Goal of the project

supermarket chain wants to set dynamic prices for products based on demand, seasonality, and competitor pricing to maximize profit.

Create a regression model to predict optimal product price based on:

1. Past sales trends (daily/weekly)

2. Seasonal factors (festivals, weekends)

3. Competitor price data

4. Customer segments (premium vs budget shoppers)

5. Use time series forecasting + machine learning hybrid approach.

6. Integrate feature engineering (like lag features, moving averages).

Expected Output:

1. A model that predicts the best selling price for next 30 days (visualize forecast vs actual).

2. Show 10%+ increase in simulated revenue when applying your predicted prices vs static pricing.

Data Importing

In [1]:
```
import numpy as np
import pandas as pd

or_data=pd.read_csv("Documents/My_projetcs/project_2/retail_store_inventory.csv")
or_data.head()
```

Out[1]:

	Date	Store ID	Product ID	Category	Region	Inventory Level	Units Sold	Units Ordered	Demand Forecast	Price	Discount	Weather Condition	Holiday/Promotion	Co
0	2022-01-01	S001	P0001	Groceries	North	231	127	55	135.47	33.50	20	Rainy	0	
1	2022-01-01	S001	P0002	Toys	South	204	150	66	144.04	63.01	20	Sunny	0	
2	2022-01-01	S001	P0003	Toys	West	102	65	51	74.02	27.99	10	Sunny	1	
3	2022-01-01	S001	P0004	Toys	North	469	61	164	62.18	32.72	10	Cloudy	1	
4	2022-01-01	S001	P0005	Electronics	East	166	14	135	9.26	73.64	0	Sunny	0	

In [2]:
```
data1=or_data.copy()
```

In [3]:
```
data1.head()
```

Out[3]:

	Date	Store ID	Product ID	Category	Region	Inventory Level	Units Sold	Units Ordered	Demand Forecast	Price	Discount	Weather Condition	Holiday/Promotion	Co
0	2022-01-01	S001	P0001	Groceries	North	231	127	55	135.47	33.50	20	Rainy	0	
1	2022-01-01	S001	P0002	Toys	South	204	150	66	144.04	63.01	20	Sunny	0	
2	2022-01-01	S001	P0003	Toys	West	102	65	51	74.02	27.99	10	Sunny	1	
3	2022-01-01	S001	P0004	Toys	North	469	61	164	62.18	32.72	10	Cloudy	1	
4	2022-01-01	S001	P0005	Electronics	East	166	14	135	9.26	73.64	0	Sunny	0	

Data Cleaning

In [4]:

```
data1.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 [5]:
```
data1['Date']=pd.to_datetime(data1['Date'])

data1.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  datetime64[ns]
 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: datetime64[ns](1), float64(3), int64(5), object(6)
memory usage: 8.4+ MB
```

In [6]:
```
data1.isnull().sum()
```

Out[6]:
```
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 [7]:
```
data1.columns[data1.isna().all()]
```

Out[7]:
```
Index([], dtype='object')
```

In [8]:
```
data1.columns[(data1=='').all()]
```

Out[8]:
```
Index([], dtype='object')
```

In [9]:
```
data1.describe(include='all')
```

Out[9]:

	Date	Store ID	Product ID	Category	Region	Inventory Level	Units Sold	Units Ordered	Demand Forecast	Price
count	73100	73100	73100	73100	73100	73100.000000	73100.000000	73100.000000	73100.000000	73100.000000
unique	NaN	5	20	5	4	NaN	NaN	NaN	NaN	NaN
top	NaN	S001	P0001	Furniture	East	NaN	NaN	NaN	NaN	NaN
freq	NaN	14620	3655	14699	18349	NaN	NaN	NaN	NaN	NaN
mean	2022-12-31 23:59:59.999999744	NaN	NaN	NaN	NaN	274.469877	136.464870	110.004473	141.494720	55.135108
min	2022-01-01 00:00:00	NaN	NaN	NaN	NaN	50.000000	0.000000	20.000000	-9.990000	10.000000
25%	2022-07-02 00:00:00	NaN	NaN	NaN	NaN	162.000000	49.000000	65.000000	53.670000	32.650000
50%	2023-01-01 00:00:00	NaN	NaN	NaN	NaN	273.000000	107.000000	110.000000	113.015000	55.050000
75%	2023-07-03 00:00:00	NaN	NaN	NaN	NaN	387.000000	203.000000	155.000000	208.052500	77.860000
max	2024-01-01 00:00:00	NaN	NaN	NaN	NaN	500.000000	499.000000	200.000000	518.550000	100.000000
std	NaN	NaN	NaN	NaN	NaN	129.949514	108.919406	52.277448	109.254076	26.021945

In [10]:
```
data1.duplicated().sum()
```

Out[10]: 0

In [11]:
```
data1.nunique()
```

Out[11]:
```
Date                   731
Store ID                 5
Product ID              20
Category                 5
Region                   4
Inventory Level        451
Units Sold             498
Units Ordered          181
Demand Forecast      31608
Price                 8999
Discount                 5
Weather Condition        4
Holiday/Promotion        2
Competitor Pricing    9751
Seasonality              4
dtype: int64
```

In [12]:
```
data1.shape
```

Out[12]:
```
(73100, 15)
```

In [13]:
```
data1=data1.drop(['Store ID','Product ID'],axis=1)
```

In [14]:
```
data1.shape
```

Out[14]:
```
(73100, 13)
```

In [15]:
```
data1['Day']=data1['Date'].dt.day
data1['week']=data1['Date'].dt.isocalendar().week
data1['weekends']=data1['Date'].dt.dayofweek >=5
data1['Month']=data1['Date'].dt.month
data1['Year']=data1['Date'].dt.year

import holidays

indian_holidays=holidays.India(years=[2022,2023,2024])

data1['Festival']=data1['Date'].isin(indian_holidays)
```

```
data1.head()
```

Out[15]:

	Date	Category	Region	Inventory Level	Units Sold	Units Ordered	Demand Forecast	Price	Discount	Weather Condition	Holiday/Promotion	Competitor Pricing	Seasor
0	2022-01-01	Groceries	North	231	127	55	135.47	33.50	20	Rainy	0	29.69	Au
1	2022-01-01	Toys	South	204	150	66	144.04	63.01	20	Sunny	0	66.16	Au
2	2022-01-01	Toys	West	102	65	51	74.02	27.99	10	Sunny	1	31.32	Sur
3	2022-01-01	Toys	North	469	61	164	62.18	32.72	10	Cloudy	1	34.74	Au
4	2022-01-01	Electronics	East	166	14	135	9.26	73.64	0	Sunny	0	68.95	Sur

In [16]:

```
data1['Sales']=data1['Units Sold']*data1['Price']*(1-data1['Discount']/100)
data1.head()
```

Out[16]:

	Date	Category	Region	Inventory Level	Units Sold	Units Ordered	Demand Forecast	Price	Discount	Weather Condition	Holiday/Promotion	Competitor Pricing	Seasor
0	2022-01-01	Groceries	North	231	127	55	135.47	33.50	20	Rainy	0	29.69	Au
1	2022-01-01	Toys	South	204	150	66	144.04	63.01	20	Sunny	0	66.16	Au
2	2022-01-01	Toys	West	102	65	51	74.02	27.99	10	Sunny	1	31.32	Sur
3	2022-01-01	Toys	North	469	61	164	62.18	32.72	10	Cloudy	1	34.74	Au
4	2022-01-01	Electronics	East	166	14	135	9.26	73.64	0	Sunny	0	68.95	Sur

EDA

In [17]:

```
data1.head()
```

Out[17]:

	Date	Category	Region	Inventory Level	Units Sold	Units Ordered	Demand Forecast	Price	Discount	Weather Condition	Holiday/Promotion	Competitor Pricing	Seasor
0	2022-01-01	Groceries	North	231	127	55	135.47	33.50	20	Rainy	0	29.69	Au
1	2022-01-01	Toys	South	204	150	66	144.04	63.01	20	Sunny	0	66.16	Au
2	2022-01-01	Toys	West	102	65	51	74.02	27.99	10	Sunny	1	31.32	Sur
3	2022-01-01	Toys	North	469	61	164	62.18	32.72	10	Cloudy	1	34.74	Au
4	2022-01-01	Electronics	East	166	14	135	9.26	73.64	0	Sunny	0	68.95	Sur

In [18]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import calendar
%matplotlib inline


# Monthly Sales
Monthly_sales=data1.groupby(['Year','Month'])['Sales'].sum().reset_index()
Monthly_sales.head()
```

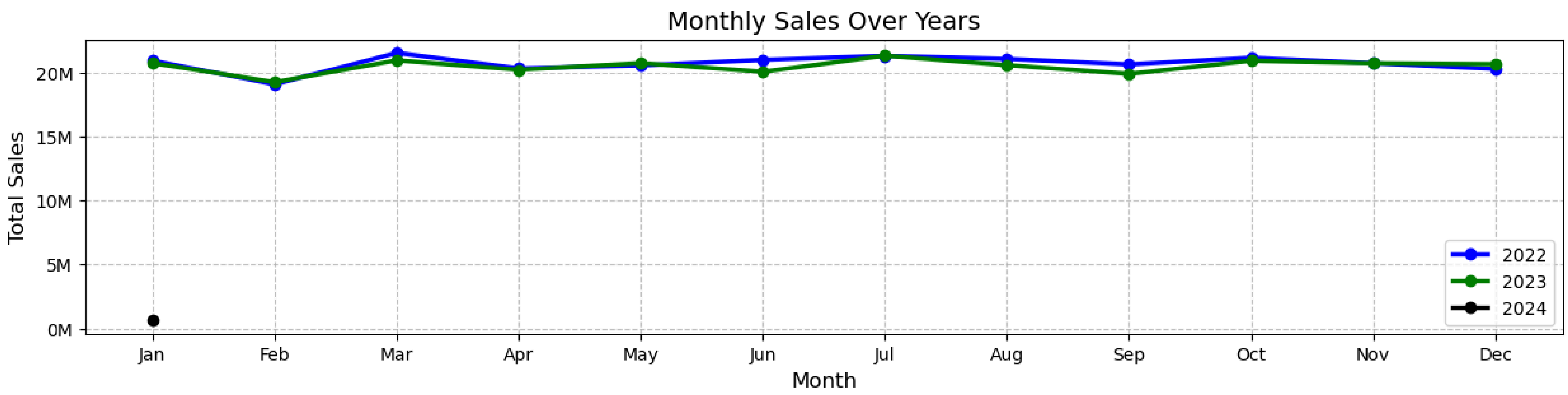
Out[18]:

	Year	Month	Sales
0	2022	1	2.091953e+07
1	2022	2	1.906103e+07
2	2022	3	2.152268e+07
3	2022	4	2.031440e+07
4	2022	5	2.054256e+07

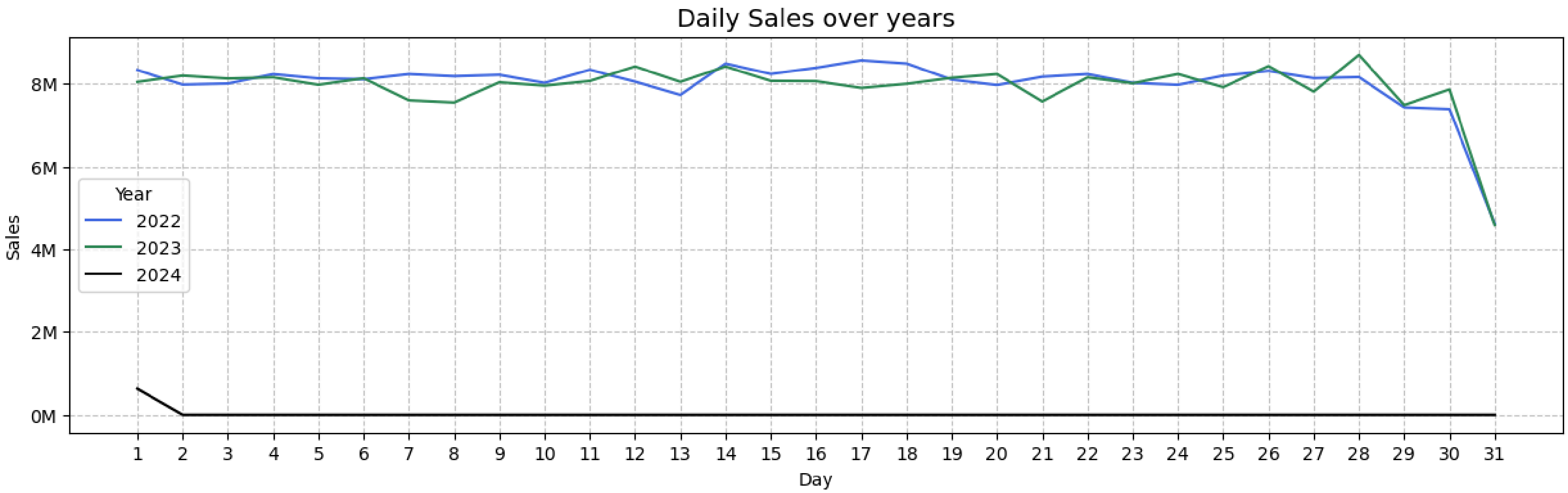
In [19]:

```python
from matplotlib.ticker import FuncFormatter

# Function to format numbers as M (lakhs)
def Millions(x, pos):
    return f'{int(x/1000000)}M'

palette = ["blue", "green", "Black"]

plt.figure(figsize=(15,3))

for i, year in enumerate(Monthly_sales['Year'].unique()):
    data = Monthly_sales[Monthly_sales['Year']==year]
    plt.plot(
        data['Month'], data['Sales'],
        marker='o', markersize=6, linewidth=2.5,
        label=str(year), color=palette[i]
    )

plt.title("Monthly Sales Over Years", fontsize=14)
plt.xlabel("Month", fontsize=12)
plt.ylabel("Total Sales", fontsize=12)


plt.xticks(range(1,13), calendar.month_abbr[1:13])
plt.gca().yaxis.set_major_formatter(FuncFormatter(Millions))

plt.grid(True, linestyle="--", alpha=0.7)
```

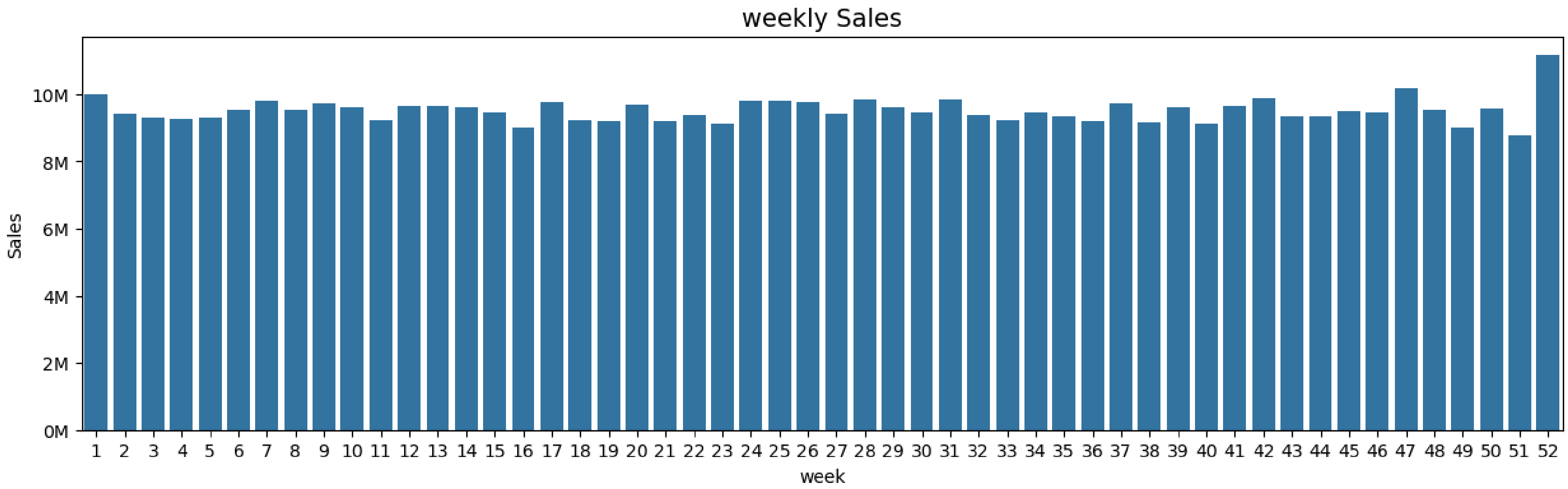
```
plt.legend(fontsize=10)
plt.show()
```



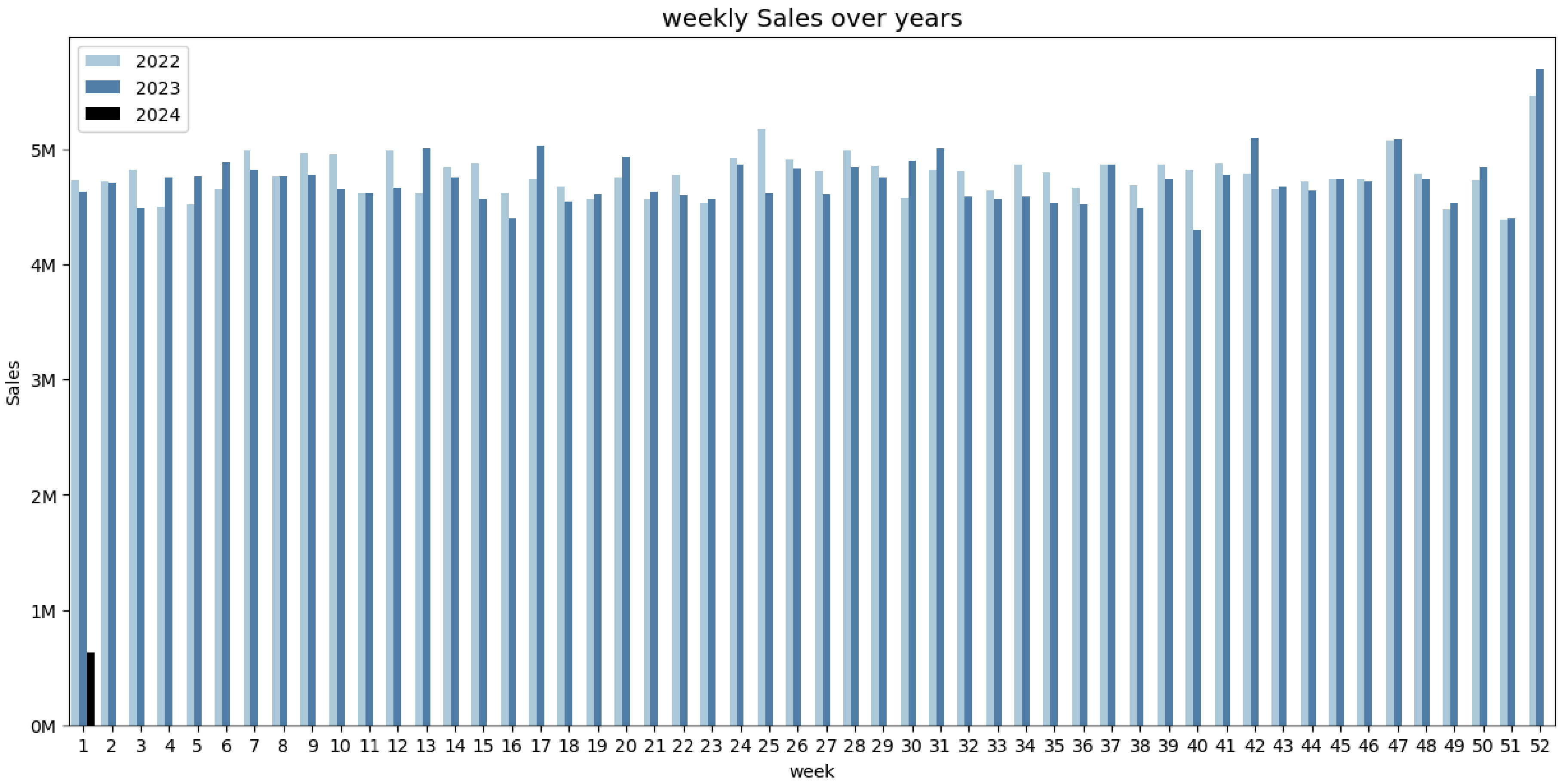
In [20]:
```
daily_sales=data1.groupby(['Year','Day'])['Sales'].sum().reset_index()

years=daily_sales['Year'].unique()
days=range(1,32)
full_index=pd.MultiIndex.from_product([years,days],names=['Year','Day'])

daily_sales=(daily_sales.set_index(['Year','Day']) .reindex(full_index,fill_value=0).reset_index())

plt.figure(figsize=(15,4))
sns.lineplot(data=daily_sales,x='Day',y='Sales',hue="Year",palette=['royalblue','seagreen','Black'])
plt.title('Daily Sales over years',fontsize=14,fontweight='10')
plt.xticks(range(1,32))
plt.gca().yaxis.set_major_formatter(FuncFormatter(Millions))
plt.grid(True,linestyle='--',alpha=0.7)
plt.show()
```



In [21]:
```
weekly_sales=data1.groupby('week')['Sales'].sum().reset_index()

plt.figure(figsize=(15,4))
sns.barplot(data=weekly_sales,x='week',y='Sales',errorbar=None)
plt.title('weekly Sales',fontsize=14,fontweight='10')
plt.gca().yaxis.set_major_formatter(FuncFormatter(Millions))
plt.show()
```

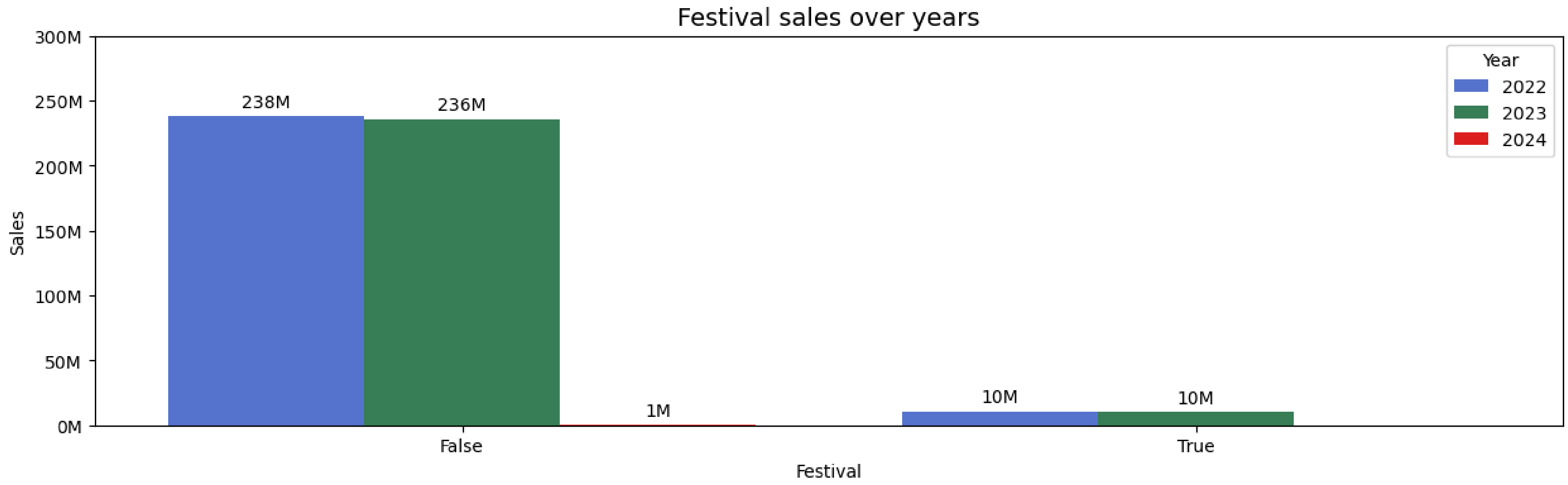


In [22]:
```
weekly_sales1=data1.groupby(['week','Year'])['Sales'].sum().reset_index()

