In [1…
```
import pandas as pd
import numpy as ny
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
```

The id is a unique identifier, each piece of data has a unique id; followed by the number of each type of item sold in each shop each day, and whether there is a discount (how much).

In [2…
```
# Load the data
df = pd.read_csv("Products_Information.csv")
```

In [3…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000888 entries, 0 to 3000887
Data columns (total 6 columns):
 #   Column         Dtype
---  ------         -----
 0   id             int64
 1   date           object
 2   store_nbr      int64
 3   product_type   object
 4   sales          float64
 5   special_offer  int64
dtypes: float64(1), int64(3), object(2)
memory usage: 137.4+ MB
```

Change the datatype of "date" to facilitate later manipulation of the date

In [4…
```
# Change the datatype of 'date'
df['date'] = pd.to_datetime(df['date'])
```

This dataset is of good quality and has a high degree of completeness. There are no duplicated rows and no missing values.

In [5]:
```
# Check data unique to avoid duplicate values
duplicate_rows = df.duplicated()
# Print duplicate rows
print(df[duplicate_rows])
# Print the neumber of duplicate rows
print("Number of duplicate rows:", duplicate_rows.sum())

# Check the missing values
print(df.isnull().sum())
```

```
Empty DataFrame
Columns: [id, date, store_nbr, product_type, sales, special_offer]
Index: []
Number of duplicate rows: 0
id              0
date            0
store_nbr       0
product_type    0
sales           0
special_offer   0
dtype: int64
```

Learn the basic statistic information.

In [6]:
```
# Check for negative values
columns_to_check = [2, 4, 5]
for column_idx in columns_to_check:
    column_name = df.columns[column_idx]
    has_negative_values = (df.iloc[:, column_idx] < 0).any()

    if has_negative_values:
        print(f"column: {column_idx + 1} ({column_name}) has negative values")
    else:
        print ("0")
```

```
0
0
0
```

In [7]:
```
# Avoid scientific counts
pd.set_option('display.float_format', lambda x: '%.3f' % x)
```

```
sales_offer = ['sales', 'special_offer']
print(df[sales_offer].describe())
```

```
            sales  special_offer
count 3000888.000    3000888.000
mean      357.776          2.603
std      1101.998         12.219
min         0.000          0.000
25%         0.000          0.000
50%        11.000          0.000
75%       195.847          0.000
max    124717.000        741.000
```

We found that the 75th percentile value of sales is 195.847. But the max value of sales is 124717, which can be seen as an extreme value. Additionally, we need to check the special_offer value corresponding to a sales amount of 0 to determine if it is an outlier and perform data replacement.

0 values

Firstly, we check the data which sales equals to 0 but special_offer is not 0.

In [8]:
```
data_0 = df[(df['sales'] == 0) & (df['special_offer'] != 0)]
print(data_0)
```

```
             id       date  store_nbr             product_type  sales  \
1348184  1348184 2015-01-29         37                   BEAUTY  0.000
2078169  2078169 2016-03-15         19  PLAYERS AND ELECTRONICS  0.000
2145487  2145487 2016-04-21          8            PERSONAL CARE  0.000
2172633  2172633 2016-05-07          2                GROCERY I  0.000
2172639  2172639 2016-05-07          2                HOME CARE  0.000
2172646  2172646 2016-05-07          2            PERSONAL CARE  0.000
2186323  2186323 2016-05-14         53                 CLEANING  0.000
2186331  2186331 2016-05-14         53       HOME AND KITCHEN I  0.000
2186332  2186332 2016-05-14         53      HOME AND KITCHEN II  0.000
2186337  2186337 2016-05-14         53                LINGERIE  0.000
2186341  2186341 2016-05-14         53            PERSONAL CARE  0.000
2577806  2577806 2016-12-20         38             FROZEN FOODS  0.000
2611440  2611440 2017-01-09         31                HOME CARE  0.000
2641500  2641500 2017-01-26         25       HOME AND KITCHEN I  0.000
2743345  2743345 2017-03-24         32         LIQUOR,WINE,BEER  0.000
2812806  2812806 2017-05-02         31                HOME CARE  0.000
2904214  2904214 2017-06-22         46      HOME AND KITCHEN II  0.000

         special_offer
1348184             1
2078169             1
2145487             1
2172633            63
2172639             5
2172646             3
2186323           279
2186331            16
2186332            10
2186337             1
2186341            88
2577806             2
2611440             5
2641500             2
2743345             1
2812806            16
2904214             1
```

We got three anomalies (their ids are 2172633, 2186323, 2186341), their discounts are huge but their sales are 0. And their product categories are "GROCERY I", "CLEANING", and "PERSONAL CARE", all of which are necessities. This is almost impossible in real life, so we need to process the three data.

I checked the sales in the seven days in the same store earlier and after the anomalous data separately, as well as the strength of the discounts.

In [9]:
```
data_G = df[(df['product_type'] == 'GROCERY I') & (df['store_nbr'] == 2) & ((df['date'] > '2016-04-30') & (df['date
print(data_G)
```

```
             id       date  store_nbr product_type      sales  special_offer
2161941  2161941 2016-05-01          2    GROCERY I   6768.000             68
2163723  2163723 2016-05-02          2    GROCERY I 124717.000             59
2165505  2165505 2016-05-03          2    GROCERY I   4323.000             75
2167287  2167287 2016-05-04          2    GROCERY I   4467.000             64
2169069  2169069 2016-05-05          2    GROCERY I   3746.000             77
2170851  2170851 2016-05-06          2    GROCERY I   4698.000             61
2172633  2172633 2016-05-07          2    GROCERY I      0.000             63
2174415  2174415 2016-05-08          2    GROCERY I   4380.000             56
2176197  2176197 2016-05-09          2    GROCERY I  12303.000             62
2177979  2177979 2016-05-10          2    GROCERY I   3288.000             61
2179761  2179761 2016-05-11          2    GROCERY I   3981.000             41
2181543  2181543 2016-05-12          2    GROCERY I   2966.000             33
2183325  2183325 2016-05-13          2    GROCERY I   3414.000             29
```

In [10]:
```
data_C = df[(df['product_type'] == 'CLEANING') & (df['store_nbr'] == 53) & ((df['date'] > '2016-05-07') & (df['date
print(data_C)
```

```
              id       date  store_nbr product_type    sales  special_offer
2175631  2175631 2016-05-08         53     CLEANING 1751.000            241
2177413  2177413 2016-05-09         53     CLEANING 1544.000            237
2179195  2179195 2016-05-10         53     CLEANING 6881.000            223
2180977  2180977 2016-05-11         53     CLEANING 1542.000            240
2182759  2182759 2016-05-12         53     CLEANING 1132.000            227
2184541  2184541 2016-05-13         53     CLEANING 1493.000            258
2186323  2186323 2016-05-14         53     CLEANING    0.000            279
2188105  2188105 2016-05-15         53     CLEANING 2556.000            276
2189887  2189887 2016-05-16         53     CLEANING 3757.000            243
2191669  2191669 2016-05-17         53     CLEANING 1618.000            252
2193451  2193451 2016-05-18         53     CLEANING 1504.000            237
2195233  2195233 2016-05-19         53     CLEANING 1586.000            264
2197015  2197015 2016-05-20         53     CLEANING 1936.000            255
```

In [11]:
```
data_P = df[(df['product_type'] == 'PERSONAL CARE') & (df['store_nbr'] == 53) & ((df['date'] > '2016-05-07') & (df[
print(data_P)
```

```
              id       date  store_nbr   product_type    sales  special_offer
2175649  2175649 2016-05-08         53  PERSONAL CARE  415.000             83
2177431  2177431 2016-05-09         53  PERSONAL CARE  327.000             72
2179213  2179213 2016-05-10         53  PERSONAL CARE 7504.000             67
2180995  2180995 2016-05-11         53  PERSONAL CARE  358.000             75
2182777  2182777 2016-05-12         53  PERSONAL CARE  262.000             73
2184559  2184559 2016-05-13         53  PERSONAL CARE  315.000             81
2186341  2186341 2016-05-14         53  PERSONAL CARE    0.000             88
2188123  2188123 2016-05-15         53  PERSONAL CARE  586.000             87
2189905  2189905 2016-05-16         53  PERSONAL CARE 4946.000             79
2191687  2191687 2016-05-17         53  PERSONAL CARE  372.000             83
2193469  2193469 2016-05-18         53  PERSONAL CARE 1408.000             81
2195251  2195251 2016-05-19         53  PERSONAL CARE  385.000             89
2197033  2197033 2016-05-20         53  PERSONAL CARE  429.000             87
```

All three days before and after the anomalies had similar discount strengths as the day of the anomalies, so we replaced the sales with the average of the sales for the three days earlier and after the anomalies.

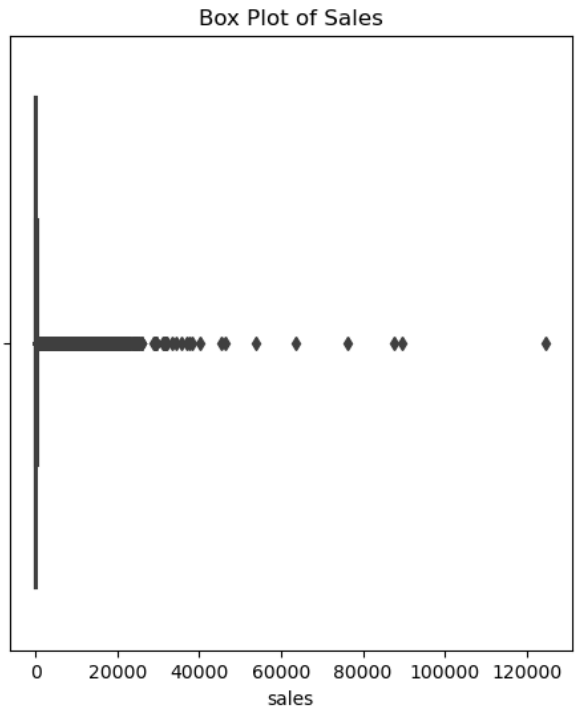
In [12]:
```
# GROCERY I
# Calculate the mean value
mean_sales_G = data_G[(data_G['date'] != '2016-05-07') & ((data_G['date'] >= '2016-05-04') & (data_G['date'] <= '20
# Replace the extreme value using mean value
df.loc[(df['product_type'] == 'GROCERY I') & (df['store_nbr'] == 2) & (df['date'] == '2016-05-07'), 'sales'] = mean
# Check if the replacement was successful
print(df[(df['product_type'] == 'GROCERY I') & (df['store_nbr'] == 2) & (df['date'] == '2016-05-07')])

# CLEANING
mean_sales_C = data_C[(data_C['date'] != '2016-05-14') & ((data_C['date'] >= '2016-05-11') & (data_C['date'] <= '20
df.loc[(df['product_type'] == 'CLEANING') & (df['store_nbr'] == 53) & (df['date'] == '2016-05-14'), 'sales'] = mean
print(df[(df['product_type'] == 'CLEANING') & (df['store_nbr'] == 53) & (df['date'] == '2016-05-14')])

# PERSONAL CARE
mean_sales_P = data_P[(data_P['date'] != '2016-05-14') & ((data_P['date'] >= '2016-05-11') & (data_P['date'] <= '20
df.loc[(df['product_type'] == 'PERSONAL CARE') & (df['store_nbr'] == 53) & (df['date'] == '2016-05-14'), 'sales'] =
print(df[(df['product_type'] == 'PERSONAL CARE') & (df['store_nbr'] == 53) & (df['date'] == '2016-05-14')])
```

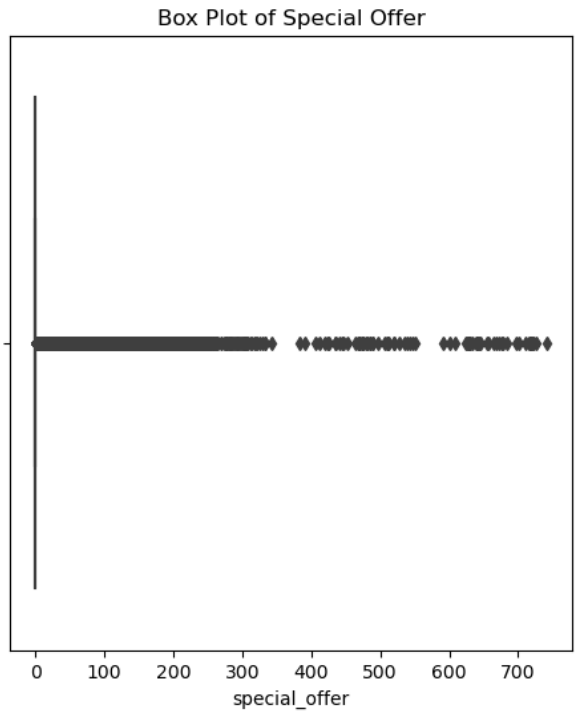
```
              id       date  store_nbr product_type    sales  special_offer
2172633  2172633 2016-05-07          2    GROCERY I 5480.333             63
              id       date  store_nbr product_type    sales  special_offer
2186323  2186323 2016-05-14         53     CLEANING 2016.333            279
              id       date  store_nbr   product_type    sales  special_offer
2186341  2186341 2016-05-14         53  PERSONAL CARE 1139.833             88
```

extreme value

We created box plots of these two features to check extreme values, which should not exist in a normal business dataset and we need to deal with them.

In [13]:
```
plt.figure(figsize=(12, 6))

# "sales" boxplot
plt.subplot(1, 2, 1)
sns.boxplot(x=df['sales'])
plt.title('Box Plot of Sales')

# "special_offer" boxplot
plt.subplot(1, 2, 2)
sns.boxplot(x=df['special_offer'])
plt.title('Box Plot of Special Offer')

plt.show()
```





As shown in box plots, The existence of values is reasonable. It illustrated that the special_offer does not have extreme value. But in the "sales" box plot, there are obvious extremes for "sales" and we need to check these values and consider the appropriate treatment for them.

We check the values of "sales" that are greater than 60000.

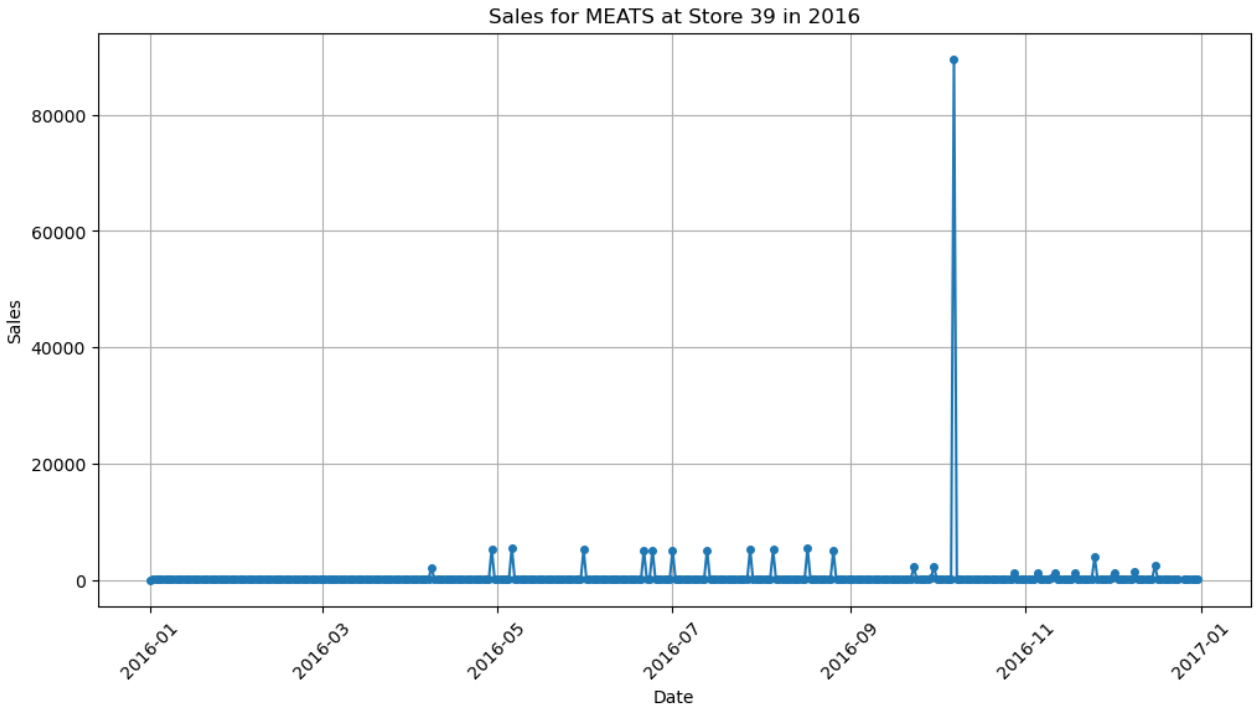
In [14]:
```
print(df[df['sales'] > 60000])
```

```
             id       date  store_nbr product_type      sales  special_offer
2139699  2139699 2016-04-18         45    GROCERY I  76090.000             38
2144154  2144154 2016-04-21         20    GROCERY I  87438.516             53
2153031  2153031 2016-04-26          2    GROCERY I  63434.000             30
2163723  2163723 2016-05-02          2    GROCERY I 124717.000             59
2445984  2445984 2016-10-07         39        MEATS  89576.360              0
```

As shown in the table, the first three higher sales figures each correspond to a higher level of special_offer.In constract,id2445984 has a sales of 89576.36, but special_offer has a value of 0. Moreover, more research is necessary in light of the remarkably high sales of id2163723.

1. In order to determine if the "sales" at id2445984 is extreme value, we examined the sales at store 39 "MEATS" across the board and plotted the bar chart.

In [21]:
```
# Creat the plot of sales - meats at Store39 in 2016
data_39meats_2016 = df[(df['product_type'] == 'MEATS') & (df['store_nbr'] == 39) & (df['date'].dt.year == 2016)]

plt.figure(figsize=(12, 6))
plt.plot(data_39meats_2016['date'], data_39meats_2016['sales'], marker='o', linestyle='-', markersize=4)
plt.title('Sales for MEATS at Store 39 in 2016')
plt.xlabel('Date')
plt.ylabel('Sales')

plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```



In the bar graph, we can clearly observe that the value of 89576.36 is an outlier and needs to be handled. In order to achieve this goal and because the most recent data is more indicative, we examined the sales and special offer of "MEATS" in shop 39 from October 2, 2016, to October 12, 2016 (five days earlier and later). Based on the results of the below query, we decided to take the mean value of sales with special_offer of 0 from 2016-10-02 to 2016-10-12(remove the outlier) to replace the extreme value 89576.360.

In [22]:
```
data_store39 = df[(df['product_type'] == 'MEATS') & (df['store_nbr'] == 39) & ((df['date'] >= '2016-10-02') & (df['
print(data_store39)
```

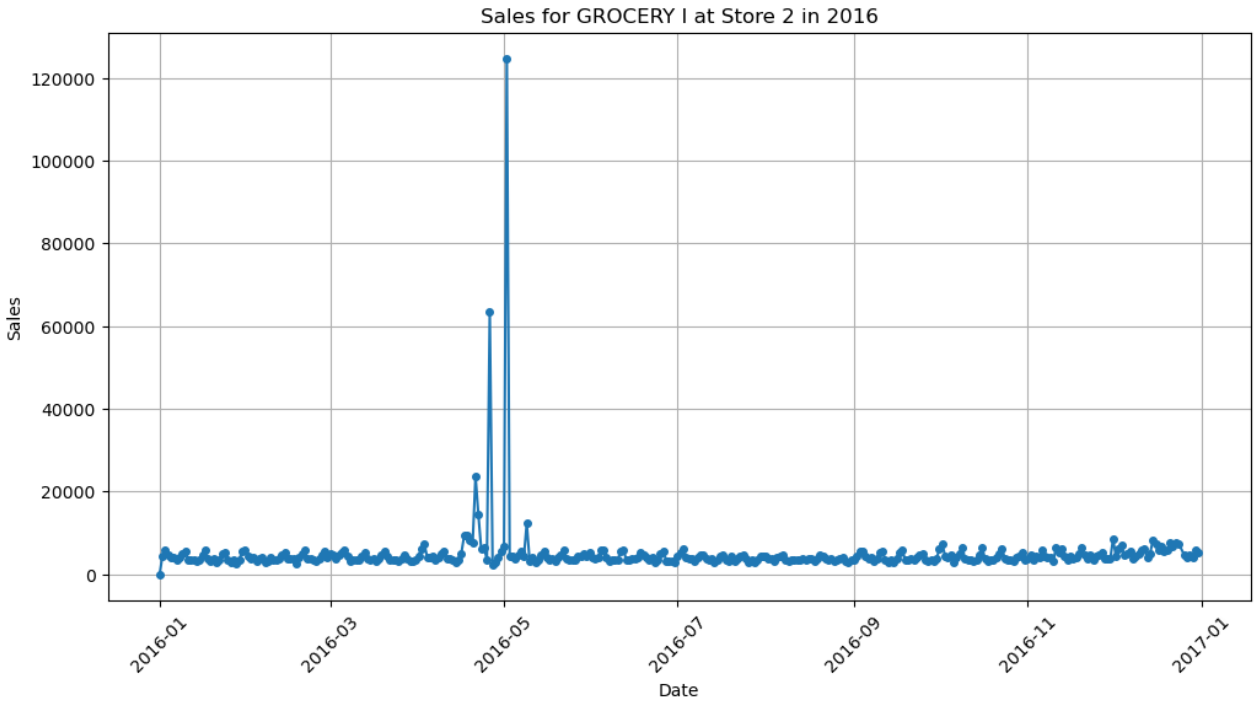
	id	date	store_nbr	product_type	sales	special_offer
2437074	2437074	2016-10-02	39	MEATS	231.323	0
2438856	2438856	2016-10-03	39	MEATS	164.107	0
2440638	2440638	2016-10-04	39	MEATS	183.746	0
2442420	2442420	2016-10-05	39	MEATS	119.624	0
2444202	2444202	2016-10-06	39	MEATS	223.671	22
2445984	2445984	2016-10-07	39	MEATS	89576.360	0
2447766	2447766	2016-10-08	39	MEATS	209.094	0
2449548	2449548	2016-10-09	39	MEATS	193.151	0
2451330	2451330	2016-10-10	39	MEATS	147.227	0
2453112	2453112	2016-10-11	39	MEATS	149.605	0
2454894	2454894	2016-10-12	39	MEATS	159.675	0

In [23]:
```
# Calculate the mean value
mean_sales = data_store39[(data_store39['special_offer'] == 0) & (data_store39['date'] != '2016-10-07')]['sales'].m

# Replace the extreme value using mean value
df.loc[(df['product_type'] == 'MEATS') & (df['store_nbr'] == 39) & (df['date'] == '2016-10-07'), 'sales'] = mean_sa

# Check if the replacement was successful
print(df[(df['product_type'] == 'MEATS') & (df['store_nbr'] == 39) & (df['date'] == '2016-10-07')])
```

	id	date	store_nbr	product_type	sales	special_offer
2445984	2445984	2016-10-07	39	MEATS	173.061	0

2.In order to determine if the "sales" at id2163723 is extreme value, we examined the sales at store 2 "GROCERY I" across the board and plotted the bar chart.

In [24]:
```
# Create the plot of sales - grocery I at Store2 in 2016
data_2grocery1_2016 = df[(df['product_type'] == 'GROCERY I') & (df['store_nbr'] == 2) & (df['date'].dt.year == 2016

plt.figure(figsize=(12, 6))
plt.plot(data_2grocery1_2016['date'], data_2grocery1_2016['sales'], marker='o', linestyle='-', markersize=4)
plt.title('Sales for GROCERY I at Store 2 in 2016')
plt.xlabel('Date')
plt.ylabel('Sales')

plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```



In the bar graph, the value of 124717 can be seen as an extreme value and needs to be handled. For this purpose, we checked the sales and the special_offer of "GROCERY I" from 2016-04-27 to 2016-05-07(5 days earlier and later) in store 39.

In [58]:
```
data_store2 = df[(df['product_type'] == 'GROCERY I') & (df['store_nbr'] == 2) & ((df['date'] >= '2016-04-27') & (df
print(data_store2)
```

```
              id       date  store_nbr product_type      sales  special_offer
2154813  2154813 2016-04-27          2    GROCERY I   2213.000             64
2156595  2156595 2016-04-28          2    GROCERY I   3023.000             61
2158377  2158377 2016-04-29          2    GROCERY I   3940.000             57
2160159  2160159 2016-04-30          2    GROCERY I   5466.000             66
2161941  2161941 2016-05-01          2    GROCERY I   6768.000             68
2163723  2163723 2016-05-02          2    GROCERY I 124717.000             59
2165505  2165505 2016-05-03          2    GROCERY I   4323.000             75
2167287  2167287 2016-05-04          2    GROCERY I   4467.000             64
2169069  2169069 2016-05-05          2    GROCERY I   3746.000             77
2170851  2170851 2016-05-06          2    GROCERY I   4698.000             61
2172633  2172633 2016-05-07          2    GROCERY I   5480.333             63
```

We decided to take the mean value of sales from 2016-04-27 to 2016-05-07(remove the outlier) to replace the extreme value 124717.000.

In [61]:
```
# Calculate the mean value
mean_sales = data_store2[(data_store2['date'] != '2016-05-02')]['sales'].mean()

# Replace the extreme value using mean value
df.loc[(df['product_type'] == 'GROCERY I') & (df['store_nbr'] == 2) & (df['date'] == '2016-05-02'), 'sales'] = mean

# Check if the replacement was successful
print(df[(df['product_type'] == 'GROCERY I') & (df['store_nbr'] == 2) & (df['date'] == '2016-05-02')])
```

```
              id       date  store_nbr product_type    sales  special_offer
2163723  2163723 2016-05-02          2    GROCERY I 4412.433             59
```

Change the datatype of "product_type"

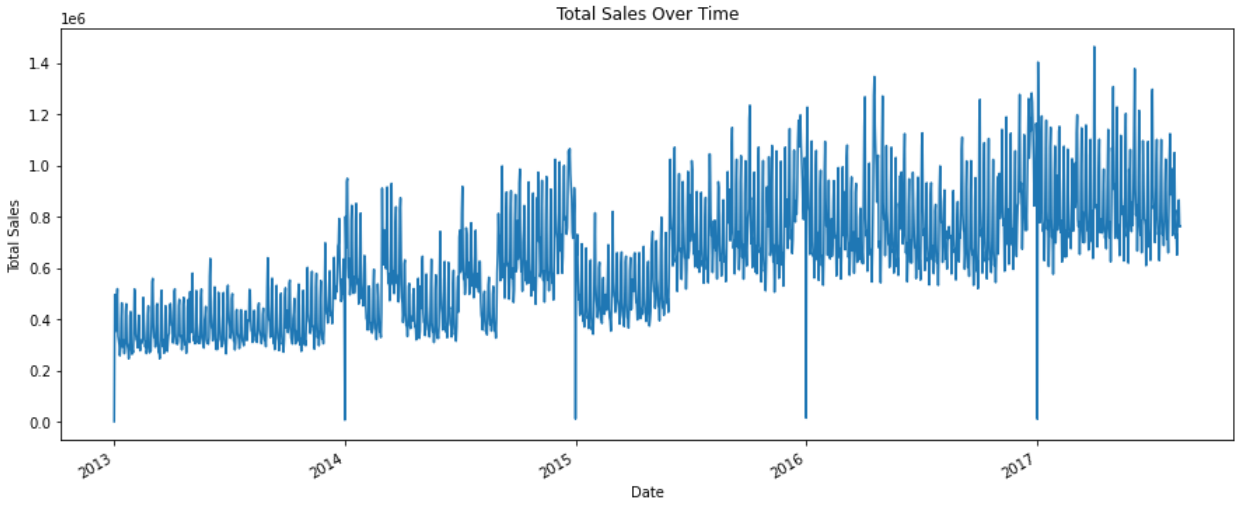
For subsequent model predictions, we replaced the different product_types in the dataset with numbers.

In [65]:
```
# Using LabelEncoder
label_encoder = LabelEncoder()
df['product_type_encoded'] = label_encoder.fit_transform(df['product_type'])

# Print the encoded results along with the original product_type values
print(df[['product_type', 'product_type_encoded']].drop_duplicates())
```

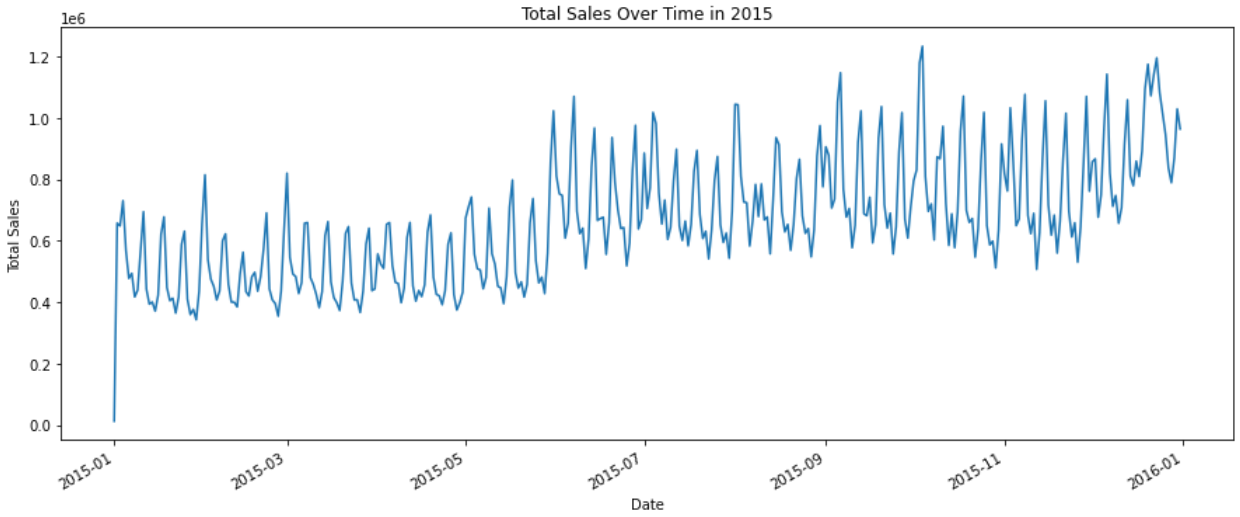
	product_type	product_type_encoded
0	AUTOMOTIVE	0
1	BABY CARE	1
2	BEAUTY	2
3	BEVERAGES	3
4	BOOKS	4
5	BREAD/BAKERY	5
6	CELEBRATION	6
7	CLEANING	7
8	DAIRY	8
9	DELI	9
10	EGGS	10
11	FROZEN FOODS	11
12	GROCERY I	12
13	GROCERY II	13
14	HARDWARE	14
15	HOME AND KITCHEN I	15
16	HOME AND KITCHEN II	16
17	HOME APPLIANCES	17
18	HOME CARE	18
19	LADIESWEAR	19
20	LAWN AND GARDEN	20
21	LINGERIE	21
22	LIQUOR,WINE,BEER	22
23	MAGAZINES	23
24	MEATS	24
25	PERSONAL CARE	25
26	PET SUPPLIES	26
27	PLAYERS AND ELECTRONICS	27
28	POULTRY	28
29	PREPARED FOODS	29
30	PRODUCE	30
31	SCHOOL AND OFFICE SUPPLIES	31
32	SEAFOOD	32

The data from many years ago, which is typically inaccurate, has little bearing on business strategies from the standpoint of store operation.As a result, we kept the data from 2014 through 2017 and eliminated the data from 2013. Meanwhile, our aim is to predict sales from 2017-07-31 to 2017-08-15. Data that is closer to the predicted date will show people's consumption patterns and the status of the economy more accurately. We therefore set the data for the year preceding the predicted date (from 2016-07-31 to 2017-07-30) as testing dataset, and set the data from 2014-01-01 to 2016-07-31 as training dataset. In the modelling, features are "date", "store_nrb", "product_type" and "special_offer". And the target is "sales".

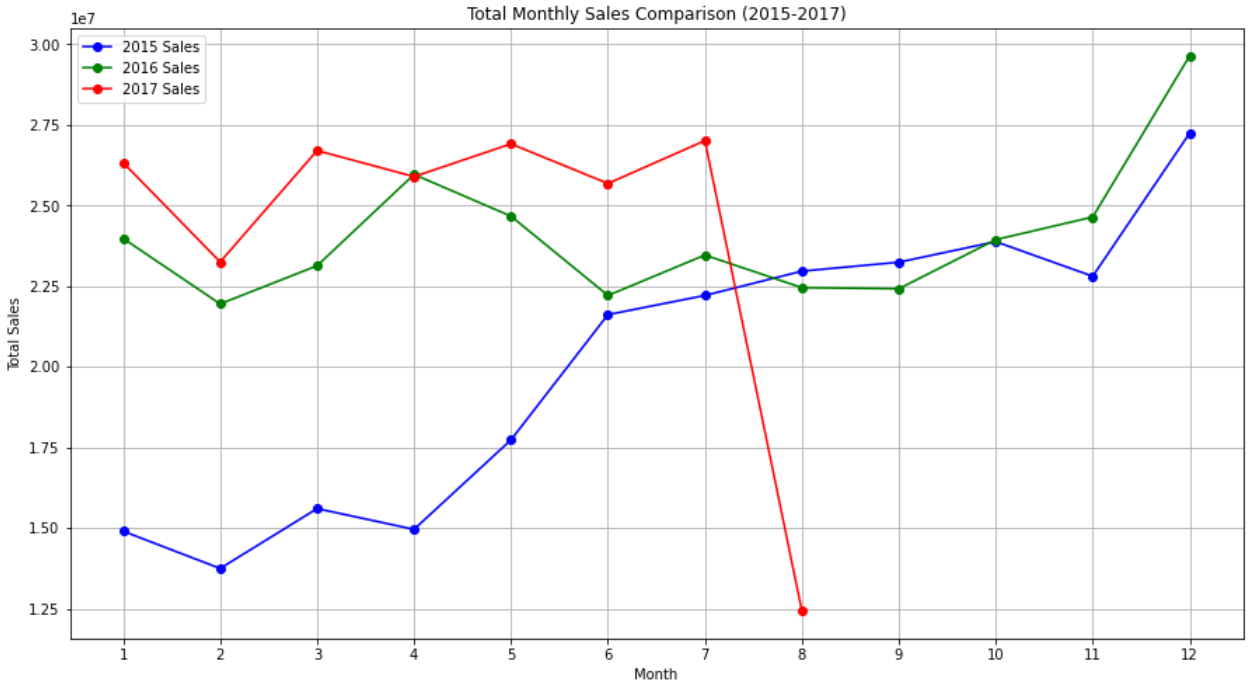
In [68]:
```
# Plot total sales over time
total_sales_over_time = df.groupby('date')['sales'].sum()
plt.figure(figsize=(15, 6))
total_sales_over_time.plot()
plt.title('Total Sales Over Time')
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.show()
```



In [70]:
```
# Plot the sales over time in 2015
sales_2015 = df[df['date'].dt.year == 2015].groupby('date')['sales'].sum()
plt.figure(figsize=(15, 6))
sales_2015.plot()
plt.title('Total Sales Over Time in 2015')
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.show()
```

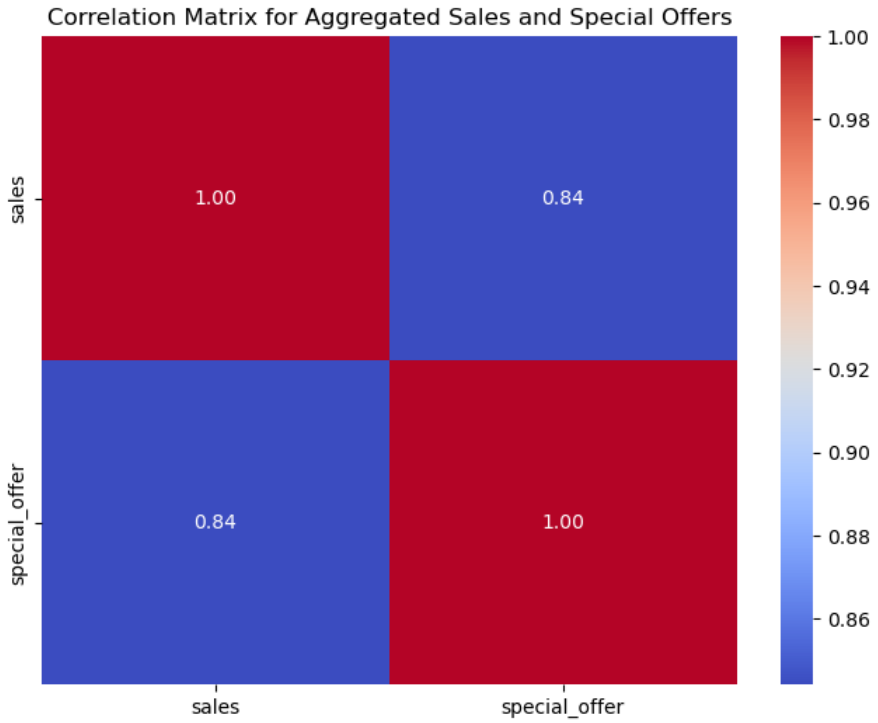


In [72]:
```
# Compare the sales in 2015, 2016 and 2017
# Generate the sales of month
sales_2015_monthly = sales_2015.resample('MS').sum()
sales_2016_monthly = df[df['date'].dt.year == 2016].groupby('date')['sales'].sum().resample('MS').sum()
sales_2017_monthly = df[df['date'].dt.year == 2017].groupby('date')['sales'].sum().resample('MS').sum()

# Create plots
plt.figure(figsize=(15, 8))
plt.plot(sales_2015_monthly.index.month, sales_2015_monthly, label='2015 Sales', color='blue', marker='o')
plt.plot(sales_2016_monthly.index.month, sales_2016_monthly, label='2016 Sales', color='green', marker='o')
plt.plot(sales_2017_monthly.index.month, sales_2017_monthly, label='2017 Sales', color='red', marker='o')

plt.title('Total Monthly Sales Comparison (2015-2017)')
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.legend()
plt.grid(True)
plt.xticks(range(1, 13))
plt.show()
```



In [25]:
```
filtered_data = df[(df['date'] >= '2015-06-01')]
# Group by store number and product type and calculating the sum of sales and special offers
aggregated_data = filtered_data.groupby(['store_nbr', 'product_type']).agg({'sales': 'sum', 'special_offer': 'sum'}

# Calculate the correlation matrix for the aggregated data
correlation_matrix_aggregated = aggregated_data[['sales', 'special_offer']].corr()

# Plot the correlation matrix for aggregated data
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix_aggregated, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix for Aggregated Sales and Special Offers")
```

Out[25]:
```
Text(0.5, 1.0, 'Correlation Matrix for Aggregated Sales and Special Offers')
```



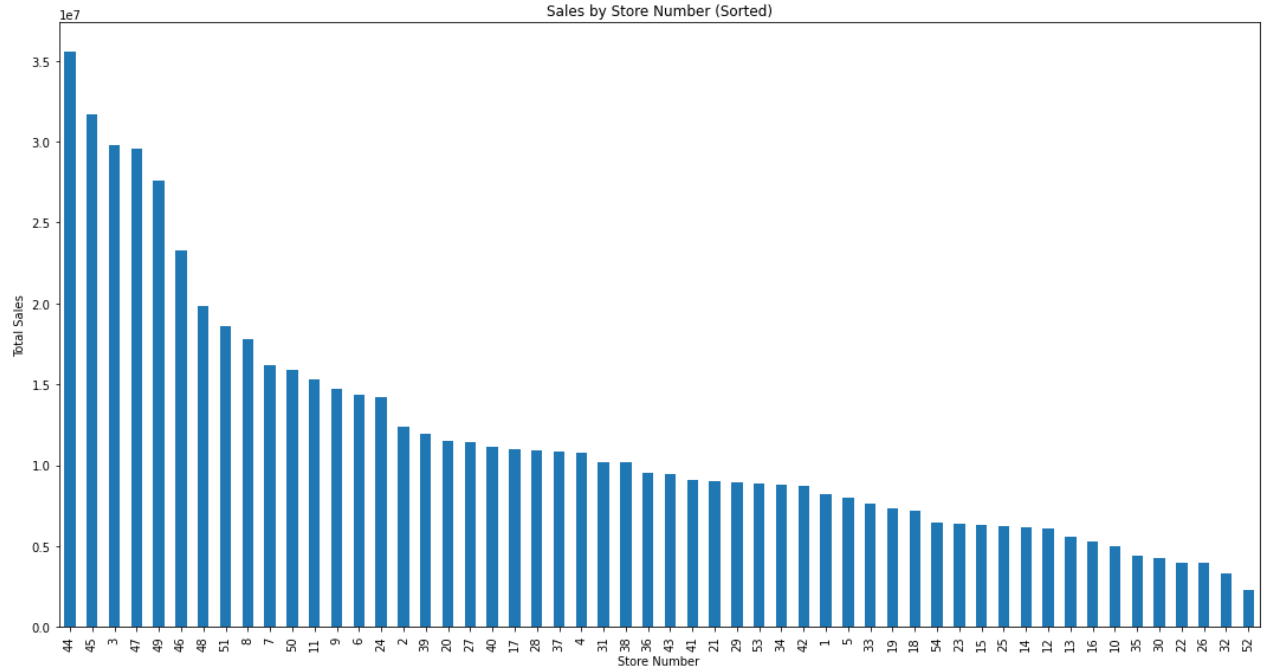
Grouping by shop and product type: we first group the data by shop number (store_nbr) and product type (product_type). This means that the data is divided into subsets, where each subset contains all the records for a particular shop and a particular product type.

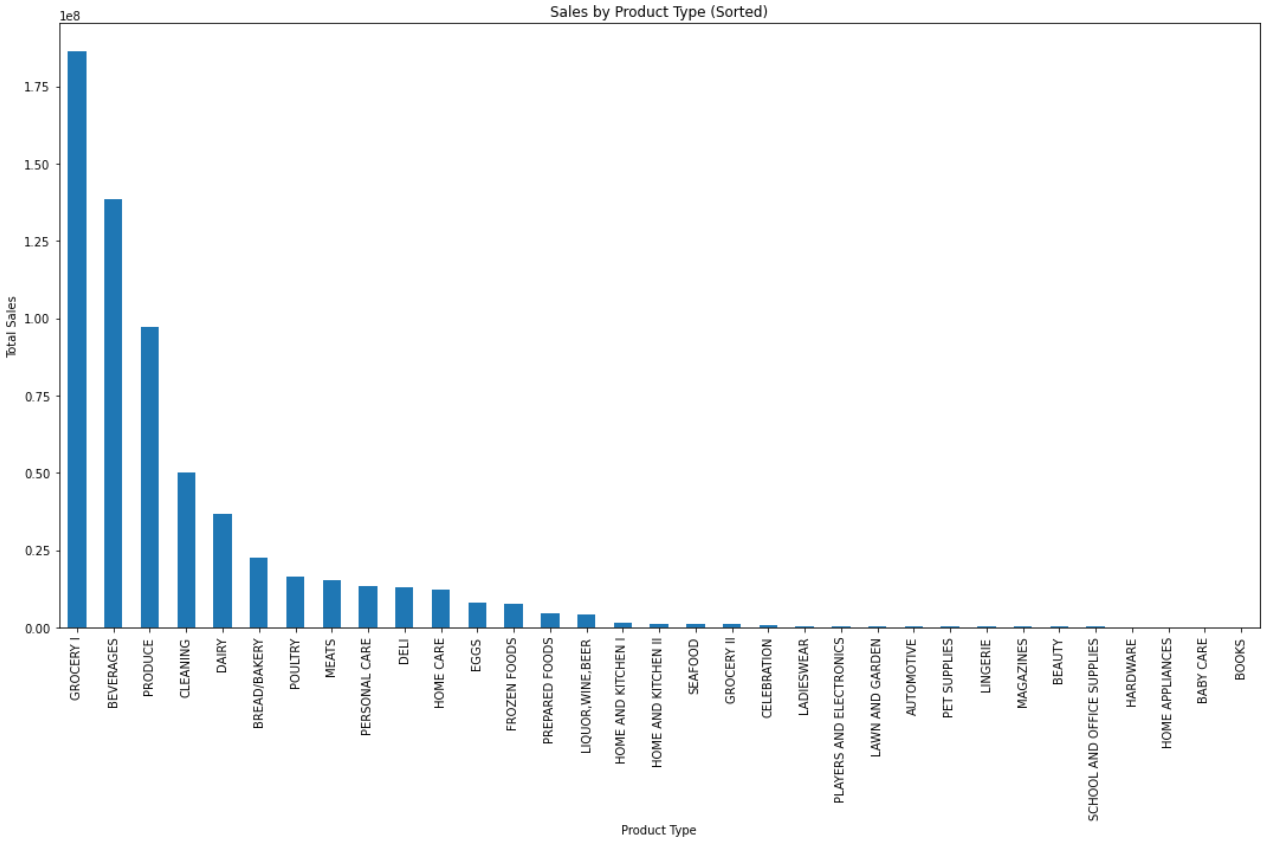
Aggregating sales and special offers: for each such subset (for each product type in each shop), we calculated the sum of sales (sales) and special offers (special_offer). This corresponds to calculating the total sales and the total special_offers for each product type in each shop for the entire time period considered.

Reset index: after aggregation, in order to keep the data actionable and clear, we used the reset_index() method. This restores the data structure to a more traditional tabular format for subsequent analysis.

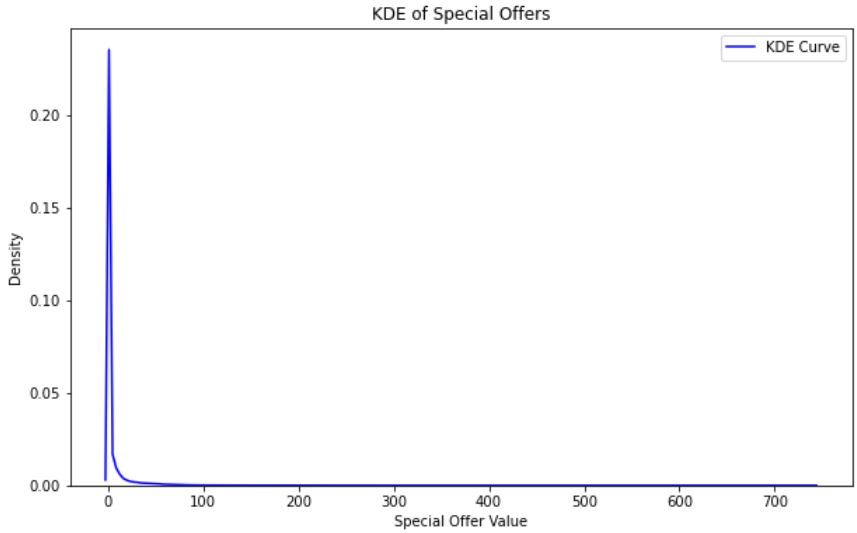
The data was considered at a higher level, i.e. aggregating the total sales for each shop and product type combination. Doing so amounts to assessing the relationship between overall sales trends and special offers across different shop and product type combinations. The results of the analyses of the aggregated data are more representative of the overall trends.

In [77]:
```
store_sales = filtered_data.groupby('store_nbr')['sales'].sum()
product_sales = filtered_data.groupby('product_type')['sales'].sum()

# Sorting sales data
sorted_store_sales = store_sales.sort_values(ascending=False)
sorted_product_sales = product_sales.sort_values(ascending=False)

# Plotting
fig, axs = plt.subplots(2, 1, figsize=(15, 18))
sorted_store_sales.plot(ax=axs[0], kind='bar', title='Sales by Store Number (Sorted)')
axs[0].set_xlabel('Store Number')
axs[0].set_ylabel('Total Sales')
sorted_product_sales.plot(ax=axs[1], kind='bar', title='Sales by Product Type (Sorted)')
axs[1].set_xlabel('Product Type')
axs[1].set_ylabel('Total Sales')

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

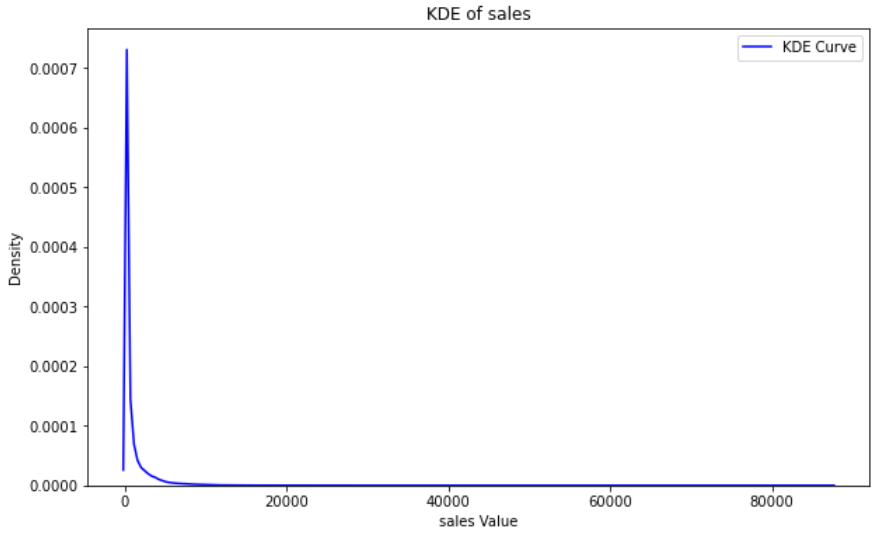




In [79]:
```
plt.figure(figsize=(10, 6))

# Plot the KDE (Kernel Density Estimate) for 'special_offer'
sns.kdeplot(filtered_data['special_offer'], color='blue', label='KDE Curve')

plt.title('KDE of Special Offers')
plt.xlabel('Special Offer Value')
plt.ylabel('Density')
plt.legend()

plt.show()
```



In [82]:
```
plt.figure(figsize=(10, 6))

# Plot the KDE (Kernel Density Estimate) for 'special_offer'
sns.kdeplot(filtered_data['sales'], color='blue', label='KDE Curve')

plt.title('KDE of sales')
plt.xlabel('sales Value')
plt.ylabel('Density')
plt.legend()

plt.show()
```



Since both the sale and special_offer features show high skewness and potential extreme values, which will have a greater impact on model training and cause the model to focus more on the parts with a larger range of values, we chose to apply MinMaxScaler to scale the sale and special_offer. This processing method scales all the values to a range between 0 and 1, preserving the distributional characteristics of the original data, while making the feature values on the same scale, which is conducive to subsequent data analysis and model training.

In [91]:
```
pd.set_option('display.float_format', lambda x: f'{x:.6f}')
# Select the 'sales' and 'special_offer' columns for MinMax scaling
data_to_scale = filtered_data[['sales', 'special_offer']]

# Create a MinMaxScaler object
min_max_scaler = MinMaxScaler()

# Scale the dadta
scaled_data = min_max_scaler.fit_transform(data_to_scale)

# Convert scaled data to string format with six decimal places
scaled_data_str = pd.DataFrame(scaled_data, columns=['sales_scaled', 'special_offer_scaled']).applymap(lambda x: f'

# Reset the index of filtered_data to ensure proper alignment during concatenation
filtered_data_reset = filtered_data.reset_index(drop=True)

# Concatenate the string formatted scaled data with the original filtered_data
concatenated_data_str = pd.concat([filtered_data_reset, scaled_data_str], axis=1)

# Convert the last two columns (string formatted scaled data) back to numeric format with six decimal places
concatenated_data_str['sales_scaled'] = concatenated_data_str['sales_scaled'].astype(float)
```

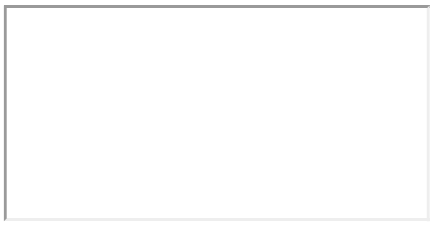
```
concatenated_data_str['special_offer_scaled'] = concatenated_data_str['special_offer_scaled'].astype(float)

# Display the first few rows of the data to confirm the rounding
concatenated_data_str
```

Out[91]:

	id	date	store_nbr	product_type	sales	special_offer	product_type_encoded	sales_scaled	special_offer_scaled
0	1566378	2015-06-01	1	AUTOMOTIVE	1.000000	0	0	0.000011	0.000000
1	1566379	2015-06-01	1	BABY CARE	0.000000	0	1	0.000000	0.000000
2	1566380	2015-06-01	1	BEAUTY	6.000000	0	2	0.000069	0.000000
3	1566381	2015-06-01	1	BEVERAGES	2210.000000	0	3	0.025275	0.000000
4	1566382	2015-06-01	1	BOOKS	0.000000	0	4	0.000000	0.000000
...	...	...	...	...	...	...	...	...	...
1405993	2972371	2017-07-30	9	POULTRY	517.911000	1	28	0.005923	0.001350
1405994	2972372	2017-07-30	9	PREPARED FOODS	145.490000	1	29	0.001664	0.001350
1405995	2972373	2017-07-30	9	PRODUCE	1882.588000	7	30	0.021530	0.009447
1405996	2972374	2017-07-30	9	SCHOOL AND OFFICE SUPPLIES	41.000000	8	31	0.000469	0.010796
1405997	2972375	2017-07-30	9	SEAFOOD	19.909000	0	32	0.000228	0.000000

1405998 rows × 9 columns

Considering that this is a business decision-making problem, we have used both year-on-year and chain-on-year approaches for modelling training and predicting. And since different products and stores may have different seasonal and cyclical sales patterns, we will use the previous 15/30/90 days historical data and last year's same period data to make predictions and find the optimal model respectively.



In [7…

```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, explained_variance_score,
```

In [8…

```
df_model = pd.read_csv("/Volumes/Manchester/Sem1/BMAN73701/coursework/Products_Information_Proc
```

In [9]:

```
# Using the fixed window method
# Assuming 'df_model' is your DataFrame
df_model['date'] = pd.to_datetime(df_model['date'])

# Filter historical data for the specified date ranges
historical_data = df_model[((df_model['date'] >= '2015-05-02') & (df_model['date'] <= '2015-07-3
                        ((df_model['date'] >= '2016-05-02') & (df_model['date'] <= '2016-07-30'))
                        ((df_model['date'] >= '2017-05-02') & (df_model['date'] <= '2017-07-30'))]

'''
Other date ranges:
# Filter historical data for the specified date ranges
# 30-day
historical_data = df_model[((df_model['date'] >= '2015-07-01') & (df_model['date'] <= '2015-07-
                            ((df_model['date'] >= '2016-07-01') & (df_model['date'] <= '2016-07-
                            ((df_model['date'] >= '2017-07-01') & (df_model['date'] <= '2017-07-

# 15-day
historical_data = df_model[((df_model['date'] >= '2015-07-16') & (df_model['date'] <= '2015-07-
                            ((df_model['date'] >= '2016-07-16') & (df_model['date'] <= '2016-07-
                            ((df_model['date'] >= '2017-07-16') & (df_model['date'] <= '2017-07-
'''

# Extract features and target variables
features = ['store_nbr', 'product_type_encoded', 'special_offer_scaled']
target = 'sales_scaled'

# Assuming 'features' and 'target' are defined earlier in your code
X = historical_data[features]
y = historical_data[target]

# Split the dataset into training and validation sets
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the KNN model
knn_model = KNeighborsRegressor()
knn_model.fit(X_train, y_train)

# Predictions on the validation set
y_valid_pred = knn_model.predict(X_valid)

# Calculate various indicators for model evaluation
mse = mean_squared_error(y_valid, y_valid_pred)
mae = mean_absolute_error(y_valid, y_valid_pred)
explained_variance = explained_variance_score(y_valid, y_valid_pred)
r2 = r2_score(y_valid, y_valid_pred)

# Calculate MSE using cross-validation
cv_mse_scores = cross_val_score(knn_model, X, y, scoring='neg_mean_squared_error', cv=5)
cv_mse_mean = -cv_mse_scores.mean()

# Print performance indicators for the validation set
print(f'Validation Mean Squared Error (MSE): {mse}')
print(f'Validation Mean Absolute Error (MAE): {mae}')
print(f'Validation Explained Variance Score: {explained_variance}')
print(f'Validation R-squared: {r2}')
print(f'Validation Cross-Validation Mean Squared Error (MSE): {cv_mse_mean}')

# Define the future prediction date range
final_prediction_start_date = '2017-07-31'
final_prediction_end_date = '2017-08-15'

# Select data for the next 16 days
future_data = df_model[(df_model['date'] >= final_prediction_start_date) & (df_model['date'] <= final_prediction_en

# Forecast for the next 16 days
future_predictions = knn_model.predict(future_data)

# Add predicted values to the original data
df_model.loc[(df_model['date'] >= final_prediction_start_date) & (df_model['date'] <= final_prediction_end_date), '

# Print results or perform other subsequent processing
print(df_model[['date', 'store_nbr', 'product_type_encoded', 'predicted_sales_scaled']])

# Extract the sales values of the next 16 days from the data
y_future = df_model[(df_model['date'] >= final_prediction_start_date) & (df_model['date'] <= final_prediction_end_d

# Future 16 days predictions
```

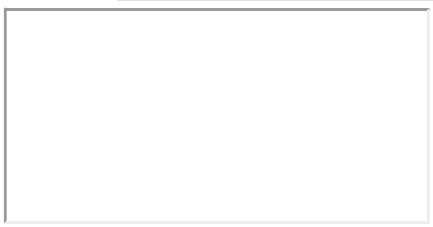
```
y_future_pred = future_predictions

# Calculate mean squared error for the future predictions
mse_future = mean_squared_error(y_future, y_future_pred)

# Calculate explained variance score for the future predictions
explained_variance_future = explained_variance_score(y_future, y_future_pred)

# Calculate mean absolute error for the future predictions
mae_future = mean_absolute_error(y_future, y_future_pred)

# Calculate R-squared value for the future predictions
r2_future = r2_score(y_future, y_future_pred)

# Calculate MSE using cross-validation for the future predictions
cv_mse_scores_future = cross_val_score(knn_model, future_data, y_future, scoring='neg_mean_squared_error', cv=5)
cv_mse_mean_future = -cv_mse_scores_future.mean()

# Print performance indicators for the future predictions
print(f'Future Mean Squared Error (MSE): {mse_future}')
print(f'Future Explained Variance Score: {explained_variance_future}')
print(f'Future Mean Absolute Error (MAE): {mae_future}')
print(f'Future R-squared: {r2_future}')
print(f'Future Cross-Validation Mean Squared Error (MSE): {cv_mse_mean_future}')
```

```
Validation Mean Squared Error (MSE): 1.969600867023054e-05
Validation Mean Absolute Error (MAE): 0.0012078998901047513
Validation Explained Variance Score: 0.9000327817724086
Validation R-squared: 0.9000327110327617
Validation Cross-Validation Mean Squared Error (MSE): 2.0561935449072206e-05
              date  store_nbr  product_type_encoded  predicted_sales_scaled
0       2015-06-01          1                     0                     NaN
1       2015-06-01          1                     1                     NaN
2       2015-06-01          1                     2                     NaN
3       2015-06-01          1                     3                     NaN
4       2015-06-01          1                     4                     NaN
...            ...        ...                   ...                     ...
1434505 2017-08-15          9                    28                0.005558
1434506 2017-08-15          9                    29                0.000969
1434507 2017-08-15          9                    30                0.024676
1434508 2017-08-15          9                    31                0.000098
1434509 2017-08-15          9                    32                0.000134

[1434510 rows x 4 columns]
Future Mean Squared Error (MSE): 1.5388982466662457e-05
Future Explained Variance Score: 0.9246530637986482
Future Mean Absolute Error (MAE): 0.0011345881383277216
Future R-squared: 0.9245030324816859
Future Cross-Validation Mean Squared Error (MSE): 1.2648884386405526e-05
```

In [ ]:

```
# Filter data for the required time period (July 2017)
'''
Other date ranges:
# Filter historical data for the specified date ranges
# 30-day
start_date = '2017-07-01'
end_date = '2017-08-15'

# 15-day
start_date = '2017-07-16'
end_date = '2017-08-15'
'''

start_date = '2017-05-02'
end_date = '2017-08-15'
data_filtered = df_model[(df_model['date'] >= start_date) & (df_model['date'] <= end_date)].copy()

# Convert date to datetime object for easier manipulation
data_filtered['date'] = pd.to_datetime(data_filtered['date'])

# Define the window size
window_size = 31  # days

'''
Other window size:
# 30-day
window_size = 31  # days

# 15-day
window_size = 16  # days
'''

# Create an empty DataFrame to store prediction results and performance indicators
predictions_df = pd.DataFrame(columns=['date', 'store_nbr', 'product_type_encoded', 'predicted_sales_scaled'])
performance_metrics_df = pd.DataFrame(columns=['start_date', 'end_date', 'MSE', 'MAE', 'Explained_Variance', 'R2',

# Define a range of k values to try
k_values = [3, 5, 7, 9, 11]  # Add more values as needed

# Create an empty DataFrame to store performance metrics for different k values
k_performance_metrics_df = pd.DataFrame(columns=['k', 'MSE', 'MAE', 'Explained_Variance', 'R2', 'CV_MSE'])
```

```
# Iterate over different k values
for k in k_values:
    # Use sliding window for prediction
    for end_date in pd.date_range(start='2017-07-31', end='2017-08-15'):
        start_date = end_date - pd.DateOffset(days=window_size - 1)

        # Select data within the sliding window
        window_data = data_filtered[(data_filtered['date'] >= start_date) & (data_filtered['date'] <= end_date)][['

        # Check if there is data within the window
        if not window_data.empty:
            # Extract features and target variables for the window
            X_window = window_data[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]
            y_window = window_data['sales_scaled']

            # Train model using cross-validation
            knn_model = KNeighborsRegressor(n_neighbors=k)
            cv_mse_scores = cross_val_score(knn_model, X_window, y_window, scoring='neg_mean_squared_error', cv=5)
            cv_mse_mean = -cv_mse_scores.mean()

            # Train model on the entire window
            knn_model.fit(X_window, y_window)

            # Forecast sales value
            predictions = knn_model.predict(X_window)

            # Add predicted values to the result DataFrame
            window_data['predicted_sales_scaled'] = predictions
            predictions_df = pd.concat([predictions_df, window_data])

            # Calculate performance indicators
            mse = mean_squared_error(y_window, predictions)
            mae = mean_absolute_error(y_window, predictions)
            explained_variance = explained_variance_score(y_window, predictions)
            r2 = r2_score(y_window, predictions)

            # Add performance indicators to the performance indicators DataFrame
            performance_metrics_df = pd.concat([performance_metrics_df, pd.DataFrame({
                'start_date': [start_date],
                'end_date': [end_date],
                'MSE': [mse],
                'MAE': [mae],
                'Explained_Variance': [explained_variance],
                'R2': [r2],
                'CV_MSE': [cv_mse_mean]
            })])
        else:
            # Handle the case when window_data is empty, e.g., set performance metrics to some default values
            # You may need to decide what values make sense as defaults in your specific case
            pass

        # Add performance indicators to the k_performance_metrics DataFrame
        k_performance_metrics_df = pd.concat([k_performance_metrics_df, pd.DataFrame({
            'k': [k],
            'MSE': [mse],
            'MAE': [mae],
            'Explained_Variance': [explained_variance],
            'R2': [r2],
            'CV_MSE': [cv_mse_mean]
        })])

# Print the final prediction results
print(predictions_df[['date', 'store_nbr', 'product_type_encoded', 'predicted_sales_scaled']])

# Print final performance indicators
print(performance_metrics_df)

# Print performance metrics for different k values
print(k_performance_metrics_df)

# Calculate the mean performance metrics for each k value
mean_performance_metrics_df = k_performance_metrics_df.groupby('k').mean().reset_index()

# Print mean performance metrics for each k value
print("Mean Performance Metrics for Each k Value:")
print(mean_performance_metrics_df)
```



In [4…

```
import pandas as pd
import numpy as np
from datetime import datetime
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error,explained_variance_score, r2_scor
import time
```

In [43]:

```
# Load the dataset
data = concatenated_data_str

# Convert the date column to datetime format
data['date'] = pd.to_datetime(data['date'])

# Choose data to train for a specific year and a 15-day window
train_data = data[((data['date'] >= '2015-07-16') & (data['date'] <= '2015-07-30')) |
                  ((data['date'] >= '2016-07-16') & (data['date'] <= '2016-07-30')) |
                  ((data['date'] >= '2017-07-16') & (data['date'] <= '2017-07-30'))]

# Selec data from July 31 to August 15, 2017 to validate
test_data = data[(data['date'] >= '2017-07-31') & (data['date'] <= '2017-08-15')]

# Split train and test data into features and target
features = ['store_nbr', 'product_type_encoded', 'special_offer']
X_train = train_data[features]
y_train = train_data['sales_scaled']
X_test = test_data[features]
y_test = test_data['sales_scaled']

print("Training started!")

# Train the Gradient Boosting model
start_time = time.time()
model = GradientBoostingRegressor(random_state=42)
model.fit(X_train, y_train)
end_time = time.time()

# Predict on the validation set
y_pred = model.predict(X_test)
print("Training time:", end_time - start_time) # Output the time spent on training

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
explained_variance = explained_variance_score(y_test, y_pred)

# Evaluate model performance using cross-validation on the training set
cv_mse_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
cv_mse_scores = -cv_mse_scores
cv_mse_mean = np.mean(cv_mse_scores)

# Output the evaluation metrics
print("Validation Set Metrics (with 15-day Windows) :")
print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R²):", r2)
print("Explained Variance:", explained_variance)
print("\nCross-Validation MSE Scores:", cv_mse_scores)
print("Average Cross-Validation MSE:", cv_mse_mean)
```

```
Training started!
Training time: 4.119743347167969
Validation Set Metrics (with 15-day Windows) :
Mean Squared Error (MSE): 4.519848287335364e-05
Mean Absolute Error (MAE): 0.0027129855370867325
R-squared (R²): 0.778260297536961
Explained Variance: 0.778416741722679

Cross-Validation MSE Scores: [5.40839127e-05 4.17404036e-05 3.00751706e-05 4.00401814e-05
 5.62793288e-05]
Average Cross-Validation MSE: 4.444379940844349e-05
```

In [44]:

```
# Load the dataset
data = concatenated_data_str

# Convert the date column to datetime format
data['date'] = pd.to_datetime(data['date'])

# Choose data to train for a specific year and a 30-day window
train_data = data[((data['date'] >= '2015-07-01') & (data['date'] <= '2015-07-30')) |
                  ((data['date'] >= '2016-07-01') & (data['date'] <= '2016-07-30')) |
                  ((data['date'] >= '2017-07-01') & (data['date'] <= '2017-07-30'))]

# Selec data from July 31 to August 15, 2017 to validate
test_data = data[(data['date'] >= '2017-07-31') & (data['date'] <= '2017-08-15')]

# Split train and test data into features and target
features = ['store_nbr', 'product_type_encoded', 'special_offer']
X_train = train_data[features]
y_train = train_data['sales_scaled']
X_test = test_data[features]
```

```
y_test = test_data['sales_scaled']

print("Training started!")

# Train the Gradient Boosting model
start_time = time.time()
model = GradientBoostingRegressor(random_state=42)
model.fit(X_train, y_train)
end_time = time.time()

# Predict on the validation set
y_pred = model.predict(X_test)
print("Training time:", end_time - start_time) # Output the time spent on training

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
explained_variance = explained_variance_score(y_test, y_pred)


# Evaluate model performance using cross-validation on the training set
cv_mse_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
cv_mse_scores = -cv_mse_scores
cv_mse_mean = np.mean(cv_mse_scores)

# Output the evaluation metrics
print("Validation Set Metrics (with 30-day Window) :")
print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R²):", r2)
print("Explained Variance:", explained_variance)
print("\nCross-Validation MSE Scores:", cv_mse_scores)
print("Average Cross-Validation MSE:", cv_mse_mean)
```

```
Training started!
Training time: 9.59675121307373
Validation Set Metrics (with 30-day Window) :
Mean Squared Error (MSE): 4.3246383336177036e-05
Mean Absolute Error (MAE): 0.0027264749566973043
R-squared (R²): 0.7878371227539632
Explained Variance: 0.7882263726248014

Cross-Validation MSE Scores: [8.24474648e-05 5.38978619e-05 3.58786535e-05 5.16666425e-05
 4.72936691e-05]
Average Cross-Validation MSE: 5.423685837002717e-05
```

In [45]:

```
# Load the dataset
data = concatenated_data_str

# Convert the date column to datetime format
data['date'] = pd.to_datetime(data['date'])

# Choose data to train for a specific year and a 90-day window
train_data = data[((data['date'] >= '2015-05-02') & (data['date'] <= '2015-07-30')) |
                  ((data['date'] >= '2016-05-02') & (data['date'] <= '2016-07-30')) |
                  ((data['date'] >= '2017-05-02') & (data['date'] <= '2017-07-30'))]

# Selec data from July 31 to August 15, 2017 to validate
test_data = data[(data['date'] >= '2017-07-31') & (data['date'] <= '2017-08-15')]

# Split train and test data into features and target
features = ['store_nbr', 'product_type_encoded', 'special_offer']
X_train = train_data[features]
y_train = train_data['sales_scaled']
X_test = test_data[features]
y_test = test_data['sales_scaled']

print("Training started!")

# Train the Gradient Boosting model
start_time = time.time()
model = GradientBoostingRegressor(random_state=42)
model.fit(X_train, y_train)
end_time = time.time()

# Predict on the validation set
y_pred = model.predict(X_test)
print("Training time:", end_time - start_time) # Output the time spent on training

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
explained_variance = explained_variance_score(y_test, y_pred)

# Evaluate model performance using cross-validation on the training set
cv_mse_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
cv_mse_scores = -cv_mse_scores
cv_mse_mean = np.mean(cv_mse_scores)

# Output the evaluation metrics
print("Validation Set Metrics (with 90-day Windows) :")
print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R²):", r2)
print("Explained Variance:", explained_variance)
```

```
print("\nCross-Validation MSE Scores:", cv_mse_scores)
print("Average Cross-Validation MSE:", cv_mse_mean)
```

```
Training started!
Training time: 17.00140404701233
Validation Set Metrics (with 90-day Windows) :
Mean Squared Error (MSE): 4.2571832054540735e-05
Mean Absolute Error (MAE): 0.002641493050981676
R-squared (R²): 0.7911464108312911
Explained Variance: 0.791252516162468

Cross-Validation MSE Scores: [8.98039234e-05 4.60797337e-05 4.25899208e-05 5.60939584e-05
 4.90431484e-05]
Average Cross-Validation MSE: 5.67221369455628e-05
```

In [ ]:

Sliding Window

In [46]:
```
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, explained_variance_score, r2_score
from sklearn.model_selection import TimeSeriesSplit, cross_val_score

data = concatenated_data_str

# Filter the data for the specific time period
start_date = '2017-07-16'
end_date = '2017-08-15'
data_filtered = data[(data['date'] >= start_date) & (data['date'] <= end_date)].copy()

# Convert the date column to a datetime object
data_filtered['date'] = pd.to_datetime(data_filtered['date'])

# Define the window size
window_size = 15  # days

# Initialize lists to store predictions and actual values
predictions = []
actuals = []

# Iterate over each day from July 31 to August 15, 2017
for day in pd.date_range(start='2017-07-31', end='2017-08-15'):
    window_start = day - timedelta(days=window_size)
    window_end = day - timedelta(days=1)

    # Filter data for this window
    train_data = data_filtered[(data_filtered['date'] >= window_start) & (data_filtered['date'] <= window_end)]

    # Training data
    X_train = train_data[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]
    y_train = train_data['sales_scaled']

    # Testing data (the day to predict)
    test_data = data_filtered[data_filtered['date'] == day]
    X_test = test_data[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]
    y_test = test_data['sales_scaled']

    # Initialize the Gradient Boosting Regressor
    model = GradientBoostingRegressor()
    model.fit(X_train, y_train)

    # Make predictions for the test day
    y_pred = model.predict(X_test)

    # Store the predictions and actual values
    predictions.extend(y_pred)
    actuals.extend(y_test)

# Convert predictions and actuals to numpy arrays
predictions = np.array(predictions)
actuals = np.array(actuals)

# Calculate the evaluation metrics
mse = mean_squared_error(actuals, predictions)
mae = mean_absolute_error(actuals, predictions)
explained_var = explained_variance_score(actuals, predictions)
r2 = r2_score(actuals, predictions)

# Perform Cross-Validation using TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=5)
cv_mse_scores = cross_val_score(model, X_train, y_train, cv=tscv, scoring='neg_mean_squared_error')
cv_mse_scores = -cv_mse_scores
cv_mse_mean = np.mean(cv_mse_scores)

# Output the evaluation metrics
print("Validation Set Metrics (with 15-day Windows) :")
print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R²):", r2)
print("Explained Variance:", explained_var)
print("\nCross-Validation MSE Scores:", cv_mse_scores)
print("Average Cross-Validation MSE:", cv_mse_mean)
```

```
Validation Set Metrics (with 15-day Windows) :
Mean Squared Error (MSE): 4.2629502610868704e-05
Mean Absolute Error (MAE): 0.0024386272699047575
R-squared (R²): 0.7908634842552625
Explained Variance: 0.7908792197734958

Cross-Validation MSE Scores: [1.13531014e-04 8.26426631e-05 2.64856678e-05 3.02254020e-05
 3.35640268e-05]
Average Cross-Validation MSE: 5.7289754808133036e-05
```

In [47]:
```python
data = concatenated_data_str

# Filter the data for the specific time period
start_date = '2017-07-01'
end_date = '2017-08-15'
data_filtered = data[(data['date'] >= start_date) & (data['date'] <= end_date)].copy()

# Convert the date column to a datetime object for easier manipulation
data_filtered['date'] = pd.to_datetime(data_filtered['date'])

# Define the window size
window_size = 30  # days

# Initialize lists to store predictions and actual values
predictions = []
actuals = []

# Loop over each day from July 31 to August 15, 2017
for day in pd.date_range(start='2017-07-31', end='2017-08-15'):
    window_start = day - timedelta(days=window_size)
    window_end = day - timedelta(days=1)

    # Filter data for this window
    train_data = data_filtered[(data_filtered['date'] >= window_start) & (data_filtered['date'] <= window_end)]

    # Training data
    X_train = train_data[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]
    y_train = train_data['sales_scaled']

    # Testing data (the day to predict)
    test_data = data_filtered[data_filtered['date'] == day]
    X_test = test_data[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]
    y_test = test_data['sales_scaled']

    # Initialize the Gradient Boosting Regressor
    model = GradientBoostingRegressor()
    model.fit(X_train, y_train)

    # Make predictions for the test day
    y_pred = model.predict(X_test)

    # Store the predictions and actual values
    predictions.extend(y_pred)
    actuals.extend(y_test)

# Convert predictions and actuals to numpy arrays
predictions = np.array(predictions)
actuals = np.array(actuals)

# Calculate the evaluation metrics
mse = mean_squared_error(actuals, predictions)
mae = mean_absolute_error(actuals, predictions)
explained_var = explained_variance_score(actuals, predictions)
r2 = r2_score(actuals, predictions)

# Perform Cross-Validation using TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=5)
cv_mse_scores = cross_val_score(model, X_train, y_train, cv=tscv, scoring='neg_mean_squared_error')
cv_mse_scores = -cv_mse_scores
cv_mse_mean = np.mean(cv_mse_scores)

# Output the evaluation metrics
print("Validation Set Metrics (with 30-day Windows):")
print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R²):", r2)
print("Explained Variance:", explained_var)
print("\nCross-Validation MSE Scores:", cv_mse_scores)
print("Average Cross-Validation MSE:", cv_mse_mean)
```

```
Validation Set Metrics (with 30-day Windows):
Mean Squared Error (MSE): 4.592532503478061e-05
Mean Absolute Error (MAE): 0.0024933889714561573
R-squared (R²): 0.7746944750941147
Explained Variance: 0.7754473904914967

Cross-Validation MSE Scores: [6.88238075e-05 6.19916491e-05 4.77261434e-05 6.13937649e-05
 3.69954697e-05]
Average Cross-Validation MSE: 5.538616690558717e-05
```

In [48]:
```python
data = concatenated_data_str

# Filter the data for the specific time period
start_date = '2017-05-02'
end_date = '2017-08-15'
data_filtered = data[(data['date'] >= start_date) & (data['date'] <= end_date)].copy()

# Convert the date column to a datetime object for easier manipulation
```

```
data_filtered['date'] = pd.to_datetime(data_filtered['date'])

# Define the window size
window_size = 90  # days

# Initialize lists to store predictions and actual values
predictions = []
actuals = []

# Loop over each day from July 31 to August 15, 2017
for day in pd.date_range(start='2017-07-31', end='2017-08-15'):
    window_start = day - timedelta(days=window_size)
    window_end = day - timedelta(days=1)

    # Filter data for this window
    train_data = data_filtered[(data_filtered['date'] >= window_start) & (data_filtered['date'] <= window_end)]

    # Training data
    X_train = train_data[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]
    y_train = train_data['sales_scaled']

    # Testing data (the day to predict)
    test_data = data_filtered[data_filtered['date'] == day]
    X_test = test_data[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]
    y_test = test_data['sales_scaled']

    # Initialize the Gradient Boosting Regressor
    model = GradientBoostingRegressor()
    model.fit(X_train, y_train)

    # Make predictions for the test day
    y_pred = model.predict(X_test)

    # Store the predictions and actual values
    predictions.extend(y_pred)
    actuals.extend(y_test)

# Convert predictions and actuals to numpy arrays
predictions = np.array(predictions)
actuals = np.array(actuals)

# Calculate the evaluation metrics
mse = mean_squared_error(actuals, predictions)
mae = mean_absolute_error(actuals, predictions)
explained_var = explained_variance_score(actuals, predictions)
r2 = r2_score(actuals, predictions)

# Perform Cross-Validation using TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=5)
cv_mse_scores = cross_val_score(model, X_train, y_train, cv=tscv, scoring='neg_mean_squared_error')
cv_mse_scores = -cv_mse_scores
cv_mse_mean = np.mean(cv_mse_scores)

# Output the evaluation metrics
print("Validation Set Metrics (with 90-day Windows):")
print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R²):", r2)
print("Explained Variance:", explained_var)
print("\nCross-Validation MSE Scores:", cv_mse_scores)
print("Average Cross-Validation MSE:", cv_mse_mean)
```

```
Validation Set Metrics (with 90-day Windows):
Mean Squared Error (MSE): 4.6836901081482816e-05
Mean Absolute Error (MAE): 0.0025517879755789365
R-squared (R²): 0.7702223647815948
Explained Variance: 0.7712487424157111

Cross-Validation MSE Scores: [9.06300298e-05 5.00012338e-05 5.43947259e-05 4.31953825e-05
 5.38417228e-05]
Average Cross-Validation MSE: 5.841261896793369e-05
```

Choose RandomizedSearchCV for hyperparameter tuning for the best model with fixed 90-day time window.

In [ ]:

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import TimeSeriesSplit, cross_val_score
import numpy as np

# Assuming 'data' is the DataFrame loaded with the necessary preprocessing as per the user's code
data = concatenated_data_str

# Convert the 'date' column to datetime format
data['date'] = pd.to_datetime(data['date'])

# Selecting data for training and testing based on specified dates
train_data = data[((data['date'] >= '2015-05-02') & (data['date'] <= '2015-07-30')) |
                  ((data['date'] >= '2016-05-02') & (data['date'] <= '2016-07-30')) |
                  ((data['date'] >= '2017-05-02') & (data['date'] <= '2017-07-30'))]
test_data = data[(data['date'] >= '2017-07-31') & (data['date'] <= '2017-08-15')]

# Split data into features and target
features = ['store_nbr', 'product_type_encoded', 'special_offer']
X_train = train_data[features]
y_train = train_data['sales_scaled']
X_test = test_data[features]
```

```
y_test = test_data['sales_scaled']

# Define a range of hyperparameters for tuning
param_distributions = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'sqrt', 'log2']
}

# Initialize the Gradient Boosting Regressor
gbm = GradientBoostingRegressor(random_state=42)

# Set up RandomizedSearchCV with n_iter=10, for example
random_search = RandomizedSearchCV(estimator=gbm, param_distributions=param_distributions,
                                   n_iter=10, cv=5, n_jobs=-1, verbose=2,
                                   scoring='neg_mean_squared_error', random_state=42)

# Fit the RandomizedSearch to the data
random_search.fit(X_train, y_train)

# Best parameters and best score
best_params = random_search.best_params_
best_score = np.sqrt(-random_search.best_score_)

# Print the best parameters and best score
print("Best parameters found: ", best_params)
print("Best RMSE score from RandomizedSearchCV: ", best_score)

# Optional: Fit the model with the best parameters found and evaluate on the test set
gbm_best = GradientBoostingRegressor(**best_params, random_state=42)
gbm_best.fit(X_train, y_train)
y_pred = gbm_best.predict(X_test)

cv_mse_scores = cross_val_score(gbm_best, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
cv_mse_average = -np.mean(cv_mse_scores)

# Calculate evaluation metrics on the test set
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
explained_var = explained_variance_score(y_test, y_pred)

# Print out the metrics
print("Test Set Metrics:")
print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("R-squared (R²):", r2)
print("Explained Variance:", explained_var)
print("Average Cross-Validation MSE (not squared):", cv_mse_average)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [4…

```
import pandas as pd
from sklearn.neural_network import MLPRegressor
import numpy as np
from sklearn.model_selection import TimeSeriesSplit, cross_validate, cross_val_score, train_test_split
import time
from sklearn.metrics import mean_squared_error, mean_absolute_error,explained_variance_score, r2_score
from sklearn.preprocessing import StandardScaler

data = pd.read_csv("data.csv")
data['date'] = pd.to_datetime(data['date'])

train_data = data[((data['date'] >= '2015-07-01') & (data['date'] <= '2015-07-30')) |
                  ((data['date'] >= '2016-07-01') & (data['date'] <= '2016-07-30')) |
                  ((data['date'] >= '2017-07-01') & (data['date'] <= '2017-07-30'))]

features = ['store_nbr', 'product_type_encoded', 'special_offer_scaled']
target = 'sales_scaled'
X = train_data[features]
y = train_data[target]

X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_valid_scaled = scaler.transform(X_valid)

model = MLPRegressor(random_state=42, max_iter=1000, hidden_layer_sizes=(100,50 ), learning_rate='adaptive'
model.fit(X_train_scaled, y_train)

y_valid_pred = model.predict(X_valid_scaled)

mse = mean_squared_error(y_valid, y_valid_pred)
mae = mean_absolute_error(y_valid, y_valid_pred)
explained_variance = explained_variance_score(y_valid, y_valid_pred)
r2 = r2_score(y_valid, y_valid_pred)

cv_mse_scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error', cv=5)
cv_mse_mean = -cv_mse_scores.mean()

print(f'Validation Mean Squared Error (MSE): {mse}')
print(f'Validation Mean Absolute Error (MAE): {mae}')
print(f'Validation Explained Variance Score: {explained_variance}')
print(f'Validation R-squared: {r2}')
print(f'Validation Cross-Validation Mean Squared Error (MSE): {cv_mse_mean}')
```

```
Validation Mean Squared Error (MSE): 7.351961176273263e-05
Validation Mean Absolute Error (MAE): 0.004344495831551979
Validation Explained Variance Score: 0.6053426352661615
Validation R-squared: 0.6044343725506479
Validation Cross-Validation Mean Squared Error (MSE): 0.00027507347223063256
```

In [2…

```
future_data = data[(data['date'] >= '2017-07-31') & (data['date'] <= '2017-08-15')][['store_nbr', 'product_

future_data_scaled = scaler.transform(future_data)
future_predictions = model.predict(future_data_scaled)

final_prediction_start_date = '2017-07-31'
final_prediction_end_date = '2017-08-15'

data.loc[(data['date'] >= final_prediction_start_date) & (data['date'] <= final_prediction_end_date), 'pred

y_future = data[(data['date'] >= final_prediction_start_date) & (data['date'] <= final_prediction_end_date)

y_future_pred = future_predictions

mse_future = mean_squared_error(y_future, y_future_pred)
explained_variance_future = explained_variance_score(y_future, y_future_pred)
mae_future = mean_absolute_error(y_future, y_future_pred)
r2_future = r2_score(y_future, y_future_pred)

cv_mse_scores_future = cross_val_score(model, future_data_scaled, y_future, scoring='neg_mean_squared_error
cv_mse_mean_future = -cv_mse_scores_future.mean()

print(f'Future Mean Squared Error (MSE): {mse_future}')
print(f'Future Mean Absolute Error (MAE): {mae_future}')
print(f'Future Explained Variance Score: {explained_variance_future}')
print(f'Future R-squared: {r2_future}')
print(f'Future Cross-Validation Mean Squared Error (MSE): {cv_mse_mean_future}')
```

```
Future Mean Squared Error (MSE): 6.993639656537683e-05
Future Mean Absolute Error (MAE): 0.004505869587677799
Future Explained Variance Score: 0.6571657140690099
Future R-squared: 0.656898312069537
Future Cross-Validation Mean Squared Error (MSE): 0.00010518798782097069
```

In [1…

```
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
from sklearn.metrics import mean_squared_error, mean_absolute_error, explained_varian
from sklearn.model_selection import TimeSeriesSplit, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor

data = pd.read_csv("data.csv")
data['date'] = pd.to_datetime(data['date'])

start_date = '2017-07-01'
end_date = '2017-08-15'
data_filtered = data[(data['date'] >= start_date) & (data['date'] <= end_date)].copy(

data_filtered['date'] = pd.to_datetime(data_filtered['date'])
# Define the window size
window_size = 30

predictions = []
actuals = []
```

In [2]:

```
for day in pd.date_range(start='2017-07-31', end='2017-08-15'):
    window_start = day - timedelta(days=window_size)
    window_end = day - timedelta(days=1)
    train_data = data_filtered[(data_filtered['date'] >= window_start) & (data_filtered

    features = ['store_nbr', 'product_type_encoded', 'special_offer_scaled']

    X_train = train_data[features]
    y_train = train_data['sales_scaled']
    test_data = data_filtered[data_filtered['date'] == day]
    X_test = test_data[features]
    y_test = test_data['sales_scaled']

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    model = MLPRegressor(random_state=42, max_iter=1500, hidden_layer_sizes=(100,100),
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    predictions.extend(y_pred)
    actuals.extend(y_test)
```

In [3]:

```
predictions = np.array(predictions)
actuals = np.array(actuals)

mse = mean_squared_error(actuals,predictions)
mae = mean_absolute_error(actuals,predictions)
r2 = r2_score(actuals,predictions)
explained_variance = explained_variance_score(actuals,predictions)
cv_mse_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
cv_mse_scores = -cv_mse_scores
cv_mse_mean = cv_mse_scores.mean()

print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("Explained Variance:", explained_variance)
print("R-squared (R²):", r2)
print("Average Cross-Validation MSE:", cv_mse_mean)
```

```
Mean Squared Error (MSE): 7.780239858356441e-05
Mean Absolute Error (MAE): 0.004678362800527392
Explained Variance: 0.6186375588491868
R-squared (R²): 0.6183084117851878
Average Cross-Validation MSE: 7.289662337371913e-05
```

In [1…

```
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
from sklearn.metrics import mean_squared_error, mean_absolute_error, explained_varian
from sklearn.model_selection import TimeSeriesSplit, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor

data = pd.read_csv("data.csv")
data['date'] = pd.to_datetime(data['date'])

start_date = '2017-07-16'
end_date = '2017-08-15'
data_filtered = data[(data['date'] >= start_date) & (data['date'] <= end_date)].copy(

data_filtered['date'] = pd.to_datetime(data_filtered['date'])
# Define the window size
window_size = 15

predictions = []
actuals = []
```

In [2]:

```
for day in pd.date_range(start='2017-07-31', end='2017-08-15'):
    window_start = day - timedelta(days=window_size)
    window_end = day - timedelta(days=1)
    train_data = data_filtered[(data_filtered['date'] >= window_start) & (data_filtered

    features = ['store_nbr', 'product_type_encoded', 'special_offer_scaled']

    X_train = train_data[features]
    y_train = train_data['sales_scaled']
    test_data = data_filtered[data_filtered['date'] == day]
    X_test = test_data[features]
    y_test = test_data['sales_scaled']

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    model = MLPRegressor(random_state=42, max_iter=1000, hidden_layer_sizes=(100,50), l
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    predictions.extend(y_pred)
    actuals.extend(y_test)
```

In [3]:

```
predictions = np.array(predictions)
actuals = np.array(actuals)

mse = mean_squared_error(actuals,predictions)
mae = mean_absolute_error(actuals,predictions)
r2 = r2_score(actuals,predictions)
explained_variance = explained_variance_score(actuals,predictions)
cv_mse_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
cv_mse_scores = -cv_mse_scores
cv_mse_mean = cv_mse_scores.mean()

print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("Explained Variance:", explained_variance)
print("R-squared (R²):", r2)
print("Average Cross-Validation MSE:", cv_mse_mean)
```

```
Mean Squared Error (MSE): 0.00010123810120725339
Mean Absolute Error (MAE): 0.005105452768417078
Explained Variance: 0.5233595749258111
R-squared (R²): 0.5033349570046363
Average Cross-Validation MSE: 7.651854215292927e-05
```

In […

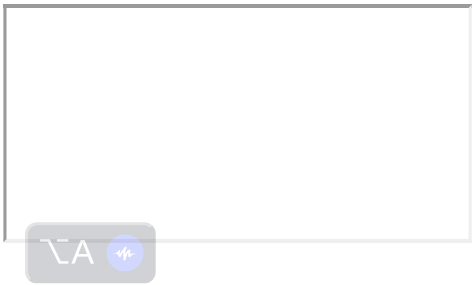
```
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
from sklearn.metrics import mean_squared_error, mean_absolute_error, explained_varian
from sklearn.model_selection import TimeSeriesSplit, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor

data = pd.read_csv("data.csv")
data['date'] = pd.to_datetime(data['date'])

start_date = '2017-05-02'
end_date = '2017-08-15'
data_filtered = data[(data['date'] >= start_date) & (data['date'] <= end_date)].copy(

data_filtered['date'] = pd.to_datetime(data_filtered['date'])
# Define the window size
window_size = 90

predictions = []
actuals = []
```

In [ ]:

```
for day in pd.date_range(start='2017-07-31', end='2017-08-15'):
    window_start = day - timedelta(days=window_size)
    window_end = day - timedelta(days=1)
    train_data = data_filtered[(data_filtered['date'] >= window_start) & (data_filtered

    features = ['store_nbr', 'product_type_encoded', 'special_offer_scaled']

    X_train = train_data[features]
    y_train = train_data['sales_scaled']
    test_data = data_filtered[data_filtered['date'] == day]
    X_test = test_data[features]
    y_test = test_data['sales_scaled']

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    model = MLPRegressor(random_state=42, max_iter=1500, hidden_layer_sizes=(200,100),
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    predictions.extend(y_pred)
    actuals.extend(y_test)
```

In [ ]:

```
predictions = np.array(predictions)
actuals = np.array(actuals)

mse = mean_squared_error(actuals,predictions)
mae = mean_absolute_error(actuals,predictions)
r2 = r2_score(actuals,predictions)
explained_variance = explained_variance_score(actuals,predictions)
cv_mse_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
cv_mse_scores = -cv_mse_scores
cv_mse_mean = cv_mse_scores.mean()

print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("Explained Variance:", explained_variance)
print("R-squared (R²):", r2)
print("Average Cross-Validation MSE:", cv_mse_mean)
```



In [8…

```
import pandas as pd
from sklearn.neural_network import MLPRegressor
import numpy as np
from sklearn.model_selection import TimeSeriesSplit, cross_validate, cross_val_score, train_test_spli
import time
from sklearn.metrics import mean_squared_error, mean_absolute_error,explained_variance_score, r2_scor
from sklearn.preprocessing import StandardScaler

data = pd.read_csv("data.csv")
data['date'] = pd.to_datetime(data['date'])

train_data = data[((data['date'] >= '2015-07-16') & (data['date'] <= '2015-07-30')) |
                  ((data['date'] >= '2016-07-16') & (data['date'] <= '2016-07-30')) |
                  ((data['date'] >= '2017-07-16') & (data['date'] <= '2017-07-30'))]

features = ['store_nbr', 'product_type_encoded', 'special_offer_scaled']
target = 'sales_scaled'
X = train_data[features]
y = train_data[target]

X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_valid_scaled = scaler.transform(X_valid)

model = MLPRegressor(random_state=42, max_iter=1000, hidden_layer_sizes=(100,50 ), learning_rate='ada
model.fit(X_train_scaled, y_train)

y_valid_pred = model.predict(X_valid_scaled)

mse = mean_squared_error(y_valid, y_valid_pred)
mae = mean_absolute_error(y_valid, y_valid_pred)
explained_variance = explained_variance_score(y_valid, y_valid_pred)
r2 = r2_score(y_valid, y_valid_pred)

cv_mse_scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error', cv=5)
cv_mse_mean = -cv_mse_scores.mean()

print(f'Validation Mean Squared Error (MSE): {mse}')
print(f'Validation Mean Absolute Error (MAE): {mae}')
print(f'Validation Explained Variance Score: {explained_variance}')
print(f'Validation R-squared: {r2}')
print(f'Validation Cross-Validation Mean Squared Error (MSE): {cv_mse_mean}')
```

```
Validation Mean Squared Error (MSE): 8.24043840257669e-05
Validation Mean Absolute Error (MAE): 0.004800877199361926
Validation Explained Variance Score: 0.5151509440877196
Validation R-squared: 0.5138836228691206
Validation Cross-Validation Mean Squared Error (MSE): 0.0004104008052078555
```

In [5]:

```
future_data = data[(data['date'] >= '2017-07-31') & (data['date'] <= '2017-08-15')][['store_nbr', 'product_type_encoded', '

future_data_scaled = scaler.transform(future_data)
future_predictions = model.predict(future_data_scaled)

final_prediction_start_date = '2017-07-31'
final_prediction_end_date = '2017-08-15'

data.loc[(data['date'] >= final_prediction_start_date) & (data['date'] <= final_prediction_end_date), 'predicted_sales_scal

#print(data[['date', 'store_nbr', 'product_type_encoded', 'predicted_sales_scaled']])

y_future = data[(data['date'] >= final_prediction_start_date) & (data['date'] <= final_prediction_end_date)]['sales_scaled'

y_future_pred = future_predictions

mse_future = mean_squared_error(y_future, y_future_pred)
explained_variance_future = explained_variance_score(y_future, y_future_pred)
mae_future = mean_absolute_error(y_future, y_future_pred)
r2_future = r2_score(y_future, y_future_pred)

cv_mse_scores_future = cross_val_score(model, future_data_scaled, y_future, scoring='neg_mean_squared_error', cv=5)
cv_mse_mean_future = -cv_mse_scores_future.mean()

print(f'Future Mean Squared Error (MSE): {mse_future}')
print(f'Future Mean Absolute Error (MAE): {mae_future}')
print(f'Future Explained Variance Score: {explained_variance_future}')
print(f'Future R-squared: {r2_future}')
print(f'Future Cross-Validation Mean Squared Error (MSE): {cv_mse_mean_future}')
```

```
Future Mean Squared Error (MSE): 9.501600232278765e-05
Future Mean Absolute Error (MAE): 0.005234992235957219
Future Explained Variance Score: 0.5446770435962283
Future R-squared: 0.5338600159806848
Future Cross-Validation Mean Squared Error (MSE): 0.00011722207389275746
```

In [5…

```
import pandas as pd
from sklearn.neural_network import MLPRegressor
import numpy as np
from sklearn.model_selection import TimeSeriesSplit, cross_validate, cross_val_score, train_test_spli
import time
from sklearn.metrics import mean_squared_error, mean_absolute_error,explained_variance_score, r2_scor
from sklearn.preprocessing import StandardScaler

data = pd.read_csv("data.csv")
data['date'] = pd.to_datetime(data['date'])

train_data = data[((data['date'] >= '2015-05-02') & (data['date'] <= '2015-07-30')) |
                  ((data['date'] >= '2016-05-02') & (data['date'] <= '2016-07-30')) |
                  ((data['date'] >= '2017-05-02') & (data['date'] <= '2017-07-30'))]

features = ['store_nbr', 'product_type_encoded', 'special_offer_scaled']
target = 'sales_scaled'
X = train_data[features]
y = train_data[target]

X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_valid_scaled = scaler.transform(X_valid)

model = MLPRegressor(random_state=42, max_iter=1000, hidden_layer_sizes=(100,100 ), learning_rate='ad
model.fit(X_train_scaled, y_train)

y_valid_pred = model.predict(X_valid_scaled)

mse = mean_squared_error(y_valid, y_valid_pred)
mae = mean_absolute_error(y_valid, y_valid_pred)
explained_variance = explained_variance_score(y_valid, y_valid_pred)
r2 = r2_score(y_valid, y_valid_pred)

cv_mse_scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error', cv=5)
cv_mse_mean = -cv_mse_scores.mean()

print(f'Validation Mean Squared Error (MSE): {mse}')
print(f'Validation Mean Absolute Error (MAE): {mae}')
print(f'Validation Explained Variance Score: {explained_variance}')
print(f'Validation R-squared: {r2}')
print(f'Validation Cross-Validation Mean Squared Error (MSE): {cv_mse_mean}')
```

```
Validation Mean Squared Error (MSE): 5.598045056859575e-05
Validation Mean Absolute Error (MAE): 0.003513910729561363
Validation Explained Variance Score: 0.7168930389171826
Validation R-squared: 0.7158706633306176
Validation Cross-Validation Mean Squared Error (MSE): 0.0001408836389699391
```

In [4]:

```
future_data = data[(data['date'] >= '2017-07-31') & (data['date'] <= '2017-08-15')][['store_nbr', 'product_type_encoded', '

future_data_scaled = scaler.transform(future_data)
future_predictions = model.predict(future_data_scaled)

final_prediction_start_date = '2017-07-31'
final_prediction_end_date = '2017-08-15'

data.loc[(data['date'] >= final_prediction_start_date) & (data['date'] <= final_prediction_end_date), 'predicted_sales_scal

y_future = data[(data['date'] >= final_prediction_start_date) & (data['date'] <= final_prediction_end_date)]['sales_scaled'

y_future_pred = future_predictions

mse_future = mean_squared_error(y_future, y_future_pred)
explained_variance_future = explained_variance_score(y_future, y_future_pred)
mae_future = mean_absolute_error(y_future, y_future_pred)
r2_future = r2_score(y_future, y_future_pred)

cv_mse_scores_future = cross_val_score(model, future_data_scaled, y_future, scoring='neg_mean_squared_error', cv=5)
cv_mse_mean_future = -cv_mse_scores_future.mean()

print(f'Future Mean Squared Error (MSE): {mse_future}')
print(f'Future Mean Absolute Error (MAE): {mae_future}')
print(f'Future Explained Variance Score: {explained_variance_future}')
print(f'Future R-squared: {r2_future}')
print(f'Future Cross-Validation Mean Squared Error (MSE): {cv_mse_mean_future}')
```

```
Future Mean Squared Error (MSE): 4.680239427782925e-05
Future Mean Absolute Error (MAE): 0.002970997416241512
Future Explained Variance Score: 0.7714895855331694
Future R-squared: 0.7703916520648988
Future Cross-Validation Mean Squared Error (MSE): 9.39069796017127e-05
```

In [ ]:

In [3…

```
pd.set_option('display.float_format', lambda x: f'{x:.6f}')
# Selecting the 'sales' and 'special_offer' columns for MinMax scaling
data_to_scale = filtered_data[['sales', 'special_offer']]

# Creating a MinMaxScaler object
min_max_scaler = MinMaxScaler()

# 对数据进行 MinMax 缩放
scaled_data = min_max_scaler.fit_transform(data_to_scale)

# Converting scaled data to string format with six decimal places
scaled_data_str = pd.DataFrame(scaled_data, columns=['sales_scaled', 'special_offer_scaled']).applyma

# Resetting the index of filtered_data to ensure proper alignment during concatenation
filtered_data_reset = filtered_data.reset_index(drop=True)

# Concatenating the string formatted scaled data with the original filtered_data
concatenated_data_str = pd.concat([filtered_data_reset, scaled_data_str], axis=1)

# Converting the last two columns (string formatted scaled data) back to numeric format with six deci
concatenated_data_str['sales_scaled'] = concatenated_data_str['sales_scaled'].astype(float)
concatenated_data_str['special_offer_scaled'] = concatenated_data_str['special_offer_scaled'].astype(

# Displaying the first few rows of the data to confirm the rounding
concatenated_data_str
```

Out[30]:

	id	date	store_nbr	product_type	sales	special_offer	product_type_encoded	sales_scaled	special
0	1566378	2015-06-01	1	AUTOMOTIVE	1.000000	0	0	0.000011	
1	1566379	2015-06-01	1	BABY CARE	0.000000	0	1	0.000000	
2	1566380	2015-06-01	1	BEAUTY	6.000000	0	2	0.000069	
3	1566381	2015-06-01	1	BEVERAGES	2210.000000	0	3	0.025275	
4	1566382	2015-06-01	1	BOOKS	0.000000	0	4	0.000000	
…	…	…	…	…	…	…	…	…	…
1434505	3000883	2017-08-15	9	POULTRY	438.133000	0	28	0.005011	
1434506	3000884	2017-08-15	9	PREPARED FOODS	154.553000	1	29	0.001768	
1434507	3000885	2017-08-15	9	PRODUCE	2419.729000	148	30	0.027673	
1434508	3000886	2017-08-15	9	SCHOOL AND OFFICE SUPPLIES	121.000000	8	31	0.001384	
1434509	3000887	2017-08-15	9	SEAFOOD	16.000000	0	32	0.000183	

1434510 rows × 9 columns

In [52]:

```
# Sliding window 30
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, explained_variance_score
from sklearn.model_selection import cross_val_predict

# Define the prediction date range
prediction_dates = pd.date_range(start='2017-07-31', end='2017-08-15')
predict_data = concatenated_data_str.copy()

# Store performance metrics for each day
performance_metrics = {'MSE': [], 'R²': [], 'MAE': [], 'Explained Variance': [], 'CV_MSE': []}

# Iterate over each combination of product type and store
for date in prediction_dates:
    # Build the training set
    train_set = predict_data.loc[
        (
            (predict_data['date'] >= date - pd.DateOffset(days=30)) &
            (predict_data['date'] < date)
        )
    ].reset_index(drop=True)
```

```
    # Build the test set
    test_set = concatenated_data_str[(concatenated_data_str['date'] == date)]

    # Extract features and target variables
    X_train = train_set[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]  # Select appropriate features
    y_train = train_set['sales_scaled']

    X_test = test_set[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]  # Select appropriate features
    y_test = test_set['sales_scaled']

    # Build the Random Forest regression model
    model = RandomForestRegressor(n_estimators=100, random_state=42)

    # Perform cross-validation prediction
    cv_predictions = cross_val_predict(model, X_train, y_train, cv=5)

    # Fit the model (using the entire training set)
    model.fit(X_train, y_train)

    # Make predictions
    predictions = model.predict(X_test)

    # Update the predictions in predict_data
    predict_data.loc[predict_data['date'] == date, 'sales_scaled'] = predictions

    # Calculate model performance (Mean Squared Error, R², MAE, and Explained Variance)
    mse = mean_squared_error(y_test, predictions)
    r_squared = r2_score(y_test, predictions)
    mae = mean_absolute_error(y_test, predictions)
    explained_var = explained_variance_score(y_test, predictions)
    # Output cross-validation model performance (Mean Squared Error and R²)
    cv_mse = mean_squared_error(y_train, cv_predictions)

    # Store performance metrics for each day
    performance_metrics['MSE'].append(mse)
    performance_metrics['R²'].append(r_squared)
    performance_metrics['MAE'].append(mae)
    performance_metrics['Explained Variance'].append(explained_var)
    performance_metrics['CV_MSE'].append(cv_mse)
    # Output model performance
    print(f'Model on date {date}: MSE = {mse:.6f}, R² = {r_squared:.6f}, MAE = {mae:.6f}, Explained Variance = {explained_v

# Calculate the average value for each metric
avg_performance = {metric: sum(values) / len(values) for metric, values in performance_metrics.items()}

# Output average performance
print(f'Average Performance over 16 Days:')
for metric, value in avg_performance.items():
    print(f'{metric}: {value:.6f}')
```

```
Model on date 2017-07-31 00:00:00: MSE = 0.000013, R² = 0.943543, MAE = 0.001017, Explained Variance = 0.943726, CV_MSE =
0.000034
Model on date 2017-08-01 00:00:00: MSE = 0.000039, R² = 0.853527, MAE = 0.001685, Explained Variance = 0.857901, CV_MSE =
0.000031
Model on date 2017-08-02 00:00:00: MSE = 0.000024, R² = 0.913662, MAE = 0.001345, Explained Variance = 0.914559, CV_MSE =
0.000027
Model on date 2017-08-03 00:00:00: MSE = 0.000049, R² = 0.668859, MAE = 0.001731, Explained Variance = 0.669817, CV_MSE =
0.000023
Model on date 2017-08-04 00:00:00: MSE = 0.000058, R² = 0.698213, MAE = 0.001884, Explained Variance = 0.700403, CV_MSE =
0.000021
Model on date 2017-08-05 00:00:00: MSE = 0.000110, R² = 0.611389, MAE = 0.002394, Explained Variance = 0.621177, CV_MSE =
0.000020
Model on date 2017-08-06 00:00:00: MSE = 0.000134, R² = 0.607048, MAE = 0.002879, Explained Variance = 0.621267, CV_MSE =
0.000021
Model on date 2017-08-07 00:00:00: MSE = 0.000053, R² = 0.704577, MAE = 0.001785, Explained Variance = 0.707020, CV_MSE =
0.000020
Model on date 2017-08-08 00:00:00: MSE = 0.000038, R² = 0.721259, MAE = 0.001597, Explained Variance = 0.721360, CV_MSE =
0.000020
Model on date 2017-08-09 00:00:00: MSE = 0.000033, R² = 0.797021, MAE = 0.001566, Explained Variance = 0.797110, CV_MSE =
0.000017
Model on date 2017-08-10 00:00:00: MSE = 0.000043, R² = 0.636223, MAE = 0.001785, Explained Variance = 0.636329, CV_MSE =
0.000016
Model on date 2017-08-11 00:00:00: MSE = 0.000066, R² = 0.658725, MAE = 0.002016, Explained Variance = 0.661739, CV_MSE =
0.000015
Model on date 2017-08-12 00:00:00: MSE = 0.000059, R² = 0.674539, MAE = 0.001793, Explained Variance = 0.677389, CV_MSE =
0.000016
Model on date 2017-08-13 00:00:00: MSE = 0.000077, R² = 0.662233, MAE = 0.002110, Explained Variance = 0.667716, CV_MSE =
0.000016
Model on date 2017-08-14 00:00:00: MSE = 0.000043, R² = 0.733834, MAE = 0.001695, Explained Variance = 0.735236, CV_MSE =
0.000015
Model on date 2017-08-15 00:00:00: MSE = 0.000041, R² = 0.732202, MAE = 0.001684, Explained Variance = 0.733130, CV_MSE =
0.000018
Average Performance over 16 Days:
MSE: 0.000055
R²: 0.726053
MAE: 0.001810
Explained Variance: 0.729117
CV_MSE: 0.000020
```

In [53]:

```
# Random Forest Sliding Window 15
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, explained_variance_score
from sklearn.model_selection import cross_val_predict

# Define the prediction date range
prediction_dates = pd.date_range(start='2017-07-31', end='2017-08-15')
predict_data = concatenated_data_str.copy()
```

```python
# Store performance metrics for each day
performance_metrics = {'MSE': [], 'R²': [], 'MAE': [], 'Explained Variance': [], 'CV_MSE':[]}

# Iterate over each combination of product type and store
for date in prediction_dates:
    # Build the training set
    train_set = predict_data.loc[
        (
            (predict_data['date'] >= date - pd.DateOffset(days=15)) &
            (predict_data['date'] < date)
        )
    ].reset_index(drop=True)

    # Build the test set
    test_set = concatenated_data_str[(concatenated_data_str['date'] == date)]

    # Extract features and target variable
    X_train = train_set[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]  # Choose appropriate features
    y_train = train_set['sales_scaled']

    X_test = test_set[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]  # Choose appropriate features
    y_test = test_set['sales_scaled']

    # Build the Random Forest regression model
    model = RandomForestRegressor(n_estimators=100, random_state=42)

    # Perform cross-validation prediction
    cv_predictions = cross_val_predict(model, X_train, y_train, cv=5)

    # Fit the model
    model.fit(X_train, y_train)

    # Make predictions
    predictions = model.predict(X_test)

    # Update the predictions in predict_data
    predict_data.loc[predict_data['date'] == date, 'sales_scaled'] = predictions

    # Calculate model performance (Mean Squared Error and R-squared)
    mse = mean_squared_error(y_test, predictions)
    r_squared = r2_score(y_test, predictions)
    mae = mean_absolute_error(y_test, predictions)
    explained_var = explained_variance_score(y_test, predictions)

    # Output cross-validation model performance (Mean Squared Error and R-squared)
    cv_mse = mean_squared_error(y_train, cv_predictions)

    # Store performance metrics for each day
    performance_metrics['MSE'].append(mse)
    performance_metrics['R²'].append(r_squared)
    performance_metrics['MAE'].append(mae)
    performance_metrics['Explained Variance'].append(explained_var)
    performance_metrics['CV_MSE'].append(cv_mse)

    # Output model performance
    print(f'Model on date {date}: MSE = {mse:.6f}, R² = {r_squared:.6f}, MAE = {mae:.6f}, Explained Variance = {explained_v

# Calculate the average value for each metric
avg_performance = {metric: sum(values) / len(values) for metric, values in performance_metrics.items()}

# Output average performance
print(f'Average Performance over 16 Days:')
for metric, value in avg_performance.items():
    print(f'{metric}: {value:.6f}')
```

```
Model on date 2017-07-31 00:00:00: MSE = 0.000014, R² = 0.939373, MAE = 0.001066, Explained Variance = 0.939557, CV_MSE =
0.000032
Model on date 2017-08-01 00:00:00: MSE = 0.000057, R² = 0.783909, MAE = 0.002018, Explained Variance = 0.790235, CV_MSE =
0.000022
Model on date 2017-08-02 00:00:00: MSE = 0.000042, R² = 0.848746, MAE = 0.001803, Explained Variance = 0.850280, CV_MSE =
0.000020
Model on date 2017-08-03 00:00:00: MSE = 0.000050, R² = 0.662486, MAE = 0.001856, Explained Variance = 0.663775, CV_MSE =
0.000023
Model on date 2017-08-04 00:00:00: MSE = 0.000065, R² = 0.664068, MAE = 0.002168, Explained Variance = 0.665278, CV_MSE =
0.000022
Model on date 2017-08-05 00:00:00: MSE = 0.000118, R² = 0.583179, MAE = 0.002653, Explained Variance = 0.590926, CV_MSE =
0.000019
Model on date 2017-08-06 00:00:00: MSE = 0.000148, R² = 0.565584, MAE = 0.003137, Explained Variance = 0.577916, CV_MSE =
0.000017
Model on date 2017-08-07 00:00:00: MSE = 0.000060, R² = 0.662489, MAE = 0.002126, Explained Variance = 0.664218, CV_MSE =
0.000016
Model on date 2017-08-08 00:00:00: MSE = 0.000045, R² = 0.672547, MAE = 0.001974, Explained Variance = 0.672577, CV_MSE =
0.000017
Model on date 2017-08-09 00:00:00: MSE = 0.000041, R² = 0.750907, MAE = 0.001850, Explained Variance = 0.751247, CV_MSE =
0.000016
Model on date 2017-08-10 00:00:00: MSE = 0.000037, R² = 0.681124, MAE = 0.001910, Explained Variance = 0.681524, CV_MSE =
0.000014
Model on date 2017-08-11 00:00:00: MSE = 0.000066, R² = 0.661504, MAE = 0.002206, Explained Variance = 0.663259, CV_MSE =
0.000009
Model on date 2017-08-12 00:00:00: MSE = 0.000061, R² = 0.664322, MAE = 0.002079, Explained Variance = 0.666267, CV_MSE =
0.000010
Model on date 2017-08-13 00:00:00: MSE = 0.000078, R² = 0.659270, MAE = 0.002307, Explained Variance = 0.663635, CV_MSE =
0.000009
Model on date 2017-08-14 00:00:00: MSE = 0.000048, R² = 0.703849, MAE = 0.002011, Explained Variance = 0.704663, CV_MSE =
0.000007
Model on date 2017-08-15 00:00:00: MSE = 0.000045, R² = 0.706725, MAE = 0.001982, Explained Variance = 0.707139, CV_MSE =
0.000005
Average Performance over 16 Days:
MSE: 0.000061
R²: 0.700630
MAE: 0.002072
Explained Variance: 0.703281
CV_MSE: 0.000016
```

In [54]:

```python
# Random Forest Sliding Window 90
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, explained_variance_score
from sklearn.model_selection import cross_val_predict

# Define the prediction date range
prediction_dates = pd.date_range(start='2017-07-31', end='2017-08-15')
predict_data = concatenated_data_str.copy()

# Store performance metrics for each day
performance_metrics = {'MSE': [], 'R²': [], 'MAE': [], 'Explained Variance': [], 'CV_MSE':[]}

# Iterate over each combination of product type and store
for date in prediction_dates:
    # Build the training set
    train_set = predict_data.loc[
        (
            (predict_data['date'] >= date - pd.DateOffset(days=90)) &
            (predict_data['date'] < date)
        )
    ].reset_index(drop=True)

    # Build the test set
    test_set = concatenated_data_str[(concatenated_data_str['date'] == date)]

    # Extract features and target variable
    X_train = train_set[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]  # Choose appropriate features
    y_train = train_set['sales_scaled']

    X_test = test_set[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]  # Choose appropriate features
    y_test = test_set['sales_scaled']

    # Build the Random Forest regression model
    model = RandomForestRegressor(n_estimators=100, random_state=42)

    # Perform cross-validation prediction
    cv_predictions = cross_val_predict(model, X_train, y_train, cv=5)

    # Fit the model
    model.fit(X_train, y_train)

    # Make predictions
    predictions = model.predict(X_test)

    # Update the predictions in predict_data
    predict_data.loc[predict_data['date'] == date, 'sales_scaled'] = predictions

    # Calculate model performance (Mean Squared Error and R-squared)
    mse = mean_squared_error(y_test, predictions)
    r_squared = r2_score(y_test, predictions)
    mae = mean_absolute_error(y_test, predictions)
    explained_var = explained_variance_score(y_test, predictions)

    # Output cross-validation model performance (Mean Squared Error and R-squared)
    cv_mse = mean_squared_error(y_train, cv_predictions)

    # Store performance metrics for each day
    performance_metrics['MSE'].append(mse)
```

```
    performance_metrics['R²'].append(r_squared)
    performance_metrics['MAE'].append(mae)
    performance_metrics['Explained Variance'].append(explained_var)
    performance_metrics['CV_MSE'].append(cv_mse)

    # Output model performance
    print(f'Model on date {date}: MSE = {mse:.6f}, R² = {r_squared:.6f}, MAE = {mae:.6f}, Explained Variance = {explained_v

# Calculate the average value for each metric
avg_performance = {metric: sum(values) / len(values) for metric, values in performance_metrics.items()}

# Output average performance
print(f'Average Performance over 16 Days:')
for metric, value in avg_performance.items():
    print(f'{metric}: {value:.6f}')
```

```
Model on date 2017-07-31 00:00:00: MSE = 0.000011, R² = 0.948845, MAE = 0.000937, Explained Variance = 0.949368, CV_MSE =
0.000029
Model on date 2017-08-01 00:00:00: MSE = 0.000029, R² = 0.889301, MAE = 0.001485, Explained Variance = 0.893151, CV_MSE =
0.000029
Model on date 2017-08-02 00:00:00: MSE = 0.000015, R² = 0.945000, MAE = 0.001119, Explained Variance = 0.945607, CV_MSE =
0.000028
Model on date 2017-08-03 00:00:00: MSE = 0.000014, R² = 0.904493, MAE = 0.001171, Explained Variance = 0.905560, CV_MSE =
0.000027
Model on date 2017-08-04 00:00:00: MSE = 0.000014, R² = 0.925429, MAE = 0.001128, Explained Variance = 0.925606, CV_MSE =
0.000027
Model on date 2017-08-05 00:00:00: MSE = 0.000027, R² = 0.903331, MAE = 0.001363, Explained Variance = 0.906052, CV_MSE =
0.000028
Model on date 2017-08-06 00:00:00: MSE = 0.000046, R² = 0.864304, MAE = 0.001896, Explained Variance = 0.869933, CV_MSE =
0.000027
Model on date 2017-08-07 00:00:00: MSE = 0.000012, R² = 0.934899, MAE = 0.001016, Explained Variance = 0.935001, CV_MSE =
0.000026
Model on date 2017-08-08 00:00:00: MSE = 0.000018, R² = 0.867280, MAE = 0.001303, Explained Variance = 0.871982, CV_MSE =
0.000026
Model on date 2017-08-09 00:00:00: MSE = 0.000015, R² = 0.906246, MAE = 0.001205, Explained Variance = 0.910911, CV_MSE =
0.000026
Model on date 2017-08-10 00:00:00: MSE = 0.000020, R² = 0.828989, MAE = 0.001402, Explained Variance = 0.836960, CV_MSE =
0.000025
Model on date 2017-08-11 00:00:00: MSE = 0.000016, R² = 0.918072, MAE = 0.001203, Explained Variance = 0.918076, CV_MSE =
0.000025
Model on date 2017-08-12 00:00:00: MSE = 0.000017, R² = 0.908261, MAE = 0.001065, Explained Variance = 0.908343, CV_MSE =
0.000025
Model on date 2017-08-13 00:00:00: MSE = 0.000019, R² = 0.914701, MAE = 0.001256, Explained Variance = 0.915210, CV_MSE =
0.000025
Model on date 2017-08-14 00:00:00: MSE = 0.000012, R² = 0.925462, MAE = 0.001057, Explained Variance = 0.926103, CV_MSE =
0.000026
Model on date 2017-08-15 00:00:00: MSE = 0.000015, R² = 0.903965, MAE = 0.001182, Explained Variance = 0.905574, CV_MSE =
0.000024
Average Performance over 16 Days:
MSE: 0.000019
R²: 0.905536
MAE: 0.001237
Explained Variance: 0.907715
CV_MSE: 0.000026
```

In [55]:

```
# Fixed window 30
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, explained_variance_score
from sklearn.model_selection import cross_val_predict

# Define the prediction date range
prediction_dates = pd.date_range(start='2017-07-31', end='2017-08-15')
predict_data = concatenated_data_str.copy()

# Store performance metrics for each day
performance_metrics = {'MSE': [], 'R²': [], 'MAE': [], 'Explained Variance': [], 'CV_MSE': []}

# Calculate the date for the same day two years ago and one year ago
two_years_ago = prediction_dates[0] - pd.DateOffset(years=2)
one_year_ago = prediction_dates[0] - pd.DateOffset(years=1)

# Build the training set
train_set = predict_data.loc[
    (
        (predict_data['date'] >= two_years_ago - pd.DateOffset(days=30)) &
        (predict_data['date'] < two_years_ago)
    ) |
    (
        (predict_data['date'] >= one_year_ago - pd.DateOffset(days=30)) &
        (predict_data['date'] < one_year_ago)
    ) |
    (
        (predict_data['date'] >= prediction_dates[0] - pd.DateOffset(days=30)) &
        (predict_data['date'] < prediction_dates[0])
    )
].reset_index(drop=True)

# Extract features and target variables
X_train = train_set[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]  # Select appropriate features
y_train = train_set['sales_scaled']

# Build the Random Forest regression model
model = RandomForestRegressor(n_estimators=100, random_state=42)

# Perform cross-validation prediction
cv_predictions = cross_val_predict(model, X_train, y_train, cv=5)

# Fit the model
```

```
model.fit(X_train, y_train)

# Iterate over prediction dates
for date in prediction_dates:
    # Build the test set
    test_set = concatenated_data_str[(concatenated_data_str['date'] == date)]

    # Extract test set features
    X_test = test_set[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]  # Select appropriate features
    y_test = test_set['sales_scaled']

    # Make predictions
    predictions = model.predict(X_test)

    # Update the predictions in predict_data
    predict_data.loc[predict_data['date'] == date, 'sales_scaled'] = predictions

    # Calculate model performance (Mean Squared Error, R², MAE, and Explained Variance)
    mse = mean_squared_error(y_test, predictions)
    r_squared = r2_score(y_test, predictions)
    mae = mean_absolute_error(y_test, predictions)
    explained_var = explained_variance_score(y_test, predictions)
    # Output cross-validation model performance (Mean Squared Error and R²)
    cv_mse = mean_squared_error(y_train, cv_predictions)

    # Store performance metrics for each day
    performance_metrics['MSE'].append(mse)
    performance_metrics['R²'].append(r_squared)
    performance_metrics['MAE'].append(mae)
    performance_metrics['Explained Variance'].append(explained_var)
    performance_metrics['CV_MSE'].append(cv_mse)

    # Output model performance
    print(f'Model on date {date}: MSE = {mse:.6f}, R² = {r_squared:.6f}, MAE = {mae:.6f}, Explained Variance = {explained_v

# Calculate the average value for each metric
avg_performance = {metric: sum(values) / len(values) for metric, values in performance_metrics.items()}

# Output average performance
print(f'Average Performance over 16 Days:')
for metric, value in avg_performance.items():
    print(f'{metric}: {value:.6f}')
```

```
Model on date 2017-07-31 00:00:00: MSE = 0.000014, R² = 0.938972, MAE = 0.001046, Explained Variance = 0.939495, CV_MSE =
0.000025
Model on date 2017-08-01 00:00:00: MSE = 0.000030, R² = 0.888261, MAE = 0.001490, Explained Variance = 0.890932, CV_MSE =
0.000025
Model on date 2017-08-02 00:00:00: MSE = 0.000020, R² = 0.927983, MAE = 0.001198, Explained Variance = 0.928619, CV_MSE =
0.000025
Model on date 2017-08-03 00:00:00: MSE = 0.000015, R² = 0.896394, MAE = 0.001072, Explained Variance = 0.897424, CV_MSE =
0.000025
Model on date 2017-08-04 00:00:00: MSE = 0.000019, R² = 0.902821, MAE = 0.001172, Explained Variance = 0.902873, CV_MSE =
0.000025
Model on date 2017-08-05 00:00:00: MSE = 0.000043, R² = 0.847701, MAE = 0.001637, Explained Variance = 0.851772, CV_MSE =
0.000025
Model on date 2017-08-06 00:00:00: MSE = 0.000061, R² = 0.821715, MAE = 0.002078, Explained Variance = 0.829222, CV_MSE =
0.000025
Model on date 2017-08-07 00:00:00: MSE = 0.000018, R² = 0.899576, MAE = 0.001114, Explained Variance = 0.899583, CV_MSE =
0.000025
Model on date 2017-08-08 00:00:00: MSE = 0.000018, R² = 0.867557, MAE = 0.001101, Explained Variance = 0.870076, CV_MSE =
0.000025
Model on date 2017-08-09 00:00:00: MSE = 0.000011, R² = 0.931943, MAE = 0.000994, Explained Variance = 0.934271, CV_MSE =
0.000025
Model on date 2017-08-10 00:00:00: MSE = 0.000017, R² = 0.854511, MAE = 0.001236, Explained Variance = 0.860853, CV_MSE =
0.000025
Model on date 2017-08-11 00:00:00: MSE = 0.000021, R² = 0.890740, MAE = 0.001256, Explained Variance = 0.890752, CV_MSE =
0.000025
Model on date 2017-08-12 00:00:00: MSE = 0.000020, R² = 0.890827, MAE = 0.001132, Explained Variance = 0.890827, CV_MSE =
0.000025
Model on date 2017-08-13 00:00:00: MSE = 0.000028, R² = 0.876761, MAE = 0.001376, Explained Variance = 0.877810, CV_MSE =
0.000025
Model on date 2017-08-14 00:00:00: MSE = 0.000013, R² = 0.922339, MAE = 0.001030, Explained Variance = 0.922365, CV_MSE =
0.000025
Model on date 2017-08-15 00:00:00: MSE = 0.000013, R² = 0.915415, MAE = 0.001068, Explained Variance = 0.915835, CV_MSE =
0.000025
Average Performance over 16 Days:
MSE: 0.000023
R²: 0.892095
MAE: 0.001250
Explained Variance: 0.893920
CV_MSE: 0.000025
```

In [56]:
```
# Fixed Window 15
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, explained_variance_score
from sklearn.model_selection import cross_val_predict

# Define the prediction date range
prediction_dates = pd.date_range(start='2017-07-31', end='2017-08-15')
predict_data = concatenated_data_str.copy()

# Store performance metrics for each day
performance_metrics = {'MSE': [], 'R²': [], 'MAE': [], 'Explained Variance': [], 'CV_MSE':[]}

# Calculate the date for the same day two years ago and one year ago
two_years_ago = prediction_dates[0] - pd.DateOffset(years=2)
one_year_ago = prediction_dates[0] - pd.DateOffset(years=1)
```

```python
# Build the training set
train_set = predict_data.loc[
    (
        (predict_data['date'] >= two_years_ago - pd.DateOffset(days=15)) &
        (predict_data['date'] < two_years_ago)
    ) |
    (
        (predict_data['date'] >= one_year_ago - pd.DateOffset(days=15)) &
        (predict_data['date'] < one_year_ago)
    ) |
    (
        (predict_data['date'] >= prediction_dates[0] - pd.DateOffset(days=15)) &
        (predict_data['date'] < prediction_dates[0])
    )
].reset_index(drop=True)

# Extract features and target variable
X_train = train_set[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]  # Choose appropriate features
y_train = train_set['sales_scaled']

# Build the Random Forest regression model
model = RandomForestRegressor(n_estimators=100, random_state=42)

# Perform cross-validation prediction
cv_predictions = cross_val_predict(model, X_train, y_train, cv=5)

# Fit the model
model.fit(X_train, y_train)

# Loop over prediction dates
for date in prediction_dates:
    # Build the test set
    test_set = concatenated_data_str[(concatenated_data_str['date'] == date)]

    # Extract test set features
    X_test = test_set[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]  # Choose appropriate features
    y_test = test_set['sales_scaled']

    # Make predictions
    predictions = model.predict(X_test)

    # Update predictions in predict_data
    predict_data.loc[predict_data['date'] == date, 'sales_scaled'] = predictions

    # Calculate model performance (Mean Squared Error, R-squared, MAE, and Explained Variance)
    mse = mean_squared_error(y_test, predictions)
    r_squared = r2_score(y_test, predictions)
    mae = mean_absolute_error(y_test, predictions)
    explained_var = explained_variance_score(y_test, predictions)

    # Output cross-validation model performance (Mean Squared Error and R-squared)
    cv_mse = mean_squared_error(y_train, cv_predictions)

    # Store performance metrics for each day
    performance_metrics['MSE'].append(mse)
    performance_metrics['R²'].append(r_squared)
    performance_metrics['MAE'].append(mae)
    performance_metrics['Explained Variance'].append(explained_var)
    performance_metrics['CV_MSE'].append(cv_mse)

    # Output model performance
    print(f'Model on date {date}: MSE = {mse:.6f}, R² = {r_squared:.6f}, MAE = {mae:.6f}, Explained Variance = {explained_v

# Calculate the average value for each metric
avg_performance = {metric: sum(values) / len(values) for metric, values in performance_metrics.items()}

# Output average performance
print(f'Average Performance over 16 Days:')
for metric, value in avg_performance.items():
    print(f'{metric}: {value:.6f}')
```

```
Model on date 2017-07-31 00:00:00: MSE = 0.000013, R² = 0.941765, MAE = 0.000990, Explained Variance = 0.942103, CV_MSE =
0.000022
Model on date 2017-08-01 00:00:00: MSE = 0.000038, R² = 0.857485, MAE = 0.001657, Explained Variance = 0.861146, CV_MSE =
0.000022
Model on date 2017-08-02 00:00:00: MSE = 0.000032, R² = 0.887101, MAE = 0.001470, Explained Variance = 0.888003, CV_MSE =
0.000022
Model on date 2017-08-03 00:00:00: MSE = 0.000015, R² = 0.895540, MAE = 0.001141, Explained Variance = 0.896425, CV_MSE =
0.000022
Model on date 2017-08-04 00:00:00: MSE = 0.000020, R² = 0.893856, MAE = 0.001321, Explained Variance = 0.893857, CV_MSE =
0.000022
Model on date 2017-08-05 00:00:00: MSE = 0.000044, R² = 0.844945, MAE = 0.001748, Explained Variance = 0.848727, CV_MSE =
0.000022
Model on date 2017-08-06 00:00:00: MSE = 0.000062, R² = 0.819799, MAE = 0.002162, Explained Variance = 0.826680, CV_MSE =
0.000022
Model on date 2017-08-07 00:00:00: MSE = 0.000017, R² = 0.905030, MAE = 0.001236, Explained Variance = 0.905050, CV_MSE =
0.000022
Model on date 2017-08-08 00:00:00: MSE = 0.000015, R² = 0.892241, MAE = 0.001167, Explained Variance = 0.894984, CV_MSE =
0.000022
Model on date 2017-08-09 00:00:00: MSE = 0.000019, R² = 0.881936, MAE = 0.001223, Explained Variance = 0.885424, CV_MSE =
0.000022
Model on date 2017-08-10 00:00:00: MSE = 0.000016, R² = 0.861854, MAE = 0.001290, Explained Variance = 0.868197, CV_MSE =
0.000022
Model on date 2017-08-11 00:00:00: MSE = 0.000020, R² = 0.895036, MAE = 0.001371, Explained Variance = 0.895064, CV_MSE =
0.000022
Model on date 2017-08-12 00:00:00: MSE = 0.000021, R² = 0.882672, MAE = 0.001242, Explained Variance = 0.882673, CV_MSE =
0.000022
Model on date 2017-08-13 00:00:00: MSE = 0.000028, R² = 0.877239, MAE = 0.001447, Explained Variance = 0.878311, CV_MSE =
0.000022
Model on date 2017-08-14 00:00:00: MSE = 0.000015, R² = 0.909705, MAE = 0.001165, Explained Variance = 0.909764, CV_MSE =
0.000022
Model on date 2017-08-15 00:00:00: MSE = 0.000014, R² = 0.910281, MAE = 0.001184, Explained Variance = 0.910989, CV_MSE =
0.000022
Average Performance over 16 Days:
MSE: 0.000024
R²: 0.884780
MAE: 0.001363
Explained Variance: 0.886712
CV_MSE: 0.000022
```

In [57]:

```python
# Fixed Window 90
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, explained_variance_score
from sklearn.model_selection import cross_val_predict

# Define the prediction date range
prediction_dates = pd.date_range(start='2017-07-31', end='2017-08-15')
predict_data = concatenated_data_str.copy()

# Store performance metrics for each day
performance_metrics = {'MSE': [], 'R²': [], 'MAE': [], 'Explained Variance': [], 'CV_MSE':[]}

# Calculate the date for the same day two years ago and one year ago
two_years_ago = prediction_dates[0] - pd.DateOffset(years=2)
one_year_ago = prediction_dates[0] - pd.DateOffset(years=1)

# Build the training set
train_set = predict_data.loc[
    (
        (predict_data['date'] >= two_years_ago - pd.DateOffset(days=90)) &
        (predict_data['date'] < two_years_ago)
    ) |
    (
        (predict_data['date'] >= one_year_ago - pd.DateOffset(days=90)) &
        (predict_data['date'] < one_year_ago)
    ) |
    (
        (predict_data['date'] >= prediction_dates[0] - pd.DateOffset(days=90)) &
        (predict_data['date'] < prediction_dates[0])
    )
].reset_index(drop=True)

# Extract features and target variable
X_train = train_set[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]  # Choose appropriate features
y_train = train_set['sales_scaled']

# Build the Random Forest regression model
model = RandomForestRegressor(n_estimators=100, random_state=42)

# Perform cross-validation prediction
cv_predictions = cross_val_predict(model, X_train, y_train, cv=5)

# Fit the model
model.fit(X_train, y_train)

# Loop over prediction dates
for date in prediction_dates:
    # Build the test set
    test_set = concatenated_data_str[(concatenated_data_str['date'] == date)]

    # Extract test set features
    X_test = test_set[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]  # Choose appropriate features
    y_test = test_set['sales_scaled']

    # Make predictions
    predictions = model.predict(X_test)

    # Update predictions in predict_data
```

```
    predict_data.loc[predict_data['date'] == date, 'sales_scaled'] = predictions

    # Calculate model performance (Mean Squared Error, R-squared, MAE, and Explained Variance)
    mse = mean_squared_error(y_test, predictions)
    r_squared = r2_score(y_test, predictions)
    mae = mean_absolute_error(y_test, predictions)
    explained_var = explained_variance_score(y_test, predictions)

    # Output cross-validation model performance (Mean Squared Error and R-squared)
    cv_mse = mean_squared_error(y_train, cv_predictions)

    # Store performance metrics for each day
    performance_metrics['MSE'].append(mse)
    performance_metrics['R²'].append(r_squared)
    performance_metrics['MAE'].append(mae)
    performance_metrics['Explained Variance'].append(explained_var)
    performance_metrics['CV_MSE'].append(cv_mse)

    # Output model performance
    print(f'Model on date {date}: MSE = {mse:.6f}, R² = {r_squared:.6f}, MAE = {mae:.6f}, Explained Variance = {explained_v

# Calculate the average value for each metric
avg_performance = {metric: sum(values) / len(values) for metric, values in performance_metrics.items()}

# Output average performance
print(f'Average Performance over 16 Days:')
for metric, value in avg_performance.items():
    print(f'{metric}: {value:.6f}')
```

```
Model on date 2017-07-31 00:00:00: MSE = 0.000013, R² = 0.943166, MAE = 0.001015, Explained Variance = 0.943738, CV_MSE =
0.000023
Model on date 2017-08-01 00:00:00: MSE = 0.000024, R² = 0.910943, MAE = 0.001432, Explained Variance = 0.914586, CV_MSE =
0.000023
Model on date 2017-08-02 00:00:00: MSE = 0.000025, R² = 0.909613, MAE = 0.001160, Explained Variance = 0.910142, CV_MSE =
0.000023
Model on date 2017-08-03 00:00:00: MSE = 0.000008, R² = 0.945349, MAE = 0.000961, Explained Variance = 0.946486, CV_MSE =
0.000023
Model on date 2017-08-04 00:00:00: MSE = 0.000011, R² = 0.944926, MAE = 0.000987, Explained Variance = 0.944926, CV_MSE =
0.000023
Model on date 2017-08-05 00:00:00: MSE = 0.000028, R² = 0.899477, MAE = 0.001383, Explained Variance = 0.903470, CV_MSE =
0.000023
Model on date 2017-08-06 00:00:00: MSE = 0.000047, R² = 0.862279, MAE = 0.001917, Explained Variance = 0.869315, CV_MSE =
0.000023
Model on date 2017-08-07 00:00:00: MSE = 0.000008, R² = 0.957618, MAE = 0.000871, Explained Variance = 0.957618, CV_MSE =
0.000023
Model on date 2017-08-08 00:00:00: MSE = 0.000010, R² = 0.927626, MAE = 0.001024, Explained Variance = 0.930038, CV_MSE =
0.000023
Model on date 2017-08-09 00:00:00: MSE = 0.000010, R² = 0.937612, MAE = 0.001057, Explained Variance = 0.941109, CV_MSE =
0.000023
Model on date 2017-08-10 00:00:00: MSE = 0.000013, R² = 0.892114, MAE = 0.001228, Explained Variance = 0.898129, CV_MSE =
0.000023
Model on date 2017-08-11 00:00:00: MSE = 0.000014, R² = 0.925948, MAE = 0.001141, Explained Variance = 0.925989, CV_MSE =
0.000023
Model on date 2017-08-12 00:00:00: MSE = 0.000012, R² = 0.932456, MAE = 0.000969, Explained Variance = 0.932458, CV_MSE =
0.000023
Model on date 2017-08-13 00:00:00: MSE = 0.000020, R² = 0.910734, MAE = 0.001257, Explained Variance = 0.911831, CV_MSE =
0.000023
Model on date 2017-08-14 00:00:00: MSE = 0.000008, R² = 0.951711, MAE = 0.000898, Explained Variance = 0.951780, CV_MSE =
0.000023
Model on date 2017-08-15 00:00:00: MSE = 0.000009, R² = 0.938866, MAE = 0.000996, Explained Variance = 0.939441, CV_MSE =
0.000023
Average Performance over 16 Days:
MSE: 0.000016
R²: 0.924402
MAE: 0.001143
Explained Variance: 0.926316
CV_MSE: 0.000023
```

In [61]:

```
# Fixed Window 90 Hyperparameter Tuning
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, explained_variance_score
from sklearn.model_selection import cross_val_predict

from sklearn.model_selection import RandomizedSearchCV

# Define the prediction date range
prediction_dates = pd.date_range(start='2017-07-31', end='2017-08-15')
predict_data = concatenated_data_str.copy()

# Store performance metrics for each day
performance_metrics = {'MSE': [], 'R²': [], 'MAE': [], 'Explained Variance': [], 'CV_MSE':[]}

# Calculate the date for the same day two years ago and one year ago
two_years_ago = prediction_dates[0] - pd.DateOffset(years=2)
one_year_ago = prediction_dates[0] - pd.DateOffset(years=1)

# Build the training set
train_set = predict_data.loc[
    (
        (predict_data['date'] >= two_years_ago - pd.DateOffset(days=90)) &
        (predict_data['date'] < two_years_ago)
    ) |
    (
        (predict_data['date'] >= one_year_ago - pd.DateOffset(days=90)) &
        (predict_data['date'] < one_year_ago)
    ) |
    (
        (predict_data['date'] >= prediction_dates[0] - pd.DateOffset(days=90)) &
```

```
        (predict_data['date'] < prediction_dates[0])
    )
].reset_index(drop=True)

# Extract features and target variable
X_train = train_set[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]  # Choose appropriate features
y_train = train_set['sales_scaled']
```

In [78]:

```
from sklearn.model_selection import train_test_split
# Define the hyperparameter search space
param_dist = {
    'n_estimators': [50, 100, 150, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2', None]
}

# Build the Random Forest regression model
base_model = RandomForestRegressor(random_state=42)

# Perform random search
random_search = RandomizedSearchCV(base_model, param_distributions=param_dist, n_iter=10, cv=3, random_state=42)

# Use a smaller dataset for hyperparameter tuning
random_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best Hyperparameters:", random_search.best_params_)
```

```
/Users/evelyn/anaconda3/lib/python3.11/site-packages/sklearn/model_selection/_validation.py:425: FitFailedWarning:
3 fits failed out of a total of 30.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
3 fits failed with the following error:
Traceback (most recent call last):
  File "/Users/evelyn/anaconda3/lib/python3.11/site-packages/sklearn/model_selection/_validation.py", line 732, in _fit_an
d_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/Users/evelyn/anaconda3/lib/python3.11/site-packages/sklearn/base.py", line 1144, in wrapper
    estimator._validate_params()
  File "/Users/evelyn/anaconda3/lib/python3.11/site-packages/sklearn/base.py", line 637, in _validate_params
    validate_parameter_constraints(
  File "/Users/evelyn/anaconda3/lib/python3.11/site-packages/sklearn/utils/_param_validation.py", line 95, in validate_par
ameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' parameter of RandomForestRegressor must be an in
t in the range [1, inf), a float in the range (0.0, 1.0], a str among {'log2', 'sqrt'} or None. Got 'auto' instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/Users/evelyn/anaconda3/lib/python3.11/site-packages/sklearn/model_selection/_search.py:976: UserWarning: One or more of t
he test scores are non-finite: [0.84874581 0.90029584 0.89477062 0.8910238  0.89464744 0.89428513
 0.87822642        nan 0.8954817  0.89387962]
  warnings.warn(
Best Hyperparameters: {'n_estimators': 150, 'min_samples_split': 10, 'min_samples_leaf': 1, 'max_features': None, 'max_dep
th': None}
```

In [84]:

```
# optimised model for 90-day fixed window
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, explained_variance_score
from sklearn.model_selection import cross_val_predict

# Define the prediction date range
prediction_dates = pd.date_range(start='2017-07-31', end='2017-08-15')
predict_data = concatenated_data_str.copy()

# Store performance metrics for each day
performance_metrics = {'MSE': [], 'R²': [], 'MAE': [], 'Explained Variance': [], 'CV_MSE':[]}

# Calculate the date for the same day two years ago and one year ago
two_years_ago = prediction_dates[0] - pd.DateOffset(years=2)
one_year_ago = prediction_dates[0] - pd.DateOffset(years=1)

# Build the training set
train_set = predict_data.loc[
    (
        (predict_data['date'] >= two_years_ago - pd.DateOffset(days=90)) &
        (predict_data['date'] < two_years_ago)
    ) |
    (
        (predict_data['date'] >= one_year_ago - pd.DateOffset(days=90)) &
        (predict_data['date'] < one_year_ago)
    ) |
    (
        (predict_data['date'] >= prediction_dates[0] - pd.DateOffset(days=90)) &
        (predict_data['date'] < prediction_dates[0])
    )
].reset_index(drop=True)

# Extract features and target variable
X_train = train_set[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]  # Choose appropriate features
y_train = train_set['sales_scaled']

# Build the Random Forest regression model
```

```
model = RandomForestRegressor(n_estimators=150, min_samples_split = 10, max_features = None, random_state=42)

# Perform cross-validation prediction
cv_predictions = cross_val_predict(model, X_train, y_train, cv=5)

# Fit the model
model.fit(X_train, y_train)

# Loop over prediction dates
for date in prediction_dates:
    # Build the test set
    test_set = concatenated_data_str[(concatenated_data_str['date'] == date)]

    # Extract test set features
    X_test = test_set[['store_nbr', 'product_type_encoded', 'special_offer_scaled']]  # Choose appropriate features
    y_test = test_set['sales_scaled']

    # Make predictions
    predictions = model.predict(X_test)

    # Update predictions in predict_data
    predict_data.loc[predict_data['date'] == date, 'sales_scaled'] = predictions

    # Calculate model performance (Mean Squared Error, R-squared, MAE, and Explained Variance)
    mse = mean_squared_error(y_test, predictions)
    r_squared = r2_score(y_test, predictions)
    mae = mean_absolute_error(y_test, predictions)
    explained_var = explained_variance_score(y_test, predictions)

    # Output cross-validation model performance (Mean Squared Error and R-squared)
    cv_mse = mean_squared_error(y_train, cv_predictions)

    # Store performance metrics for each day
    performance_metrics['MSE'].append(mse)
    performance_metrics['R²'].append(r_squared)
    performance_metrics['MAE'].append(mae)
    performance_metrics['Explained Variance'].append(explained_var)
    performance_metrics['CV_MSE'].append(cv_mse)

    # Output model performance
    print(f'Model on date {date}: MSE = {mse:.6f}, R² = {r_squared:.6f}, MAE = {mae:.6f}, Explained Variance = {explained_v

# Calculate the average value for each metric
avg_performance = {metric: sum(values) / len(values) for metric, values in performance_metrics.items()}

# Output average performance
print(f'Average Performance over 16 Days:')
for metric, value in avg_performance.items():
    print(f'{metric}: {value:.6f}')
```

```
Model on date 2017-07-31 00:00:00: MSE = 0.000010, R² = 0.954474, MAE = 0.000933, Explained Variance = 0.955123, CV_MSE =
0.000021
Model on date 2017-08-01 00:00:00: MSE = 0.000021, R² = 0.919403, MAE = 0.001386, Explained Variance = 0.923161, CV_MSE =
0.000021
Model on date 2017-08-02 00:00:00: MSE = 0.000025, R² = 0.911585, MAE = 0.001144, Explained Variance = 0.912090, CV_MSE =
0.000021
Model on date 2017-08-03 00:00:00: MSE = 0.000007, R² = 0.951886, MAE = 0.000919, Explained Variance = 0.952995, CV_MSE =
0.000021
Model on date 2017-08-04 00:00:00: MSE = 0.000009, R² = 0.952314, MAE = 0.000923, Explained Variance = 0.952317, CV_MSE =
0.000021
Model on date 2017-08-05 00:00:00: MSE = 0.000028, R² = 0.901065, MAE = 0.001353, Explained Variance = 0.905117, CV_MSE =
0.000021
Model on date 2017-08-06 00:00:00: MSE = 0.000047, R² = 0.863080, MAE = 0.001909, Explained Variance = 0.870208, CV_MSE =
0.000021
Model on date 2017-08-07 00:00:00: MSE = 0.000006, R² = 0.965793, MAE = 0.000808, Explained Variance = 0.965796, CV_MSE =
0.000021
Model on date 2017-08-08 00:00:00: MSE = 0.000009, R² = 0.934976, MAE = 0.000992, Explained Variance = 0.937541, CV_MSE =
0.000021
Model on date 2017-08-09 00:00:00: MSE = 0.000009, R² = 0.947278, MAE = 0.001000, Explained Variance = 0.950783, CV_MSE =
0.000021
Model on date 2017-08-10 00:00:00: MSE = 0.000012, R² = 0.901235, MAE = 0.001207, Explained Variance = 0.907331, CV_MSE =
0.000021
Model on date 2017-08-11 00:00:00: MSE = 0.000013, R² = 0.933549, MAE = 0.001092, Explained Variance = 0.933574, CV_MSE =
0.000021
Model on date 2017-08-12 00:00:00: MSE = 0.000012, R² = 0.936739, MAE = 0.000933, Explained Variance = 0.936741, CV_MSE =
0.000021
Model on date 2017-08-13 00:00:00: MSE = 0.000019, R² = 0.916594, MAE = 0.001225, Explained Variance = 0.917634, CV_MSE =
0.000021
Model on date 2017-08-14 00:00:00: MSE = 0.000007, R² = 0.959271, MAE = 0.000841, Explained Variance = 0.959368, CV_MSE =
0.000021
Model on date 2017-08-15 00:00:00: MSE = 0.000008, R² = 0.947756, MAE = 0.000938, Explained Variance = 0.948331, CV_MSE =
0.000021
Average Performance over 16 Days:
MSE: 0.000015
R²: 0.931062
MAE: 0.001100
Explained Variance: 0.933007
CV_MSE: 0.000021
```

In [ ]:

Considering that this is a business decision-making problem, we have used both year-on-year and chain-on-year approaches for modelling training and predicting. And since different products and stores may have different seasonal and cyclical sales patterns, we will use the previous 15/30/90 days historical data and last year's same period data to make predictions and find the optimal model respectively.