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

import warnings
warnings.filterwarnings("ignore")

df=pd.read_csv(r"C:\Users\mreth\Downloads\Data.csv")

df.head()

  item  store_id  2023/1  2023/2  2023/3  2023/4  2023/5  2023/6  
2023/7  \
0    A         1     NaN     NaN     4.0     NaN     5.0     NaN   
5.0
1    A         2     5.0     NaN     5.0     NaN     NaN     NaN   
NaN
2    A         3     5.0    10.0    50.0    10.0    30.0    10.0   
30.0
3    A         4    20.0     NaN    20.0     NaN    30.0     0.0   
NaN
4    A         5     NaN    20.0     NaN    25.0    20.0    10.0   
NaN

   2023/8  ...  2022/3  2022/4  2022/5  2022/6  2022/7  2022/8  2022/9  
\
0     NaN  ...    10.0     NaN     NaN     NaN     NaN     NaN     NaN   

1     5.0  ...     5.0     5.0     NaN     6.0     5.0     NaN     2.0   

2    15.0  ...    20.0    35.0    20.0    27.0    21.0     NaN    20.0   

3    40.0  ...    10.0    20.0    10.0    20.0    20.0     NaN    20.0   

4    20.0  ...    10.0    10.0    10.0    10.0     NaN    20.0     NaN   


   2022/10  2022/11  2022/12  
0      NaN      NaN      NaN  
1      5.0      NaN      NaN  
2     15.0     15.0     20.0  
3     20.0      NaN     10.0  
4      NaN     20.0      5.0  

[5 rows x 23 columns]

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5393 entries, 0 to 5392
```

```
Data columns (total 23 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   item      5393 non-null   object
 1   store_id  5393 non-null   int64
 2   2023/1    1276 non-null   float64
 3   2023/2    1425 non-null   float64
 4   2023/3    1992 non-null   float64
 5   2023/4    2498 non-null   float64
 6   2023/5    1357 non-null   float64
 7   2023/6    1126 non-null   float64
 8   2023/7    1604 non-null   float64
 9   2023/8    1366 non-null   float64
 10  2023/9    1169 non-null   float64
 11  2022/1    942 non-null    float64
 12  2022/2    1010 non-null   float64
 13  2022/3    1074 non-null   float64
 14  2022/4    1954 non-null   float64
 15  2022/5    1447 non-null   float64
 16  2022/6    1623 non-null   float64
 17  2022/7    891 non-null    float64
 18  2022/8    832 non-null    float64
 19  2022/9    1000 non-null   float64
 20  2022/10   1516 non-null   float64
 21  2022/11   1531 non-null   float64
 22  2022/12   1366 non-null   float64
dtypes: float64(21), int64(1), object(1)
memory usage: 969.2+ KB
```

```
df.describe()
```

```
          store_id       2023/1       2023/2       2023/3       2023/4
\
count  5393.000000  1276.000000  1425.000000  1992.000000  2498.000000

mean    472.825885    13.234326    18.954386    23.986948    46.966773

std     277.945220    23.597323    43.535857    58.504996   173.626181

min       1.000000    -1.000000   -15.000000   -13.000000   -20.000000

25%     233.000000     4.000000     5.000000     5.000000     5.000000

50%     464.000000     8.000000    10.000000    10.000000    10.000000

75%     709.000000    11.000000    20.000000    20.000000    26.000000

max    1032.000000   350.000000   800.000000  1000.000000  3800.000000


            2023/5       2023/6       2023/7       2023/8       2023/9
```

```
...  \
count  1357.000000  1126.000000  1604.000000  1366.000000  1169.000000
...
mean      8.263080    10.635879    11.956359    11.161054    10.617622
...
std      49.766246    15.616934    13.191212    14.669761    14.315611
...
min    -967.000000    -1.000000     0.000000    -5.000000   -10.000000
...
25%       3.000000     3.000000     4.000000     4.000000     3.000000
...
50%       8.000000     7.000000    10.000000     8.000000     7.000000
...
75%      15.000000    10.000000    15.000000    12.000000    10.000000
...
max     600.000000   240.000000   140.000000   240.000000   230.000000
...

             2022/3       2022/4       2022/5       2022/6       2022/7
\
count  1074.000000  1954.000000  1447.000000  1623.000000   891.000000

mean     13.155493    24.653531    24.974430    29.182378    46.138047

std      24.831905    50.762534    61.716066    60.893228   184.439911

min       0.000000     1.000000   -17.000000   -20.000000   -20.000000

25%       4.000000     7.000000     5.000000     6.000000     4.000000

50%       7.000000    10.000000    10.000000    13.000000    10.000000

75%      11.000000    20.000000    20.000000    25.000000    20.000000

max     305.000000   760.000000  1000.000000   640.000000  2800.000000


            2022/8       2022/9      2022/10      2022/11      2022/12

count  832.000000  1000.000000  1516.000000  1531.000000  1366.000000

mean    18.627404    11.474000    11.143140    12.710647    16.885066

std     63.958324    22.752824    16.045232    21.341555    59.126591

min      1.000000  -300.000000   -13.000000   -20.000000   -20.000000

25%      4.000000     5.000000     4.000000     4.000000     4.000000

50%      8.000000    10.000000     8.000000     8.000000     8.000000
```

```
75%     13.000000    13.000000    11.000000    12.000000    14.000000

max    980.000000   292.000000   215.000000   290.000000  1400.000000


[8 rows x 22 columns]

# Replace missing values (NaN) with 0
df.fillna(0, inplace=True)

# Display the first few rows after replacing missing values with 0
print(df.head())

  item  store_id  2023/1  2023/2  2023/3  2023/4  2023/5  2023/6
2023/7  \
0    A         1     0.0     0.0     4.0     0.0     5.0     0.0
5.0
1    A         2     5.0     0.0     5.0     0.0     0.0     0.0
0.0
2    A         3     5.0    10.0    50.0    10.0    30.0    10.0
30.0
3    A         4    20.0     0.0    20.0     0.0    30.0     0.0
0.0
4    A         5     0.0    20.0     0.0    25.0    20.0    10.0
0.0

   2023/8  ...  2022/3  2022/4  2022/5  2022/6  2022/7  2022/8  2022/9
\
0     0.0  ...    10.0     0.0     0.0     0.0     0.0     0.0     0.0

1     5.0  ...     5.0     5.0     0.0     6.0     5.0     0.0     2.0

2    15.0  ...    20.0    35.0    20.0    27.0    21.0     0.0    20.0

3    40.0  ...    10.0    20.0    10.0    20.0    20.0     0.0    20.0

4    20.0  ...    10.0    10.0    10.0    10.0     0.0    20.0     0.0


   2022/10  2022/11  2022/12
0      0.0      0.0      0.0
1      5.0      0.0      0.0
2     15.0     15.0     20.0
3     20.0      0.0     10.0
4      0.0     20.0      5.0

[5 rows x 23 columns]
```

Product Insight

```
# Create a copy of the original DataFrame to work on product insights
df_pi = df.copy()

#  Create 'Accumulated Sales in 2023' by summing up sales for the
months in 2023
df_pi['accumulate_sales_2023'] = df_pi[['2023/1', '2023/2', '2023/3',
'2023/4', '2023/5', '2023/6', '2023/7', '2023/8',
'2023/9']].sum(axis=1)

# Create 'Accumulated Sales in 2022' by summing up sales for the
months in 2022
df_pi['accumulate_sales_2022'] = df_pi[['2022/1', '2022/2', '2022/3',
'2022/4', '2022/5', '2022/6', '2022/7', '2022/8', '2022/9', '2022/10',
'2022/11', '2022/12']].sum(axis=1)

# Select only the columns that we need for the product insights
result_df = df_pi[['item', 'accumulate_sales_2023',
'accumulate_sales_2022']]

# Group by 'item' and sum the accumulated sales for each product
pi_result = result_df.groupby('item').sum()

# Display the results for review
print(pi_result.head(10))
```

```
      accumulate_sales_2023  accumulate_sales_2022
item
A                   76261.0                89259.0
B                   75129.0                84832.0
C                   57549.0                66626.0
D                   36190.0                42797.0
E                    4699.0                 4810.0
F                    7621.0                 2632.0
G                   18336.0                 6011.0
H                    3242.0                 1784.0
```

```
# Calculate the growth rate between 2022 and 2023
pi_result['growth_rate'] = ((pi_result['accumulate_sales_2023'] -
pi_result['accumulate_sales_2022']) /
pi_result['accumulate_sales_2022']) * 100

# Display the result with growth rates
print(pi_result)
```

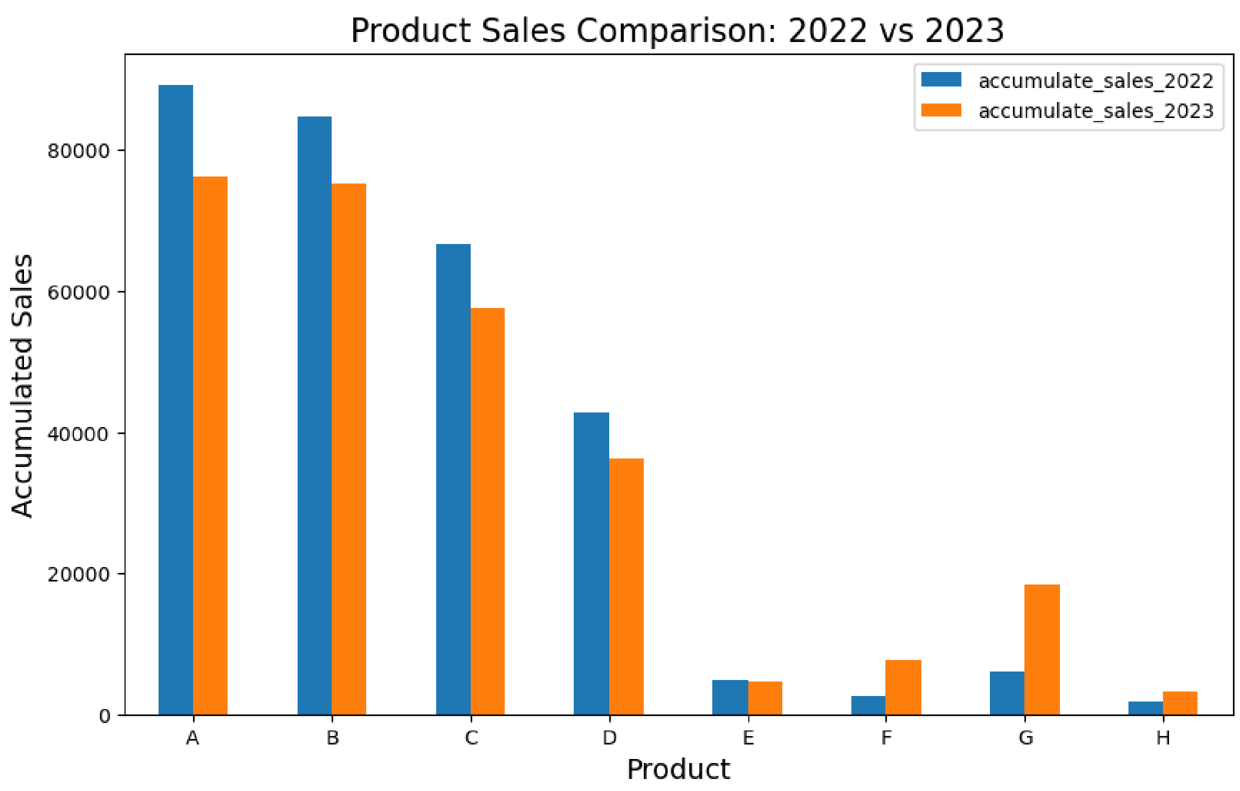
```
      accumulate_sales_2023  accumulate_sales_2022  growth_rate
item
A                   76261.0                89259.0   -14.562117
B                   75129.0                84832.0   -11.437901
```

```
C                     57549.0                 66626.0   -13.623811
D                     36190.0                 42797.0   -15.437998
E                      4699.0                  4810.0    -2.307692
F                      7621.0                  2632.0   189.551672
G                     18336.0                  6011.0   205.040759
H                      3242.0                  1784.0    81.726457
```

Insights and Key Observations from Product Sales Data

- **Declining Products**:
  - Items **A**, **B**, **C**, and **D** have experienced a **decline in sales** from 2022 to 2023, with decreases ranging between 11.5% to 15.4%. These products may require attention to understand the causes of the drop in demand, such as changes in market trends, increased competition, or ineffective marketing efforts.
- **Growth Products**:
  - Items **F**, **G**, and **H** have shown **significant sales growth** in 2023 compared to 2022. For example, **Item G** saw a **205% increase**, and **Item F** grew by **189.5%**. These items could be considered emerging products with high potential, warranting further investment in marketing and inventory.
- **Stable Product**:
  - **Item E** remained relatively **stable** with no significant changes in sales between 2022 and 2023. This product may not require immediate changes in strategy, but it should be monitored for potential future shifts in demand.
- **General Trend**:
  - There is a clear **divergence in product performance**: while some items are rapidly growing, others are showing signs of decline. This calls for a **targeted approach** where marketing efforts should focus on boosting the performance of declining products, while ensuring continued support for high-growth items.

```python
# Plot a bar chart to compare accumulated sales in 2022 and 2023
pi_result[['accumulate_sales_2022',
'accumulate_sales_2023']].plot(kind='bar', figsize=(10, 6))

# Add title and labels
plt.title('Product Sales Comparison: 2022 vs 2023', fontsize=16)
plt.xlabel('Product', fontsize=14)
plt.ylabel('Accumulated Sales', fontsize=14)

# Rotate x-axis labels for better readability
plt.xticks(rotation=0)

plt.show()
```



```
# Use melt to convert the data into long format
date_columns = [col for col in df.columns if '/' in col]  #
Identifying date columns

df_long = pd.melt(df, id_vars=['item', 'store_id'],
value_vars=date_columns,
                  var_name='date', value_name='sales')

# Convert the 'date' column into proper datetime format
df_long['date'] = pd.to_datetime(df_long['date'], format='%Y/%m')

# Display the first few rows of the reshaped dataframe
print(df_long.head(10))
```

```
  item  store_id       date  sales
0    A         1 2023-01-01    0.0
1    A         2 2023-01-01    5.0
2    A         3 2023-01-01    5.0
3    A         4 2023-01-01   20.0
4    A         5 2023-01-01    0.0
5    A         6 2023-01-01   20.0
6    A         7 2023-01-01   30.0
7    A         8 2023-01-01   20.0
8    A         9 2023-01-01    0.0
9    A        10 2023-01-01   37.0
```

Data Reshaping Using `melt()`

In this step, we are transforming our dataset from **wide format** (where each month is represented as a column) into **long format** using the `melt()` function. This approach simplifies working with time-based data, making it easier to perform time-series analysis and visualizations.

By converting the data to long format:

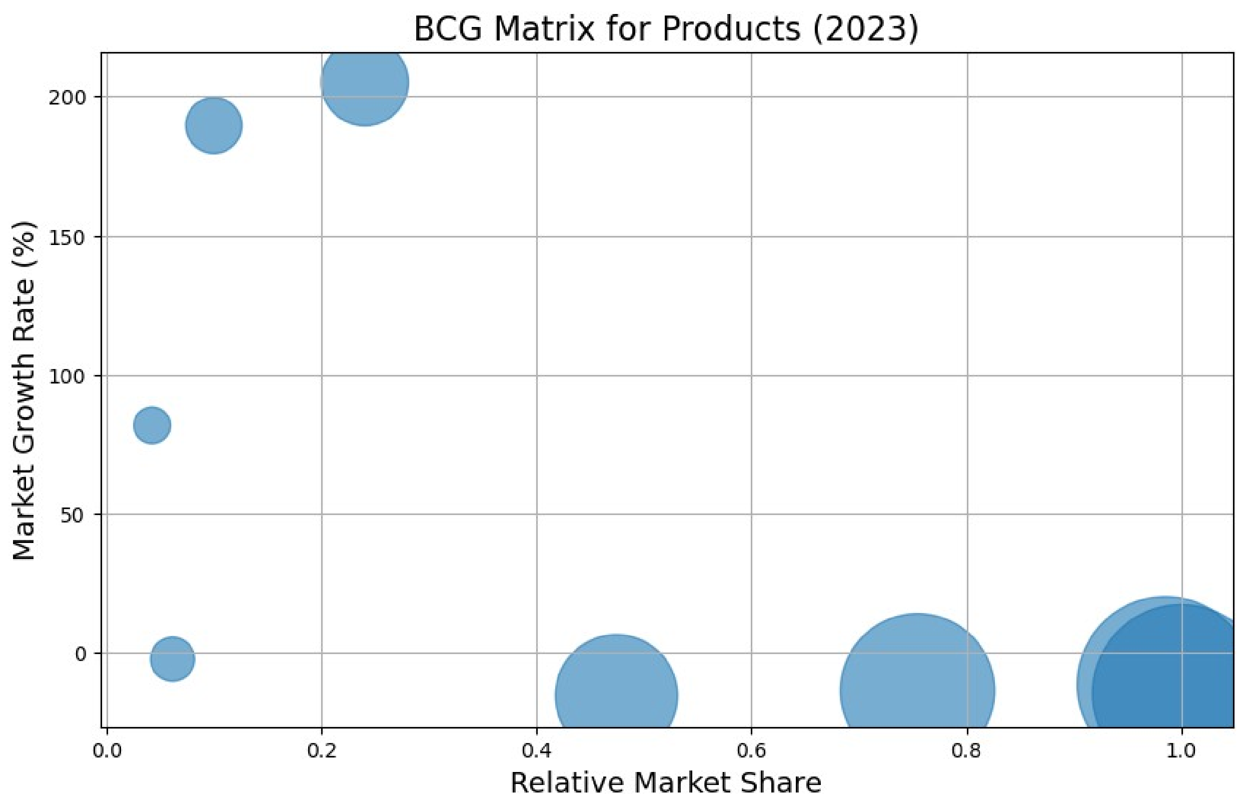
- We get a single `date` column that contains all the months, making it easier to group and analyze monthly trends.
- The `sales` column will hold the corresponding sales values for each product and store across different months.

This structure is particularly useful when performing time-series analysis, such as tracking product sales over time or identifying seasonal patterns.

```
# Calculate market growth rate (sales growth) as a percentage change
from 2022 to 2023
pi_result['market_growth_rate'] = ((pi_result['accumulate_sales_2023']
- pi_result['accumulate_sales_2022']) /
pi_result['accumulate_sales_2022']) * 100

# Calculate relative market share by dividing 2023 sales by the max
sales in 2023
pi_result['market_share'] = pi_result['accumulate_sales_2023'] /
pi_result['accumulate_sales_2023'].max()

# Display the results for BCG Matrix
print(pi_result[['market_growth_rate', 'market_share']])
```

```
      market_growth_rate  market_share
item
A             -14.562117      1.000000
B             -11.437901      0.985156
C             -13.623811      0.754632
D             -15.437998      0.474554
E              -2.307692      0.061617
F             189.551672      0.099933
G             205.040759      0.240437
H              81.726457      0.042512
```

```
# Plot BCG Matrix using market growth rate and market share
plt.figure(figsize=(10, 6))
plt.scatter(pi_result['market_share'],
pi_result['market_growth_rate'],
s=pi_result['accumulate_sales_2023']*0.1, alpha=0.6)

# Add labels and title
plt.title('BCG Matrix for Products (2023)', fontsize=16)
plt.xlabel('Relative Market Share', fontsize=14)
plt.ylabel('Market Growth Rate (%)', fontsize=14)

# Add grid
plt.grid(True)

# Show plot
plt.show()
```



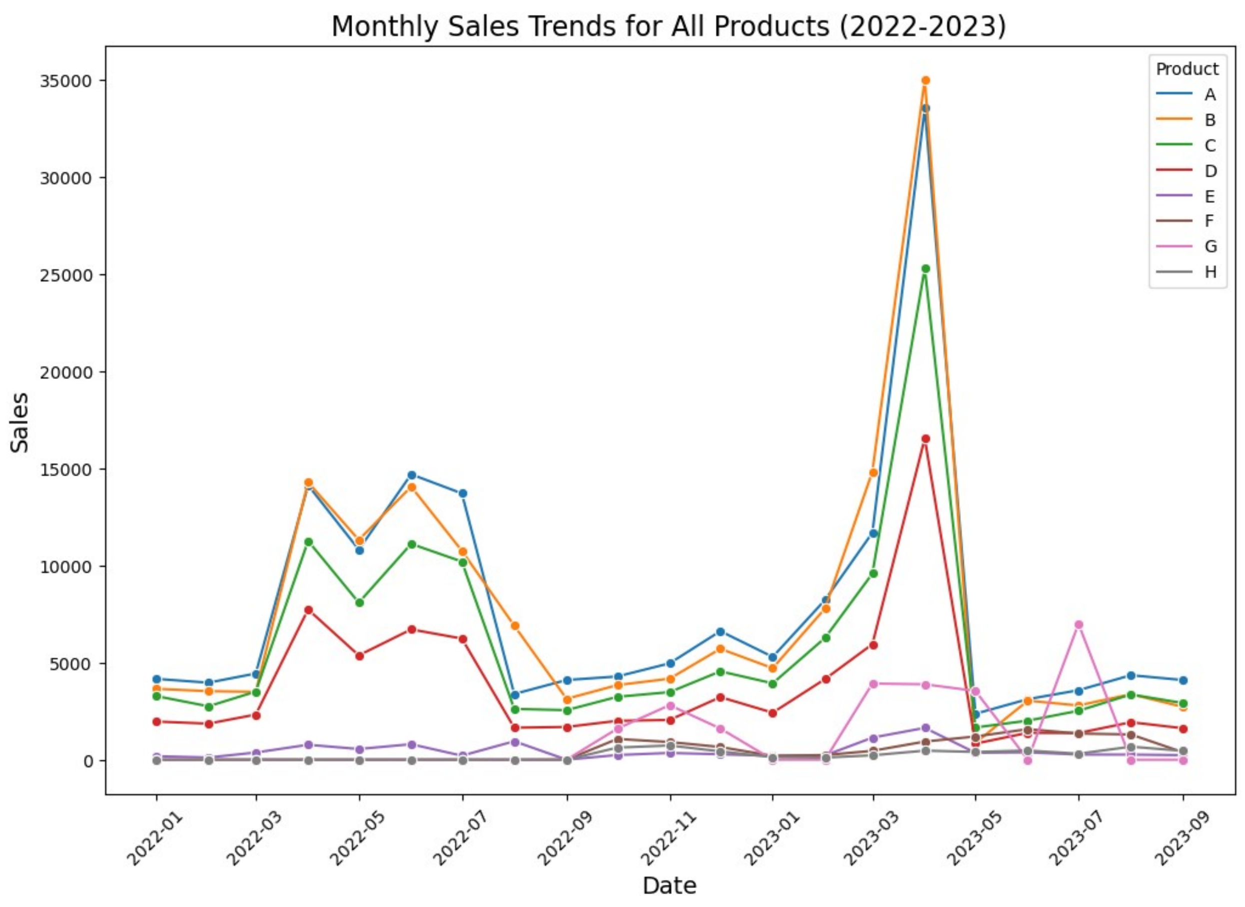
Insights from BCG Matrix for Products (2023)

The BCG Matrix provides insights into product performance by visualizing market growth rate and relative market share. Here are the key observations:

1. **High Growth, Low Market Share (Question Marks)**:
   - Some products (such as those near the top of the chart) show high market growth rates but have relatively low market shares. These products may represent opportunities for further investment to increase market dominance.
2. **High Growth, High Market Share (Stars)**:
   - Products with high market growth and high market share (e.g., those located in the top right quadrant) are performing well. These are the company's "Stars" and should continue to receive attention and resources to maintain their position in the market.
3. **Low Growth, High Market Share (Cash Cows)**:

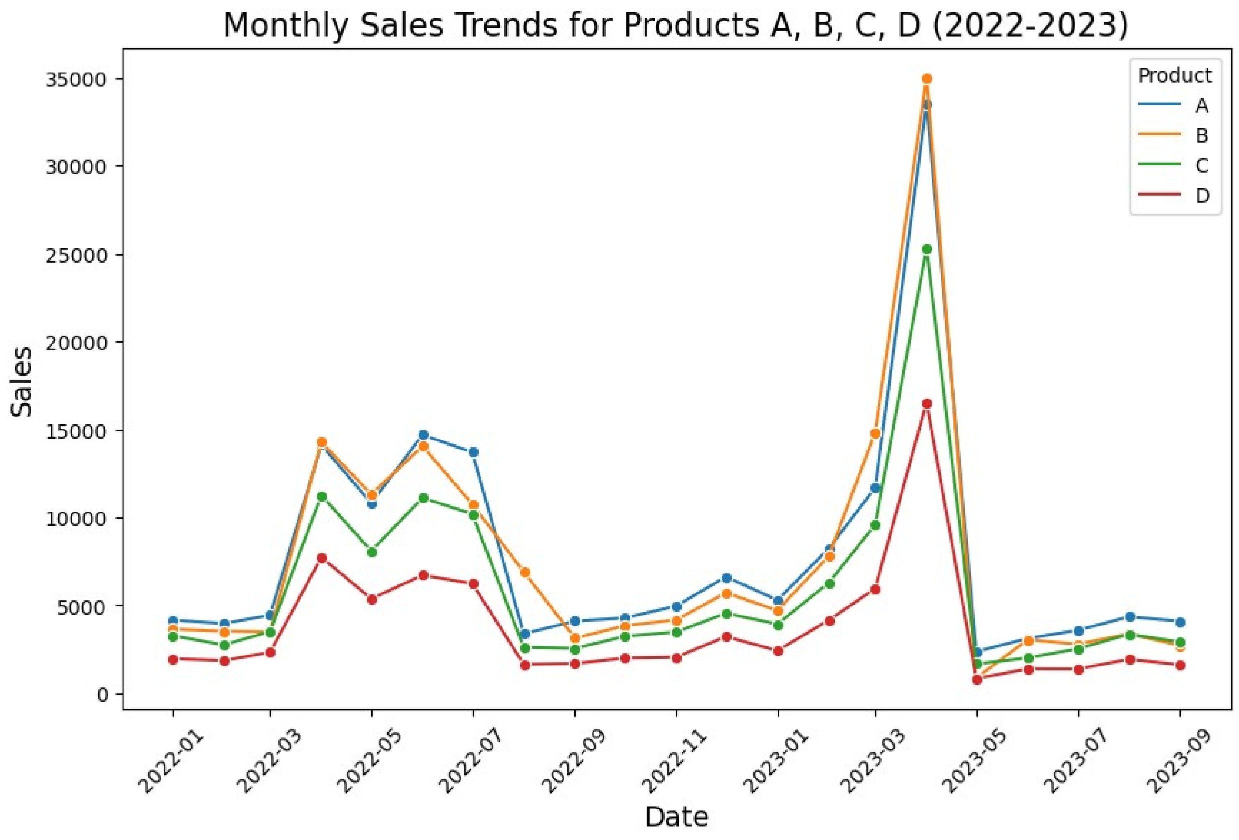
    - Products positioned near the bottom right show lower market growth rates but still maintain a high relative market share. These are "Cash Cows" — established products that generate stable returns with minimal investment.
4. **Low Growth, Low Market Share (Dogs)**:
    - Products in the bottom left quadrant have low market growth and low market share. These "Dogs" might need to be reconsidered, as they may not be contributing significantly to the company's overall growth.

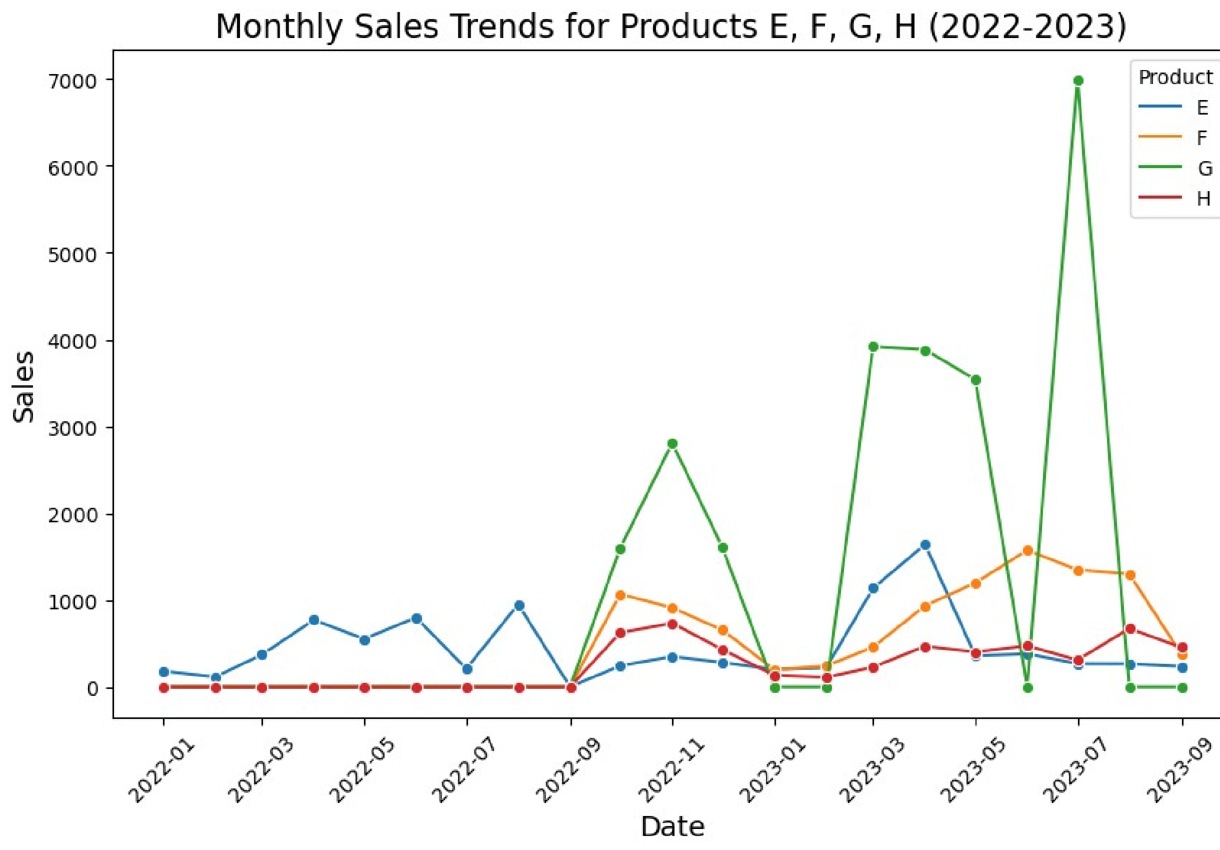
```
# Group data by item and date, summing the sales for each month
df_grouped = df_long.groupby(['item', 'date']).sum().reset_index()

# Set up the figure
plt.figure(figsize=(12, 8))

# Plot the sales trend for each item in a single line graph with
different colors
sns.lineplot(x='date', y='sales', hue='item', data=df_grouped,
marker='o')

# Add title and labels
plt.title('Monthly Sales Trends for All Products (2022-2023)',
fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Sales', fontsize=14)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

# Show the plot
plt.legend(title='Product')
plt.show()
```



```python
# Create a subset for items A, B, C, D
df_abcd = df_grouped[df_grouped['item'].isin(['A', 'B', 'C', 'D'])]

# Create a subset for items E, F, G, H
df_efgh = df_grouped[df_grouped['item'].isin(['E', 'F', 'G', 'H'])]

# Plot 1: Sales trend for items A, B, C, D
plt.figure(figsize=(10, 6))
sns.lineplot(x='date', y='sales', hue='item', data=df_abcd,
marker='o')
plt.title('Monthly Sales Trends for Products A, B, C, D (2022-2023)',
fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Sales', fontsize=14)
plt.xticks(rotation=45)
plt.legend(title='Product')
plt.show()

# Plot 2: Sales trend for items E, F, G, H
plt.figure(figsize=(10, 6))
```

```python
sns.lineplot(x='date', y='sales', hue='item', data=df_efgh,
marker='o')
plt.title('Monthly Sales Trends for Products E, F, G, H (2022-2023)',
fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Sales', fontsize=14)
plt.xticks(rotation=45)
plt.legend(title='Product')
plt.show()
```

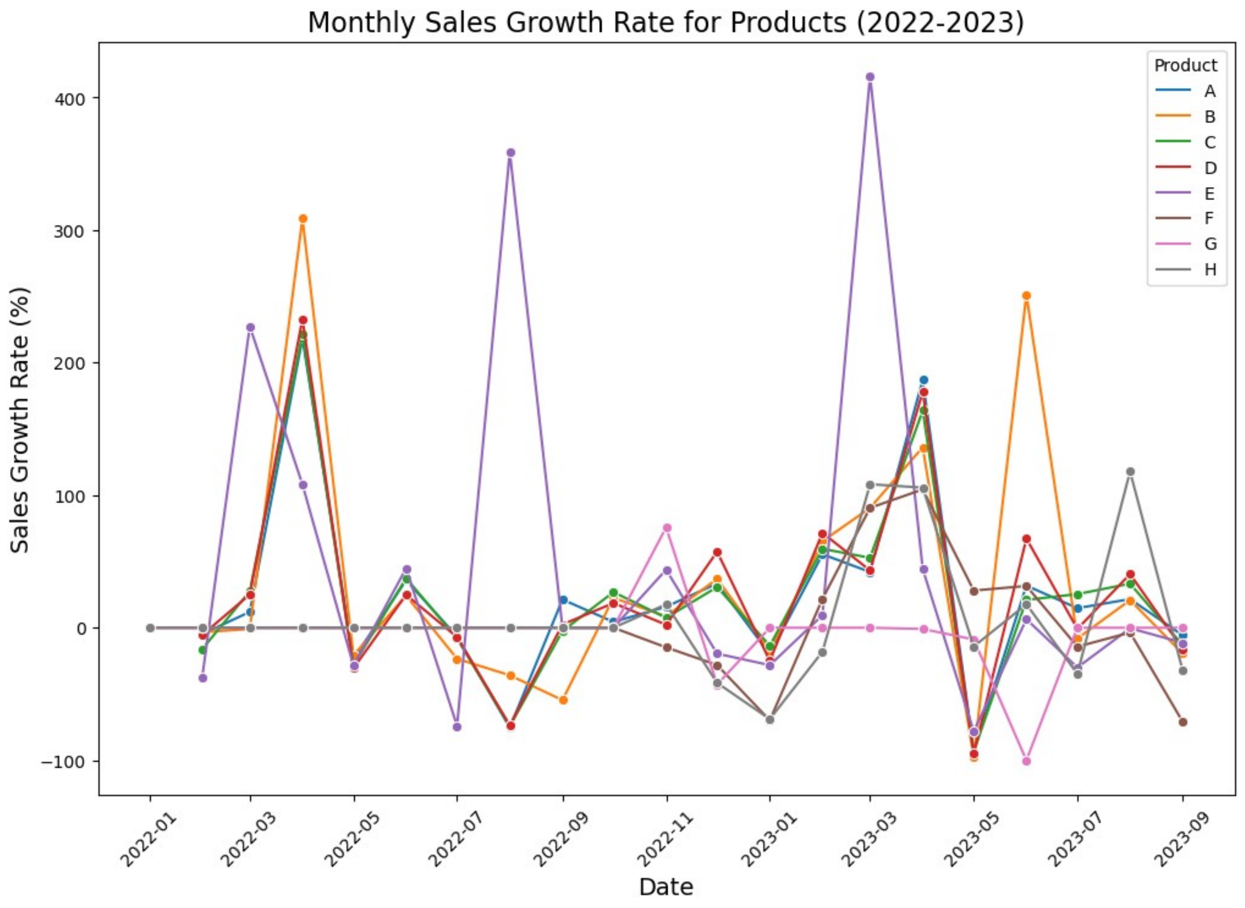




Observations from Monthly Sales Trends (2022-2023)

- **Seasonality**: Both groups exhibit potential **seasonal sales trends**, with distinct sales spikes occurring in **March** of both years, particularly in 2023. This suggests there may be external factors or promotional efforts driving these increases.
- **Focus on Product G**: Given its high sales volatility and large spikes, **Product G** could be an emerging performer with untapped potential, warranting further focus and analysis.
- **Low Performers**: **Products H** and **E** show consistently low sales, with **Product H** particularly underperforming. These products may need further investigation to understand the cause of low demand.

```
# Calculate percentage change (growth rate) while avoiding division by
zero
df_grouped['sales_growth_rate'] = df_grouped.groupby('item')
['sales'].pct_change() * 100

# Handle cases where the previous month's sales were zero to avoid
extreme growth rates
df_grouped['sales_growth_rate'] = df_grouped.apply(
    lambda row: 0 if row['sales'] == 0 or row['sales_growth_rate'] >
1000 or row['sales_growth_rate'] < -1000 else
row['sales_growth_rate'],
    axis=1
)

# Visualize the corrected growth rates with line plots
plt.figure(figsize=(12, 8))
sns.lineplot(x='date', y='sales_growth_rate', hue='item',
data=df_grouped, marker='o')

# Add title and labels
plt.title('Monthly Sales Growth Rate for Products (2022-2023)',
fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Sales Growth Rate (%)', fontsize=14)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

# Show the plot
plt.legend(title='Product')
plt.show()
```



Key Insights from Monthly Sales Growth Rate (2022-2023)

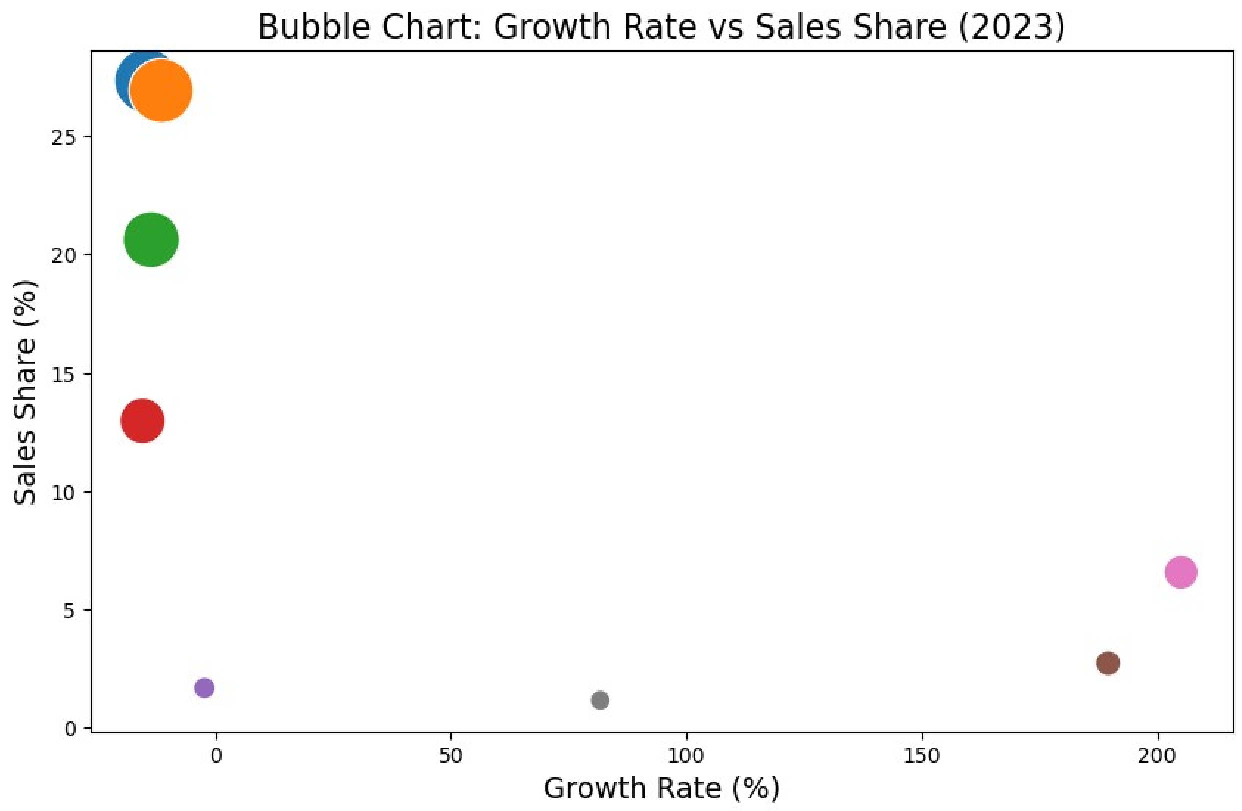
- **Significant growth spikes** are seen in Products B and G, particularly in **March-April** periods of both years, indicating potential seasonality or promotional effects.
- **Product G** shows extreme volatility, with frequent sharp increases and declines, suggesting unstable performance.
- **Products E and H** show minimal fluctuations, indicating stable but low growth across the period.
- Multiple products exhibit **negative growth** during mid-2022 and early 2023, suggesting sales drops that may need further investigation.

```
# Calculate total sales for 2022 and 2023 across all products
total_sales_2022 = pi_result['accumulate_sales_2022'].sum()
total_sales_2023 = pi_result['accumulate_sales_2023'].sum()

# Calculate the sales share for 2022 and 2023
pi_result['sales_share_2022'] = (pi_result['accumulate_sales_2022'] /
total_sales_2022) * 100
pi_result['sales_share_2023'] = (pi_result['accumulate_sales_2023'] /
total_sales_2023) * 100

# Display the result with sales share
print(pi_result[['accumulate_sales_2022', 'accumulate_sales_2023',
'sales_share_2022', 'sales_share_2023']])
```

```
      accumulate_sales_2022  accumulate_sales_2023
sales_share_2022  \
item

A                   89259.0                76261.0          29.877390

B                   84832.0                75129.0          28.395553

C                   66626.0                57549.0          22.301515

D                   42797.0                36190.0          14.325308

E                    4810.0                 4699.0           1.610036

F                    2632.0                 7621.0           0.881001

G                    6011.0                18336.0           2.012043

H                    1784.0                 3242.0           0.597153


      sales_share_2023
item
A            27.331047
B            26.925351
```

```
C            20.624886
D            12.970071
E             1.684066
F             2.731277
G             6.571407
H             1.161895
```

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

# Create a new DataFrame for the bubble chart data
bubble_data = pd.DataFrame({
    'item': pi_result.index,
    'growth_rate': pi_result['growth_rate'],
    'sales_share': pi_result['sales_share_2023'],
    'sales_volume': pi_result['accumulate_sales_2023']
})

# Create the bubble chart
plt.figure(figsize=(10, 6))
bubble_plot = sns.scatterplot(
    x='growth_rate',
    y='sales_share',
    size='sales_volume',
    hue='item',
    data=bubble_data,
    sizes=(100, 1000),   # Adjust bubble size range
    legend=False
)

# Add title and labels
plt.title('Bubble Chart: Growth Rate vs Sales Share (2023)',
fontsize=16)
plt.xlabel('Growth Rate (%)', fontsize=14)
plt.ylabel('Sales Share (%)', fontsize=14)

# Display the plot
plt.show()
```

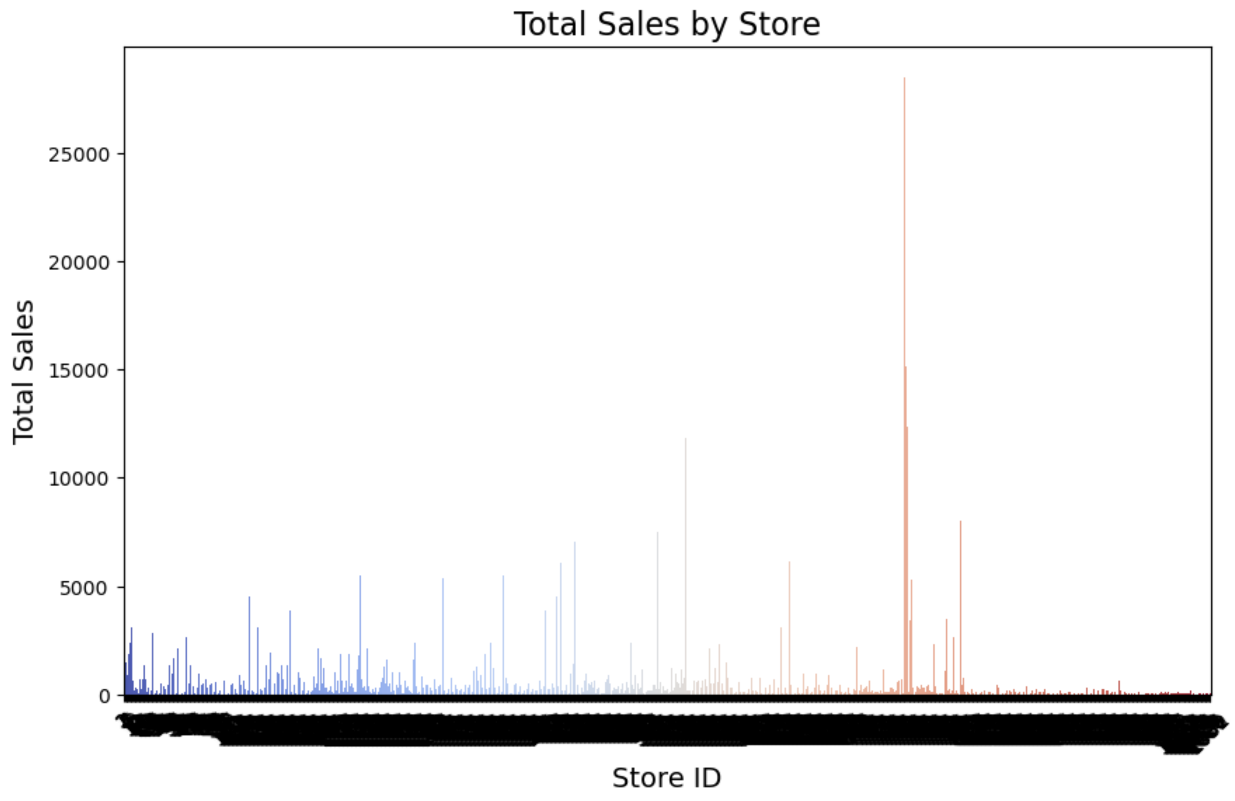


Key Insights from the Bubble Chart: Growth Rate vs Sales Share (2023)

- **Products A, B, and C** have the **largest sales share** in 2023 (over 20%), indicating that they dominate the market. However, their growth rates are relatively low, suggesting they are **mature products** with stable sales but limited growth potential.
- **Product G** stands out with a **high growth rate** (over 200%) and a moderate sales share (~6.5%), indicating it is an **emerging product** with significant potential for future growth.
- **Products F and H** show **smaller sales shares** but still exhibit **positive growth**. **Product F** has a growth rate of over 100%, suggesting it's experiencing growth but hasn't yet captured a large market share.
- **Product D** has a decent sales share (~13%) but shows limited growth, suggesting it is stable but not growing significantly.
- **Product E** has the smallest sales share and minimal growth, making it the weakest performer among all products.

Store Insights

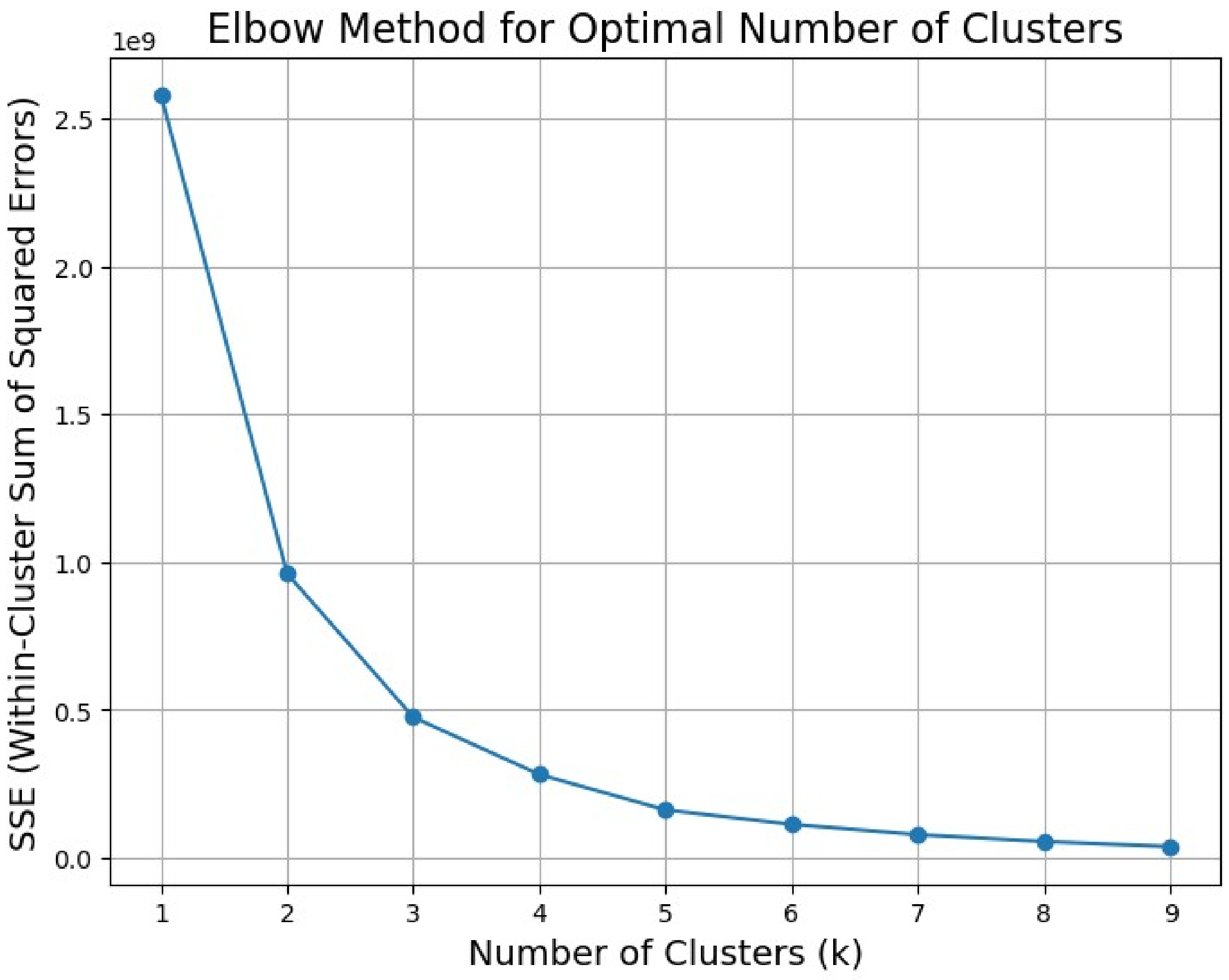
```
# Calculate total sales for each store across all months and products
df_store_sales = df_long.groupby('store_id')
['sales'].sum().reset_index()

# Sort stores by total sales in descending order
df_store_sales = df_store_sales.sort_values(by='sales',
ascending=False)

# Visualize total sales by store with a bar chart
plt.figure(figsize=(10, 6))
sns.barplot(x='store_id', y='sales', data=df_store_sales,
palette='coolwarm')

# Add title and labels
plt.title('Total Sales by Store', fontsize=16)
plt.xlabel('Store ID', fontsize=14)
plt.ylabel('Total Sales', fontsize=14)

# Rotate x-axis labels for better readability if necessary
plt.xticks(rotation=45)

plt.show()
```



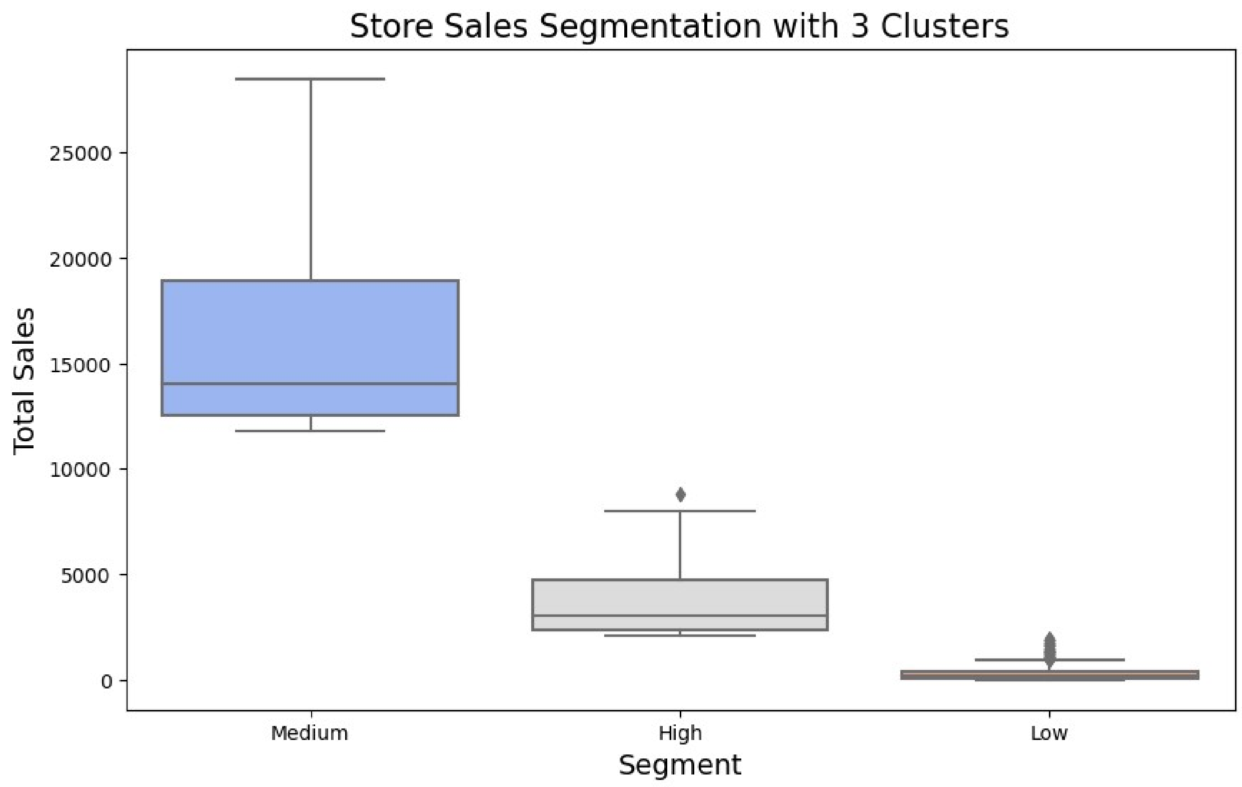
Elbow Method to Find Optimal Clusters

```
from sklearn.cluster import KMeans

# Reshape the data to fit the K-Means clustering model
X = df_store_sales[['sales']].values

# Calculate the Within-Cluster Sum of Squared Errors (SSE) for
```

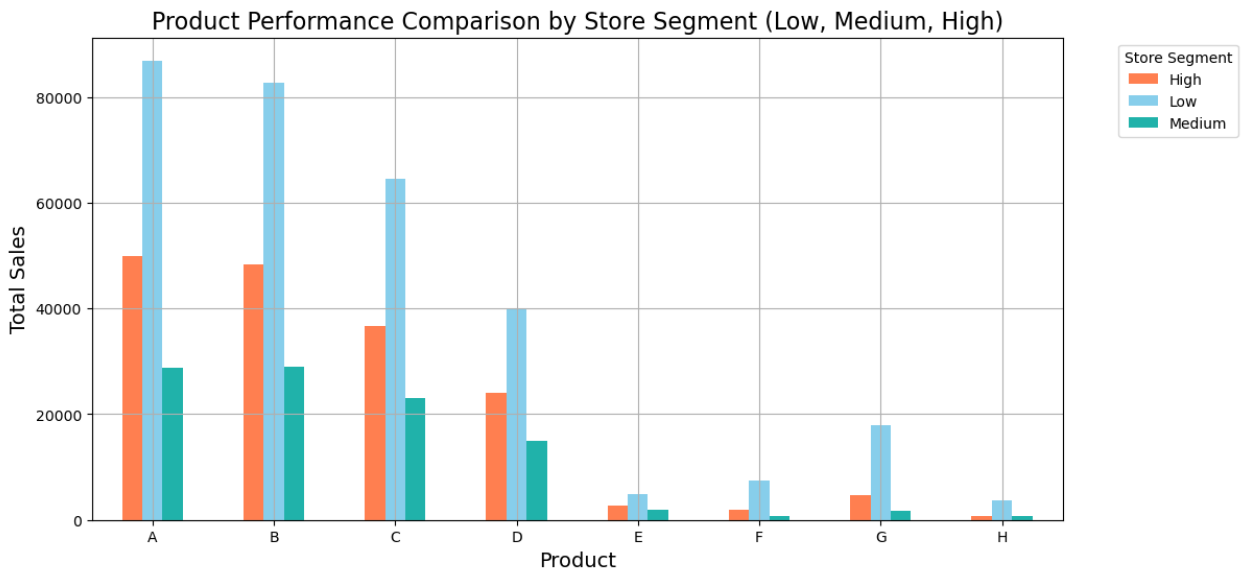
```
different cluster sizes
sse = []
cluster_range = range(1, 10)  # Try k values from 1 to 10

for k in cluster_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    sse.append(kmeans.inertia_)  # Inertia is the sum of squared
distances to the closest cluster center

# Plot the SSE values to find the "elbow"
plt.figure(figsize=(8, 6))
plt.plot(cluster_range, sse, marker='o')
plt.title('Elbow Method for Optimal Number of Clusters', fontsize=16)
plt.xlabel('Number of Clusters (k)', fontsize=14)
plt.ylabel('SSE (Within-Cluster Sum of Squared Errors)', fontsize=14)
plt.grid(True)
plt.show()
```



```python
# Apply K-Means clustering with 3 clusters
optimal_k = 3  # Based on the elbow method

kmeans = KMeans(n_clusters=optimal_k, random_state=42)
df_store_sales['segment'] = kmeans.fit_predict(X)

# Map the segment labels to meaningful names (optional)
df_store_sales['segment'] = df_store_sales['segment'].map({0: 'Low',
1: 'Medium', 2: 'High'})

# Visualize the store segmentation
plt.figure(figsize=(10, 6))
sns.boxplot(x='segment', y='sales', data=df_store_sales,
palette='coolwarm')

# Add title and labels
plt.title(f'Store Sales Segmentation with {optimal_k} Clusters',
fontsize=16)
plt.xlabel('Segment', fontsize=14)
plt.ylabel('Total Sales', fontsize=14)
plt.show()
```



```python
# Ensure the 'segment' column is in df_long by merging it from
df_store_sales
df_long = df_long.merge(df_store_sales[['store_id', 'segment']],
on='store_id', how='left')

# Group the data by 'segment' and 'item' to calculate the total sales
for each product in each store segment
df_item_sales_by_segment = df_long.groupby(['segment', 'item'])
['sales'].sum().reset_index()

# Pivot the table to make it easier to plot (items as rows, segments
as columns)
df_item_sales_pivot = df_item_sales_by_segment.pivot(index='item',
columns='segment', values='sales').fillna(0)

# Visualize the product performance across the store segments
df_item_sales_pivot.plot(kind='bar', figsize=(12, 6),
color=['#FF7F50', '#87CEEB', '#20B2AA'])
plt.title('Product Performance Comparison by Store Segment (Low,
Medium, High)', fontsize=16)
plt.xlabel('Product', fontsize=14)
plt.ylabel('Total Sales', fontsize=14)
plt.xticks(rotation=0)
plt.legend(title='Store Segment', bbox_to_anchor=(1.05, 1), loc='upper
left')
plt.grid(True)
plt.show()
```



```
print("Product Sales by Store Segment:")
print(df_item_sales_pivot)
```

```
Product Sales by Store Segment:
segment     High      Low   Medium
item
A        49849.0  86867.0  28804.0
B        48318.0  82687.0  28956.0
C        36661.0  64410.0  23104.0
D        23985.0  40027.0  14975.0
E         2747.0   4861.0   1901.0
F         2005.0   7435.0    813.0
G         4698.0  17985.0   1664.0
H          710.0   3656.0    660.0
```

Product-Level Insights

Based on the store segmentation and product sales comparison, the following key product-related insights were identified:

1. **High Sales in Low-Performing Stores**:
   - Despite being labeled as "low-performing," stores in this segment generate the highest sales across core products like **A, B, and C**. This suggests that the segmentation based on total sales may not fully reflect the actual performance of the stores, and the label "low-performing" may be misleading for certain key products.
2. **Underperformance of Medium-Tier Stores**:
   - **Medium-tier stores** consistently underperform across all product categories. They contribute the least to sales, even for top products like **A, B, and C**. These stores have untapped potential but may face operational bottlenecks or poor product availability.
3. **Moderate Performance in High-Tier Stores**:
   - **High-tier stores** show moderate performance in key products like **A, B, and C**, lagging behind low-performing stores. This suggests high-tier stores are not fully capitalizing on sales opportunities in these products. However, they do perform relatively better in niche products like **E and G**.

Supply Chain Perspective Insights

1. **Possible Inventory and Distribution Issues in High-Tier Stores**:
   - The relatively lower sales of core products **A, B, and C** in high-tier stores may indicate **inventory management issues** or **supply chain bottlenecks**. High-tier stores should be performing better in these high-demand products, but their lagging performance suggests **supply shortages**, **late deliveries**, or **inconsistent stock levels** affecting their ability to meet customer demand.
2. **Stocking Strategy in Medium-Tier Stores**:
   - The poor performance of medium-tier stores across all product categories suggests a need for a **better stocking and replenishment strategy**. These stores

might be facing frequent **stockouts**, preventing them from capitalizing on sales opportunities. Addressing these supply chain inefficiencies could help boost sales across all product categories.

3. **Excessive Inventory in Low-Performing Stores**:
   - Low-performing stores, despite generating high sales in key products, may be **overstocked** or have **excess inventory**. This could indicate an imbalance in supply chain distribution, where some stores receive excess stock while others face shortages. Fine-tuning the distribution system could help align inventory with actual store demand, reducing the risk of overstocking or understocking.
4. **Niche Product Availability**:
   - The low sales of **niche products (E, F, G, H)** across all store segments may suggest that these products are **either not stocked regularly** or **not prioritized** in terms of supply. If these products are meant to play a larger role in the product lineup, the supply chain needs to focus on improving the **availability and distribution** of these items across all store segments.

Conclusion from Supply Chain Perspective

- **Inventory Optimization Needed**: Both **high-tier and medium-tier stores** seem to struggle with product availability for key items. Optimizing inventory levels and distribution schedules could help improve sales performance.

- **Distribution Imbalance**: The high sales in "low-performing" stores suggest an **imbalance in product distribution**, where certain stores are overstocked while others may face shortages. The supply chain should aim to distribute stock more effectively based on the actual demand at each store.

- **Focus on Medium-Tier Stores**: Since these stores are consistently underperforming, further investigation into **supply chain inefficiencies**, such as late deliveries or stockouts, is necessary. Ensuring that medium-tier stores receive timely and adequate inventory could help unlock additional sales potential.