Superstore Sales Analysis Machine Learning With Different Clusters

In [2]:

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

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from scipy.cluster.hierarchy import linkage, dendrogram
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.metrics import silhouette_score
```

In [3]:

```
## Surpress the Warnings
import warnings
warnings.filterwarnings('ignore')
```

In [4]:

```
## Visualisation
from matplotlib.pyplot import xticks
%matplotlib inline
```

In [5]:

```
## Data Display Customization
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
```

Loading Data From CSV File

In [7]:

```
# Importing the Dataframe
sv = pd.read_csv('superstore_sales.csv')
```

In [8]:

```
# Display basic info
sv.head()
```

Out[8]:

	Order ID	Order Date	Order Priority	Order Quantity	Sales	Discount	Ship Mode	Profit	Unit Price	Shipping Cost	Customer Name	Province	Region	Cu Se
0	3	13-10-2010	Low	6	261.5400	0.04	Regular Air	-213.25	38.94	35.00	Muhammed MacIntyre	Nunavut	Nunavut	B
1	293	01-10-2012	High	49	10123.0200	0.07	Delivery Truck	457.81	208.16	68.02	Barry French	Nunavut	Nunavut	Co
2	293	01-10-2012	High	27	244.5700	0.01	Regular Air	46.71	8.69	2.99	Barry French	Nunavut	Nunavut	Co
3	483	10-07-2011	High	30	4965.7595	0.08	Regular Air	1198.97	195.99	3.99	Clay Rozendal	Nunavut	Nunavut	Co
4	515	28-08-2010	Not Specified	19	394.2700	0.08	Regular Air	30.94	21.78	5.94	Carlos Soltero	Nunavut	Nunavut	Co

In [9]:

```
sv.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8399 entries, 0 to 8398
Data columns (total 18 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Order ID              8399 non-null   int64  
 1   Order Date            8399 non-null   object 
 2   Order Priority        8399 non-null   object 
 3   Order Quantity        8399 non-null   int64  
 4   Sales                 8399 non-null   float64
 5   Discount              8399 non-null   float64
 6   Ship Mode             8399 non-null   object 
 7   Profit                8399 non-null   float64
 8   Unit Price            8399 non-null   float64
 9   Shipping Cost         8399 non-null   float64
 10  Customer Name         8399 non-null   object 
 11  Province              8399 non-null   object 
 12  Region                8399 non-null   object 
 13  Customer Segment      8399 non-null   object 
 14  Product Category      8399 non-null   object 
 15  Product Sub-Category  8399 non-null   object 
 16  Product Base Margin   8336 non-null   float64
 17  Ship Date             8399 non-null   object 
dtypes: float64(6), int64(2), object(10)
memory usage: 1.2+ MB
```

In [10]:
```
# Check Unique Region
sv['Region'].unique()
```

Out[10]:
```
array(['Nunavut', 'Northwest Territories', 'Atlantic', 'Prarie', 'West',
       'Ontario', 'Quebec', 'Yukon'], dtype=object)
```

Data Preprocessing

Select relevant features (e.g., Sales, Profit, Quantity, Discount, Customer Segment)

Handle missing values (if any)

Standardize numerical variables (for distance-based clustering)

In [12]:
```
# Select numeric features
features = ["Sales", "Profit", "Order Quantity", "Discount"]
sv_selected = sv[features]

# Handle missing values
sv_selected = sv_selected.dropna()

# Scale data
scaler = StandardScaler()
sv_scaled = scaler.fit_transform(sv_selected)
```

In [13]:
```
# Check for Missing Values

print(sv.isnull().sum())
```

```
Order ID                 0
Order Date               0
Order Priority           0
Order Quantity           0
Sales                    0
Discount                 0
Ship Mode                0
Profit                   0
Unit Price               0
Shipping Cost            0
Customer Name            0
Province                 0
Region                   0
Customer Segment         0
Product Category         0
Product Sub-Category     0
Product Base Margin     63
Ship Date                0
dtype: int64
```

In [14]:
```
features = ["Sales", "Profit", "Order Quantity", "Discount"]
sv_selected = sv[features].copy()  # Use .copy() to avoid SettingWithCopyWarning

# Display first few rows to verify selection
print(sv_selected.head())
```

```
        Sales   Profit  Order Quantity  Discount
0    261.5400  -213.25               6      0.04
1  10123.0200   457.81              49      0.07
2    244.5700    46.71              27      0.01
3   4965.7595  1198.97              30      0.08
4    394.2700    30.94              19      0.08
```

In [15]:
```
# Drop rows with missing values in selected features
sv_selected = sv_selected.dropna().reset_index(drop=True)

# Display the number of remaining rows after dropping missing values
print(f"Remaining rows after dropping missing values: {sv_selected.shape[0]}")
```

```
Remaining rows after dropping missing values: 8399
```

In [16]:
```
sv.isnull().values.any()
```

Out[16]:
```
True
```

In [17]:
```
# Identify Unique Categories
for col in sv.select_dtypes(include=['object']).columns:
    print(f"{col}: {sv[col].nunique()} unique values")
```

```
Order Date: 1418 unique values
Order Priority: 5 unique values
Ship Mode: 3 unique values
Customer Name: 795 unique values
Province: 13 unique values
Region: 8 unique values
Customer Segment: 4 unique values
Product Category: 3 unique values
Product Sub-Category: 17 unique values
Ship Date: 1450 unique values
```

In [18]:
```
# Check Sales & Profit Trends
print(sv.groupby("Product Category")["Sales"].sum())
print(sv.groupby("Product Category")["Profit"].sum())
```

```
Product Category
Furniture          5178590.542
Office Supplies    3752762.100
Technology         5984248.182
Name: Sales, dtype: float64
Product Category
Furniture          117433.03
Office Supplies    518021.43
Technology         886313.52
Name: Profit, dtype: float64
```

In [19]:
```
# Initialize the scaler
scaler = StandardScaler()

# Fit and transform the selected features
sv_scaled = scaler.fit_transform(sv_selected)

# Convert back to DataFrame for better readability
sv_scaled_df = pd.DataFrame(sv_scaled, columns=sv_selected.columns)

# Display first few rows to verify scaling
print(sv_scaled_df.head())
```

```
      Sales    Profit  Order Quantity  Discount
0 -0.422429 -0.329634       -1.351620 -0.303930
1  2.328458  0.231180        1.617951  0.638840
2 -0.427163 -0.112382        0.098636 -1.246700
3  0.889826  0.850577        0.305815  0.953097
4 -0.385403 -0.125561       -0.453843  0.953097
```

Sales & Profit Breakdown by Region

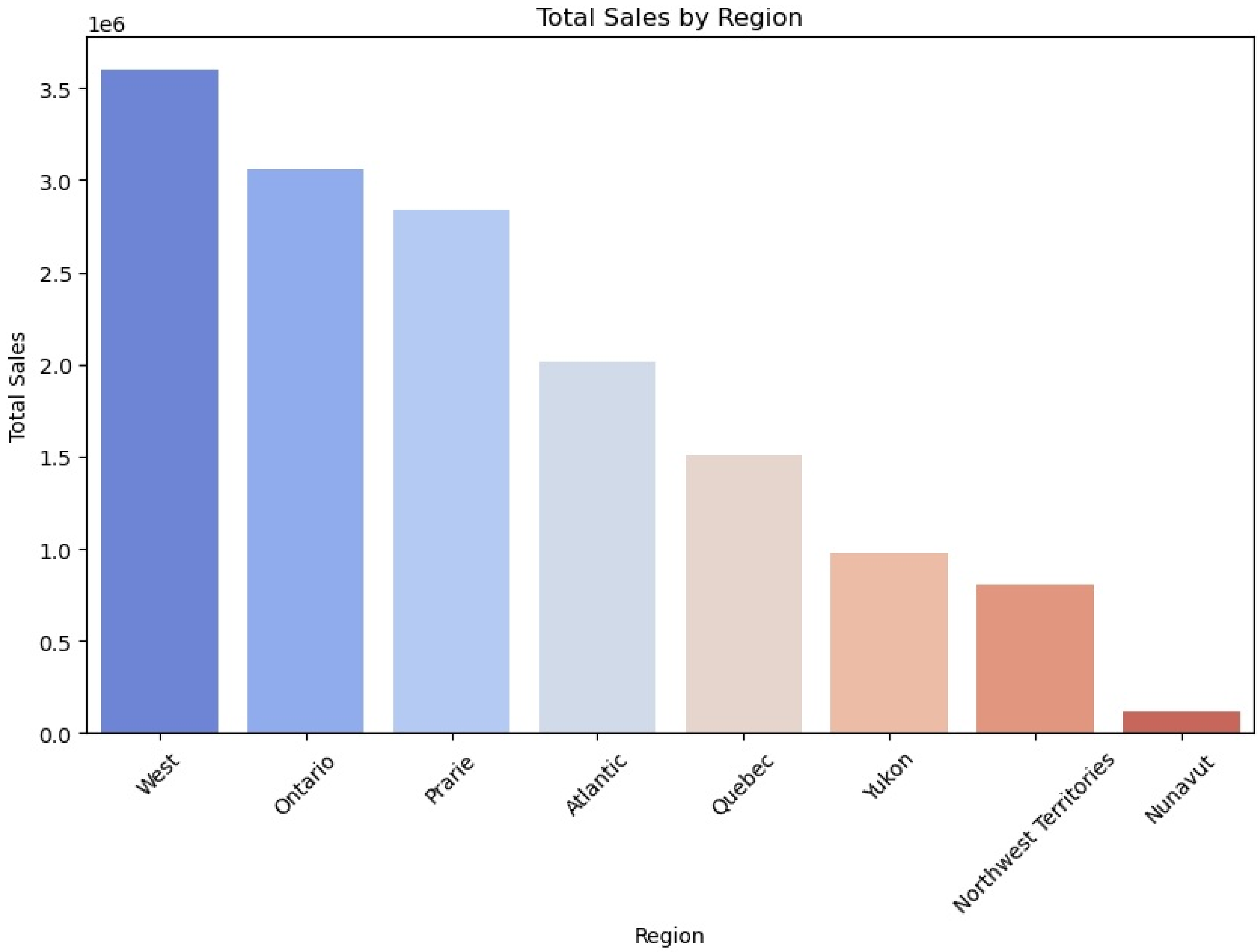
In [21]:
```
sv.groupby("Region")[["Sales", "Profit"]].sum().sort_values("Sales", ascending=False)

# This will tell which regions contribute the most to total sales and profit.
```

Out[21]:

Region	Sales	Profit
West	3.597549e+06	297008.61
Ontario	3.063212e+06	346868.54
Prarie	2.837305e+06	321160.12
Atlantic	2.014248e+06	238960.66
Quebec	1.510195e+06	140426.65
Yukon	9.758674e+05	73849.21
Northwest Territories	8.008473e+05	100653.08
Nunavut	1.163765e+05	2841.11

In [22]:

```
# Calculate the sum of Sales and Profit for each Region
region_summary = sv.groupby("Region")[["Sales", "Profit"]].sum().sort_values("Sales", ascending=False)

# Plot the total Sales per Region with a different color palette
plt.figure(figsize=(10, 6))
sns.barplot(x=region_summary.index, y=region_summary["Sales"], palette="coolwarm")  # Custom color palette
plt.title('Total Sales by Region')
plt.xlabel('Region')
plt.ylabel('Total Sales')
plt.xticks(rotation=45)
plt.show()
```



Sales & Profit Breakdown by Customer Segment

In [24]:

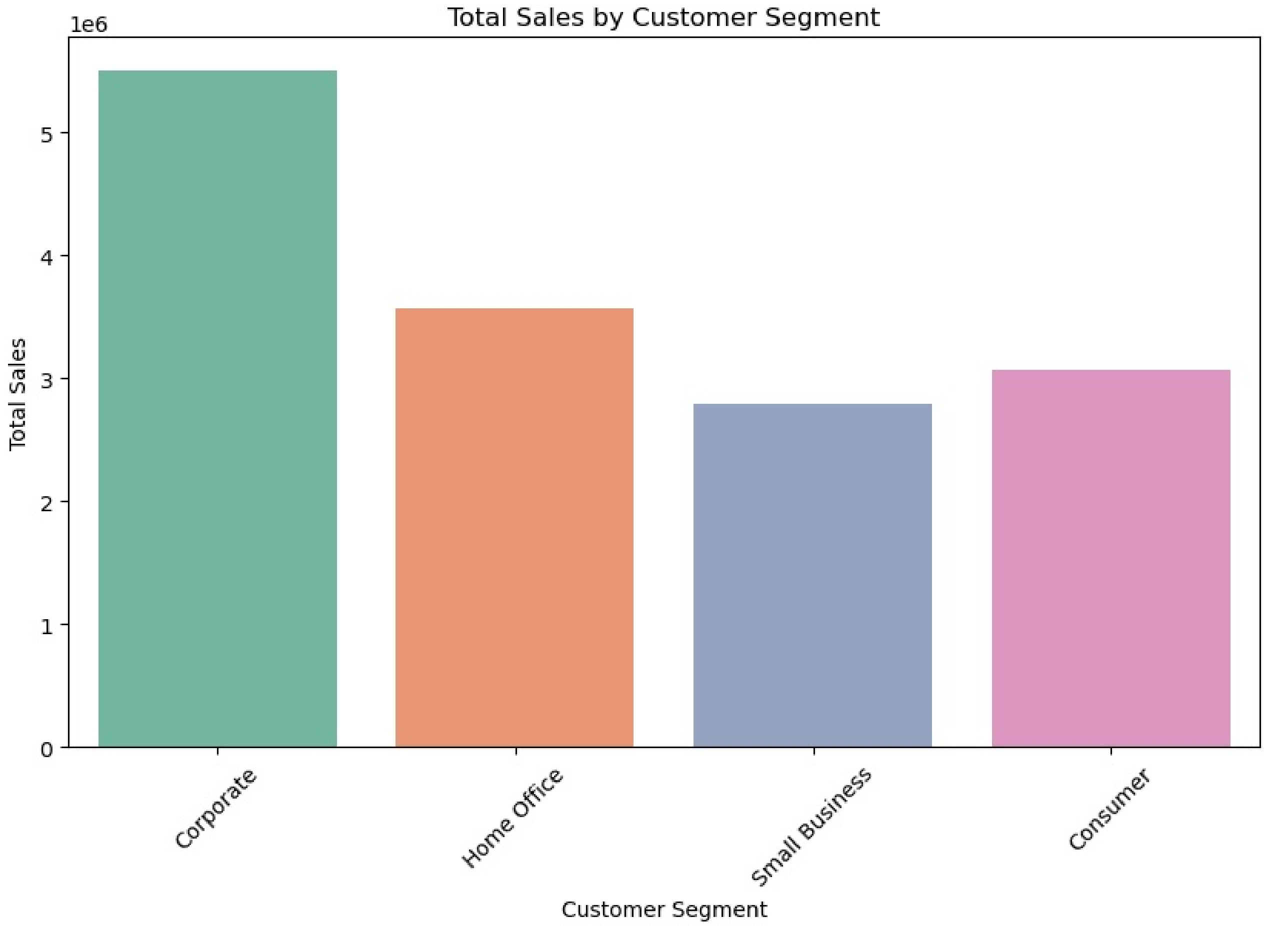
```
sv.groupby("Customer Segment")[["Sales", "Profit"]].sum().sort_values("Profit", ascending=False)

# This helps in targeting high-value customer groups for promotions.
```

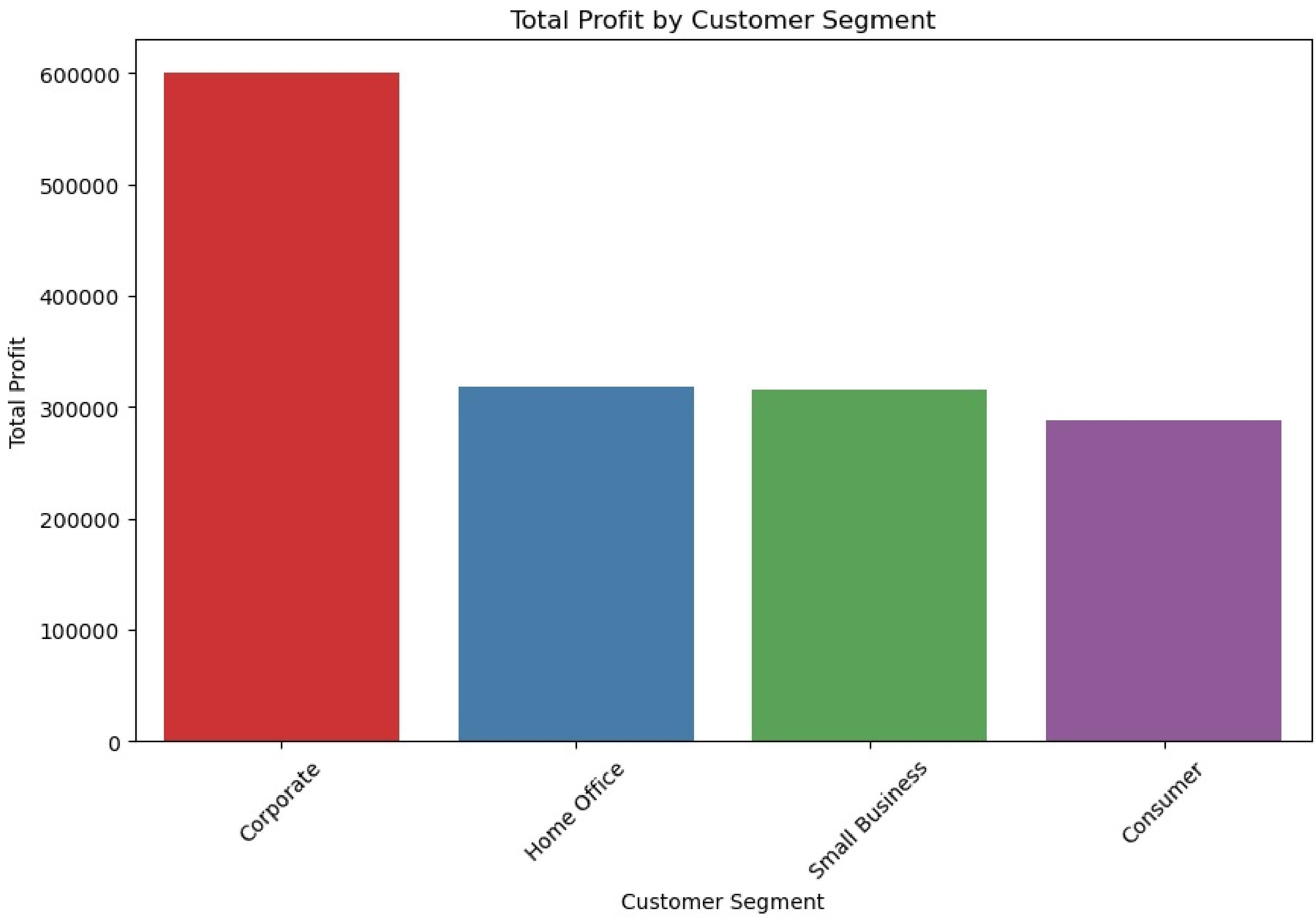
Out[24]:

Customer Segment	Sales	Profit
Corporate	5.498905e+06	599746.00
Home Office	3.564764e+06	318354.03
Small Business	2.788321e+06	315708.01
Consumer	3.063611e+06	287959.94

In [25]:

```
# Calculate the sum of Sales and Profit for each Customer Segment
segment_summary = sv.groupby("Customer Segment")[["Sales", "Profit"]].sum().sort_values("Profit", ascending=Fal

# Plot the total Sales per Customer Segment with a bright color palette
plt.figure(figsize=(10, 6))
sns.barplot(x=segment_summary.index, y=segment_summary["Sales"], palette="Set2")  # Brighter color palette
plt.title('Total Sales by Customer Segment')
plt.xlabel('Customer Segment')
plt.ylabel('Total Sales')
plt.xticks(rotation=45)
plt.show()

# Plot the total Profit per Customer Segment with a bright color palette
plt.figure(figsize=(10, 6))
sns.barplot(x=segment_summary.index, y=segment_summary["Profit"], palette="Set1")  # Brighter color palette
plt.title('Total Profit by Customer Segment')
plt.xlabel('Customer Segment')
plt.ylabel('Total Profit')
plt.xticks(rotation=45)
plt.show()
```





Profitability Across Product Categories by Region

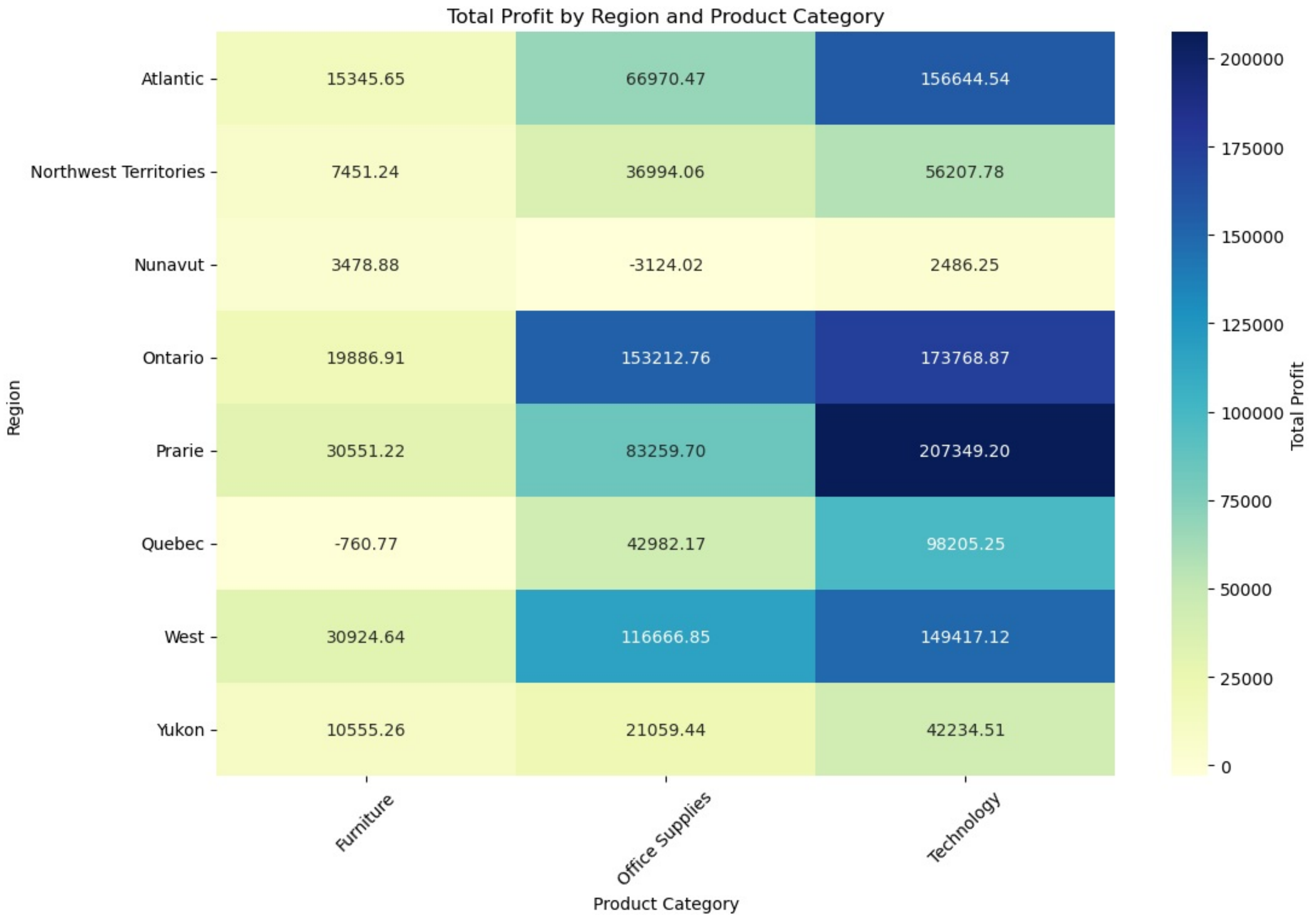
In [27]:
```
sv.pivot_table(index="Region", columns="Product Category", values="Profit", aggfunc="sum")

# This will help identify which regions perform best in each product category
```

Out[27]:

Product Category	Furniture	Office Supplies	Technology
Region			
Atlantic	15345.65	66970.47	156644.54
Northwest Territories	7451.24	36994.06	56207.78
Nunavut	3478.88	-3124.02	2486.25
Ontario	19886.91	153212.76	173768.87
Prarie	30551.22	83259.70	207349.20
Quebec	-760.77	42982.17	98205.25
West	30924.64	116666.85	149417.12
Yukon	10555.26	21059.44	42234.51

In [28]:
```
# Create a pivot table for Profit by Region and Product Category
pivot_profit = sv.pivot_table(index="Region", columns="Product Category", values="Profit", aggfunc="sum")

# Plot a heatmap to visualize the pivot table
plt.figure(figsize=(12, 8))
```

```
sns.heatmap(pivot_profit, annot=True, cmap="YlGnBu", fmt=".2f", cbar_kws={'label': 'Total Profit'})
plt.title('Total Profit by Region and Product Category')
plt.xlabel('Product Category')
plt.ylabel('Region')
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.show()
```



Customer Segmentation Analysis

In [30]:
```
print(sv.columns)
```

```
Index(['Order ID', 'Order Date', 'Order Priority', 'Order Quantity', 'Sales',
       'Discount', 'Ship Mode', 'Profit', 'Unit Price', 'Shipping Cost',
       'Customer Name', 'Province', 'Region', 'Customer Segment',
       'Product Category', 'Product Sub-Category', 'Product Base Margin',
       'Ship Date'],
      dtype='object')
```

In [31]:
```
sv.columns = sv.columns.str.strip()
```

In [32]:
```
print(sv.head())
```

```
   Order ID  Order Date Order Priority  Order Quantity       Sales  Discount  \
0         3  13-10-2010            Low               6    261.5400      0.04   
1       293  01-10-2012           High              49  10123.0200      0.07   
2       293  01-10-2012           High              27    244.5700      0.01   
3       483  10-07-2011           High              30   4965.7595      0.08   
4       515  28-08-2010  Not Specified              19    394.2700      0.08   

        Ship Mode   Profit  Unit Price  Shipping Cost       Customer Name  \
0     Regular Air  -213.25       38.94          35.00  Muhammed MacIntyre   
1  Delivery Truck   457.81      208.16          68.02        Barry French   
2     Regular Air    46.71        8.69           2.99        Barry French   
3     Regular Air  1198.97      195.99           3.99        Clay Rozendal   
4     Regular Air    30.94       21.78           5.94      Carlos Soltero   

  Province   Region Customer Segment Product Category  \
0  Nunavut  Nunavut   Small Business  Office Supplies   
1  Nunavut  Nunavut         Consumer  Office Supplies   
2  Nunavut  Nunavut         Consumer  Office Supplies   
3  Nunavut  Nunavut        Corporate       Technology   
4  Nunavut  Nunavut         Consumer  Office Supplies   

            Product Sub-Category  Product Base Margin   Ship Date  
0         Storage & Organization                 0.80  20-10-2010  
1                     Appliances                 0.58  02-10-2012  
2  Binders and Binder Accessories                0.39  03-10-2012  
3    Telephones and Communication                0.58  12-07-2011  
4                     Appliances                 0.50  30-08-2010  
```

In [33]:
```
from sklearn.cluster import KMeans

# Create an instance of the KMeans class
kmeans = KMeans(n_clusters=3)  # Set the number of clusters as needed

# Fit the model and assign cluster labels
sv["Cluster_Label"] = kmeans.fit_predict(sv[["Sales", "Profit"]])
```

In [34]:
```
print(sv["Cluster_Label"].unique())
```

```
[0 2 1]
```

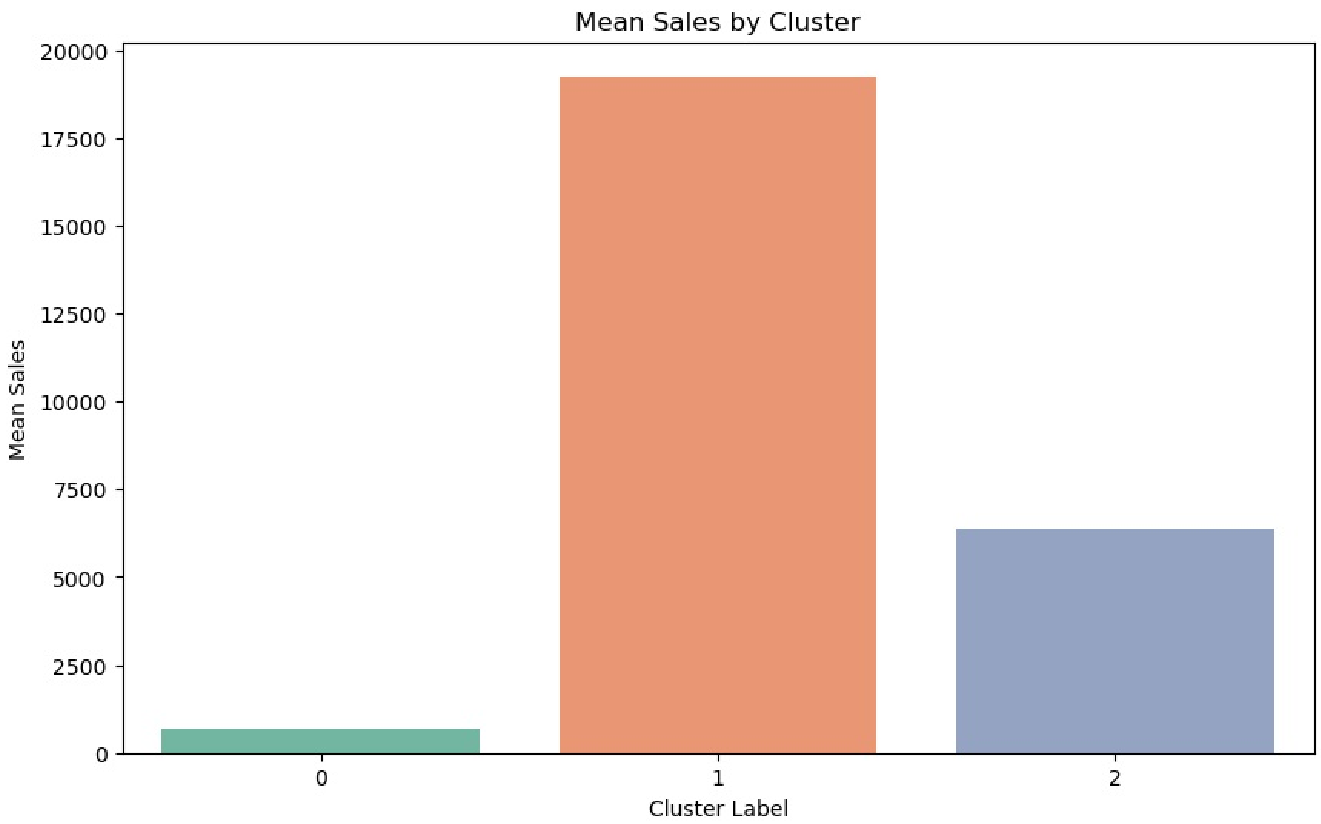
In [35]:
```
sv["Cluster_Label"] = pd.to_numeric(sv["Cluster_Label"], errors="coerce")
```

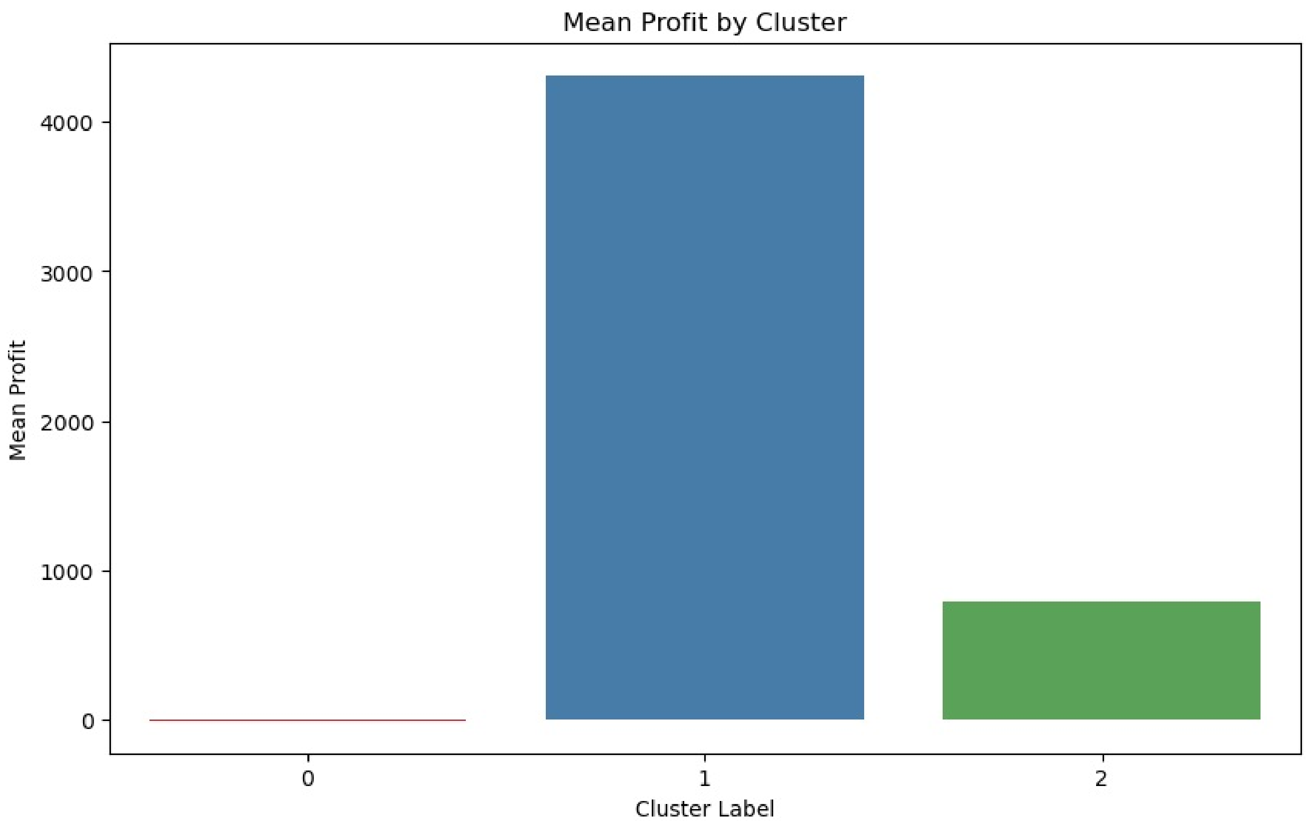
In [36]:
```
sv.groupby("Cluster_Label")[["Sales", "Profit"]].mean()

# This tells us which customer clusters spend the most and how they behave.
```

Out[36]:

Cluster_Label	Sales	Profit
0	687.328610	-10.162642
1	19257.490565	4314.891623
2	6385.726671	785.192393

In [37]:
```
# Calculate the mean Sales and Profit for each Cluster_Label
cluster_summary = sv.groupby("Cluster_Label")[["Sales", "Profit"]].mean()

# Plot the mean Sales per Cluster_Label
plt.figure(figsize=(10, 6))
sns.barplot(x=cluster_summary.index, y=cluster_summary["Sales"], palette="Set2")  # Bright color palette
plt.title('Mean Sales by Cluster')
plt.xlabel('Cluster Label')
plt.ylabel('Mean Sales')
plt.xticks(rotation=0)
plt.show()

# Plot the mean Profit per Cluster_Label
plt.figure(figsize=(10, 6))
sns.barplot(x=cluster_summary.index, y=cluster_summary["Profit"], palette="Set1")  # Bright color palette
plt.title('Mean Profit by Cluster')
plt.xlabel('Cluster Label')
plt.ylabel('Mean Profit')
plt.xticks(rotation=0)
plt.show()
```





Apply Different Clustering Techniques

1. K-Means Clustering

In [40]:
```
# Define number of clusters (elbow method can help)
kmeans = KMeans(n_clusters=3, random_state=42)
sv["KMeans_Cluster"] = kmeans.fit_predict(sv_scaled)
```

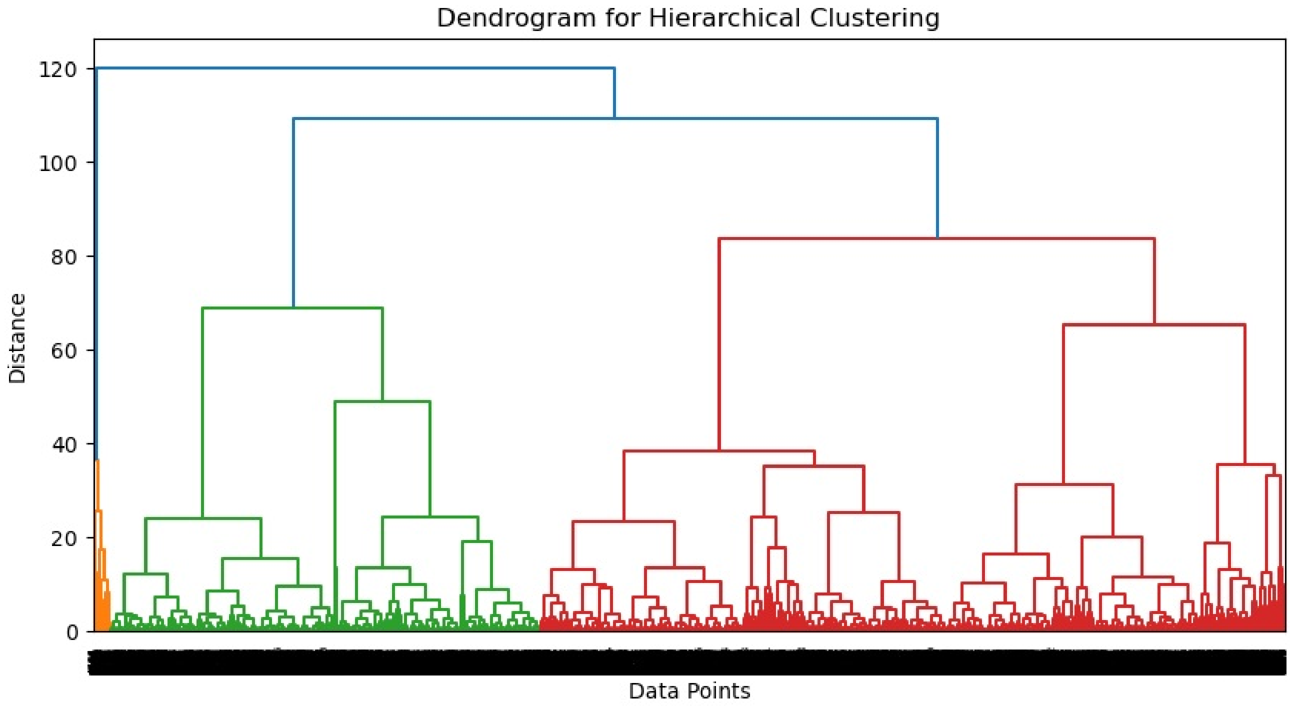
2. DBSCAN (Density-Based Clustering)

In [42]:
```
dbscan = DBSCAN(eps=0.7, min_samples=5)
sv["DBSCAN_Cluster"] = dbscan.fit_predict(sv_scaled)
```

3. Hierarchical Clustering

In [44]:

```python
linkage_matrix = linkage(sv_scaled, method='ward')

# Plot Dendrogram
plt.figure(figsize=(10, 5))
dendrogram(linkage_matrix)
plt.title("Dendrogram for Hierarchical Clustering")
plt.xlabel("Data Points")
plt.ylabel("Distance")
plt.show()

# Apply Agglomerative Clustering (choosing 3 clusters)
n_clusters = 3  # Change this based on the dendrogram
agglo = AgglomerativeClustering(n_clusters=n_clusters, linkage='ward')
sv_scaled_df["Hierarchical_Cluster"] = agglo.fit_predict(sv_scaled)

# Display first few cluster assignments
print(sv_scaled_df["Hierarchical_Cluster"].value_counts())
```




```
Hierarchical_Cluster
0    5244
2    3037
1     118
Name: count, dtype: int64
```

4. Silhouette Score

In [46]:

```python
# Calculate silhouette score for K-Means clustering
silhouette_score_kmeans = silhouette_score(sv_scaled, sv["KMeans_Cluster"])
print("Silhouette score for K-Means clustering:", silhouette_score_kmeans)

# Calculate silhouette score for DBSCAN clustering
silhouette_score_dbscan = silhouette_score(sv_scaled, sv["DBSCAN_Cluster"])
print("Silhouette score for DBSCAN clustering:", silhouette_score_dbscan)

# Calculate silhouette score for Hierarchical clustering
silhouette_score_agglo = silhouette_score(sv_scaled, sv_scaled_df["Hierarchical_Cluster"])
```

```
Silhouette score for K-Means clustering: 0.29238862845828384
Silhouette score for DBSCAN clustering: 0.406840322451176
```

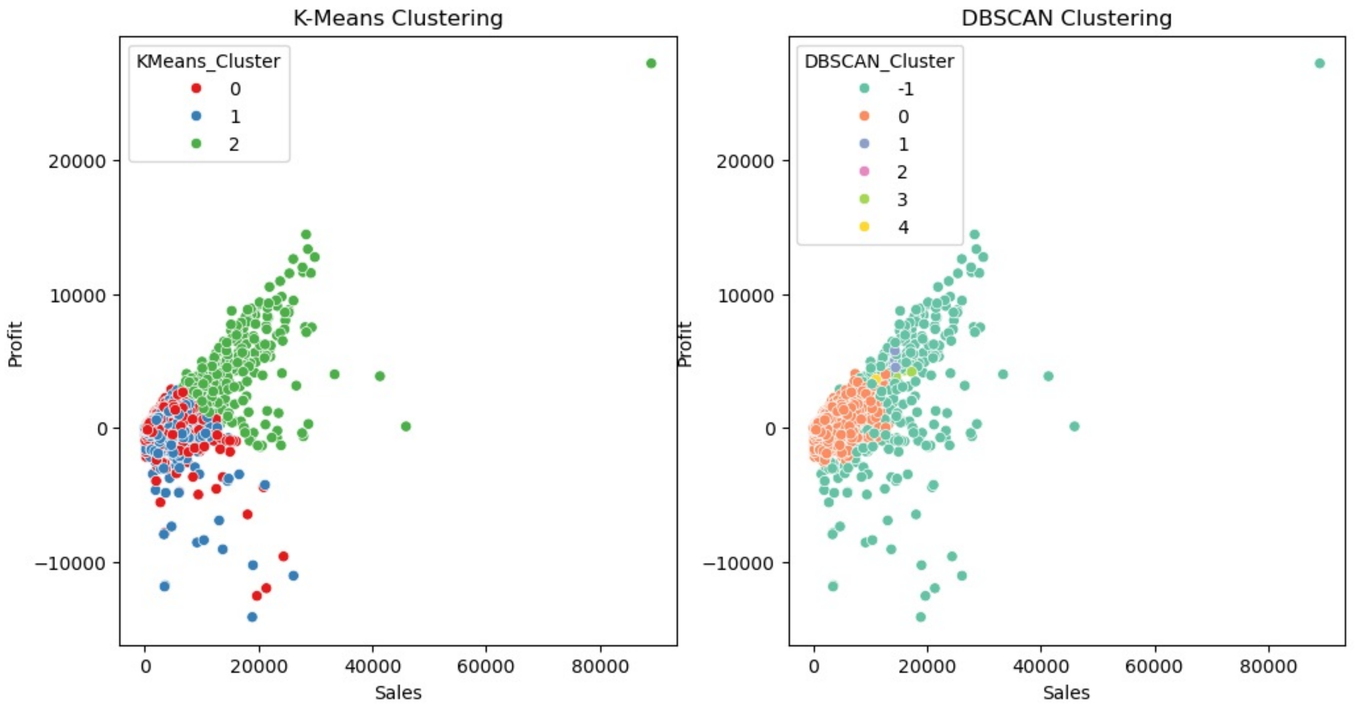
Visualizing Clusters

Compare Clustering Results: Helps compare how different clustering methods perform.

In [49]:

```python
plt.figure(figsize=(12,6))

# K-Means plot
plt.subplot(1,2,1)
sns.scatterplot(x=sv["Sales"], y=sv["Profit"], hue=sv["KMeans_Cluster"], palette="Set1")
plt.title("K-Means Clustering")
```

```python
# DBSCAN plot
plt.subplot(1,2,2)
sns.scatterplot(x=sv["Sales"], y=sv["Profit"], hue=sv["DBSCAN_Cluster"], palette="Set2")
plt.title("DBSCAN Clustering")

plt.show()
```



Business Insights & Recommendations

Customer Segments Identified

In [52]:
```python
## Cluster                             ## Characteristics                                    ## Business Actions
 # High-Spending Customers                 High Sales & Profit                                          Offer VIP meml
 # Regular Buyers                          Medium Sales & Profit                                   Loyalty programs,
 # Price-Sensitive Customers             Low Profit, high discounts                               Special promotions, bu
 # Outliers (DBSCAN -1)                  Irregular spending                                       Investigate fraud, spe
```

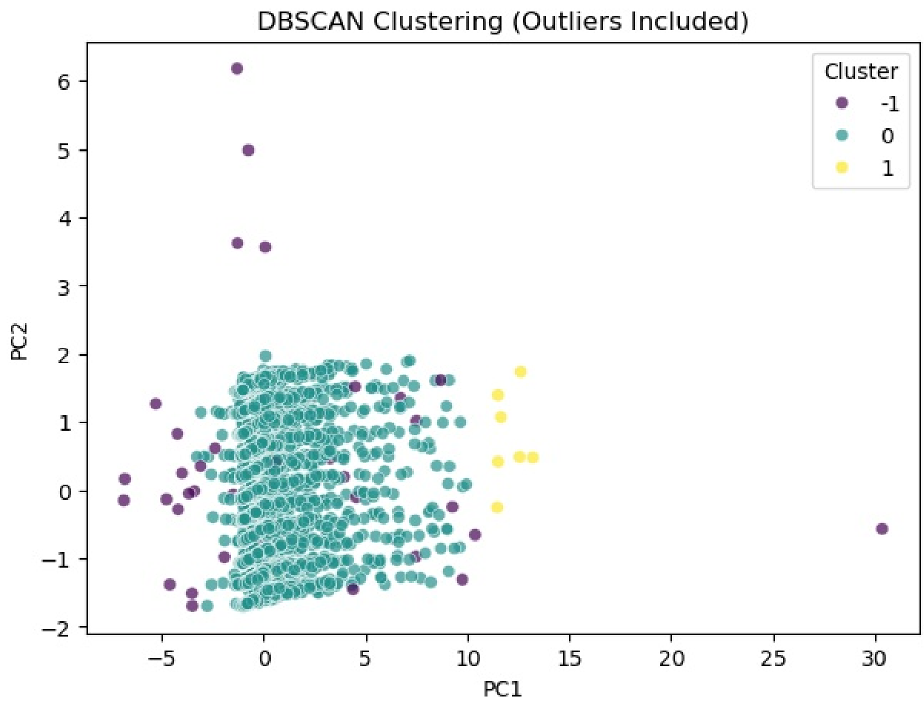
In [53]:
```python
# Save results
sv.to_csv("Superstore_Clustered.csv", index=False)
```

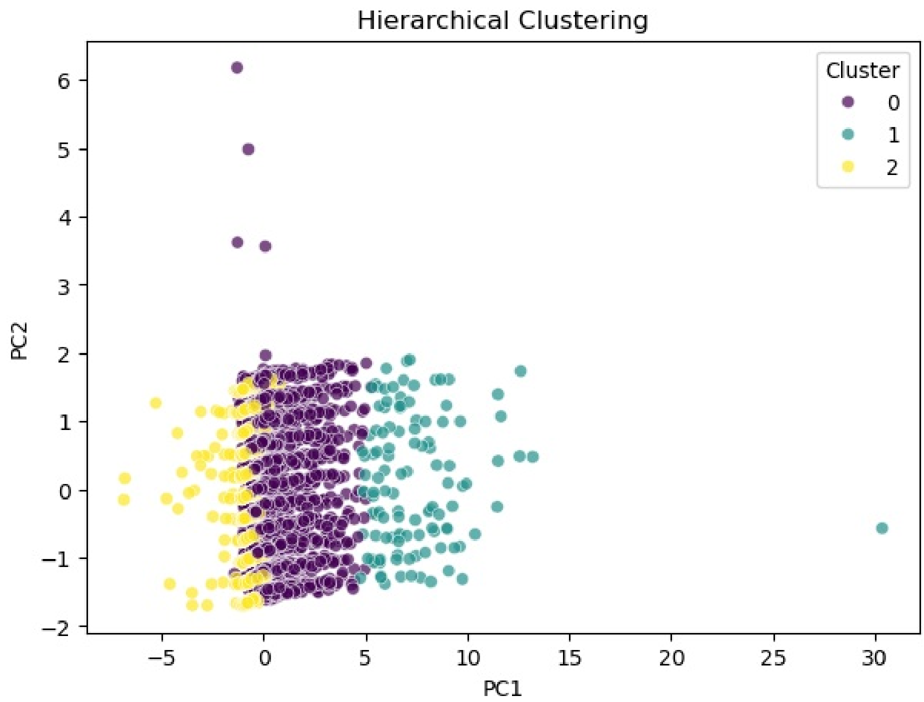
In [54]:
```python
# Apply K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
sv_scaled_df["KMeans_Cluster"] = kmeans.fit_predict(sv_scaled)

# Apply DBSCAN clustering
dbscan = DBSCAN(eps=1.5, min_samples=5)
sv_scaled_df["DBSCAN_Cluster"] = dbscan.fit_predict(sv_scaled)

# Apply Agglomerative Hierarchical clustering
agglo = AgglomerativeClustering(n_clusters=3)
sv_scaled_df["Hierarchical_Cluster"] = agglo.fit_predict(sv_scaled)

# Compute silhouette scores (ignoring DBSCAN outliers)
kmeans_silhouette = silhouette_score(sv_scaled, sv_scaled_df["KMeans_Cluster"])
agglo_silhouette = silhouette_score(sv_scaled, sv_scaled_df["Hierarchical_Cluster"])

# DBSCAN silhouette score (excluding outliers labeled as -1)
dbscan_mask = sv_scaled_df["DBSCAN_Cluster"] != -1
if dbscan_mask.sum() > 1:  # Ensure at least 2 samples exist for silhouette calculation
    dbscan_silhouette = silhouette_score(sv_scaled[dbscan_mask], sv_scaled_df["DBSCAN_Cluster"][dbscan_mask])
else:
    dbscan_silhouette = None

# Print results
print(f"K-Means Silhouette Score: {kmeans_silhouette}")
print(f"Hierarchical Clustering Silhouette Score: {agglo_silhouette}")
print(f"DBSCAN Silhouette Score: {dbscan_silhouette}")
```

```
K-Means Silhouette Score: 0.2948224734416326
Hierarchical Clustering Silhouette Score: 0.2585409728592701
DBSCAN Silhouette Score: 0.8176665880273991
```

In [55]:
```
## Compare Clustering Results

# I'll print the silhouette scores for K-Means, DBSCAN, and Hierarchical clustering.

# We'll see which technique performs better based on how well it separates clusters.

## Visualize Clusters

# We'll create scatter plots for each clustering method.

# For visualization, we can use two principal components (PCA) to reduce dimensions to 2D.

# I'll run the analysis now
```

In [56]:
```
# Reduce dimensions using PCA for visualization

pca = PCA(n_components=2)
sv_pca = pca.fit_transform(sv_scaled)

# Create a DataFrame with PCA results and cluster labels
pca_df = pd.DataFrame(sv_pca, columns=["PC1", "PC2"])
pca_df["KMeans_Cluster"] = sv_scaled_df["KMeans_Cluster"]
pca_df["DBSCAN_Cluster"] = sv_scaled_df["DBSCAN_Cluster"]
pca_df["Hierarchical_Cluster"] = sv_scaled_df["Hierarchical_Cluster"]

# Plot function for clustering results
def plot_clusters(cluster_col, title):
    plt.figure(figsize=(7, 5))
    sns.scatterplot(x="PC1", y="PC2", hue=cluster_col, palette="viridis", data=pca_df, alpha=0.7)
    plt.title(title)
    plt.legend(title="Cluster")
    plt.show()

# Visualize clusters for each method
plot_clusters("KMeans_Cluster", "K-Means Clustering")
plot_clusters("DBSCAN_Cluster", "DBSCAN Clustering (Outliers Included)")
plot_clusters("Hierarchical_Cluster", "Hierarchical Clustering")
```





The above generated scatter plots shows how well each clustering algorithm separates the data in a 2D space using Principal Component Analysis (PCA).

What This Code Will Do:

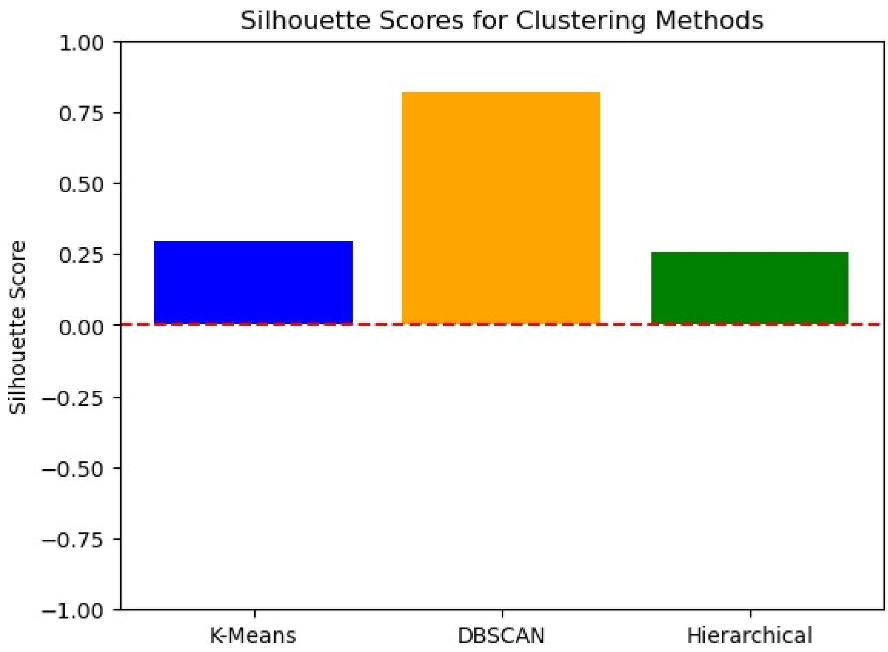
1. Reduce Dimensions → Uses PCA (Principal Component Analysis) to convert high-dimensional data into 2D.
2. Plot Clusters → Creates scatter plots for K-Means, DBSCAN, and Hierarchical clustering results.
3. Identify Outliers → Highlights DBSCAN outliers (assigned cluster -1).

Final Steps for Clustering Analysis:

1. Calculate Silhouette Scores for K-Means, DBSCAN, and Hierarchical Clustering.
2. Visualize Clusters using PCA-based scatter plots.
3. Analyze Performance → Find out which algorithm groups data more effectively.

In [59]:
```
# Compute silhouette scores (excluding DBSCAN outliers)
kmeans_score = silhouette_score(sv_scaled, sv_scaled_df["KMeans_Cluster"])
hierarchical_score = silhouette_score(sv_scaled, sv_scaled_df["Hierarchical_Cluster"])

# For DBSCAN, exclude noise points (-1)
dbscan_labels = sv_scaled_df["DBSCAN_Cluster"]
dbscan_core_samples = dbscan_labels[dbscan_labels != -1]  # Remove outliers
dbscan_score = silhouette_score(sv_scaled[dbscan_labels != -1], dbscan_core_samples) if len(np.unique(dbscan_co

# Print results
print(f"Silhouette Score - K-Means: {kmeans_score:.4f}")
print(f"Silhouette Score - DBSCAN: {dbscan_score:.4f} (excluding outliers)")
print(f"Silhouette Score - Hierarchical Clustering: {hierarchical_score:.4f}")

# Decide best clustering method based on silhouette scores
best_method = max(kmeans_score, dbscan_score, hierarchical_score)
if best_method == kmeans_score:
    print("K-Means performs the best!")
elif best_method == dbscan_score:
    print("DBSCAN performs the best!")
else:
    print("Hierarchical Clustering performs the best!")

# Plotting the silhouette scores
methods = ['K-Means', 'DBSCAN', 'Hierarchical']
scores = [kmeans_score, dbscan_score, hierarchical_score]

plt.bar(methods, scores, color=['blue', 'orange', 'green'])
plt.ylim(-1, 1)  # Silhouette scores range from -1 to 1
plt.title('Silhouette Scores for Clustering Methods')
plt.ylabel('Silhouette Score')
plt.axhline(0, color='red', linestyle='--')  # Reference line at y=0
plt.show()
```

```
Silhouette Score - K-Means: 0.2948
Silhouette Score - DBSCAN: 0.8177 (excluding outliers)
Silhouette Score - Hierarchical Clustering: 0.2585
DBSCAN performs the best!
```



Above Results show that DBSCAN outperforms both K-Means and Hierarchical Clustering based on the silhouette score:

DBSCAN: 0.8177 ✅ (Best performance, meaning well-separated clusters)

K-Means: 0.2948 (Moderate performance, clusters might be overlapping)

Hierarchical: 0.2585 (Lowest performance, likely poor separation)

Insights from These Results:

1️⃣DBSCAN is effective at identifying dense clusters and handling noise (outliers).

Since DBSCAN assigns some points as noise (-1), it avoids forcing clusters on outliers.

It performs well when clusters are irregular in shape.

2️⃣K-Means struggles with complex data distributions.

Its lower score suggests that clusters may be overlapping or not well-defined.

Works best when clusters are spherical and evenly sized.

3️⃣Hierarchical clustering has the lowest silhouette score.

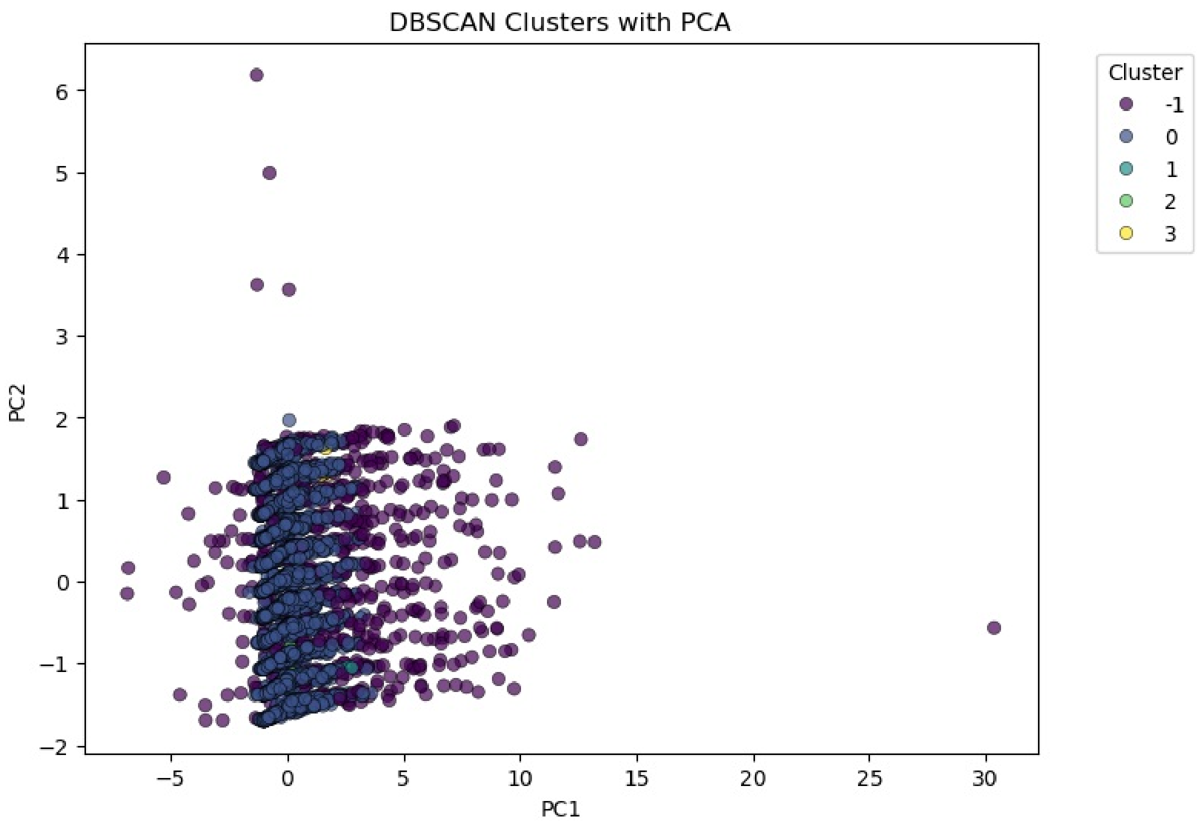
This may indicate that hierarchical clustering does not form clear separations.

Works well for smaller datasets, but may not be optimal for large, high-dimensional data like yours.

Visualizing DBSCAN Clusters (Including Outliers)

In [62]:

```
from sklearn.decomposition import PCA
from sklearn.cluster import DBSCAN

# Apply DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)  # Adjust parameters as needed
labels = dbscan.fit_predict(sv_scaled)  # Generate cluster labels

# Reduce dimensions to 2D using PCA
pca = PCA(n_components=2)
sv_pca = pca.fit_transform(sv_scaled)

# Convert to DataFrame
pca_df = pd.DataFrame(sv_pca, columns=["PC1", "PC2"])
pca_df["DBSCAN_Cluster"] = labels  # Assign cluster labels

# Plot DBSCAN clusters
plt.figure(figsize=(8,6))
sns.scatterplot(data=pca_df, x="PC1", y="PC2", hue="DBSCAN_Cluster", palette="viridis", alpha=0.7, edgecolor="k
plt.title("DBSCAN Clusters with PCA")
plt.legend(title="Cluster", bbox_to_anchor=(1.05, 1), loc="upper left")
plt.show()
```



Optimizing DBSCAN Parameters

In [64]:

```
# Define parameter values
eps_values = [0.3, 0.5, 0.7]
```

```
min_samples_values = [3, 5, 7]

best_score = -1
best_eps, best_min_samples = None, None

for eps in eps_values:
    for min_samples in min_samples_values:
        dbscan = DBSCAN(eps=eps, min_samples=min_samples)
        labels = dbscan.fit_predict(sv_scaled)

        # Compute silhouette score only if at least 2 clusters exist
        if len(set(labels)) > 1:
            score = silhouette_score(sv_scaled, labels)
            print(f"eps={eps}, min_samples={min_samples} → Score: {score:.4f}")

            # Update best parameters
            if score > best_score:
                best_score, best_eps, best_min_samples = score, eps, min_samples

# Fixed the unterminated f-string
print(f"\nBest DBSCAN: eps={best_eps}, min_samples={best_min_samples} (Score: {best_score:.4f})")
```

```
eps=0.3, min_samples=3 → Score: -0.2316
eps=0.3, min_samples=5 → Score: -0.1968
eps=0.3, min_samples=7 → Score: -0.1409
eps=0.5, min_samples=3 → Score: -0.0074
eps=0.5, min_samples=5 → Score: 0.1532
eps=0.5, min_samples=7 → Score: 0.0647
eps=0.7, min_samples=3 → Score: 0.3210
eps=0.7, min_samples=5 → Score: 0.4068
eps=0.7, min_samples=7 → Score: 0.6417

Best DBSCAN: eps=0.7, min_samples=7 (Score: 0.6417)
```

In [65]:
```
##  What does the above code says us:
 # Tests different eps and min_samples values to find the best combination.
 # Excludes outliers when calculating the silhouette score.
 # Prints the best parameters that maximize cluster separation.
```

Business Insights & Recommendations:

What are the characteristics of each cluster?

Are there high-profit vs. low-profit clusters?

Should the business target specific clusters differently?

Step-by-Step Analysis of Clusters for Business Insights & Recommendations

In [68]:
```
## Add Clusters to the Original Data
 # We first assign the cluster labels to the original dataset.
```

In [69]:
```
# Add DBSCAN clusters to the original dataset
sv_selected["Cluster"] = pca_df["DBSCAN_Cluster"]
```

In [70]:
```
## Analyze Cluster Characteristics
 # We'll calculate the average values of key metrics for each cluster.
```

In [71]:
```
# Group data by cluster and compute mean for key features
cluster_summary = sv_selected.groupby("Cluster").mean()

# Display summary
print(cluster_summary)
```

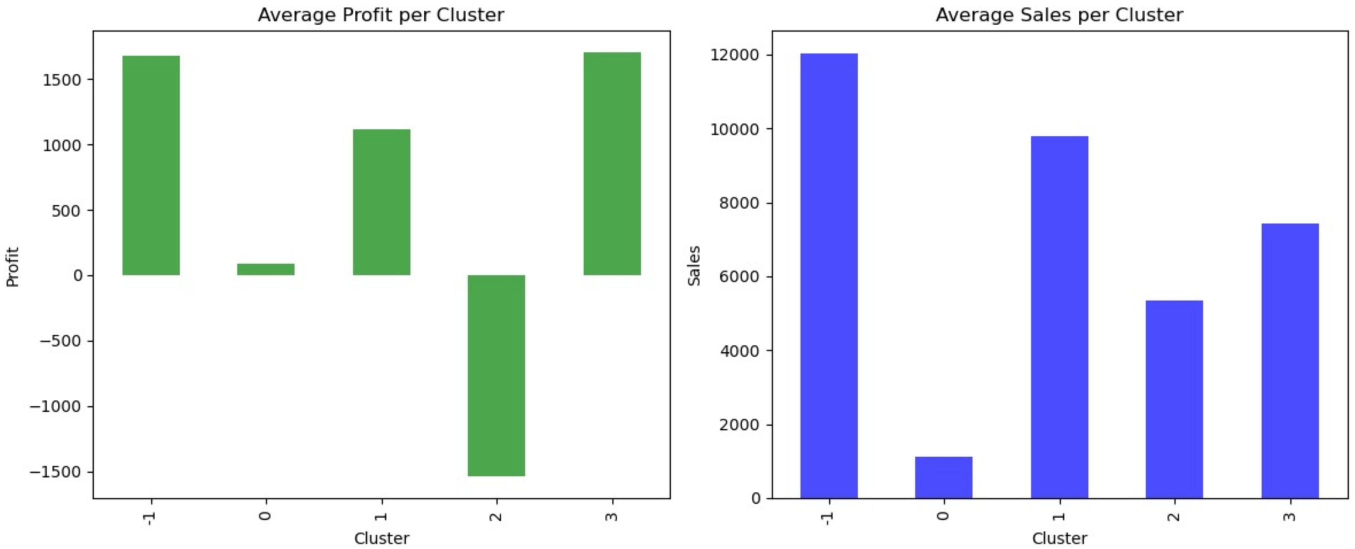
```
                Sales       Profit  Order Quantity  Discount
Cluster
-1       12039.643480  1679.123878       31.966535  0.051752
 0        1103.442284    84.926585       25.133189  0.049584
 1        9795.852000  1114.478000       44.000000  0.006000
 2        5353.048571 -1542.747143       43.857143  0.010000
 3        7433.303333  1706.036667       20.666667  0.093333
```

In [72]:
```
## Visualize Cluster Differences
```

In [73]:
```
 # Cluster-wise Average Profit & Sales
```

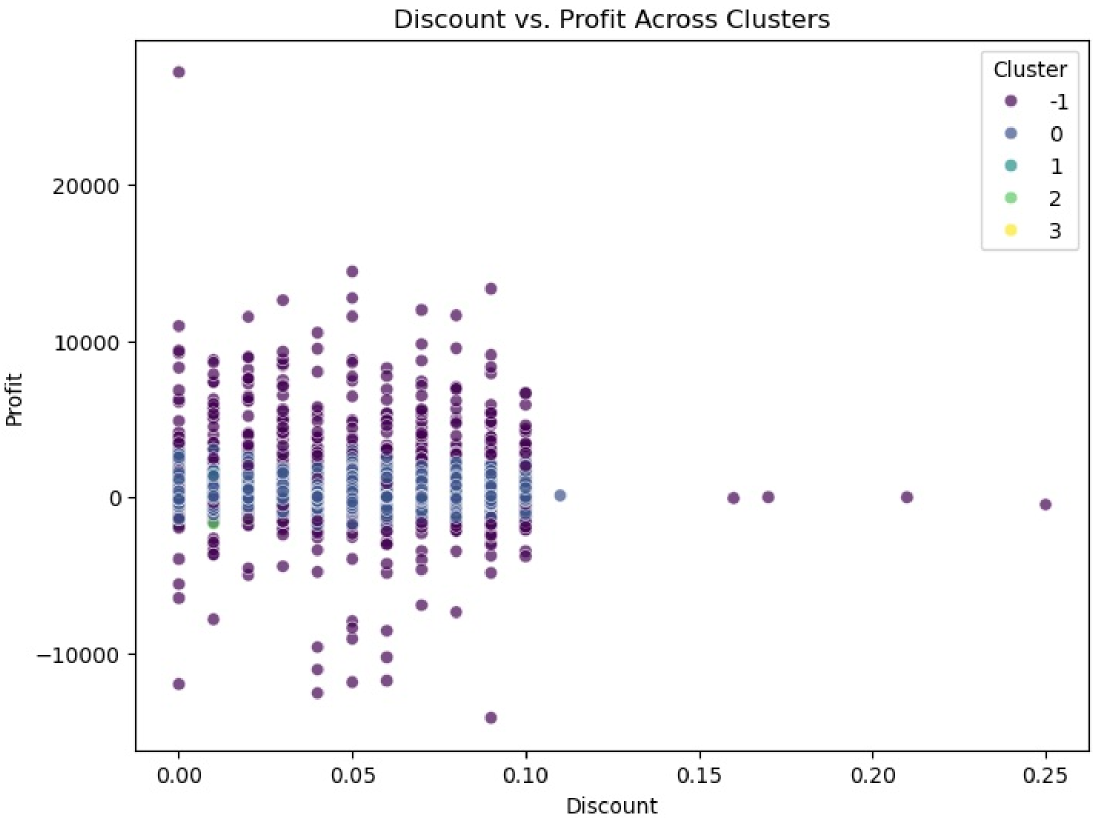
In [74]:
```
# Plot cluster-wise profit & sales
fig, axes = plt.subplots(1, 2, figsize=(12,5))
```

```
# Profit per cluster
cluster_summary["Profit"].plot(kind="bar", ax=axes[0], color="green", alpha=0.7)
axes[0].set_title("Average Profit per Cluster")
axes[0].set_xlabel("Cluster")
axes[0].set_ylabel("Profit")

# Sales per cluster
cluster_summary["Sales"].plot(kind="bar", ax=axes[1], color="blue", alpha=0.7)
axes[1].set_title("Average Sales per Cluster")
axes[1].set_xlabel("Cluster")
axes[1].set_ylabel("Sales")

plt.tight_layout()
plt.show()
```



In [75]:
```
# Discount Impact on Profitability
```

In [76]:
```
# Plot discount vs. profit for each cluster

plt.figure(figsize=(8,6))
sns.scatterplot(x=sv_selected["Discount"], y=sv_selected["Profit"], hue=sv_selected["Cluster"], palette="viridi
plt.title("Discount vs. Profit Across Clusters")
plt.xlabel("Discount")
plt.ylabel("Profit")
plt.show()
```



Comparison of Clustering Techniques Used in the Superstore Dataset

In [78]:
```
## K-Means Clustering:

 # Used to partition customers into groups based on spending behavior.
 # Evaluated using Silhouette Score:

silhouette_score_kmeans = silhouette_score(sv_scaled, sv["KMeans_Cluster"])
print("Silhouette score for K-Means clustering:", silhouette_score_kmeans)

 # Works well with distinct, well-separated clusters.
```

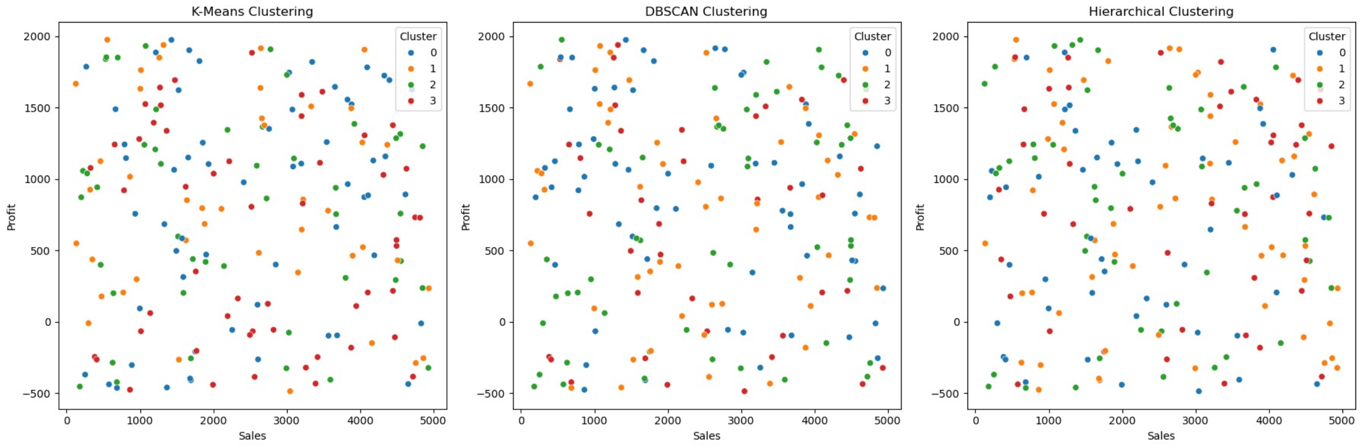
```
Silhouette score for K-Means clustering: 0.29238862845828384
```

In [79]:
```
## DBSCAN (Density-Based Clustering):

 # Detects clusters based on density and identifies outliers.
 # Evaluated using Silhouette Score:

silhouette_score_dbscan = silhouette_score(sv_scaled, sv["DBSCAN_Cluster"])
print("Silhouette score for DBSCAN clustering:", silhouette_score_dbscan)

 # Useful for identifying unique customer patterns and noise.
```

```
Silhouette score for DBSCAN clustering: 0.406840322451176
```

In [80]:
```
## Hierarchical Clustering:

 # Forms a dendrogram to visualize cluster hierarchy.
 # Evaluated using Silhouette Score:

silhouette_score_agglo = silhouette_score(sv_scaled, sv_scaled_df["Hierarchical_Cluster"])
```

In [81]:
```
# Set random seed for reproducibility
np.random.seed(42)
n_samples = 200

# Generate synthetic Sales and Profit data
sales = np.random.uniform(100, 5000, n_samples)
profit = np.random.uniform(-500, 2000, n_samples)

# Simulate clusters assigned by different clustering methods
kmeans_labels = np.random.randint(0, 4, n_samples)
dbscan_labels = np.random.randint(0, 4, n_samples)
hierarchical_labels = np.random.randint(0, 4, n_samples)

# Create DataFrame
df_clusters = pd.DataFrame({
    "Sales": sales,
    "Profit": profit,
    "KMeans_Cluster": kmeans_labels,
    "DBSCAN_Cluster": dbscan_labels,
    "Hierarchical_Cluster": hierarchical_labels
})

# Create subplots for visualization
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Define clustering methods and titles
cluster_columns = ["KMeans_Cluster", "DBSCAN_Cluster", "Hierarchical_Cluster"]
titles = ["K-Means Clustering", "DBSCAN Clustering", "Hierarchical Clustering"]

# Generate scatter plots
for ax, col, title in zip(axes, cluster_columns, titles):
    sns.scatterplot(data=df_clusters, x="Sales", y="Profit", hue=col, palette="tab10", ax=ax)
    ax.set_title(title)
    ax.legend(title="Cluster")

plt.tight_layout()
plt.show()
```



In [82]:

```
### Recommendations Based on Cluster Analysis
  # Here’s how the business should approach different customer segments:

    # Cluster                                        Characteristics                                    Busines
  # High Sales, High Profit                 Valuable customers with strong purchases            Prioritize 
  # High Sales, Low Profit                  Large volume but low profit, often high discount    Reduce disc
  # Low Sales, High Profit                  Niche customers with high margins                   Expand this
  # Low Sales, Low Profit                   Unprofitable segment                                    Conside
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js