In [21]:
```
!pip install pandas numpy matplotlib seaborn scikit-learn
```

```
Requirement already satisfied: pandas in /home/sargam/.conda/envs/notebook
s/lib/python3.10/site-packages (2.3.3)
Requirement already satisfied: numpy in /home/sargam/.conda/envs/notebook
s/lib/python3.10/site-packages (2.2.6)
Requirement already satisfied: matplotlib in /home/sargam/.conda/envs/note
books/lib/python3.10/site-packages (3.10.6)
Requirement already satisfied: seaborn in /home/sargam/.conda/envs/noteboo
ks/lib/python3.10/site-packages (0.13.2)
Requirement already satisfied: scikit-learn in /home/sargam/.conda/envs/no
tebooks/lib/python3.10/site-packages (1.7.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /home/sargam/.con
da/envs/notebooks/lib/python3.10/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /home/sargam/.conda/envs/no
tebooks/lib/python3.10/site-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /home/sargam/.conda/envs/
notebooks/lib/python3.10/site-packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /home/sargam/.conda/env
s/notebooks/lib/python3.10/site-packages (from matplotlib) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /home/sargam/.conda/envs/no
tebooks/lib/python3.10/site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /home/sargam/.conda/en
vs/notebooks/lib/python3.10/site-packages (from matplotlib) (4.60.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /home/sargam/.conda/en
vs/notebooks/lib/python3.10/site-packages (from matplotlib) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /home/sargam/.conda/env
s/notebooks/lib/python3.10/site-packages (from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in /home/sargam/.conda/envs/noteb
ooks/lib/python3.10/site-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /home/sargam/.conda/env
s/notebooks/lib/python3.10/site-packages (from matplotlib) (3.2.5)
Requirement already satisfied: scipy>=1.8.0 in /home/sargam/.conda/envs/no
tebooks/lib/python3.10/site-packages (from scikit-learn) (1.15.3)
Requirement already satisfied: joblib>=1.2.0 in /home/sargam/.conda/envs/n
otebooks/lib/python3.10/site-packages (from scikit-learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /home/sargam/.cond
a/envs/notebooks/lib/python3.10/site-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: six>=1.5 in /home/sargam/.conda/envs/notebo
oks/lib/python3.10/site-packages (from python-dateutil>=2.8.2->pandas) (1.
17.0)
```

In [22]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.cluster import KMeans, AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
from collections import Counter
import warnings
warnings.filterwarnings("ignore")
```

In [23]:
```python
df = pd.read_csv("/home/sargam/Downloads/sales_data_sample.csv", encoding
```

In [24]:
```python
print("Data Loaded Successfully.")
print("-" * 30)
```

```
Data Loaded Successfully.
------------------------------
```

In [25]:

```
print("Initial Data Head:")
print(df.head())
print("-" * 30)
print("Initial Data Information (Data Types and Non-Null Counts):")
df.info()
```

```
Initial Data Head:
   ORDERNUMBER  QUANTITYORDERED  PRICEEACH  ORDERLINENUMBER    SALES  \
0        10107               30      95.70                2  2871.00
1        10121               34      81.35                5  2765.90
2        10134               41      94.74                2  3884.34
3        10145               45      83.26                6  3746.70
4        10159               49     100.00               14  5205.27

         ORDERDATE   STATUS  QTR_ID  MONTH_ID  YEAR_ID  ...  \
0   2/24/2003 0:00  Shipped       1         2     2003  ...
1    5/7/2003 0:00  Shipped       2         5     2003  ...
2    7/1/2003 0:00  Shipped       3         7     2003  ...
3   8/25/2003 0:00  Shipped       3         8     2003  ...
4  10/10/2003 0:00  Shipped       4        10     2003  ...

                    ADDRESSLINE1  ADDRESSLINE2           CITY STATE  \
0        897 Long Airport Avenue           NaN            NYC    NY
1             59 rue de l'Abbaye           NaN          Reims   NaN
2  27 rue du Colonel Pierre Avia           NaN          Paris   NaN
3             78934 Hillside Dr.           NaN       Pasadena    CA
4                7734 Strong St.           NaN  San Francisco    CA

  POSTALCODE COUNTRY TERRITORY CONTACTLASTNAME CONTACTFIRSTNAME DEALSIZE
0      10022     USA       NaN              Yu             Kwai    Small
1      51100  France      EMEA        Henriot             Paul    Small
2      75508  France      EMEA        Da Cunha           Daniel   Medium
3      90003     USA       NaN           Young            Julie   Medium
4        NaN     USA       NaN           Brown            Julie   Medium

[5 rows x 25 columns]
------------------------------
Initial Data Information (Data Types and Non-Null Counts):
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2823 entries, 0 to 2822
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ORDERNUMBER       2823 non-null   int64
 1   QUANTITYORDERED   2823 non-null   int64
 2   PRICEEACH         2823 non-null   float64
 3   ORDERLINENUMBER   2823 non-null   int64
 4   SALES             2823 non-null   float64
 5   ORDERDATE         2823 non-null   object
 6   STATUS            2823 non-null   object
 7   QTR_ID            2823 non-null   int64
 8   MONTH_ID          2823 non-null   int64
 9   YEAR_ID           2823 non-null   int64
 10  PRODUCTLINE       2823 non-null   object
 11  MSRP              2823 non-null   int64
 12  PRODUCTCODE       2823 non-null   object
 13  CUSTOMERNAME      2823 non-null   object
 14  PHONE             2823 non-null   object
 15  ADDRESSLINE1      2823 non-null   object
 16  ADDRESSLINE2      302 non-null    object
 17  CITY              2823 non-null   object
 18  STATE             1337 non-null   object
 19  POSTALCODE        2747 non-null   object
 20  COUNTRY           2823 non-null   object
 21  TERRITORY         1749 non-null   object
 22  CONTACTLASTNAME   2823 non-null   object
```

```
 23  CONTACTFIRSTNAME  2823 non-null   object
 24  DEALSIZE          2823 non-null   object
dtypes: float64(2), int64(7), object(16)
memory usage: 551.5+ KB
```

In [26]:

```python
print("\nMissing Value Counts Before Cleaning:")
print(df.isnull().sum())
print("-" * 30)
```

```
Missing Value Counts Before Cleaning:
ORDERNUMBER            0
QUANTITYORDERED        0
PRICEEACH              0
ORDERLINENUMBER        0
SALES                  0
ORDERDATE              0
STATUS                 0
QTR_ID                 0
MONTH_ID               0
YEAR_ID                0
PRODUCTLINE            0
MSRP                   0
PRODUCTCODE            0
CUSTOMERNAME           0
PHONE                  0
ADDRESSLINE1           0
ADDRESSLINE2        2521
CITY                   0
STATE               1486
POSTALCODE            76
COUNTRY                0
TERRITORY           1074
CONTACTLASTNAME        0
CONTACTFIRSTNAME       0
DEALSIZE               0
dtype: int64
------------------------------
```

In [27]:

```python
columns_to_drop = [
    "ORDERNUMBER", "ORDERLINENUMBER", "PHONE", "ADDRESSLINE1", "ADDRESSLI
    "STATE", "POSTALCODE", "TERRITORY", "CONTACTLASTNAME", "CONTACTFIRSTN
    "ORDERDATE", "QTR_ID", "MONTH_ID", "YEAR_ID"
]
df_clean = df.drop(columns=columns_to_drop, errors='ignore')
```

In [28]:

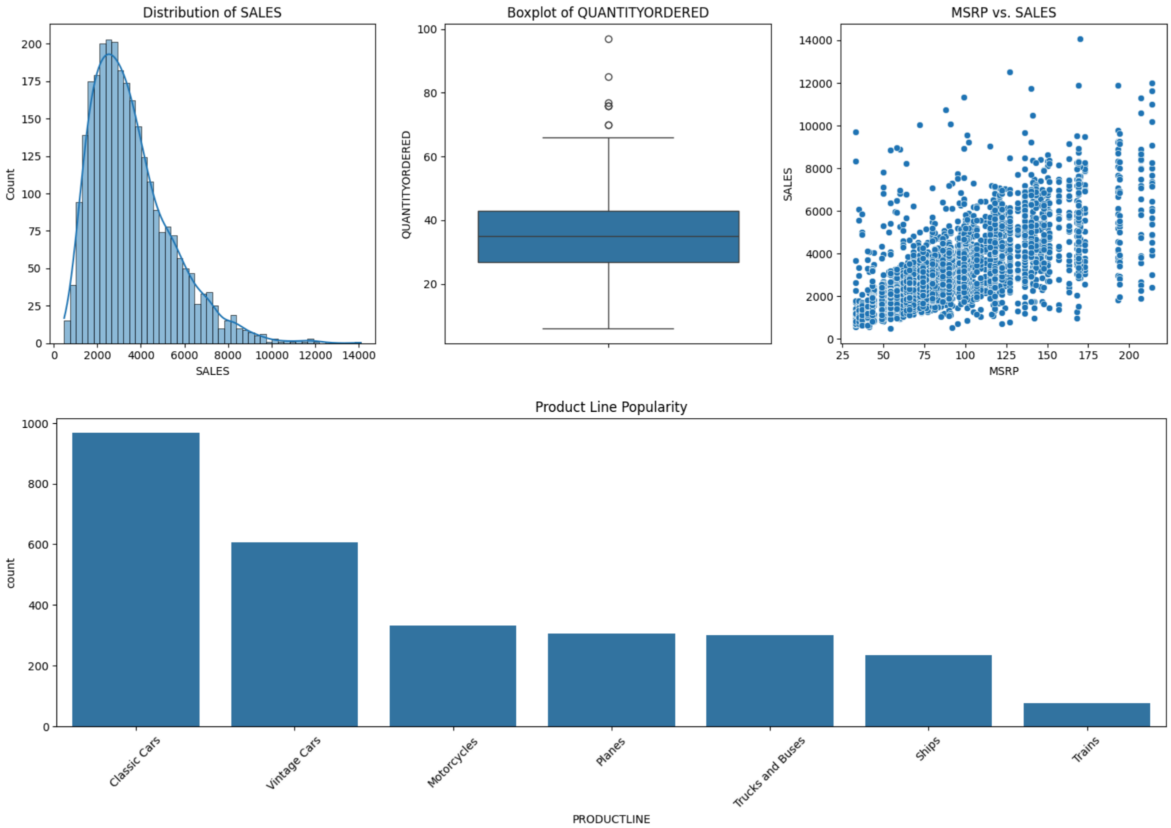
```python
print("Columns Dropped. Data Cleaned Info:")
df_clean.info()
```

```
Columns Dropped. Data Cleaned Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2823 entries, 0 to 2822
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   QUANTITYORDERED  2823 non-null   int64
 1   PRICEEACH        2823 non-null   float64
 2   SALES            2823 non-null   float64
 3   STATUS           2823 non-null   object
 4   PRODUCTLINE      2823 non-null   object
 5   MSRP             2823 non-null   int64
 6   PRODUCTCODE      2823 non-null   object
 7   CUSTOMERNAME     2823 non-null   object
 8   CITY             2823 non-null   object
 9   COUNTRY          2823 non-null   object
 10  DEALSIZE         2823 non-null   object
dtypes: float64(2), int64(2), object(7)
memory usage: 242.7+ KB
```

In [29]:

```
# 3.1 Review Descriptive Statistics
# ***FIXED: Removed 'DAYS_SINCE_LAST_ORDER' from this step.***
print("\nDescriptive Statistics on Key Numerical Features:")
print(df_clean[['QUANTITYORDERED', 'SALES', 'MSRP']].describe())

# 3.2 Numerical Feature Visualization
# ***ACTION: Visualize the distributions of key numerical features.***
plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
sns.histplot(df_clean['SALES'], kde=True, bins=50)
plt.title('Distribution of SALES')

plt.subplot(1, 3, 2)
sns.boxplot(y=df_clean['QUANTITYORDERED'])
plt.title('Boxplot of QUANTITYORDERED')

plt.subplot(1, 3, 3)
sns.scatterplot(x='MSRP', y='SALES', data=df_clean)
plt.title('MSRP vs. SALES')
plt.tight_layout()
plt.show()

# 3.3 Categorical Feature Visualization
# ***ACTION: Visualize counts of categorical features.***
plt.figure(figsize=(18, 5))
sns.countplot(x='PRODUCTLINE', data=df_clean, order=df_clean['PRODUCTLINE
plt.title('Product Line Popularity')
plt.xticks(rotation=45)
plt.show()
```

```
Descriptive Statistics on Key Numerical Features:
       QUANTITYORDERED         SALES         MSRP
count      2823.000000   2823.000000  2823.000000
mean         35.092809   3553.889072   100.715551
std           9.741443   1841.865106    40.187912
min           6.000000    482.130000    33.000000
25%          27.000000   2203.430000    68.000000
50%          35.000000   3184.800000    99.000000
75%          43.000000   4508.000000   124.000000
max          97.000000  14082.800000   214.000000
```



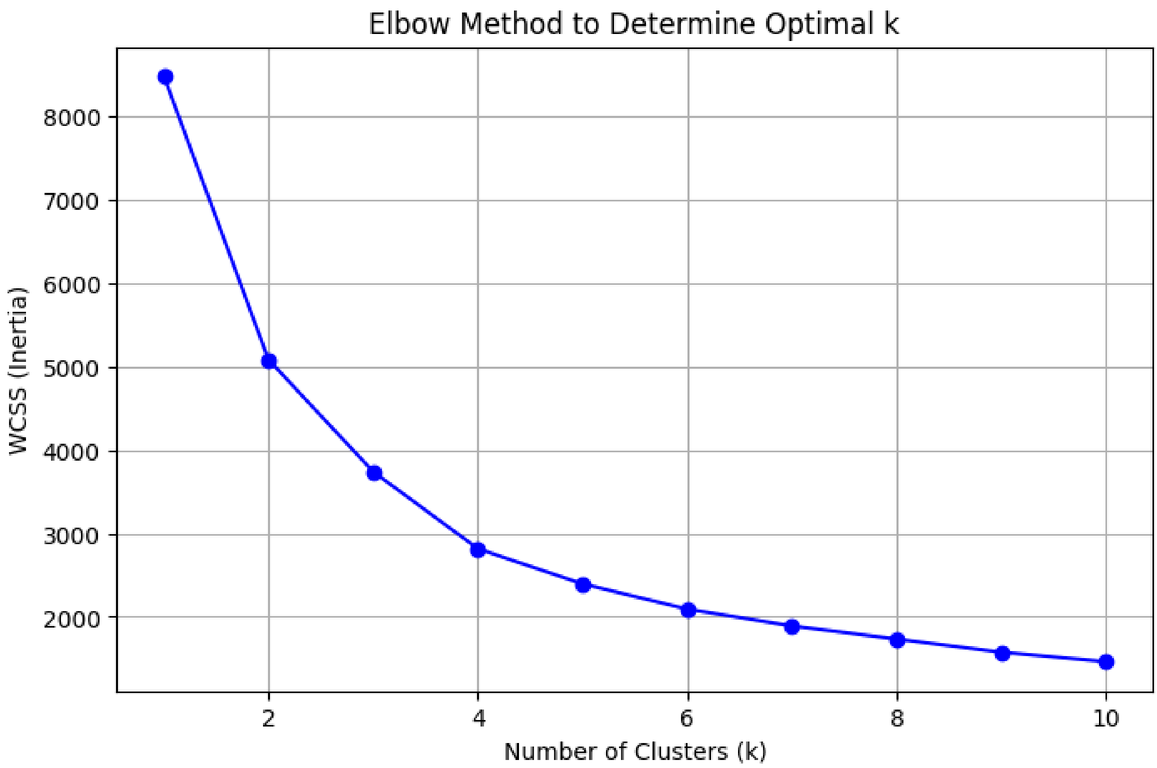
In [30]:

```python
# 4.1 Categorical Feature Encoding

list_cat = df_clean.select_dtypes(include=['object']).columns.tolist()
le = LabelEncoder()
for col in list_cat:
    df_clean[col] = le.fit_transform(df_clean[col])

print("\nFinal Data Info After Encoding:")
df_clean.info()

# 4.2 Feature Selection for Clustering
# We select features that define customer value (SALES, MSRP) and transac
X = df_clean[['SALES', 'MSRP', 'QUANTITYORDERED']]

# 4.3 Feature Scaling
# ***JUSTIFICATION: Scaling is MANDATORY for K-Means to prevent features
# from dominating the distance calculation.***
scaler = StandardScaler()
scaled_data = scaler.fit_transform(X)
scaled_df = pd.DataFrame(scaled_data, columns=X.columns)

print("\nFirst 5 Rows of Scaled Data (Ready for K-Means):")
print(scaled_df.head())
```

```
Final Data Info After Encoding:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2823 entries, 0 to 2822
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   QUANTITYORDERED  2823 non-null   int64  
 1   PRICEEACH        2823 non-null   float64
 2   SALES            2823 non-null   float64
 3   STATUS           2823 non-null   int64  
 4   PRODUCTLINE      2823 non-null   int64  
 5   MSRP             2823 non-null   int64  
 6   PRODUCTCODE      2823 non-null   int64  
 7   CUSTOMERNAME     2823 non-null   int64  
 8   CITY             2823 non-null   int64  
 9   COUNTRY          2823 non-null   int64  
 10  DEALSIZE         2823 non-null   int64  
dtypes: float64(2), int64(9)
memory usage: 242.7 KB

First 5 Rows of Scaled Data (Ready for K-Means):
      SALES      MSRP  QUANTITYORDERED
0 -0.370825 -0.142246        -0.522891
1 -0.427897 -0.142246        -0.112201
2  0.179443 -0.142246         0.606505
3  0.104701 -0.142246         1.017195
4  0.896740 -0.142246         1.427884
```

In [31]:

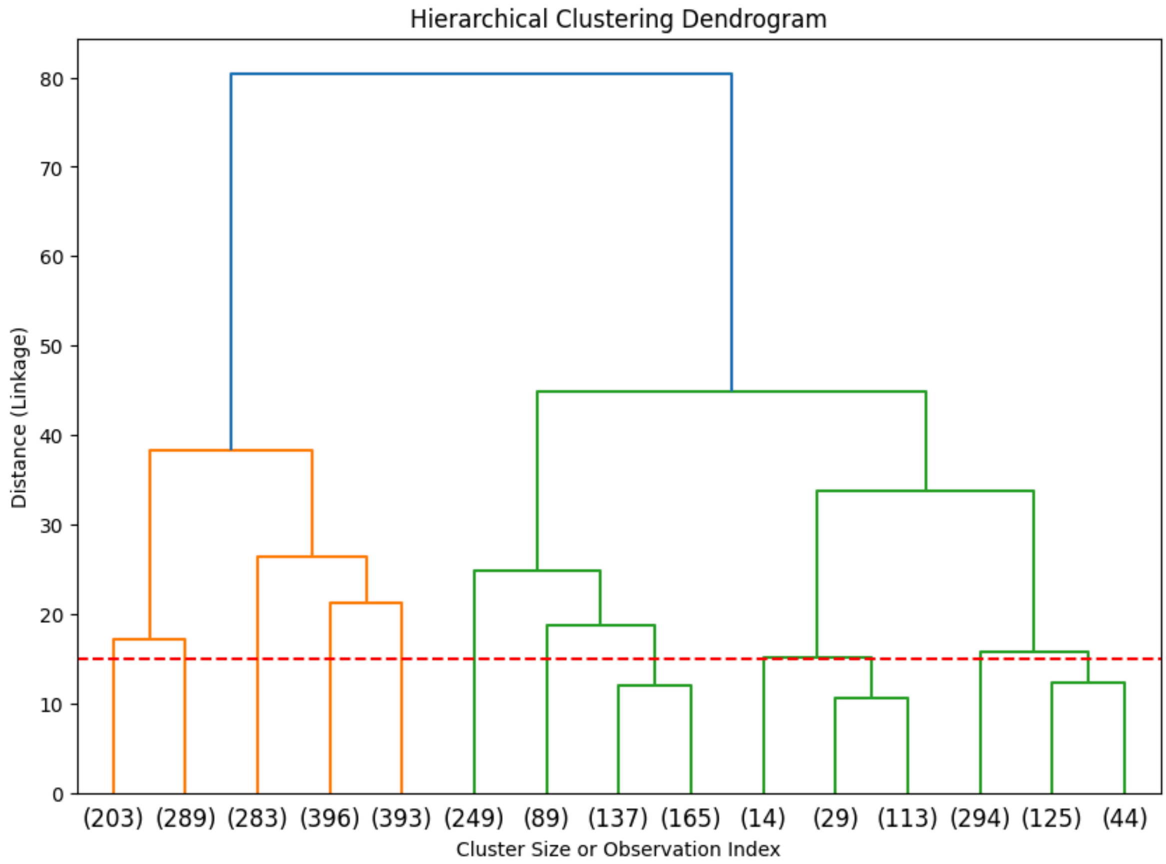
```
# 5.1 Determine Optimal k using the Elbow Method
wcss = []
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10) # n_init=10
    kmeans.fit(scaled_data)
    wcss.append(kmeans.inertia_)

# 5.2 Plot the Elbow Curve
plt.figure(figsize=(8, 5))
plt.plot(k_range, wcss, marker='o', linestyle='-', color='blue')
plt.title('Elbow Method to Determine Optimal k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('WCSS (Inertia)')
plt.grid(True)
plt.show()

# 5.3 Select Optimal k
# Based on the plot, we often choose k=3 or k=4. Let's proceed with **k=3
optimal_k = 3

print(f"\nOptimal k selected: {optimal_k}")
```




```
Optimal k selected: 3
```

In [32]:

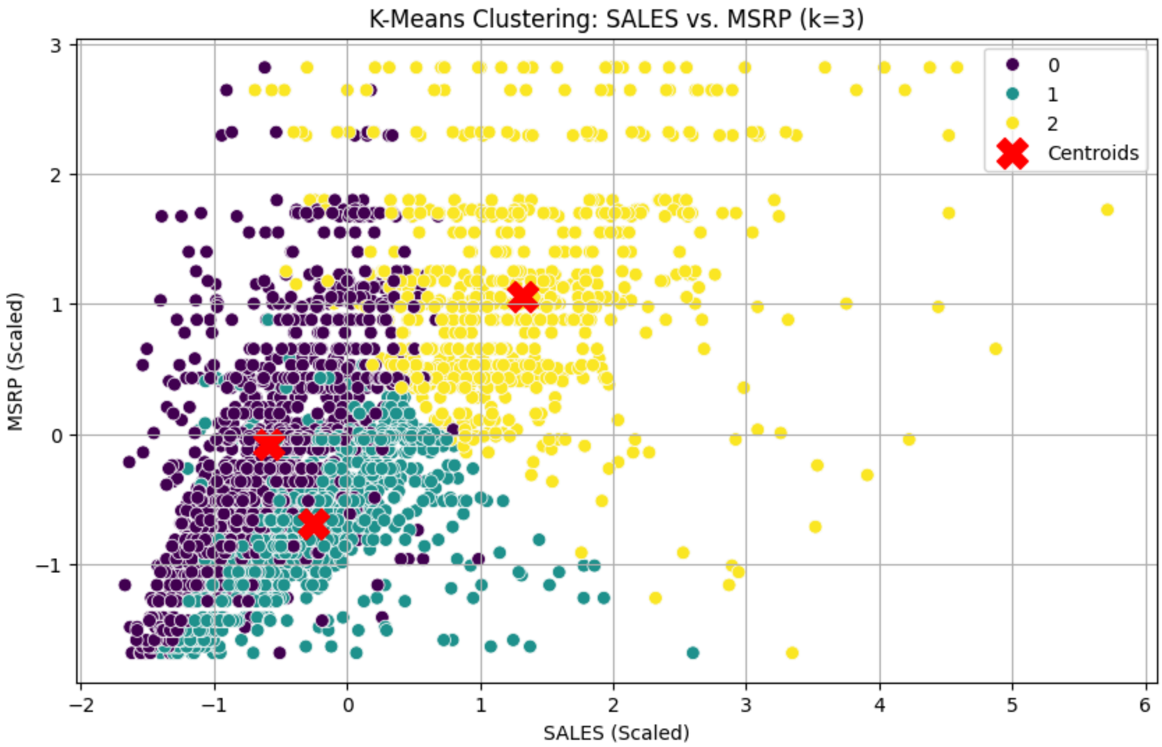
```
# 6.1 Generate Linkage Matrix

linked = linkage(scaled_data, method='ward')

# 6.2 Plot the Dendrogram

plt.figure(figsize=(10, 7))

dendrogram(linked, orientation='top', p=15, truncate_mode='lastp', show_l
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Cluster Size or Observation Index')
plt.ylabel('Distance (Linkage)')
plt.axhline(y=15, color='r', linestyle='--') # Horizontal line for cuttin
plt.show()


# 6.3 Final HAC Training and Assignment

hac = AgglomerativeClustering(n_clusters=optimal_k, metric='euclidean', l
df_clean['HAC_Cluster'] = hac.fit_predict(scaled_data)

print(f"\nHierarchical Clustering complete with k={optimal_k}.")
```




```
Hierarchical Clustering complete with k=3.
```

In [33]:
```
# 6.1 Final K-Means Training
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
df_clean['Cluster'] = kmeans.fit_predict(scaled_data)
```

In [34]:
```
# 6.2 Cluster Visualization

plt.figure(figsize=(10, 6))
# Plotting two of the chosen features
sns.scatterplot(x=scaled_df['SALES'], y=scaled_df['MSRP'],
                hue=df_clean['Cluster'], palette='viridis', s=50)

# Plot the Centroids (Cluster Centers)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            marker='X', s=250, color='red', label='Centroids')
plt.title(f'K-Means Clustering: SALES vs. MSRP (k={optimal_k})')
plt.xlabel('SALES (Scaled)')
plt.ylabel('MSRP (Scaled)')
plt.legend()
plt.grid(True)
plt.show()
```



In [35]:
```
# 6.3 Cluster Profiling (The Ultimate Evaluation)

cluster_profile = df_clean.groupby('Cluster')[X.columns.tolist()].mean()
print("\n--- Cluster Profiling (Mean Feature Values for Interpretation) -
print(cluster_profile)
```

```
--- Cluster Profiling (Mean Feature Values for Interpretation) ---
               SALES        MSRP  QUANTITYORDERED
Cluster
0        2469.047831   97.366102        25.985593
1        3087.418518   72.864606        41.352878
2        5990.289135  143.377305        42.007092
```

In [ ]: