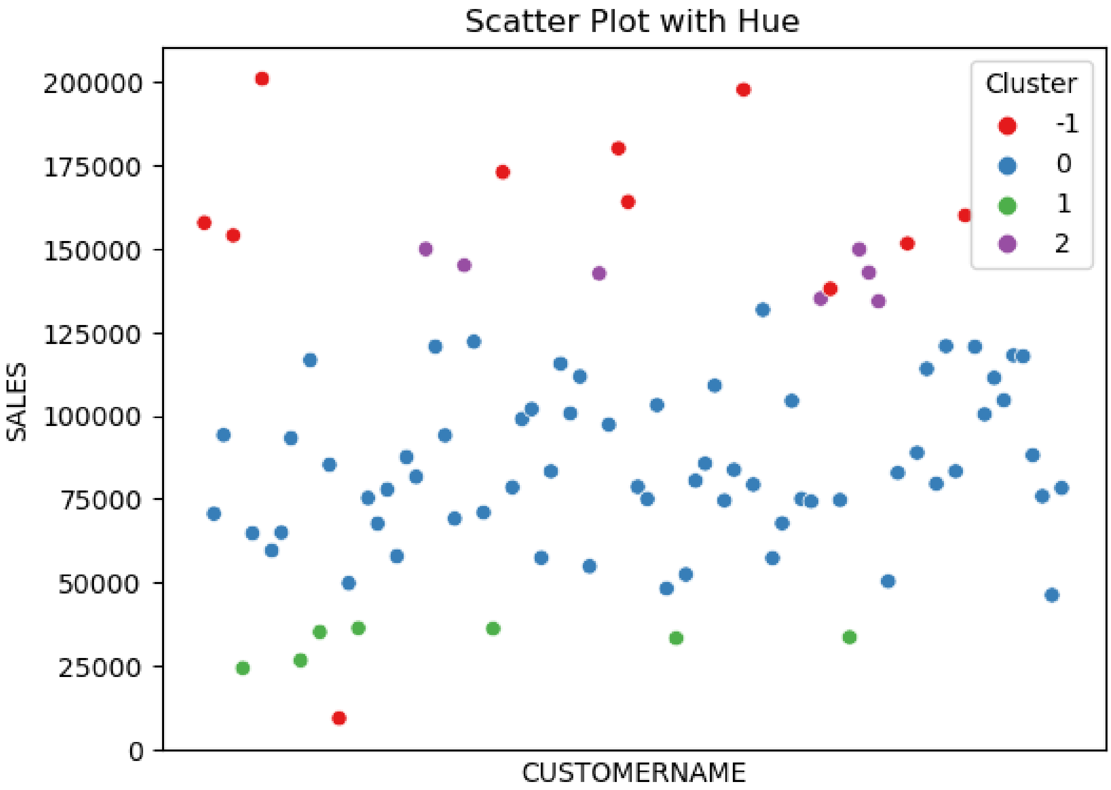
Out[24]:

	QUANTITYORDERED	PRICEEACH	SALES	Cluster
CUSTOMERNAME				
AV Stores, Co.	1778	3975.33	157807.81	-1
Alpha Cognac	687	1701.95	70488.44	0
Amica Models & Co.	843	2218.41	94117.26	0
Anna's Decorations, Ltd	1469	3843.67	153996.13	-1
Atelier graphique	270	558.43	24179.96	1
...	...	...	...	...
Vida Sport, Ltd	1078	2713.09	117713.56	0
Vitachrome Inc.	787	2108.11	88041.26	0
Volvo Model Replicas, Co	647	1720.14	75754.88	0
West Coast Collectables Co.	511	1030.99	46084.64	0
giftsbymail.co.uk	895	2131.78	78240.84	0