plt.figure(figsize=(15,7))
sns.barplot(data=weekly_sales1,x='week',y='Sales',errorbar=None,hue='Year',palette = ['#a6cee3', '#4682B4', 'Black'] )
plt.title('weekly Sales over years',fontsize=14,fontweight='10')
plt.gca().yaxis.set_major_formatter(FuncFormatter(Millions))
plt.legend(fontsize=10)
plt.show()
```

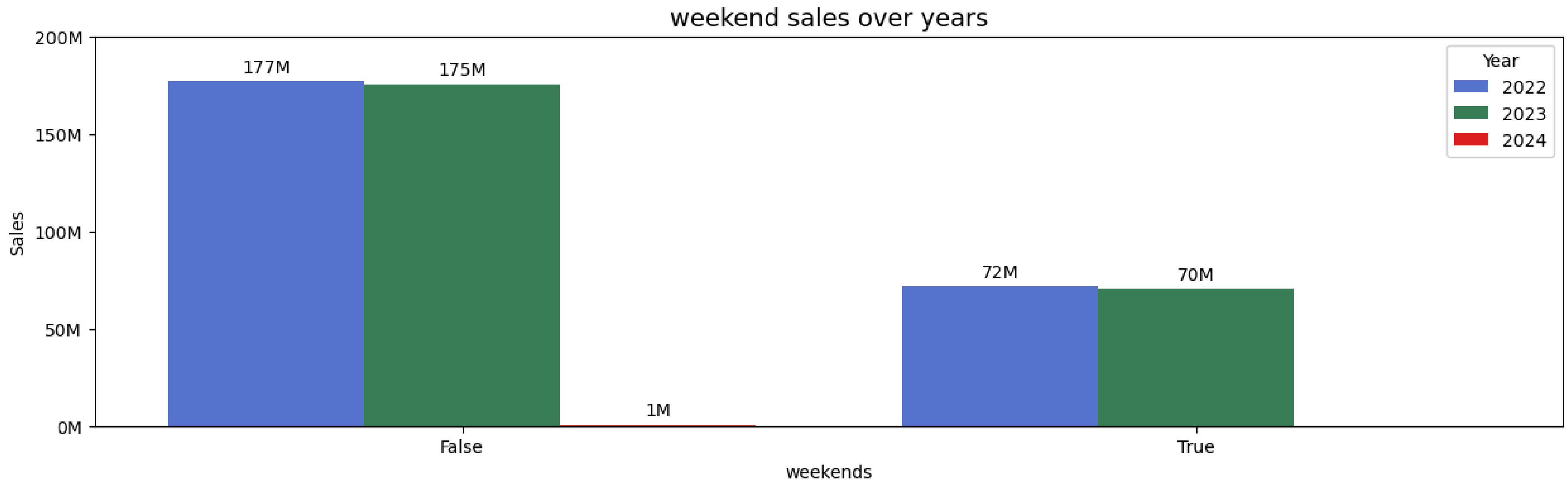


In [23]:
```
festival_sales=data1.groupby(['Year','Festival'])['Sales'].sum().reset_index()

plt.figure(figsize=(15,4))
ax=sns.barplot(data=festival_sales,x='Festival',y='Sales',hue="Year",palette=['royalblue','seagreen','Red'])

plt.title('Festival sales over years',fontsize=14,fontweight='10')
ax.yaxis.set_major_formatter(FuncFormatter(Millions))

for container in ax.containers:
    labels=[f'{v.get_height()*1e-6:.0f}M' for v in container]
    ax.bar_label(container,labels=labels,label_type='edge',padding=3)
```

```
plt.yticks(range(0, 300_000_001, 50_000_000))
plt.show()
```



In [24]:
```
weekends_sales=data1.groupby(['Year','weekends'])['Sales'].sum().reset_index()

plt.figure(figsize=(15,4))
ax=sns.barplot(data=weekends_sales,x='weekends',y='Sales',hue="Year",palette=['royalblue','seagreen','Red'])

plt.title('weekend sales over years',fontsize=14,fontweight='10')
ax.yaxis.set_major_formatter(FuncFormatter(Millions))

for container in ax.containers:
    labels=[f'{v.get_height()*1e-6:.0f}M' for v in container]
    ax.bar_label(container,labels=labels,label_type='edge',padding=3)


plt.yticks(range(0, 200_000_001, 50_000_000))
plt.show()
```



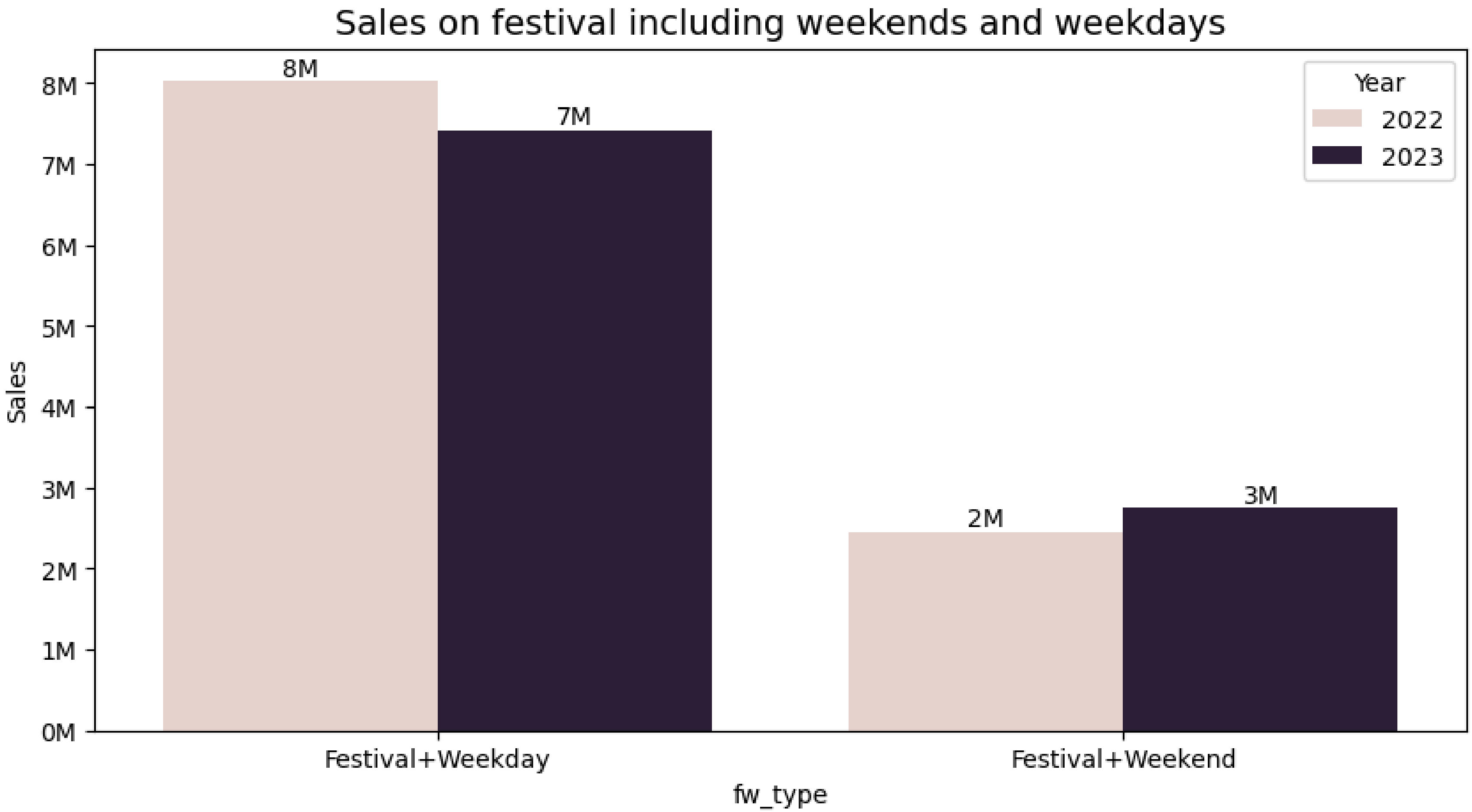
In [25]:
```
# festival+weekend shortly as fw
fw=data1[['Sales','Festival','weekends','Year']].copy()
fw=fw[fw['Festival']==True].copy()
fw = fw.reset_index(drop=True)
fw.head()
```

Out[25]:

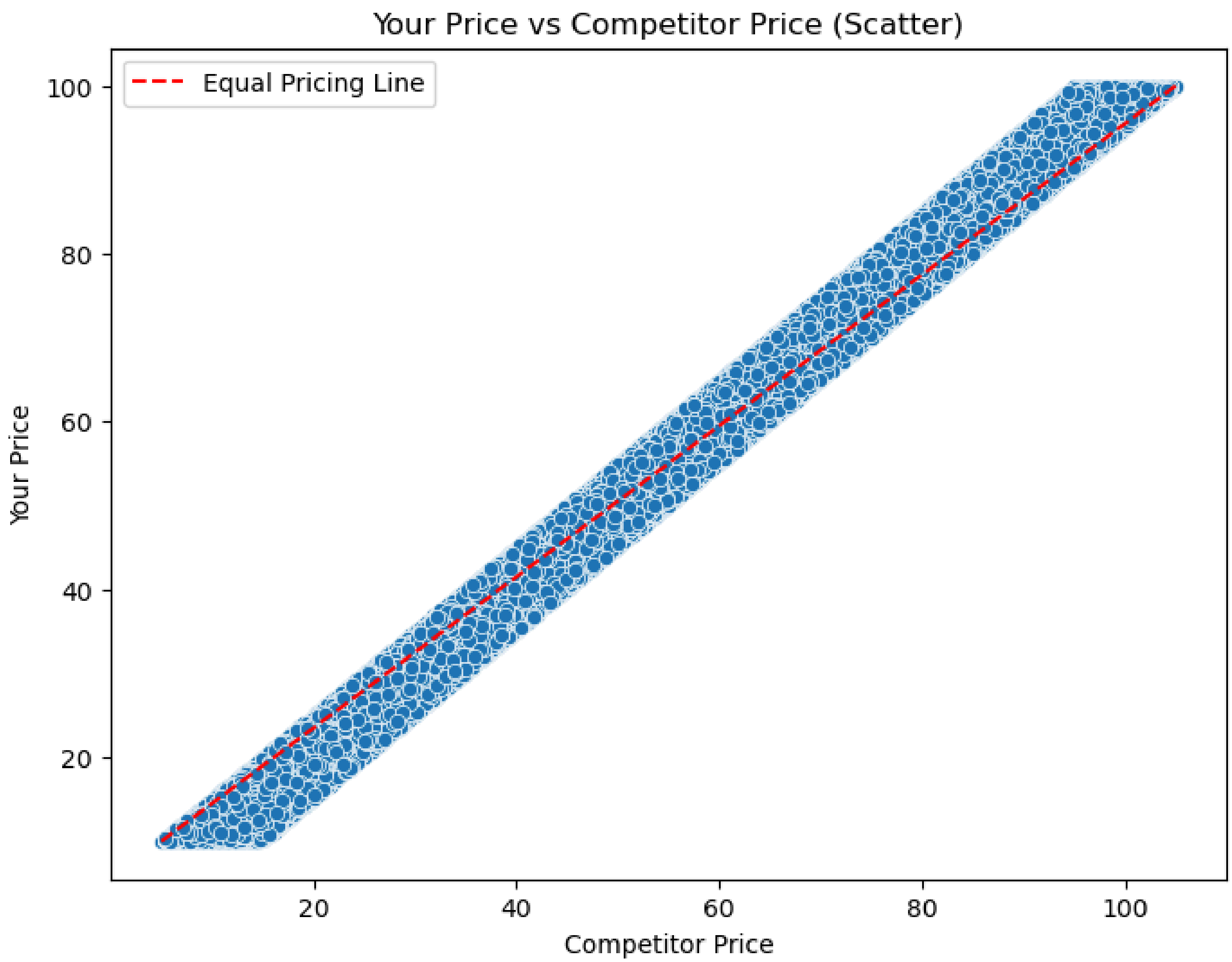
	Sales	Festival	weekends	Year
0	2768.794	True	False	2022
1	783.420	True	False	2022
2	7112.286	True	False	2022
3	5266.600	True	False	2022
4	154.000	True	False	2022

In [26]:
```
fw['fw_type']=fw['weekends'].apply(
    lambda x: 'Festival+Weekend' if x else 'Festival+Weekday'
)
```

```
fw_sales=fw.groupby(['Year','fw_type'])['Sales'].sum().reset_index()

plt.figure(figsize=(10,5))
ax=sns.barplot(data=fw_sales,x='fw_type',y='Sales',hue='Year')
ax.yaxis.set_major_formatter(FuncFormatter(Millions))

for container in ax.containers:
    labels=[f'{v.get_height()*1e-6:.0f}M' for v in container]
    ax.bar_label(container,labels=labels,label_type='edge')


plt.title('Sales on festival including weekends and weekdays',fontsize=14)
plt.show()
```

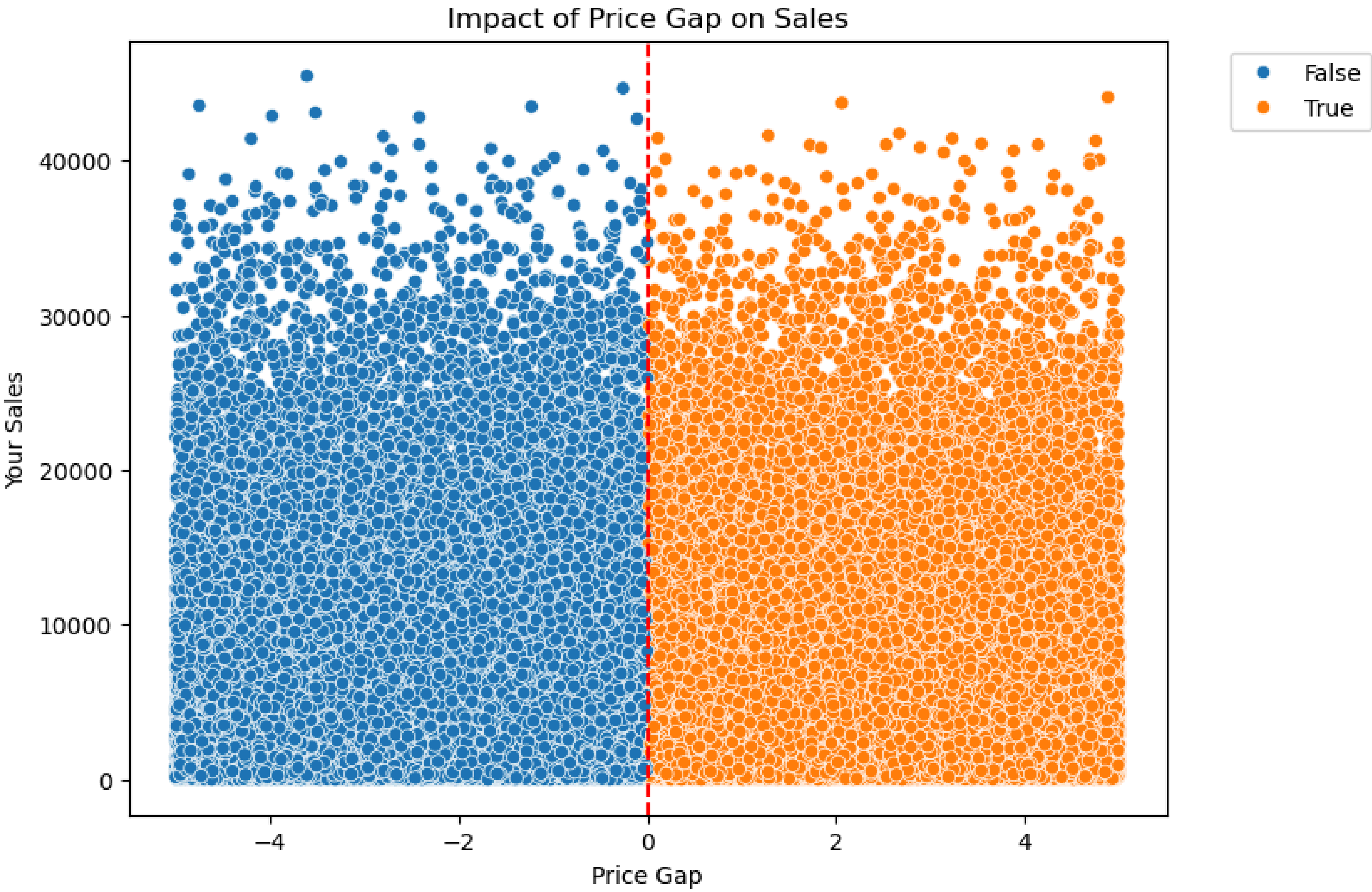


In [27]:
```
plt.figure(figsize=(8,6))
sns.scatterplot(x=data1['Competitor Pricing'], y=data1['Price'])
plt.plot([data1['Competitor Pricing'].min(), data1['Competitor Pricing'].max()],
         [data1['Price'].min(), data1['Price'].max()],
         'r--', label="Equal Pricing Line")
plt.xlabel("Competitor Price")
plt.ylabel("Your Price")
plt.title("Your Price vs Competitor Price (Scatter)")
plt.legend()
plt.show()
```



In [28]:
```
df=data1.copy()
df['Price Gap'] = data1['Price'] - data1['Competitor Pricing']

plt.figure(figsize=(8,6))
sns.scatterplot(x=df['Price Gap'], y=df['Sales'], hue=df['Price Gap']>0)
plt.axvline(0, color='red', linestyle='--')
plt.xlabel("Price Gap")
```

```
plt.ylabel("Your Sales")
plt.title("Impact of Price Gap on Sales")
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



In [29]:
```
df['Discount_Flag'] = df['Discount'].apply(lambda x: 1 if x > 0 else 0)
```

```
df['Competitor_Cheaper'] = (df['Competitor Pricing'] < df['Price']).astype(int)


discount_ratio = df.groupby('Category').apply(
    lambda g: g[g['Discount_Flag']==1]['Units Sold'].sum() / g['Units Sold'].sum()
).reset_index(name='discount_ratio')

discount_ratio['Segment_x'] = discount_ratio['discount_ratio'].apply(
    lambda x: 'Budget' if x > 0.5 else 'Premium'
)

df = df.merge(discount_ratio[['Category','Segment_x']], on='Category', how='left')

df['Segment_y'] = df['Competitor_Cheaper'].apply(lambda x: 'Budget' if x==1 else 'Premium')


df_final = df[['Discount_Flag','Competitor_Cheaper','Segment_x','Segment_y']]

df_final.head()
```

Out[29]:

	Discount_Flag	Competitor_Cheaper	Segment_x	Segment_y
0	1	1	Budget	Budget
1	1	0	Budget	Premium
2	1	0	Budget	Premium
3	1	0	Budget	Premium
4	0	1	Budget	Budget

In [30]:
```
df['Avg_Order_Value'] = df['Sales'] / df['Units Sold']

median_value = df['Avg_Order_Value'].median()
df['Customer_Segment'] = ['Premium' if x > median_value else 'Budget' for x in df['Avg_Order_Value']]
```

```
df_final = df[['Discount_Flag','Competitor_Cheaper','Segment_x','Segment_y','Avg_Order_Value','Customer_Segment']].copy()
df_final.head()
```

Out[30]:

	Discount_Flag	Competitor_Cheaper	Segment_x	Segment_y	Avg_Order_Value	Customer_Segment
0	1	1	Budget	Budget	26.800	Budget
1	1	0	Budget	Premium	50.408	Premium
2	1	0	Budget	Premium	25.191	Budget
3	1	0	Budget	Premium	29.448	Budget
4	0	1	Budget	Budget	73.640	Premium

In [31]:
```
print(df_final['Customer_Segment'].value_counts())
```

```
Customer_Segment
Budget     36731
Premium    36369
Name: count, dtype: int64
```

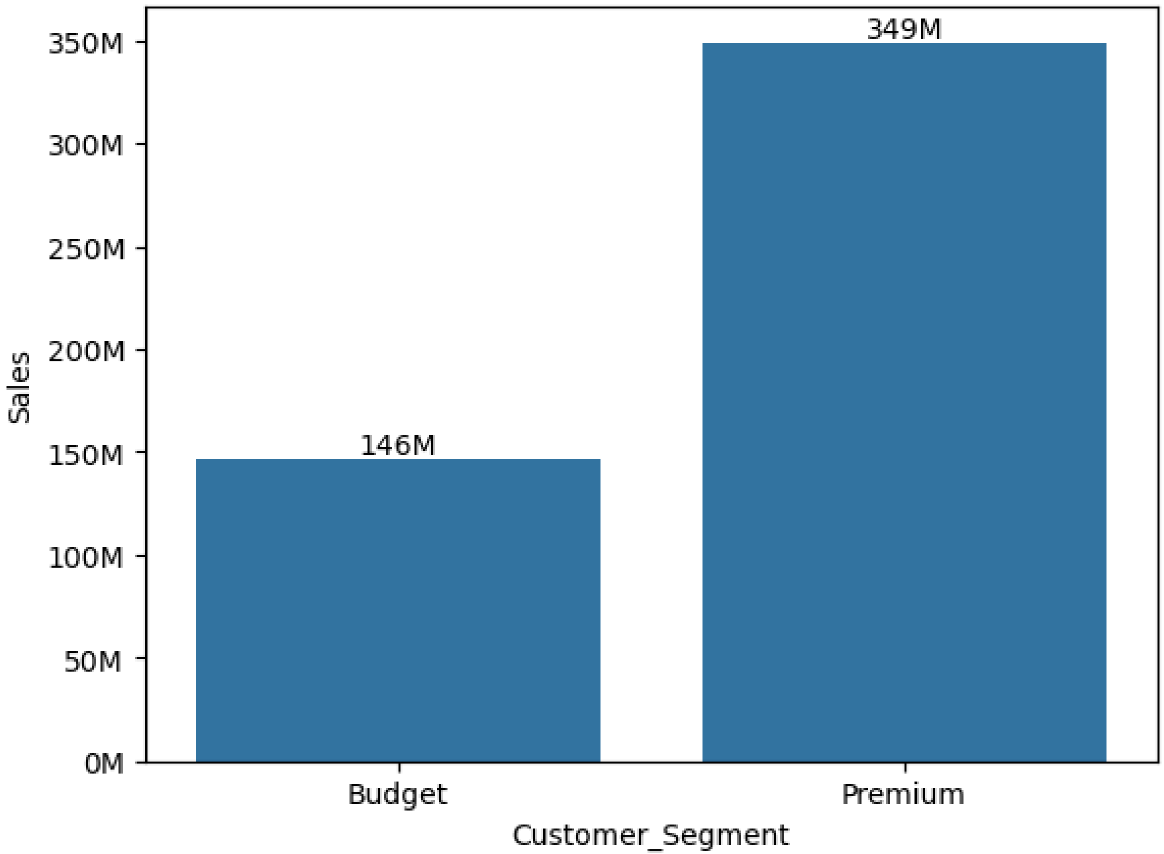
In [32]:
```
df_combined=pd.concat([df_final['Customer_Segment'],data1['Sales']],axis=1)
customers_segment=df_combined.groupby('Customer_Segment')['Sales'].sum().reset_index()
customers_segment.head()
```

Out[32]:

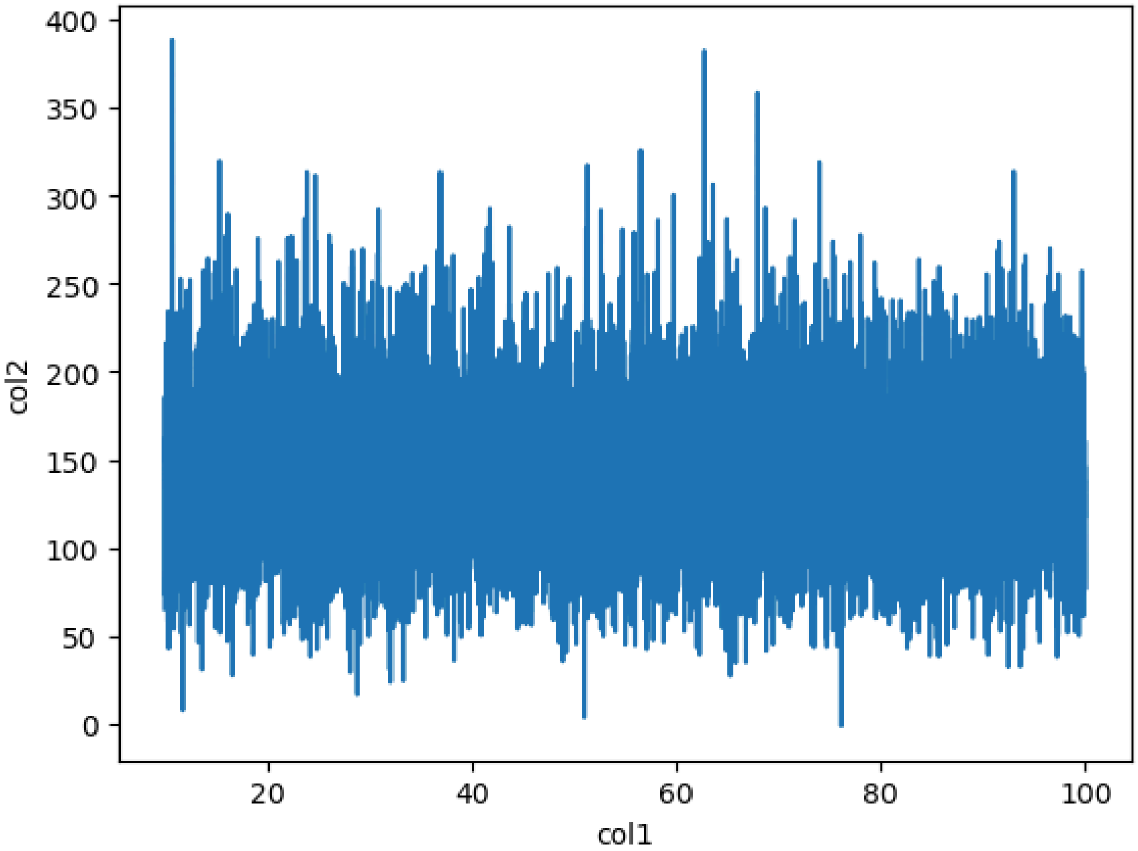
	Customer_Segment	Sales
0	Budget	1.460139e+08
1	Premium	3.489575e+08

In [33]:
```
ax=sns.barplot(data=customers_segment,x='Customer_Segment',y='Sales')
ax.yaxis.set_major_formatter(FuncFormatter(Millions))

for container in ax.containers:
    labels=[f'{v.get_height()*1e-6:.0f}M' for v in container]
    ax.bar_label(container,labels=labels,label_type='edge')
```

```
plt.show()
```



In [34]:
```
sns.lineplot(x=data1['Price'], y=data1['Demand Forecast'], errorbar=None)  # ci=None disables the confidence interval
plt.xlabel('col1')
plt.ylabel('col2')
plt.show()
```



In [35]:
```
seasonal_sales=data1.groupby(['Seasonality','Category','Year'])['Sales'].sum().reset_index()
seasonal_sales
```

Out[35]:

	Seasonality	Category	Year	Sales
0	Autumn	Clothing	2022	1.219360e+07
1	Autumn	Clothing	2023	1.244188e+07
2	Autumn	Clothing	2024	6.756344e+03
3	Autumn	Electronics	2022	1.233930e+07
4	Autumn	Electronics	2023	1.199309e+07
5	Autumn	Electronics	2024	1.956753e+03
6	Autumn	Furniture	2022	1.200984e+07
7	Autumn	Furniture	2023	1.283389e+07
8	Autumn	Furniture	2024	5.418867e+04
9	Autumn	Groceries	2022	1.297704e+07
10	Autumn	Groceries	2023	1.285403e+07
11	Autumn	Groceries	2024	1.656049e+04
12	Autumn	Toys	2022	1.196246e+07
13	Autumn	Toys	2023	1.243491e+07
14	Autumn	Toys	2024	2.995070e+04
15	Spring	Clothing	2022	1.238263e+07
16	Spring	Clothing	2023	1.206745e+07
17	Spring	Clothing	2024	2.617996e+04
18	Spring	Electronics	2022	1.220919e+07
19	Spring	Electronics	2023	1.221990e+07
20	Spring	Electronics	2024	4.724825e+04

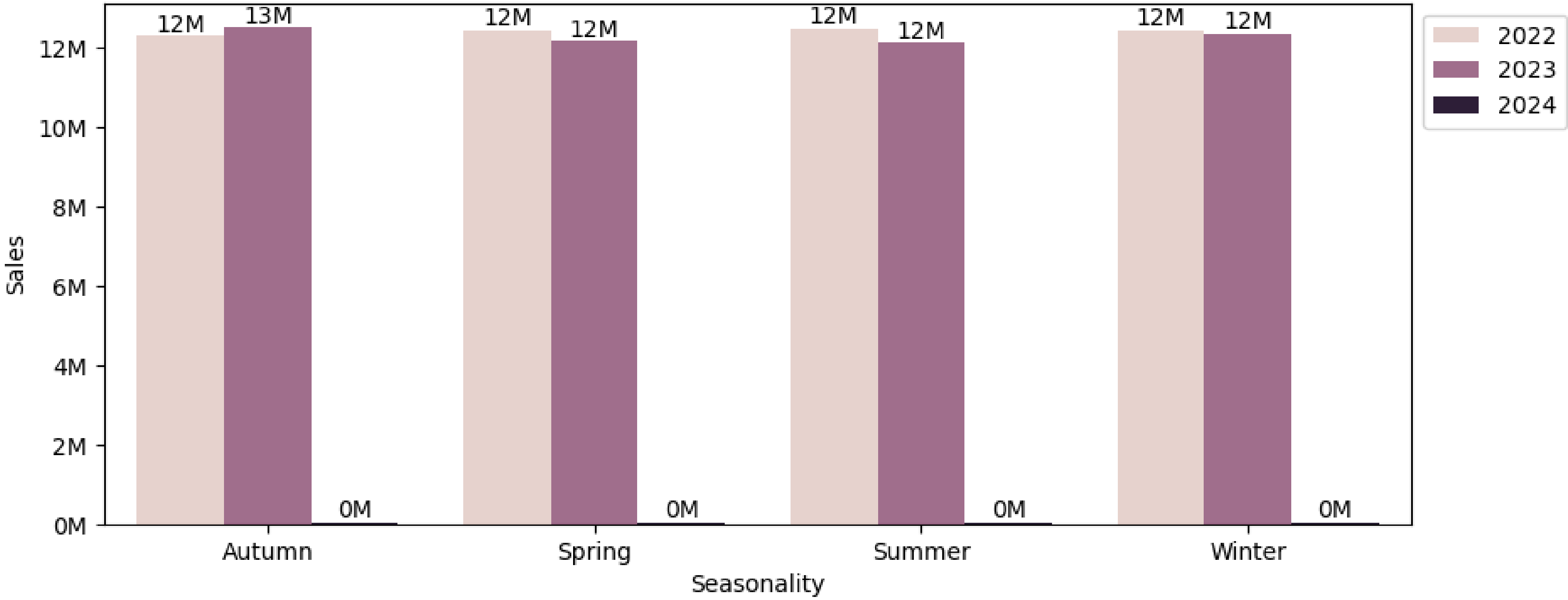
	Seasonality	Category	Year	Sales
21	Spring	Furniture	2022	1.281932e+07
22	Spring	Furniture	2023	1.279839e+07
23	Spring	Furniture	2024	3.471517e+04
24	Spring	Groceries	2022	1.196353e+07
25	Spring	Groceries	2023	1.182807e+07
26	Spring	Groceries	2024	3.872324e+04
27	Spring	Toys	2022	1.290534e+07
28	Spring	Toys	2023	1.193271e+07
29	Spring	Toys	2024	2.223962e+04
30	Summer	Clothing	2022	1.208502e+07
31	Summer	Clothing	2023	1.233262e+07
32	Summer	Clothing	2024	3.188510e+04
33	Summer	Electronics	2022	1.194781e+07
34	Summer	Electronics	2023	1.267945e+07
35	Summer	Electronics	2024	4.082071e+04
36	Summer	Furniture	2022	1.237651e+07
37	Summer	Furniture	2023	1.195202e+07
38	Summer	Furniture	2024	4.965300e+03
39	Summer	Groceries	2022	1.324416e+07
40	Summer	Groceries	2023	1.172613e+07
41	Summer	Groceries	2024	1.520009e+04

	Seasonality	Category	Year	Sales
42	Summer	Toys	2022	1.276436e+07
43	Summer	Toys	2023	1.194802e+07
44	Summer	Toys	2024	1.859331e+04
45	Winter	Clothing	2022	1.258104e+07
46	Winter	Clothing	2023	1.252181e+07
47	Winter	Clothing	2024	5.897699e+03
48	Winter	Electronics	2022	1.206953e+07
49	Winter	Electronics	2023	1.179105e+07
50	Winter	Electronics	2024	4.624375e+04
51	Winter	Furniture	2022	1.264326e+07
52	Winter	Furniture	2023	1.262070e+07
53	Winter	Furniture	2024	8.302486e+04
54	Winter	Groceries	2022	1.285890e+07
55	Winter	Groceries	2023	1.238925e+07
56	Winter	Groceries	2024	3.737191e+04
57	Winter	Toys	2022	1.212201e+07
58	Winter	Toys	2023	1.252300e+07
59	Winter	Toys	2024	6.562084e+04

In [36]:

```
plt.figure(figsize=(10,4))
ax=sns.barplot(data=seasonal_sales,x='Seasonality',y='Sales',hue='Year',errorbar=None)
ax.yaxis.set_major_formatter(FuncFormatter(Millions))

for container in ax.containers:
```

```
    labels=[f'{v.get_height()*1e-6:.0f}M' for v in container]
    ax.bar_label(container,labels=labels,label_type='edge')

plt.legend(bbox_to_anchor=(1,1),loc='upper left')
plt.show()
```



Feature Engineering

In [37]:
```
data1.columns
```

Out[37]:
```
Index(['Date', 'Category', 'Region', 'Inventory Level', 'Units Sold',
       'Units Ordered', 'Demand Forecast', 'Price', 'Discount',
       'Weather Condition', 'Holiday/Promotion', 'Competitor Pricing',
       'Seasonality', 'Day', 'week', 'weekends', 'Month', 'Year', 'Festival',
       'Sales'],
      dtype='object')
```

In [38]:
```
#Moving averages
data1=data1.sort_values(['Category','Date']).reset_index()

data1['Unitssold_MA7']=data1.groupby('Category')["Units Sold"].transform(lambda x :x.shift(1).rolling(window=7).mean())
data1['Unitssold_MA30']=data1.groupby('Category')["Units Sold"].transform(lambda x :x.shift(1).rolling(window=30).mean())

data1.head()
```

Out[38]:

	index	Date	Category	Region	Inventory Level	Units Sold	Units Ordered	Demand Forecast	Price	Discount	...	Seasonality	Day	week	weekends	Mor
0	7	2022-01-01	Clothing	North	380	312	54	329.73	97.99	5	...	Spring	1	52	True	
1	11	2022-01-01	Clothing	West	66	24	70	26.75	58.25	20	...	Spring	1	52	True	
2	13	2022-01-01	Clothing	West	193	12	187	6.80	78.11	0	...	Spring	1	52	True	
3	14	2022-01-01	Clothing	North	379	369	154	363.46	92.99	15	...	Winter	1	52	True	
4	17	2022-01-01	Clothing	South	241	151	47	147.27	19.57	5	...	Autumn	1	52	True	

5 rows × 23 columns

In [39]:
```
data1['Unitssold_MA7'].fillna(df.groupby('Category')['Units Sold'].transform('mean'), inplace=True)

data1['Unitssold_MA30'].fillna(df.groupby('Category')['Units Sold'].transform('mean'), inplace=True)
data1.head()
```

Out[39]:

	index	Date	Category	Region	Inventory Level	Units Sold	Units Ordered	Demand Forecast	Price	Discount	...	Seasonality	Day	week	weekends	Mor
0	7	2022-01-01	Clothing	North	380	312	54	329.73	97.99	5	...	Spring	1	52	True	
1	11	2022-01-01	Clothing	West	66	24	70	26.75	58.25	20	...	Spring	1	52	True	
2	13	2022-01-01	Clothing	West	193	12	187	6.80	78.11	0	...	Spring	1	52	True	
3	14	2022-01-01	Clothing	North	379	369	154	363.46	92.99	15	...	Winter	1	52	True	
4	17	2022-01-01	Clothing	South	241	151	47	147.27	19.57	5	...	Autumn	1	52	True	

5 rows × 23 columns

In [40]:

```
#lag features
data1['lag1']=data1.groupby('Category')['Units Sold'].shift(1)
data1['lag7']=data1.groupby('Category')['Units Sold'].shift(7)

data1['lag1'] = data1.groupby('Category')['lag1'].transform(lambda x: x.fillna(x.mean()))
data1['lag7'] = data1.groupby('Category')['lag7'].transform(lambda x: x.fillna(x.mean()))

data1.head()
```

Out[40]:

	index	Date	Category	Region	Inventory Level	Units Sold	Units Ordered	Demand Forecast	Price	Discount	...	week	weekends	Month	Year	Festival
0	7	2022-01-01	Clothing	North	380	312	54	329.73	97.99	5	...	52	True	1	2022	False
1	11	2022-01-01	Clothing	West	66	24	70	26.75	58.25	20	...	52	True	1	2022	False
2	13	2022-01-01	Clothing	West	193	12	187	6.80	78.11	0	...	52	True	1	2022	False
3	14	2022-01-01	Clothing	North	379	369	154	363.46	92.99	15	...	52	True	1	2022	False
4	17	2022-01-01	Clothing	South	241	151	47	147.27	19.57	5	...	52	True	1	2022	False

5 rows × 25 columns

In [41]:

```
data1=data1.drop(['Sales', 'Date', 'Units Ordered','index'],axis=1)
data1.head()
```

Out[41]:

	Category	Region	Inventory Level	Units Sold	Demand Forecast	Price	Discount	Weather Condition	Holiday/Promotion	Competitor Pricing	...	Day	week	weeken
0	Clothing	North	380	312	329.73	97.99	5	Cloudy	0	100.09	...	1	52	Tru
1	Clothing	West	66	24	26.75	58.25	20	Snowy	0	62.21	...	1	52	Tru
2	Clothing	West	193	12	6.80	78.11	0	Sunny	0	80.06	...	1	52	Tru
3	Clothing	North	379	369	363.46	92.99	15	Snowy	0	95.80	...	1	52	Tru
4	Clothing	South	241	151	147.27	19.57	5	Cloudy	0	23.13	...	1	52	Tru

5 rows × 21 columns

Encoding

In [42]:
```
data1.shape
```

Out[42]: (73100, 21)

In [43]:
```
categorical_columns=['Category','Region','Weather Condition','Seasonality']

data1['Festival']=data1['Festival'].astype(int)
data1['weekends']=data1['weekends'].astype(int)

encoded_data=pd.get_dummies(data1,columns=categorical_columns,drop_first=True)

encoded_data.head()
```

Out[43]:

	Inventory Level	Units Sold	Demand Forecast	Price	Discount	Holiday/Promotion	Competitor Pricing	Day	week	weekends	...	Category_Toys	Region_Nor
0	380	312	329.73	97.99	5	0	100.09	1	52	1	...	False	Tr
1	66	24	26.75	58.25	20	0	62.21	1	52	1	...	False	Fal
2	193	12	6.80	78.11	0	0	80.06	1	52	1	...	False	Fal
3	379	369	363.46	92.99	15	0	95.80	1	52	1	...	False	Tr
4	241	151	147.27	19.57	5	0	23.13	1	52	1	...	False	Fal

5 rows × 30 columns

In [44]:
```
encoded_data.isna().sum()
```

Out[44]:
```
Inventory Level              0
Units Sold                   0
Demand Forecast              0
Price                        0
Discount                     0
Holiday/Promotion            0
Competitor Pricing           0
Day                          0
week                         0
weekends                     0
Month                        0
Year                         0
Festival                     0
Unitssold_MA7                0
Unitssold_MA30               0
lag1                         0
lag7                         0
Category_Electronics         0
Category_Furniture           0
Category_Groceries           0
Category_Toys                0
Region_North                 0
Region_South                 0
Region_West                  0
Weather Condition_Rainy      0
Weather Condition_Snowy      0
Weather Condition_Sunny      0
Seasonality_Spring           0
Seasonality_Summer           0
Seasonality_Winter           0
dtype: int64
```

Modeling

In [45]:
```
x=encoded_data.drop('Units Sold',axis=1)
y=encoded_data['Units Sold']
```

```
#Data split

from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

In [46]:
```
x_train.shape,y_train.shape
```

Out[46]:
```
((58480, 29), (58480,))
```

In [47]:
```
x_test.shape,y_test.shape
```

Out[47]:
```
((14620, 29), (14620,))
```

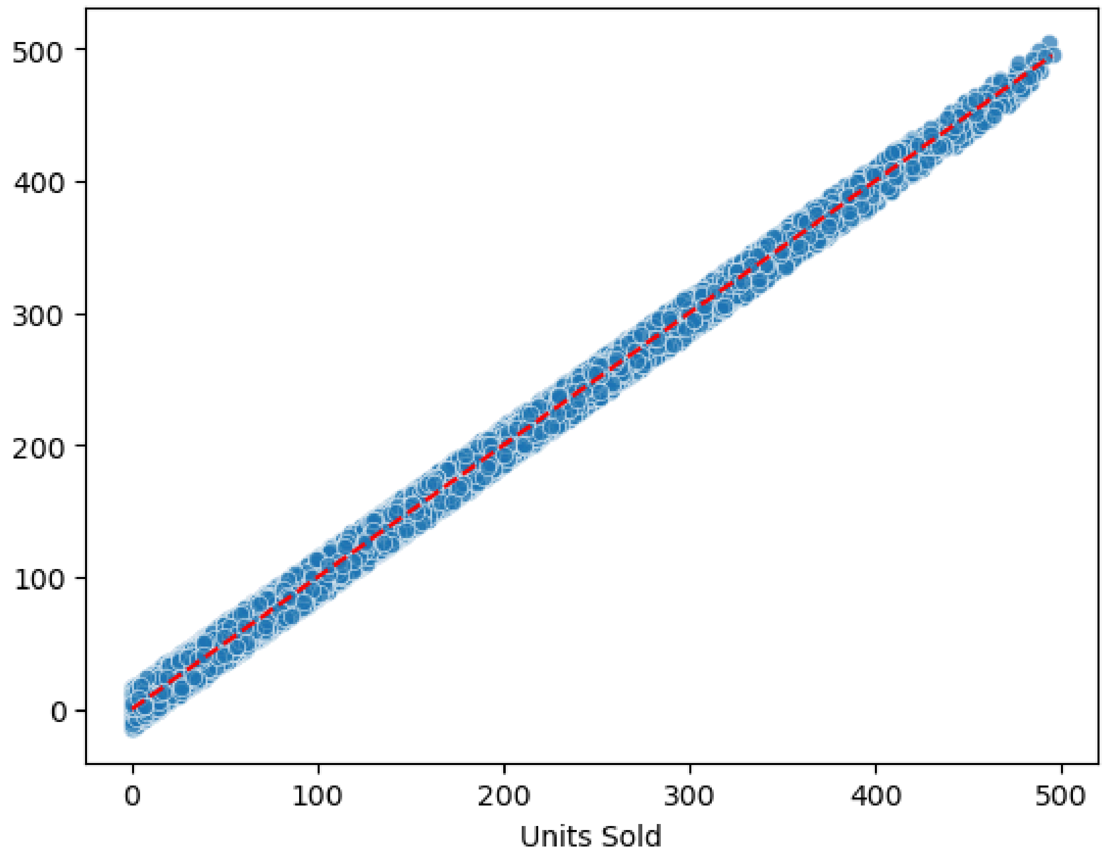
Linear regression model

In [48]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score

model=LinearRegression()
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
```

In [49]:
```
y_pred
```

Out[49]:
```
array([389.63491933, 281.09323898, 148.83860222, ...,  63.15390248,
        71.3223332 ,  81.51414376])
```

In [50]:
```
sns.scatterplot(x=y_test,y=y_pred,alpha=0.7)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.show()
```



In [51]:
```
print('mae:',mean_absolute_error(y_test,y_pred))
print('mse:',mean_squared_error(y_test,y_pred))
print('rmse:',np.sqrt(mean_squared_error(y_test,y_pred)))
print('r2 score:',r2_score(y_test,y_pred))
```

```
mae: 7.490459252676142
mse: 74.98616850268004
rmse: 8.659455439153206
r2 score: 0.9936078324057547
```

Random forest regressor model

In [52]:
```
from sklearn.ensemble import RandomForestRegressor

rf_model=RandomForestRegressor()
rf_model.fit(x_train,y_train)
y_pred1=rf_model.predict(x_test)
```

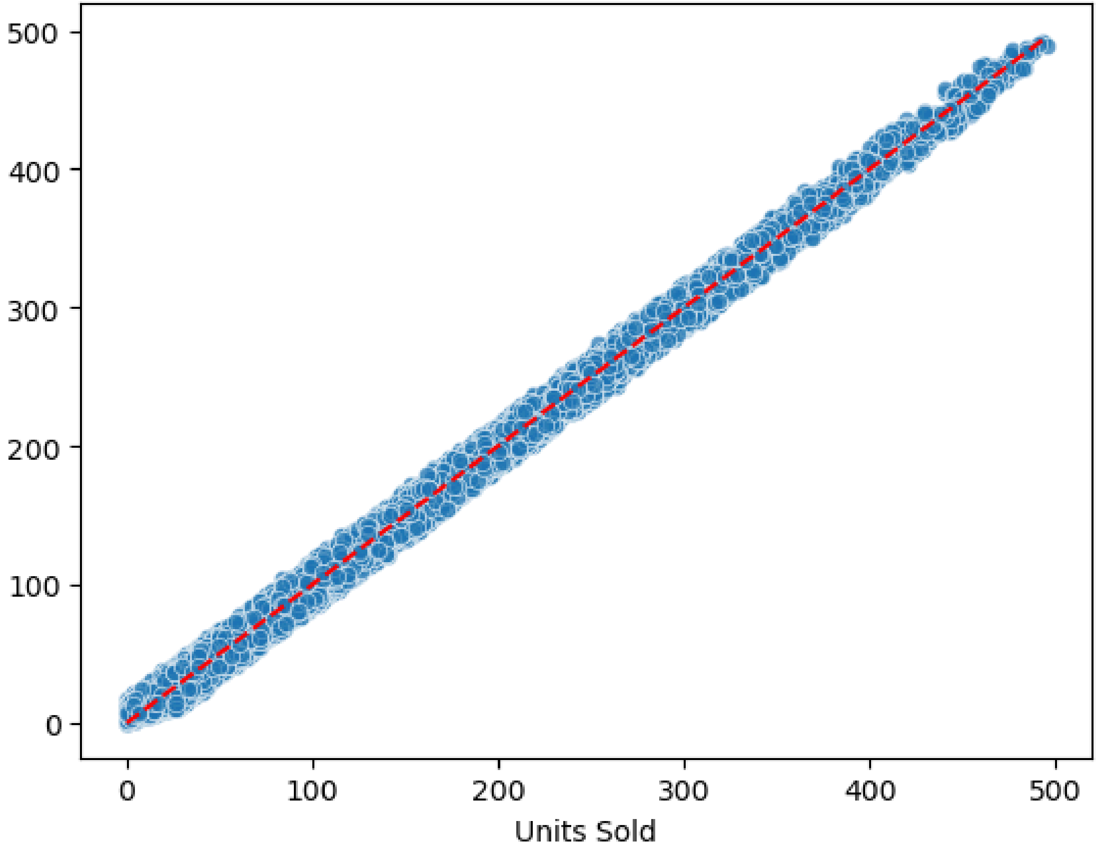
In [53]:
```
y_pred1
```

Out[53]:
```
array([387.99, 285.21, 148.73, ...,  57.33,  71.81,  80.24])
```

In [54]:
```
print('mae:',mean_absolute_error(y_test,y_pred1))
print('mse:',mean_squared_error(y_test,y_pred1))
print('rmse:',np.sqrt(mean_squared_error(y_test,y_pred1)))
print('r2 score:',r2_score(y_test,y_pred1))
```

```
mae: 7.287563611491108
mse: 73.66008597811216
rmse: 8.582545425345103
r2 score: 0.9937208738093908
```

In [55]:
```
sns.scatterplot(x=y_test,y=y_pred1,alpha=0.7)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.show()
```



extreme gradient boosting

In [56]:
```
import xgboost as xgb

xgb_model=xgb.XGBRegressor(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=6,
    random_state=42
)
```

```
xgb_model.fit(x_train,y_train)
y_pred2=xgb_model.predict(x_test)
```

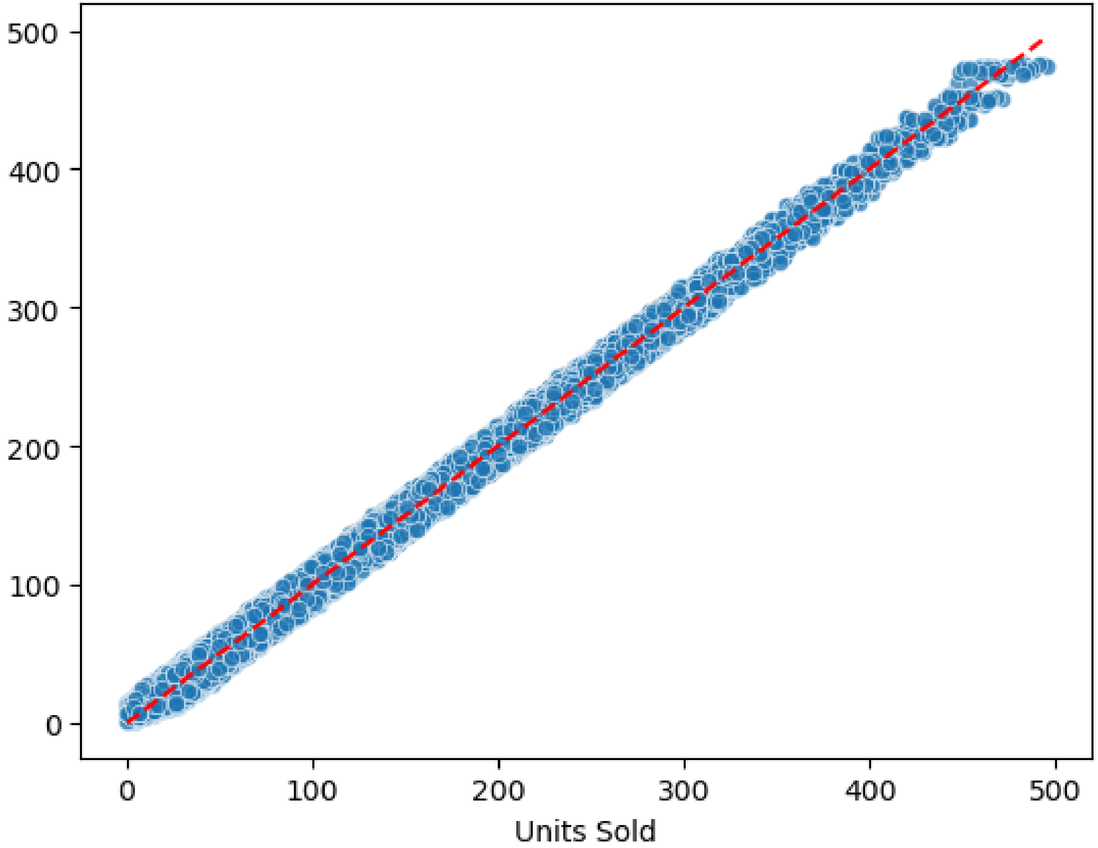
In [57]:
```
y_pred2
```

Out[57]:
```
array([389.03024 , 285.5362  , 149.07155 , ...,  58.913464,  71.91212 ,
        82.616806], dtype=float32)
```

In [58]:
```
print('mae:',mean_absolute_error(y_test,y_pred2))
print('mse:',mean_squared_error(y_test,y_pred2))
print('rmse:',np.sqrt(mean_squared_error(y_test,y_pred2)))
print('r2 score:',r2_score(y_test,y_pred2))
```

```
mae: 7.23458081363164
mse: 71.7167968270309
rmse: 8.468577024921654
r2 score: 0.9938865287586411
```

In [59]:
```
sns.scatterplot(x=y_test,y=y_pred2,alpha=0.7)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.show()
```



In [60]:
```
metrics_dict = {
    'MAE': [
        mean_absolute_error(y_test, y_pred),
        mean_absolute_error(y_test, y_pred1),
        mean_absolute_error(y_test, y_pred2)
    ],
    'MSE': [
        mean_squared_error(y_test, y_pred),
        mean_squared_error(y_test, y_pred1),
        mean_squared_error(y_test, y_pred2)
    ],
    'RMSE': [
        np.sqrt(mean_squared_error(y_test, y_pred)),
```

```
        np.sqrt(mean_squared_error(y_test, y_pred1)),
        np.sqrt(mean_squared_error(y_test, y_pred2))
    ],
    'R2_Score': [
        r2_score(y_test, y_pred),
        r2_score(y_test, y_pred1),
        r2_score(y_test, y_pred2)
    ]
}

# Create DataFrame
df_metrics = pd.DataFrame(metrics_dict, index=['LinearRegression', 'RandomForestRegressor', 'xgb']).T

df_metrics
```

Out[60]:

	LinearRegression	RandomForestRegressor	xgb
MAE	7.490459	7.287564	7.234581
MSE	74.986169	73.660086	71.716797
RMSE	8.659455	8.582545	8.468577
R2_Score	0.993608	0.993721	0.993887

Forecasting

In [65]:

```
forecast_data=data1.copy()
forecast_data['Sales']=forecast_data['Units Sold']*forecast_data['Price']*(1-forecast_data['Discount']/100)
forecast_data.head()
```

Out[65]:

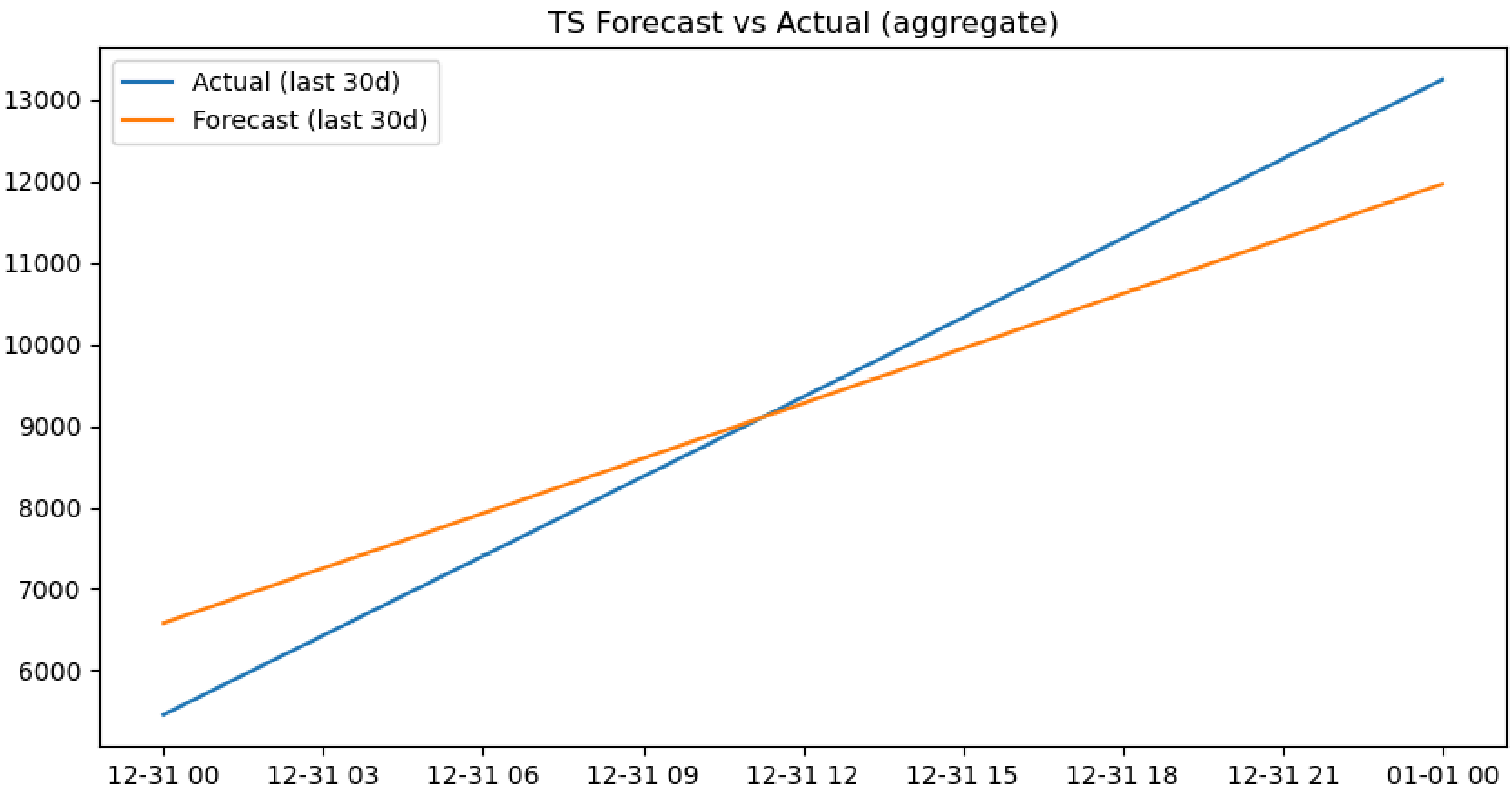
	Category	Region	Inventory Level	Units Sold	Demand Forecast	Price	Discount	Weather Condition	Holiday/Promotion	Competitor Pricing	...	week	weekends	M
0	Clothing	North	380	312	329.73	97.99	5	Cloudy	0	100.09	...	52	1	
1	Clothing	West	66	24	26.75	58.25	20	Snowy	0	62.21	...	52	1	
2	Clothing	West	193	12	6.80	78.11	0	Sunny	0	80.06	...	52	1	
3	Clothing	North	379	369	363.46	92.99	15	Snowy	0	95.80	...	52	1	
4	Clothing	South	241	151	147.27	19.57	5	Cloudy	0	23.13	...	52	1	

5 rows × 22 columns

In [81]:

```
cats = forecast_data['Category'].unique().tolist()
ts_forecasts = []

for cat in cats:
    ser = df.loc[df['Category']==cat, ['Date','Units Sold']].set_index('Date').fillna(0)
    y = ser['Units Sold']
    train, test = y.iloc[:-30], y.iloc[-30:]

    try:
        model = sm.tsa.SARIMAX(train, order=(1,1,1), seasonal_order=(1,1,1,7),
                               enforce_stationarity=False, enforce_invertibility=False)
        res = model.fit(disp=False)
        fc = res.get_forecast(steps=len(test)).predicted_mean
    except:
        fc = pd.Series(np.repeat(train.tail(7).mean(), len(test)), index=test.index)

    ts_forecasts.append(pd.DataFrame({
        'Date': test.index,
        'Category': cat,
        'ts_forecast': fc.values,
        'ts_actual': test.values
    }))
```

```
ts_forecasts = pd.concat(ts_forecasts).reset_index(drop=True)


agg_ts = ts_forecasts.groupby('Date')[['ts_forecast','ts_actual']].sum().reset_index()

plt.figure(figsize=(10,5))
plt.plot(agg_ts['Date'], agg_ts['ts_actual'], label='Actual (last 30d)')
plt.plot(agg_ts['Date'], agg_ts['ts_forecast'], label='Forecast (last 30d)')
plt.legend(); plt.title("TS Forecast vs Actual (aggregate)"); plt.show()
```



In [80]:
```
ml_df = pd.get_dummies(forecast_data, columns=['Category','Region','Weather Condition','Seasonality'], drop_first=True)
FEATURE_COLS = [c for c in ml_df.columns if c not in ['Date','Units Sold','Sales']]
X, y = ml_df[FEATURE_COLS], ml_df['Units Sold']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
xgb_model = xgb.XGBRegressor(n_estimators=300, learning_rate=0.05, max_depth=6)
xgb_model.fit(X_train, y_train, eval_set=[(X_test,y_test)], verbose=False)
```

Out[80]:

XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.05, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=6, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
```

In [79]:

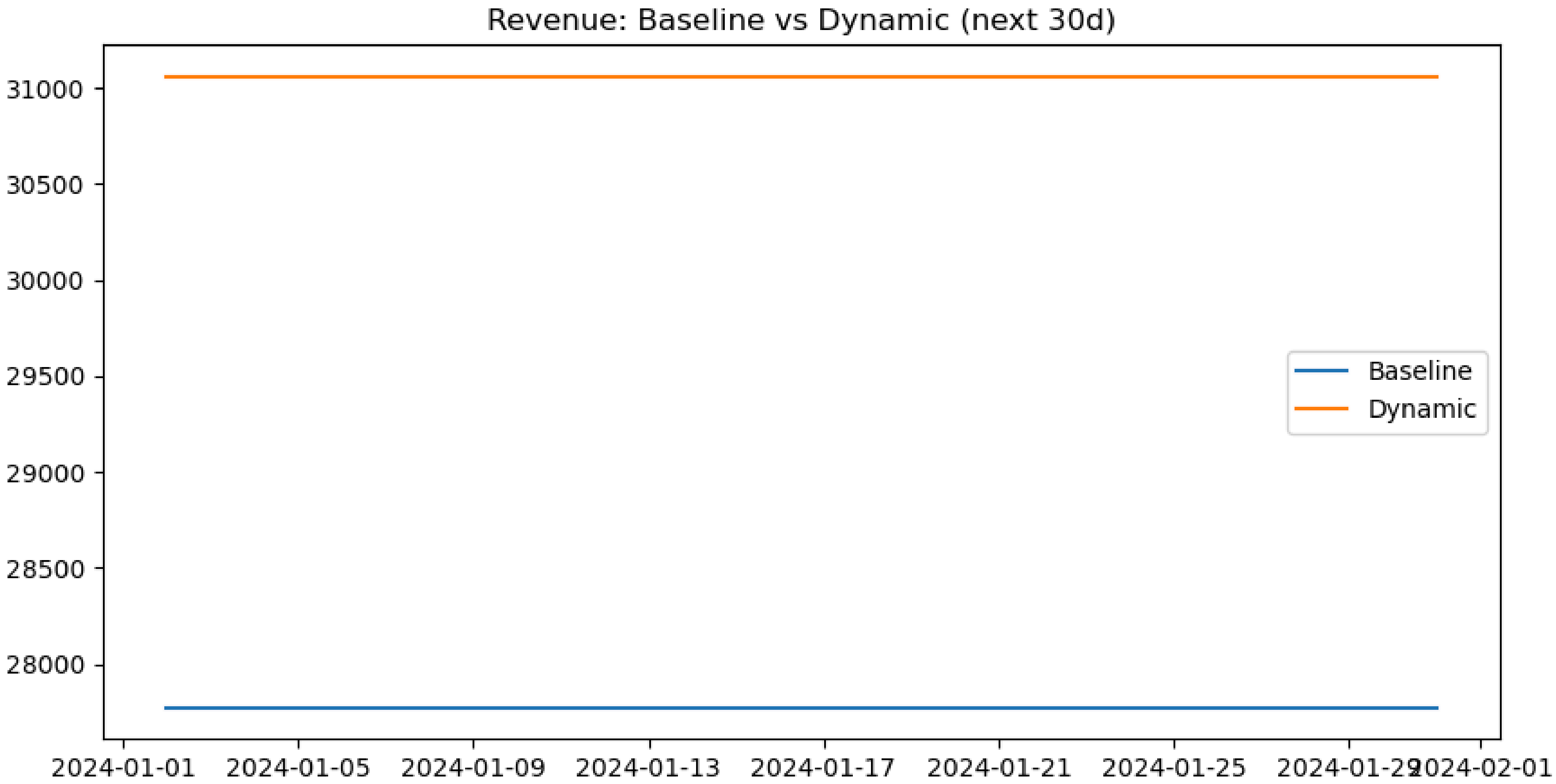
```
last_date = df['Date'].max()
future_dates = pd.date_range(last_date+pd.Timedelta(days=1), periods=30)

future_rows = []
for cat in cats:
    template = df[df['Category']==cat].iloc[-1].to_dict()
    for d in future_dates:
        r = template.copy(); r['Date'] = d; future_rows.append(r)
future_df = pd.DataFrame(future_rows)

future_enc = pd.get_dummies(future_df, columns=['Category','Region'], drop_first=True)
for c in FEATURE_COLS:
    if c not in future_enc.columns: future_enc[c] = 0
future_enc = future_enc[FEATURE_COLS]

PRICE_RANGE, GRID_STEPS = 0.3, 13
future_df['best_price'], future_df['best_rev'] = np.nan, np.nan

for i,row in future_df.iterrows():
    base_price = row.get('Competitor Pricing', row['Price'])
```

```
    cand_prices = np.linspace(base_price*(1-PRICE_RANGE), base_price*(1+PRICE_RANGE), GRID_STEPS)
    best_rev, best_price = -1, row['Price']
    for p in cand_prices:
        feat = future_enc.iloc[[i]].copy(); feat['Price'] = p
        units_hat = max(0, xgb_model.predict(feat)[0])
        rev = units_hat * p * (1 - row['Discount']/100)
        if rev > best_rev: best_rev, best_price = rev, p
    future_df.loc[i,'best_price'] = best_price
    future_df.loc[i,'best_rev'] = best_rev
```

In [78]:

```
median_price = df.groupby('Category')['Price'].median().to_dict()
future_df['baseline_price'] = future_df['Category'].map(median_price)

baseline_revs = []
for i,row in future_df.iterrows():
    feat = future_enc.iloc[[i]].copy(); feat['Price'] = row['baseline_price']
    units_hat = max(0, xgb_model.predict(feat)[0])
    baseline_revs.append(units_hat*row['baseline_price']*(1-row['Discount']/100))
future_df['baseline_rev'] = baseline_revs

uplift = (future_df['best_rev'].sum()-future_df['baseline_rev'].sum())/future_df['baseline_rev'].sum()*100
print(f"Revenue uplift = {uplift:.2f}%")


daily = future_df.groupby('Date')[['best_rev','baseline_rev']].sum().reset_index()
plt.figure(figsize=(10,5))
plt.plot(daily['Date'], daily['baseline_rev'], label='Baseline')
plt.plot(daily['Date'], daily['best_rev'], label='Dynamic')
plt.legend(); plt.title("Revenue: Baseline vs Dynamic (next 30d)"); plt.show()
```

```
Revenue uplift = 11.83%
```



In [77]:
```
if uplift < 10:
    print("Uplift < 10% — trying expanded search (±40% and ±50%)")
    found = False
    best_candidate = None

    for try_range in [0.4, 0.5]:
        cand_df = future_df.copy()
        cand_df['best_price'] = np.nan
        cand_df['best_rev'] = np.nan

        for i,row in cand_df.iterrows():
            base_price = row.get('Competitor Pricing', row['Price'])
```

```
            cand_prices = np.linspace(base_price*(1-try_range), base_price*(1+try_range), 21)

            best_rev, best_price = -1, row['Price']
            for p in cand_prices:
                feat = future_enc.iloc[[i]].copy(); feat['Price'] = p
                units_hat = max(0, xgb_model.predict(feat)[0])
                rev = units_hat * p * (1 - row['Discount']/100)
                if rev > best_rev: best_rev, best_price = rev, p
            cand_df.loc[i,'best_price'] = best_price
            cand_df.loc[i,'best_rev'] = best_rev

        cand_uplift = (cand_df['best_rev'].sum()-cand_df['baseline_rev'].sum())/cand_df['baseline_rev'].sum()*100
        print(f"Range ±{int(try_range*100)}% → uplift {cand_uplift:.2f}%")

        if cand_uplift >= 10:
            print(f"✅ Found schedule with uplift ≥10% using ±{int(try_range*100)}%")
            future_df = cand_df.copy()
            uplift = cand_uplift
            found = True
            break

        # Track best candidate if still <10
        if best_candidate is None or cand_uplift > best_candidate['uplift']:
            best_candidate = {'df': cand_df.copy(), 'uplift': cand_uplift}

    if not found and best_candidate is not None:
        print(f" Could not reach ≥10% uplift. Returning best possible uplift: {best_candidate['uplift']:.2f}%")
        future_df = best_candidate['df']
        uplift = best_candidate['uplift']

print(f"Final uplift after fallback search: {uplift:.2f}%")
```

```
Final uplift after fallback search: 11.83%
```

In [ ]: