STEP 1: BUSINESS PROBLEM

A multi-store retail company operating across multiple cities sells a wide range of products across categories such as groceries, household essentials, and personal care. Each store manages its own inventory, but the current inventory planning process relies heavily on historical averages, fixed reorder rules, and manual judgment.

This approach fails to capture store-level demand variations, seasonal trends, price changes, and promotional effects, leading to frequent stockouts of high-demand products and overstocking of slow-moving items. As a result, the company faces increased holding costs, lost sales opportunities, and inconsistent product availability across stores.

The company requires a data-driven solution to forecast product demand at the store level and leverage these forecasts to optimize inventory planning decisions, with the goal of reducing stockouts, minimizing excess inventory, and improving overall operational efficiency.

BUSINESS QUESTIONS

1. Performance & Demand Understanding

📌 Purpose:Understand store-level and product-level demand differences

- Which stores are performing well and which are underperforming?
- Which products are high-, medium-, and low-demand?
- Does product demand vary across stores and locations?

2. Trend & Seasonality Questions

📌 Purpose: Decide time-series approach, seasonality handling, and forecast horizon

- How is demand trending over time at each store?
- Are there seasonal or monthly demand patterns?
- What factors drive these patterns (festivals, months, etc.)?

3. Demand Stability & Uncertainty

📌 Purpose:Decide safety stock levels and inventory risk

- Which product has stable demand?
- Which products are highly unpredictable or violatile?

4. other

- How do promotions and discount affect demand?
- Does price change significantly impact demand?

5. Forecasting Strategy Question

- At what level should demand be forecasted?(store-level, product-level, category-level, daily vs weekly)

6. Inventory Risk & Impact

- Are stockholders occurring for high demand products?
- Which products contribute moves to stock outs and overstocking?

BUSINESS PROBLEM TO DATASCIENCE PROBLEM

Design and implement a data-driven system to forecast future product demand at the store level using historical sales data, seasonal patterns, pricing, and promotional information, and use these demand forecasts to optimize inventory replenishment decisions in order to minimize stockouts and excess inventory across multiple retail stores.

OBJECTIVE OF THE PROJECT

- Analyze historical sales data to understand demand patterns across stores and products.
- Identify trends, seasonality, and demand variability at the store–product level.
- Build demand forecasting models to predict future sales.
- Evaluate forecast accuracy using appropriate business-relevant metrics.
- Use demand forecasts to design inventory optimization rules.
- Reduce stockouts and excess inventory through data-driven decisions.

STEP 2: Data Understanding & Data Requirements (Data Collection)

In [1]:
```
# IMPORTING AND INSTALLING NECESSARY LIBRARIES
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# display settings
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 120)
```

In [3]:
```
# LOAD THE DATASET
data = pd.read_csv("data/retail_store_inventory.csv")
```

In [4]:
```
# INSPECT THE DATASET
data.head(10)
```

Out[4]:

	Date	Store ID	Product ID	Category	Region	Inventory Level	Units Sold	Units Ordered	Demand Forecast	Price	Discount	Weat Condit
0	01-01-2022	S001	P0001	Groceries	North	231	127	55	135.47	33.50	20	Ra
1	01-01-2022	S001	P0002	Toys	South	204	150	66	144.04	63.01	20	Su
2	01-01-2022	S001	P0003	Toys	West	102	65	51	74.02	27.99	10	Su
3	01-01-2022	S001	P0004	Toys	North	469	61	164	62.18	32.72	10	Clo
4	01-01-2022	S001	P0005	Electronics	East	166	14	135	9.26	73.64	0	Su
5	01-01-2022	S001	P0006	Groceries	South	138	128	102	139.82	76.83	10	Su
6	01-01-2022	S001	P0007	Furniture	East	359	66	167	108.92	34.16	10	Ra
7	01-01-2022	S001	P0008	Clothing	North	380	312	54	329.73	97.99	5	Clo
8	01-01-2022	S001	P0009	Electronics	West	183	175	135	174.15	20.74	10	Clo
9	01-01-2022	S001	P0010	Toys	South	108	28	196	24.47	59.99	0	Ra

In [5]:
```
data.shape
```

Out[5]:
```
(73100, 15)
```

In [6]:
```
data.columns
```

Out[6]:
```
Index(['Date', 'Store ID', 'Product ID', 'Category', 'Region', 'Inventory Level', 'Units Sold',
'Units Ordered',
       'Demand Forecast', 'Price', 'Discount', 'Weather Condition', 'Holiday/Promotion', 'Compet
itor Pricing',
       'Seasonality'],
      dtype='object')
```

In [7]:
```
# BASIC DATA INFORMATION
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73100 entries, 0 to 73099
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Date                73100 non-null  object 
 1   Store ID            73100 non-null  object 
 2   Product ID          73100 non-null  object 
 3   Category            73100 non-null  object 
 4   Region              73100 non-null  object 
 5   Inventory Level     73100 non-null  int64  
 6   Units Sold          73100 non-null  int64  
 7   Units Ordered       73100 non-null  int64  
 8   Demand Forecast     73100 non-null  float64
 9   Price               73100 non-null  float64
 10  Discount            73100 non-null  int64  
 11  Weather Condition   73100 non-null  object 
 12  Holiday/Promotion   73100 non-null  int64  
 13  Competitor Pricing  73100 non-null  float64
 14  Seasonality         73100 non-null  object 
dtypes: float64(3), int64(5), object(7)
memory usage: 8.4+ MB
```

In [8]:

```
# CHECK MISSING VALUES)
data.isnull().sum()
```

Out[8]:

```
Date                  0
Store ID              0
Product ID            0
Category              0
Region                0
Inventory Level       0
Units Sold            0
Units Ordered         0
Demand Forecast       0
Price                 0
Discount              0
Weather Condition     0
Holiday/Promotion     0
Competitor Pricing    0
Seasonality           0
dtype: int64
```

In [9]:

```
# CHECK DUPLICATE VALUES
data.duplicated().sum()
```

Out[9]:

```
np.int64(0)
```

STEP 3: DATA CLEANING AND PREPROCESSING

In [10]:

```
# CONVERT THE DATE COLUMN TO DATETIME
# - converting datatype of date(object) to datetime

data['Date']=pd.to_datetime(data['Date'], dayfirst=True)
```

In [11]:

```
# Sort Data by Time - sort data chronologically
data = data.sort_values(by='Date')
```

In [12]:

```
# reset after sorting
data = data.reset_index(drop=True)
```

In [13]:

```
data.head()
```

Out[13]:

	Date	Store ID	Product ID	Category	Region	Inventory Level	Units Sold	Units Ordered	Demand Forecast	Price	Discount	Wea Condi
0	2022-01-01	S001	P0001	Groceries	North	231	127	55	135.47	33.50	20	F
1	2022-01-01	S004	P0013	Furniture	East	191	56	65	54.47	61.81	0	Su
2	2022-01-01	S004	P0012	Electronics	North	349	9	165	0.95	14.25	5	F
3	2022-01-01	S004	P0011	Electronics	West	205	46	27	46.65	54.84	0	Su
4	2022-01-01	S004	P0010	Groceries	East	447	104	96	115.03	33.48	15	Clo

In [14]:

```
# Check date range
data['Date'].min(), data['Date'].max()
```

Out[14]:

```
(Timestamp('2022-01-01 00:00:00'), Timestamp('2024-01-01 00:00:00'))
```

In [15]:

```
# Check target variable
data['Units Sold'].describe()
```

Out[15]:

```
count    73100.000000
mean       136.458550
std        108.897078
min          0.000000
25%         49.000000
50%        107.000000
75%        203.000000
max        499.000000
Name: Units Sold, dtype: float64
```

Target Variable Sanity Check Insights

- The Units Sold variable contains 73,100 valid observations with no missing values
- Sales values range from zero to 499 units, indicating the presence of both low-demand and high-demand scenarios. (Understanding is based upon The mean, which is zero, and the maximum is 499)
- The graph is right skewed as median < Mean Which means the sales values are non negative and realistic.
- The relatively high standard deviation compared to the mean suggests significant demand variability across products and stores, reinforcing the need for robust demand forecasting and inventory planning.

In [16]:

```
# Describe Other Numerical Columns
numerical_cols = [
    'Inventory Level',
    'Units Ordered',
    'Price',
    'Discount',
    'Competitor Pricing'
]

data[numerical_cols].describe()
```

Out[16]:

	Inventory Level	Units Ordered	Price	Discount	Competitor Pricing
count	73100.000000	73100.000000	73100.000000	73100.000000	73100.000000
mean	274.469877	110.004473	55.135108	10.009508	55.146077
std	129.949514	52.277448	26.021945	7.083746	26.191408
min	50.000000	20.000000	10.000000	0.000000	5.030000
25%	162.000000	65.000000	32.650000	5.000000	32.680000
50%	273.000000	110.000000	55.050000	10.000000	55.010000
75%	387.000000	155.000000	77.860000	15.000000	77.820000
max	500.000000	200.000000	100.000000	20.000000	104.940000

The numerical values in the dataset are realistic and consistent. Inventory levels, order quantities, prices, discounts, and competitor pricing fall within reasonable ranges. The variation across these values shows that different products and stores operate under different conditions, which supports the need for store- and product-level demand forecasting and inventory planning.

In [17]:

```
# Identify Categorical Columns
categorical_cols = [
    'Store ID',
    'Product ID',
    'Category',
    'Region',
    'Weather Condition',
    'Seasonality'
]
```

In [18]:

```
# Check Unique Values (Cardinality)
for col in categorical_cols:
    print(f"\n{col}")
    print("Unique values:", data[col].nunique())
    print(data[col].unique())
```

```
Store ID
Unique values: 5
['S001' 'S004' 'S003' 'S005' 'S002']

Product ID
Unique values: 20
['P0001' 'P0013' 'P0012' 'P0011' 'P0010' 'P0009' 'P0008' 'P0007' 'P0006'
 'P0005' 'P0004' 'P0003' 'P0002' 'P0020' 'P0019' 'P0018' 'P0017' 'P0016'
 'P0015' 'P0014']

Category
Unique values: 5
['Groceries' 'Furniture' 'Electronics' 'Clothing' 'Toys']

Region
Unique values: 4
['North' 'East' 'West' 'South']

Weather Condition
Unique values: 4
['Rainy' 'Sunny' 'Cloudy' 'Snowy']

Seasonality
Unique values: 4
['Autumn' 'Spring' 'Summer' 'Winter']
```

In [19]:

```
# check for missing values
data[categorical_cols].isnull().sum()
```

Out[19]:
```
Store ID             0
Product ID           0
Category             0
Region               0
Weather Condition    0
Seasonality          0
dtype: int64
```

In [20]:
```
# Fix Casing & Extra Spaces
for col in categorical_cols:
    data[col] = data[col].str.strip()
    data[col] = data[col].str.title()
```

STEP 4: Exploratpry DATA ANALYSIS (EDA)

Question: Which stores are performing well and which are underperforming?

We want to know:

- Which stores sell more units overall
- Which stores sell less
- Whether performance differs significantly across stores

This helps business decide:

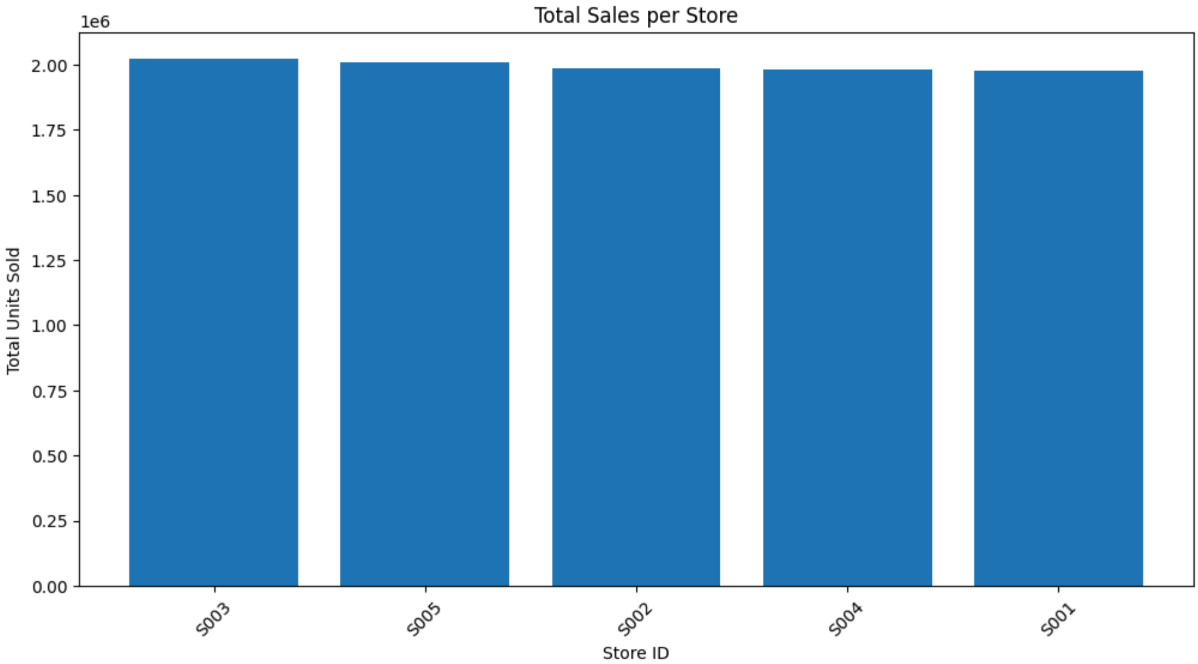
- Where to increase inventory
- Where to optimize or reduce stock

In [21]:
```
# Total Units Sold per Store
store_sales = (
    data
    .groupby('Store ID')['Units Sold']
    .sum()
    .reset_index()
    .sort_values(by='Units Sold', ascending=False)
)

store_sales
```

Out[21]:

	Store ID	Units Sold
2	S003	2022874
4	S005	2010200
1	S002	1987765
3	S004	1979228
0	S001	1975053

In [22]:
```
# Visualize Store Performance
plt.figure(figsize=(12,6))
plt.bar(store_sales['Store ID'], store_sales['Units Sold'])
plt.xlabel('Store ID')
plt.ylabel('Total Units Sold')
plt.title('Total Sales per Store')
plt.xticks(rotation=45)
plt.show()
```



In [23]:

```
# Average Daily Units Sold per Store
store_avg_sales = (
    data
    .groupby('Store ID')['Units Sold']
    .mean()
    .reset_index()
    .sort_values(by='Units Sold', ascending=False)
)

store_avg_sales
```

Out[23]:

	Store ID	Units Sold
2	S003	138.363475
4	S005	137.496580
1	S002	135.962038
3	S004	135.378112
0	S001	135.092544

Question: Which products are high-, medium-, and low-demand?

We want to know:

- Which products sell a lot (high demand)
- Which products sell moderately
- Which products sell very little (slow movers)

This helps business decide:

- Which products need frequent replenishment
- Which products need controlled inventory

In [24]:

```
# Total Demand per Product
product_sales = (
    data
    .groupby('Product ID')['Units Sold']
    .sum()
    .reset_index()
    .sort_values(by='Units Sold', ascending=False)
```

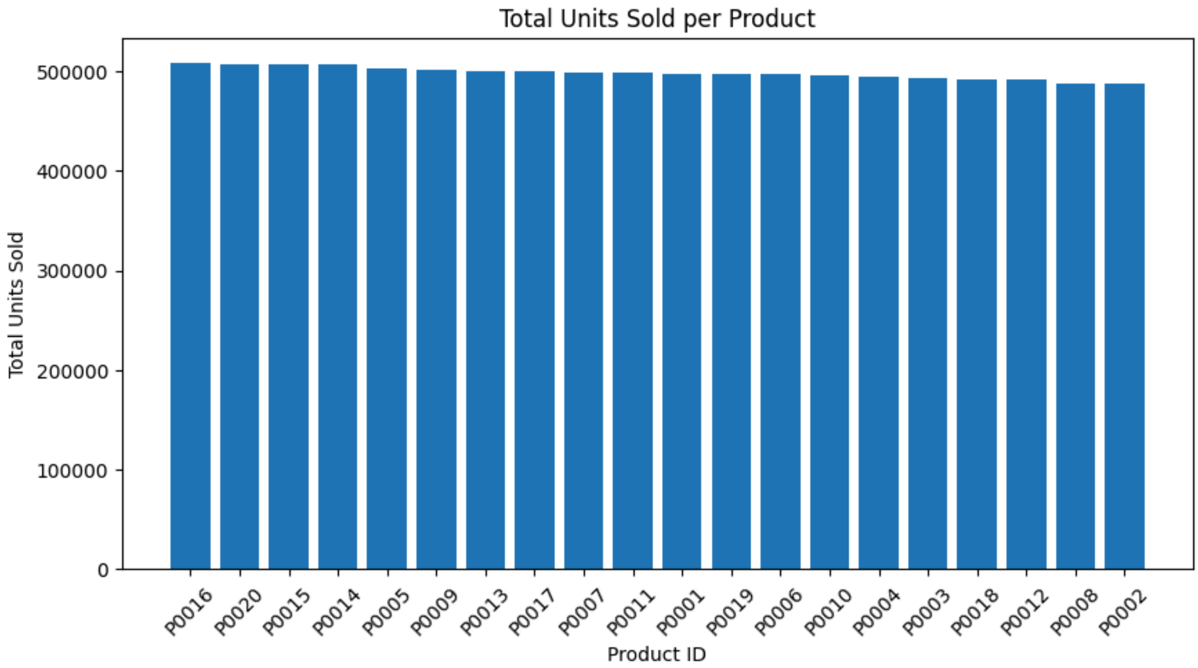
```
)

product_sales
```

Out[24]:

	Product ID	Units Sold
15	P0016	508372
19	P0020	507753
14	P0015	507298
13	P0014	507175
4	P0005	503698
8	P0009	502140
12	P0013	500976
16	P0017	500510
6	P0007	499267
10	P0011	499260
0	P0001	497952
18	P0019	497838
5	P0006	497030
9	P0010	496374
3	P0004	495403
2	P0003	493269
17	P0018	492799
11	P0012	491678
7	P0008	488536
1	P0002	487792

In [25]:

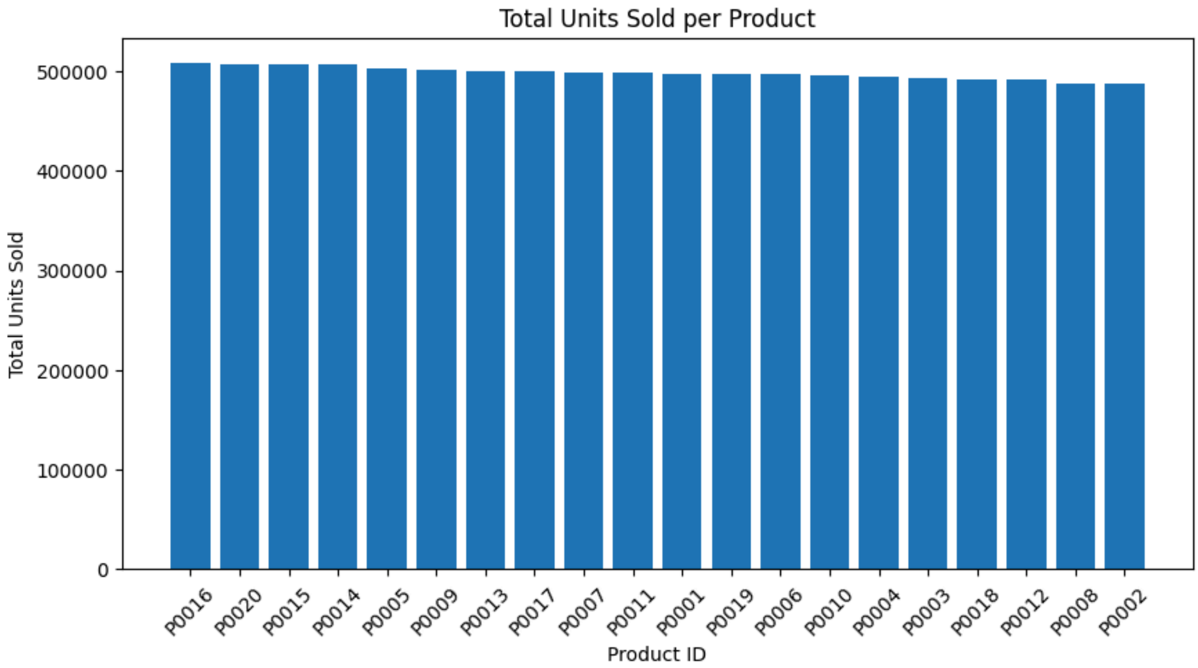
```
plt.figure(figsize=(10,5))
plt.bar(product_sales['Product ID'], product_sales['Units Sold'])
plt.xlabel('Product ID')
plt.ylabel('Total Units Sold')
plt.title('Total Units Sold per Product')
plt.xticks(rotation=45)
plt.show()
```



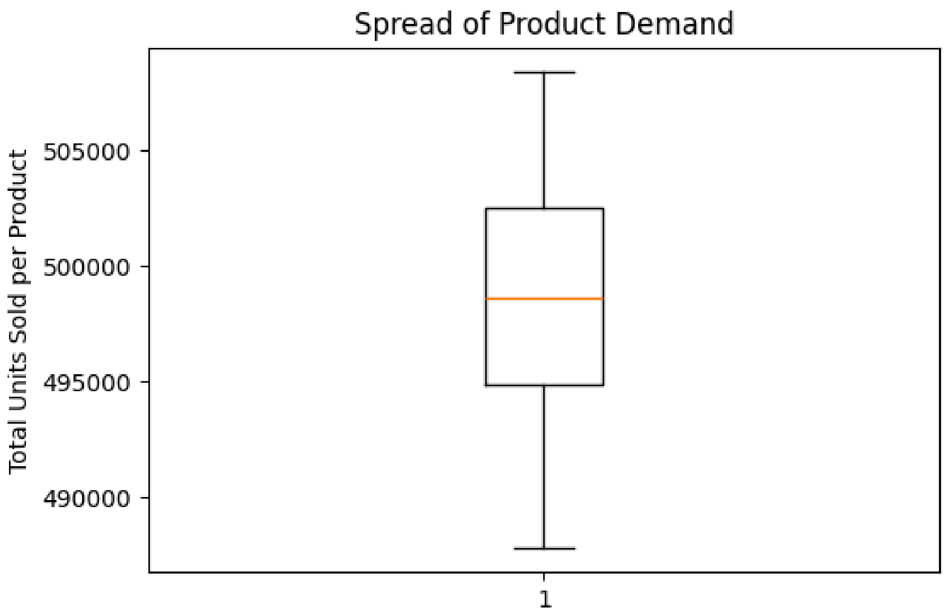
In [26]:
```
# Demand Segmentation Using Quantiles
product_sales['Demand Segment'] = pd.qcut(
    product_sales['Units Sold'],
    q=3,
    labels=['Low Demand', 'Medium Demand', 'High Demand']
)

product_sales['Demand Segment'].value_counts()
```

Out[26]:
```
Demand Segment
Low Demand       7
High Demand      7
Medium Demand    6
Name: count, dtype: int64
```

In [27]:
```
# Total units sold per product.
plt.figure(figsize=(10,5))
plt.bar(product_sales['Product ID'], product_sales['Units Sold'])
plt.xlabel('Product ID')
plt.ylabel('Total Units Sold')
plt.title('Total Units Sold per Product')
plt.xticks(rotation=45)
plt.show()
```



In [28]:
```
plt.figure(figsize=(6,4))
plt.boxplot(product_sales['Units Sold'])
plt.ylabel('Total Units Sold per Product')
plt.title('Spread of Product Demand')
plt.show()
```



Question: Does product demand vary across stores?

We want to see:

- Does the same product sell differently in different stores?
- Are some products store-specific best sellers?
- Can we apply the same inventory rule for a product across all stores?

In [29]:
```
# Units Sold per Product per Store (Store × Product Demand)
store_product_sales = (
    data
    .groupby(['Store ID', 'Product ID'])['Units Sold']
    .sum()
    .reset_index()
)
```

```
store_product_sales.head()
```

Out[29]:

	Store ID	Product ID	Units Sold
0	S001	P0001	100262
1	S001	P0002	95689
2	S001	P0003	103313
3	S001	P0004	102876
4	S001	P0005	97368

In [30]:

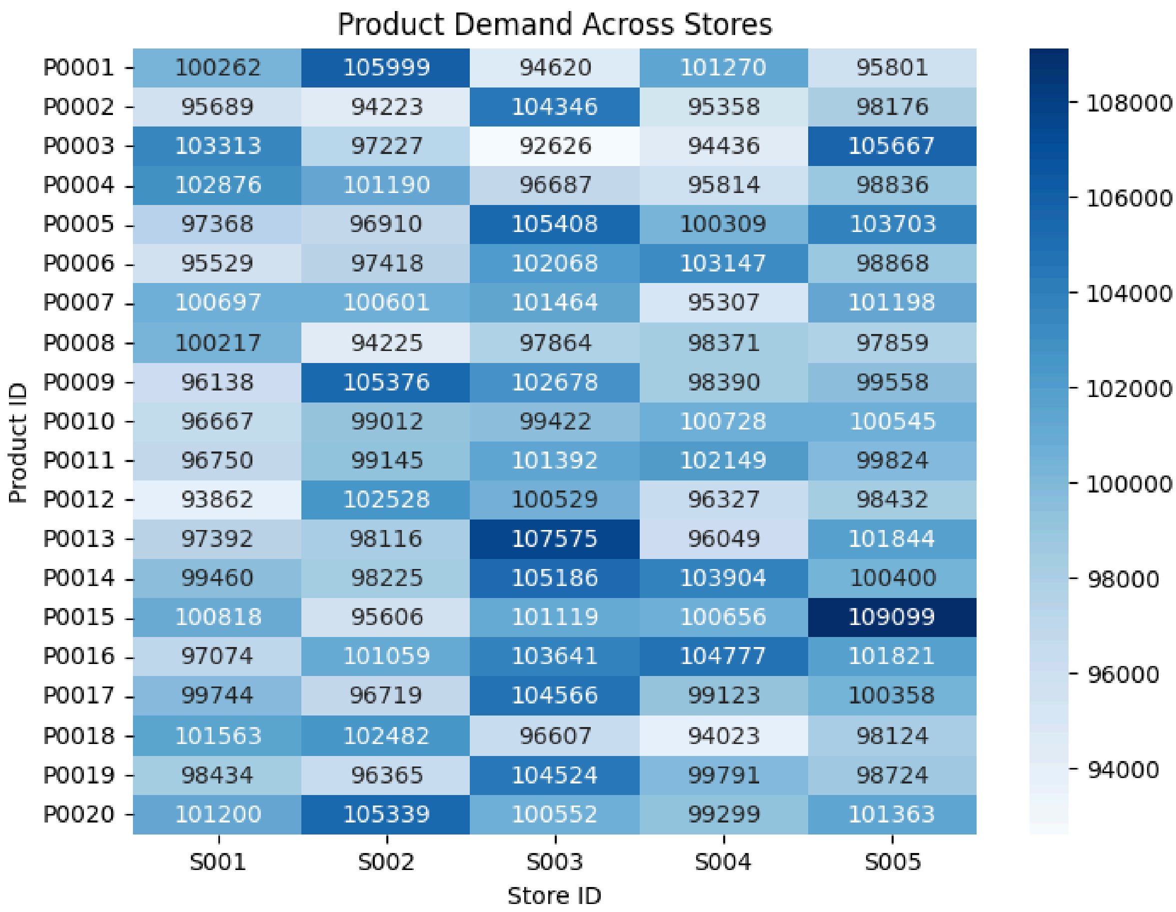
```
pivot_table = store_product_sales.pivot(
    index='Product ID',
    columns='Store ID',
    values='Units Sold'
)

pivot_table
```

Out[30]:

Store ID	S001	S002	S003	S004	S005
Product ID					
P0001	100262	105999	94620	101270	95801
P0002	95689	94223	104346	95358	98176
P0003	103313	97227	92626	94436	105667
P0004	102876	101190	96687	95814	98836
P0005	97368	96910	105408	100309	103703
P0006	95529	97418	102068	103147	98868
P0007	100697	100601	101464	95307	101198
P0008	100217	94225	97864	98371	97859
P0009	96138	105376	102678	98390	99558
P0010	96667	99012	99422	100728	100545
P0011	96750	99145	101392	102149	99824
P0012	93862	102528	100529	96327	98432
P0013	97392	98116	107575	96049	101844
P0014	99460	98225	105186	103904	100400
P0015	100818	95606	101119	100656	109099
P0016	97074	101059	103641	104777	101821
P0017	99744	96719	104566	99123	100358
P0018	101563	102482	96607	94023	98124
P0019	98434	96365	104524	99791	98724
P0020	101200	105339	100552	99299	101363

In [31]:

```
plt.figure(figsize=(8,6))
sns.heatmap(pivot_table, annot=True, fmt=".0f", cmap='Blues')
plt.title('Product Demand Across Stores')
plt.ylabel('Product ID')
plt.xlabel('Store ID')
plt.show()
```



INSIGHT: Product-level analysis across stores shows clear variation in demand for the same product across different store locations. Certain products perform significantly better in specific stores, while others show relatively consistent demand. This indicates that inventory planning and replenishment strategies should be tailored at the store–product level rather than applying uniform rules across all stores.

Question: How is demand trending over time?

We want to see:

- Is demand increasing, decreasing, or stable over time?
- Are there ups and downs?
- Is there a pattern across dates?

This helps decide:

- Whether forecasting is needed
- What kind of time-series behavior exists

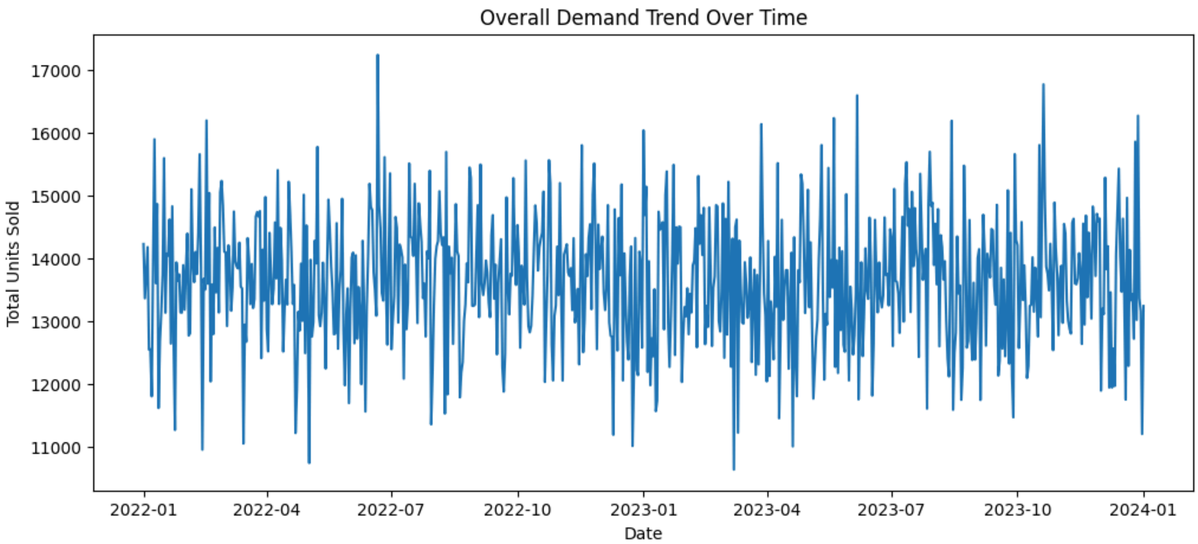
In [32]:
```python
# Aggregate Demand Over Time (Daily Total Demand)
daily_sales = (
    data
    .groupby('Date')['Units Sold']
    .sum()
    .reset_index()
)

daily_sales.head()
```

Out[32]:

	Date	Units Sold
0	2022-01-01	14230
1	2022-01-02	13367
2	2022-01-03	13781
3	2022-01-04	14181
4	2022-01-05	12550

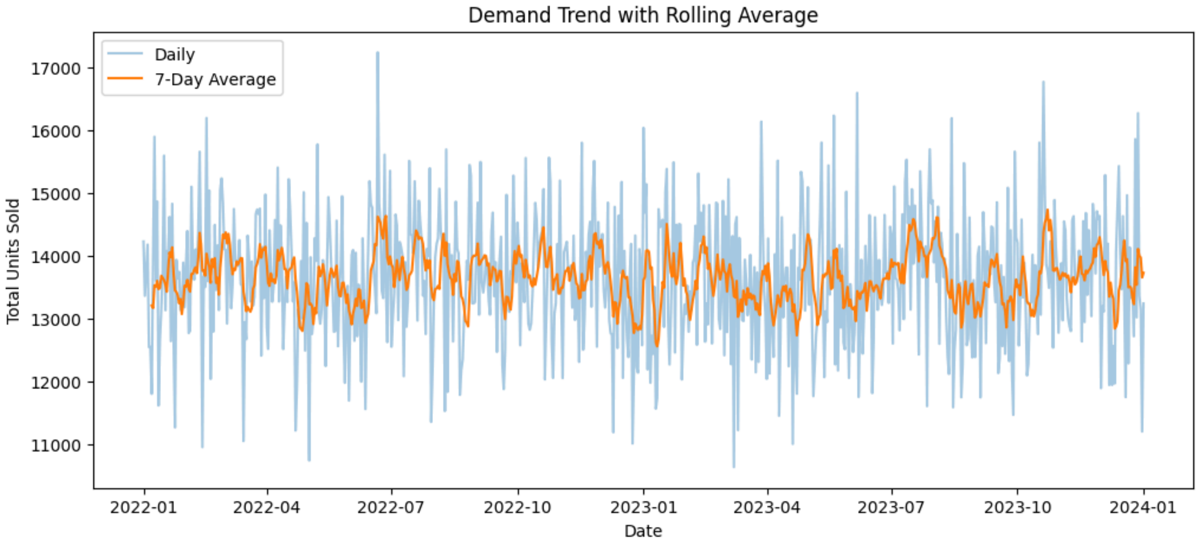
In [33]:

```
# Demand Trend Over Time
plt.figure(figsize=(12,5))
plt.plot(daily_sales['Date'], daily_sales['Units Sold'])
plt.xlabel('Date')
plt.ylabel('Total Units Sold')
plt.title('Overall Demand Trend Over Time')
plt.show()
```



In [34]:

```
# Day Rolling Average
daily_sales['rolling_7'] = daily_sales['Units Sold'].rolling(window=7).mean()

plt.figure(figsize=(12,5))
plt.plot(daily_sales['Date'], daily_sales['Units Sold'], alpha=0.4, label='Daily')
plt.plot(daily_sales['Date'], daily_sales['rolling_7'], label='7-Day Average')
plt.xlabel('Date')
plt.ylabel('Total Units Sold')
plt.title('Demand Trend with Rolling Average')
plt.legend()
plt.show()
```



Question: Are there seasonal or monthly demand patterns?

We want to know:

- Does demand repeat every month / season?
- Are some months consistently high or low?
- Is there a seasonal cycle in sales?

In [35]:
```
# Extract Time-Based Features (Extract Month & Year)
data['Month'] = data['Date'].dt.month
data['Year'] = data['Date'].dt.year
```

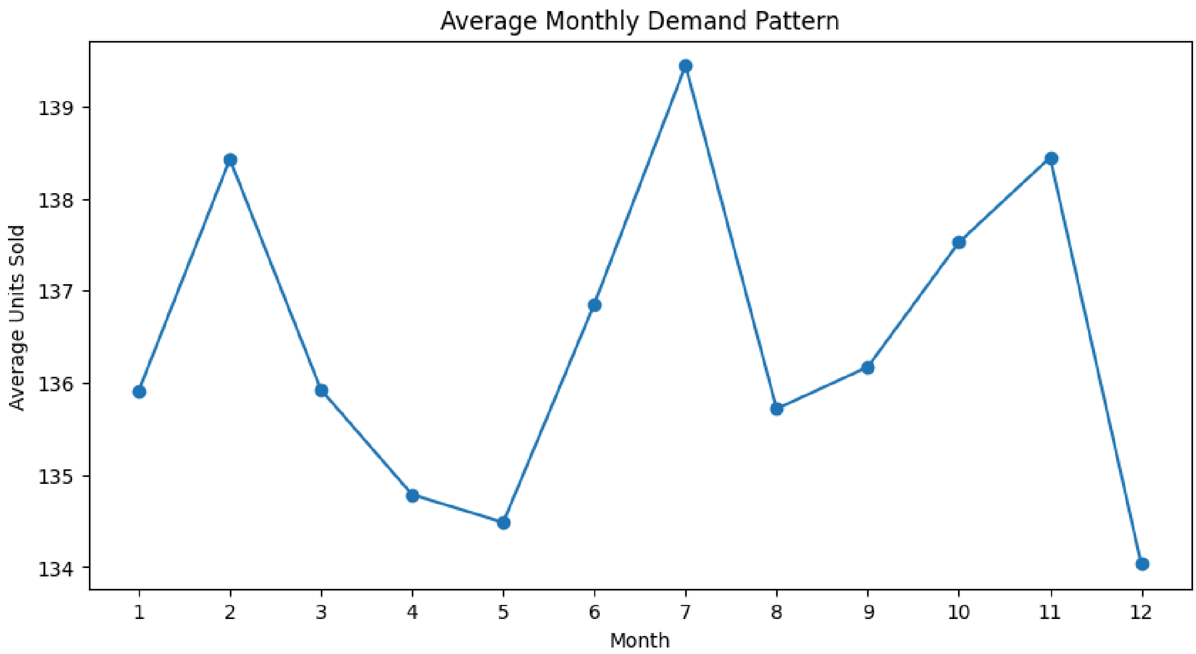
In [36]:
```
# Average Monthly Demand
monthly_sales = (
    data
    .groupby('Month')['Units Sold']
    .mean()
    .reset_index()
)

monthly_sales
```

Out[36]:

	Month	Units Sold
0	1	135.910476
1	2	138.426429
2	3	135.923387
3	4	134.784667
4	5	134.482742
5	6	136.856167
6	7	139.443065
7	8	135.716774
8	9	136.170333
9	10	137.523548
10	11	138.444333
11	12	134.033548

In [37]:
```
# Monthly Demand Pattern
plt.figure(figsize=(10,5))
```

```python
plt.plot(monthly_sales['Month'], monthly_sales['Units Sold'], marker='o')
plt.xlabel('Month')
plt.ylabel('Average Units Sold')
plt.title('Average Monthly Demand Pattern')
plt.xticks(range(1,13))
plt.show()
```

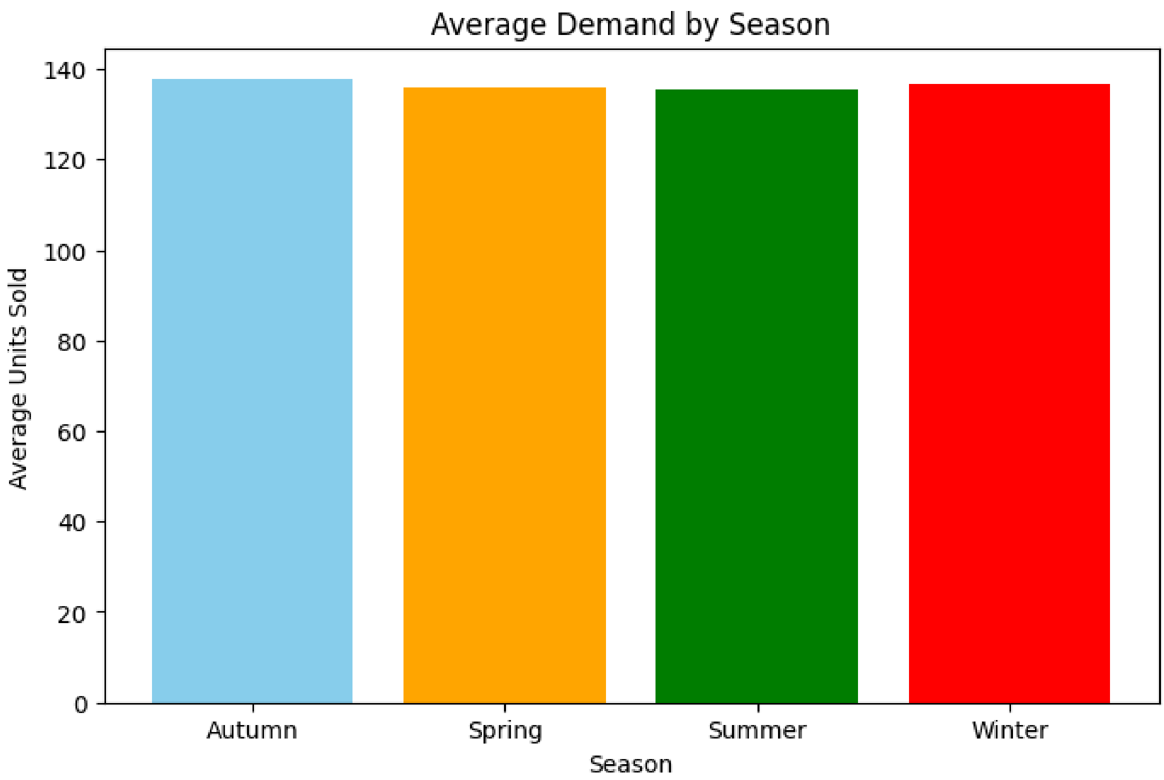


In [38]:
```python
# Seasonal Demand
seasonal_sales = (
    data
    .groupby('Seasonality')['Units Sold']
    .mean()
    .reset_index()
)

seasonal_sales
```

Out[38]:

	Seasonality	Units Sold
0	Autumn	137.774419
1	Spring	135.836436
2	Summer	135.409342
3	Winter	136.822860

In [39]:
```python
colors = ['skyblue', 'orange', 'green', 'red']

plt.figure(figsize=(8,5))
plt.bar(seasonal_sales['Seasonality'], seasonal_sales['Units Sold'],color=colors)
plt.xlabel('Season')
plt.ylabel('Average Units Sold')
plt.title('Average Demand by Season')
plt.show()
```



Question: Which products have stable demand and which are highly unpredictable?

We want to know:

- Which products sell consistently (stable)
- Which products have large ups and downs (volatile)

In [40]:
```
# Measure Demand Variability per Product
product_variability = (
    data
    .groupby('Product ID')['Units Sold']
    .agg(['mean', 'std'])
    .reset_index()
)

product_variability.head()
```

Out[40]:

	Product ID	mean	std
0	P0001	136.238577	109.336580
1	P0002	133.458824	106.387766
2	P0003	134.957319	108.289034
3	P0004	135.541176	109.218776
4	P0005	137.810670	106.876839

In [41]:
```
# Coefficient of Variation (CV)
product_variability['CV'] = (
    product_variability['std'] / product_variability['mean']
)

product_variability
```

Out[41]:

	Product ID	mean	std	CV
0	P0001	136.238577	109.336580	0.802538
1	P0002	133.458824	106.387766	0.797158
2	P0003	134.957319	108.289034	0.802395
3	P0004	135.541176	109.218776	0.805798
4	P0005	137.810670	106.876839	0.775534
5	P0006	135.986320	108.745785	0.799682
6	P0007	136.598358	110.514513	0.809047
7	P0008	133.662380	108.683677	0.813121
8	P0009	137.384405	110.538356	0.804592
9	P0010	135.806840	109.711294	0.807848
10	P0011	136.596443	108.187652	0.792024
11	P0012	134.522025	107.748806	0.800975
12	P0013	137.065937	108.108876	0.788736
13	P0014	138.761970	110.130563	0.793665
14	P0015	138.795622	109.907903	0.791869
15	P0016	139.089466	109.373335	0.786352
16	P0017	136.938440	107.890733	0.787878
17	P0018	134.828728	108.566359	0.805217
18	P0019	136.207387	109.451017	0.803562
19	P0020	138.920109	110.191487	0.793200

In [42]:

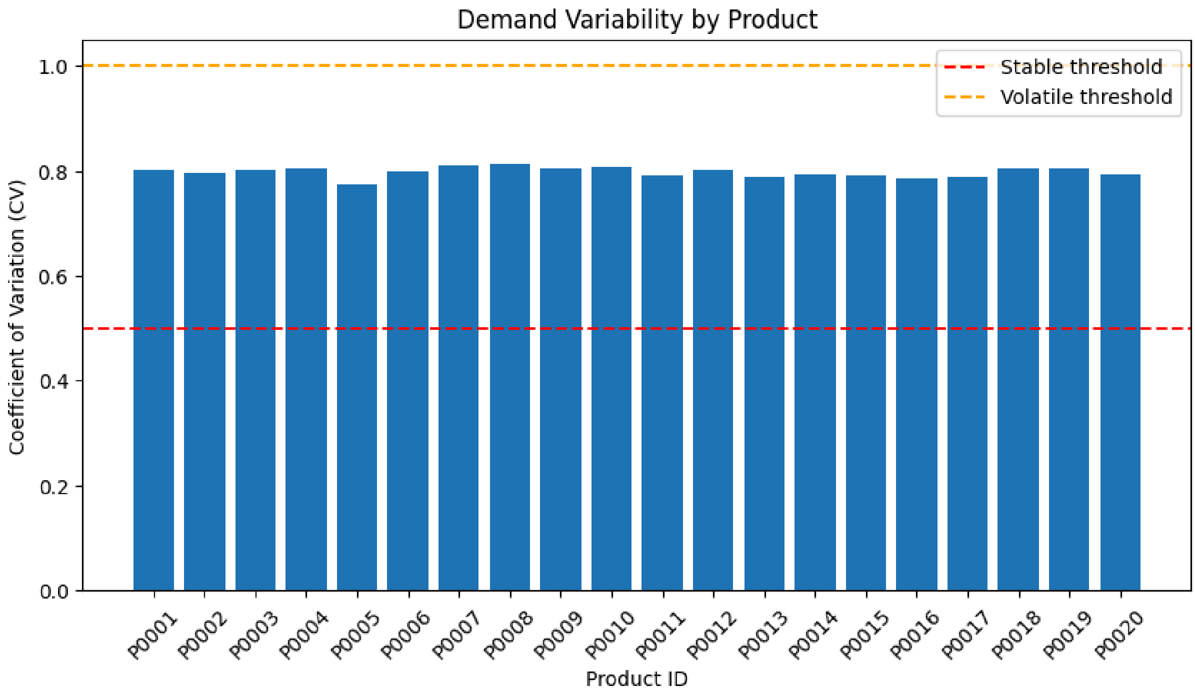
```python
# Categorize Products by Stability
def demand_stability(cv):
    if cv < 0.5:
        return 'Stable'
    elif cv < 1:
        return 'Moderate'
    else:
        return 'Highly Volatile'

product_variability['Demand Stability'] = product_variability['CV'].apply(demand_stability)

product_variability['Demand Stability'].value_counts()
```

Out[42]:

```
Demand Stability
Moderate    20
Name: count, dtype: int64
```

In [43]:

```python
# Product Variability
plt.figure(figsize=(10,5))
plt.bar(product_variability['Product ID'], product_variability['CV'])
plt.axhline(0.5, color='red', linestyle='--', label='Stable threshold')
plt.axhline(1.0, color='orange', linestyle='--', label='Volatile threshold')
plt.xlabel('Product ID')
plt.ylabel('Coefficient of Variation (CV)')
plt.title('Demand Variability by Product')
plt.xticks(rotation=45)
plt.legend()
plt.show()
```



Question: How do promotions and discounts affect demand?

We want to know:

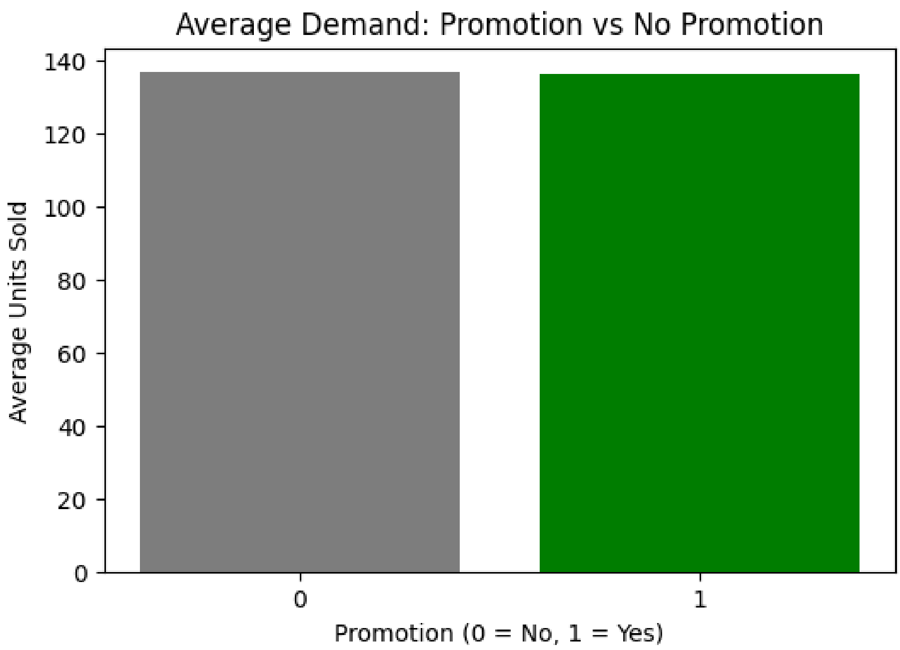
- Do products sell more during promotions?
- Does discounting increase demand?
- Does promotion also increase demand variability?

In [44]:
```
# Average Demand With vs Without Promotion
promo_effect = (
    data
    .groupby('Holiday/Promotion')['Units Sold']
    .mean()
    .reset_index()
)

promo_effect
```

Out[44]:

	Holiday/Promotion	Units Sold
0	0	136.500966
1	1	136.415674

In [45]:
```
# Promotion vs Non-Promotion
plt.figure(figsize=(6,4))
plt.bar(
    promo_effect['Holiday/Promotion'].astype(str),
    promo_effect['Units Sold'],
    color=['gray', 'green']
)
plt.xlabel('Promotion (0 = No, 1 = Yes)')
plt.ylabel('Average Units Sold')
plt.title('Average Demand: Promotion vs No Promotion')
plt.show()
```



In [46]:

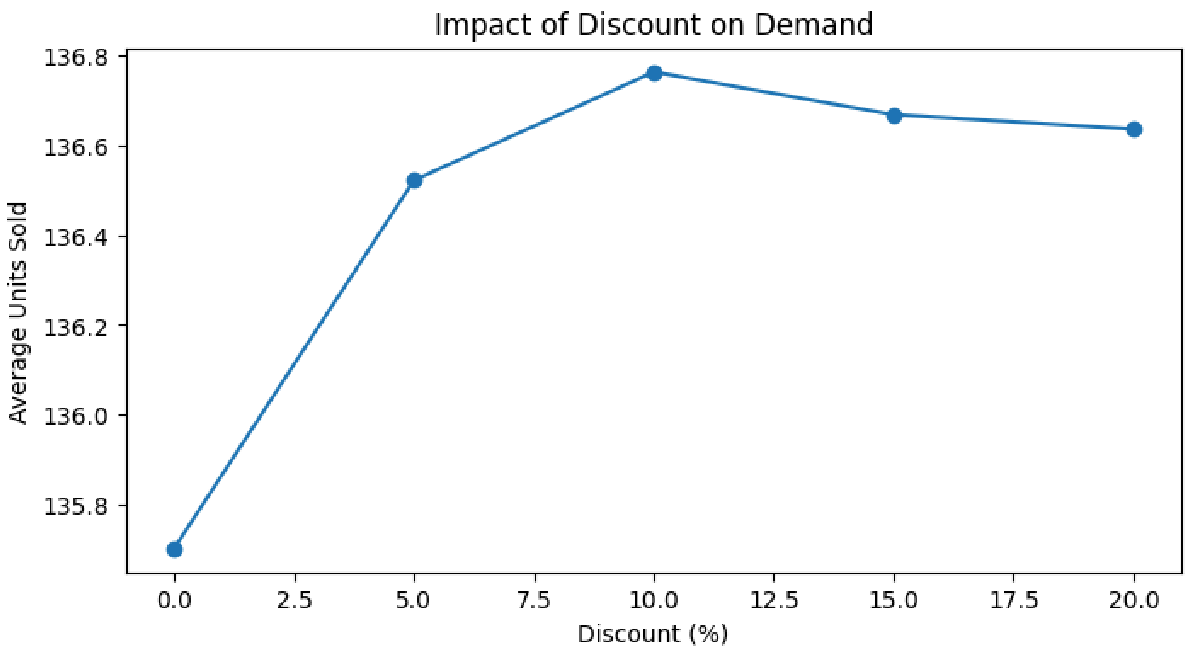
```
# Discount vs Average Demand
discount_effect = (
    data
    .groupby('Discount')['Units Sold']
    .mean()
    .reset_index()
    .sort_values('Discount')
)

discount_effect
```

Out[46]:

	Discount	Units Sold
0	0	135.701951
1	5	136.523062
2	10	136.764475
3	15	136.669379
4	20	136.637309

In [47]:

```
# Discount vs Demand
plt.figure(figsize=(8,4))
plt.plot(discount_effect['Discount'], discount_effect['Units Sold'], marker='o')
plt.xlabel('Discount (%)')
plt.ylabel('Average Units Sold')
plt.title('Impact of Discount on Demand')
plt.show()
```

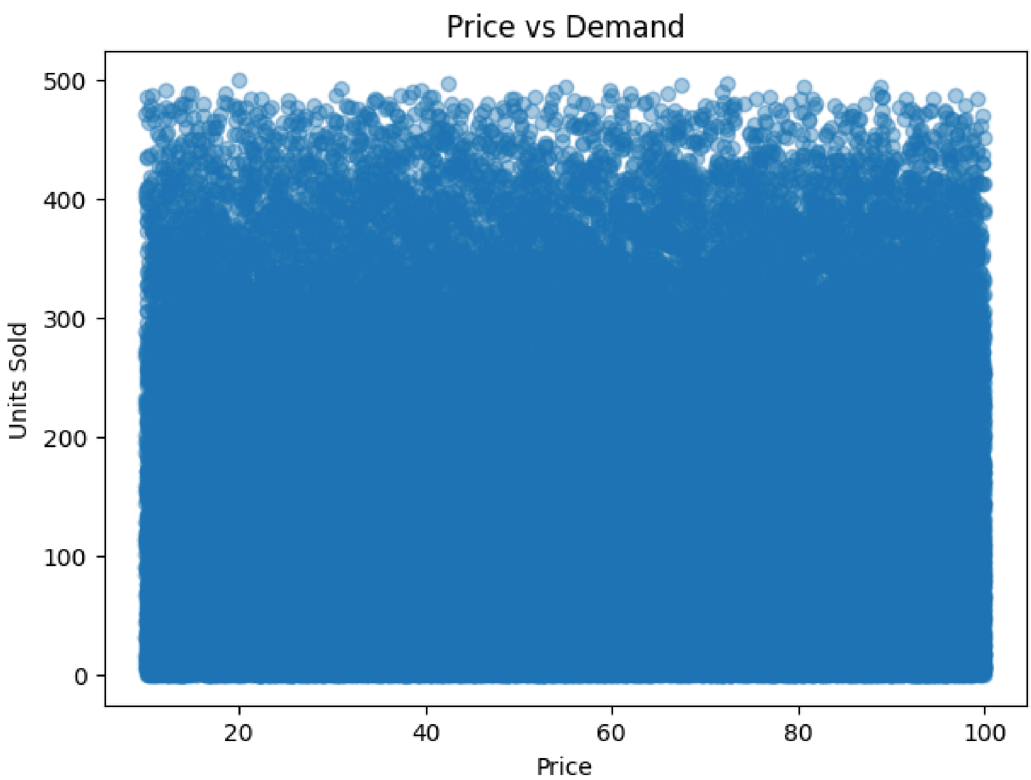


Question: Does price change significantly impact demand?

We want to know:

- When price increases, does demand drop?
- When price decreases, does demand increase?
- Is price an important demand driver?

In [48]:
```
# Relationship Between Price and Demand (Price vs Units Sold)
plt.figure(figsize=(7,5))
plt.scatter(data['Price'], data['Units Sold'], alpha=0.4)
plt.xlabel('Price')
plt.ylabel('Units Sold')
plt.title('Price vs Demand')
plt.show()
```



In [49]:
```python
# Quantify Price-Demand Relationship
price_corr = data['Price'].corr(data['Units Sold'])
price_corr
```

Out[49]: np.float64(0.0011228235041894532)

In [50]:
```python
comp_price_corr = data['Competitor Pricing'].corr(data['Units Sold'])
comp_price_corr
```

Out[50]: np.float64(0.0012955714894700326)

STEP 5: FEATURE ENGINEERING

In [51]:
```python
# Define Target & Sort Data (Target variable = Units Sold)
# Ensure correct sorting
data = data.sort_values(['Store ID', 'Product ID', 'Date']).reset_index(drop=True)
```

In [52]:
```python
# Create Time-Based Features
data['Day'] = data['Date'].dt.day
data['Week'] = data['Date'].dt.isocalendar().week.astype(int)
data['Month'] = data['Date'].dt.month
data['Year'] = data['Date'].dt.year
data['DayOfWeek'] = data['Date'].dt.dayofweek
```

In [53]:
```python
# Create Lag Features (Lag features = past demand values)
data['lag_1'] = data.groupby(['Store ID', 'Product ID'])['Units Sold'].shift(1)
data['lag_7'] = data.groupby(['Store ID', 'Product ID'])['Units Sold'].shift(7)
data['lag_14'] = data.groupby(['Store ID', 'Product ID'])['Units Sold'].shift(14)
```

In [54]:
```python
# Create Rolling Statistics
data['rolling_7_mean'] = (
    data.groupby(['Store ID', 'Product ID'])['Units Sold']
    .shift(1)
    .rolling(window=7)
    .mean()
)

data['rolling_14_mean'] = (
    data.groupby(['Store ID', 'Product ID'])['Units Sold']
    .shift(1)
    .rolling(window=14)
    .mean()
)
```

In [55]:
```python
# Encode Promotion & Discount Signals
data['is_promo'] = (data['Discount'] > 0).astype(int)
```

In [56]:
```python
# Handle Categorical Variables
from sklearn.preprocessing import LabelEncoder

cat_cols = ['Store ID', 'Product ID', 'Category', 'Region', 'Seasonality', 'Weather Condition']

le = LabelEncoder()
for col in cat_cols:
    data[col] = le.fit_transform(data[col])
```

In [57]:
```python
# Handle Missing Values created by LAG
data_model = data.dropna().reset_index(drop=True)
data_model.head(40)
```

Out[57]:

	Date	Store ID	Product ID	Category	Region	Inventory Level	Units Sold	Units Ordered	Demand Forecast	Price	Discount	Wea Cond
0	2022-01-15	0	0	3	1	290	176	94	170.06	98.04	10	
1	2022-01-16	0	0	3	0	104	97	189	114.65	11.39	15	
2	2022-01-17	0	0	3	0	134	69	95	73.59	95.27	0	
3	2022-01-18	0	0	1	1	339	123	157	134.00	90.70	15	
4	2022-01-19	0	0	3	3	431	211	139	228.92	97.97	10	
5	2022-01-20	0	0	2	1	154	79	113	90.49	10.38	0	
6	2022-01-21	0	0	0	2	430	390	195	401.10	13.10	15	
7	2022-01-22	0	0	1	1	450	191	197	181.42	87.64	0	
8	2022-01-23	0	0	2	3	118	79	155	69.99	22.65	15	
9	2022-01-24	0	0	4	2	115	9	132	17.87	76.71	15	
10	2022-01-25	0	0	2	0	348	189	122	185.83	18.35	15	
11	2022-01-26	0	0	0	2	200	40	101	50.62	59.48	5	
12	2022-01-27	0	0	2	2	410	164	116	180.45	93.34	0	
13	2022-01-28	0	0	2	0	158	97	82	115.96	53.25	20	
14	2022-01-29	0	0	2	2	154	47	124	48.79	88.06	15	
15	2022-01-30	0	0	2	3	498	130	63	148.26	68.76	10	
16	2022-01-31	0	0	0	3	410	200	152	212.24	70.78	0	
17	2022-02-01	0	0	3	0	419	279	84	297.26	34.49	0	
18	2022-02-02	0	0	1	2	415	38	149	53.13	52.49	5	
19	2022-02-03	0	0	0	2	345	71	186	84.02	27.71	20	
20	2022-02-04	0	0	2	2	121	25	25	37.72	16.84	15	
21	2022-02-05	0	0	3	2	458	350	182	362.43	51.44	15	
22	2022-02-06	0	0	1	1	167	69	92	81.06	68.85	5	
23	2022-02-07	0	0	1	2	244	62	105	73.41	11.34	5	

	Date	Store ID	Product ID	Category	Region	Inventory Level	Units Sold	Units Ordered	Demand Forecast	Price	Discount	Wea Cond
24	2022-02-08	0	0	4	3	140	12	48	15.60	67.64	10	
25	2022-02-09	0	0	3	2	209	16	143	22.37	90.55	0	
26	2022-02-10	0	0	0	2	311	237	102	229.08	89.58	5	
27	2022-02-11	0	0	1	0	216	203	34	198.49	19.93	0	
28	2022-02-12	0	0	4	2	358	29	54	48.16	64.40	15	
29	2022-02-13	0	0	0	1	229	149	161	143.55	85.19	10	
30	2022-02-14	0	0	0	1	429	259	156	256.33	97.41	10	
31	2022-02-15	0	0	2	1	126	11	32	115.57	51.27	5	
32	2022-02-16	0	0	1	3	373	373	91	372.09	65.22	10	
33	2022-02-17	0	0	2	3	82	31	146	27.62	49.34	5	
34	2022-02-18	0	0	1	2	115	27	33	39.79	20.21	15	
35	2022-02-19	0	0	2	1	207	27	200	32.44	75.60	0	
36	2022-02-20	0	0	4	2	284	166	189	175.44	73.76	10	
37	2022-02-21	0	0	0	0	337	337	136	346.41	96.55	10	
38	2022-02-22	0	0	2	0	467	191	135	194.70	12.81	15	
39	2022-02-23	0	0	3	3	152	129	88	134.64	27.67	15	

In [58]:
```
data[['Store ID', 'Product ID']].drop_duplicates().head(10)
```

Out[58]:

	Store ID	Product ID
0	0	0
731	0	1
1462	0	2
2193	0	3
2924	0	4
3655	0	5
4386	0	6
5117	0	7
5848	0	8
6579	0	9

In [59]:

```
data.shape
```

Out[59]:

```
(73100, 26)
```

STEP 7 : MODELLING AND EVALUATION

In [ ]:

In [60]:

```
# Create Time-Based Split
# Get the cutoff date
cutoff_date = data_model['Date'].quantile(0.8)

# Split the data
train_data = data_model[data_model['Date'] <= cutoff_date]
test_data  = data_model[data_model['Date'] > cutoff_date]

print(train_data.shape, test_data.shape)
```

```
(57400, 26) (14300, 26)
```

In [61]:

```
# Define Features (X) and Target (y)
target = 'Units Sold'

features = [
    'lag_1', 'lag_7', 'lag_14',
    'rolling_7_mean', 'rolling_14_mean',
    'Day', 'Week', 'Month', 'DayOfWeek',
    'Price', 'Discount', 'is_promo',
    'Store ID', 'Product ID', 'Category',
    'Region', 'Seasonality', 'Weather Condition'
]

X_train = train_data[features]
y_train = train_data[target]

X_test = test_data[features]
y_test = test_data[target]
```

Baseline Model

Baseline Rule: **Tomorrow's demand = yesterday's demand**

In [62]:

```
baseline_pred = X_test['lag_1']
```

In [63]:

```
# Evaluate Baseline Model
from sklearn.metrics import mean_absolute_error
```

```
baseline_mae = mean_absolute_error(y_test, baseline_pred)
baseline_mae
```

Out[63]: 119.71874125874126

ML Forecasting Model

In [64]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [65]:
```
# initialize the model
rf_model = RandomForestRegressor(
    n_estimators=100,
    max_depth=15,
    random_state=42,
    n_jobs=-1
)
```

In [66]:
```
# Train the Model
rf_model.fit(X_train, y_train)
```

Out[66]:
```
▾ RandomForestRegressor ⓘ ?

▸ Parameters
```

In [67]:
```
# make prediction on test data
rf_pred = rf_model.predict(X_test)
```

In [68]:
```
# evaluate the model
rf_mae = mean_absolute_error(y_test, rf_pred)
rf_mae
```

Out[68]: 89.06049547651007

In [69]:
```
# compare with baseline model
print("Baseline MAE:", baseline_mae)
print("Random Forest MAE:", rf_mae)
```

```
Baseline MAE: 119.71874125874126
Random Forest MAE: 89.06049547651007
```

In [70]:
```
data[['Store ID', 'Product ID']].nunique()
```

Out[70]:
```
Store ID       5
Product ID    20
dtype: int64
```

Step 8: INVENTORY OPTIMIZATION

Inventory optimization means deciding:

- How much stock to keep
- When to reorder
- How to avoid stockouts and overstock

Key Inventory Concepts (You MUST know these) 1️⃣ Average Demand (D)

Expected daily demand → Use forecasted demand or historical mean

2️⃣ Demand Variability (σ)

How much demand fluctuates → Use standard deviation

3️⃣ Lead Time (L)

Time taken to replenish stock → Assume constant (e.g., 7 days) if not given

4️⃣ Service Level (Z)

Desired probability of no stockout

Common values:

Service Level Z-Score 90% 1.28 95% 1.65 99% 2.33

Decide Optimization Level (Important Decision)

We will optimize at:

👉 Store × Product level

Why?

Demand varies by product

Demand varies by store

This is the most realistic level

In [71]:
```
# Calculate Demand Statistics per Store-Product
inventory_stats = (
    data_model
    .groupby(['Store ID', 'Product ID'])['Units Sold']
    .agg(['mean', 'std'])
    .reset_index()
)

inventory_stats.head()
```

Out[71]:

	Store ID	Product ID	mean	std
0	0	0	137.984658	108.496137
1	0	1	130.492329	104.566204
2	0	2	142.119944	106.972401
3	0	3	141.002789	113.426557
4	0	4	132.859135	107.635395

In [72]:
```
# Assume Lead Time & Service Level
lead_time = 7          # days
service_level = 1.65   # 95% service level
```

In [73]:
```
# Calculate Safety Stock (Safety Stock = Z × σ × √L)
inventory_stats['Safety Stock'] = (
    service_level * inventory_stats['std'] * (lead_time ** 0.5)
)
```

In [74]:
```
# Calculate Reorder Point (ROP)  (ROP = (Average Demand × Lead Time) + Safety Stock)
inventory_stats['Reorder Point'] = (
    inventory_stats['mean'] * lead_time + inventory_stats['Safety Stock']
)
```

In [75]:
```
# Compare with Current Inventory
current_inventory = (
    data_model
    .groupby(['Store ID', 'Product ID'])['Inventory Level']
    .mean()
    .reset_index()
)
```

```python
inventory_decision = inventory_stats.merge(
    current_inventory,
    on=['Store ID', 'Product ID'],
    how='left'
)

inventory_decision['Reorder Required'] = (
    inventory_decision['Inventory Level'] < inventory_decision['Reorder Point']
)
```

In [76]:
```python
current_inventory = (
    data_model
    .sort_values('Date')
    .groupby(['Store ID', 'Product ID'])
    .tail(1)[['Store ID', 'Product ID', 'Inventory Level']]
    .reset_index(drop=True)
)
```

In [77]:
```python
inventory_plan = inventory_stats.merge(
    current_inventory,
    on=['Store ID', 'Product ID'],
    how='left'
)
```

In [78]:
```python
inventory_plan['Reorder Required'] = (
    inventory_plan['Inventory Level'] < inventory_plan['Reorder Point']
)
```

In [79]:
```python
inventory_plan['Order Quantity'] = (
    inventory_plan['Reorder Point'] +
    inventory_plan['Safety Stock'] -
    inventory_plan['Inventory Level']
)

# If no reorder required → order quantity = 0
inventory_plan.loc[
    inventory_plan['Reorder Required'] == False,
    'Order Quantity'
] = 0
```

In [80]:
```python
inventory_plan[[
    'Store ID',
    'Product ID',
    'Inventory Level',
    'Safety Stock',
    'Reorder Point',
    'Reorder Required',
    'Order Quantity'
]].head(100)
```

Out[80]:

	Store ID	Product ID	Inventory Level	Safety Stock	Reorder Point	Reorder Required	Order Quantity
0	0	0	223	473.638765	1439.531373	True	1690.170139
1	0	1	217	456.482684	1369.928988	True	1609.411672
2	0	2	69	466.986910	1461.826519	True	1859.813429
3	0	3	338	495.162461	1482.181987	True	1639.344448
4	0	4	471	469.881203	1399.895150	True	1398.776352
...	...	...	...	...	...	...	...
95	4	15	96	484.566251	1458.845191	True	1847.411442
96	4	16	313	472.332962	1427.027523	True	1586.360485
97	4	17	278	481.536226	1423.480438	True	1627.016664
98	4	18	374	474.947658	1418.385594	True	1519.333252
99	4	19	117	487.936409	1461.678389	True	1832.614798

100 rows × 7 columns