90 rows × 4 columns

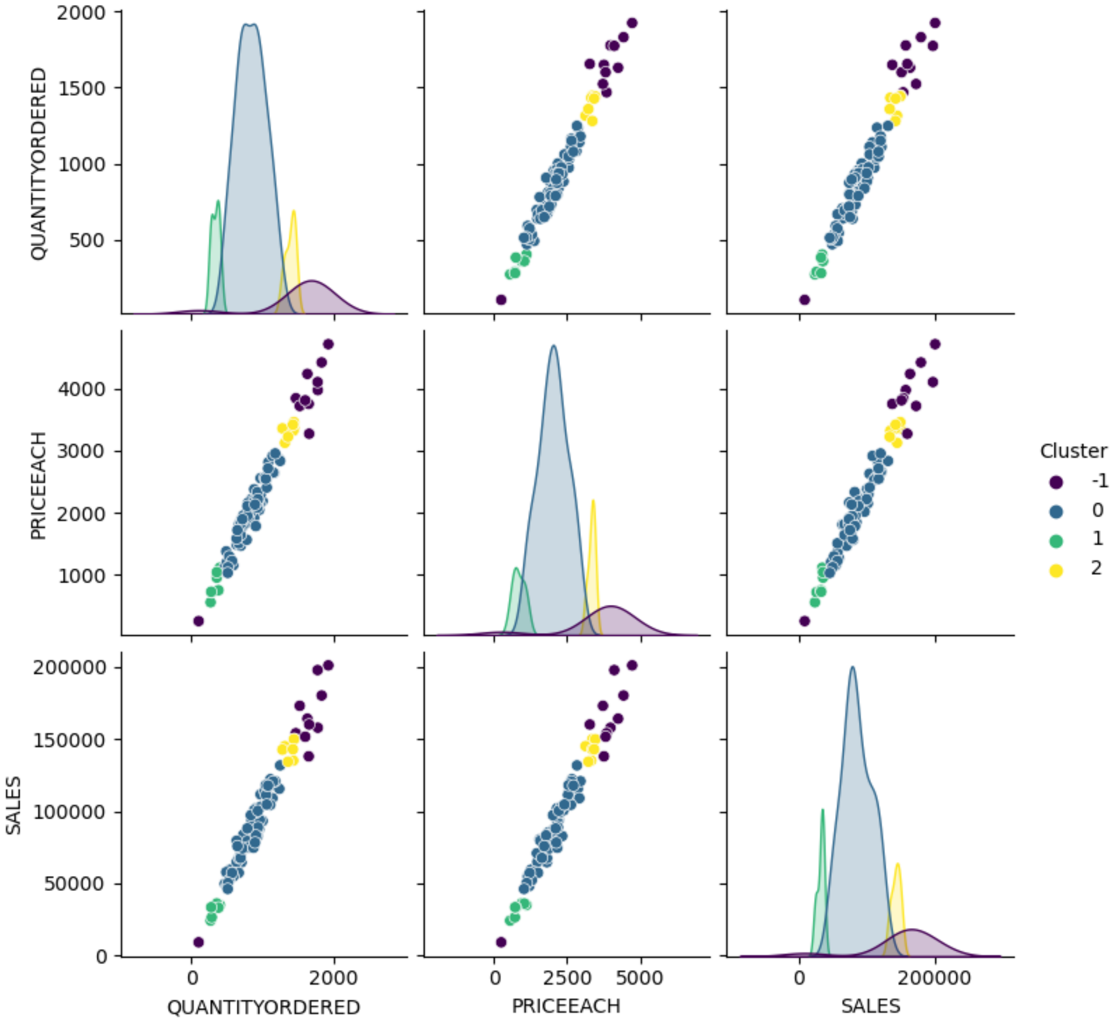
In [25]:

```
# Scatter plot with hue
sns.scatterplot(x='CUSTOMERNAME', y='SALES', hue='Cluster',palette='Set1',
# Remove X-axis labels
plt.xticks([])
plt.title('Scatter Plot with Hue')
plt.show()
```



In [26]:

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

# Assuming 'hue_column' is the column you want to use as the hue
sns.pairplot(df, hue='Cluster', palette='viridis')
plt.show()
```



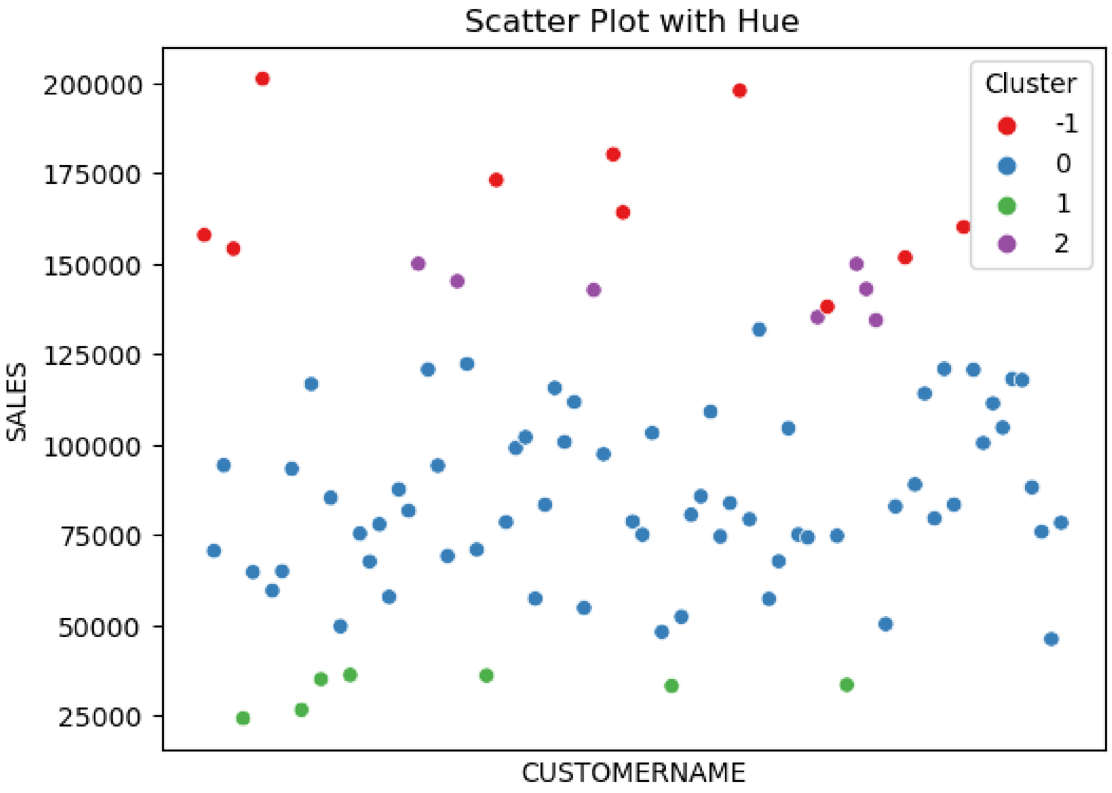
In [27]:

```
# Find the index of the row with the lowest value in the 'SALES' column
index_of_min_sales = df['SALES'].idxmin()

# Drop the row with the lowest value
df_without_min_sales = df.drop(index_of_min_sales)

df = df_without_min_sales
```

RESULT: WE HAVE 4 DIFFERENT GROUPS OF CUSTOMERS

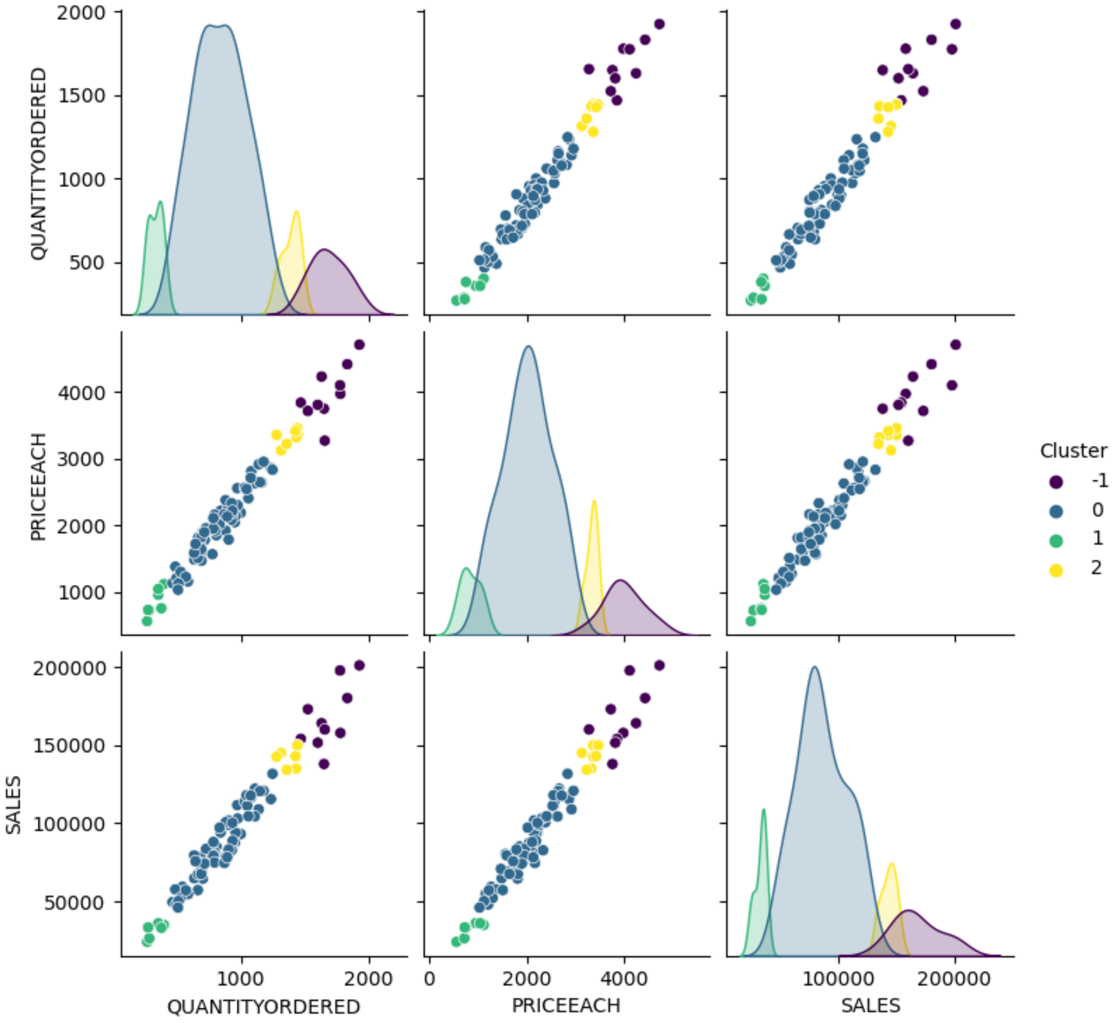
In [28]:

```python
# Scatter plot with hue
sns.scatterplot(x='CUSTOMERNAME', y='SALES', hue='Cluster',palette='Set1',
# Remove X-axis labels
plt.xticks([])
plt.title('Scatter Plot with Hue')
plt.show()
```



In [29]:

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

# Assuming 'hue_column' is the column you want to use as the hue
sns.pairplot(df, hue='Cluster', palette='viridis')
plt.show()
```



In [ ]: