FINAL

DATA70002 Assignment Student ID number:11489360

In [3]:
```
# For traditions sake start with the phrase 'Hello World'
print ('Hello World')
```

```
Hello World
```

In [4]:
```
#Import the required packages
import pandas as pd
from numpy.random import seed
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, calinski_harabasz_score
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale, MinMaxScaler
import seaborn as sns
import matplotlib.pyplot as plt
```

In [5]:
```
# Remove warning messages
import warnings
warnings.filterwarnings('ignore')
```

1. Review Data Provided

In [7]:
```
# Read in the CSV file and name df
store= pd.read_csv("store.csv")
```

In [8]:
```
# Begin review of df
store.shape
```

Out[8]:
```
(1115, 10)
```

In [9]:
```
# Df has 1115 rows and 10 columns(cols)
# Continue review of df
# Display df's first 5 rows
store.head().T
```

Out[9]:

	0	1	2	3	4
Store	1	2	3	4	5
StoreType	c	a	a	c	a
Assortment	a	a	a	c	a
CompetitionDistance	1270.0	570.0	14130.0	620.0	29910.0
CompetitionOpenSinceMonth	9.0	11.0	12.0	9.0	4.0
CompetitionOpenSinceYear	2008.0	2007.0	2006.0	2009.0	2015.0
Promo2	0	1	1	0	0
Promo2SinceWeek	NaN	13.0	14.0	NaN	NaN
Promo2SinceYear	NaN	2010.0	2011.0	NaN	NaN
PromoInterval	NaN	Jan,Apr,Jul,Oct	Jan,Apr,Jul,Oct	NaN	NaN

In [10]:
```
# Display df's last 10 rows
store.tail(10).T
```

Out[10]:

	1105	1106	1107	1108	1109
Store	1106	1107	1108	1109	1110
StoreType	a	a	a	c	c
Assortment	c	a	a	a	c
CompetitionDistance	5330.0	1400.0	540.0	3490.0	900.0
CompetitionOpenSinceMonth	9.0	6.0	4.0	4.0	9.0
CompetitionOpenSinceYear	2011.0	2012.0	2004.0	2011.0	2010.0
Promo2	1	1	0	1	0
Promo2SinceWeek	31.0	13.0	NaN	22.0	NaN
Promo2SinceYear	2013.0	2010.0	NaN	2012.0	NaN
PromoInterval	Jan,Apr,Jul,Oct	Jan,Apr,Jul,Oct	NaN	Jan,Apr,Jul,Oct	NaN

In [11]:
```
# There are numerous columns that are not visible in the df review so change
store_transposed = store.T
store.T.head(10)
```

Out[11]:

	0	1	2	3	4	5
Store	1	2	3	4	5	6
StoreType	c	a	a	c	a	a
Assortment	a	a	a	c	a	a
CompetitionDistance	1270.0	570.0	14130.0	620.0	29910.0	310.0
CompetitionOpenSinceMonth	9.0	11.0	12.0	9.0	4.0	12.0
CompetitionOpenSinceYear	2008.0	2007.0	2006.0	2009.0	2015.0	2013.0
Promo2	0	1	1	0	0	0
Promo2SinceWeek	NaN	13.0	14.0	NaN	NaN	NaN
Promo2SinceYear	NaN	2010.0	2011.0	NaN	NaN	NaN
PromoInterval	NaN	Jan,Apr,Jul,Oct	Jan,Apr,Jul,Oct	NaN	NaN	NaN

10 rows × 1115 columns

In [12]:
```
# Continue review of df

# Display df's info -shape, columns, non-null columns and datatypes(dtype)
store.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1115 entries, 0 to 1114
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Store                      1115 non-null   int64  
 1   StoreType                  1115 non-null   object 
 2   Assortment                 1115 non-null   object 
 3   CompetitionDistance        1112 non-null   float64
 4   CompetitionOpenSinceMonth  761 non-null    float64
 5   CompetitionOpenSinceYear   761 non-null    float64
 6   Promo2                     1115 non-null   int64  
 7   Promo2SinceWeek            571 non-null    float64
 8   Promo2SinceYear            571 non-null    float64
 9   PromoInterval              571 non-null    object 
dtypes: float64(5), int64(2), object(3)
memory usage: 87.2+ KB
```

In [13]:
```
# Check for duplicates
duplicates = store[store.duplicated()]

# If there are duplicates, it will print them
if not duplicates.empty:
    print("Duplicates found:\n", duplicates)
else:
    print("No duplicates found.")
```

```
No duplicates found.
```

In [14]:
```
# Sum up the distances for non-null values
store['CompetitionDistance'].sum()
```

Out[14]:
```
6010250.0
```

In [15]:
```
# Calculate mean long hand
+6010250.0/1112
```

Out[15]:
```
5404.901079136691
```

In [16]:
```
# Filter rows with null values for 'CompetitionDistance'
rows_with_null_distance = store[store['CompetitionDistance'].isnull()]

# Display the filtered DataFrame
print(rows_with_null_distance)
```

```
     Store StoreType Assortment  CompetitionDistance  \
290    291         d          a                  NaN
621    622         a          c                  NaN
878    879         d          a                  NaN

     CompetitionOpenSinceMonth  CompetitionOpenSinceYear  Promo2  \
290                        NaN                       NaN       0
621                        NaN                       NaN       0
878                        NaN                       NaN       1

     Promo2SinceWeek  Promo2SinceYear    PromoInterval
290              NaN              NaN              NaN
621              NaN              NaN              NaN
878              5.0           2013.0  Feb,May,Aug,Nov
```

In [17]:
```
# Calculate the mean of CompetitionDistance
competition_distance_mean = round(store['CompetitionDistance'].mean(), 0)

# Fill missing values with the mean
store['CompetitionDistance'] = store['CompetitionDistance'].fillna(competit
```

In [18]:
```
# Continue review of df
# Review details of one row for reasonableness
store.loc[290]
```

Out[18]:
```
Store                        291
StoreType                      d
Assortment                     a
CompetitionDistance       5405.0
CompetitionOpenSinceMonth    NaN
CompetitionOpenSinceYear     NaN
Promo2                         0
Promo2SinceWeek              NaN
Promo2SinceYear              NaN
PromoInterval                NaN
Name: 290, dtype: object
```

In [19]:
```
# Continue review of df
# Review details of one row for reasonableness
store.loc[621]
```

Out[19]:
```
Store                        622
StoreType                      a
Assortment                     c
CompetitionDistance       5405.0
CompetitionOpenSinceMonth    NaN
CompetitionOpenSinceYear     NaN
Promo2                         0
Promo2SinceWeek              NaN
Promo2SinceYear              NaN
PromoInterval                NaN
Name: 621, dtype: object
```

In [20]:
```
store.nunique()
```

Out[20]:
```
Store                        1115
StoreType                       4
Assortment                      3
CompetitionDistance           655
CompetitionOpenSinceMonth      12
CompetitionOpenSinceYear       23
Promo2                          2
Promo2SinceWeek                24
Promo2SinceYear                 7
PromoInterval                   3
dtype: int64
```

In [21]:
```
# Display df's info -shape, columns, non-null columns and datatypes(dtype)
store.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1115 entries, 0 to 1114
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Store                      1115 non-null   int64  
 1   StoreType                  1115 non-null   object 
 2   Assortment                 1115 non-null   object 
 3   CompetitionDistance        1115 non-null   float64
 4   CompetitionOpenSinceMonth  761 non-null    float64
 5   CompetitionOpenSinceYear   761 non-null    float64
 6   Promo2                     1115 non-null   int64  
 7   Promo2SinceWeek            571 non-null    float64
 8   Promo2SinceYear            571 non-null    float64
 9   PromoInterval              571 non-null    object 
dtypes: float64(5), int64(2), object(3)
memory usage: 87.2+ KB
```

In [22]:
```
# Filter rows for years 2013, 2014, and 2015
competition_years = store[store['CompetitionOpenSinceYear'].isin([2013, 201

# Count occurrences of each year
year_counts = competition_years['CompetitionOpenSinceYear'].value_counts()

# Display the counts
print(year_counts)
```

```
CompetitionOpenSinceYear
2013.0    83
2014.0    70
2015.0    38
Name: count, dtype: int64
```

In [23]:
```
# Count Promo2 values
store["Promo2"].value_counts()
```

Out[23]:
```
Promo2
1    571
0    544
Name: count, dtype: int64
```

In [24]:
```
# Filter rows for years 2013, 2014, and 2015
competition_years = store[store['Promo2SinceYear'].isin([2013, 2014, 2015])]

# Count occurrences of each year
year_counts = competition_years['Promo2SinceYear'].value_counts()

# Display the counts
print(year_counts)
```

```
Promo2SinceYear
2013.0    120
2014.0     95
2015.0     10
Name: count, dtype: int64
```

In [25]:
```
# Find the minimum and maximum values
min_week = store['Promo2SinceWeek'].min()
max_week = store['Promo2SinceWeek'].max()

# Display the range
print("Range of Promo2SinceWeek:", min_week, "-", max_week)
```

```
Range of Promo2SinceWeek: 1.0 - 50.0
```

In [26]:
```
# Find the minimum and maximum values
min_week = store['Promo2SinceYear'].min()
max_week = store['Promo2SinceYear'].max()

# Display the range
print("Range of Promo2SinceYear:", min_week, "-", max_week)
```

```
Range of Promo2SinceYear: 2009.0 - 2015.0
```

In [27]:
```
# Filter the DataFrame for the year 2015
weeks_2015 = store[store['Promo2SinceYear'] == 2015]['Promo2SinceWeek']

# Display the weeks in 2015
print("Weeks in 2015:", weeks_2015)
```

```
Weeks in 2015: 27      6.0
264    14.0
330    14.0
427    23.0
628    23.0
748    14.0
871    23.0
874    18.0
875    18.0
945    14.0
Name: Promo2SinceWeek, dtype: float64
```

In [28]:
```
# Storetype value count
store["StoreType"].value_counts()
```

Out[28]:
```
StoreType
a    602
d    348
c    148
b     17
Name: count, dtype: int64
```

In [29]:
```
# Assortment value count
store["Assortment"].value_counts()
```

Out[29]:
```
Assortment
a    593
c    513
b      9
Name: count, dtype: int64
```

In [30]:
```
# Group two columns together
combinations_count = store.groupby(['StoreType', 'Assortment']).size()

print(combinations_count)
```

```
StoreType  Assortment
a          a             381
           c             221
b          a               7
           b               9
           c               1
c          a              77
           c              71
d          a             128
           c             220
dtype: int64
```

In [31]:
```
# Check no zero count
zero_count = store.loc[store['CompetitionDistance'] == 0, 'CompetitionDistar

print("Count of zeros in the column:", zero_count)
```

```
Count of zeros in the column: 0
```

In [32]:
```
# Show non-null rows
null_rows = store[store['CompetitionDistance'].isnull()]

print("Null rows for the column:")
print(null_rows)
```

```
Null rows for the column:
Empty DataFrame
Columns: [Store, StoreType, Assortment, CompetitionDistance, CompetitionOpe
nSinceMonth, CompetitionOpenSinceYear, Promo2, Promo2SinceWeek, Promo2Since
Year, PromoInterval]
Index: []
```

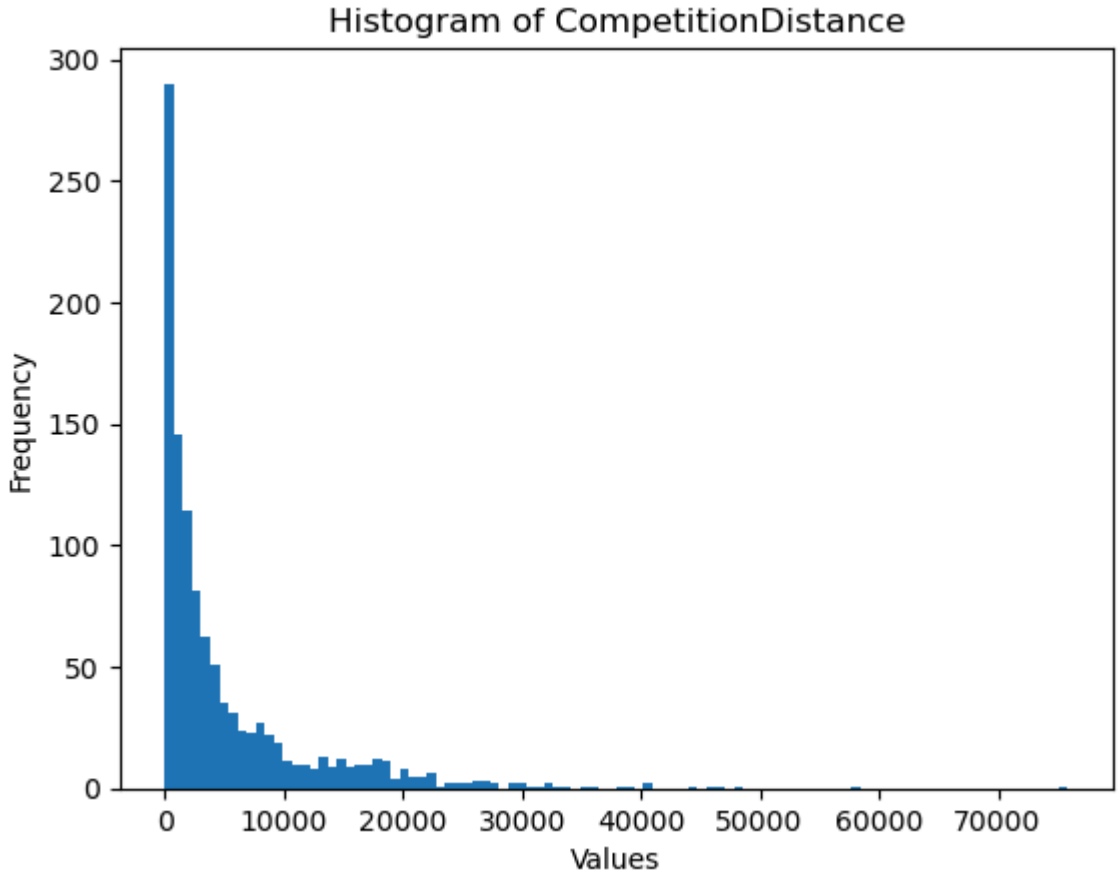
In [33]:
```
# Get statistics
column_stats = store['CompetitionDistance'].describe()

print("Summary statistics for the column:")
print(column_stats)
```

```
Summary statistics for the column:
count     1115.000000
mean      5404.901345
std       7652.849306
min         20.000000
25%        720.000000
50%       2330.000000
75%       6875.000000
max      75860.000000
Name: CompetitionDistance, dtype: float64
```

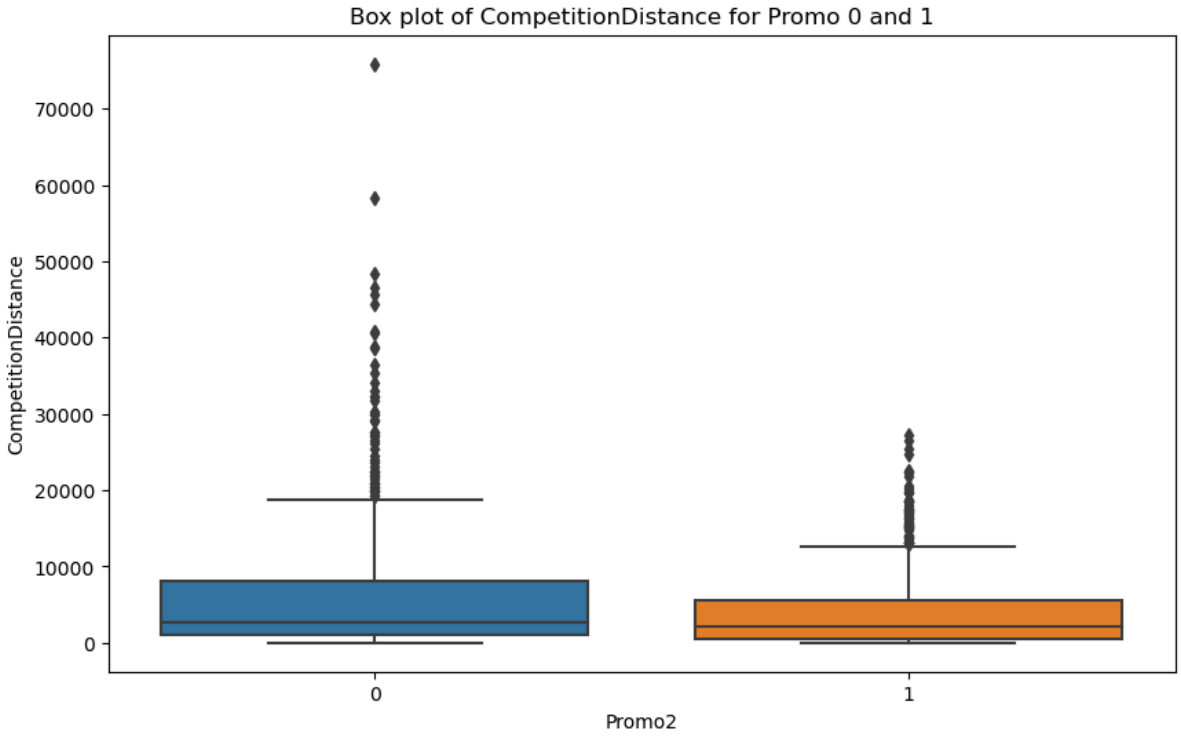
In [34]:
```
# Calculate the mode for the column
mode_result = store['CompetitionDistance'].mode()

# Extract the mode value(s)
mode_values = mode_result.values

# Count how many times the mode value appears in the column
mode_count = (store['CompetitionDistance'] == mode_values[0]).sum()

print("Mode value:", mode_values[0])
print("Count:", mode_count)
```

```
Mode value: 250.0
Count: 12
```

In [35]:
```
plt.hist(store['CompetitionDistance'], bins=100)  # Adjust the number of bi
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.title('Histogram of ' + 'CompetitionDistance')  # Adjust the title as ne
plt.show()
```



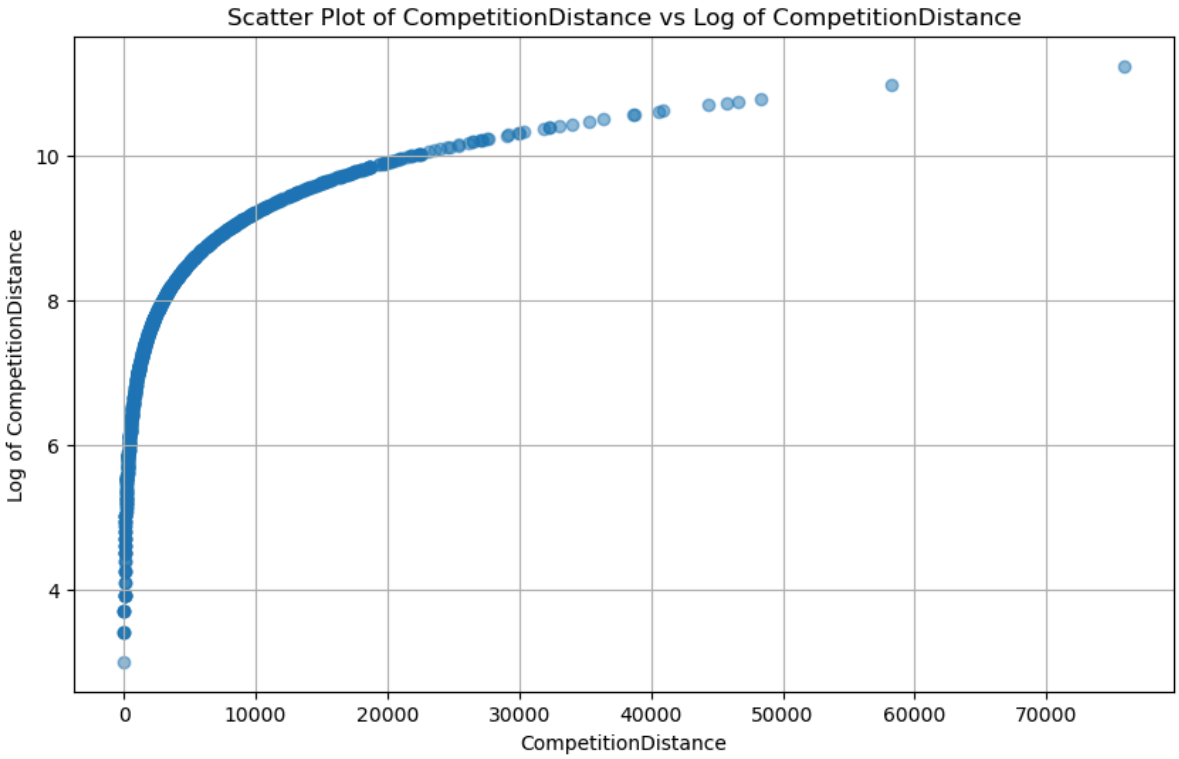
In [36]:
```
# Filter the DataFrame based on 'Promo' values
promo_0 = store[store['Promo2'] == 0]
promo_1 = store[store['Promo2'] == 1]

# Create a box plot
plt.figure(figsize=(10, 6))
sns.boxplot(x='Promo2', y='CompetitionDistance', data=store)
plt.title('Box plot of CompetitionDistance for Promo 0 and 1')
plt.xlabel('Promo2')
plt.ylabel('CompetitionDistance')
plt.show()
```



In [37]:
```
# Calculate the correlation
correlation = store['Promo2'].corr(store['CompetitionDistance'])

# Display the correlation
print("Correlation between Promo2 and CompetitionDistance:", correlation)
```

```
Correlation between Promo2 and CompetitionDistance: -0.14551726681979368
```

In [38]:
```
import numpy as np
import matplotlib.pyplot as plt

# Scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(store['CompetitionDistance'], np.log(store['CompetitionDistance
plt.xlabel('CompetitionDistance')
plt.ylabel('Log of CompetitionDistance')
plt.title('Scatter Plot of CompetitionDistance vs Log of CompetitionDistance
plt.grid(True)
plt.show()
```



In [39]:
```
# Calculate the logarithm of CompetitionDistance
log_competition_distance = np.log(store['CompetitionDistance'])

# Print maximum and minimum values
print("Maximum log(CompetitionDistance):", log_competition_distance.max())
print("Minimum log(CompetitionDistance):", log_competition_distance.min())
```

```
Maximum log(CompetitionDistance): 11.23664481524289
Minimum log(CompetitionDistance): 2.995732273553991
```

In [40]:
```
# Calculate deciles
deciles = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
decile_values = store['CompetitionDistance'].quantile(deciles)

for i, decile in enumerate(deciles):
    print(f"Decile {decile * 10}: {decile_values.iloc[i]}")
```

```
Decile 1.0: 250.0
Decile 2.0: 520.0
Decile 3.0: 1043.9999999999998
Decile 4.0: 1600.0
Decile 5.0: 2330.0
Decile 6.0: 3464.0
Decile 7.0: 5338.0
Decile 8.0: 8654.0
Decile 9.0: 15616.000000000005
```

In [41]:
```
# Calculate deciles
deciles = store['CompetitionDistance'].quantile([0.1, 0.2, 0.3, 0.4, 0.5, 0.

# Define a function to assign decile rank
def assign_decile(distance):
    for i, decile_value in enumerate(deciles):
        if distance <= decile_value:
            return i + 1  # Decile ranks start from 1
    return 10  # Assign 10 if distance is greater than the maximum decile va

# Create a new DataFrame 'store2' as a copy of 'store'
store2 = store.copy()
```

```
# Create a new column 'CompetitionDistanceDecile' with decile ranks in the
store2['CompetitionDistanceDecile'] = store['CompetitionDistance'].apply(ass

# Display the DataFrame with the new column
print(store2)
```

```
      Store StoreType Assortment  CompetitionDistance  \
0         1         c          a               1270.0
1         2         a          a                570.0
2         3         a          a              14130.0
3         4         c          c                620.0
4         5         a          a              29910.0
...     ...       ...        ...                  ...
1110   1111         a          a               1900.0
1111   1112         c          c               1880.0
1112   1113         a          c               9260.0
1113   1114         a          c                870.0
1114   1115         d          c               5350.0

      CompetitionOpenSinceMonth  CompetitionOpenSinceYear  Promo2  \
0                           9.0                    2008.0       0
1                          11.0                    2007.0       1
2                          12.0                    2006.0       1
3                           9.0                    2009.0       0
4                           4.0                    2015.0       0
...                         ...                       ...     ...
1110                        6.0                    2014.0       1
1111                        4.0                    2006.0       0
1112                        NaN                       NaN       0
1113                        NaN                       NaN       0
1114                        NaN                       NaN       1

      Promo2SinceWeek  Promo2SinceYear     PromoInterval  \
0                 NaN              NaN               NaN
1                13.0           2010.0   Jan,Apr,Jul,Oct
2                14.0           2011.0   Jan,Apr,Jul,Oct
3                 NaN              NaN               NaN
4                 NaN              NaN               NaN
...               ...              ...               ...
1110             31.0           2013.0   Jan,Apr,Jul,Oct
1111              NaN              NaN               NaN
1112              NaN              NaN               NaN
1113              NaN              NaN               NaN
1114             22.0           2012.0  Mar,Jun,Sept,Dec

      CompetitionDistanceDecile
0                             4
1                             3
2                             9
3                             3
4                            10
...                         ...
1110                          5
1111                          5
1112                          9
1113                          3
1114                          8

[1115 rows x 11 columns]
```

In [42]:
```
# Get statistics
column_stats = store['CompetitionOpenSinceYear'].describe()
```

```
print("Summary statistics for the column:")
print(column_stats)
```

```
Summary statistics for the column:
count     761.000000
mean     2008.668857
std         6.195983
min      1900.000000
25%      2006.000000
50%      2010.000000
75%      2013.000000
max      2015.000000
Name: CompetitionOpenSinceYear, dtype: float64
```

In [43]:
```
# Calculate deciles
deciles = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
decile_values = store['CompetitionOpenSinceYear'].quantile(deciles)

for i, decile in enumerate(deciles):
    print(f"Decile {decile * 10}: {decile_values.iloc[i]}")
```

```
Decile 1.0: 2003.0
Decile 2.0: 2005.0
Decile 3.0: 2007.0
Decile 4.0: 2008.0
Decile 5.0: 2010.0
Decile 6.0: 2011.0
Decile 7.0: 2012.0
Decile 8.0: 2013.0
Decile 9.0: 2014.0
```

In [44]:
```
 store["Promo2"].value_counts()
```

Out[44]:
```
Promo2
1    571
0    544
Name: count, dtype: int64
```

In [45]:
```
# Get statistics
column_stats = store['Promo2SinceYear'].describe()

print("Summary statistics for the column:")
print(column_stats)
```

```
Summary statistics for the column:
count     571.000000
mean     2011.763573
std         1.674935
min      2009.000000
25%      2011.000000
50%      2012.000000
75%      2013.000000
max      2015.000000
Name: Promo2SinceYear, dtype: float64
```

In [46]:
```
# store2 info
store2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1115 entries, 0 to 1114
Data columns (total 11 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Store                      1115 non-null   int64
 1   StoreType                  1115 non-null   object
 2   Assortment                 1115 non-null   object
 3   CompetitionDistance        1115 non-null   float64
 4   CompetitionOpenSinceMonth  761 non-null    float64
 5   CompetitionOpenSinceYear   761 non-null    float64
 6   Promo2                     1115 non-null   int64
 7   Promo2SinceWeek            571 non-null    float64
 8   Promo2SinceYear            571 non-null    float64
 9   PromoInterval              571 non-null    object
 10  CompetitionDistanceDecile  1115 non-null   int64
dtypes: float64(5), int64(3), object(3)
memory usage: 95.9+ KB
```

In [47]:
```
# Create a new column 'NewCategory' by combining 'StoreType' and 'Assortment
store2['Type_Assort'] = store2['StoreType'].str.upper() + store['Assortment

# Display the DataFrame with the new column
print(store)
```

```
      Store StoreType Assortment  CompetitionDistance  \
0         1         c          a               1270.0
1         2         a          a                570.0
2         3         a          a              14130.0
3         4         c          c                620.0
4         5         a          a              29910.0
...     ...       ...        ...                 ...
1110   1111         a          a               1900.0
1111   1112         c          c               1880.0
1112   1113         a          c               9260.0
1113   1114         a          c                870.0
1114   1115         d          c               5350.0

      CompetitionOpenSinceMonth  CompetitionOpenSinceYear  Promo2  \
0                           9.0                    2008.0       0
1                          11.0                    2007.0       1
2                          12.0                    2006.0       1
3                           9.0                    2009.0       0
4                           4.0                    2015.0       0
...                         ...                       ...     ...
1110                        6.0                    2014.0       1
1111                        4.0                    2006.0       0
1112                        NaN                       NaN       0
1113                        NaN                       NaN       0
1114                        NaN                       NaN       1

      Promo2SinceWeek  Promo2SinceYear     PromoInterval
0                 NaN              NaN               NaN
1                13.0           2010.0   Jan,Apr,Jul,Oct
2                14.0           2011.0   Jan,Apr,Jul,Oct
3                 NaN              NaN               NaN
4                 NaN              NaN               NaN
...               ...              ...               ...
1110             31.0           2013.0   Jan,Apr,Jul,Oct
1111              NaN              NaN               NaN
1112              NaN              NaN               NaN
1113              NaN              NaN               NaN
1114             22.0           2012.0  Mar,Jun,Sept,Dec

[1115 rows x 10 columns]
```

In [48]:
```
store2['Type_Assort'].value_counts()
```

Out[48]:
```
Type_Assort
Aa    381
Ac    221
Dc    220
Da    128
Ca     77
Cc     71
Bb      9
Ba      7
Bc      1
Name: count, dtype: int64
```

In [49]:
```
# Define a dictionary mapping categories to numbers
promo_interval_mapping = {
    'Jan,Apr,Jul,Oct': 1,
    'Feb,May,Aug,Nov': 2,
    'Mar,Jun,Sept,Dec': 3
}

# Map the categories to numbers
```

```
store2['PromoInterval'] = store2['PromoInterval'].map(promo_interval_mapping
store2.tail().T
```

Out[49]:

	1110	1111	1112	1113	1114
Store	1111	1112	1113	1114	1115
StoreType	a	c	a	a	d
Assortment	a	c	c	c	c
CompetitionDistance	1900.0	1880.0	9260.0	870.0	5350.0
CompetitionOpenSinceMonth	6.0	4.0	NaN	NaN	NaN
CompetitionOpenSinceYear	2014.0	2006.0	NaN	NaN	NaN
Promo2	1	0	0	0	1
Promo2SinceWeek	31.0	NaN	NaN	NaN	22.0
Promo2SinceYear	2013.0	NaN	NaN	NaN	2012.0
PromoInterval	1.0	NaN	NaN	NaN	3.0
CompetitionDistanceDecile	5	5	9	3	8
Type_Assort	Aa	Cc	Ac	Ac	Dc

In [50]:
```
# Replace NaN values with 0
store2.fillna(0, inplace=True)
store2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1115 entries, 0 to 1114
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Store                      1115 non-null   int64
 1   StoreType                  1115 non-null   object
 2   Assortment                 1115 non-null   object
 3   CompetitionDistance        1115 non-null   float64
 4   CompetitionOpenSinceMonth  1115 non-null   float64
 5   CompetitionOpenSinceYear   1115 non-null   float64
 6   Promo2                     1115 non-null   int64
 7   Promo2SinceWeek            1115 non-null   float64
 8   Promo2SinceYear            1115 non-null   float64
 9   PromoInterval              1115 non-null   float64
 10  CompetitionDistanceDecile  1115 non-null   int64
 11  Type_Assort                1115 non-null   object
dtypes: float64(6), int64(3), object(3)
memory usage: 104.7+ KB
```

In [51]:
```
# Check the categories to numbers
store2.tail().T
```

Out[51]:

	1110	1111	1112	1113	1114
Store	1111	1112	1113	1114	1115
StoreType	a	c	a	a	d
Assortment	a	c	c	c	c
CompetitionDistance	1900.0	1880.0	9260.0	870.0	5350.0
CompetitionOpenSinceMonth	6.0	4.0	0.0	0.0	0.0
CompetitionOpenSinceYear	2014.0	2006.0	0.0	0.0	0.0
Promo2	1	0	0	0	1
Promo2SinceWeek	31.0	0.0	0.0	0.0	22.0
Promo2SinceYear	2013.0	0.0	0.0	0.0	2012.0
PromoInterval	1.0	0.0	0.0	0.0	3.0
CompetitionDistanceDecile	5	5	9	3	8
Type_Assort	Aa	Cc	Ac	Ac	Dc

In [52]:
```
store2['PromoInterval'].value_counts()
```

Out[52]:
```
PromoInterval
0.0    544
1.0    335
2.0    130
3.0    106
Name: count, dtype: int64
```

In [53]:
```
# Define a mapping for year to number
year_mapping = {0: -31, 2015: 0, 2014: 52, 2013: 104, 2012: 156, 2011: 208,


# Replace values in the 'Year' column using the mapping
store2['Promo2SinceYear'] = store2['Promo2SinceYear'].replace(year_mapping)

# Display the DataFrame with the updated 'Year' column
print(store2)
```

```
      Store StoreType Assortment  CompetitionDistance  \
0         1         c          a               1270.0
1         2         a          a                570.0
2         3         a          a              14130.0
3         4         c          c                620.0
4         5         a          a              29910.0
...     ...       ...        ...                 ...
1110   1111         a          a               1900.0
1111   1112         c          c               1880.0
1112   1113         a          c               9260.0
1113   1114         a          c                870.0
1114   1115         d          c               5350.0

      CompetitionOpenSinceMonth  CompetitionOpenSinceYear  Promo2  \
0                           9.0                    2008.0       0
1                          11.0                    2007.0       1
2                          12.0                    2006.0       1
3                           9.0                    2009.0       0
4                           4.0                    2015.0       0
...                         ...                       ...     ...
1110                        6.0                    2014.0       1
1111                        4.0                    2006.0       0
1112                        0.0                       0.0       0
1113                        0.0                       0.0       0
1114                        0.0                       0.0       1

      Promo2SinceWeek  Promo2SinceYear  PromoInterval  \
0                 0.0            -31.0            0.0
1                13.0            260.0            1.0
2                14.0            208.0            1.0
3                 0.0            -31.0            0.0
4                 0.0            -31.0            0.0
...               ...              ...            ...
1110             31.0            104.0            1.0
1111              0.0            -31.0            0.0
1112              0.0            -31.0            0.0
1113              0.0            -31.0            0.0
1114             22.0            156.0            3.0

      CompetitionDistanceDecile Type_Assort
0                             4          Ca
1                             3          Aa
2                             9          Aa
3                             3          Cc
4                            10          Aa
...                         ...         ...
1110                          5          Aa
1111                          5          Cc
1112                          9          Ac
1113                          3          Ac
1114                          8          Dc

[1115 rows x 12 columns]
```

In [54]:
```
# Calculate the PromoRun
store2['PromoRun'] = store2['Promo2SinceYear'] - store2['Promo2SinceWeek'] -

# Display the DataFrame with the new 'PromoRun' column
print(store2)
```

```
      Store StoreType Assortment  CompetitionDistance  \
0         1         c          a               1270.0
1         2         a          a                570.0
2         3         a          a              14130.0
3         4         c          c                620.0
4         5         a          a              29910.0
...     ...       ...        ...                  ...
1110   1111         a          a               1900.0
1111   1112         c          c               1880.0
1112   1113         a          c               9260.0
1113   1114         a          c                870.0
1114   1115         d          c               5350.0

      CompetitionOpenSinceMonth  CompetitionOpenSinceYear  Promo2  \
0                           9.0                    2008.0       0
1                          11.0                    2007.0       1
2                          12.0                    2006.0       1
3                           9.0                    2009.0       0
4                           4.0                    2015.0       0
...                         ...                       ...     ...
1110                        6.0                    2014.0       1
1111                        4.0                    2006.0       0
1112                        0.0                       0.0       0
1113                        0.0                       0.0       0
1114                        0.0                       0.0       1

      Promo2SinceWeek  Promo2SinceYear  PromoInterval  \
0                 0.0            -31.0            0.0
1                13.0            260.0            1.0
2                14.0            208.0            1.0
3                 0.0            -31.0            0.0
4                 0.0            -31.0            0.0
...               ...              ...            ...
1110             31.0            104.0            1.0
1111              0.0            -31.0            0.0
1112              0.0            -31.0            0.0
1113              0.0            -31.0            0.0
1114             22.0            156.0            3.0

      CompetitionDistanceDecile Type_Assort  PromoRun
0                             4          Ca       0.0
1                             3          Aa     278.0
2                             9          Aa     225.0
3                             3          Cc       0.0
4                            10          Aa       0.0
...                         ...         ...       ...
1110                          5          Aa     104.0
1111                          5          Cc       0.0
1112                          9          Ac       0.0
1113                          3          Ac       0.0
1114                          8          Dc     165.0

[1115 rows x 13 columns]
```

In [55]:
```
store2['PromoRun'].value_counts()
```

Out[55]:
```
PromoRun
0.0      544
225.0     63
43.0      48
104.0     37
307.0     35
278.0     34
130.0     33
73.0      32
299.0     30
165.0     25
186.0     19
199.0     16
256.0     14
221.0     14
230.0     14
134.0     13
147.0     11
125.0     10
99.0      10
217.0      8
152.0      7
313.0      7
121.0      7
65.0       6
173.0      6
252.0      5
160.0      5
108.0      4
17.0       4
204.0      4
191.0      4
139.0      4
273.0      4
82.0       3
286.0      3
234.0      3
38.0       3
8.0        3
95.0       2
212.0      2
143.0      2
13.0       2
117.0      2
247.0      1
159.0      1
243.0      1
60.0       1
34.0       1
305.0      1
241.0      1
69.0       1
112.0      1
169.0      1
25.0       1
265.0      1
90.0       1
Name: count, dtype: int64
```

In [56]:
```
store2['PromoRun'].value_counts().sort_index(ascending=True)
```

Out[56]:
```
PromoRun
0.0      544
8.0        3
13.0       2
17.0       4
25.0       1
34.0       1
38.0       3
43.0      48
60.0       1
65.0       6
69.0       1
73.0      32
82.0       3
90.0       1
95.0       2
99.0      10
104.0     37
108.0      4
112.0      1
117.0      2
121.0      7
125.0     10
130.0     33
134.0     13
139.0      4
143.0      2
147.0     11
152.0      7
159.0      1
160.0      5
165.0     25
169.0      1
173.0      6
186.0     19
191.0      4
199.0     16
204.0      4
212.0      2
217.0      8
221.0     14
225.0     63
230.0     14
234.0      3
241.0      1
243.0      1
247.0      1
252.0      5
256.0     14
265.0      1
273.0      4
278.0     34
286.0      3
299.0     30
305.0      1
307.0     35
313.0      7
Name: count, dtype: int64
```

In [57]:
```
# Given value counts for PromoRun
promo_run_counts = {
    0.0: 544,
    8.0: 3,
    13.0: 2,
```

```
    17.0: 4,
    25.0: 1,
    34.0: 1,
    38.0: 3,
    43.0: 48,
    60.0: 1,
    65.0: 6,
    69.0: 1,
    73.0: 32,
    82.0: 3,
    90.0: 1,
    95.0: 2,
    99.0: 10,
    104.0: 37,
    108.0: 4,
    112.0: 1,
    117.0: 2,
    121.0: 7,
    125.0: 10,
    130.0: 33,
    134.0: 13
}

# Step 1: Create a DataFrame from the value counts
promo_run_df = pd.DataFrame(list(promo_run_counts.items()), columns=['PromoR

# Step 2: Calculate the new column (135 - PromoRun) * Count
promo_run_df['NewColumn'] = (135 - promo_run_df['PromoRun']) * promo_run_df

# Print the resulting DataFrame
print(promo_run_df)
```

```
    PromoRun  Count  NewColumn
0        0.0    544    73440.0
1        8.0      3      381.0
2       13.0      2      244.0
3       17.0      4      472.0
4       25.0      1      110.0
5       34.0      1      101.0
6       38.0      3      291.0
7       43.0     48     4416.0
8       60.0      1       75.0
9       65.0      6      420.0
10      69.0      1       66.0
11      73.0     32     1984.0
12      82.0      3      159.0
13      90.0      1       45.0
14      95.0      2       80.0
15      99.0     10      360.0
16     104.0     37     1147.0
17     108.0      4      108.0
18     112.0      1       23.0
19     117.0      2       36.0
20     121.0      7       98.0
21     125.0     10      100.0
22     130.0     33      165.0
23     134.0     13       13.0
```

In [58]:
```
# Given value counts for PromoRun
promo_run_counts = {
    0.0: 544,
    8.0: 3,
    13.0: 2,
    17.0: 4,
```

```
    25.0: 1,
    34.0: 1,
    38.0: 3,
    43.0: 48,
    60.0: 1,
    65.0: 6,
    69.0: 1,
    73.0: 32,
    82.0: 3,
    90.0: 1,
    95.0: 2,
    99.0: 10,
    104.0: 37,
    108.0: 4,
    112.0: 1,
    117.0: 2,
    121.0: 7,
    125.0: 10,
    130.0: 33,
    134.0: 13
}

# Step 1: Create a DataFrame from the value counts
promo_run_df = pd.DataFrame(list(promo_run_counts.items()), columns=['PromoR

# Step 2: Calculate the new column (135 - PromoRun) * Count
promo_run_df['NewColumn'] = (135 - promo_run_df['PromoRun']) * promo_run_df

# Step 3: Calculate the sum of the new column
sum_new_column = promo_run_df['NewColumn'].sum()

# Step 4: Calculate the sum of the Count column
sum_count = promo_run_df['Count'].sum()
# Print the resulting DataFrame and the sum of the new column
print(promo_run_df)
print(f"Sum of NewColumn: {sum_new_column}")
print(f"Sum of Count: {sum_count}")
```

```
    PromoRun  Count  NewColumn
0        0.0    544    73440.0
1        8.0      3      381.0
2       13.0      2      244.0
3       17.0      4      472.0
4       25.0      1      110.0
5       34.0      1      101.0
6       38.0      3      291.0
7       43.0     48     4416.0
8       60.0      1       75.0
9       65.0      6      420.0
10      69.0      1       66.0
11      73.0     32     1984.0
12      82.0      3      159.0
13      90.0      1       45.0
14      95.0      2       80.0
15      99.0     10      360.0
16     104.0     37     1147.0
17     108.0      4      108.0
18     112.0      1       23.0
19     117.0      2       36.0
20     121.0      7       98.0
21     125.0     10      100.0
22     130.0     33      165.0
23     134.0     13       13.0
Sum of NewColumn: 84334.0
Sum of Count: 769
```

In [59]:
```
84334.0*7
```

Out[59]:
```
590338.0
```

In [60]:
```
(769-544)* 135 *7
```

Out[60]:
```
212625
```

In [61]:
```
73440*7
```

Out[61]:
```
514080
```

In [62]:
```
# Step describe
store2['PromoRun'].describe()
```

Out[62]:
```
count    1115.000000
mean       90.042152
std       107.039025
min         0.000000
25%         0.000000
50%        38.000000
75%       186.000000
max       313.000000
Name: PromoRun, dtype: float64
```

In [63]:
```
# Step info
store2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1115 entries, 0 to 1114
Data columns (total 13 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Store                      1115 non-null   int64
 1   StoreType                  1115 non-null   object
 2   Assortment                 1115 non-null   object
 3   CompetitionDistance        1115 non-null   float64
 4   CompetitionOpenSinceMonth  1115 non-null   float64
 5   CompetitionOpenSinceYear   1115 non-null   float64
 6   Promo2                     1115 non-null   int64
 7   Promo2SinceWeek            1115 non-null   float64
 8   Promo2SinceYear            1115 non-null   float64
 9   PromoInterval              1115 non-null   float64
 10  CompetitionDistanceDecile  1115 non-null   int64
 11  Type_Assort                1115 non-null   object
 12  PromoRun                   1115 non-null   float64
dtypes: float64(7), int64(3), object(3)
memory usage: 113.4+ KB
```

In [64]:
```
# Specify the columns to drop
columns_to_drop = ['StoreType', 'Assortment', 'CompetitionOpenSinceMonth',
                   'CompetitionOpenSinceYear', 'Promo2SinceWeek', 'Promo2Si

# Drop the columns
store3 = store2.drop(columns=columns_to_drop)

# Display the DataFrame after dropping columns
store3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1115 entries, 0 to 1114
Data columns (total 7 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Store                      1115 non-null   int64
 1   CompetitionDistance        1115 non-null   float64
 2   Promo2                     1115 non-null   int64
 3   PromoInterval              1115 non-null   float64
 4   CompetitionDistanceDecile  1115 non-null   int64
 5   Type_Assort                1115 non-null   object
 6   PromoRun                   1115 non-null   float64
dtypes: float64(3), int64(3), object(1)
memory usage: 61.1+ KB
```

In [65]:
```
# Read in the CSV file and name df
train= pd.read_csv("train(2).csv")
```

In [66]:
```
# Begin review of df
train.shape
```

Out[66]:
```
(1017209, 9)
```

In [67]:
```
# Df has 1017209 rows and 9 columns(cols)
# Continue review of df
# Display df's head rows
train.head(1116)
```

Out[67]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	Schoo
0	1	5	31/07/2015	5263	555	1	1	0	
1	2	5	31/07/2015	6064	625	1	1	0	
2	3	5	31/07/2015	8314	821	1	1	0	
3	4	5	31/07/2015	13995	1498	1	1	0	
4	5	5	31/07/2015	4822	559	1	1	0	
...	...	...	...	...	...	...	...	...	
1111	1112	5	31/07/2015	9626	767	1	1	0	
1112	1113	5	31/07/2015	7289	720	1	1	0	
1113	1114	5	31/07/2015	27508	3745	1	1	0	
1114	1115	5	31/07/2015	8680	538	1	1	0	
1115	1	4	30/07/2015	5020	546	1	1	0	

1116 rows × 9 columns

In [68]:
```
# Display df's first 5 rows
train.tail(130)
```

Out[68]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	Sch
1017079	985	2	01/01/2013	0	0	0	0	a	
1017080	986	2	01/01/2013	0	0	0	0	a	
1017081	987	2	01/01/2013	0	0	0	0	a	
1017082	989	2	01/01/2013	0	0	0	0	a	
1017083	990	2	01/01/2013	0	0	0	0	a	
...	...	...	...	...	...	...	...	...	
1017204	1111	2	01/01/2013	0	0	0	0	a	
1017205	1112	2	01/01/2013	0	0	0	0	a	
1017206	1113	2	01/01/2013	0	0	0	0	a	
1017207	1114	2	01/01/2013	0	0	0	0	a	
1017208	1115	2	01/01/2013	0	0	0	0	a	

130 rows × 9 columns

In [69]:
```
# Step info
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1017209 entries, 0 to 1017208
Data columns (total 9 columns):
 #   Column         Non-Null Count    Dtype 
---  ------         --------------    ----- 
 0   Store          1017209 non-null  int64 
 1   DayOfWeek      1017209 non-null  int64 
 2   Date           1017209 non-null  object
 3   Sales          1017209 non-null  int64 
 4   Customers      1017209 non-null  int64 
 5   Open           1017209 non-null  int64 
 6   Promo          1017209 non-null  int64 
 7   StateHoliday   1017209 non-null  object
 8   SchoolHoliday  1017209 non-null  int64 
dtypes: int64(7), object(2)
memory usage: 69.8+ MB
```

In [70]:
```
# Step unique values
train.nunique()
```

Out[70]:
```
Store             1115
DayOfWeek            7
Date               942
Sales            21734
Customers         4086
Open                 2
Promo                2
StateHoliday         5
SchoolHoliday        2
dtype: int64
```

In [71]:
```
# Step count
train.count()
```

Out[71]:
```
Store            1017209
DayOfWeek        1017209
Date             1017209
Sales            1017209
Customers        1017209
Open             1017209
Promo            1017209
StateHoliday     1017209
SchoolHoliday    1017209
dtype: int64
```

In [72]:
```
# Step value count
train["Open"].value_counts()
```

Out[72]:
```
Open
1    844392
0    172817
Name: count, dtype: int64
```

In [73]:
```
# Step value counts
train["StateHoliday"].value_counts()
```

Out[73]:
```
StateHoliday
0    855087
0    131072
a     20260
b      6690
c      4100
Name: count, dtype: int64
```

In [74]:
```
# Replace 0 and '0' with 'd' in the StateHoliday column of df2
train['StateHoliday'].replace({0: 'd', '0': 'd'}, inplace=True)
```

In [75]:
```
# Step value counts
train["StateHoliday"].value_counts()
```

Out[75]:
```
StateHoliday
d    986159
a     20260
b      6690
c      4100
Name: count, dtype: int64
```

In [76]:
```
# Filter out rows where 'StateHoliday' is 'a'
state_holiday_a = train[train['StateHoliday'] == 'a']

# Get unique dates for 'StateHoliday' = 'a'
unique_dates_a = state_holiday_a['Date'].unique()

# Count occurrences of each unique date
date_counts_a = state_holiday_a['Date'].value_counts()

print("Unique dates for 'StateHoliday' = 'a':")
print(unique_dates_a)

print("Count of each unique date for 'StateHoliday' = 'a':")
print(date_counts_a)
```

```
Unique dates for 'StateHoliday' = 'a':
['04/06/2015' '25/05/2015' '14/05/2015' '01/05/2015' '06/01/2015'
 '01/01/2015' '19/11/2014' '01/11/2014' '31/10/2014' '03/10/2014'
 '19/06/2014' '09/06/2014' '29/05/2014' '01/05/2014' '06/01/2014'
 '01/01/2014' '20/11/2013' '01/11/2013' '31/10/2013' '03/10/2013'
 '15/08/2013' '30/05/2013' '20/05/2013' '09/05/2013' '01/05/2013'
 '06/01/2013' '01/01/2013']
Count of each unique date for 'StateHoliday' = 'a':
Date
01/05/2014    1115
09/06/2014    1115
14/05/2015    1115
01/05/2015    1115
01/05/2013    1115
09/05/2013    1115
20/05/2013    1115
03/10/2013    1115
01/01/2014    1115
25/05/2015    1115
29/05/2014    1115
01/01/2013    1114
01/01/2015    1079
03/10/2014     935
19/06/2014     766
30/05/2013     766
04/06/2015     766
01/11/2013     579
01/11/2014     399
06/01/2014     309
06/01/2015     309
06/01/2013     309
15/08/2013     180
31/10/2014     167
31/10/2013     167
20/11/2013      75
19/11/2014      75
Name: count, dtype: int64
```

In [77]:

```python
# Filter out rows where 'StateHoliday' is 'b'
state_holiday_b = train[train['StateHoliday'] == 'b']

# Get unique dates for 'StateHoliday' = 'a'
unique_dates_b = state_holiday_b['Date'].unique()

# Count occurrences of each unique date
date_counts_b = state_holiday_b['Date'].value_counts()

print("Unique dates for 'StateHoliday' = 'b':")
print(unique_dates_b)

print("Count of each unique date for 'StateHoliday' = 'b':")
print(date_counts_b)
```

```
Unique dates for 'StateHoliday' = 'b':
['06/04/2015' '03/04/2015' '21/04/2014' '18/04/2014' '01/04/2013'
 '29/03/2013']
Count of each unique date for 'StateHoliday' = 'b':
Date
06/04/2015    1115
03/04/2015    1115
21/04/2014    1115
18/04/2014    1115
01/04/2013    1115
29/03/2013    1115
Name: count, dtype: int64
```

In [78]:
```
# Filter out rows where 'StateHoliday' is 'a'
state_holiday_c = train[train['StateHoliday'] == 'c']

# Get unique dates for 'StateHoliday' = 'a'
unique_dates_c = state_holiday_c['Date'].unique()

# Count occurrences of each unique date
date_counts_c = state_holiday_c['Date'].value_counts()

print("Unique dates for 'StateHoliday' = 'c':")
print(unique_dates_c)

print("Count of each unique date for 'StateHoliday' = 'c':")
print(date_counts_c)
```

```
Unique dates for 'StateHoliday' = 'c':
['26/12/2014' '25/12/2014' '26/12/2013' '25/12/2013']
Count of each unique date for 'StateHoliday' = 'c':
Date
26/12/2013    1115
25/12/2013    1115
26/12/2014     935
25/12/2014     935
Name: count, dtype: int64
```

In [79]:
```
# Step value count
train["DayOfWeek"].value_counts()
```

Out[79]:
```
DayOfWeek
5    145845
4    145845
3    145665
2    145664
1    144730
7    144730
6    144730
Name: count, dtype: int64
```

In [80]:
```
# Step value count
train["Date"].value_counts()
```

Out[80]:
```
Date
31/07/2015    1115
06/11/2013    1115
18/11/2013    1115
17/11/2013    1115
16/11/2013    1115
              ...
28/10/2014     935
27/10/2014     935
26/10/2014     935
25/10/2014     935
08/12/2014     935
Name: count, Length: 942, dtype: int64
```

In [81]:
```
# Step value count
train["Promo"].value_counts()
```

Out[81]:
```
Promo
0    629129
1    388080
Name: count, dtype: int64
```

In [82]:
```
# Step info
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1017209 entries, 0 to 1017208
Data columns (total 9 columns):
 #   Column         Non-Null Count    Dtype 
---  ------         --------------    ----- 
 0   Store          1017209 non-null  int64 
 1   DayOfWeek      1017209 non-null  int64 
 2   Date           1017209 non-null  object
 3   Sales          1017209 non-null  int64 
 4   Customers      1017209 non-null  int64 
 5   Open           1017209 non-null  int64 
 6   Promo          1017209 non-null  int64 
 7   StateHoliday   1017209 non-null  object
 8   SchoolHoliday  1017209 non-null  int64 
dtypes: int64(7), object(2)
memory usage: 69.8+ MB
```

In [83]:
```
# Step value count
date_counts = train["Date"].value_counts()

# Filter stores with counts not equal to 1115
dates_not_1115 = date_counts[date_counts != 1115]

print("Dates with counts not equal to 1115:")
print(dates_not_1115)
```

```
Dates with counts not equal to 1115:
Date
01/01/2013    1114
31/12/2014     935
26/08/2014     935
04/09/2014     935
03/09/2014     935
              ...
28/10/2014     935
27/10/2014     935
26/10/2014     935
25/10/2014     935
08/12/2014     935
Name: count, Length: 185, dtype: int64
```

In [84]:
```
# Step value count
date_counts = train["Date"].value_counts()

# Filter dates with counts not equal to 1115
dates_not_1115 = date_counts[date_counts != 1115]

print("Dates with counts not equal to 1115:")
print(dates_not_1115)

# If you want to print just the dates, convert the index to a list
dates_not_1115_list = dates_not_1115.index.tolist()
print("Dates with counts not equal to 1115:")
print(dates_not_1115_list)
```

```
Dates with counts not equal to 1115:
Date
01/01/2013    1114
31/12/2014     935
26/08/2014     935
04/09/2014     935
03/09/2014     935
               ...
28/10/2014     935
27/10/2014     935
26/10/2014     935
25/10/2014     935
08/12/2014     935
Name: count, Length: 185, dtype: int64
Dates with counts not equal to 1115:
['01/01/2013', '31/12/2014', '26/08/2014', '04/09/2014', '03/09/2014', '02/
09/2014', '01/09/2014', '31/08/2014', '30/08/2014', '29/08/2014', '28/08/20
14', '27/08/2014', '25/08/2014', '30/12/2014', '24/08/2014', '23/08/2014',
'22/08/2014', '21/08/2014', '20/08/2014', '19/08/2014', '18/08/2014', '17/0
8/2014', '16/08/2014', '05/09/2014', '06/09/2014', '07/09/2014', '08/09/201
4', '27/09/2014', '26/09/2014', '25/09/2014', '24/09/2014', '23/09/2014',
'22/09/2014', '21/09/2014', '20/09/2014', '19/09/2014', '18/09/2014', '17/0
9/2014', '16/09/2014', '15/09/2014', '14/09/2014', '13/09/2014', '12/09/201
4', '11/09/2014', '10/09/2014', '09/09/2014', '15/08/2014', '14/08/2014',
'13/08/2014', '21/07/2014', '19/07/2014', '18/07/2014', '17/07/2014', '16/0
7/2014', '15/07/2014', '14/07/2014', '13/07/2014', '12/07/2014', '11/07/201
4', '10/07/2014', '09/07/2014', '08/07/2014', '07/07/2014', '06/07/2014',
'05/07/2014', '04/07/2014', '03/07/2014', '02/07/2014', '01/07/2014', '20/0
7/2014', '22/07/2014', '12/08/2014', '23/07/2014', '11/08/2014', '10/08/201
4', '09/08/2014', '08/08/2014', '07/08/2014', '06/08/2014', '05/08/2014',
'04/08/2014', '03/08/2014', '02/08/2014', '01/08/2014', '31/07/2014', '30/0
7/2014', '29/07/2014', '28/07/2014', '27/07/2014', '26/07/2014', '25/07/201
4', '24/07/2014', '28/09/2014', '29/09/2014', '30/09/2014', '07/12/2014',
'05/12/2014', '04/12/2014', '03/12/2014', '02/12/2014', '01/12/2014', '30/1
1/2014', '29/11/2014', '28/11/2014', '27/11/2014', '26/11/2014', '25/11/201
4', '24/11/2014', '23/11/2014', '22/11/2014', '21/11/2014', '20/11/2014',
'19/11/2014', '18/11/2014', '17/11/2014', '06/12/2014', '09/12/2014', '15/1
1/2014', '10/12/2014', '29/12/2014', '28/12/2014', '27/12/2014', '26/12/201
4', '25/12/2014', '24/12/2014', '23/12/2014', '22/12/2014', '21/12/2014',
'20/12/2014', '19/12/2014', '18/12/2014', '17/12/2014', '16/12/2014', '15/1
2/2014', '14/12/2014', '13/12/2014', '12/12/2014', '11/12/2014', '16/11/201
4', '14/11/2014', '01/10/2014', '22/10/2014', '20/10/2014', '19/10/2014',
'18/10/2014', '17/10/2014', '16/10/2014', '15/10/2014', '14/10/2014', '13/1
0/2014', '12/10/2014', '11/10/2014', '10/10/2014', '09/10/2014', '08/10/201
4', '07/10/2014', '06/10/2014', '05/10/2014', '04/10/2014', '03/10/2014',
'02/10/2014', '21/10/2014', '23/10/2014', '13/11/2014', '24/10/2014', '12/1
1/2014', '11/11/2014', '10/11/2014', '09/11/2014', '08/11/2014', '07/11/201
4', '06/11/2014', '05/11/2014', '04/11/2014', '03/11/2014', '02/11/2014',
'01/11/2014', '31/10/2014', '30/10/2014', '29/10/2014', '28/10/2014', '27/1
0/2014', '26/10/2014', '25/10/2014', '08/12/2014']
```

In [85]:
```
# Step value count
train["Store"].value_counts()
```

Out[85]:
```
Store
1      942
726    942
708    942
709    942
713    942
      ...
159    758
637    758
636    758
633    758
155    758
Name: count, Length: 1115, dtype: int64
```

In [86]:
```
# Step value count
store_counts = train["Store"].value_counts()

# Filter stores with counts not equal to 942
stores_not_942 = store_counts[store_counts != 942]

print("Stores with counts not equal to 942:")
print(stores_not_942)
```

```
Stores with counts not equal to 942:
Store
988     941
539     758
46      758
89      758
1107    758
       ...
159     758
637     758
636     758
633     758
155     758
Name: count, Length: 181, dtype: int64
```

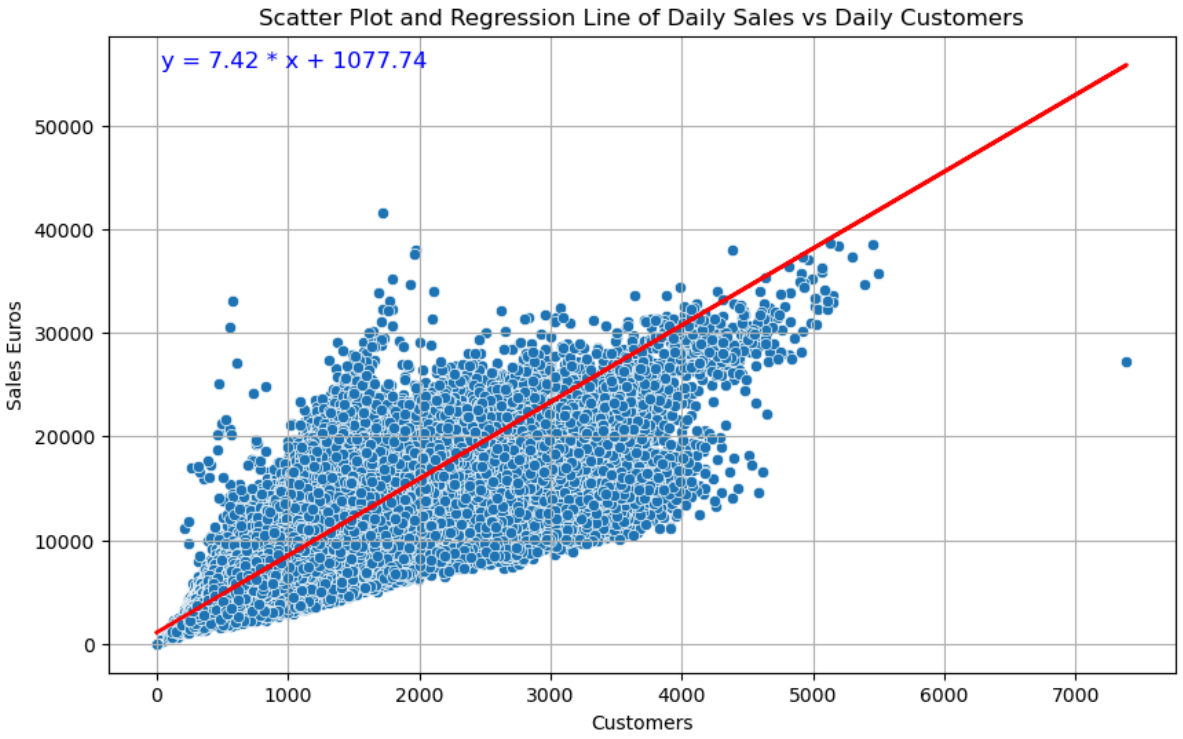
In [87]:
```
# Step describe
train.describe()
```

Out[87]:

	Store	DayOfWeek	Sales	Customers	Open	Pr
count	1.017209e+06	1.017209e+06	1.017209e+06	1.017209e+06	1.017209e+06	1.017209e
mean	5.584297e+02	3.998341e+00	5.773819e+03	6.331459e+02	8.301067e-01	3.815145
std	3.219087e+02	1.997391e+00	3.849926e+03	4.644117e+02	3.755392e-01	4.857586
min	1.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e
25%	2.800000e+02	2.000000e+00	3.727000e+03	4.050000e+02	1.000000e+00	0.000000e
50%	5.580000e+02	4.000000e+00	5.744000e+03	6.090000e+02	1.000000e+00	0.000000e
75%	8.380000e+02	6.000000e+00	7.856000e+03	8.370000e+02	1.000000e+00	1.000000e
max	1.115000e+03	7.000000e+00	4.155100e+04	7.388000e+03	1.000000e+00	1.000000e

In [88]:
```
import statsmodels.api as sm

# Create scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Customers', y='Sales', data=train)

# Fit linear regression model
```

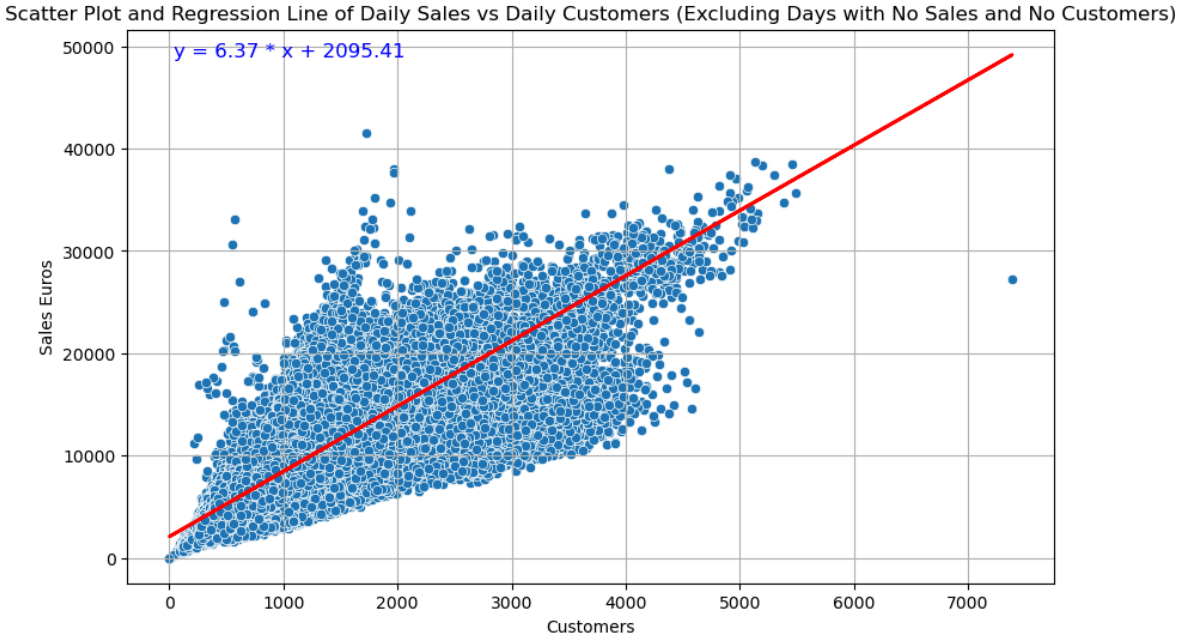
```
X =train['Customers']
y = train['Sales']
X = sm.add_constant(X)  # Add constant for intercept
model = sm.OLS(y, X).fit()

# Plot regression line
plt.plot(X['Customers'], model.predict(X), color='red', linewidth=2)

# Annotate regression equation
slope = model.params['Customers']
intercept = model.params['const']
equation = f'y = {slope:.2f} * x + {intercept:.2f}'
plt.annotate(equation, xy=(0.05, 0.95), xycoords='axes fraction', fontsize=1

# Set plot labels and title
plt.xlabel('Customers')
plt.ylabel('Sales Euros')
plt.title('Scatter Plot and Regression Line of Daily Sales vs Daily Custome

# Show plot
plt.grid(True)
plt.show()
```



In [89]:

```
import statsmodels.api as sm

# Filter the DataFrame to exclude days with no sales and no customers
filtered_data = train[train['Open'] == 1]

# Create scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Customers', y='Sales', data=filtered_data)

# Fit linear regression model
X = filtered_data['Customers']
y = filtered_data['Sales']
X = sm.add_constant(X)  # Add constant for intercept
model = sm.OLS(y, X).fit()

# Plot regression line
plt.plot(X['Customers'], model.predict(X), color='red', linewidth=2)
```

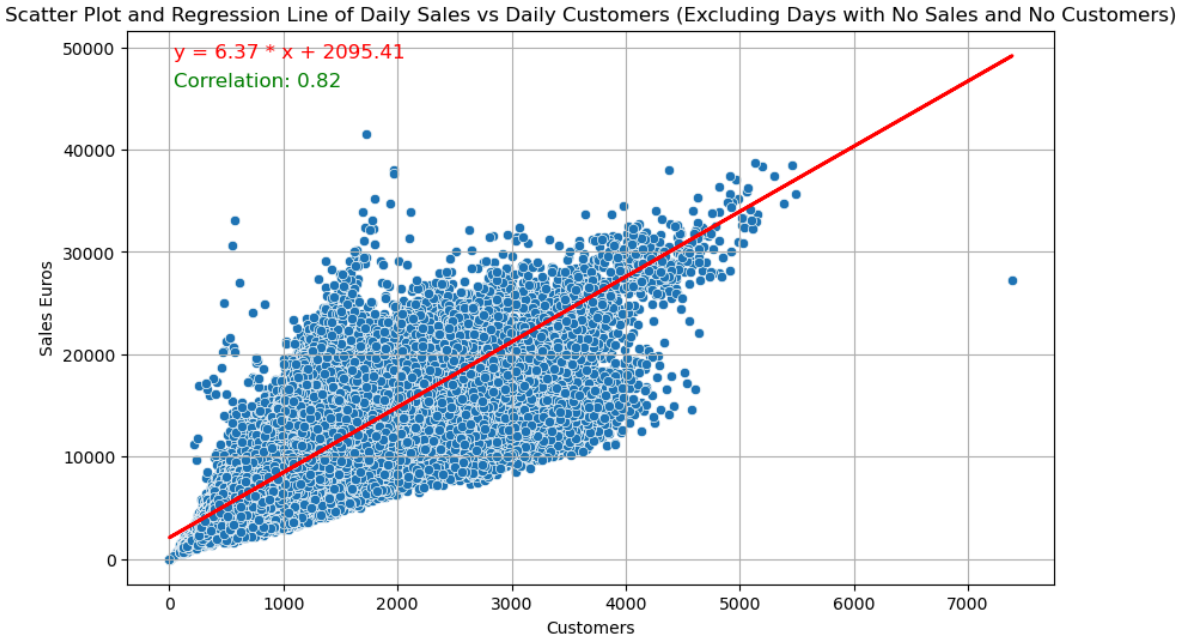
```
# Annotate regression equation
slope = model.params['Customers']
intercept = model.params['const']
equation = f'y = {slope:.2f} * x + {intercept:.2f}'
plt.annotate(equation, xy=(0.05, 0.95), xycoords='axes fraction', fontsize=1

# Set plot labels and title
plt.xlabel('Customers')
plt.ylabel('Sales Euros')
plt.title('Scatter Plot and Regression Line of Daily Sales vs Daily Custome

# Show plot
plt.grid(True)
plt.show()
```



In [90]:

```
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import pearsonr

# Filter the DataFrame to exclude days with no sales and no customers
filtered_data = train[train['Open'] == 1]

# Create scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Customers', y='Sales', data=filtered_data)

# Fit linear regression model
X = filtered_data['Customers']
y = filtered_data['Sales']
X = sm.add_constant(X)  # Add constant for intercept
model = sm.OLS(y, X).fit()

# Plot regression line
plt.plot(X['Customers'], model.predict(X), color='red', linewidth=2)

# Annotate regression equation
slope = model.params['Customers']
intercept = model.params['const']
equation = f'y = {slope:.2f} * x + {intercept:.2f}'
plt.annotate(equation, xy=(0.05, 0.95), xycoords='axes fraction', fontsize=1
```

```
# Calculate and annotate correlation coefficient
correlation, _ = pearsonr(filtered_data['Customers'], filtered_data['Sales']
correlation_text = f'Correlation: {correlation:.2f}'
plt.annotate(correlation_text, xy=(0.05, 0.90), xycoords='axes fraction', fo

# Set plot labels and title
plt.xlabel('Customers')
plt.ylabel('Sales Euros')
plt.title('Scatter Plot and Regression Line of Daily Sales vs Daily Customer

# Show plot
plt.grid(True)
plt.show()
```



In [91]:
```
# Find the index of the row with the maximum number of customers
max_customers_index = train['Customers'].idxmax()

# Retrieve the row with the maximum number of customers
row_with_max_customers = train.loc[max_customers_index]

print("Row with the maximum number of customers:")
print(row_with_max_customers)
```

```
Row with the maximum number of customers:
Store                 817
DayOfWeek               2
Date           22/01/2013
Sales               27190
Customers            7388
Open                    1
Promo                   1
StateHoliday            d
SchoolHoliday           0
Name: 993496, dtype: object
```

In [92]:
```
27190/7388
```

Out[92]:
```
3.680292365998917
```

In [93]:
```
26190/3788
```

Out[93]:
```
6.913938753959873
```

In [94]:
```
6.37*3788+2095
```

Out[94]:
```
26224.56
```

In [95]:
```
# Correct the Customers number
train.at[max_customers_index, 'Customers'] = 3788

# Print the corrected row
print("Row with the corrected number of customers:")
print(train.loc[max_customers_index])
```

```
Row with the corrected number of customers:
Store                 817
DayOfWeek               2
Date           22/01/2013
Sales               27190
Customers            3788
Open                    1
Promo                   1
StateHoliday            d
SchoolHoliday           0
Name: 993496, dtype: object
```

In [96]:
```
# Read in the CSV file and name df
test= pd.read_csv("test(1).csv")
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41088 entries, 0 to 41087
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Store          41088 non-null  int64  
 1   DayOfWeek      41088 non-null  int64  
 2   Date           41088 non-null  object 
 3   Sales          0 non-null      float64
 4   Customers      0 non-null      float64
 5   Open           41077 non-null  float64
 6   Promo          41088 non-null  int64  
 7   StateHoliday   41088 non-null  object 
 8   SchoolHoliday  41088 non-null  int64  
dtypes: float64(3), int64(4), object(2)
memory usage: 2.8+ MB
```

In [97]:
```
# Step shape
test.shape
```

Out[97]:
```
(41088, 9)
```

In [98]:
```
# Step head
test.head()
```

Out[98]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHol
0	1	4	17/09/2015	NaN	NaN	1.0	1	0	
1	3	4	17/09/2015	NaN	NaN	1.0	1	0	
2	7	4	17/09/2015	NaN	NaN	1.0	1	0	
3	8	4	17/09/2015	NaN	NaN	1.0	1	0	
4	9	4	17/09/2015	NaN	NaN	1.0	1	0	

In [99]:
```
# Step tail
test.tail()
```

Out[99]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	Scho
41083	1111	6	01/08/2015	NaN	NaN	1.0	0	0	
41084	1112	6	01/08/2015	NaN	NaN	1.0	0	0	
41085	1113	6	01/08/2015	NaN	NaN	1.0	0	0	
41086	1114	6	01/08/2015	NaN	NaN	1.0	0	0	
41087	1115	6	01/08/2015	NaN	NaN	1.0	0	0	

In [100…
```
# Step info
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41088 entries, 0 to 41087
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Store          41088 non-null  int64 
 1   DayOfWeek      41088 non-null  int64 
 2   Date           41088 non-null  object
 3   Sales          0 non-null      float64
 4   Customers      0 non-null      float64
 5   Open           41077 non-null  float64
 6   Promo          41088 non-null  int64 
 7   StateHoliday   41088 non-null  object
 8   SchoolHoliday  41088 non-null  int64 
dtypes: float64(3), int64(4), object(2)
memory usage: 2.8+ MB
```

In [101…
```
# Step values unique
test.nunique()
```

Out[101]:
```
Store            856
DayOfWeek          7
Date              48
Sales              0
Customers          0
Open               2
Promo              2
StateHoliday       2
SchoolHoliday      2
dtype: int64
```

In [102…
```
856*48
```

Out[102]:
```
41088
```

In [103…
```
# Step  count
test.count()
```

Out[103]:
```
Store            41088
DayOfWeek        41088
Date             41088
Sales                0
Customers            0
Open             41077
Promo            41088
StateHoliday     41088
SchoolHoliday    41088
dtype: int64
```

In [104…
```
# Step value count
test['Open'].value_counts()
```

Out[104]:
```
Open
1.0    35093
0.0     5984
Name: count, dtype: int64
```

In [105…
```
# Step value count
test['Promo'].value_counts()
```

Out[105]:
```
Promo
0    24824
1    16264
Name: count, dtype: int64
```

In [106…
```
# Step value count
test['StateHoliday'].value_counts()
```

Out[106]:
```
StateHoliday
0    40908
a      180
Name: count, dtype: int64
```

In [107…
```
# Step value count
test['SchoolHoliday'].value_counts()
```

Out[107]:
```
SchoolHoliday
0    22866
1    18222
Name: count, dtype: int64
```

In [108…
```
# Replace all occurrences of 0 with 'd' in the 'StateHoliday' column
test['StateHoliday'] = test['StateHoliday'].replace(0, 'd')
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41088 entries, 0 to 41087
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Store          41088 non-null  int64
 1   DayOfWeek      41088 non-null  int64
 2   Date           41088 non-null  object
 3   Sales          0 non-null      float64
 4   Customers      0 non-null      float64
 5   Open           41077 non-null  float64
 6   Promo          41088 non-null  int64
 7   StateHoliday   41088 non-null  object
 8   SchoolHoliday  41088 non-null  int64
dtypes: float64(3), int64(4), object(2)
memory usage: 2.8+ MB
```

In [109…
```
# Step value count
test['StateHoliday'].value_counts()
```

Out[109]:
```
StateHoliday
0    40908
a      180
Name: count, dtype: int64
```

In [110…
```
# Convert the 'StateHoliday' column to strings
test['StateHoliday'] = test['StateHoliday'].astype(str)

# Replace all occurrences of '0' with 'd' in the 'StateHoliday' column
test['StateHoliday'] = test['StateHoliday'].replace('0', 'd')
```

In [111…
```
# Step value count
test['StateHoliday'].value_counts()
```

Out[111]:
```
StateHoliday
d    40908
a      180
Name: count, dtype: int64
```

In [112…
```
# Step describe
test.describe()
```

Out[112]:

	Store	DayOfWeek	Sales	Customers	Open	Promo	Schoo
count	41088.000000	41088.000000	0.0	0.0	41077.000000	41088.000000	4108
mean	555.899533	3.979167	NaN	NaN	0.854322	0.395833	
std	320.274496	2.015481	NaN	NaN	0.352787	0.489035	
min	1.000000	1.000000	NaN	NaN	0.000000	0.000000	
25%	279.750000	2.000000	NaN	NaN	1.000000	0.000000	
50%	553.500000	4.000000	NaN	NaN	1.000000	0.000000	
75%	832.250000	6.000000	NaN	NaN	1.000000	1.000000	
max	1115.000000	7.000000	NaN	NaN	1.000000	1.000000	

In [113…
```
# Select rows where 'Open' is NaN
rows_with_nan_open = test[test['Open'].isna()]

# Display the selected rows
print(rows_with_nan_open)
```

```
       Store  DayOfWeek        Date  Sales  Customers  Open  Promo  \
479      622          4  17/09/2015    NaN        NaN   NaN      1
1335     622          3  16/09/2015    NaN        NaN   NaN      1
2191     622          2  15/09/2015    NaN        NaN   NaN      1
3047     622          1  14/09/2015    NaN        NaN   NaN      1
4759     622          6  12/09/2015    NaN        NaN   NaN      0
5615     622          5  11/09/2015    NaN        NaN   NaN      0
6471     622          4  10/09/2015    NaN        NaN   NaN      0
7327     622          3  09/09/2015    NaN        NaN   NaN      0
8183     622          2  08/09/2015    NaN        NaN   NaN      0
9039     622          1  07/09/2015    NaN        NaN   NaN      0
10751    622          6  05/09/2015    NaN        NaN   NaN      0

      StateHoliday  SchoolHoliday
479              d              0
1335             d              0
2191             d              0
3047             d              0
4759             d              0
5615             d              0
6471             d              0
7327             d              0
8183             d              0
9039             d              0
10751            d              0
```

In [114…

```
# Replace NaN values in 'Open' column with 1
test['Open'].fillna(1, inplace=True)

# Verify the changes
print(test)
```

```
       Store  DayOfWeek        Date  Sales  Customers  Open  Promo  \
0          1          4  17/09/2015    NaN        NaN   1.0      1
1          3          4  17/09/2015    NaN        NaN   1.0      1
2          7          4  17/09/2015    NaN        NaN   1.0      1
3          8          4  17/09/2015    NaN        NaN   1.0      1
4          9          4  17/09/2015    NaN        NaN   1.0      1
...      ...        ...         ...    ...        ...   ...    ...
41083   1111          6  01/08/2015    NaN        NaN   1.0      0
41084   1112          6  01/08/2015    NaN        NaN   1.0      0
41085   1113          6  01/08/2015    NaN        NaN   1.0      0
41086   1114          6  01/08/2015    NaN        NaN   1.0      0
41087   1115          6  01/08/2015    NaN        NaN   1.0      0

      StateHoliday  SchoolHoliday
0                d              0
1                d              0
2                d              0
3                d              0
4                d              0
...            ...            ...
41083            d              0
41084            d              0
41085            d              0
41086            d              0
41087            d              1

[41088 rows x 9 columns]
```

In [115…

```
# Step info
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41088 entries, 0 to 41087
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Store          41088 non-null  int64  
 1   DayOfWeek      41088 non-null  int64  
 2   Date           41088 non-null  object 
 3   Sales          0 non-null      float64
 4   Customers      0 non-null      float64
 5   Open           41088 non-null  float64
 6   Promo          41088 non-null  int64  
 7   StateHoliday   41088 non-null  object 
 8   SchoolHoliday  41088 non-null  int64  
dtypes: float64(3), int64(4), object(2)
memory usage: 2.8+ MB
```

In [116…

```python
import pandas as pd


# Concatenate the two DataFrames
agg_df = pd.concat([test, train])

# Reset the index
agg_df.reset_index(drop=True, inplace=True)

# Verify the aggregated DataFrame
print(agg_df)
```

```
         Store  DayOfWeek        Date  Sales  Customers  Open  Promo  \
0            1          4  17/09/2015    NaN        NaN   1.0      1   
1            3          4  17/09/2015    NaN        NaN   1.0      1   
2            7          4  17/09/2015    NaN        NaN   1.0      1   
3            8          4  17/09/2015    NaN        NaN   1.0      1   
4            9          4  17/09/2015    NaN        NaN   1.0      1   
...        ...        ...         ...    ...        ...   ...    ...   
1058292   1111          2  01/01/2013    0.0        0.0   0.0      0   
1058293   1112          2  01/01/2013    0.0        0.0   0.0      0   
1058294   1113          2  01/01/2013    0.0        0.0   0.0      0   
1058295   1114          2  01/01/2013    0.0        0.0   0.0      0   
1058296   1115          2  01/01/2013    0.0        0.0   0.0      0   

        StateHoliday  SchoolHoliday  
0                  d              0  
1                  d              0  
2                  d              0  
3                  d              0  
4                  d              0  
...              ...            ...  
1058292            a              1  
1058293            a              1  
1058294            a              1  
1058295            a              1  
1058296            a              1  

[1058297 rows x 9 columns]
```

In [117…

```python
# Step info
agg_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1058297 entries, 0 to 1058296
Data columns (total 9 columns):
 #   Column         Non-Null Count    Dtype
---  ------         --------------    -----
 0   Store          1058297 non-null  int64
 1   DayOfWeek      1058297 non-null  int64
 2   Date           1058297 non-null  object
 3   Sales          1017209 non-null  float64
 4   Customers      1017209 non-null  float64
 5   Open           1058297 non-null  float64
 6   Promo          1058297 non-null  int64
 7   StateHoliday   1058297 non-null  object
 8   SchoolHoliday  1058297 non-null  int64
dtypes: float64(3), int64(4), object(2)
memory usage: 72.7+ MB
```

In [118…
```
# Step head
agg_df.head()
```

Out[118]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHo
0	1	4	17/09/2015	NaN	NaN	1.0	1	d	
1	3	4	17/09/2015	NaN	NaN	1.0	1	d	
2	7	4	17/09/2015	NaN	NaN	1.0	1	d	
3	8	4	17/09/2015	NaN	NaN	1.0	1	d	
4	9	4	17/09/2015	NaN	NaN	1.0	1	d	

In [119…
```
# Step unique value count
agg_df.nunique()
```

Out[119]:
```
Store            1115
DayOfWeek           7
Date              990
Sales           21734
Customers        4085
Open                2
Promo               2
StateHoliday        4
SchoolHoliday       2
dtype: int64
```

In [120…
```
# Step  count
agg_df.count()
```

Out[120]:
```
Store            1058297
DayOfWeek        1058297
Date             1058297
Sales            1017209
Customers        1017209
Open             1058297
Promo            1058297
StateHoliday     1058297
SchoolHoliday    1058297
dtype: int64
```

In [121…
```
# Step head
agg_df.head()
```

Out[121]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHc
0	1	4	17/09/2015	NaN	NaN	1.0	1	d	
1	3	4	17/09/2015	NaN	NaN	1.0	1	d	
2	7	4	17/09/2015	NaN	NaN	1.0	1	d	
3	8	4	17/09/2015	NaN	NaN	1.0	1	d	
4	9	4	17/09/2015	NaN	NaN	1.0	1	d	

In [122…

```
import pandas as pd

# Assuming 'df1' and 'df2' are your DataFrames
merged_df = pd.merge(agg_df, store3, on='Store', how='outer')

# Verify the merged DataFrame
print(merged_df)
```

```
         Store  DayOfWeek        Date    Sales  Customers  Open  Promo  \
0            1          4  17/09/2015      NaN        NaN   1.0      1
1            1          3  16/09/2015      NaN        NaN   1.0      1
2            1          2  15/09/2015      NaN        NaN   1.0      1
3            1          1  14/09/2015      NaN        NaN   1.0      1
4            1          7  13/09/2015      NaN        NaN   0.0      0
...        ...        ...         ...     ...        ...   ...    ...
1058292   1110          6  05/01/2013   3156.0      428.0   1.0      0
1058293   1110          5  04/01/2013   3933.0      537.0   1.0      0
1058294   1110          4  03/01/2013   3508.0      491.0   1.0      0
1058295   1110          3  02/01/2013   4126.0      507.0   1.0      0
1058296   1110          2  01/01/2013      0.0        0.0   0.0      0

        StateHoliday  SchoolHoliday  CompetitionDistance  Promo2  \
0                  d              0               1270.0       0
1                  d              0               1270.0       0
2                  d              0               1270.0       0
3                  d              0               1270.0       0
4                  d              0               1270.0       0
...              ...            ...                  ...     ...
1058292            d              0                900.0       0
1058293            d              1                900.0       0
1058294            d              1                900.0       0
1058295            d              1                900.0       0
1058296            a              1                900.0       0

         PromoInterval  CompetitionDistanceDecile Type_Assort  PromoRun
0                  0.0                          4          Ca       0.0
1                  0.0                          4          Ca       0.0
2                  0.0                          4          Ca       0.0
3                  0.0                          4          Ca       0.0
4                  0.0                          4          Ca       0.0
...                ...                        ...         ...       ...
1058292            0.0                          3          Cc       0.0
1058293            0.0                          3          Cc       0.0
1058294            0.0                          3          Cc       0.0
1058295            0.0                          3          Cc       0.0
1058296            0.0                          3          Cc       0.0

[1058297 rows x 15 columns]
```

In [123…

```
# Merged df info
merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1058297 entries, 0 to 1058296
Data columns (total 15 columns):
 #   Column                      Non-Null Count    Dtype  
---  ------                      --------------    -----  
 0   Store                       1058297 non-null  int64  
 1   DayOfWeek                   1058297 non-null  int64  
 2   Date                        1058297 non-null  object 
 3   Sales                       1017209 non-null  float64
 4   Customers                   1017209 non-null  float64
 5   Open                        1058297 non-null  float64
 6   Promo                       1058297 non-null  int64  
 7   StateHoliday                1058297 non-null  object 
 8   SchoolHoliday               1058297 non-null  int64  
 9   CompetitionDistance         1058297 non-null  float64
 10  Promo2                      1058297 non-null  int64  
 11  PromoInterval               1058297 non-null  float64
 12  CompetitionDistanceDecile   1058297 non-null  int64  
 13  Type_Assort                 1058297 non-null  object 
 14  PromoRun                    1058297 non-null  float64
dtypes: float64(6), int64(6), object(3)
memory usage: 121.1+ MB
```

In [124…

```
# Step unique count
merged_df.nunique()
```

Out[124]:

```
Store                        1115
DayOfWeek                       7
Date                          990
Sales                       21734
Customers                    4085
Open                            2
Promo                           2
StateHoliday                    4
SchoolHoliday                   2
CompetitionDistance           655
Promo2                          2
PromoInterval                   4
CompetitionDistanceDecile      10
Type_Assort                     9
PromoRun                       56
dtype: int64
```

In [125…

```
# Step head
merged_df.head().T
```

Out[125]:

	0	1	2	3	4
Store	1	1	1	1	1
DayOfWeek	4	3	2	1	7
Date	17/09/2015	16/09/2015	15/09/2015	14/09/2015	13/09/2015
Sales	NaN	NaN	NaN	NaN	NaN
Customers	NaN	NaN	NaN	NaN	NaN
Open	1.0	1.0	1.0	1.0	0.0
Promo	1	1	1	1	0
StateHoliday	d	d	d	d	d
SchoolHoliday	0	0	0	0	0
CompetitionDistance	1270.0	1270.0	1270.0	1270.0	1270.0
Promo2	0	0	0	0	0
PromoInterval	0.0	0.0	0.0	0.0	0.0
CompetitionDistanceDecile	4	4	4	4	4
Type_Assort	Ca	Ca	Ca	Ca	Ca
PromoRun	0.0	0.0	0.0	0.0	0.0

In [126…

```
# Step count
merged_df.count()
```

Out[126]:

```
Store                       1058297
DayOfWeek                   1058297
Date                        1058297
Sales                       1017209
Customers                   1017209
Open                        1058297
Promo                       1058297
StateHoliday                1058297
SchoolHoliday               1058297
CompetitionDistance         1058297
Promo2                      1058297
PromoInterval               1058297
CompetitionDistanceDecile   1058297
Type_Assort                 1058297
PromoRun                    1058297
dtype: int64
```

In [127…

```
# Convert 'Date' column to datetime format
merged_df['Date'] = pd.to_datetime(merged_df['Date'], format='%d/%m/%Y')

# Create a new column 'DateIdentifier' to store the unique identifier for ea
merged_df['DateIdentifier'] = merged_df['Date'].rank(method='dense').astype(

# Print the DataFrame to verify the new column
merged_df.head(20).T
```

Out[127]:

	0	1	2	3	4	5	
Store	1	1	1	1	1	1	
DayOfWeek	4	3	2	1	7	6	
Date	2015-09-17 00:00:00	2015-09-16 00:00:00	2015-09-15 00:00:00	2015-09-14 00:00:00	2015-09-13 00:00:00	2015-09-12 00:00:00	00:(
Sales	NaN	NaN	NaN	NaN	NaN	NaN	
Customers	NaN	NaN	NaN	NaN	NaN	NaN	
Open	1.0	1.0	1.0	1.0	0.0	1.0	
Promo	1	1	1	1	0	0	
StateHoliday	d	d	d	d	d	d	
SchoolHoliday	0	0	0	0	0	0	
CompetitionDistance	1270.0	1270.0	1270.0	1270.0	1270.0	1270.0	1
Promo2	0	0	0	0	0	0	
PromoInterval	0.0	0.0	0.0	0.0	0.0	0.0	
CompetitionDistanceDecile	4	4	4	4	4	4	
Type_Assort	Ca	Ca	Ca	Ca	Ca	Ca	
PromoRun	0.0	0.0	0.0	0.0	0.0	0.0	
DateIdentifier	990	989	988	987	986	985	

In [128…

```
# Print the DataFrame to verify the new column
merged_df.tail()
```

Out[128]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	Scho
1058292	1110	6	2013-01-05	3156.0	428.0	1.0	0	d	
1058293	1110	5	2013-01-04	3933.0	537.0	1.0	0	d	
1058294	1110	4	2013-01-03	3508.0	491.0	1.0	0	d	
1058295	1110	3	2013-01-02	4126.0	507.0	1.0	0	d	
1058296	1110	2	2013-01-01	0.0	0.0	0.0	0	a	

In [129…

```
# Step unique count
merged_df.nunique()
```

Out[129]:
```
Store                        1115
DayOfWeek                       7
Date                          990
Sales                       21734
Customers                    4085
Open                            2
Promo                           2
StateHoliday                    4
SchoolHoliday                   2
CompetitionDistance           655
Promo2                          2
PromoInterval                   4
CompetitionDistanceDecile      10
Type_Assort                     9
PromoRun                       56
DateIdentifier                990
dtype: int64
```

In [130…
```
# Step value count
merged_df["Open"].value_counts()
```

Out[130]:
```
Open
1.0    879496
0.0    178801
Name: count, dtype: int64
```

In [131…
```
# Step value count
merged_df["Promo2"].value_counts()
```

Out[131]:
```
Promo2
1    533034
0    525263
Name: count, dtype: int64
```

In [132…
```
# Step value count
store3["PromoRun"].value_counts()
```

Out[132]:
```
PromoRun
0.0      544
225.0     63
43.0      48
104.0     37
307.0     35
278.0     34
130.0     33
73.0      32
299.0     30
165.0     25
186.0     19
199.0     16
256.0     14
221.0     14
230.0     14
134.0     13
147.0     11
125.0     10
99.0      10
217.0      8
152.0      7
313.0      7
121.0      7
65.0       6
173.0      6
252.0      5
160.0      5
108.0      4
17.0       4
204.0      4
191.0      4
139.0      4
273.0      4
82.0       3
286.0      3
234.0      3
38.0       3
8.0        3
95.0       2
212.0      2
143.0      2
13.0       2
117.0      2
247.0      1
159.0      1
243.0      1
60.0       1
34.0       1
305.0      1
241.0      1
69.0       1
112.0      1
169.0      1
25.0       1
265.0      1
90.0       1
Name: count, dtype: int64
```

In [133…
```
# Step value count
merged_df["PromoRun"].value_counts()
```

Out[133]:
```
PromoRun
0.0      525263
225.0     58050
43.0      38688
104.0     35710
307.0     34178
130.0     31550
278.0     31068
73.0      30936
299.0     26940
165.0     23094
186.0     18762
199.0     15288
221.0     13860
256.0     13308
230.0     12980
134.0     12270
147.0     10242
125.0      9668
99.0       9620
217.0      7136
152.0      6930
121.0      6930
313.0      6330
65.0       5892
173.0      5748
160.0      4214
252.0      4030
17.0       3960
108.0      3960
273.0      3960
191.0      3816
139.0      3776
204.0      3776
38.0       2970
82.0       2970
234.0      2922
286.0      2786
8.0        2602
95.0       1980
117.0      1980
143.0      1980
13.0       1932
212.0      1932
112.0       990
69.0        990
243.0       990
60.0        990
169.0       990
159.0       990
90.0        990
265.0       942
25.0        942
305.0       942
241.0       942
247.0       806
34.0        806
Name: count, dtype: int64
```

In [134…
```
# Step value count
merged_df['P2C'] = merged_df['PromoRun'] * 7 + merged_df['DateIdentifier']-9
merged_df["P2C"].value_counts()
```

Out[134]:
```
P2C
-4.0       720
-3.0       720
-2.0       720
-1.0       720
-5.0       707
           ...
 2223.0      6
 2222.0      6
 2221.0      6
 2220.0      6
 2197.0      6
Name: count, Length: 3181, dtype: int64
```

In [135…
```
# Calculate the min and max of the P2C column
min_p2c = merged_df['P2C'].min()
max_p2c = merged_df['P2C'].max()

# Print the min and max values
print(f"Minimum P2C: {min_p2c}")
print(f"Maximum P2C: {max_p2c}")
```

```
Minimum P2C: -942.0
Maximum P2C: 2238.0
```

In [136…
```
# Calculate the min and max of the P2C column
min_p2c = merged_df['P2C'].min()
max_p2c = merged_df['P2C'].max()

# Print the min and max values
print(f"Minimum P2C: {min_p2c}")
print(f"Maximum P2C: {max_p2c}")
```

```
Minimum P2C: -942.0
Maximum P2C: 2238.0
```

In [137…
```
# Step identifier
merged_df['P2cal'] = merged_df['PromoRun'] * 7 + merged_df['DateIdentifier']
```

In [138…
```
# Step convert to 0
merged_df['P2cal'] = merged_df['P2cal'].apply(lambda x: 1 if x > 0 else 0)
```

In [139…
```
# Step value count
merged_df["P2cal"].value_counts()
```

Out[139]:
```
P2cal
0    579063
1    479234
Name: count, dtype: int64
```

In [140…
```
# Step 1: Apply the condition to set P2cal to 0 where Promo2 is 0
merged_df['P2cal'] = merged_df.apply(lambda row: 0 if row['Promo2'] == 0 els

# Step 2: Count the occurrences of each value in the P2cal column
p2cal_counts = merged_df['P2cal'].value_counts()

# Print the counts
print(p2cal_counts)
```

```
P2cal
0    595936
1    462361
Name: count, dtype: int64
```

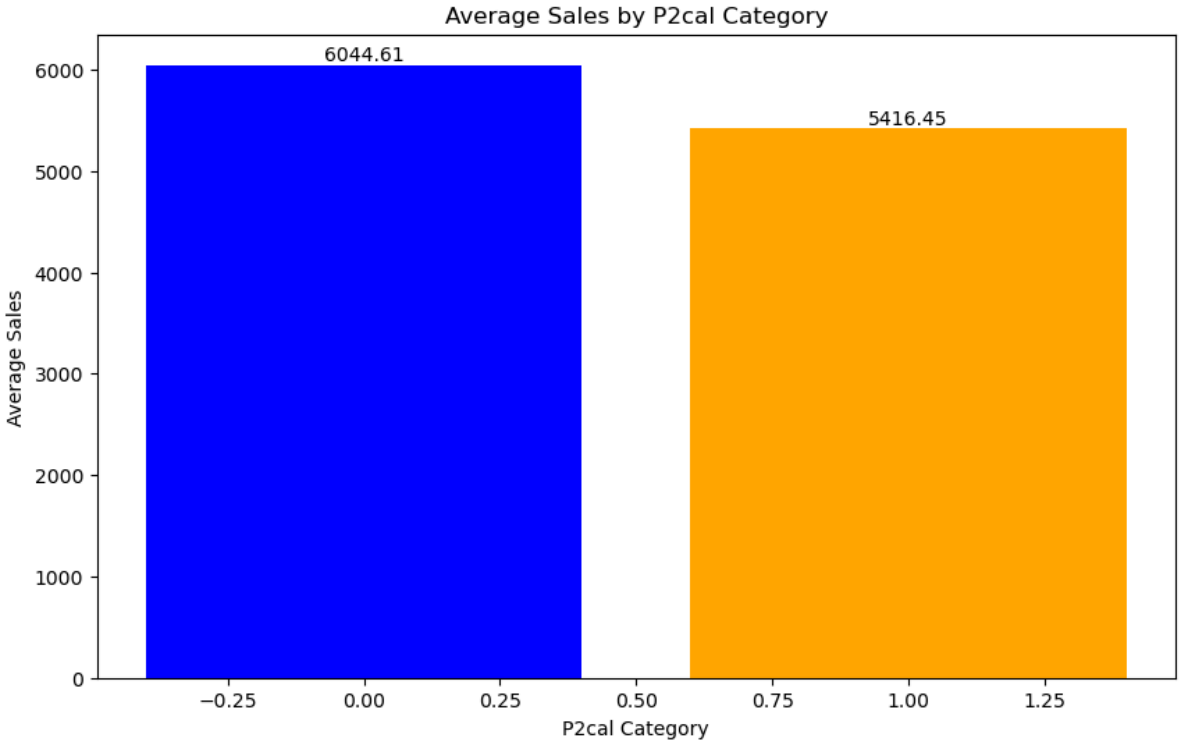
In [141…
```
595936-579063
```

Out[141]:
```
16873
```

In [142…
```
+16873/47
```

Out[142]:
```
359.0
```

In [143…
```
# Calculate the average sales for each P2cal category
avg_sales_by_p2cal = merged_df.groupby('P2cal')['Sales'].mean().reset_index(

# Plotting the bar chart
plt.figure(figsize=(10, 6))
plt.bar(avg_sales_by_p2cal['P2cal'], avg_sales_by_p2cal['Sales'], color=['b

# Adding titles and labels
plt.title('Average Sales by P2cal Category')
plt.xlabel('P2cal Category')
plt.ylabel('Average Sales')

# Adding the average sales values on top of the bars
for index, value in avg_sales_by_p2cal.iterrows():
    plt.text(value['P2cal'], value['Sales'], f'{value["Sales"]:.2f}', ha='ce

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



In [144…
```
# Filter the DataFrame for stores with P2Calc equal to 0
p2calc_0_stores = merged_df[merged_df['P2cal'] == 0]['Store'].unique()

# Filter the DataFrame for stores with P2Calc equal to 1
p2calc_1_stores = merged_df[merged_df['P2cal'] == 1]['Store'].unique()

# Find the intersection of stores with both P2Calc equal to 0 and 1
common_stores = np.intersect1d(p2calc_0_stores, p2calc_1_stores)

# Filter the original DataFrame for these common stores
common_stores_df = merged_df[merged_df['Store'].isin(common_stores)]
```

```
# Rename the DataFrame if needed
common_stores_df = common_stores_df.rename(columns={'P2cal': 'CommonP2cal'})

# Display the DataFrame
common_stores_df.head().T
```

Out[144]:

	12686	12687	12688	12689	12690
Store	20	20	20	20	20
DayOfWeek	4	3	2	1	7
Date	2015-09-17 00:00:00	2015-09-16 00:00:00	2015-09-15 00:00:00	2015-09-14 00:00:00	2015-09-13 00:00:00
Sales	NaN	NaN	NaN	NaN	NaN
Customers	NaN	NaN	NaN	NaN	NaN
Open	1.0	1.0	1.0	1.0	0.0
Promo	1	1	1	1	0
StateHoliday	d	d	d	d	d
SchoolHoliday	0	0	0	1	0
CompetitionDistance	2340.0	2340.0	2340.0	2340.0	2340.0
Promo2	1	1	1	1	1
PromoInterval	1.0	1.0	1.0	1.0	1.0
CompetitionDistanceDecile	6	6	6	6	6
Type_Assort	Da	Da	Da	Da	Da
PromoRun	43.0	43.0	43.0	43.0	43.0
DateIdentifier	990	989	988	987	986
P2C	348.0	347.0	346.0	345.0	344.0
CommonP2cal	1	1	1	1	1

In [145…

```
# Step tail
common_stores_df.tail().T
```

Out[145]:

	1034742	1034743	1034744	1034745	1034746
Store	1006	1006	1006	1006	1006
DayOfWeek	6	5	4	3	2
Date	2013-01-05 00:00:00	2013-01-04 00:00:00	2013-01-03 00:00:00	2013-01-02 00:00:00	2013-01-01 00:00:00
Sales	5408.0	4802.0	4577.0	5590.0	0.0
Customers	843.0	732.0	768.0	803.0	0.0
Open	1.0	1.0	1.0	1.0	0.0
Promo	0	0	0	0	0
StateHoliday	d	d	d	d	a
SchoolHoliday	0	1	1	1	1
CompetitionDistance	3890.0	3890.0	3890.0	3890.0	3890.0
Promo2	1	1	1	1	1
PromoInterval	2.0	2.0	2.0	2.0	2.0
CompetitionDistanceDecile	7	7	7	7	7
Type_Assort	Cc	Cc	Cc	Cc	Cc
PromoRun	130.0	130.0	130.0	130.0	130.0
DateIdentifier	5	4	3	2	1
P2C	-28.0	-29.0	-30.0	-31.0	-32.0
CommonP2cal	0	0	0	0	0

In [146…

```
# Step info
common_stores_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 209326 entries, 12686 to 1034746
Data columns (total 18 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   Store                      209326 non-null  int64
 1   DayOfWeek                  209326 non-null  int64
 2   Date                       209326 non-null  datetime64[ns]
 3   Sales                      199438 non-null  float64
 4   Customers                  199438 non-null  float64
 5   Open                       209326 non-null  float64
 6   Promo                      209326 non-null  int64
 7   StateHoliday               209326 non-null  object
 8   SchoolHoliday              209326 non-null  int64
 9   CompetitionDistance        209326 non-null  float64
 10  Promo2                     209326 non-null  int64
 11  PromoInterval              209326 non-null  float64
 12  CompetitionDistanceDecile  209326 non-null  int64
 13  Type_Assort                209326 non-null  object
 14  PromoRun                   209326 non-null  float64
 15  DateIdentifier             209326 non-null  int64
 16  P2C                        209326 non-null  float64
 17  CommonP2cal                209326 non-null  int64
dtypes: datetime64[ns](1), float64(7), int64(8), object(2)
memory usage: 30.3+ MB
```

In [147…
```
# Step unique count
common_stores_df.nunique()
```

Out[147]:
```
Store                        225
DayOfWeek                      7
Date                         990
Sales                      14441
Customers                  2456
Open                          2
Promo                         2
StateHoliday                  4
SchoolHoliday                 2
CompetitionDistance         188
Promo2                        1
PromoInterval                 3
CompetitionDistanceDecile    10
Type_Assort                   8
PromoRun                     23
DateIdentifier              990
P2C                        1872
CommonP2cal                   2
dtype: int64
```

In [148…
```
# Step value count
common_stores_df["CommonP2cal"].value_counts()
```

Out[148]:
```
CommonP2cal
1    138653
0     70673
Name: count, dtype: int64
```

In [149…
```
# Filter the DataFrame to only include rows where Sales is greater than 0
fdf = common_stores_df[common_stores_df['Sales'] > 0]

# Print the first few rows of the filtered DataFrame to verify
print(fdf.head())

# Print the shape of the filtered DataFrame to see the new size
print(fdf.shape)
```

```
       Store  DayOfWeek       Date    Sales  Customers  Open  Promo  \
12734     20          5 2015-07-31   9593.0      974.0   1.0      1
12735     20          4 2015-07-30  12379.0     1185.0   1.0      1
12736     20          3 2015-07-29  10728.0     1040.0   1.0      1
12737     20          2 2015-07-28  10884.0     1048.0   1.0      1
12738     20          1 2015-07-27  11255.0     1183.0   1.0      1

      StateHoliday  SchoolHoliday  CompetitionDistance  Promo2  PromoInterv
al  \
12734            d              0               2340.0       1
1.0
12735            d              0               2340.0       1
1.0
12736            d              0               2340.0       1
1.0
12737            d              0               2340.0       1
1.0
12738            d              0               2340.0       1
1.0

       CompetitionDistanceDecile Type_Assort  PromoRun  DateIdentifier    P
2C  \
12734                          6          Da      43.0             942  30
0.0
12735                          6          Da      43.0             941  29
9.0
12736                          6          Da      43.0             940  29
8.0
12737                          6          Da      43.0             939  29
7.0
12738                          6          Da      43.0             938  29
6.0

       CommonP2cal
12734            1
12735            1
12736            1
12737            1
12738            1
(165063, 18)
```

In [150…

```
# Calculate the min and max of the P2C column
min_p2c = fdf['PromoRun'].min()
max_p2c = fdf['PromoRun'].max()

# Print the min and max values
print(f"Minimum P2C: {min_p2c}")
print(f"Maximum P2C: {max_p2c}")
```

```
Minimum P2C: 8.0
Maximum P2C: 134.0
```

In [151…

```
# Group by 'Store' and 'CommonP2cal', and calculate the average sales
grouped_sales = fdf.groupby(['Store', 'CommonP2cal'])['Sales'].mean().reset_

# Print the first few rows to verify the result
print(grouped_sales.head(10))
```

```
   Store  CommonP2cal        Sales
0     20            0  7556.577528
1     20            1  8269.344828
2     28            0  5461.003205
3     28            1  4675.583333
4     30            0  5973.762570
5     30            1  4714.651551
6     36            0  9229.238839
7     36            1  9709.678161
8     39            0  4345.865169
9     39            1  5002.643927
```

In [152…
```
# Pivot the DataFrame to have 'CommonP2cal' as columns
pivot_sales = grouped_sales.pivot(index='Store', columns='CommonP2cal', valu

# Calculate the difference between the average sales for each store
pivot_sales['Sales_Difference'] = pivot_sales[0] - pivot_sales[1]

# Print the first few rows to verify the result
print(pivot_sales.head())
```

```
CommonP2cal            0            1  Sales_Difference
Store
20           7556.577528  8269.344828       -712.767299
28           5461.003205  4675.583333        785.419872
30           5973.762570  4714.651551       1259.111019
36           9229.238839  9709.678161       -480.439322
39           4345.865169  5002.643927       -656.778758
```

In [153…
```
# Calculate the sum of average sales for stores when CommonP2cal is 0 and 1
sum_average_sales_0 = pivot_sales[0].sum()
sum_average_sales_1 = pivot_sales[1].sum()

# Calculate the difference in the sum of average sales
sum_sales_difference = sum_average_sales_0 - sum_average_sales_1

# Print the results
print("Sum of Average Sales for CommonP2cal = 0:", sum_average_sales_0)
print("Sum of Average Sales for CommonP2cal = 1:", sum_average_sales_1)
print("Difference in Sum of Average Sales:", sum_sales_difference)
```

```
Sum of Average Sales for CommonP2cal = 0: 1358983.4682871385
Sum of Average Sales for CommonP2cal = 1: 1474976.7888079167
Difference in Sum of Average Sales: -115993.32052077819
```

In [154…
```
1358983.46/225
```

Out[154]:
```
6039.926488888888
```

In [155…
```
1474976.78/225
```

Out[155]:
```
6555.452355555556
```

In [156…
```
6555.452355555556-6039.926488888888
```

Out[156]:
```
515.5258666666678
```

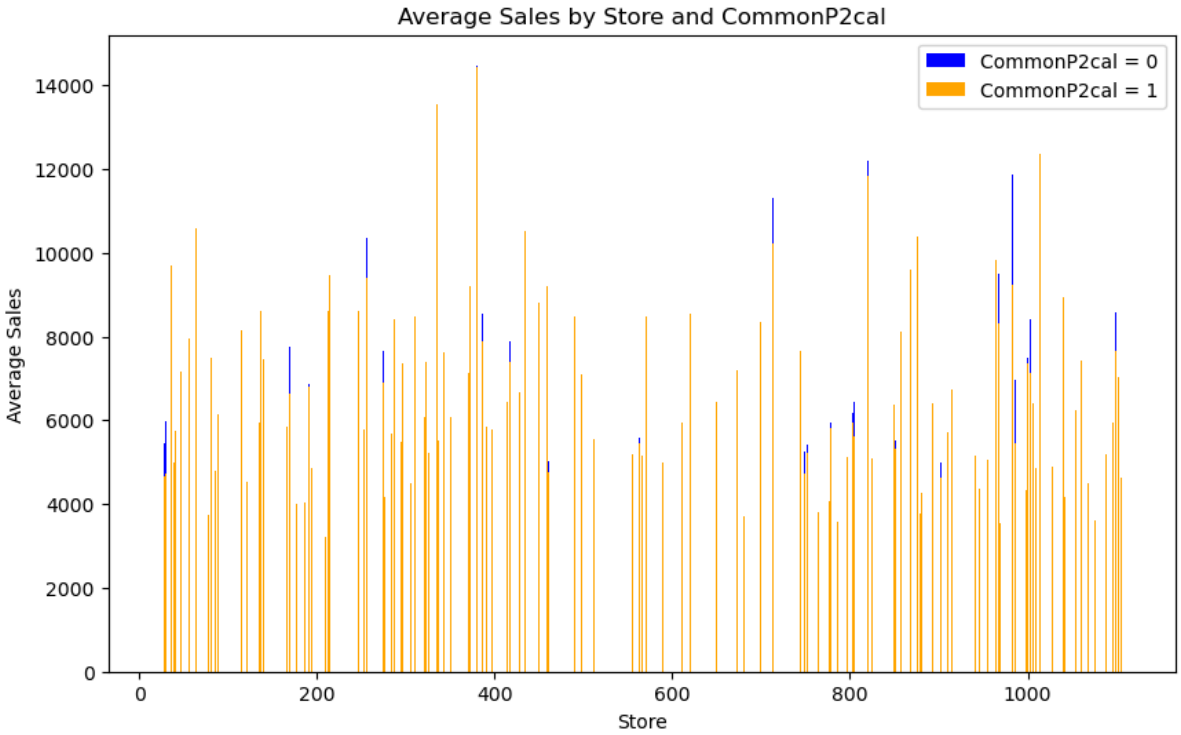
In [157…
```
# Calculate sales increase by Promo2 (8.54%)
(6555.452355555556-6039.926488888888)/6039.926488888888
```

Out[157]:
```
0.08535300348688589
```

In [158…
```
# Calculate the total difference between average sales at 0 and 1 for all st
total_difference = pivot_sales['Sales_Difference'].sum()

# Count the number of positive differences
positive_differences = (pivot_sales['Sales_Difference'] > 0).sum()

# Count the number of negative differences
negative_differences = (pivot_sales['Sales_Difference'] < 0).sum()

# Print the results
print("Total Difference:", total_difference)
print("Number of Positive Differences:", positive_differences)
print("Number of Negative Differences:", negative_differences)
```

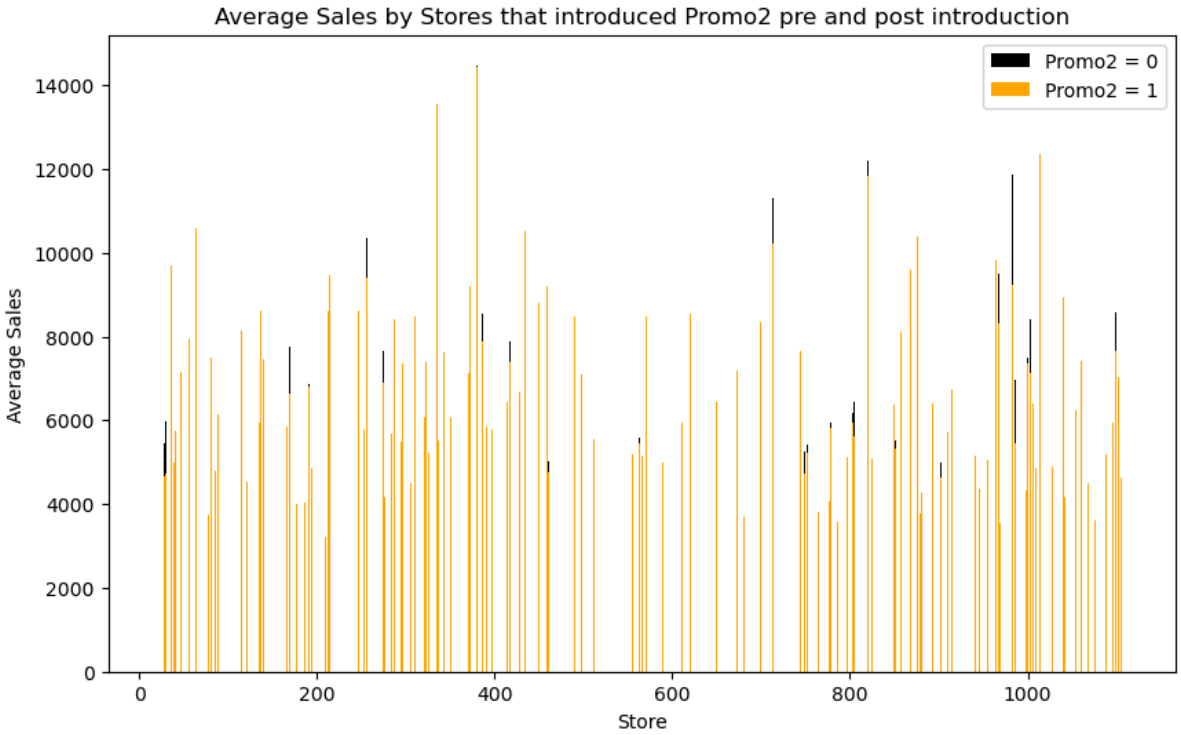
```
Total Difference: -115993.32052077833
Number of Positive Differences: 39
Number of Negative Differences: 186
```

In [159…
```
import matplotlib.pyplot as plt

# Filter the DataFrame for stores with CommonP2cal equal to 0
common_p2cal_0 = grouped_sales[grouped_sales['CommonP2cal'] == 0]

# Filter the DataFrame for stores with CommonP2cal equal to 1
common_p2cal_1 = grouped_sales[grouped_sales['CommonP2cal'] == 1]

# Set up the bar plot
plt.figure(figsize=(10, 6))

# Plot the average sales for CommonP2cal = 0
plt.bar(common_p2cal_0['Store'], common_p2cal_0['Sales'], color='blue', labe

# Plot the average sales for CommonP2cal = 1
plt.bar(common_p2cal_1['Store'], common_p2cal_1['Sales'], color='orange', la

# Add labels and title
plt.xlabel('Store')
plt.ylabel('Average Sales')
plt.title('Average Sales by Store and CommonP2cal')
plt.legend()

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



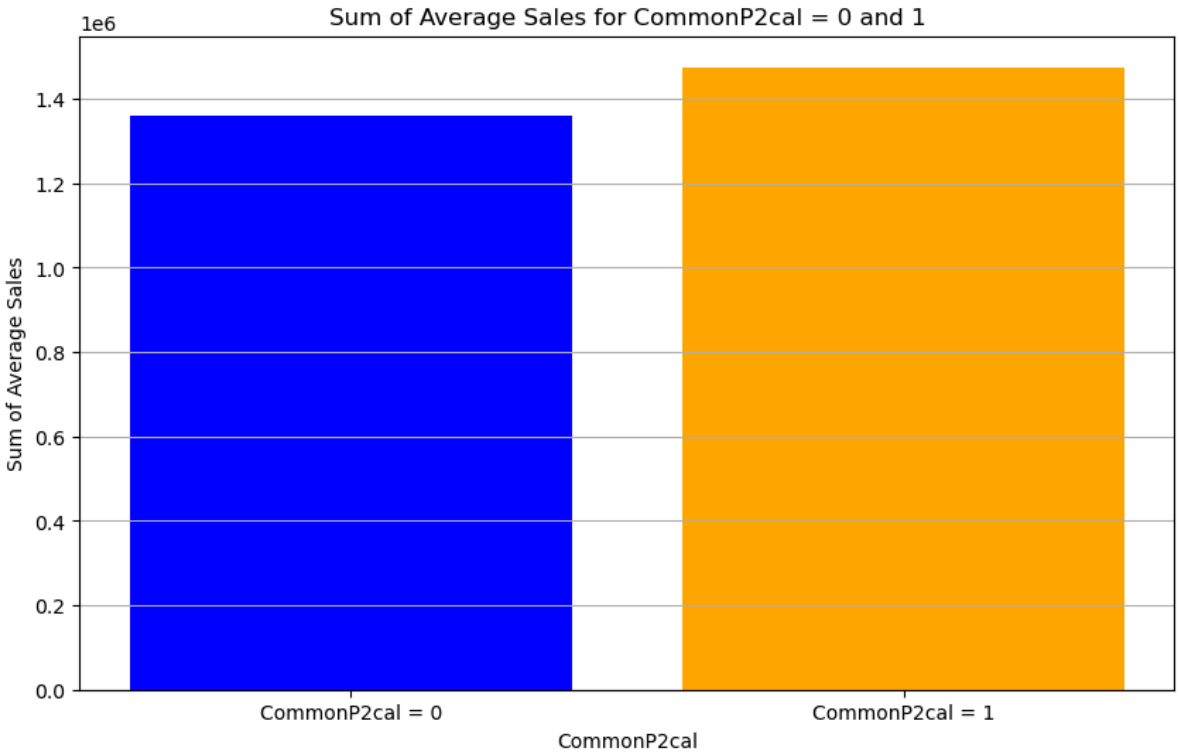
In [160…

```
import matplotlib.pyplot as plt

# Filter the DataFrame for stores with CommonP2cal equal to 0
common_p2cal_0 = grouped_sales[grouped_sales['CommonP2cal'] == 0]

# Filter the DataFrame for stores with CommonP2cal equal to 1
common_p2cal_1 = grouped_sales[grouped_sales['CommonP2cal'] == 1]

# Set up the bar plot
plt.figure(figsize=(10, 6))

# Plot the average sales for CommonP2cal = 0
plt.bar(common_p2cal_0['Store'], common_p2cal_0['Sales'], color='black', lab

# Plot the average sales for CommonP2cal = 1
plt.bar(common_p2cal_1['Store'], common_p2cal_1['Sales'], color='orange', la

# Add labels and title
plt.xlabel('Store')
plt.ylabel('Average Sales')
plt.title('Average Sales by Stores that introduced Promo2 pre and post intro
plt.legend()

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



In [161…

```
import matplotlib.pyplot as plt

# Create a bar chart
plt.figure(figsize=(10, 6))

# Plot the sum of average sales for CommonP2cal = 0 and 1
plt.bar(['CommonP2cal = 0', 'CommonP2cal = 1'], [sum_average_sales_0, sum_av

# Add labels and title
plt.xlabel('CommonP2cal')
plt.ylabel('Sum of Average Sales')
plt.title('Sum of Average Sales for CommonP2cal = 0 and 1')
plt.grid(axis='y')

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



In [162…
```
# Calculate the week number
merged_df['WeekNumber'] = (merged_df['DateIdentifier'] - 1) // 7 + 1

# Print the DataFrame to verify the new column
print(merged_df.head())
```

```
   Store  DayOfWeek       Date  Sales  Customers  Open  Promo StateHoliday
\
0      1          4 2015-09-17    NaN        NaN   1.0      1            d
1      1          3 2015-09-16    NaN        NaN   1.0      1            d
2      1          2 2015-09-15    NaN        NaN   1.0      1            d
3      1          1 2015-09-14    NaN        NaN   1.0      1            d
4      1          7 2015-09-13    NaN        NaN   0.0      0            d

   SchoolHoliday  CompetitionDistance  Promo2  PromoInterval  \
0              0               1270.0       0            0.0
1              0               1270.0       0            0.0
2              0               1270.0       0            0.0
3              0               1270.0       0            0.0
4              0               1270.0       0            0.0

   CompetitionDistanceDecile Type_Assort  PromoRun  DateIdentifier   P2C  \
0                          4          Ca       0.0             990  47.0
1                          4          Ca       0.0             989  46.0
2                          4          Ca       0.0             988  45.0
3                          4          Ca       0.0             987  44.0
4                          4          Ca       0.0             986  43.0

   P2cal  WeekNumber
0      0         142
1      0         142
2      0         142
3      0         141
4      0         141
```

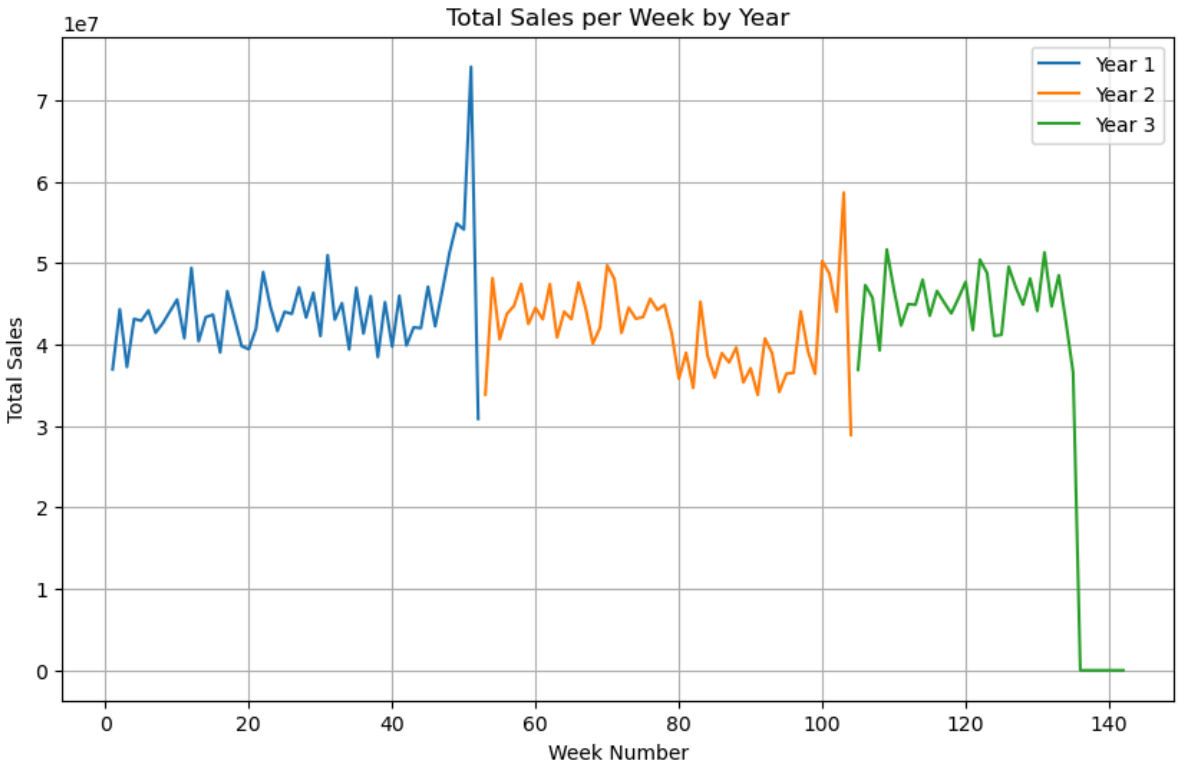
In [163…
```
merged_df.tail().T
```

Out[163]:

	1058292	1058293	1058294	1058295	1058296
Store	1110	1110	1110	1110	1110
DayOfWeek	6	5	4	3	2
Date	2013-01-05 00:00:00	2013-01-04 00:00:00	2013-01-03 00:00:00	2013-01-02 00:00:00	2013-01-01 00:00:00
Sales	3156.0	3933.0	3508.0	4126.0	0.0
Customers	428.0	537.0	491.0	507.0	0.0
Open	1.0	1.0	1.0	1.0	0.0
Promo	0	0	0	0	0
StateHoliday	d	d	d	d	a
SchoolHoliday	0	1	1	1	1
CompetitionDistance	900.0	900.0	900.0	900.0	900.0
Promo2	0	0	0	0	0
PromoInterval	0.0	0.0	0.0	0.0	0.0
CompetitionDistanceDecile	3	3	3	3	3
Type_Assort	Cc	Cc	Cc	Cc	Cc
PromoRun	0.0	0.0	0.0	0.0	0.0
DateIdentifier	5	4	3	2	1
P2C	-938.0	-939.0	-940.0	-941.0	-942.0
P2cal	0	0	0	0	0
WeekNumber	1	1	1	1	1

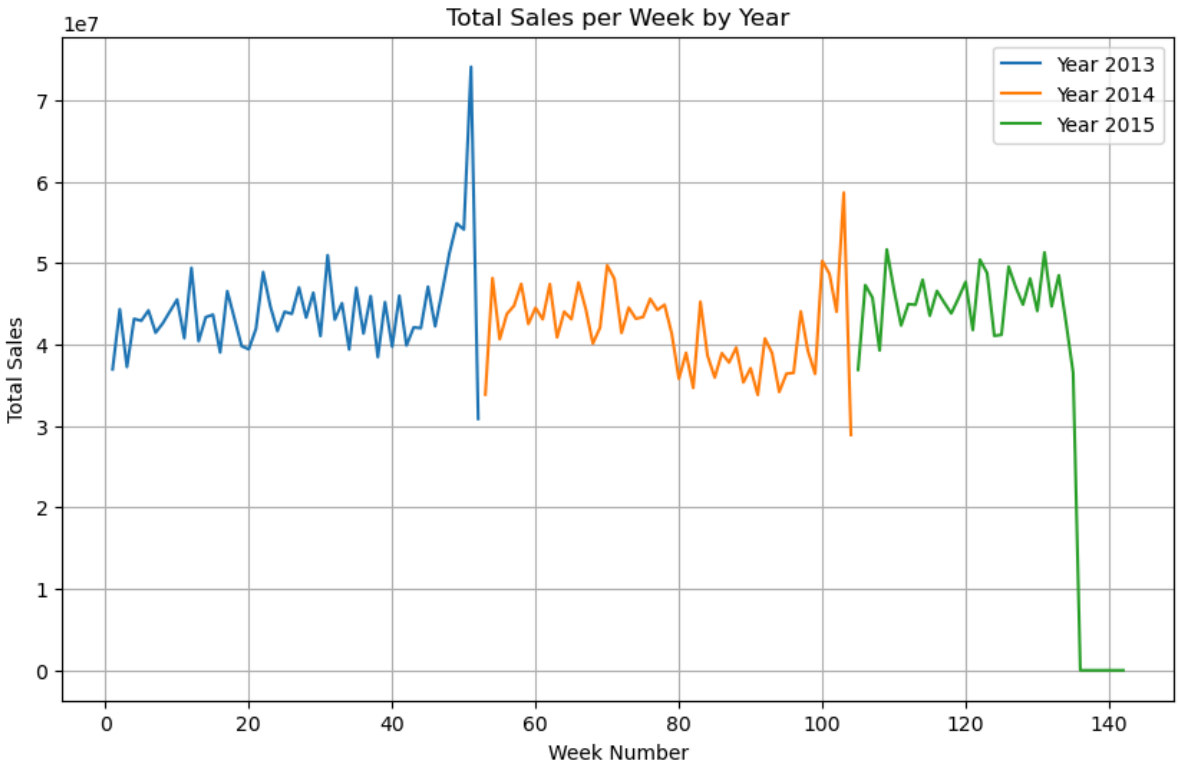
In [164…

```
import matplotlib.pyplot as plt

# Assuming 'train' is your DataFrame

# Calculate the year number based on the week number
merged_df['Year'] = ((merged_df['WeekNumber'] - 1) // 52) + 1

# Aggregate sales data by year and week, and calculate total sales per week
weekly_sales_by_year = merged_df.groupby(['Year', 'WeekNumber'])['Sales'].su

# Plot the total sales per week for each year
plt.figure(figsize=(10, 6))

for year in range(1, merged_df['Year'].max() + 1):
    sales_year = weekly_sales_by_year.loc[year]
    plt.plot(sales_year.index, sales_year.values, label=f'Year {year}')

plt.title('Total Sales per Week by Year')
plt.xlabel('Week Number')
plt.ylabel('Total Sales')
plt.grid(True)
plt.legend()
plt.show()
```



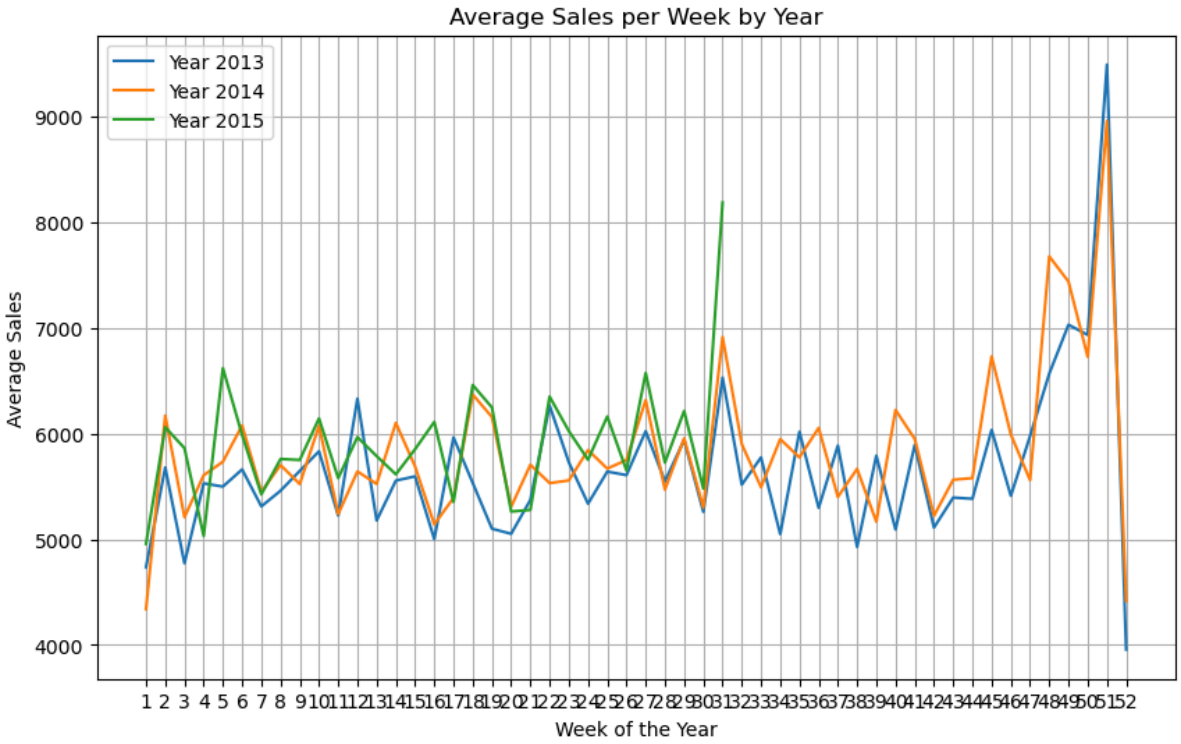
In [165…

```
import matplotlib.pyplot as plt

# Assuming 'merged_df' is your DataFrame

# Calculate the year number based on the week number
merged_df['Year'] = ((merged_df['WeekNumber'] - 1) // 52) + 1

# Map the year numbers to actual years
year_map = {1: 2013, 2: 2014, 3: 2015}

# Aggregate sales data by year and week, and calculate total sales per week
weekly_sales_by_year = merged_df.groupby(['Year', 'WeekNumber'])['Sales'].su

# Plot the total sales per week for each year
plt.figure(figsize=(10, 6))

for year in range(1, merged_df['Year'].max() + 1):
    sales_year = weekly_sales_by_year.loc[year]
    plt.plot(sales_year.index, sales_year.values, label=f'Year {year_map[yea

plt.title('Total Sales per Week by Year')
plt.xlabel('Week Number')
plt.ylabel('Total Sales')
plt.grid(True)
plt.legend()
plt.show()
```



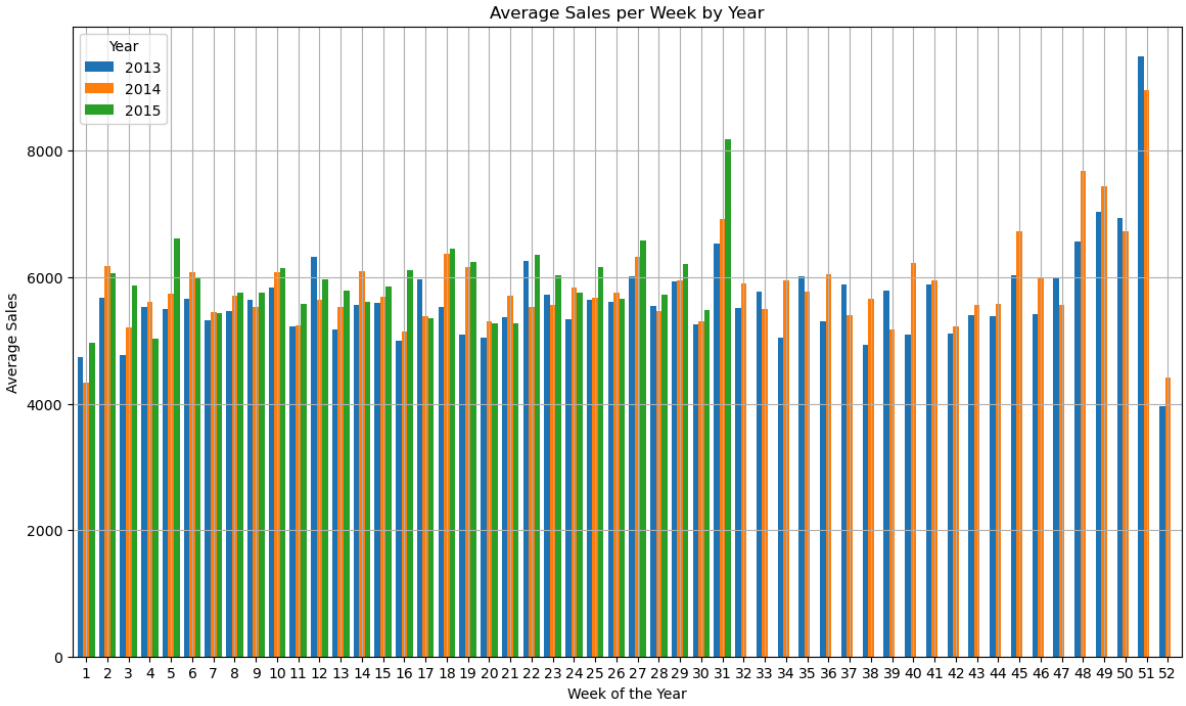
In [166…

```
# Calculate the week of the year (1-52)
merged_df['WeekOfYear'] = (merged_df['WeekNumber'] - 1) % 52 + 1

# Calculate the year (2013, 2014, 2015)
merged_df['Year'] = ((merged_df['WeekNumber'] - 1) // 52) + 2013

# Group data by year and week, and calculate average sales per week
weekly_average_sales = merged_df.groupby(['Year', 'WeekOfYear'])['Sales'].me

# Plot the average sales per week for each year
plt.figure(figsize=(10, 6))

for year in range(2013, 2016):
    year_data = weekly_average_sales[weekly_average_sales['Year'] == year]
    plt.plot(year_data['WeekOfYear'], year_data['Sales'], label=f'Year {yea

plt.title('Average Sales per Week by Year')
plt.xlabel('Week of the Year')
plt.ylabel('Average Sales')
plt.grid(True)
plt.xticks(range(1, 53))  # Set x-axis ticks to display weeks 1-52
plt.legend()

plt.show()
```



In [167…

```
import pandas as pd
import matplotlib.pyplot as plt

# Calculate the week of the year (1-52)
merged_df['WeekOfYear'] = (merged_df['WeekNumber'] - 1) % 52 + 1

# Calculate the year (2013, 2014, 2015)
merged_df['Year'] = ((merged_df['WeekNumber'] - 1) // 52) + 2013

# Group data by year and week, and calculate average sales per week
weekly_average_sales = merged_df.groupby(['Year', 'WeekOfYear'])['Sales'].me

# Pivot the data for easier plotting
pivot_df = weekly_average_sales.pivot(index='WeekOfYear', columns='Year', va

# Plot the average sales per week for each year as a bar chart
pivot_df.plot(kind='bar', figsize=(14, 8), width=0.8)

plt.title('Average Sales per Week by Year')
plt.xlabel('Week of the Year')
plt.ylabel('Average Sales')
plt.grid(True)
plt.xticks(rotation=0)  # Rotate x-axis labels to 0 degrees for better reada
plt.legend(title='Year')

plt.show()
```



In [168…
```
# Step info
merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1058297 entries, 0 to 1058296
Data columns (total 21 columns):
 #   Column                     Non-Null Count    Dtype
---  ------                     --------------    -----
 0   Store                      1058297 non-null  int64
 1   DayOfWeek                  1058297 non-null  int64
 2   Date                       1058297 non-null  datetime64[ns]
 3   Sales                      1017209 non-null  float64
 4   Customers                  1017209 non-null  float64
 5   Open                       1058297 non-null  float64
 6   Promo                      1058297 non-null  int64
 7   StateHoliday               1058297 non-null  object
 8   SchoolHoliday              1058297 non-null  int64
 9   CompetitionDistance        1058297 non-null  float64
 10  Promo2                     1058297 non-null  int64
 11  PromoInterval              1058297 non-null  float64
 12  CompetitionDistanceDecile  1058297 non-null  int64
 13  Type_Assort                1058297 non-null  object
 14  PromoRun                   1058297 non-null  float64
 15  DateIdentifier             1058297 non-null  int64
 16  P2C                        1058297 non-null  float64
 17  P2cal                      1058297 non-null  int64
 18  WeekNumber                 1058297 non-null  int64
 19  Year                       1058297 non-null  int64
 20  WeekOfYear                 1058297 non-null  int64
dtypes: datetime64[ns](1), float64(7), int64(11), object(2)
memory usage: 169.6+ MB
```

In [169…
```
# Step describe
merged_df.describe().T
```

Out[169]:

	count	mean	min	25%	50%	
Store	1058297.0	558.331493	1.0	280.0	558.0	
DayOfWeek	1058297.0	3.997596	1.0	2.0	4.0	
Date	1058297	2014-04-30 11:48:53.200415488	2013-01-01 00:00:00	2013-08-26 00:00:00	2014-04-20 00:00:00	20 00
Sales	1017209.0	5773.818972	0.0	3727.0	5744.0	
Customers	1017209.0	633.142407	0.0	405.0	609.0	
Open	1058297.0	0.831048	0.0	1.0	1.0	
Promo	1058297.0	0.38207	0.0	0.0	0.0	
SchoolHoliday	1058297.0	0.188929	0.0	0.0	0.0	
CompetitionDistance	1058297.0	5416.793017	20.0	710.0	2330.0	
Promo2	1058297.0	0.503671	0.0	0.0	1.0	
PromoInterval	1058297.0	0.813569	0.0	0.0	1.0	
CompetitionDistanceDecile	1058297.0	5.483524	1.0	3.0	5.0	
PromoRun	1058297.0	88.900305	0.0	0.0	13.0	
DateIdentifier	1058297.0	485.492282	1.0	238.0	475.0	
P2C	1058297.0	164.794417	-942.0	-479.0	-72.0	
P2cal	1058297.0	0.436892	0.0	0.0	0.0	
WeekNumber	1058297.0	69.785393	1.0	34.0	68.0	
Year	1058297.0	2013.880448	2013.0	2013.0	2014.0	
WeekOfYear	1058297.0	24.002071	1.0	12.0	23.0	

In [170…

```
# Step head
dfz = merged_df[merged_df['Sales'] != 0]

# Print the first few rows of the new DataFrame to verify
dfz.head().T
```

Out[170]:

	0	1	2	3	4
Store	1	1	1	1	1
DayOfWeek	4	3	2	1	7
Date	2015-09-17 00:00:00	2015-09-16 00:00:00	2015-09-15 00:00:00	2015-09-14 00:00:00	2015-09-13 00:00:00
Sales	NaN	NaN	NaN	NaN	NaN
Customers	NaN	NaN	NaN	NaN	NaN
Open	1.0	1.0	1.0	1.0	0.0
Promo	1	1	1	1	0
StateHoliday	d	d	d	d	d
SchoolHoliday	0	0	0	0	0
CompetitionDistance	1270.0	1270.0	1270.0	1270.0	1270.0
Promo2	0	0	0	0	0
PromoInterval	0.0	0.0	0.0	0.0	0.0
CompetitionDistanceDecile	4	4	4	4	4
Type_Assort	Ca	Ca	Ca	Ca	Ca
PromoRun	0.0	0.0	0.0	0.0	0.0
DateIdentifier	990	989	988	987	986
P2C	47.0	46.0	45.0	44.0	43.0
P2cal	0	0	0	0	0
WeekNumber	142	142	142	141	141
Year	2015	2015	2015	2015	2015
WeekOfYear	38	38	38	37	37

In [171…

```
# Step info
dfz.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 885426 entries, 0 to 1058295
Data columns (total 21 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   Store                      885426 non-null  int64
 1   DayOfWeek                  885426 non-null  int64
 2   Date                       885426 non-null  datetime64[ns]
 3   Sales                      844338 non-null  float64
 4   Customers                  844338 non-null  float64
 5   Open                       885426 non-null  float64
 6   Promo                      885426 non-null  int64
 7   StateHoliday               885426 non-null  object
 8   SchoolHoliday              885426 non-null  int64
 9   CompetitionDistance        885426 non-null  float64
 10  Promo2                     885426 non-null  int64
 11  PromoInterval              885426 non-null  float64
 12  CompetitionDistanceDecile  885426 non-null  int64
 13  Type_Assort                885426 non-null  object
 14  PromoRun                   885426 non-null  float64
 15  DateIdentifier             885426 non-null  int64
 16  P2C                        885426 non-null  float64
 17  P2cal                      885426 non-null  int64
 18  WeekNumber                 885426 non-null  int64
 19  Year                       885426 non-null  int64
 20  WeekOfYear                 885426 non-null  int64
dtypes: datetime64[ns](1), float64(7), int64(11), object(2)
memory usage: 148.6+ MB
```

In [172…

```
# Get the top 10 rows with the largest "Sales"
top_10_sales = dfz.nlargest(10, 'Sales')

top_10_sales.T
```

Out[172]:

	667283	190693	190665	190652	826975	989181	87
Store	909	262	262	262	57	817	
DayOfWeek	1	5	5	4	1	1	
Date	2015-06-22 00:00:00	2015-04-03 00:00:00	2015-05-01 00:00:00	2015-05-14 00:00:00	2014-06-16 00:00:00	2013-12-16 00:00:00	201 00:(
Sales	41551.0	38722.0	38484.0	38367.0	38037.0	38025.0	37(
Customers	1721.0	5132.0	5458.0	5192.0	1970.0	4381.0	1
Open	1.0	1.0	1.0	1.0	1.0	1.0	
Promo	0	1	1	0	1	1	
StateHoliday	d	b	a	a	d	d	
SchoolHoliday	0	0	0	1	0	0	
CompetitionDistance	1680.0	1180.0	1180.0	1180.0	420.0	140.0	15
Promo2	1	0	0	0	0	0	
PromoInterval	2.0	0.0	0.0	0.0	0.0	0.0	
CompetitionDistanceDecile	5	4	4	4	2	1	
Type_Assort	Ac	Ba	Ba	Ba	Dc	Aa	
PromoRun	299.0	0.0	0.0	0.0	0.0	0.0	
DateIdentifier	903	823	851	864	532	350	
P2C	2053.0	-120.0	-92.0	-79.0	-411.0	-593.0	1
P2cal	1	0	0	0	0	0	
WeekNumber	129	118	122	124	76	50	
Year	2015	2015	2015	2015	2014	2013	
WeekOfYear	25	14	18	20	24	50	

In [173…

```
import pandas as pd

# Reindex to move 'Sales' to the first column
cols = ['Sales'] + [col for col in merged_df.columns if col != 'Sales']
df_reindexed = dfz.reindex(columns=cols)

# Drop the columns 'Customers', 'Open', and 'Promo2'
dfzc = df_reindexed.drop(columns=['Customers', "Date", "P2C", 'Promo2'])

dfzc.head().T
```

Out[173]:

	0	1	2	3	4
Sales	NaN	NaN	NaN	NaN	NaN
Store	1	1	1	1	1
DayOfWeek	4	3	2	1	7
Open	1.0	1.0	1.0	1.0	0.0
Promo	1	1	1	1	0
StateHoliday	d	d	d	d	d
SchoolHoliday	0	0	0	0	0
CompetitionDistance	1270.0	1270.0	1270.0	1270.0	1270.0
PromoInterval	0.0	0.0	0.0	0.0	0.0
CompetitionDistanceDecile	4	4	4	4	4
Type_Assort	Ca	Ca	Ca	Ca	Ca
PromoRun	0.0	0.0	0.0	0.0	0.0
DateIdentifier	990	989	988	987	986
P2cal	0	0	0	0	0
WeekNumber	142	142	142	141	141
Year	2015	2015	2015	2015	2015
WeekOfYear	38	38	38	37	37

In [174…

```
# Step value count
dfzc.describe().T
```

Out[174]:

	count	mean	std	min	25%	50%	
Sales	844338.0	6955.959134	3103.815515	46.0	4859.0	6369.0	8
Store	885426.0	558.304348	321.663682	1.0	280.0	558.0	
DayOfWeek	885426.0	3.541641	1.741012	1.0	2.0	4.0	
Open	885426.0	0.993242	0.081931	0.0	1.0	1.0	
Promo	885426.0	0.444011	0.496856	0.0	0.0	0.0	
SchoolHoliday	885426.0	0.205175	0.403829	0.0	0.0	0.0	
CompetitionDistance	885426.0	5440.909720	7773.780012	20.0	710.0	2330.0	6
PromoInterval	885426.0	0.811643	0.981459	0.0	0.0	1.0	
CompetitionDistanceDecile	885426.0	5.483130	2.896799	1.0	3.0	5.0	
PromoRun	885426.0	88.646062	106.564424	0.0	0.0	13.0	
DateIdentifier	885426.0	489.271051	287.997254	1.0	240.0	479.0	
P2cal	885426.0	0.436699	0.495977	0.0	0.0	0.0	
WeekNumber	885426.0	70.370649	41.140541	1.0	35.0	69.0	
Year	885426.0	2013.889520	0.798811	2013.0	2013.0	2014.0	2
WeekOfYear	885426.0	24.115617	14.264523	1.0	12.0	24.0	

In [175…
```
# Get the top 10 rows with the largest "Sales"
top_10_sales = dfzc.nlargest(10, 'Sales')

top_10_sales.T
```

Out[175]:

	667283	190693	190665	190652	826975	989181	870489
Sales	41551.0	38722.0	38484.0	38367.0	38037.0	38025.0	37646.0
Store	909	262	262	262	57	817	261
DayOfWeek	1	5	5	4	1	1	1
Open	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Promo	0	1	1	0	1	1	1
StateHoliday	d	b	a	a	d	d	d
SchoolHoliday	0	0	0	1	0	0	0
CompetitionDistance	1680.0	1180.0	1180.0	1180.0	420.0	140.0	15340.0
PromoInterval	2.0	0.0	0.0	0.0	0.0	0.0	1.0
CompetitionDistanceDecile	5	4	4	4	2	1	9
Type_Assort	Ac	Ba	Ba	Ba	Dc	Aa	Dc
PromoRun	299.0	0.0	0.0	0.0	0.0	0.0	305.0
DateIdentifier	903	823	851	864	532	350	350
P2cal	1	0	0	0	0	0	1
WeekNumber	129	118	122	124	76	50	50
Year	2015	2015	2015	2015	2014	2013	2013
WeekOfYear	25	14	18	20	24	50	50

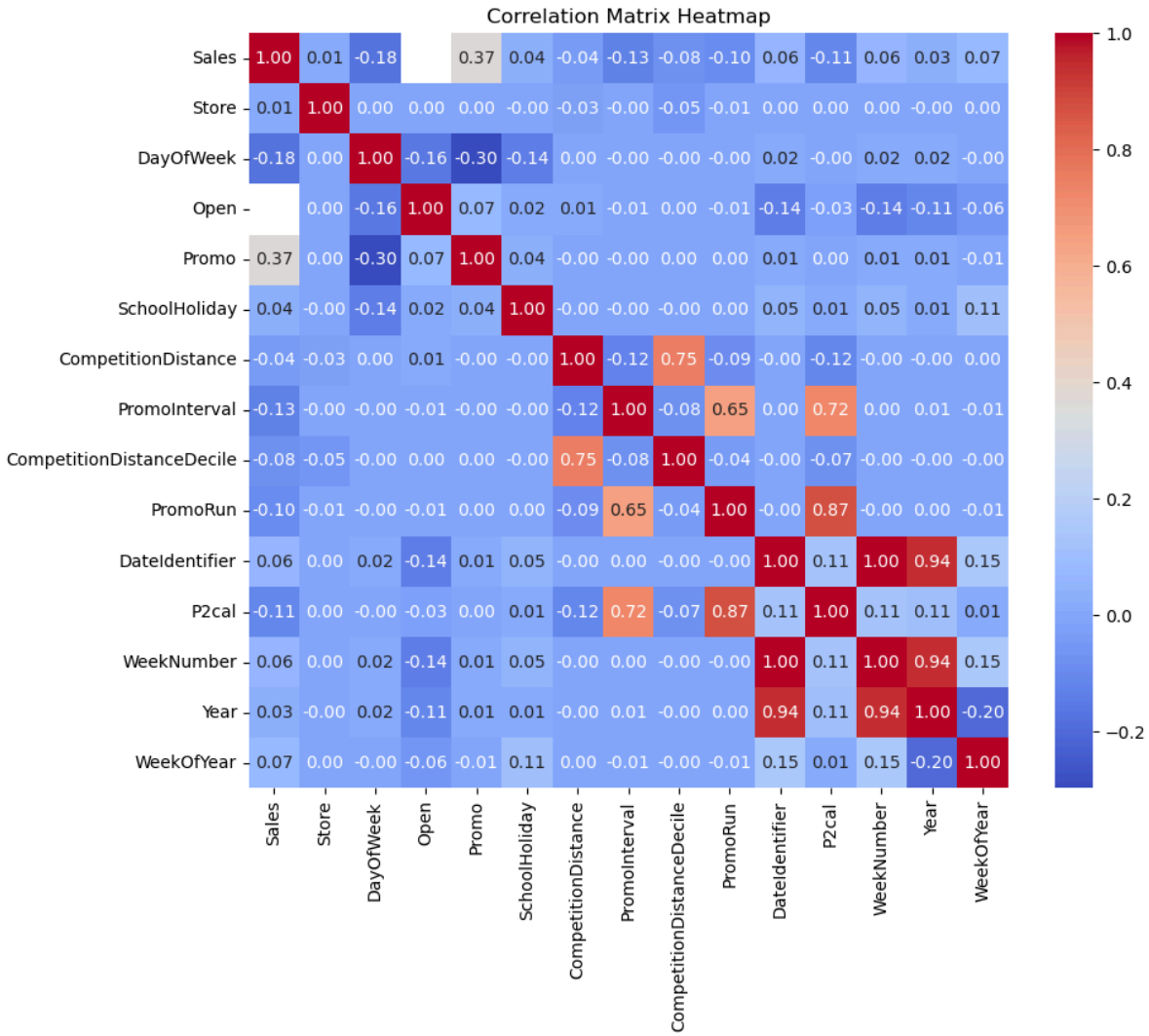
In [176…
```
import pandas as pd

# Assuming df is your DataFrame
numeric_dfzc = dfzc.select_dtypes(include=['number'])

# Calculate the correlation matrix
correlation_matrix = numeric_dfzc.corr()

numeric_dfzc.corr()
```

Out[176]:

	Sales	Store	DayOfWeek	Open	Promo	Schoo
Sales	1.000000	0.007723	-0.178753	NaN	0.368199	0
Store	0.007723	1.000000	0.000233	0.000784	0.000021	-0
DayOfWeek	-0.178753	0.000233	1.000000	-0.161987	-0.297459	-0
Open	NaN	0.000784	-0.161987	1.000000	0.073382	0
Promo	0.368199	0.000021	-0.297459	0.073382	1.000000	0
SchoolHoliday	0.038635	-0.000392	-0.144800	0.019688	0.036575	1
CompetitionDistance	-0.036401	-0.029640	0.004690	0.007342	-0.002074	-0
PromoInterval	-0.131638	-0.000124	-0.001479	-0.011772	-0.000173	-0
CompetitionDistanceDecile	-0.075098	-0.049341	-0.000626	0.001547	0.000307	-0
PromoRun	-0.098959	-0.009657	-0.002512	-0.010759	0.000173	0
DateIdentifier	0.062734	0.000135	0.015778	-0.136213	0.008914	0
P2cal	-0.114502	0.001870	-0.000315	-0.026481	0.000978	0
WeekNumber	0.062167	0.000134	0.015487	-0.135564	0.009777	
Year	0.033802	-0.000212	0.016003	-0.114673	0.011920	0
WeekOfYear	0.074203	0.001002	-0.001934	-0.057056	-0.006514	0

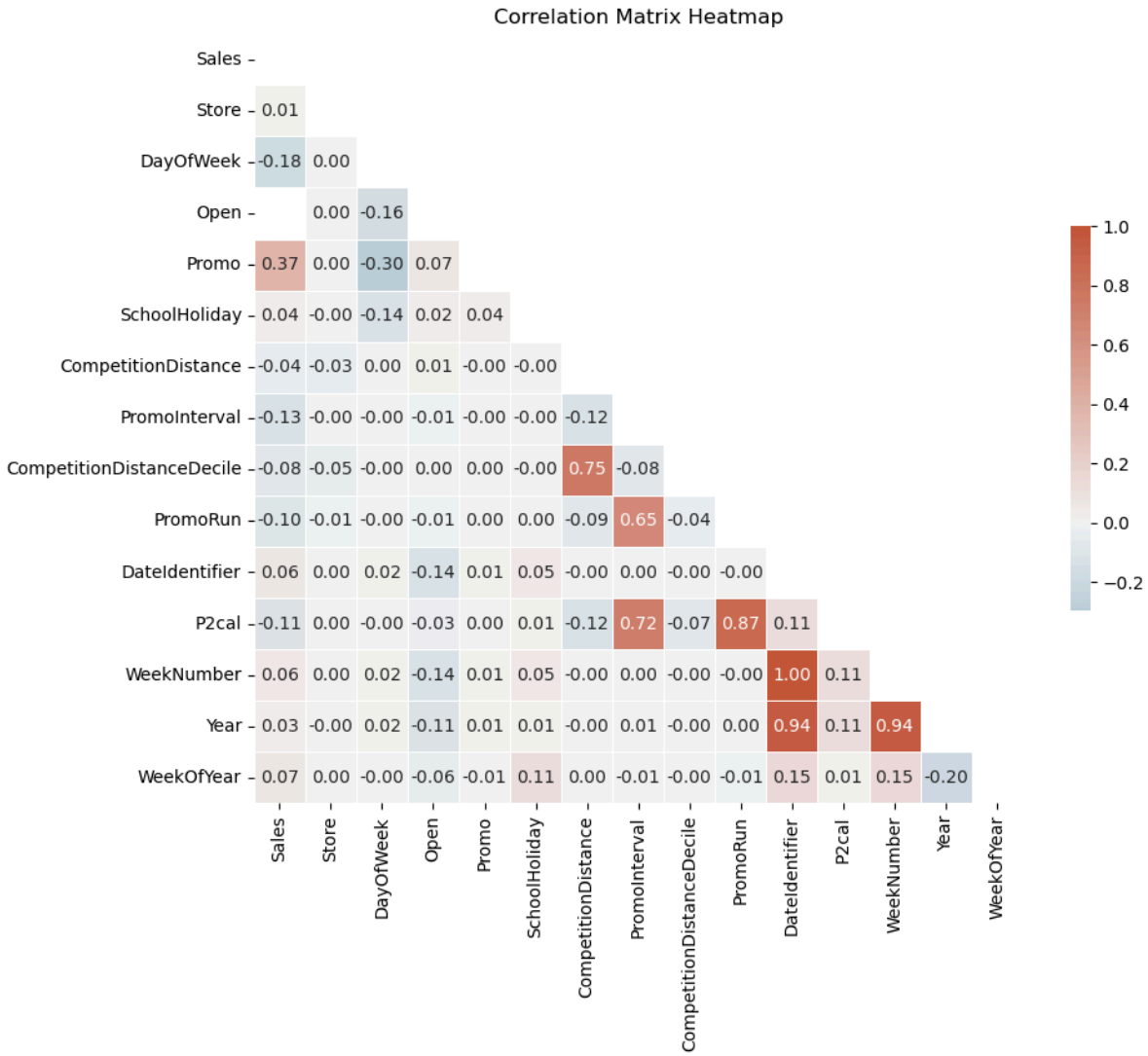
In [177…

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

# Assuming df is your DataFrame
numeric_dfzc = dfzc.select_dtypes(include=['number'])

# Calculate the correlation matrix
correlation_matrix = numeric_dfzc.corr()

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix Heatmap')
plt.show()
```



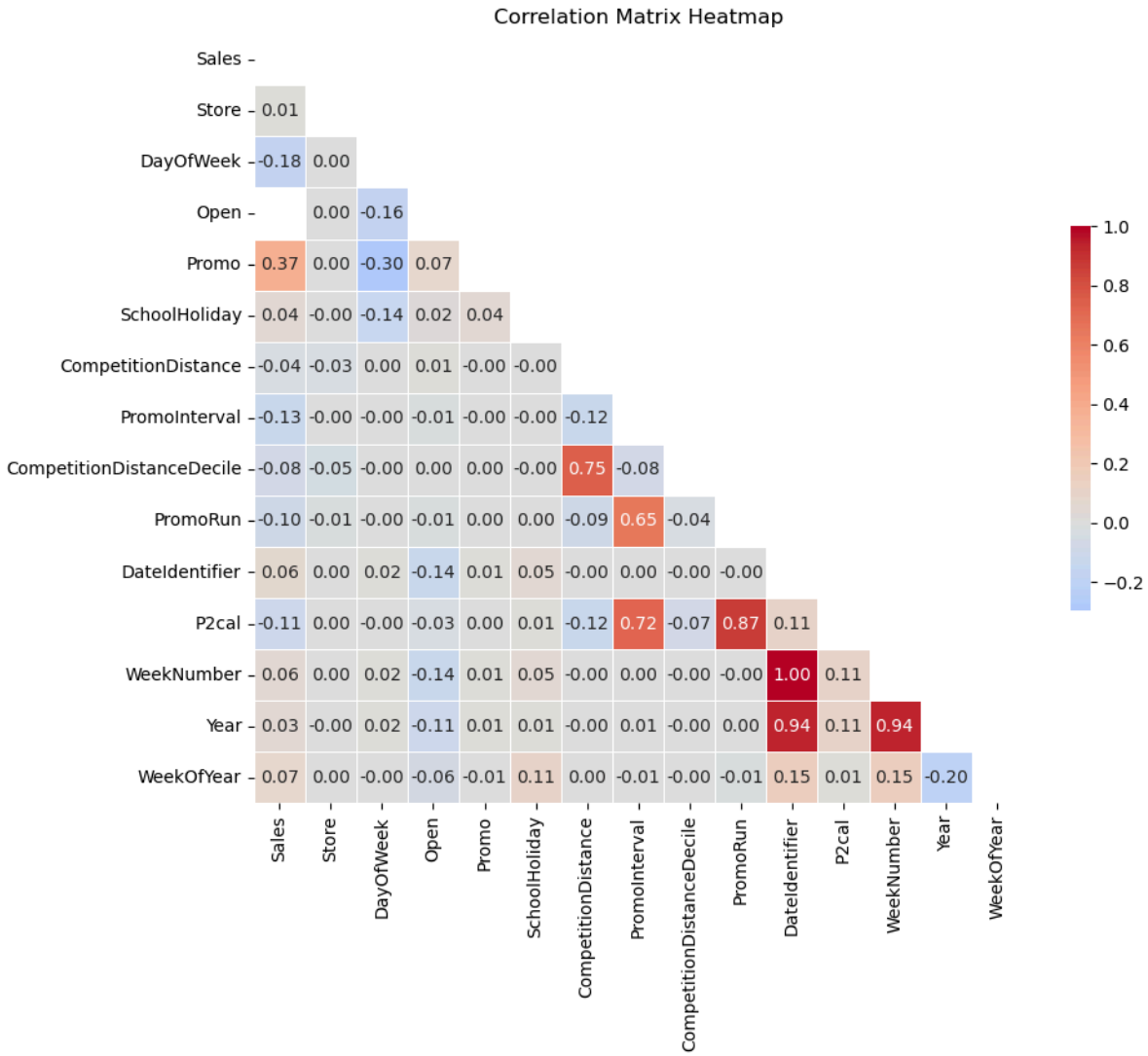
In [178…

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Assuming df is your DataFrame
numeric_dfzc = dfzc.select_dtypes(include=['number'])

# Calculate the correlation matrix
correlation_matrix = numeric_dfzc.corr()

# Create a mask to display only one half of the matrix
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))

# Set up the matplotlib figure
plt.figure(figsize=(10, 8))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(correlation_matrix, mask=mask, cmap=cmap, vmax=1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True,

plt.title('Correlation Matrix Heatmap')
plt.show()
```



In [179…

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Assuming df is your DataFrame
numeric_dfzc = dfzc.select_dtypes(include=['number'])

# Calculate the correlation matrix
correlation_matrix = numeric_dfzc.corr()

# Create a mask to display only one half of the matrix
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))

# Set up the matplotlib figure
plt.figure(figsize=(10, 8))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(correlation_matrix, mask=mask, cmap='coolwarm', vmax=1, center=0
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True,

plt.title('Correlation Matrix Heatmap')
plt.show()
```



In [180…

```
# Verify the removals
print(dfzc.head())
print(dfzc.dtypes)
```

```
   Sales  Store  DayOfWeek  Open  Promo StateHoliday  SchoolHoliday  \
0    NaN      1          4   1.0      1            d              0
1    NaN      1          3   1.0      1            d              0
2    NaN      1          2   1.0      1            d              0
3    NaN      1          1   1.0      1            d              0
4    NaN      1          7   0.0      0            d              0

   CompetitionDistance  PromoInterval  CompetitionDistanceDecile Type_Assor
t  \
0               1270.0            0.0                          4          C
a
1               1270.0            0.0                          4          C
a
2               1270.0            0.0                          4          C
a
3               1270.0            0.0                          4          C
a
4               1270.0            0.0                          4          C
a

   PromoRun  DateIdentifier  P2cal  WeekNumber  Year  WeekOfYear
0       0.0             990      0         142  2015          38
1       0.0             989      0         142  2015          38
2       0.0             988      0         142  2015          38
3       0.0             987      0         141  2015          37
4       0.0             986      0         141  2015          37
Sales                        float64
Store                          int64
DayOfWeek                      int64
Open                         float64
Promo                          int64
StateHoliday                  object
SchoolHoliday                  int64
CompetitionDistance          float64
PromoInterval                float64
CompetitionDistanceDecile      int64
Type_Assort                   object
PromoRun                     float64
DateIdentifier                 int64
P2cal                          int64
WeekNumber                     int64
Year                           int64
WeekOfYear                     int64
dtype: object
```

In [181…

```
# Step unique count
dfzc.nunique()
```

Out[181]:
```
Sales                        21733
Store                         1115
DayOfWeek                        7
Open                             2
Promo                            2
StateHoliday                     4
SchoolHoliday                    2
CompetitionDistance            655
PromoInterval                    4
CompetitionDistanceDecile       10
Type_Assort                      9
PromoRun                        56
DateIdentifier                 990
P2cal                            2
WeekNumber                     142
Year                             3
WeekOfYear                      52
dtype: int64
```

In [182…
```
from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Encode Type_Assort column
dfzc['Type_Assort'] = label_encoder.fit_transform(dfz['Type_Assort'])

# Check the mapping of original categories to encoded values
print("Mapping of original categories to encoded values:")
for category, encoded_value in zip(label_encoder.classes_, label_encoder.tra
    print(f"{category}: {encoded_value}")
```

```
Mapping of original categories to encoded values:
Aa: 0
Ac: 1
Ba: 2
Bb: 3
Bc: 4
Ca: 5
Cc: 6
Da: 7
Dc: 8
```

In [183…
```
from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Encode Type_Assort column
dfzc['StateHoliday'] = label_encoder.fit_transform(dfz['StateHoliday'])

# Check the mapping of original categories to encoded values
print("Mapping of original categories to encoded values:")
for category, encoded_value in zip(label_encoder.classes_, label_encoder.tra
    print(f"{category}: {encoded_value}")
```

```
Mapping of original categories to encoded values:
a: 0
b: 1
c: 2
d: 3
```

In [184…
```
# Step info
dfzc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 885426 entries, 0 to 1058295
Data columns (total 17 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   Sales                      844338 non-null  float64
 1   Store                      885426 non-null  int64
 2   DayOfWeek                  885426 non-null  int64
 3   Open                       885426 non-null  float64
 4   Promo                      885426 non-null  int64
 5   StateHoliday               885426 non-null  int64
 6   SchoolHoliday              885426 non-null  int64
 7   CompetitionDistance        885426 non-null  float64
 8   PromoInterval              885426 non-null  float64
 9   CompetitionDistanceDecile  885426 non-null  int64
 10  Type_Assort                885426 non-null  int64
 11  PromoRun                   885426 non-null  float64
 12  DateIdentifier             885426 non-null  int64
 13  P2cal                      885426 non-null  int64
 14  WeekNumber                 885426 non-null  int64
 15  Year                       885426 non-null  int64
 16  WeekOfYear                 885426 non-null  int64
dtypes: float64(5), int64(12)
memory usage: 121.6 MB
```

In [185…

```
dfzc["Open"].value_counts()
```

Out[185]:

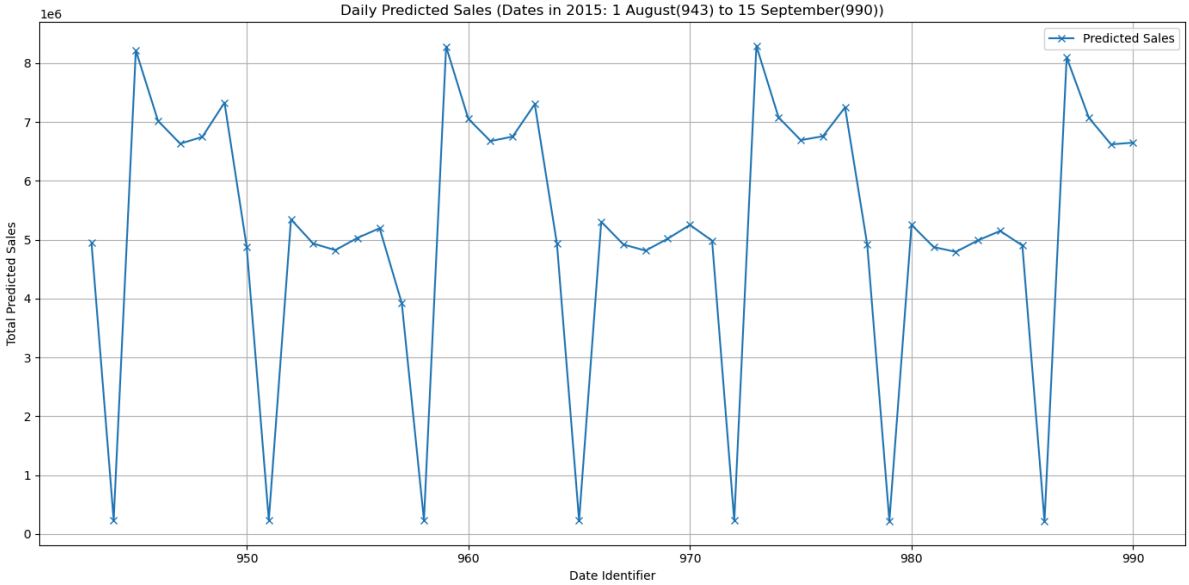
```
Open
1.0    879442
0.0      5984
Name: count, dtype: int64
```

In [275…

```
from sklearn.ensemble import RandomForestRegressor
# Remove rows where Open = 0
dfzc = dfzc[dfzc['Open'] != 0]

# Filter the DataFrame to include only DateIdentifiers 882-942 for training
train_df = dfzc[dfzc['DateIdentifier'] <= 942].dropna(subset=['Sales'])

# Filter the DataFrame to include only DateIdentifiers 943-990 for testing
test_df = dfzc[(dfzc['DateIdentifier'] >= 943) & (dfzc['DateIdentifier'] <=

# Features to use for prediction
features = ['Store', 'DayOfWeek', 'Promo', 'StateHoliday', 'SchoolHoliday',
            'CompetitionDistance', 'PromoInterval', 'CompetitionDistanceDec
            'Type_Assort', 'PromoRun', 'P2cal', 'WeekNumber', 'Year', 'Week(

# Separate features and target variable for training
X_train = train_df[features]
y_train = train_df['Sales']

# Fit the Random Forest model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict sales for the test set
X_test = test_df[features]
test_df['Predicted_Sales'] = rf_model.predict(X_test)

# Show feature importances in a table
feature_importances = pd.DataFrame({
    'Feature': features,
    'Importance': rf_model.feature_importances_
}).sort_values(by='Importance', ascending=False)
```

```
print("Feature Importances:")
print(feature_importances)

# Calculate total predicted sales for each day
daily_sales_prediction = test_df.groupby('DateIdentifier').agg({
    'Predicted_Sales': 'sum'
}).reset_index()

# Calculate total predicted sales for the entire prediction period
total_predicted_sales = daily_sales_prediction['Predicted_Sales'].sum()

# Plotting the predicted sales for each day
plt.figure(figsize=(14, 7))
plt.plot(daily_sales_prediction['DateIdentifier'], daily_sales_prediction['P
plt.xlabel('Date Identifier')
plt.ylabel('Total Predicted Sales')
plt.title('Daily Predicted Sales (Dates in 2015: 1 August(943) to 15 Septemb
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Print the total predicted sales for the entire prediction period
print("\nTotal Predicted Sales for the entire prediction period (DateIdentif
print(total_predicted_sales)

# Print the daily sales prediction table
print("\nDaily Predicted Sales for each DateIdentifier (943-990):")
print(daily_sales_prediction)
```

```
Feature Importances:
                      Feature  Importance
0                       Store    0.244465
5         CompetitionDistance    0.243527
2                       Promo    0.135518
9                     PromoRun    0.071999
8                  Type_Assort    0.071614
1                    DayOfWeek    0.071325
13                  WeekOfYear    0.065632
11                  WeekNumber    0.047363
7   CompetitionDistanceDecile    0.020601
6                PromoInterval    0.016632
4                SchoolHoliday    0.006086
12                        Year    0.002640
3                 StateHoliday    0.001649
10                       P2cal    0.000949
```


Daily Predicted Sales (Dates in 2015: 1 August(943) to 15 September(990))
1e6
Predicted Sales
Total Predicted Sales
Date Identifier
0
1
2
3
4
5
6
7
8
950
960
970
980
990

```
Total Predicted Sales for the entire prediction period (DateIdentifier 943-
990):
247306853.77

Daily Predicted Sales for each DateIdentifier (943-990):
    DateIdentifier  Predicted_Sales
0              943       4943651.01
1              944        237473.12
2              945       8216755.75
3              946       7020315.48
4              947       6631753.66
5              948       6747087.14
6              949       7329149.82
7              950       4882197.43
8              951        235757.68
9              952       5345355.40
10             953       4935322.07
11             954       4823067.33
12             955       5030460.83
13             956       5195136.97
14             957       3921599.06
15             958        234367.71
16             959       8281347.69
17             960       7055294.81
18             961       6675810.42
19             962       6751562.00
20             963       7303388.65
21             964       4930941.38
22             965        230743.05
23             966       5306343.67
24             967       4918921.10
25             968       4814472.44
26             969       5017131.33
27             970       5251235.52
28             971       4984132.69
29             972        229314.48
30             973       8292467.82
31             974       7078944.47
32             975       6691521.11
33             976       6757660.81
34             977       7248738.55
35             978       4918770.23
36             979        222552.25
37             980       5252496.54
38             981       4875075.38
39             982       4793543.58
40             983       4989567.90
41             984       5149026.18
42             985       4901239.01
43             986        215951.28
44             987       8097176.32
45             988       7073041.54
46             989       6620173.15
47             990       6648817.96
```

In [187…

```
import pandas as pd


# Filter DataFrame to include only rows with DateIdentifier between 1 and 94
dfzc942 = dfzc[(dfzc['DateIdentifier'] >= 1) & (dfzc['DateIdentifier'] <= 94

dfzc942.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 844338 entries, 48 to 1058295
Data columns (total 17 columns):
 #   Column                     Non-Null Count   Dtype  
---  ------                     --------------   -----  
 0   Sales                      844338 non-null  float64
 1   Store                      844338 non-null  int64  
 2   DayOfWeek                  844338 non-null  int64  
 3   Open                       844338 non-null  float64
 4   Promo                      844338 non-null  int64  
 5   StateHoliday               844338 non-null  int64  
 6   SchoolHoliday              844338 non-null  int64  
 7   CompetitionDistance        844338 non-null  float64
 8   PromoInterval              844338 non-null  float64
 9   CompetitionDistanceDecile  844338 non-null  int64  
 10  Type_Assort                844338 non-null  int64  
 11  PromoRun                   844338 non-null  float64
 12  DateIdentifier             844338 non-null  int64  
 13  P2cal                      844338 non-null  int64  
 14  WeekNumber                 844338 non-null  int64  
 15  Year                       844338 non-null  int64  
 16  WeekOfYear                 844338 non-null  int64  
dtypes: float64(5), int64(12)
memory usage: 116.0 MB
```

In [189…

```python
# Step add columns
df = dfzc942

# 1. Calculate average sales per store
df['AvgSalesPerStore'] = df.groupby('Store')['Sales'].transform('mean')

# 2. Rank stores based on their average sales, lowest average sales as rank
store_avg_sales = df.groupby('Store')['AvgSalesPerStore'].first()
store_avg_sales_ranked = store_avg_sales.rank(method='min')
df['StoreRank'] = df['Store'].map(store_avg_sales_ranked)

# 3. Calculate and normalize average sales for each variable
variables_to_normalize = [
    'DayOfWeek', 'Promo', 'StateHoliday', 'SchoolHoliday', 'CompetitionDista
    'PromoInterval', 'Type_Assort', 'PromoRun', 'P2cal', 'Year', 'WeekOfYea
]

for variable in variables_to_normalize:
    # Calculate average sales for each unique value of the variable
    avg_sales = df.groupby(variable)['Sales'].mean()
    # Normalize the average sales by dividing by the minimum average sales
    min_avg_sales = avg_sales.min()
    normalized_avg_sales = avg_sales / min_avg_sales
    # Create a new column in the DataFrame with these normalized values
    df[f'Index_{variable}'] = df[variable].map(normalized_avg_sales)

# Print the updated DataFrame
df.head(7).T
```

Out[189]:

	48	49	50	51	
Sales	5263.000000	5020.000000	4782.000000	5011.000000	61
Store	1.000000	1.000000	1.000000	1.000000	
DayOfWeek	5.000000	4.000000	3.000000	2.000000	
Open	1.000000	1.000000	1.000000	1.000000	
Promo	1.000000	1.000000	1.000000	1.000000	
StateHoliday	3.000000	3.000000	3.000000	3.000000	
SchoolHoliday	1.000000	1.000000	1.000000	1.000000	
CompetitionDistance	1270.000000	1270.000000	1270.000000	1270.000000	12
PromoInterval	0.000000	0.000000	0.000000	0.000000	
CompetitionDistanceDecile	4.000000	4.000000	4.000000	4.000000	
Type_Assort	5.000000	5.000000	5.000000	5.000000	
PromoRun	0.000000	0.000000	0.000000	0.000000	
DateIdentifier	942.000000	941.000000	940.000000	939.000000	9
P2cal	0.000000	0.000000	0.000000	0.000000	
WeekNumber	135.000000	135.000000	135.000000	135.000000	1
Year	2015.000000	2015.000000	2015.000000	2015.000000	20
WeekOfYear	31.000000	31.000000	31.000000	31.000000	
AvgSalesPerStore	4759.096031	4759.096031	4759.096031	4759.096031	47
StoreRank	155.000000	155.000000	155.000000	155.000000	1
Index_DayOfWeek	1.203903	1.152020	1.145309	1.206520	
Index_Promo	1.387686	1.387686	1.387686	1.387686	
Index_StateHoliday	1.000000	1.000000	1.000000	1.000000	
Index_SchoolHoliday	1.044004	1.044004	1.044004	1.044004	
Index_CompetitionDistance	3.087343	3.087343	3.087343	3.087343	
Index_CompetitionDistanceDecile	1.067718	1.067718	1.067718	1.067718	
Index_PromoInterval	1.182585	1.182585	1.182585	1.182585	
Index_Type_Assort	1.062482	1.062482	1.062482	1.062482	
Index_PromoRun	1.655080	1.655080	1.655080	1.655080	
Index_P2cal	1.109664	1.109664	1.109664	1.109664	
Index_Year	1.037487	1.037487	1.037487	1.037487	
Index_WeekOfYear	1.319977	1.319977	1.319977	1.319977	

In [191…

```
# Get the top 10 rows with the largest "Sales"
top_10_sales = dfzc942.nlargest(10, 'Sales')

top_10_sales.T
```

Out[191]:

	667283	190693	190665	19065
Sales	41551.000000	38722.000000	38484.000000	38367.00000
Store	909.000000	262.000000	262.000000	262.00000
DayOfWeek	1.000000	5.000000	5.000000	4.00000
Open	1.000000	1.000000	1.000000	1.00000
Promo	0.000000	1.000000	1.000000	0.00000
StateHoliday	3.000000	1.000000	0.000000	0.00000
SchoolHoliday	0.000000	0.000000	0.000000	1.00000
CompetitionDistance	1680.000000	1180.000000	1180.000000	1180.00000
PromoInterval	2.000000	0.000000	0.000000	0.00000
CompetitionDistanceDecile	5.000000	4.000000	4.000000	4.00000
Type_Assort	1.000000	2.000000	2.000000	2.00000
PromoRun	299.000000	0.000000	0.000000	0.00000
DateIdentifier	903.000000	823.000000	851.000000	864.00000
P2cal	1.000000	0.000000	0.000000	0.00000
WeekNumber	129.000000	118.000000	122.000000	124.00000
Year	2015.000000	2015.000000	2015.000000	2015.00000
WeekOfYear	25.000000	14.000000	18.000000	20.00000
AvgSalesPerStore	10728.983526	20718.515924	20718.515924	20718.51592
StoreRank	1049.000000	1114.000000	1114.000000	1114.00000
Index_DayOfWeek	1.398491	1.203903	1.203903	1.15202
Index_Promo	1.000000	1.387686	1.387686	1.00000
Index_StateHoliday	1.000000	1.421908	1.220523	1.22052
Index_SchoolHoliday	1.000000	1.000000	1.000000	1.04400
Index_CompetitionDistance	2.765446	5.400135	5.400135	5.40013
Index_CompetitionDistanceDecile	1.022505	1.067718	1.067718	1.06771
Index_PromoInterval	1.034022	1.182585	1.182585	1.18258
Index_Type_Assort	1.177201	1.730559	1.730559	1.73055
Index_PromoRun	1.411736	1.655080	1.655080	1.65508
Index_P2cal	1.000000	1.109664	1.109664	1.10966
Index_Year	1.037487	1.037487	1.037487	1.03748
Index_WeekOfYear	1.170234	1.254170	1.419287	1.13198

In [193…

```
# Step unique count
df.nunique()
```

Out[193]:
```
Sales                                21733
Store                                 1115
DayOfWeek                                7
Open                                     1
Promo                                    2
StateHoliday                             4
SchoolHoliday                            2
CompetitionDistance                    655
PromoInterval                            4
CompetitionDistanceDecile               10
Type_Assort                              9
PromoRun                                56
DateIdentifier                         942
P2cal                                    2
WeekNumber                             135
Year                                     3
WeekOfYear                              52
AvgSalesPerStore                      1115
StoreRank                             1115
Index_DayOfWeek                          7
Index_Promo                              2
Index_StateHoliday                       4
Index_SchoolHoliday                      2
Index_CompetitionDistance              655
Index_CompetitionDistanceDecile         10
Index_PromoInterval                      4
Index_Type_Assort                        9
Index_PromoRun                          56
Index_P2cal                              2
Index_Year                               3
Index_WeekOfYear                        52
dtype: int64
```

In [195…
```
# Group by 'DayOfWeek' and 'NormalizedAvgSales_DayOfWeek' and aggregate sale
grouped_df = df.groupby(['DayOfWeek', 'Index_DayOfWeek']).agg({
    'Sales': 'sum',
    'Store': 'count'
}).reset_index()

# Print the grouped DataFrame
print(grouped_df)
```

```
   DayOfWeek  Index_DayOfWeek         Sales   Store
0          1         1.398491  1.130203e+09  137557
1          2         1.206520  1.020412e+09  143955
2          3         1.145309  9.549629e+08  141922
3          4         1.152020  9.111777e+08  134626
4          5         1.203903  9.805559e+08  138633
5          6         1.000000  8.463177e+08  144052
6          7         1.399933  2.955143e+07    3593
```

In [197…
```
# Group by 'DayOfWeek' and 'NormalizedAvgSales_DayOfWeek' and aggregate sale
grouped_df = df.groupby(['Promo', 'Index_Promo']).agg({
    'Sales': 'sum',
    'Store': 'count'
}).reset_index()

# Print the grouped DataFrame
print(grouped_df)
```

```
   Promo  Index_Promo         Sales   Store
0      0     1.000000  2.771974e+09  467463
1      1     1.387686  3.101206e+09  376875
```

In [199…
```
# Group by 'DayOfWeek' and 'NormalizedAvgSales_DayOfWeek' and aggregate sale
grouped_df = df.groupby(['StateHoliday', 'Index_StateHoliday']).agg({
    'Sales': 'sum',
    'Store': 'count'
}).reset_index()

# Print the grouped DataFrame
print(grouped_df)
```

```
   StateHoliday  Index_StateHoliday         Sales   Store
0             0            1.220523  5.890305e+06     694
1             1            1.421908  1.433744e+06     145
2             2            1.401179  6.918060e+05      71
3             3            1.000000  5.865165e+09  843428
```

In [201…
```
# Group by 'DayOfWeek' and 'NormalizedAvgSales_DayOfWeek' and aggregate sale
grouped_df = df.groupby(['SchoolHoliday', 'Index_SchoolHoliday']).agg({
    'Sales': 'sum',
    'Store': 'count'
}).reset_index()

# Print the grouped DataFrame
print(grouped_df)
```

```
   SchoolHoliday  Index_SchoolHoliday         Sales   Store
0              0             1.000000  4.696261e+09  680893
1              1             1.044004  1.176920e+09  163445
```

In [203…
```
# Group by 'DayOfWeek' and 'NormalizedAvgSales_DayOfWeek' and aggregate sale
grouped_df = df.groupby(['CompetitionDistanceDecile', 'Index_CompetitionDist
    'Sales': 'sum',
    'Store': 'count'
}).reset_index()

# Print the grouped DataFrame
print(grouped_df)
```

```
   CompetitionDistanceDecile  Index_CompetitionDistanceDecile        Sales
\
0                          1                         1.208441  753551816.0
1                          2                         1.069891  554244270.0
2                          3                         1.000000  543213671.0
3                          4                         1.067718  598114948.0
4                          5                         1.022505  569653722.0
5                          6                         1.007164  548792329.0
6                          7                         1.029301  578120573.0
7                          8                         1.021911  568355358.0
8                          9                         1.022288  568727384.0
9                         10                         1.038881  590406552.0

   Store
0  94196
1  78254
2  82057
3  84620
4  84157
5  82310
6  84844
7  84014
8  84038
9  85848
```

In [205…
```
# Group by 'DayOfWeek' and 'NormalizedAvgSales_DayOfWeek' and aggregate sale
grouped_df = df.groupby(['Type_Assort', 'Index_Type_Assort']).agg({
```

```
    'Sales': 'sum',
    'Store': 'count'
}).reset_index()

# Print the grouped DataFrame
print(grouped_df)
```

```
   Type_Assort  Index_Type_Assort         Sales   Store
0            0           1.016232  1.870105e+09  286028
1            1           1.177201  1.295230e+09  171014
2            2           1.730559  7.135776e+07    6409
3            3           1.343306  7.094631e+07    8209
4            4           2.793012  1.692732e+07     942
5            5           1.062482  4.002604e+08   58554
6            6           1.093904  3.829610e+08   54414
7            7           1.000000  6.040268e+08   93884
8            8           1.094778  1.161366e+09  164884
```

In [207…
```
# Group by 'DayOfWeek' and 'NormalizedAvgSales_DayOfWeek' and aggregate sale
grouped_df = df.groupby(['PromoInterval', 'Index_PromoInterval']).agg({
    'Sales': 'sum',
    'Store': 'count'
}).reset_index()

# Print the grouped DataFrame
print(grouped_df)
```

```
   PromoInterval  Index_PromoInterval         Sales   Store
0            0.0             1.182585  3.111543e+09  423292
1            1.0             1.082125  1.630451e+09  242397
2            2.0             1.034022  6.298691e+08   97998
3            3.0             1.000000  5.013176e+08   80651
```

In [209…
```
# Group by 'DayOfWeek' and 'NormalizedAvgSales_DayOfWeek' and aggregate sale
grouped_df = df.groupby(['P2cal', 'Index_P2cal']).agg({
    'Sales': 'sum',
    'Store': 'count'
}).reset_index()

# Print the grouped DataFrame
print(grouped_df)
```

```
   P2cal  Index_P2cal         Sales   Store
0      0     1.109664  3.498041e+09  481529
1      1     1.000000  2.375139e+09  362809
```

In [211…
```
# Group by 'DayOfWeek' and 'NormalizedAvgSales_DayOfWeek' and aggregate sale
grouped_df = df.groupby(['Year', 'Index_Year']).agg({
    'Sales': 'sum',
    'Store': 'count'
}).reset_index()

# Print the grouped DataFrame
print(grouped_df)
```

```
   Year  Index_Year         Sales   Store
0  2013    1.000000  2.298341e+09  336809
1  2014    1.028825  2.173778e+09  309630
2  2015    1.037487  1.401061e+09  197899
```

In [213…
```
# Group by 'DayOfWeek' and 'NormalizedAvgSales_DayOfWeek' and aggregate sale
grouped_df = df.groupby(['WeekOfYear', 'Index_WeekOfYear']).agg({
    'Sales': 'sum',
    'Store': 'count'
}).reset_index()
```

```
# Print the grouped DataFrame
print(grouped_df)
```

```
    WeekOfYear  Index_WeekOfYear        Sales  Store
0            1          1.113700  107722091.0  16141
1            2          1.181268  139804642.0  19750
2            3          1.029845  123667041.0  20039
3            4          1.050478  126195126.0  20047
4            5          1.158938  139349564.0  20065
5            6          1.152543  138359606.0  20033
6            7          1.051745  126328395.0  20044
7            8          1.097463  132082764.0  20084
8            9          1.098584  132072914.0  20062
9           10          1.171106  140896773.0  20077
10          11          1.040587  125212627.0  20080
11          12          1.162751  140024066.0  20096
12          13          1.197630  128693792.0  17932
13          14          1.254170  134814540.0  17938
14          15          1.109355  133759952.0  20121
15          16          1.179762  126865652.0  17945
16          17          1.081536  130425195.0  20124
17          18          1.419287  143335455.0  16853
18          19          1.197164  136635562.0  19046
19          20          1.131985  121938240.0  17976
20          21          1.118192  127662475.0  19052
21          22          1.286842  141650134.0  18369
22          23          1.228131  135106478.0  18358
23          24          1.095807  132172371.0  20128
24          25          1.170234  136374002.0  19447
25          26          1.101268  132765126.0  20118
26          27          1.224569  139675289.0  19034
27          28          1.085657  123811380.0  19031
28          29          1.174652  133840948.0  19014
29          30          1.039190  118505909.0  19030
30          31          1.319977  132752275.0  16783
31          32          1.105201   81686676.0  12334
32          33          1.108906   81023585.0  12193
33          34          1.060114   78341600.0  12332
34          35          1.148857   84768793.0  12313
35          36          1.098933   80979769.0  12297
36          37          1.101053   81300912.0  12322
37          38          1.021678   75550191.0  12340
38          39          1.067250   79022365.0  12356
39          40          1.293271   80490375.0  10386
40          41          1.146741   84935666.0  12360
41          42          1.000000   74072971.0  12361
42          43          1.060079   78523205.0  12361
43          44          1.186894   78535491.0  11042
44          45          1.233433   91164468.0  12334
45          46          1.102189   81457489.0  12333
46          47          1.137499   83037786.0  12182
47          48          1.373586  101523386.0  12334
48          49          1.401277  103570004.0  12334
49          50          1.328190   98160156.0  12333
50          51          1.793053  132752284.0  12355
51          52          1.199226   59783067.0   8319
```

In [215…

```
# Group by 'WeekOfYear' and 'NormalizedAvgSales_WeekOfYear' and aggregate sa
grouped_df = df.groupby(['WeekOfYear', 'Index_WeekOfYear']).agg({
    'Sales': 'sum',
    'Store': 'count'
}).reset_index()
```

```
# Sort the grouped DataFrame by 'NormalizedAvgSales_WeekOfYear' in descendi
grouped_df = grouped_df.sort_values(by='Index_WeekOfYear', ascending=False)

# Print the sorted grouped DataFrame
print(grouped_df)
```

	WeekOfYear	Index_WeekOfYear	Sales	Store
50	51	1.793053	132752284.0	12355
17	18	1.419287	143335455.0	16853
48	49	1.401277	103570004.0	12334
47	48	1.373586	101523386.0	12334
49	50	1.328190	98160156.0	12333
30	31	1.319977	132752275.0	16783
39	40	1.293271	80490375.0	10386
21	22	1.286842	141650134.0	18369
13	14	1.254170	134814540.0	17938
44	45	1.233433	91164468.0	12334
22	23	1.228131	135106478.0	18358
26	27	1.224569	139675289.0	19034
51	52	1.199226	59783067.0	8319
12	13	1.197630	128693792.0	17932
18	19	1.197164	136635562.0	19046
43	44	1.186894	78535491.0	11042
1	2	1.181268	139804642.0	19750
15	16	1.179762	126865652.0	17945
28	29	1.174652	133840948.0	19014
9	10	1.171106	140896773.0	20077
24	25	1.170234	136374002.0	19447
11	12	1.162751	140024066.0	20096
4	5	1.158938	139349564.0	20065
5	6	1.152543	138359606.0	20033
34	35	1.148857	84768793.0	12313
40	41	1.146741	84935666.0	12360
46	47	1.137499	83037786.0	12182
19	20	1.131985	121938240.0	17976
20	21	1.118192	127662475.0	19052
0	1	1.113700	107722091.0	16141
14	15	1.109355	133759952.0	20121
32	33	1.108906	81023585.0	12193
31	32	1.105201	81686676.0	12334
45	46	1.102189	81457489.0	12333
25	26	1.101268	132765126.0	20118
36	37	1.101053	81300912.0	12322
35	36	1.098933	80979769.0	12297
8	9	1.098584	132072914.0	20062
7	8	1.097463	132082764.0	20084
23	24	1.095807	132172371.0	20128
27	28	1.085657	123811380.0	19031
16	17	1.081536	130425195.0	20124
38	39	1.067250	79022365.0	12356
33	34	1.060114	78341600.0	12332
42	43	1.060079	78523205.0	12361
6	7	1.051745	126328395.0	20044
3	4	1.050478	126195126.0	20047
10	11	1.040587	125212627.0	20080
29	30	1.039190	118505909.0	19030
2	3	1.029845	123667041.0	20039
37	38	1.021678	75550191.0	12340
41	42	1.000000	74072971.0	12361

In [217…

```
# Step describe
df.describe().T
```

Out[217]:

	count	mean	std	min	
Sales	844338.0	6955.959134	3103.815515	46.000000	4859.
Store	844338.0	558.421374	321.730861	1.000000	280.
DayOfWeek	844338.0	3.520350	1.723712	1.000000	2.
Open	844338.0	1.000000	0.000000	1.000000	1.
Promo	844338.0	0.446356	0.497114	0.000000	0.
StateHoliday	844338.0	2.997107	0.090334	0.000000	3.
SchoolHoliday	844338.0	0.193578	0.395102	0.000000	0.
CompetitionDistance	844338.0	5458.019004	7799.457551	20.000000	710.
PromoInterval	844338.0	0.805774	0.980689	0.000000	0.
CompetitionDistanceDecile	844338.0	5.484583	2.898723	1.000000	3.
Type_Assort	844338.0	3.325376	3.358630	0.000000	0.
PromoRun	844338.0	88.099685	106.493952	0.000000	0.
DateIdentifier	844338.0	466.047670	274.494262	1.000000	228.
P2cal	844338.0	0.429696	0.495033	0.000000	0.
WeekNumber	844338.0	67.055272	39.215345	1.000000	33.
Year	844338.0	2013.835481	0.778602	2013.000000	2013.
WeekOfYear	844338.0	23.610282	14.410994	1.000000	11.
AvgSalesPerStore	844338.0	6955.959134	2408.370239	2703.736573	5323.
StoreRank	844338.0	560.454461	322.504175	1.000000	280.
Index_DayOfWeek	844338.0	1.183976	0.118474	1.000000	1.
Index_Promo	844338.0	1.173046	0.192724	1.000000	1.
Index_StateHoliday	844338.0	1.000287	0.009165	1.000000	1.
Index_SchoolHoliday	844338.0	1.008518	0.017386	1.000000	1.
Index_CompetitionDistance	844338.0	2.572721	0.640911	1.000000	2.
Index_CompetitionDistanceDecile	844338.0	1.050756	0.059799	1.000000	1
Index_PromoInterval	844338.0	1.119061	0.067883	1.000000	1.
Index_Type_Assort	844338.0	1.081166	0.106010	1.000000	1.
Index_PromoRun	844338.0	1.566175	0.153203	1.000000	1.
Index_P2cal	844338.0	1.062542	0.054287	1.000000	1.
Index_Year	844338.0	1.019357	0.016105	1.000000	1.
Index_WeekOfYear	844338.0	1.160783	0.120566	1.000000	1.

In [219…

```
# Assuming df is your DataFrame
numeric_df = df.select_dtypes(include=['number'])

# Calculate the correlation matrix
correlation_matrix = numeric_df.corr()

numeric_df.corr()
```

Out[219]:

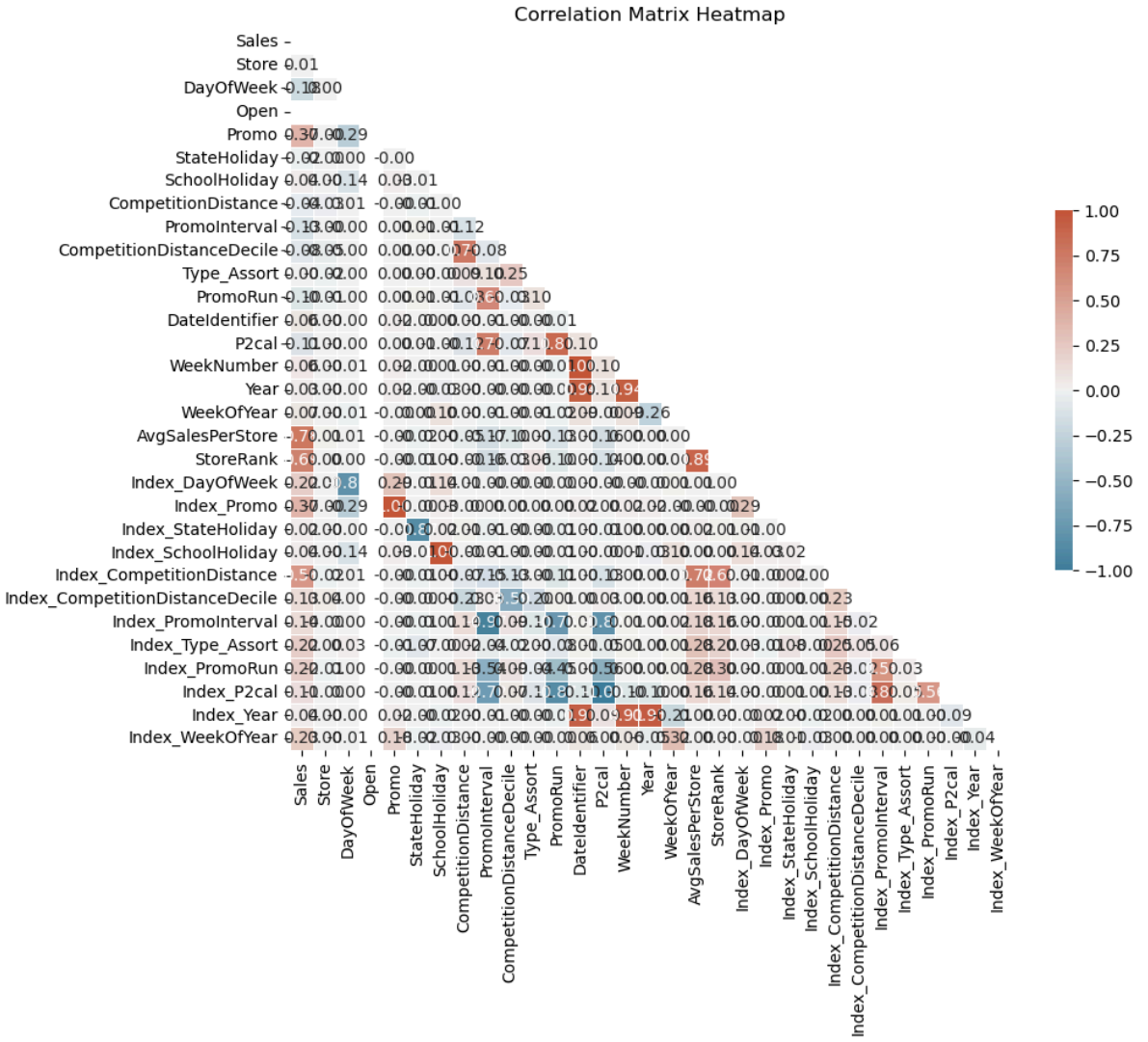
	Sales	Store	DayOfWeek	Open	Promo	Sta
Sales	1.000000	0.007723	-0.178753	NaN	0.368199	
Store	0.007723	1.000000	0.000343	NaN	-0.000015	-
DayOfWeek	-0.178753	0.000343	1.000000	NaN	-0.289268	
Open	NaN	NaN	NaN	NaN	NaN	
Promo	0.368199	-0.000015	-0.289268	NaN	1.000000	-
StateHoliday	-0.017897	-0.002697	0.001424	NaN	-0.004419	
SchoolHoliday	0.038635	0.000526	-0.139319	NaN	0.028971	-
CompetitionDistance	-0.036401	-0.027027	0.005535	NaN	-0.002389	
PromoInterval	-0.131638	-0.001496	-0.003161	NaN	0.000427	
CompetitionDistanceDecile	-0.075098	-0.046866	-0.000530	NaN	0.000270	
Type_Assort	0.000698	-0.016799	-0.001812	NaN	0.000337	
PromoRun	-0.098959	-0.010011	-0.004028	NaN	0.000703	
DateIdentifier	0.062734	0.000812	-0.004828	NaN	0.018360	-
P2cal	-0.114502	0.000647	-0.004099	NaN	0.002465	
WeekNumber	0.062167	0.000810	-0.005033	NaN	0.019213	-
Year	0.033802	0.000316	-0.001085	NaN	0.019872	-
WeekOfYear	0.074203	0.001315	-0.010648	NaN	-0.003548	
AvgSalesPerStore	0.775939	0.009953	0.006507	NaN	-0.002656	
StoreRank	0.692339	0.000534	0.003985	NaN	-0.001471	
Index_DayOfWeek	0.224254	0.000448	-0.797102	NaN	0.290163	-
Index_Promo	0.368199	-0.000015	-0.289268	NaN	1.000000	-
Index_StateHoliday	0.020533	0.002218	-0.003556	NaN	-0.000550	
Index_SchoolHoliday	0.038635	0.000526	-0.139319	NaN	0.028971	-
Index_CompetitionDistance	0.558298	-0.021343	0.005744	NaN	-0.002377	
Index_CompetitionDistanceDecile	0.127542	0.041671	0.001260	NaN	-0.000306	-
Index_PromoInterval	0.135946	-0.002753	0.003372	NaN	-0.000139	-
Index_Type_Assort	0.219744	0.001491	0.027322	NaN	-0.012524	-
Index_PromoRun	0.219224	-0.010809	0.001720	NaN	-0.000015	-
Index_P2cal	0.114502	-0.000647	0.004099	NaN	-0.002465	-
Index_Year	0.035408	0.000767	-0.001898	NaN	0.018310	-
Index_WeekOfYear	0.232775	0.000289	-0.005167	NaN	0.179334	

31 rows × 31 columns

In [220…

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Assuming df is your DataFrame
numeric_df = df.select_dtypes(include=['number'])
```

```
# Calculate the correlation matrix
correlation_matrix = numeric_df.corr()

# Create a mask to display only one half of the matrix
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))

# Set up the matplotlib figure
plt.figure(figsize=(10, 8))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(correlation_matrix, mask=mask, cmap=cmap, vmax=1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True,

plt.title('Correlation Matrix Heatmap')
plt.show()
```



In [221…

```
import pandas as pd

# Assuming dfzc942 is your DataFrame
dft = dfzc942

# Filter the DataFrame to include only the first 881 days
dft_first_881_days = dft[dft['DateIdentifier'] <= 881]

# 1. Calculate average sales per store using only the first 881 days
store_avg_sales = dft_first_881_days.groupby('Store')['Sales'].mean()
dft['AvgSalesPerStore'] = dft['Store'].map(store_avg_sales)
```

```
# 2. Rank stores based on their average sales from the first 881 days, lowes
store_avg_sales_ranked = store_avg_sales.rank(method='min')
dft['StoreRank'] = dft['Store'].map(store_avg_sales_ranked)

# 3. Calculate and normalize average sales for each variable using only the
variables_to_normalize = [
    'DayOfWeek', 'Promo', 'StateHoliday', 'SchoolHoliday', 'CompetitionDista
    'PromoInterval', 'Type_Assort', 'PromoRun', 'P2cal', 'Year', 'WeekOfYear
]

for variable in variables_to_normalize:
    # Calculate average sales for each unique value of the variable using or
    avg_sales = dft_first_881_days.groupby(variable)['Sales'].mean()
    # Normalize the average sales by dividing by the minimum average sales
    min_avg_sales = avg_sales.min()
    normalized_avg_sales = avg_sales / min_avg_sales
    # Create a new column in the DataFrame with these normalized values
    dft[f'Index_{variable}'] = dft[variable].map(normalized_avg_sales)

# Print the updated DataFrame
dft.head(7).T
```

Out[221]:

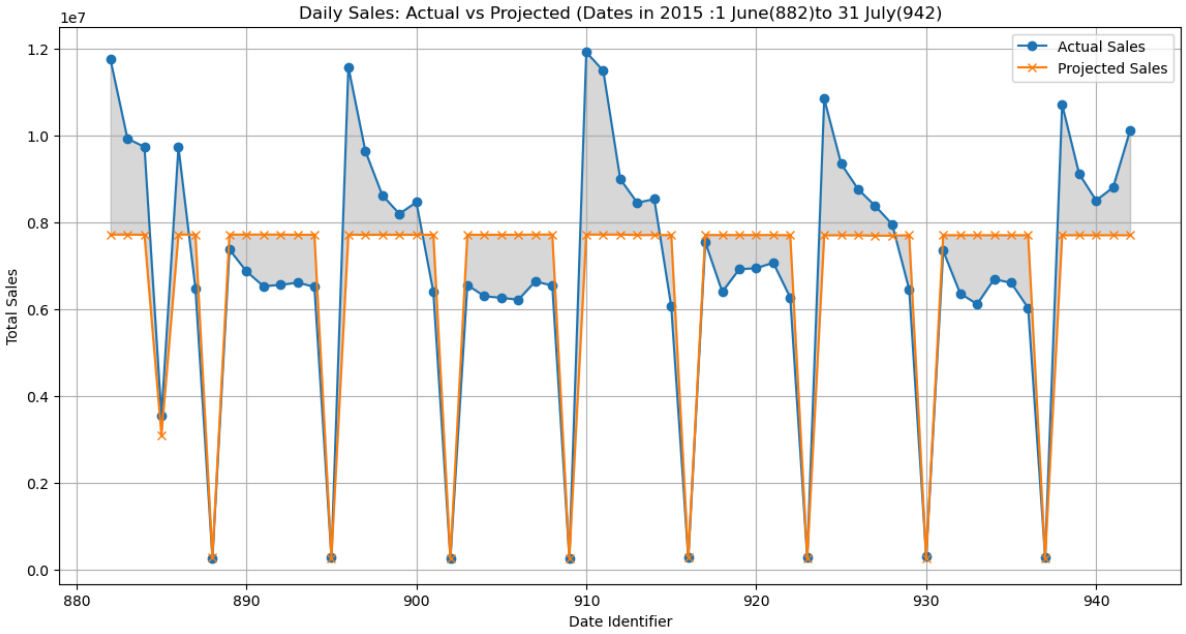
	48	49	50	51	
Sales	5263.000000	5020.000000	4782.000000	5011.000000	61
Store	1.000000	1.000000	1.000000	1.000000	
DayOfWeek	5.000000	4.000000	3.000000	2.000000	
Open	1.000000	1.000000	1.000000	1.000000	
Promo	1.000000	1.000000	1.000000	1.000000	
StateHoliday	3.000000	3.000000	3.000000	3.000000	
SchoolHoliday	1.000000	1.000000	1.000000	1.000000	
CompetitionDistance	1270.000000	1270.000000	1270.000000	1270.000000	12
PromoInterval	0.000000	0.000000	0.000000	0.000000	
CompetitionDistanceDecile	4.000000	4.000000	4.000000	4.000000	
Type_Assort	5.000000	5.000000	5.000000	5.000000	
PromoRun	0.000000	0.000000	0.000000	0.000000	
DateIdentifier	942.000000	941.000000	940.000000	939.000000	9
P2cal	0.000000	0.000000	0.000000	0.000000	
WeekNumber	135.000000	135.000000	135.000000	135.000000	1
Year	2015.000000	2015.000000	2015.000000	2015.000000	20
WeekOfYear	31.000000	31.000000	31.000000	31.000000	
AvgSalesPerStore	4781.312757	4781.312757	4781.312757	4781.312757	4
StoreRank	164.000000	164.000000	164.000000	164.000000	1
Index_DayOfWeek	1.200379	1.148938	1.139140	1.198516	
Index_Promo	1.383223	1.383223	1.383223	1.383223	
Index_StateHoliday	1.000000	1.000000	1.000000	1.000000	
Index_SchoolHoliday	1.038444	1.038444	1.038444	1.038444	
Index_CompetitionDistance	3.078043	3.078043	3.078043	3.078043	
Index_CompetitionDistanceDecile	1.067662	1.067662	1.067662	1.067662	
Index_PromoInterval	1.182367	1.182367	1.182367	1.182367	
Index_Type_Assort	1.064799	1.064799	1.064799	1.064799	
Index_PromoRun	1.672982	1.672982	1.672982	1.672982	
Index_P2cal	1.110357	1.110357	1.110357	1.110357	
Index_Year	1.032501	1.032501	1.032501	1.032501	
Index_WeekOfYear	1.302267	1.302267	1.302267	1.302267	

In [225…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score

# Filter the DataFrame to include only DateIdentifiers 882-942
df_predict = dft[(dft['DateIdentifier'] >= 882) & (dft['DateIdentifier'] <=

# Predict sales based on AvgSalesPerStore
```

```
df_predict['Predicted_Sales'] = df_predict['AvgSalesPerStore']

# Create a DataFrame to compare actual and predicted sales
comparison_df = df_predict[['Store', 'DateIdentifier', 'Sales', 'Predicted_S
comparison_df['Difference'] = comparison_df['Sales'] - comparison_df['Predic

# Ensure no NaN values in actual and predicted sales
comparison_df['Sales'].fillna(0, inplace=True)
comparison_df['Predicted_Sales'].fillna(0, inplace=True)

# Calculate evaluation metrics
actual_sales = comparison_df['Sales']
predicted_sales = comparison_df['Predicted_Sales']
mse = mean_squared_error(actual_sales, predicted_sales)
r2 = r2_score(actual_sales, predicted_sales)
rmspe = np.sqrt(np.mean(((actual_sales - predicted_sales) / actual_sales.rep

print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Root Mean Squared Percentage Error (RMSPE):", rmspe)

# Show the first few rows of the comparison
print("Comparison of Actual and Predicted Sales:")
print(comparison_df.head())

# Calculate total sales per day (projected vs actual)
daily_sales_comparison = comparison_df.groupby('DateIdentifier').agg({
    'Sales': 'sum',
    'Predicted_Sales': 'sum'
}).reset_index()
daily_sales_comparison['Difference'] = daily_sales_comparison['Sales'] - dai

print("\nDaily Sales Comparison (Actual vs Projected):")
print(daily_sales_comparison.head())

# Plotting daily sales comparison
plt.figure(figsize=(14, 7))
plt.plot(daily_sales_comparison['DateIdentifier'], daily_sales_comparison['S
plt.plot(daily_sales_comparison['DateIdentifier'], daily_sales_comparison['P
plt.fill_between(daily_sales_comparison['DateIdentifier'], daily_sales_compa
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Daily Sales: Actual vs Projected (Dates in 2015 :1 June(882)to 31
plt.legend()
plt.grid(True)
```

```
Mean Squared Error: 3918601.8070568186
R-squared: 0.5951536370125072
Root Mean Squared Percentage Error (RMSPE): 0.3177693042338759
Comparison of Actual and Predicted Sales:
    Store  DateIdentifier   Sales  Predicted_Sales   Difference
48      1             942  5263.0      4781.312757   481.687243
49      1             941  5020.0      4781.312757   238.687243
50      1             940  4782.0      4781.312757     0.687243
51      1             939  5011.0      4781.312757   229.687243
52      1             938  6102.0      4781.312757  1320.687243

Daily Sales Comparison (Actual vs Projected):
   DateIdentifier       Sales  Predicted_Sales    Difference
0             882  11743615.0     7.713688e+06  4.029927e+06
1             883   9925193.0     7.713688e+06  2.211505e+06
2             884   9731791.0     7.707099e+06  2.024692e+06
3             885   3534231.0     3.098032e+06  4.361987e+05
4             886   9724893.0     7.713688e+06  2.011205e+06
```

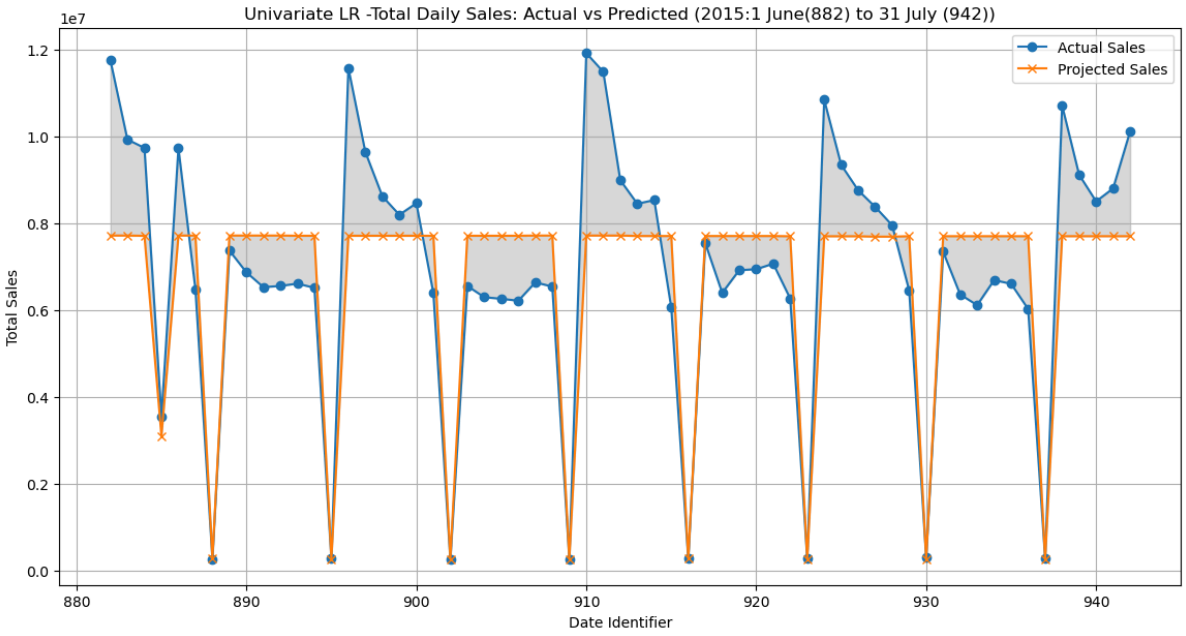


In [227…

```
# Filter the DataFrame to include only DateIdentifiers 882-942
df_predict = dft[(dft['DateIdentifier'] >= 882) & (dft['DateIdentifier'] <=

# Predict sales based on AvgSalesPerStore
df_predict['Predicted_Sales'] = df_predict['AvgSalesPerStore']

# Create a DataFrame to compare actual and predicted sales
comparison_df = df_predict[['Store', 'DateIdentifier', 'Sales', 'Predicted_S
comparison_df['Difference'] = comparison_df['Sales'] - comparison_df['Predic

# Ensure no NaN values in actual and predicted sales
comparison_df['Sales'].fillna(0, inplace=True)
comparison_df['Predicted_Sales'].fillna(0, inplace=True)

# Calculate evaluation metrics
actual_sales = comparison_df['Sales']
predicted_sales = comparison_df['Predicted_Sales']
mse = mean_squared_error(actual_sales, predicted_sales)
r2 = r2_score(actual_sales, predicted_sales)
rmspe = np.sqrt(np.mean(((actual_sales - predicted_sales) / actual_sales.rep

print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Root Mean Squared Percentage Error (RMSPE):", rmspe)

# Show the first few rows of the comparison
print("Comparison of Actual and Predicted Sales:")
print(comparison_df.head())

# Calculate total sales per day (projected vs actual)
daily_sales_comparison = comparison_df.groupby('DateIdentifier').agg({
    'Sales': 'sum',
    'Predicted_Sales': 'sum'
}).reset_index()
daily_sales_comparison['Difference'] = daily_sales_comparison['Sales'] - dai

print("\nDaily Sales Comparison (Actual vs Projected):")
print(daily_sales_comparison.head())

# Plotting daily sales comparison
plt.figure(figsize=(14, 7))
plt.plot(daily_sales_comparison['DateIdentifier'], daily_sales_comparison['S
plt.plot(daily_sales_comparison['DateIdentifier'], daily_sales_comparison['P
```

```
plt.fill_between(daily_sales_comparison['DateIdentifier'], daily_sales_compa
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Univariate LR -Total Daily Sales: Actual vs Predicted (2015:1 Jur
plt.legend()
plt.grid(True)
plt.show()

# Calculate total actual and predicted sales
total_actual_sales = comparison_df['Sales'].sum()
total_predicted_sales = comparison_df['Predicted_Sales'].sum()
total_difference = total_actual_sales - total_predicted_sales

print("\nTotal Sales Comparison (Actual vs Projected):")
print("Total Actual Sales:", total_actual_sales)
print("Total Predicted Sales:", total_predicted_sales)
print("Total Difference:", total_difference)
```

```
Mean Squared Error: 3918601.8070568186
R-squared: 0.5951536370125072
Root Mean Squared Percentage Error (RMSPE): 0.3177693042338759
Comparison of Actual and Predicted Sales:
    Store  DateIdentifier   Sales  Predicted_Sales   Difference
48      1             942  5263.0      4781.312757   481.687243
49      1             941  5020.0      4781.312757   238.687243
50      1             940  4782.0      4781.312757     0.687243
51      1             939  5011.0      4781.312757   229.687243
52      1             938  6102.0      4781.312757  1320.687243

Daily Sales Comparison (Actual vs Projected):
   DateIdentifier       Sales  Predicted_Sales    Difference
0             882  11743615.0     7.713688e+06  4.029927e+06
1             883   9925193.0     7.713688e+06  2.211505e+06
2             884   9731791.0     7.707099e+06  2.024692e+06
3             885   3534231.0     3.098032e+06  4.361987e+05
4             886   9724893.0     7.713688e+06  2.011205e+06
```




```
Total Sales Comparison (Actual vs Projected):
Total Actual Sales: 419685989.0
Total Predicted Sales: 405914594.4421616
Total Difference: 13771394.55783838
```

In [229…

```
# Step info
dft.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 844338 entries, 48 to 1058295
Data columns (total 31 columns):
 #   Column                            Non-Null Count   Dtype  
---  ------                            --------------   -----  
 0   Sales                             844338 non-null  float64
 1   Store                             844338 non-null  int64  
 2   DayOfWeek                         844338 non-null  int64  
 3   Open                              844338 non-null  float64
 4   Promo                             844338 non-null  int64  
 5   StateHoliday                      844338 non-null  int64  
 6   SchoolHoliday                     844338 non-null  int64  
 7   CompetitionDistance               844338 non-null  float64
 8   PromoInterval                     844338 non-null  float64
 9   CompetitionDistanceDecile         844338 non-null  int64  
 10  Type_Assort                       844338 non-null  int64  
 11  PromoRun                          844338 non-null  float64
 12  DateIdentifier                    844338 non-null  int64  
 13  P2cal                             844338 non-null  int64  
 14  WeekNumber                        844338 non-null  int64  
 15  Year                              844338 non-null  int64  
 16  WeekOfYear                        844338 non-null  int64  
 17  AvgSalesPerStore                  844338 non-null  float64
 18  StoreRank                         844338 non-null  float64
 19  Index_DayOfWeek                   844338 non-null  float64
 20  Index_Promo                       844338 non-null  float64
 21  Index_StateHoliday                844338 non-null  float64
 22  Index_SchoolHoliday               844338 non-null  float64
 23  Index_CompetitionDistance         844338 non-null  float64
 24  Index_CompetitionDistanceDecile   844338 non-null  float64
 25  Index_PromoInterval               844338 non-null  float64
 26  Index_Type_Assort                 844338 non-null  float64
 27  Index_PromoRun                    844338 non-null  float64
 28  Index_P2cal                       844338 non-null  float64
 29  Index_Year                        844338 non-null  float64
 30  Index_WeekOfYear                  844338 non-null  float64
dtypes: float64(19), int64(12)
memory usage: 206.1 MB
```

In [231…

```
# Step describe
dft.describe().T
```

Out[231]:

	count	mean	std	min	
Sales	844338.0	6955.959134	3103.815515	46.000000	4859.
Store	844338.0	558.421374	321.730861	1.000000	280.
DayOfWeek	844338.0	3.520350	1.723712	1.000000	2.
Open	844338.0	1.000000	0.000000	1.000000	1.
Promo	844338.0	0.446356	0.497114	0.000000	0.
StateHoliday	844338.0	2.997107	0.090334	0.000000	3.
SchoolHoliday	844338.0	0.193578	0.395102	0.000000	0.
CompetitionDistance	844338.0	5458.019004	7799.457551	20.000000	710.
PromoInterval	844338.0	0.805774	0.980689	0.000000	0.
CompetitionDistanceDecile	844338.0	5.484583	2.898723	1.000000	3.
Type_Assort	844338.0	3.325376	3.358630	0.000000	0.
PromoRun	844338.0	88.099685	106.493952	0.000000	0.
DateIdentifier	844338.0	466.047670	274.494262	1.000000	228.
P2cal	844338.0	0.429696	0.495033	0.000000	0.
WeekNumber	844338.0	67.055272	39.215345	1.000000	33.
Year	844338.0	2013.835481	0.778602	2013.000000	2013.
WeekOfYear	844338.0	23.610282	14.410994	1.000000	11.
AvgSalesPerStore	844338.0	6939.648847	2406.931948	2704.002743	5308.
StoreRank	844338.0	560.467236	322.471276	1.000000	281.
Index_DayOfWeek	844338.0	1.179314	0.116446	1.000000	1
Index_Promo	844338.0	1.171054	0.190505	1.000000	1.
Index_StateHoliday	844338.0	1.000289	0.009216	1.000000	1.
Index_SchoolHoliday	844338.0	1.007442	0.015189	1.000000	1.
Index_CompetitionDistance	844338.0	2.566522	0.640961	1.000000	2
Index_CompetitionDistanceDecile	844338.0	1.050668	0.061146	1.000000	1.
Index_PromoInterval	844338.0	1.118601	0.067981	1.000000	1.
Index_Type_Assort	844338.0	1.081210	0.105402	1.000000	1
Index_PromoRun	844338.0	1.582770	0.155480	1.000000	1
Index_P2cal	844338.0	1.062937	0.054630	1.000000	1.
Index_Year	844338.0	1.018188	0.014882	1.000000	1.
Index_WeekOfYear	844338.0	1.156926	0.119490	1.000000	1.

In [233…

```
# Define the dependent and independent variables
X = dft[[
    "AvgSalesPerStore", 'Index_DayOfWeek', 'Index_Promo',
    'Index_StateHoliday', 'Index_SchoolHoliday',
    'Index_CompetitionDistanceDecile', 'Index_PromoInterval',
    'Index_Type_Assort', 'Index_P2cal',
    'Index_Year', 'Index_WeekOfYear',
]]
y = dft['Sales']
```

```
# Add a constant to the independent variables (intercept)
X = sm.add_constant(X)

# Fit the linear regression model
model = sm.OLS(y, X).fit()

# Print the summary of the regression model
model.summary()
```

Out[233]:

OLS Regression Results

Dep. Variable:	Sales	**R-squared:**	0.785
Model:	OLS	**Adj. R-squared:**	0.785
Method:	Least Squares	**F-statistic:**	2.806e+05
Date:	Tue, 04 Jun 2024	**Prob (F-statistic):**	0.00
Time:	17:24:41	**Log-Likelihood:**	-7.3375e+06
No. Observations:	844338	**AIC:**	1.468e+07
Df Residuals:	844326	**BIC:**	1.468e+07
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.97
const	-2.589e+04	231.832	-111.694	0.000	-2.63e+04	-2.54e+
AvgSalesPerStore	1.0007	0.001	1435.349	0.000	0.999	1.0
Index_DayOfWeek	3497.9580	14.186	246.579	0.000	3470.154	3525.7
Index_Promo	4882.4626	8.733	559.058	0.000	4865.345	4899.5
Index_StateHoliday	207.7412	170.468	1.219	0.223	-126.371	541.8
Index_SchoolHoliday	3312.8300	104.112	31.820	0.000	3108.774	3516.8
Index_CompetitionDistanceDecile	-170.6103	26.015	-6.558	0.000	-221.598	-119.6
Index_PromoInterval	197.9506	40.961	4.833	0.000	117.669	278.2
Index_Type_Assort	41.1775	15.533	2.651	0.008	10.732	71.6
Index_P2cal	-442.1522	50.991	-8.671	0.000	-542.092	-342.2
Index_Year	7338.2648	106.881	68.658	0.000	7128.782	7547.7
Index_WeekOfYear	4692.6658	13.350	351.499	0.000	4666.499	4718.8

Omnibus:	260203.424	**Durbin-Watson:**	1.352
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2410716.111
Skew:	1.218	**Prob(JB):**	0.00
Kurtosis:	10.911	**Cond. No.**	1.29e+06

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 1.29e+06. This might indicate that there are strong multicollinearity or other numerical problems.

In [235…

```
# Define the dependent and independent variables
X = dft[[
    "AvgSalesPerStore", 'Index_DayOfWeek', 'Index_Promo',
    'Index_StateHoliday', 'Index_SchoolHoliday',
    'Index_CompetitionDistanceDecile', 'Index_PromoInterval',
    'Index_Type_Assort', 'Index_P2cal',
    'Index_Year', 'Index_WeekOfYear',
]]
```

```
y = dft['Sales']

# Add a constant to the independent variables (intercept)
X = sm.add_constant(X)

# Fit the linear regression model
model = sm.OLS(y, X).fit()

# Print the summary of the regression model
model.summary()
```

Out[235]:

OLS Regression Results

Dep. Variable:	Sales	**R-squared:**	0.785
Model:	OLS	**Adj. R-squared:**	0.785
Method:	Least Squares	**F-statistic:**	2.806e+05
Date:	Tue, 04 Jun 2024	**Prob (F-statistic):**	0.00
Time:	17:24:42	**Log-Likelihood:**	-7.3375e+06
No. Observations:	844338	**AIC:**	1.468e+07
Df Residuals:	844326	**BIC:**	1.468e+07
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.97
const	-2.589e+04	231.832	-111.694	0.000	-2.63e+04	-2.54e+
AvgSalesPerStore	1.0007	0.001	1435.349	0.000	0.999	1.0
Index_DayOfWeek	3497.9580	14.186	246.579	0.000	3470.154	3525.7
Index_Promo	4882.4626	8.733	559.058	0.000	4865.345	4899.5
Index_StateHoliday	207.7412	170.468	1.219	0.223	-126.371	541.8
Index_SchoolHoliday	3312.8300	104.112	31.820	0.000	3108.774	3516.8
Index_CompetitionDistanceDecile	-170.6103	26.015	-6.558	0.000	-221.598	-119.6
Index_PromoInterval	197.9506	40.961	4.833	0.000	117.669	278.2
Index_Type_Assort	41.1775	15.533	2.651	0.008	10.732	71.6
Index_P2cal	-442.1522	50.991	-8.671	0.000	-542.092	-342.2
Index_Year	7338.2648	106.881	68.658	0.000	7128.782	7547.7
Index_WeekOfYear	4692.6658	13.350	351.499	0.000	4666.499	4718.8

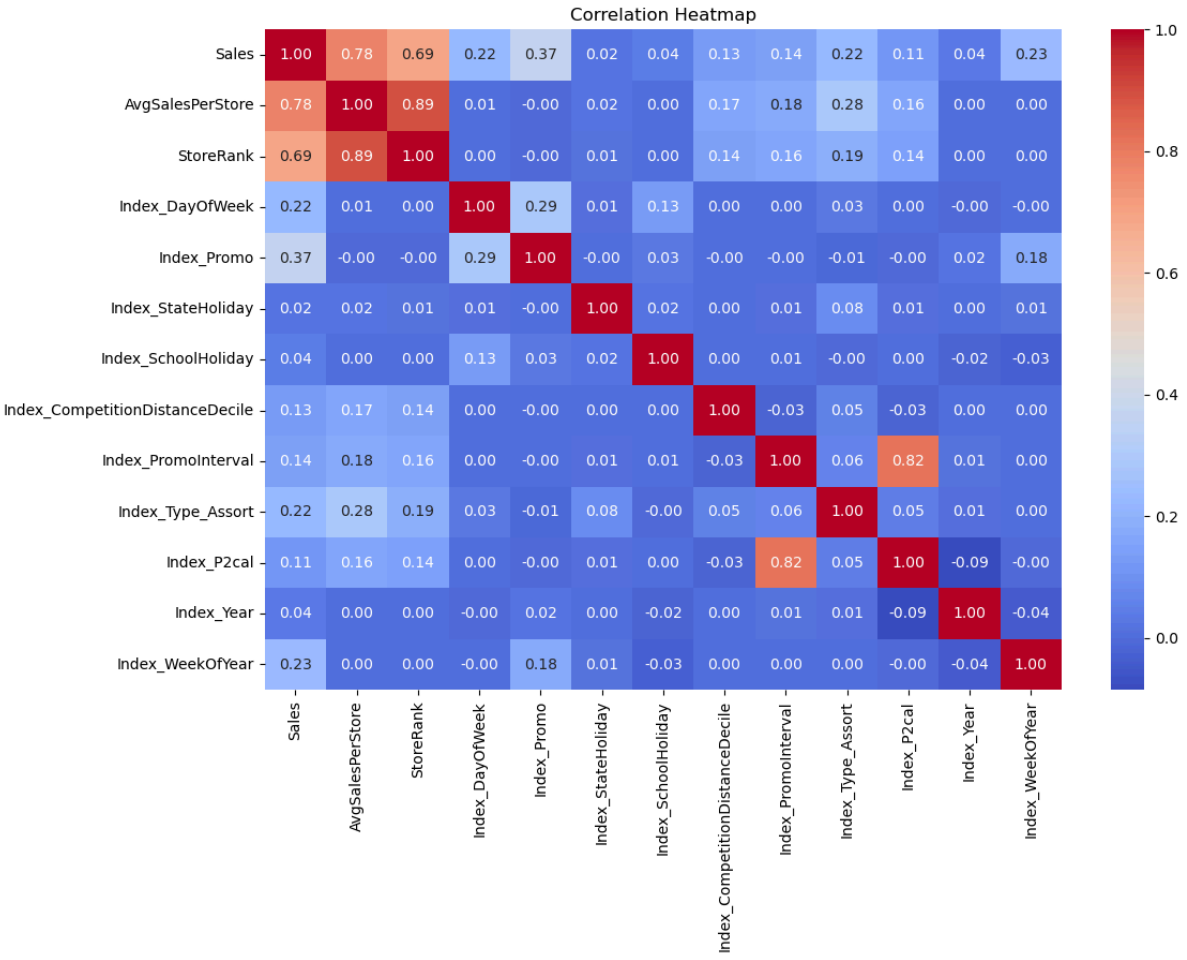
Omnibus:	260203.424	**Durbin-Watson:**	1.352
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2410716.111
Skew:	1.218	**Prob(JB):**	0.00
Kurtosis:	10.911	**Cond. No.**	1.29e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.29e+06. This might indicate that there are strong multicollinearity or other numerical problems.

In [237…

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming dft is your DataFrame
columns_to_correlate = [
    'Sales', "AvgSalesPerStore", 'StoreRank', 'Index_DayOfWeek', 'Index_Pror
    'Index_StateHoliday', 'Index_SchoolHoliday',
```

```
    'Index_CompetitionDistanceDecile', 'Index_PromoInterval',
    'Index_Type_Assort', 'Index_P2cal',
    'Index_Year', 'Index_WeekOfYear'
]

# Filter the DataFrame to include only the relevant columns
correlation_df = dft[columns_to_correlate]

# Calculate the correlation matrix
correlation_matrix = correlation_df.corr()

# Create a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```



In [239…

```
# Define the dependent and independent variables
X = dft[[
    'StoreRank', 'Index_DayOfWeek', 'Index_Promo',
    'Index_StateHoliday', 'Index_SchoolHoliday',
    'Index_CompetitionDistanceDecile', 'Index_PromoInterval',
    'Index_Type_Assort', 'Index_P2cal',
    'Index_Year', 'Index_WeekOfYear'
]]
y = dft['Sales']

# Add a constant to the independent variables (intercept)
X = sm.add_constant(X)

# Fit the linear regression model
model = sm.OLS(y, X).fit()
```

```
# Print the summary of the regression model
model.summary()
```

Out[239]:

OLS Regression Results

Dep. Variable:	Sales	**R-squared:**	0.671
Model:	OLS	**Adj. R-squared:**	0.671
Method:	Least Squares	**F-statistic:**	1.569e+05
Date:	Tue, 04 Jun 2024	**Prob (F-statistic):**	0.00
Time:	17:24:44	**Log-Likelihood:**	-7.5169e+06
No. Observations:	844338	**AIC:**	1.503e+07
Df Residuals:	844326	**BIC:**	1.503e+07
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.97
const	-2.883e+04	286.667	-100.556	0.000	-2.94e+04	-2.83e+
StoreRank	6.4179	0.006	1026.955	0.000	6.406	6.4
Index_DayOfWeek	3492.2725	17.544	199.053	0.000	3457.886	3526.6
Index_Promo	4884.0542	10.801	452.186	0.000	4862.885	4905.2
Index_StateHoliday	235.8019	210.827	1.118	0.263	-177.412	649.0
Index_SchoolHoliday	3370.3485	128.760	26.175	0.000	3117.983	3622.7
Index_CompetitionDistanceDecile	1672.6071	32.009	52.255	0.000	1609.871	1735.3
Index_PromoInterval	913.8865	50.644	18.045	0.000	814.626	1013.1
Index_Type_Assort	2543.7803	18.812	135.219	0.000	2506.909	2580.6
Index_P2cal	219.8526	63.053	3.487	0.000	96.271	343.4
Index_Year	7393.0482	132.185	55.930	0.000	7133.970	7652.1
Index_WeekOfYear	4688.9166	16.511	283.985	0.000	4656.555	4721.2

Omnibus:	427616.487	**Durbin-Watson:**	0.885
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	6193793.606
Skew:	2.094	**Prob(JB):**	0.00
Kurtosis:	15.590	**Cond. No.**	1.13e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.13e+05. This might indicate that there are strong multicollinearity or other numerical problems.

In [241…

```
# Define the dependent and independent variables
X = dft[[
    "AvgSalesPerStore",'Index_DayOfWeek', 'Index_Promo',
    'Index_StateHoliday', 'Index_SchoolHoliday',
    'Index_CompetitionDistanceDecile', 'Index_PromoInterval',
```

```
    'Index_Type_Assort', 'Index_P2cal',
    'Index_Year', 'Index_WeekOfYear'
]]

y = dft['Sales']

# Add a constant to the independent variables (intercept)
X = sm.add_constant(X)

# Fit the linear regression model
model = sm.OLS(y, X).fit()

# Print the summary of the regression model
model.summary()
```

Out[241]:

OLS Regression Results

Dep. Variable:	Sales	**R-squared:**	0.785
Model:	OLS	**Adj. R-squared:**	0.785
Method:	Least Squares	**F-statistic:**	2.806e+05
Date:	Tue, 04 Jun 2024	**Prob (F-statistic):**	0.00
Time:	17:24:45	**Log-Likelihood:**	-7.3375e+06
No. Observations:	844338	**AIC:**	1.468e+07
Df Residuals:	844326	**BIC:**	1.468e+07
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.97
const	-2.589e+04	231.832	-111.694	0.000	-2.63e+04	-2.54e+
AvgSalesPerStore	1.0007	0.001	1435.349	0.000	0.999	1.0
Index_DayOfWeek	3497.9580	14.186	246.579	0.000	3470.154	3525.7
Index_Promo	4882.4626	8.733	559.058	0.000	4865.345	4899.5
Index_StateHoliday	207.7412	170.468	1.219	0.223	-126.371	541.8
Index_SchoolHoliday	3312.8300	104.112	31.820	0.000	3108.774	3516.8
Index_CompetitionDistanceDecile	-170.6103	26.015	-6.558	0.000	-221.598	-119.6
Index_PromoInterval	197.9506	40.961	4.833	0.000	117.669	278.2
Index_Type_Assort	41.1775	15.533	2.651	0.008	10.732	71.6
Index_P2cal	-442.1522	50.991	-8.671	0.000	-542.092	-342.2
Index_Year	7338.2648	106.881	68.658	0.000	7128.782	7547.7
Index_WeekOfYear	4692.6658	13.350	351.499	0.000	4666.499	4718.8

Omnibus:	260203.424	**Durbin-Watson:**	1.352
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2410716.111
Skew:	1.218	**Prob(JB):**	0.00
Kurtosis:	10.911	**Cond. No.**	1.29e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.29e+06. This might indicate that there are strong multicollinearity or other numerical problems.

In [243…

```
# Define the dependent and independent variables
X = dft[[
    "AvgSalesPerStore",'Index_DayOfWeek', 'Index_Promo', 'Index_Year', 'Inde
y = dft['Sales']

# Add a constant to the independent variables (intercept)
X = sm.add_constant(X)
```

```
# Fit the linear regression model
model = sm.OLS(y, X).fit()

# Print the summary of the regression model
model.summary()
```

Out[243]:

OLS Regression Results

Dep. Variable:	Sales	**R-squared:**	0.785
Model:	OLS	**Adj. R-squared:**	0.785
Method:	Least Squares	**F-statistic:**	6.163e+05
Date:	Tue, 04 Jun 2024	**Prob (F-statistic):**	0.00
Time:	17:24:46	**Log-Likelihood:**	-7.3381e+06
No. Observations:	844338	**AIC:**	1.468e+07
Df Residuals:	844332	**BIC:**	1.468e+07
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-2.287e+04	110.231	-207.431	0.000	-2.31e+04	-2.26e+04
AvgSalesPerStore	1.0000	0.001	1536.382	0.000	0.999	1.001
Index_DayOfWeek	3557.8571	14.068	252.896	0.000	3530.283	3585.431
Index_Promo	4880.7665	8.737	558.628	0.000	4863.642	4897.891
Index_Year	7420.0602	105.389	70.407	0.000	7213.502	7626.618
Index_WeekOfYear	4680.9281	13.349	350.656	0.000	4654.764	4707.092

Omnibus:	260349.393	**Durbin-Watson:**	1.351
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	2413351.931
Skew:	1.219	**Prob(JB):**	0.00
Kurtosis:	10.916	**Cond. No.**	7.11e+05

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 7.11e+05. This might indicate that there are strong multicollinearity or other numerical problems.

In [245…

```
# Define the dependent and independent variables
X = dft[[
    "AvgSalesPerStore"  ]]
y = dft['Sales']

# Add a constant to the independent variables (intercept)
X = sm.add_constant(X)

# Fit the linear regression model
model = sm.OLS(y, X).fit()
```

```
# Calculate evaluation metrics
actual_sales = comparison_df['Sales']
predicted_sales = comparison_df['Predicted_Sales']
mse = mean_squared_error(actual_sales, predicted_sales)
r2 = r2_score(actual_sales, predicted_sales)
rmspe = np.sqrt(np.mean(((actual_sales - predicted_sales) / actual_sales) *

print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Root Mean Squared Percentage Error:", rmspe)

# Print the summary of the regression model
model.summary()
```

```
Mean Squared Error: 3918601.8070568186
R-squared: 0.5951536370125072
Root Mean Squared Percentage Error: 0.3177693042338759
```

Out[245]:

OLS Regression Results

Dep. Variable:	Sales	**R-squared:**	0.602
Model:	OLS	**Adj. R-squared:**	0.602
Method:	Least Squares	**F-statistic:**	1.277e+06
Date:	Tue, 04 Jun 2024	**Prob (F-statistic):**	0.00
Time:	17:24:48	**Log-Likelihood:**	-7.5980e+06
No. Observations:	844338	**AIC:**	1.520e+07
Df Residuals:	844336	**BIC:**	1.520e+07
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	13.1155	6.504	2.017	0.044	0.368	25.863
AvgSalesPerStore	1.0005	0.001	1129.894	0.000	0.999	1.002

Omnibus:	171931.162	**Durbin-Watson:**	1.189
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	629448.232
Skew:	0.997	**Prob(JB):**	0.00
Kurtosis:	6.731	**Cond. No.**	2.24e+04

Notes:

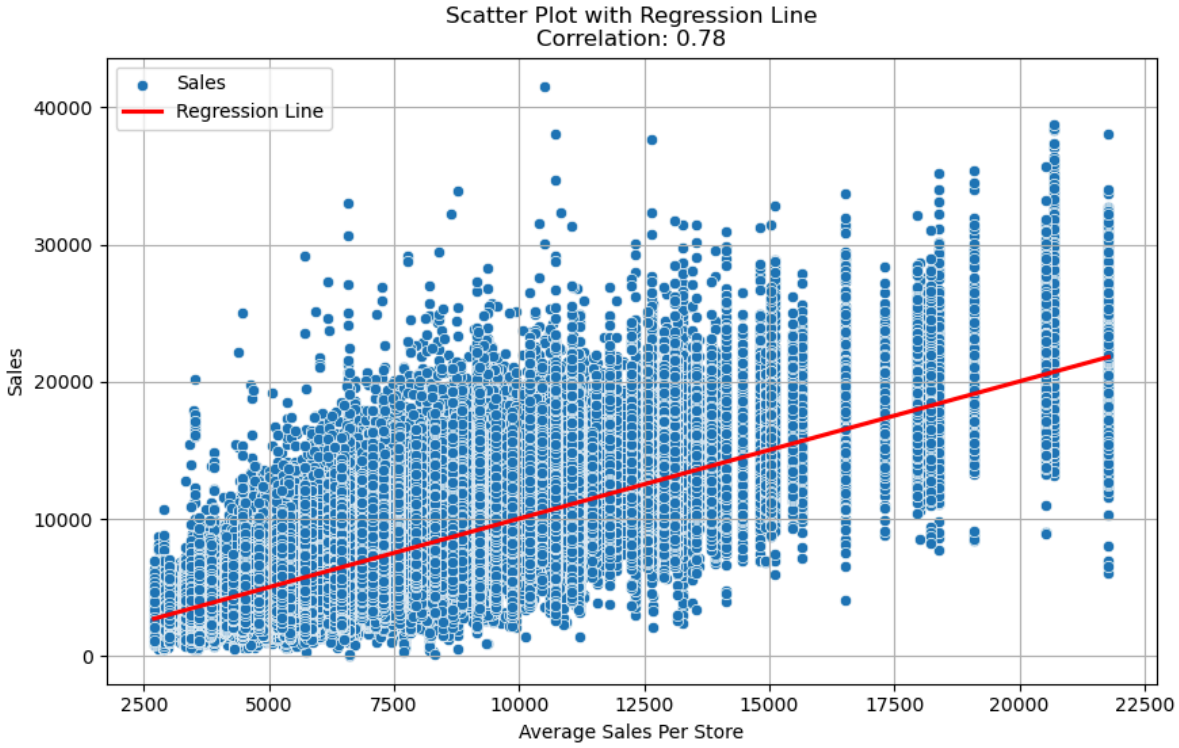
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 2.24e+04. This might indicate that there are strong multicollinearity or other numerical problems.

In [247…

```
# Define the dependent and independent variables
X = dft["AvgSalesPerStore"]
y = dft['Sales']

# Fit the linear regression model
X_with_const = sm.add_constant(X)
model = sm.OLS(y, X_with_const).fit()
```

```
# Calculate the correlation coefficient
correlation = X.corr(y)

# Plot the scatter plot with the regression line
plt.figure(figsize=(10, 6))
sns.scatterplot(x=X, y=y, label='Sales')
sns.regplot(x=X, y=y, scatter=False, color='red', label='Regression Line')

plt.xlabel('Average Sales Per Store')
plt.ylabel('Sales')
plt.title(f'Scatter Plot with Regression Line\nCorrelation: {correlation:.2f
plt.legend()
plt.grid(True)
plt.show()

# Print the summary of the regression model
print(model.summary())
```

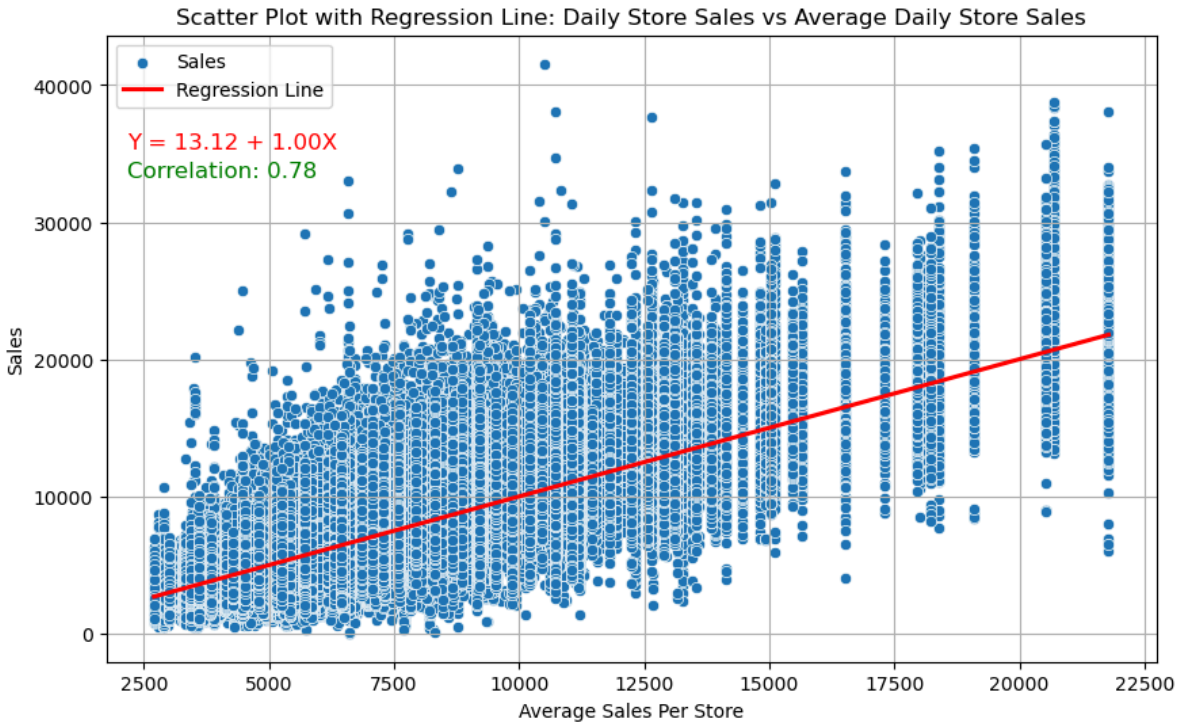


```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.602
Model:                            OLS   Adj. R-squared:                  0.602
Method:                 Least Squares   F-statistic:                 1.277e+06
Date:                Tue, 04 Jun 2024   Prob (F-statistic):               0.00
Time:                        17:25:41   Log-Likelihood:            -7.5980e+06
No. Observations:              844338   AIC:                         1.520e+07
Df Residuals:                  844336   BIC:                         1.520e+07
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         13.1155      6.504      2.017      0.044       0.368      25.863
AvgSalesPerStore     1.0005      0.001   1129.894      0.000       0.999      1.002
==============================================================================
Omnibus:                   171931.162   Durbin-Watson:                   1.189
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           629448.232
Skew:                           0.997   Prob(JB):                         0.00
Kurtosis:                       6.731   Cond. No.                     2.24e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.24e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [248…

```
# Define the dependent and independent variables
X = dft["AvgSalesPerStore"]
y = dft['Sales']

# Fit the linear regression model
X_with_const = sm.add_constant(X)
model = sm.OLS(y, X_with_const).fit()

# Extract the model coefficients
intercept = model.params[0]
slope = model.params[1]

# Calculate the correlation coefficient
correlation = X.corr(y)
```

```
# Plot the scatter plot with the regression line
plt.figure(figsize=(10, 6))
sns.scatterplot(x=X, y=y, label='Sales')
sns.regplot(x=X, y=y, scatter=False, color='red', label='Regression Line')

# Display the regression equation on the plot
equation_text = f'Y = {intercept:.2f} + {slope:.2f}X'
plt.text(X.max()*0.1, y.max()*0.85, equation_text, fontsize=12, color='red')

# Display the correlation coefficient on the plot
correlation_text = f'Correlation: {correlation:.2f}'
plt.text(X.max()*0.1, y.max()*0.8, correlation_text, fontsize=12, color='gre

plt.xlabel('Average Sales Per Store')
plt.ylabel('Sales')
plt.title('Scatter Plot with Regression Line: Daily Store Sales vs Average D
plt.legend()
plt.grid(True)
plt.show()

# Print the summary of the regression model
print(model.summary())
```



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.602
Model:                            OLS   Adj. R-squared:                  0.602
Method:                 Least Squares   F-statistic:                 1.277e+06
Date:                Tue, 04 Jun 2024   Prob (F-statistic):               0.00
Time:                        17:26:36   Log-Likelihood:            -7.5980e+06
No. Observations:              844338   AIC:                         1.520e+07
Df Residuals:                  844336   BIC:                         1.520e+07
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         13.1155      6.504      2.017      0.044       0.368      25.863
AvgSalesPerStore     1.0005      0.001   1129.894      0.000       0.999      1.002
==============================================================================
Omnibus:                   171931.162   Durbin-Watson:                   1.189
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           629448.232
Skew:                           0.997   Prob(JB):                         0.00
Kurtosis:                       6.731   Cond. No.                     2.24e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.24e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [249…

```python
import pandas as pd
import numpy as np
import statsmodels.api as sm

# Define the dependent and independent variables
X = dft[[
    "AvgSalesPerStore", 'Index_DayOfWeek', 'Index_Promo',
    'Index_StateHoliday', 'Index_SchoolHoliday',
    'Index_P2cal',
    'Index_Year', 'Index_WeekOfYear'
]]
y = dft['Sales']

# Add a constant to the independent variables (intercept)
X = sm.add_constant(X)
```

```
# Fit the linear regression model
model = sm.OLS(y, X).fit()

# Print the summary of the regression model
print(model.summary())

# Get the model parameters (coefficients)
params = model.params

# Create the regression equation as a string
equation = f'Sales = {params["const"]:.2f}'
for col in X.columns[1:]:
    equation += f' + ({params[col]:.2f} * {col})'

print("Regression Equation:")
print(equation)
```

OLS Regression Results

Dep. Variable:	Sales	R-squared:	0.785
Model:	OLS	Adj. R-squared:	0.785
Method:	Least Squares	F-statistic:	3.858e+05
Date:	Tue, 04 Jun 2024	Prob (F-statistic):	0.00
Time:	17:26:36	Log-Likelihood:	-7.3375e+06
No. Observations:	844338	AIC:	1.468e+07
Df Residuals:	844329	BIC:	1.468e+07
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-2.615e+04	229.580	-113.923	0.000	-2.66e+04	-2.57e+04
AvgSalesPerStore	1.0007	0.001	1518.834	0.000	0.999	1.002
Index_DayOfWeek	3499.2500	14.180	246.778	0.000	3471.458	3527.042
Index_Promo	4881.8843	8.732	559.096	0.000	4864.770	4898.998
Index_StateHoliday	246.4854	169.959	1.450	0.147	-86.628	579.599
Index_SchoolHoliday	3313.1572	104.108	31.824	0.000	3109.109	3517.205
Index_P2cal	-229.2822	29.130	-7.871	0.000	-286.376	-172.188
Index_Year	7416.3158	105.748	70.132	0.000	7209.054	7623.578
Index_WeekOfYear	4693.7537	13.349	351.612	0.000	4667.590	4719.918

Omnibus:	259935.133	Durbin-Watson:	1.352
Prob(Omnibus):	0.000	Jarque-Bera (JB):	2409464.305
Skew:	1.216	Prob(JB):	0.00
Kurtosis:	10.910	Cond. No.	1.28e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.28e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
Regression Equation:
Sales = -26154.38 + (1.00 * AvgSalesPerStore) + (3499.25 * Index_DayOfWeek)

```
+ (4881.88 * Index_Promo) + (246.49 * Index_StateHoliday) + (3313.16 * Inde
x_SchoolHoliday) + (-229.28 * Index_P2cal) + (7416.32 * Index_Year) + (469
3.75 * Index_WeekOfYear)
```

In [250…

```
# Define the dependent and independent variables
X = dft[[
    "AvgSalesPerStore", 'Index_DayOfWeek', 'Index_Promo',
    'Index_Year', 'Index_WeekOfYear'
]]
y = dft['Sales']

# Add a constant to the independent variables (intercept)
X = sm.add_constant(X)

# Fit the linear regression model
model = sm.OLS(y, X).fit()

# Print the summary of the regression model
print(model.summary())

# Get the model parameters (coefficients)
params = model.params

# Create the regression equation as a string
equation = f'Sales = {params["const"]:.2f}'
for col in X.columns[1:]:
    equation += f' + ({params[col]:.2f} * {col})'

print("Regression Equation:")
print(equation)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Sales   R-squared:                       0.785
Model:                            OLS   Adj. R-squared:                  0.785
Method:                 Least Squares   F-statistic:                 6.163e+05
Date:                Tue, 04 Jun 2024   Prob (F-statistic):               0.00
Time:                        17:26:36   Log-Likelihood:            -7.3381e+06
No. Observations:              844338   AIC:                         1.468e+07
Df Residuals:                  844332   BIC:                         1.468e+07
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          -2.287e+04    110.231   -207.431      0.000   -2.31e+04   -2.26e+04
AvgSalesPerStore     1.0000      0.001   1536.382      0.000       0.999       1.001
Index_DayOfWeek   3557.8571     14.068    252.896      0.000    3530.283    3585.431
Index_Promo       4880.7665      8.737    558.628      0.000    4863.642    4897.891
Index_Year        7420.0602    105.389     70.407      0.000    7213.502    7626.618
Index_WeekOfYear  4680.9281     13.349    350.656      0.000    4654.764    4707.092
==============================================================================
Omnibus:                   260349.393   Durbin-Watson:                   1.351
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          2413351.931
Skew:                           1.219   Prob(JB):                         0.00
Kurtosis:                      10.916   Cond. No.                     7.11e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 7.11e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
Regression Equation:
Sales = -22865.33 + (1.00 * AvgSalesPerStore) + (3557.86 * Index_DayOfWeek) + (4880.77 * Index_Promo) + (7420.06 * Index_Year) + (4680.93 * Index_WeekOfYear)
```

In [251…

```
# Step info
dft.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 844338 entries, 48 to 1058295
Data columns (total 31 columns):
 #   Column                            Non-Null Count   Dtype  
---  ------                            --------------   -----  
 0   Sales                             844338 non-null  float64
 1   Store                             844338 non-null  int64  
 2   DayOfWeek                         844338 non-null  int64  
 3   Open                              844338 non-null  float64
 4   Promo                             844338 non-null  int64  
 5   StateHoliday                      844338 non-null  int64  
 6   SchoolHoliday                     844338 non-null  int64  
 7   CompetitionDistance               844338 non-null  float64
 8   PromoInterval                     844338 non-null  float64
 9   CompetitionDistanceDecile         844338 non-null  int64  
 10  Type_Assort                       844338 non-null  int64  
 11  PromoRun                          844338 non-null  float64
 12  DateIdentifier                    844338 non-null  int64  
 13  P2cal                             844338 non-null  int64  
 14  WeekNumber                        844338 non-null  int64  
 15  Year                              844338 non-null  int64  
 16  WeekOfYear                        844338 non-null  int64  
 17  AvgSalesPerStore                  844338 non-null  float64
 18  StoreRank                         844338 non-null  float64
 19  Index_DayOfWeek                   844338 non-null  float64
 20  Index_Promo                       844338 non-null  float64
 21  Index_StateHoliday                844338 non-null  float64
 22  Index_SchoolHoliday               844338 non-null  float64
 23  Index_CompetitionDistance         844338 non-null  float64
 24  Index_CompetitionDistanceDecile   844338 non-null  float64
 25  Index_PromoInterval               844338 non-null  float64
 26  Index_Type_Assort                 844338 non-null  float64
 27  Index_PromoRun                    844338 non-null  float64
 28  Index_P2cal                       844338 non-null  float64
 29  Index_Year                        844338 non-null  float64
 30  Index_WeekOfYear                  844338 non-null  float64
dtypes: float64(19), int64(12)
memory usage: 206.1 MB
```

In [252…

```python
import pandas as pd
import numpy as np
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Define the dependent and independent variables
X = dft[[
    "AvgSalesPerStore", 'Index_DayOfWeek', 'Index_Promo',
    'Index_StateHoliday', 'Index_SchoolHoliday',
    'Index_P2cal',
    'Index_Year', 'Index_WeekOfYear'
]]
y = dft['Sales']

# Add a constant to the independent variables (intercept)
X = sm.add_constant(X)

# Fit the linear regression model
model = sm.OLS(y, X).fit()

# Print the summary of the regression model
print(model.summary())

# Get the model parameters (coefficients)
```

```
params = model.params

# Create the regression equation as a string
equation = f'Sales = {params["const"]:.2f}'
for col in X.columns[1:]:
    equation += f' + ({params[col]:.2f} * {col})'

print("Regression Equation:")
print(equation)

# Filter the DataFrame for DateIdentifier 882-942
prediction_df = dft[(dft['DateIdentifier'] >= 882) & (dft['DateIdentifier']

# Manually compute predicted sales using the regression coefficients
prediction_df['Predicted_Sales'] = (
    params["const"]
    + params["AvgSalesPerStore"] * prediction_df["AvgSalesPerStore"]
    + params["Index_DayOfWeek"] * prediction_df["Index_DayOfWeek"]
    + params["Index_Promo"] * prediction_df["Index_Promo"]
    + params["Index_StateHoliday"] * prediction_df["Index_StateHoliday"]
    + params["Index_SchoolHoliday"] * prediction_df["Index_SchoolHoliday"]
    + params["Index_P2cal"] * prediction_df["Index_P2cal"]
    + params["Index_Year"] * prediction_df["Index_Year"]
    + params["Index_WeekOfYear"] * prediction_df["Index_WeekOfYear"]
)

# Compare predicted and actual sales
comparison_df = prediction_df[['DateIdentifier', 'Store', 'Sales', 'Predicte
comparison_df['Difference'] = comparison_df['Sales'] - comparison_df['Predi

# Show the first few rows of the comparison
print(comparison_df.head())

# Calculate evaluation metrics
actual_sales = comparison_df['Sales']
predicted_sales = comparison_df['Predicted_Sales']
mse = mean_squared_error(actual_sales, predicted_sales)
r2 = r2_score(actual_sales, predicted_sales)
rmspe = np.sqrt(np.mean(((actual_sales - predicted_sales) / actual_sales) **

print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Root Mean Squared Percentage Error:", rmspe)

# Plot actual vs predicted sales for DateIdentifier 882-942
plt.figure(figsize=(12, 8))
plt.plot(comparison_df['DateIdentifier'], comparison_df['Sales'], label='Act
plt.plot(comparison_df['DateIdentifier'], comparison_df['Predicted_Sales'],
plt.fill_between(comparison_df['DateIdentifier'], comparison_df['Sales'], co
plt.xlabel('Date Identifier')
plt.ylabel('Sales')
plt.title('Multi-variate LR - Stores: Daily Actual vs Predicted Sales (2015
plt.legend()
plt.grid(True)
plt.show()

# Aggregate daily total sales for all stores
daily_sales_actual = comparison_df.groupby('DateIdentifier')['Sales'].sum()
daily_sales_predicted = comparison_df.groupby('DateIdentifier')['Predicted_S

# Plot total daily sales: predicted vs actual
plt.figure(figsize=(12, 8))
plt.plot(daily_sales_actual.index, daily_sales_actual.values, label='Total A
plt.plot(daily_sales_predicted.index, daily_sales_predicted.values, label='T
```

```
plt.fill_between(daily_sales_actual.index, daily_sales_actual.values, daily_
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Multi-variate LR-Total Daily Sales: Actual vs Predicted (2015:1 J
plt.legend()
plt.grid(True)
plt.show()
```

OLS Regression Results

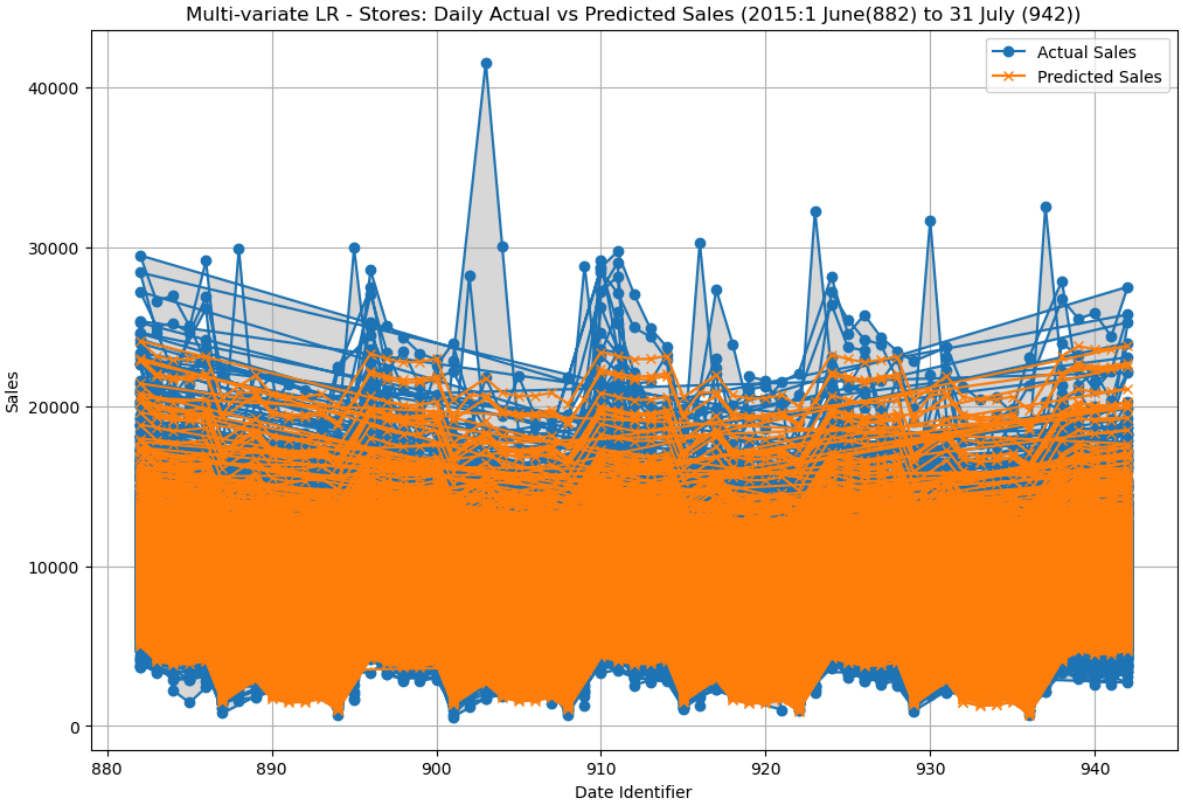
Dep. Variable:	Sales	R-squared:	0.785
Model:	OLS	Adj. R-squared:	0.785
Method:	Least Squares	F-statistic:	3.858e+05
Date:	Tue, 04 Jun 2024	Prob (F-statistic):	0.00
Time:	17:26:37	Log-Likelihood:	-7.3375e+06
No. Observations:	844338	AIC:	1.468e+07
Df Residuals:	844329	BIC:	1.468e+07
Df Model:	8		
Covariance Type:	nonrobust		

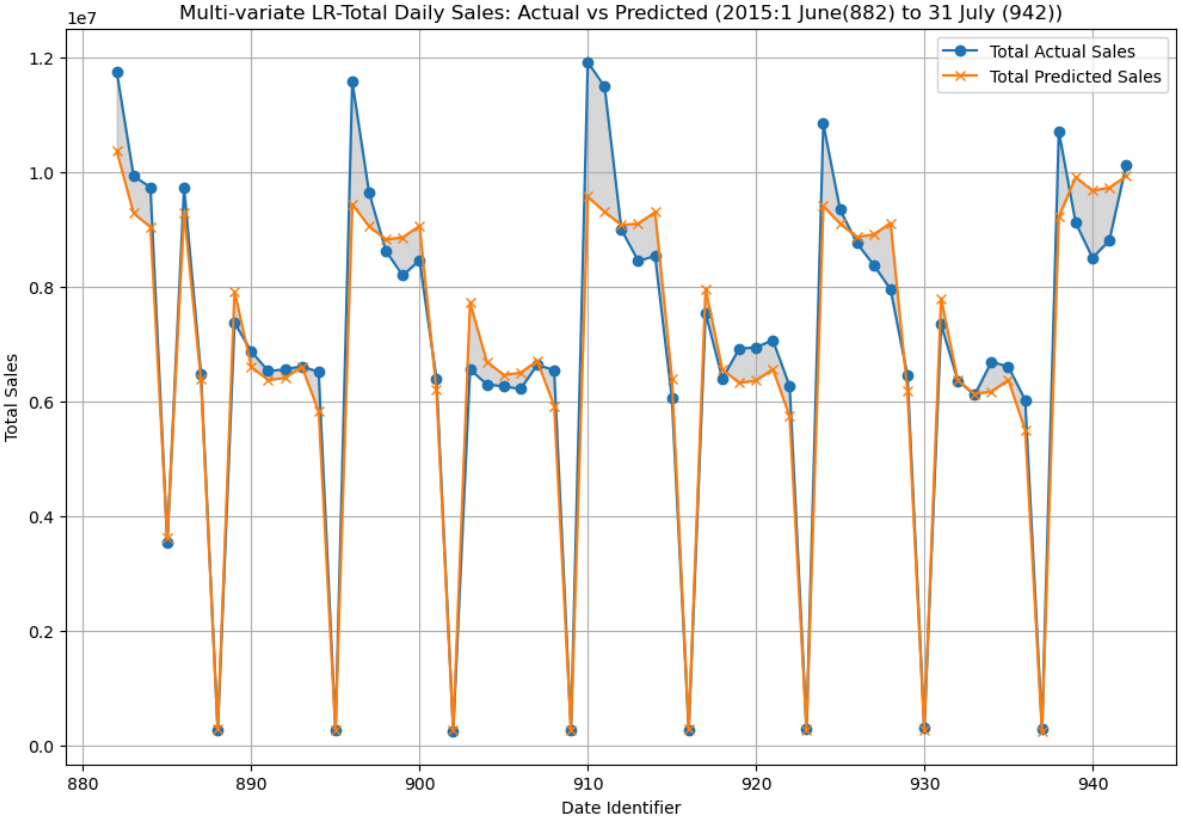
	coef	std err	t	P>|t|	[0.025	0.975]
const	-2.615e+04	229.580	-113.923	0.000	-2.66e+04	-2.57e+04
AvgSalesPerStore	1.0007	0.001	1518.834	0.000	0.999	1.002
Index_DayOfWeek	3499.2500	14.180	246.778	0.000	3471.458	3527.042
Index_Promo	4881.8843	8.732	559.096	0.000	4864.770	4898.998
Index_StateHoliday	246.4854	169.959	1.450	0.147	-86.628	579.599
Index_SchoolHoliday	3313.1572	104.108	31.824	0.000	3109.109	3517.205
Index_P2cal	-229.2822	29.130	-7.871	0.000	-286.376	-172.188
Index_Year	7416.3158	105.748	70.132	0.000	7209.054	7623.578
Index_WeekOfYear	4693.7537	13.349	351.612	0.000	4667.590	4719.918

Omnibus:	259935.133	Durbin-Watson:	1.352
Prob(Omnibus):	0.000	Jarque-Bera (JB):	2409464.305
Skew:	1.216	Prob(JB):	0.00
Kurtosis:	10.910	Cond. No.	1.28e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.28e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
Regression Equation:
Sales = -26154.38 + (1.00 * AvgSalesPerStore) + (3499.25 * Index_DayOfWeek)

```
+ (4881.88 * Index_Promo) + (246.49 * Index_StateHoliday) + (3313.16 * Inde
x_SchoolHoliday) + (-229.28 * Index_P2cal) + (7416.32 * Index_Year) + (469
3.75 * Index_WeekOfYear)
    DateIdentifier  Store   Sales  Predicted_Sales   Difference
48             942      1  5263.0      6785.729383 -1522.729383
49             941      1  5020.0      6605.723997 -1585.723997
50             940      1  4782.0      6571.436351 -1789.436351
51             939      1  5011.0      6779.207561 -1768.207561
52             938      1  6102.0      6156.485755   -54.485755
Mean Squared Error: 2055457.6584662928
R-squared: 0.7876424810996876
Root Mean Squared Percentage Error: 0.2175959074890229
```





In [253…

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Define the dependent and independent variables
X = dft[[
    "AvgSalesPerStore", 'Index_DayOfWeek', 'Index_Promo',
    'Index_StateHoliday', 'Index_SchoolHoliday',
    'Index_P2cal',
    'Index_Year', 'Index_WeekOfYear'
]]
y = dft['Sales']

# Add a constant to the independent variables (intercept)
X = sm.add_constant(X)

# Fit the linear regression model
model = sm.OLS(y, X).fit()

# Print the summary of the regression model
print(model.summary())

# Get the model parameters (coefficients)
params = model.params

# Create the regression equation as a string
equation = f'Sales = {params["const"]:.2f}'
for col in X.columns[1:]:
    equation += f' + ({params[col]:.2f} * {col})'

print("Regression Equation:")
print(equation)

# Filter the DataFrame for DateIdentifier 882-942
prediction_df = dft[(dft['DateIdentifier'] >= 882) & (dft['DateIdentifier']

# Manually compute predicted sales using the regression coefficients
prediction_df['Predicted_Sales'] = (
    params["const"]
    + params["AvgSalesPerStore"] * prediction_df["AvgSalesPerStore"]
    + params["Index_DayOfWeek"] * prediction_df["Index_DayOfWeek"]
    + params["Index_Promo"] * prediction_df["Index_Promo"]
    + params["Index_StateHoliday"] * prediction_df["Index_StateHoliday"]
    + params["Index_SchoolHoliday"] * prediction_df["Index_SchoolHoliday"]
    + params["Index_P2cal"] * prediction_df["Index_P2cal"]
    + params["Index_Year"] * prediction_df["Index_Year"]
    + params["Index_WeekOfYear"] * prediction_df["Index_WeekOfYear"]
)

# Compare predicted and actual sales
comparison_df = prediction_df[['DateIdentifier', 'Store', 'Sales', 'Predicte
comparison_df['Difference'] = comparison_df['Sales'] - comparison_df['Predi

# Show the first few rows of the comparison
print(comparison_df.head())

# Calculate evaluation metrics
actual_sales = comparison_df['Sales']
predicted_sales = comparison_df['Predicted_Sales']
mse = mean_squared_error(actual_sales, predicted_sales)
r2 = r2_score(actual_sales, predicted_sales)
```

```
rmspe = np.sqrt(np.mean(((actual_sales - predicted_sales) / actual_sales) *

print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Root Mean Squared Percentage Error:", rmspe)

# Plot actual vs predicted sales for DateIdentifier 882-942
plt.figure(figsize=(12, 8))
plt.plot(comparison_df['DateIdentifier'], comparison_df['Sales'], label='Act
plt.plot(comparison_df['DateIdentifier'], comparison_df['Predicted_Sales'],
plt.fill_between(comparison_df['DateIdentifier'], comparison_df['Sales'], co
plt.xlabel('Date Identifier')
plt.ylabel('Sales')
plt.title('Multi-variate LR - Stores: Daily Actual vs Predicted Sales (2015:
plt.legend()
plt.grid(True)
plt.show()

# Aggregate daily total sales for all stores
daily_sales_actual = comparison_df.groupby('DateIdentifier')['Sales'].sum()
daily_sales_predicted = comparison_df.groupby('DateIdentifier')['Predicted_S

# Plot total daily sales: predicted vs actual
plt.figure(figsize=(12, 8))
plt.plot(daily_sales_actual.index, daily_sales_actual.values, label='Total A
plt.plot(daily_sales_predicted.index, daily_sales_predicted.values, label='T
plt.fill_between(daily_sales_actual.index, daily_sales_actual.values, daily_
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Multi-variate LR-Total Daily Sales: Actual vs Predicted (2015:1 J
plt.legend()
plt.grid

# Calculate total actual and predicted sales
total_actual_sales = comparison_df['Sales'].sum()
total_predicted_sales = comparison_df['Predicted_Sales'].sum()
total_difference = total_actual_sales - total_predicted_sales

print("\nTotal Sales Comparison (Actual vs Projected):")
print("Total Actual Sales:", total_actual_sales)
print("Total Predicted Sales:", total_predicted_sales)
print("Total Difference:", total_difference)
```

OLS Regression Results

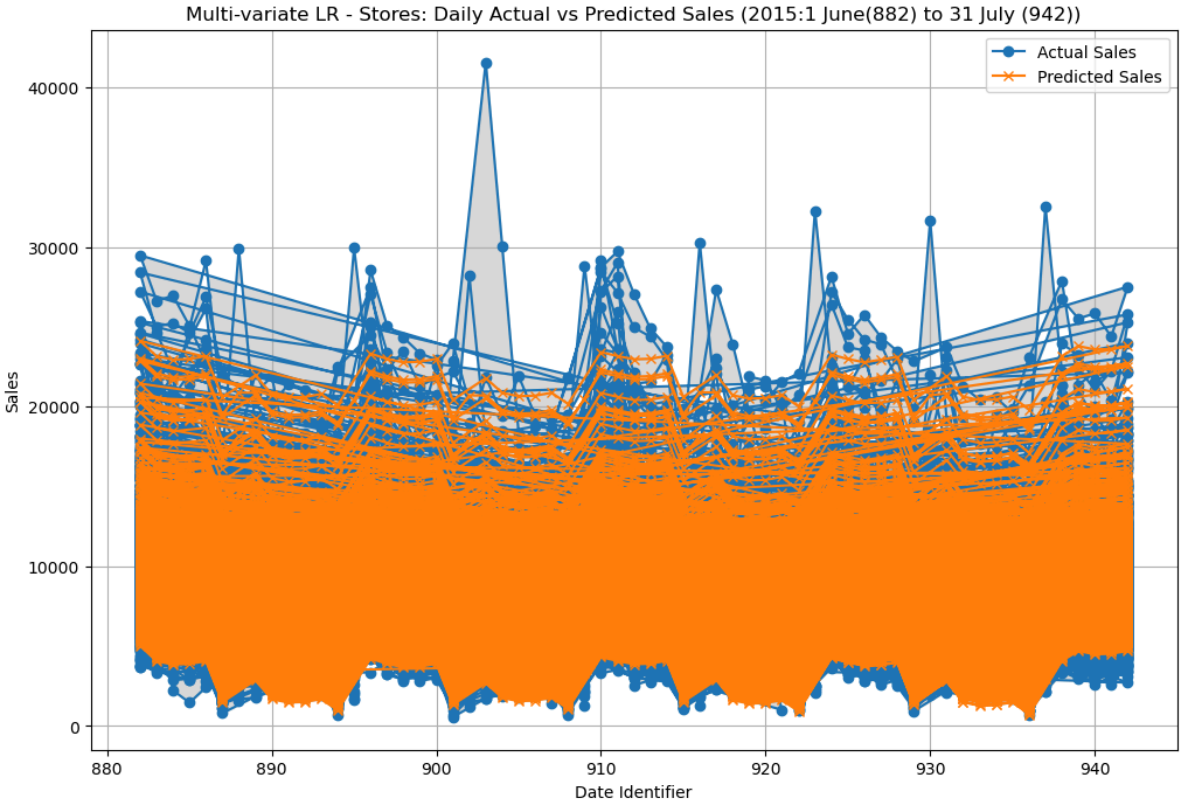
Dep. Variable:	Sales	R-squared:	0.785
Model:	OLS	Adj. R-squared:	0.785
Method:	Least Squares	F-statistic:	3.858e+05
Date:	Tue, 04 Jun 2024	Prob (F-statistic):	0.00
Time:	17:26:39	Log-Likelihood:	-7.3375e+06
No. Observations:	844338	AIC:	1.468e+07
Df Residuals:	844329	BIC:	1.468e+07
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-2.615e+04	229.580	-113.923	0.000	-2.66e+04	-2.57e+04
AvgSalesPerStore	1.0007	0.001	1518.834	0.000	0.999	1.002
Index_DayOfWeek	3499.2500	14.180	246.778	0.000	3471.458	3527.042
Index_Promo	4881.8843	8.732	559.096	0.000	4864.770	4898.998
Index_StateHoliday	246.4854	169.959	1.450	0.147	-86.628	579.599
Index_SchoolHoliday	3313.1572	104.108	31.824	0.000	3109.109	3517.205
Index_P2cal	-229.2822	29.130	-7.871	0.000	-286.376	-172.188
Index_Year	7416.3158	105.748	70.132	0.000	7209.054	7623.578
Index_WeekOfYear	4693.7537	13.349	351.612	0.000	4667.590	4719.918

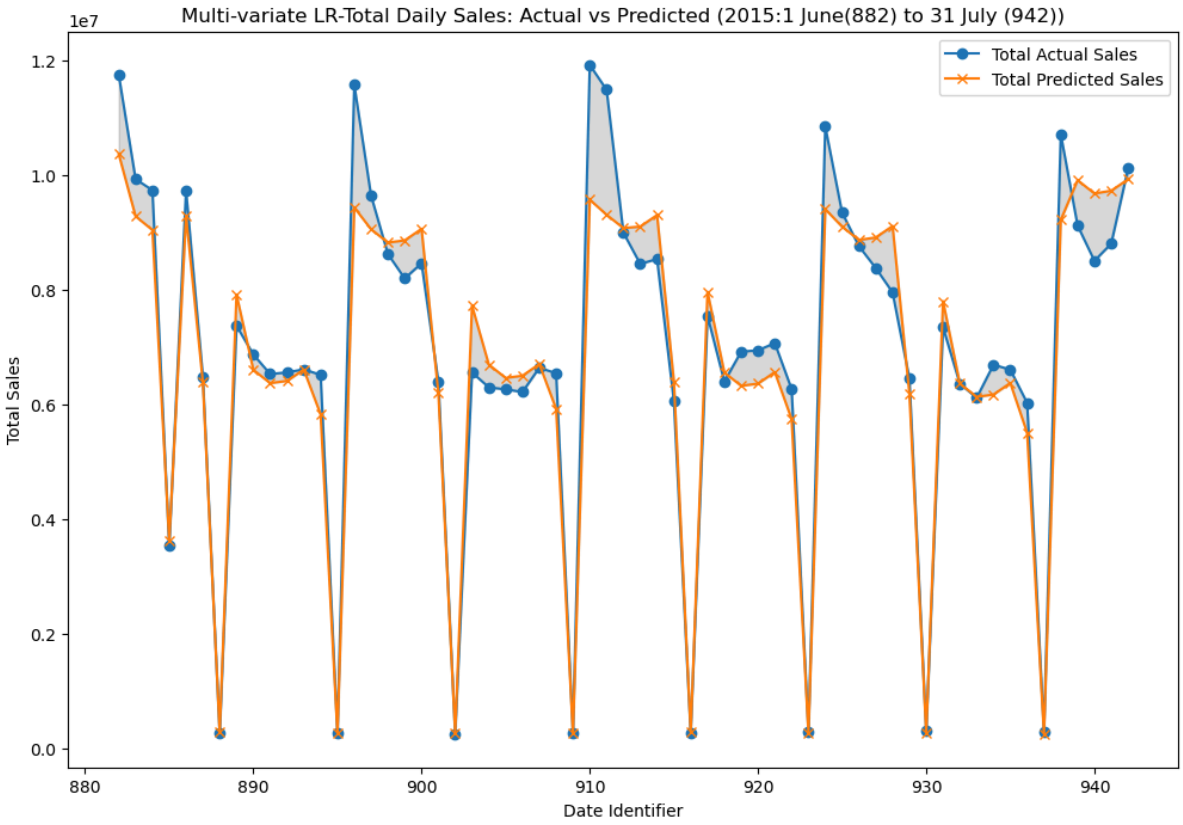
Omnibus:	259935.133	Durbin-Watson:	1.352
Prob(Omnibus):	0.000	Jarque-Bera (JB):	2409464.305
Skew:	1.216	Prob(JB):	0.00
Kurtosis:	10.910	Cond. No.	1.28e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.28e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
Regression Equation:
Sales = -26154.38 + (1.00 * AvgSalesPerStore) + (3499.25 * Index_DayOfWeek)

```
+ (4881.88 * Index_Promo) + (246.49 * Index_StateHoliday) + (3313.16 * Inde
x_SchoolHoliday) + (-229.28 * Index_P2cal) + (7416.32 * Index_Year) + (469
3.75 * Index_WeekOfYear)
    DateIdentifier  Store   Sales  Predicted_Sales   Difference
48             942      1  5263.0      6785.729383 -1522.729383
49             941      1  5020.0      6605.723997 -1585.723997
50             940      1  4782.0      6571.436351 -1789.436351
51             939      1  5011.0      6779.207561 -1768.207561
52             938      1  6102.0      6156.485755   -54.485755
Mean Squared Error: 2055457.6584662928
R-squared: 0.7876424810996876
Root Mean Squared Percentage Error: 0.2175959074890229
```



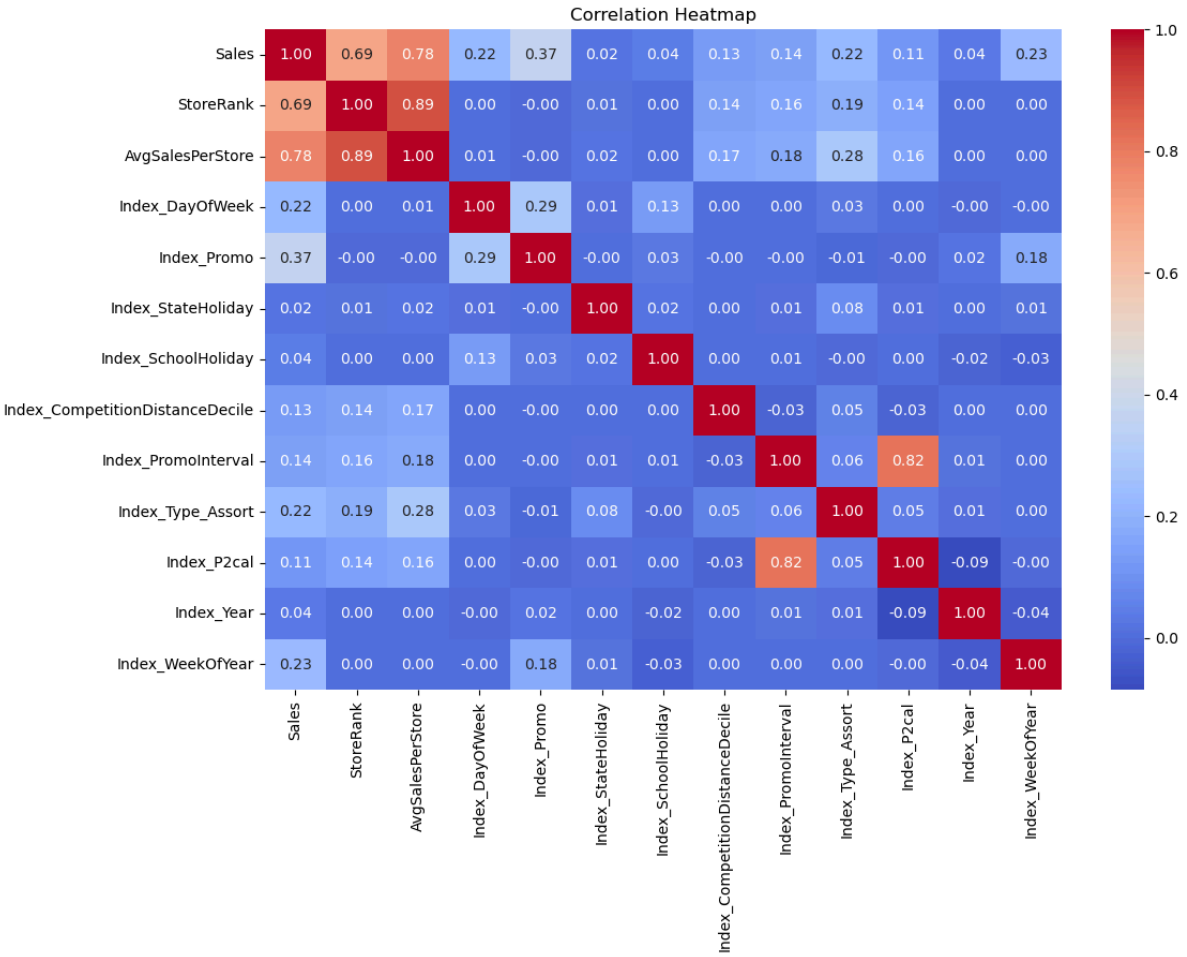

```
Total Sales Comparison (Actual vs Projected):
Total Actual Sales: 419685989.0
Total Predicted Sales: 411680410.5277975
Total Difference: 8005578.47220248
```



In [254…

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error, r2_score

# Assuming dft is your DataFrame
columns_to_correlate = [
    'Sales', 'StoreRank',  "AvgSalesPerStore",'Index_DayOfWeek', 'Index_Pror
    'Index_StateHoliday', 'Index_SchoolHoliday',
    'Index_CompetitionDistanceDecile', 'Index_PromoInterval',
    'Index_Type_Assort', 'Index_P2cal',
    'Index_Year', 'Index_WeekOfYear'

]

# Filter the DataFrame to include only the relevant columns for correlation
correlation_df = dft[columns_to_correlate]

# Calculate the correlation matrix
correlation_matrix = correlation_df.corr()

# Create a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()

# Define the dependent and independent variables for the regression model
X = dft[[
    'StoreRank',  "AvgSalesPerStore",'Index_DayOfWeek', 'Index_Promo',
    'Index_StateHoliday', 'Index_SchoolHoliday',
    'Index_CompetitionDistanceDecile', 'Index_PromoInterval',
    'Index_Type_Assort', 'Index_P2cal',
    'Index_Year', 'Index_WeekOfYear'
]]
```

```
y = dft['Sales']

# Add a constant to the independent variables (intercept)
X = sm.add_constant(X)

# Fit the linear regression model
model = sm.OLS(y, X).fit()

# Print the summary of the regression model
print(model.summary())

# Get the model parameters (coefficients)
params = model.params

# Create the regression equation as a string
equation = f'Sales = {params["const"]:.2f}'
for col in X.columns[1:]:
    equation += f' + ({params[col]:.2f} * {col})'

print("Regression Equation:")
print(equation)
```



OLS Regression Results

Dep. Variable:	Sales	R-squared:	0.785
Model:	OLS	Adj. R-squared:	0.785
Method:	Least Squares	F-statistic:	2.572e+05
Date:	Tue, 04 Jun 2024	Prob (F-statistic):	0.00
Time:	17:26:41	Log-Likelihood:	-7.3375e+06
No. Observations:	844338	AIC:	1.468e+07
Df Residuals:	844325	BIC:	1.468e+07
Df Model:	12		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-2.589e+04	231.833	-111.693	0.000	-2.63e+04	-2.54e+04
StoreRank	-0.0069	0.011	-0.631	0.528	-0.028	0.014
AvgSalesPerStore	1.0015	0.001	668.664	0.000	0.999	1.004
Index_DayOfWeek	3497.9529	14.186	246.578	0.000	3470.149	3525.757
Index_Promo	4882.4652	8.733	559.058	0.000	4865.348	4899.582
Index_StateHoliday	207.5126	170.469	1.217	0.223	-126.600	541.626
Index_SchoolHoliday	3312.8685	104.112	31.820	0.000	3108.813	3516.924
Index_CompetitionDistanceDecile	-171.1323	26.028	-6.575	0.000	-222.146	-120.119
Index_PromoInterval	198.2221	40.963	4.839	0.000	117.936	278.509
Index_Type_Assort	39.8981	15.665	2.547	0.011	9.195	70.601
Index_P2cal	-442.3948	50.992	-8.676	0.000	-542.338	-342.452
Index_Year	7338.2877	106.881	68.658	0.000	7128.804	7547.771
Index_WeekOfYear	4692.6688	13.350	351.499	0.000	4666.502	4718.835

Omnibus:	260158.273	Durbin-Watson:	1.352
Prob(Omnibus):	0.000	Jarque-Bera (JB):	2410124.612
Skew:	1.218	Prob(JB):	0.00
Kurtosis:	10.910	Cond. No.	1.29e+06

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is corr
ectly specified.
[2] The condition number is large, 1.29e+06. This might indicate that there
are
strong multicollinearity or other numerical problems.
Regression Equation:
Sales = -25894.10 + (-0.01 * StoreRank) + (1.00 * AvgSalesPerStore) + (349
7.95 * Index_DayOfWeek) + (4882.47 * Index_Promo) + (207.51 * Index_StateHo
liday) + (3312.87 * Index_SchoolHoliday) + (-171.13 * Index_CompetitionDist
anceDecile) + (198.22 * Index_PromoInterval) + (39.90 * Index_Type_Assort)
+ (-442.39 * Index_P2cal) + (7338.29 * Index_Year) + (4692.67 * Index_WeekO
fYear)
```

In [255…

```
# Step tail
dft.tail(7).T
```

Out[255]:

	1058288	1058289	1058290	1058292	
Sales	4897.000000	5183.000000	8002.000000	3156.000000	39
Store	1110.000000	1110.000000	1110.000000	1110.000000	1
DayOfWeek	3.000000	2.000000	1.000000	6.000000	
Open	1.000000	1.000000	1.000000	1.000000	
Promo	1.000000	1.000000	1.000000	0.000000	
StateHoliday	3.000000	3.000000	3.000000	3.000000	
SchoolHoliday	0.000000	0.000000	0.000000	0.000000	
CompetitionDistance	900.000000	900.000000	900.000000	900.000000	9
PromoInterval	0.000000	0.000000	0.000000	0.000000	
CompetitionDistanceDecile	3.000000	3.000000	3.000000	3.000000	
Type_Assort	6.000000	6.000000	6.000000	6.000000	
PromoRun	0.000000	0.000000	0.000000	0.000000	
DateIdentifier	9.000000	8.000000	7.000000	5.000000	
P2cal	0.000000	0.000000	0.000000	0.000000	
WeekNumber	2.000000	2.000000	1.000000	1.000000	
Year	2013.000000	2013.000000	2013.000000	2013.000000	2(
WeekOfYear	2.000000	2.000000	1.000000	1.000000	
AvgSalesPerStore	4520.365753	4520.365753	4520.365753	4520.365753	45
StoreRank	124.000000	124.000000	124.000000	124.000000	
Index_DayOfWeek	1.139140	1.198516	1.391329	1.000000	
Index_Promo	1.383223	1.383223	1.383223	1.000000	
Index_StateHoliday	1.000000	1.000000	1.000000	1.000000	
Index_SchoolHoliday	1.000000	1.000000	1.000000	1.000000	
Index_CompetitionDistance	1.883922	1.883922	1.883922	1.883922	
Index_CompetitionDistanceDecile	1.000000	1.000000	1.000000	1.000000	
Index_PromoInterval	1.182367	1.182367	1.182367	1.182367	
Index_Type_Assort	1.094733	1.094733	1.094733	1.094733	
Index_PromoRun	1.672982	1.672982	1.672982	1.672982	
Index_P2cal	1.110357	1.110357	1.110357	1.110357	
Index_Year	1.000000	1.000000	1.000000	1.000000	
Index_WeekOfYear	1.181268	1.181268	1.113700	1.113700	

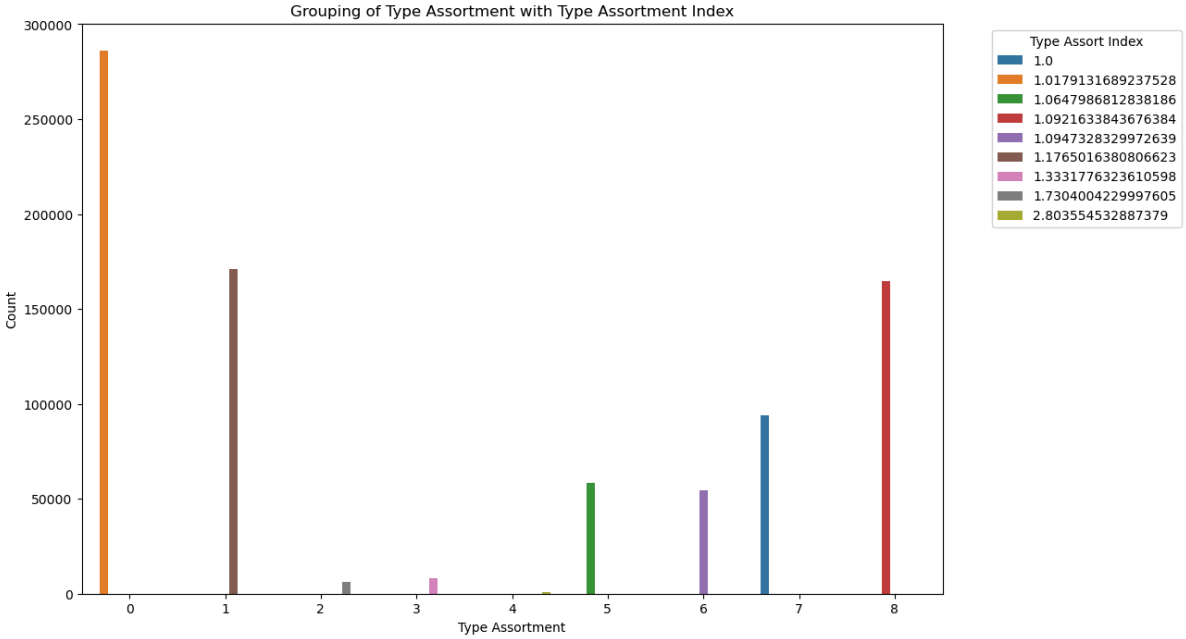
In [256…

```
# Step info
dft.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 844338 entries, 48 to 1058295
Data columns (total 31 columns):
 #   Column                            Non-Null Count   Dtype
---  ------                            --------------   -----
 0   Sales                             844338 non-null  float64
 1   Store                             844338 non-null  int64
 2   DayOfWeek                         844338 non-null  int64
 3   Open                              844338 non-null  float64
 4   Promo                             844338 non-null  int64
 5   StateHoliday                      844338 non-null  int64
 6   SchoolHoliday                     844338 non-null  int64
 7   CompetitionDistance               844338 non-null  float64
 8   PromoInterval                     844338 non-null  float64
 9   CompetitionDistanceDecile         844338 non-null  int64
 10  Type_Assort                       844338 non-null  int64
 11  PromoRun                          844338 non-null  float64
 12  DateIdentifier                    844338 non-null  int64
 13  P2cal                             844338 non-null  int64
 14  WeekNumber                        844338 non-null  int64
 15  Year                              844338 non-null  int64
 16  WeekOfYear                        844338 non-null  int64
 17  AvgSalesPerStore                  844338 non-null  float64
 18  StoreRank                         844338 non-null  float64
 19  Index_DayOfWeek                   844338 non-null  float64
 20  Index_Promo                       844338 non-null  float64
 21  Index_StateHoliday                844338 non-null  float64
 22  Index_SchoolHoliday               844338 non-null  float64
 23  Index_CompetitionDistance         844338 non-null  float64
 24  Index_CompetitionDistanceDecile   844338 non-null  float64
 25  Index_PromoInterval               844338 non-null  float64
 26  Index_Type_Assort                 844338 non-null  float64
 27  Index_PromoRun                    844338 non-null  float64
 28  Index_P2cal                       844338 non-null  float64
 29  Index_Year                        844338 non-null  float64
 30  Index_WeekOfYear                  844338 non-null  float64
dtypes: float64(19), int64(12)
memory usage: 206.1 MB
```

In [257…

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

# Assuming dfzc881 is your DataFrame

# Create a count plot to visualize the grouping of Type_Assort with Type_Ass
plt.figure(figsize=(12, 8))
sns.countplot(x='Type_Assort', hue='Index_Type_Assort', data=dft)

# Set plot labels and title
plt.xlabel('Type Assortment')
plt.ylabel('Count')
plt.title('Grouping of Type Assortment with Type Assortment Index')

# Show legend
plt.legend(title='Type Assort Index', bbox_to_anchor=(1.05, 1), loc='upper

# Show plot
plt.show()
```



In [258…
```
# List of columns to drop
columns_to_drop = ['Promo', 'StateHoliday', 'SchoolHoliday', 'CompetitionDis
                   'PromoInterval' , 'PromoRun', 'P2cal',
                   'WeekNumber', 'Year', 'WeekOfYear']

# Drop the columns from the DataFrame
df.drop(columns=columns_to_drop, inplace=True)

# Print the updated DataFrame
df.head()
```

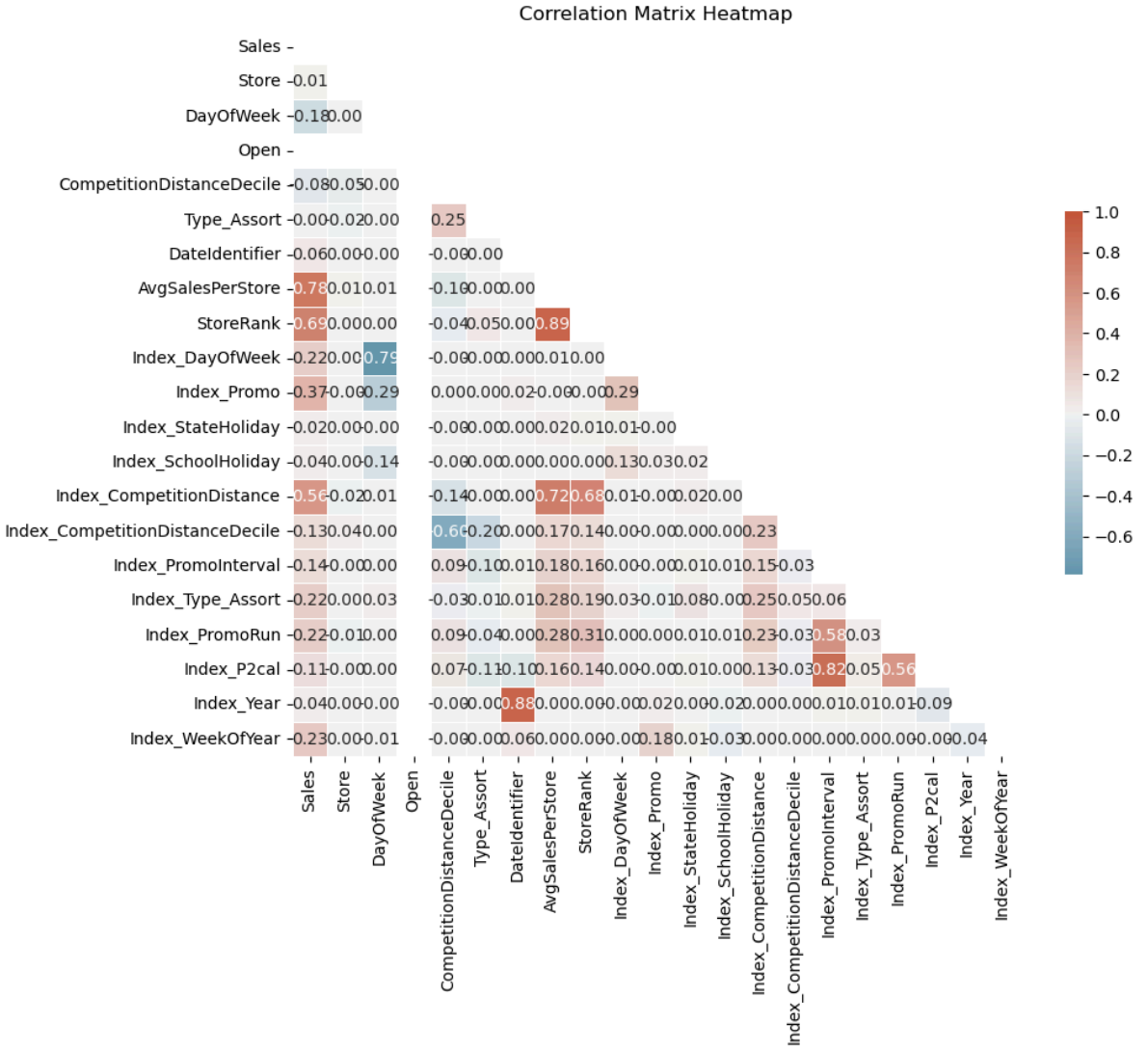
Out[258]:

	Sales	Store	DayOfWeek	Open	CompetitionDistanceDecile	Type_Assort	DateIdentif
48	5263.0	1	5	1.0	4	5	9
49	5020.0	1	4	1.0	4	5	9
50	4782.0	1	3	1.0	4	5	9
51	5011.0	1	2	1.0	4	5	9
52	6102.0	1	1	1.0	4	5	9

5 rows × 21 columns

In [259…
```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Assuming df is your DataFrame
numeric_df = df.select_dtypes(include=['number'])

# Calculate the correlation matrix
correlation_matrix = numeric_df.corr()

# Create a mask to display only one half of the matrix
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))

# Set up the matplotlib figure
plt.figure(figsize=(10, 8))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)
```

```
# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(correlation_matrix, mask=mask, cmap=cmap, vmax=1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True,

plt.title('Correlation Matrix Heatmap')
plt.show()
```



In [260…

```
# Filter DataFrame to include only rows with DateIdentifier between 1 and 94
dfzc881 = dfzc[(dfzc['DateIdentifier'] >= 1) & (dfzc['DateIdentifier'] <= 88

dfzc881.T
```

Out[260]:

	110	111	112	113	114	117	118	119
Sales	5592.0	4656.0	4111.0	4083.0	4211.0	4276.0	4459.0	3755.0
Store	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
DayOfWeek	6.0	5.0	4.0	3.0	2.0	6.0	5.0	4.0
Open	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Promo	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0
StateHoliday	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0
SchoolHoliday	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
CompetitionDistance	1270.0	1270.0	1270.0	1270.0	1270.0	1270.0	1270.0	1270.0
PromoInterval	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
CompetitionDistanceDecile	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0
Type_Assort	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0
PromoRun	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DateIdentifier	880.0	879.0	878.0	877.0	876.0	873.0	872.0	871.0
P2cal	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
WeekNumber	126.0	126.0	126.0	126.0	126.0	125.0	125.0	125.0
Year	2015.0	2015.0	2015.0	2015.0	2015.0	2015.0	2015.0	2015.0
WeekOfYear	22.0	22.0	22.0	22.0	22.0	21.0	21.0	21.0

17 rows × 785727 columns

In [261…

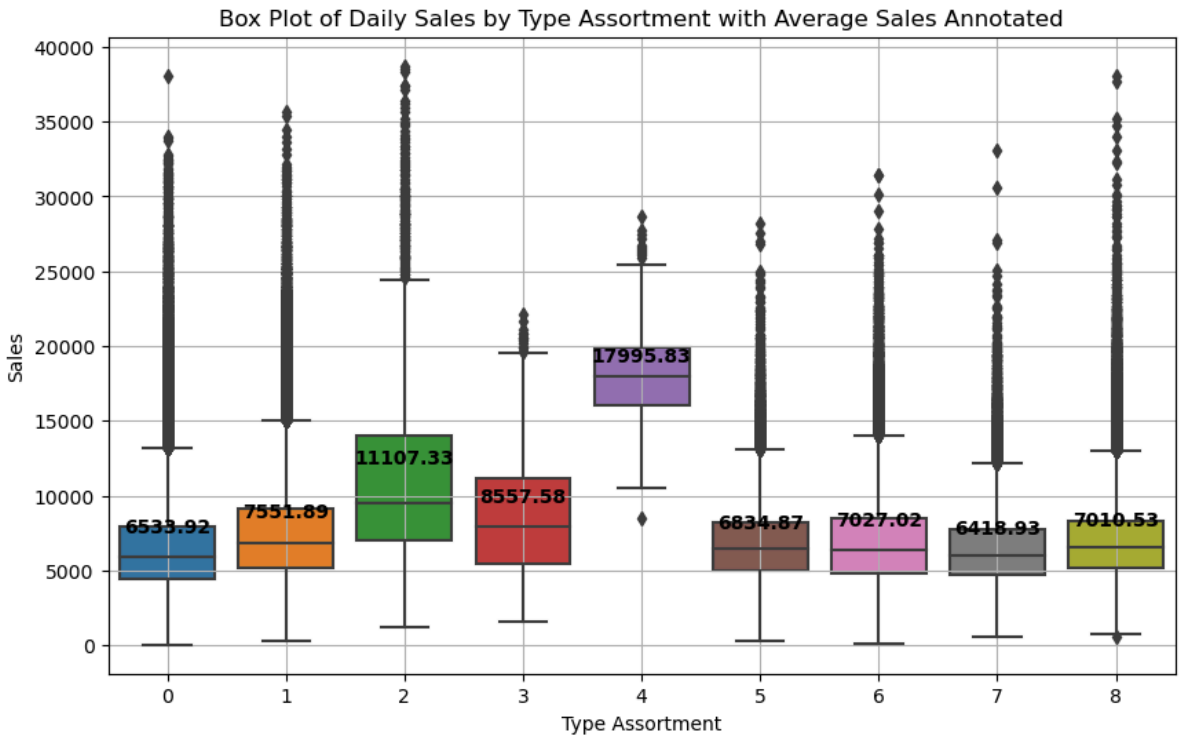
```
# Step info
dfzc881.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 785727 entries, 110 to 1058295
Data columns (total 17 columns):
 #   Column                     Non-Null Count   Dtype  
---  ------                     --------------   -----  
 0   Sales                      785727 non-null  float64
 1   Store                      785727 non-null  int64  
 2   DayOfWeek                  785727 non-null  int64  
 3   Open                       785727 non-null  float64
 4   Promo                      785727 non-null  int64  
 5   StateHoliday               785727 non-null  int64  
 6   SchoolHoliday              785727 non-null  int64  
 7   CompetitionDistance        785727 non-null  float64
 8   PromoInterval              785727 non-null  float64
 9   CompetitionDistanceDecile  785727 non-null  int64  
 10  Type_Assort                785727 non-null  int64  
 11  PromoRun                   785727 non-null  float64
 12  DateIdentifier             785727 non-null  int64  
 13  P2cal                      785727 non-null  int64  
 14  WeekNumber                 785727 non-null  int64  
 15  Year                       785727 non-null  int64  
 16  WeekOfYear                 785727 non-null  int64  
dtypes: float64(5), int64(12)
memory usage: 124.0 MB
```

In [262…

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

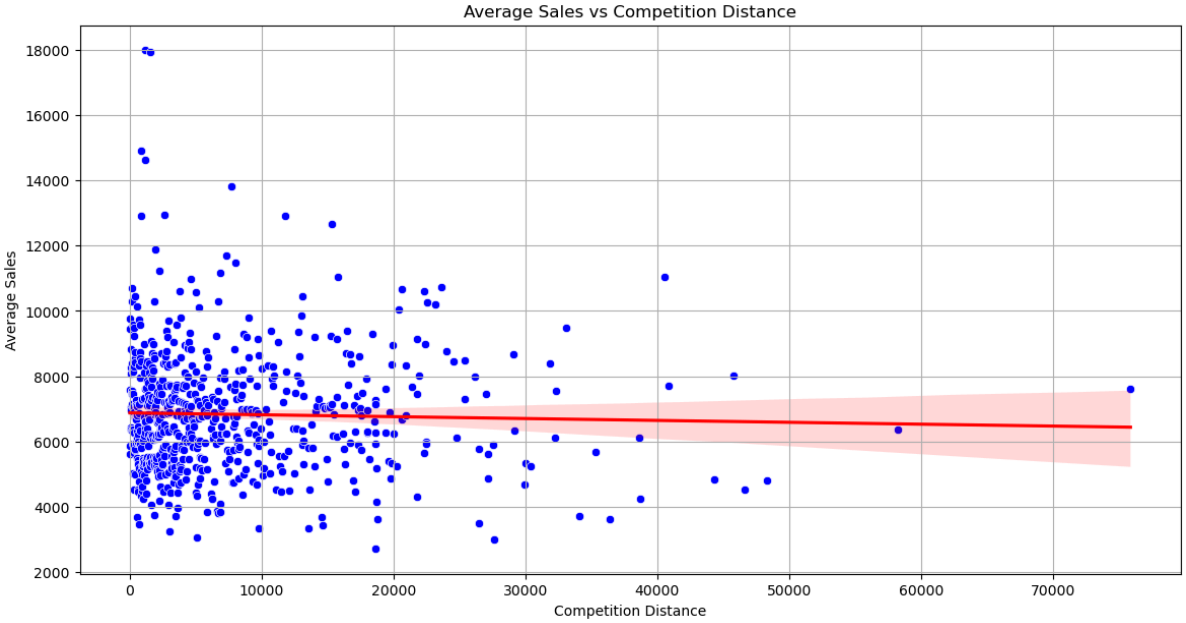
```
import pandas as pd

# Assuming dfzc881 is your DataFrame

# Calculate the average daily sales for each Type_Assort category
avg_sales_per_type = dfzc881.groupby('Type_Assort')['Sales'].mean().reset_ir

# Create the box plot
plt.figure(figsize=(10, 6))
sns.boxplot(x='Type_Assort', y='Sales', data=dfzc881)

# Annotate the box plot with the average sales values
for i, row in avg_sales_per_type.iterrows():
    plt.text(x=i, y=row['Sales'] + 1000, s=f"{row['Sales']:.2f}", horizonta

# Set plot labels and title
plt.xlabel('Type Assortment')
plt.ylabel('Sales')
plt.title('Box Plot of Daily Sales by Type Assortment with Average Sales Anr
plt.grid(True)

# Show plot
plt.show()
```



In [263…

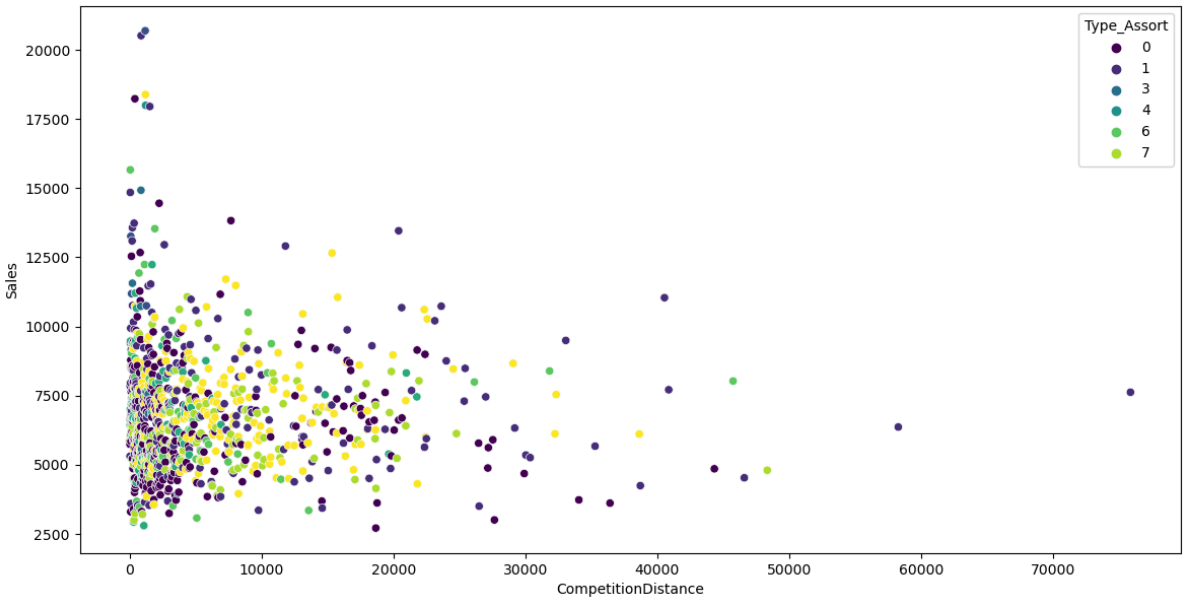
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming dfzv881 is your DataFrame
df = dfzc881

# Calculate the average sales by CompetitionDistance
avg_sales_by_distance = df.groupby('CompetitionDistance')['Sales'].mean().re

# Plotting the scatter plot
plt.figure(figsize=(14, 7))
sns.scatterplot(x='CompetitionDistance', y='Sales', data=avg_sales_by_distar

# Add a regression line
sns.regplot(x='CompetitionDistance', y='Sales', data=avg_sales_by_distance,
```

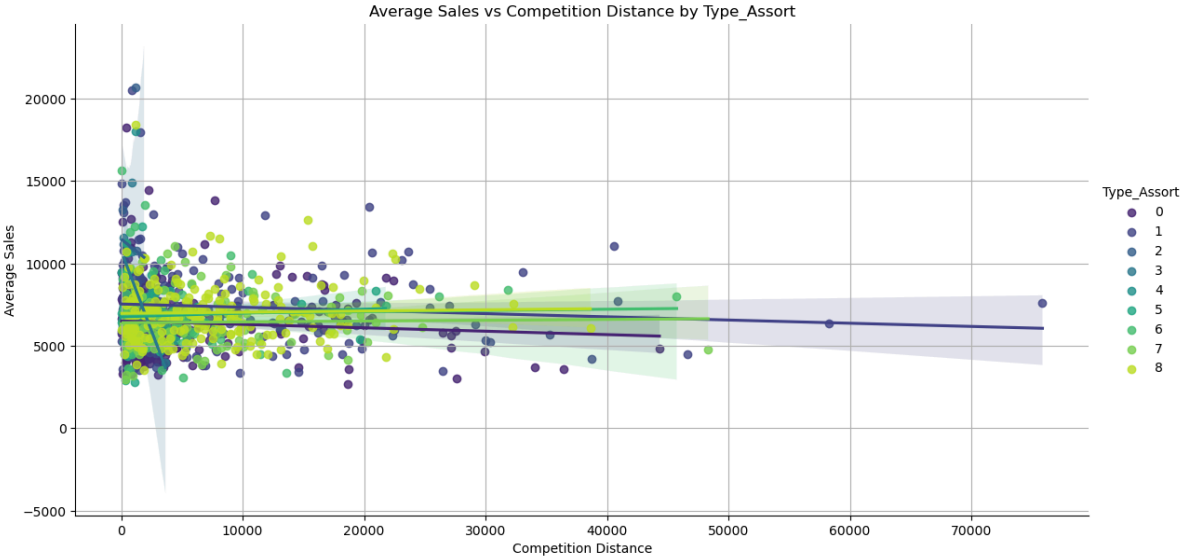
```
# Set titles and labels
plt.title('Average Sales vs Competition Distance')
plt.xlabel('Competition Distance')
plt.ylabel('Average Sales')

# Show plot
plt.grid(True)
plt.show()
```



In [264…

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming dfzv881 is your DataFrame
df = dfzc881

# Calculate the average sales by CompetitionDistance and Type_Assort
avg_sales_by_distance_type = df.groupby(['CompetitionDistance', 'Type_Assor

# Plotting the scatter plot with hue for Type_Assort
plt.figure(figsize=(14, 7))
sns.scatterplot(x='CompetitionDistance', y='Sales', hue='Type_Assort', data=

# Add a regression line for each Type_Assort
sns.lmplot(x='CompetitionDistance', y='Sales', hue='Type_Assort', data=avg_s

# Set titles and labels
plt.title('Average Sales vs Competition Distance by Type_Assort')
plt.xlabel('Competition Distance')
plt.ylabel('Average Sales')

# Show plot
plt.grid(True)
plt.show()
```

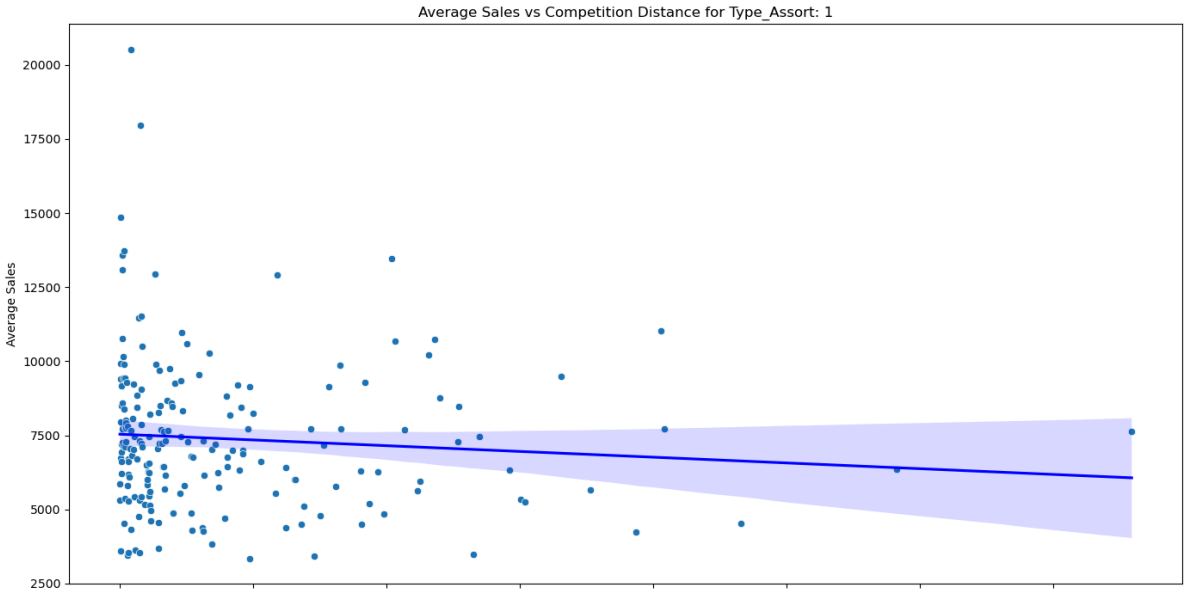


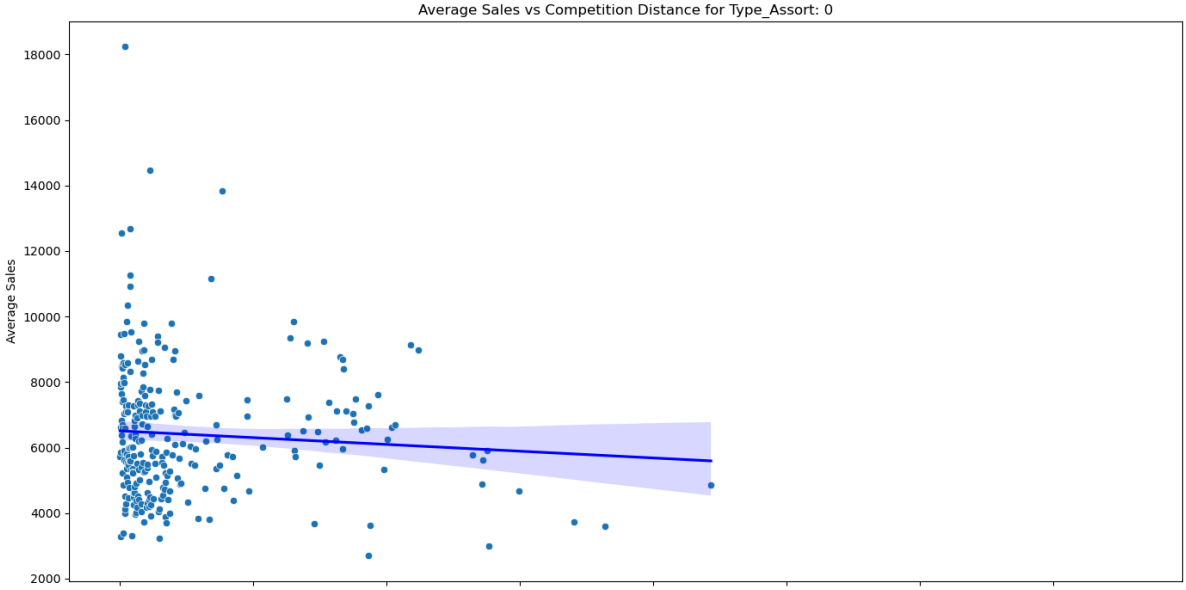


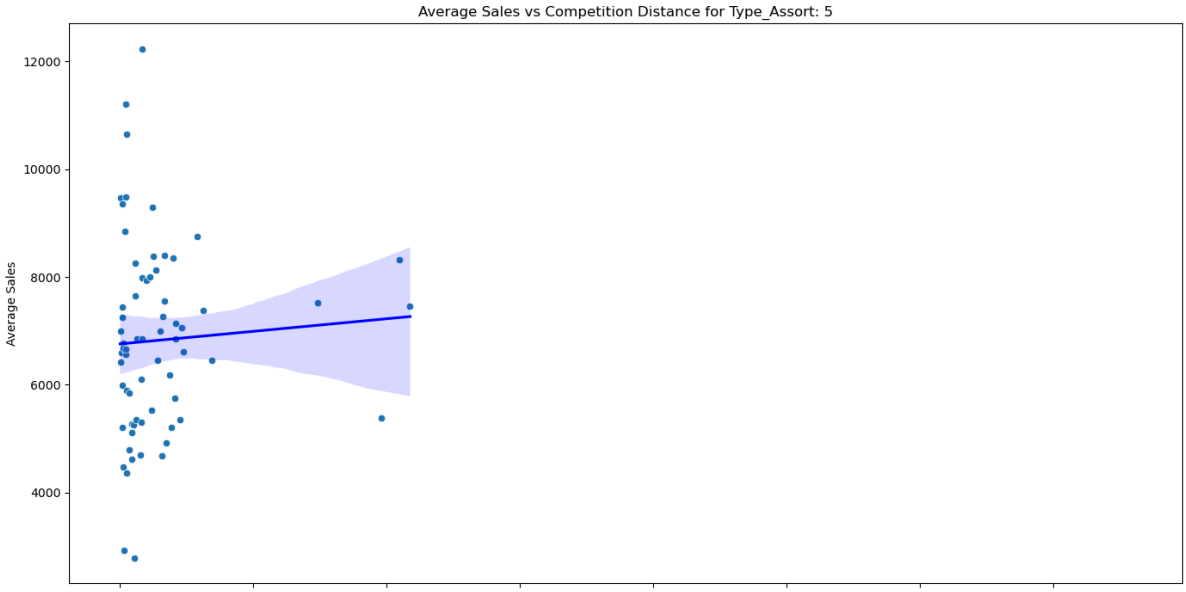
In [265…

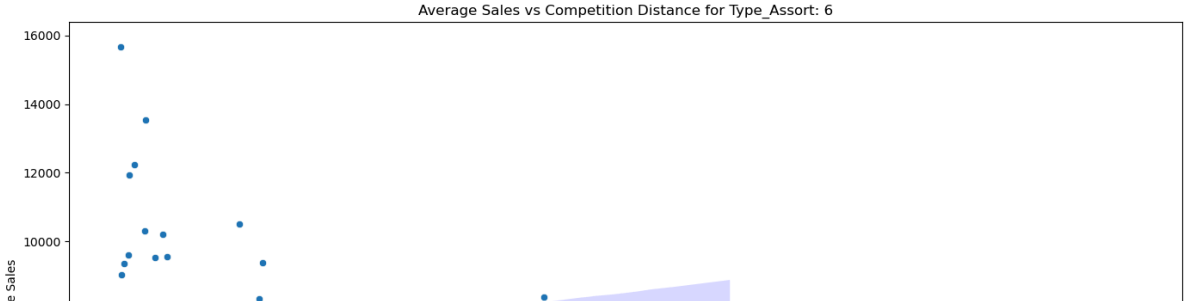
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming dfzv881 is your DataFrame
df = dfzc881

# Calculate the average sales by CompetitionDistance and Type_Assort
avg_sales_by_distance_type = df.groupby(['CompetitionDistance', 'Type_Assort

# Get the unique Type_Assort values
type_assorts = avg_sales_by_distance_type['Type_Assort'].unique()

# Plotting separate scatter plots for each Type_Assort
fig, axes = plt.subplots(len(type_assorts), 1, figsize=(14, 7 * len(type_ass

for i, type_assort in enumerate(type_assorts):
    subset = avg_sales_by_distance_type[avg_sales_by_distance_type['Type_Ass
    sns.scatterplot(x='CompetitionDistance', y='Sales', data=subset, ax=axes
    sns.regplot(x='CompetitionDistance', y='Sales', data=subset, ax=axes[i],
    axes[i].set_title(f'Average Sales vs Competition Distance for Type_Asso
    axes[i].set_xlabel('Competition Distance')
    axes[i].set_ylabel('Average Sales')

# Show plot
```

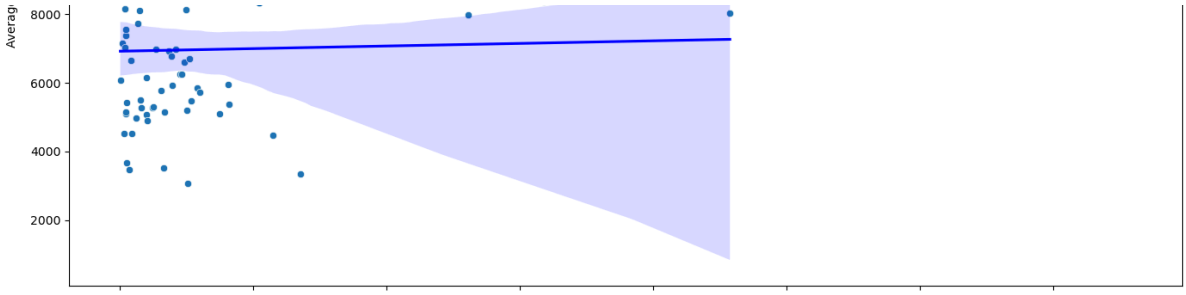
```
plt.tight_layout()
plt.show()
```

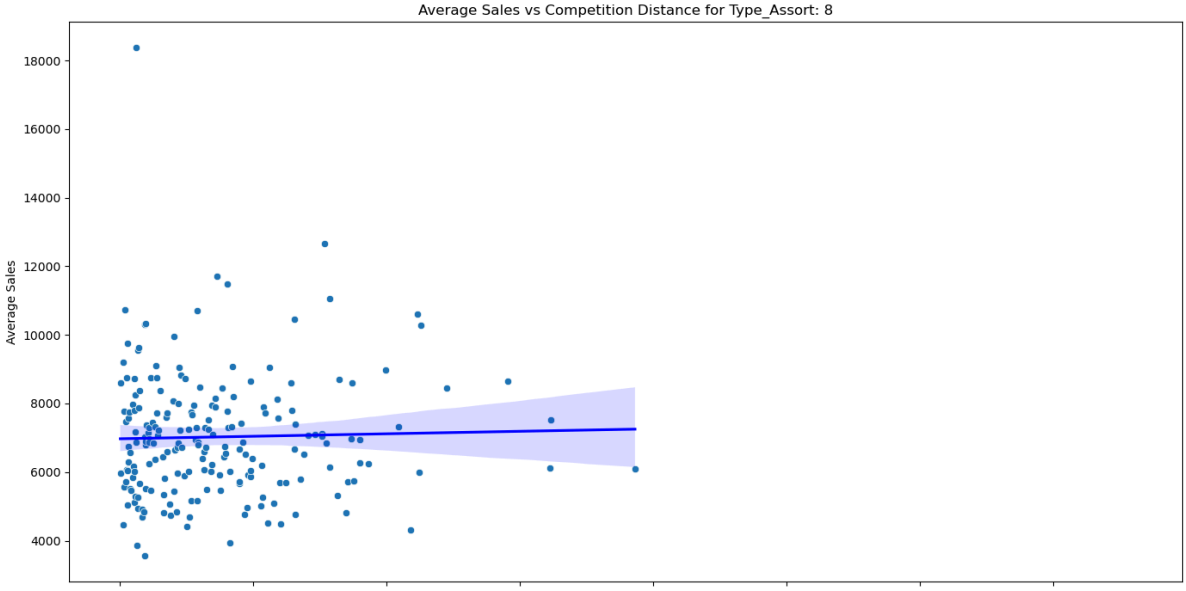

Average Sales vs Competition Distance for Type_Assort: 1
Average Sales
20000
17500
15000
12500
10000
7500
5000
2500

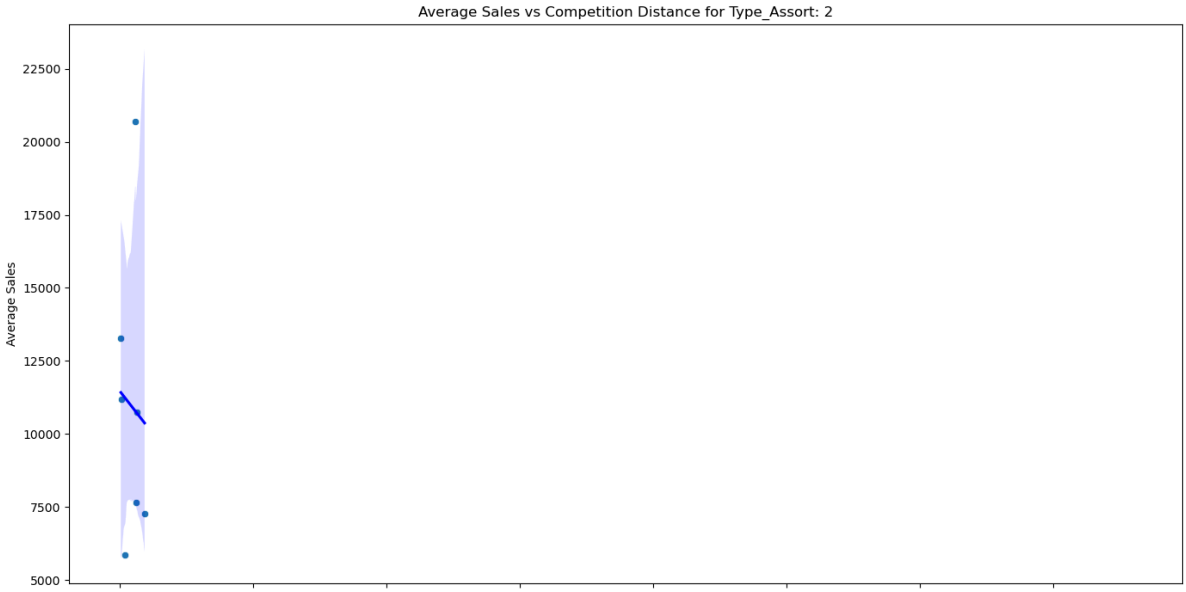

Average Sales vs Competition Distance for Type_Assort: 0
Average Sales
18000
16000
14000
12000
10000
8000
6000
4000
2000

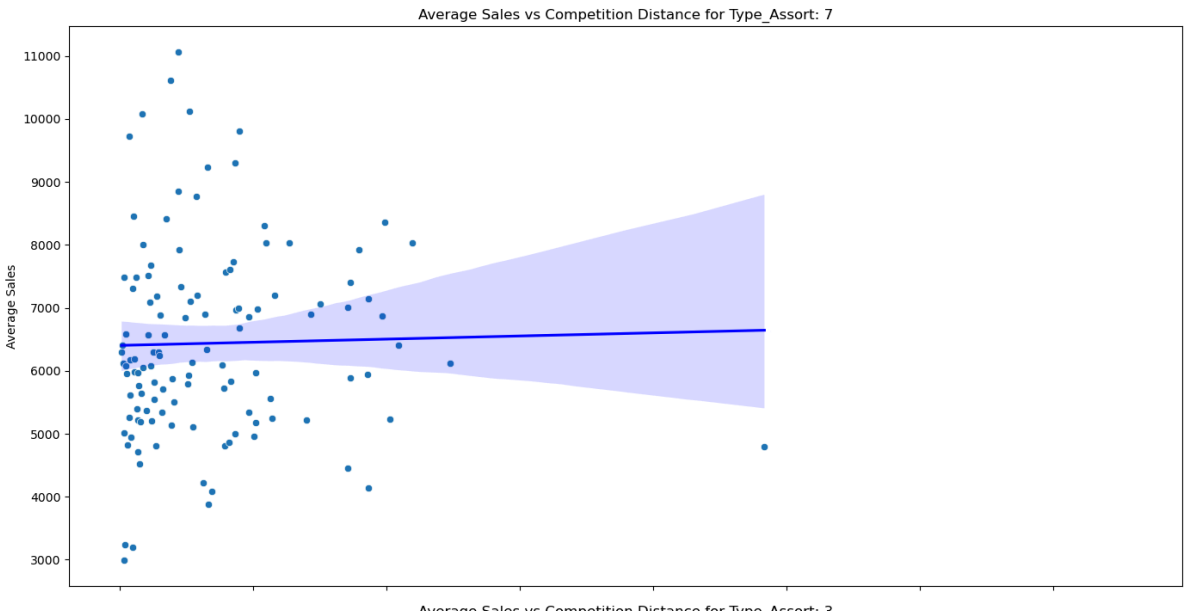

Average Sales vs Competition Distance for Type_Assort: 5
Average Sales
12000
10000
8000
6000
4000


Average Sales vs Competition Distance for Type_Assort: 6
16000
14000
12000
10000


Averag
8000
6000
4000
2000


Average Sales vs Competition Distance for Type_Assort: 8
Average Sales
18000
16000
14000
12000
10000
8000
6000
4000


Average Sales vs Competition Distance for Type_Assort: 2
Average Sales
22500
20000
17500
15000
12500
10000
7500
5000


Average Sales vs Competition Distance for Type_Assort: 7
Average Sales
11000
10000
9000
8000
7000
6000
5000
4000
3000

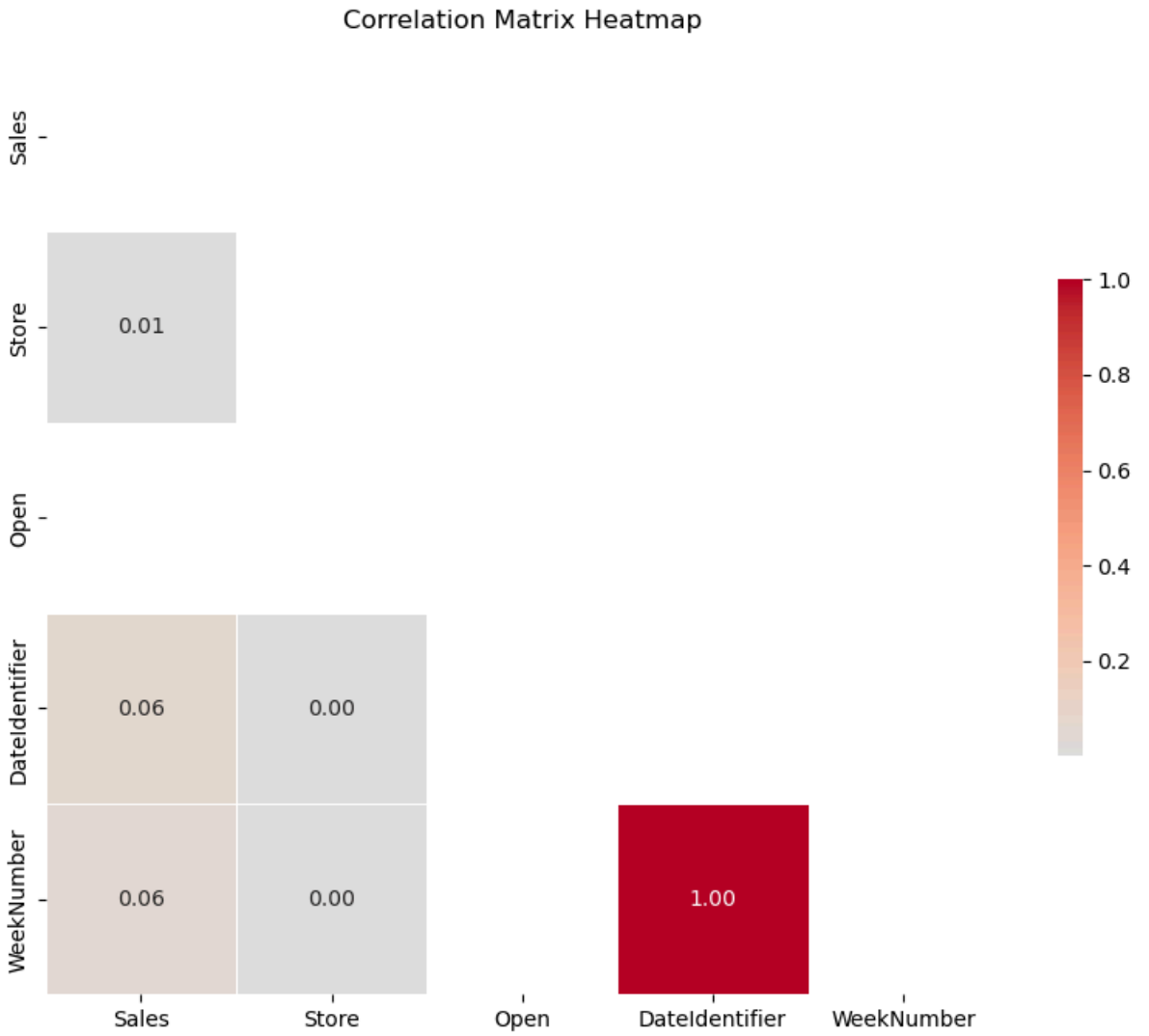
In [266…

```
# List of columns to remove
cols_to_remove = ['DayOfWeek', 'Promo', 'StateHoliday', 'SchoolHoliday',
                  'CompetitionDistance', 'CompetitionDistanceDecile',
                  'PromoInterval', 'Type_Assort', 'PromoRun', 'P2cal',
                  'Year', 'WeekOfYear']

# Create new DataFrame by dropping the specified columns
dfzn881 = dfzc881.drop(columns=cols_to_remove)

# Display the first few rows of the new DataFrame to confirm
dfzn881.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 785727 entries, 110 to 1058295
Data columns (total 5 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   Sales           785727 non-null  float64
 1   Store           785727 non-null  int64  
 2   Open            785727 non-null  float64
 3   DateIdentifier  785727 non-null  int64  
 4   WeekNumber      785727 non-null  int64  
dtypes: float64(2), int64(3)
memory usage: 52.1 MB
```

In [267…

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Assuming df is your DataFrame
numeric_df = dfzn881.select_dtypes(include=['number'])

# Calculate the correlation matrix
correlation_matrix = numeric_df.corr()

# Create a mask to display only one half of the matrix
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))

# Set up the matplotlib figure
plt.figure(figsize=(10, 8))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(correlation_matrix, mask=mask, cmap="coolwarm", vmax=1, center=0
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True,

plt.title('Correlation Matrix Heatmap')
plt.show()
```



In [268…
```
# Filter DataFrame to include only rows with DateIdentifier between 882 and
dfzc61 = dfzc[(dfzc['DateIdentifier'] >= 882) & (dfzc['DateIdentifier'] <= 9

dfzc61
```

Out[268]:

	Sales	Store	DayOfWeek	Open	Promo	StateHoliday	SchoolHoliday	Competiti
48	5263.0	1	5	1.0	1	3	1	
49	5020.0	1	4	1.0	1	3	1	
50	4782.0	1	3	1.0	1	3	1	
51	5011.0	1	2	1.0	1	3	1	
52	6102.0	1	1	1.0	1	3	1	
...	...	...	...	...	...	...	...	
1057411	5433.0	1110	5	1.0	1	3	0	
1057412	5578.0	1110	4	1.0	1	3	0	
1057413	5794.0	1110	3	1.0	1	3	0	
1057414	6906.0	1110	2	1.0	1	3	0	
1057415	8293.0	1110	1	1.0	1	3	0	

58611 rows × 17 columns

In [269…
```
dfzc61.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 58611 entries, 48 to 1057415
Data columns (total 17 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Sales                      58611 non-null  float64
 1   Store                      58611 non-null  int64
 2   DayOfWeek                  58611 non-null  int64
 3   Open                       58611 non-null  float64
 4   Promo                      58611 non-null  int64
 5   StateHoliday               58611 non-null  int64
 6   SchoolHoliday              58611 non-null  int64
 7   CompetitionDistance        58611 non-null  float64
 8   PromoInterval              58611 non-null  float64
 9   CompetitionDistanceDecile  58611 non-null  int64
 10  Type_Assort                58611 non-null  int64
 11  PromoRun                   58611 non-null  float64
 12  DateIdentifier             58611 non-null  int64
 13  P2cal                      58611 non-null  int64
 14  WeekNumber                 58611 non-null  int64
 15  Year                       58611 non-null  int64
 16  WeekOfYear                 58611 non-null  int64
dtypes: float64(5), int64(12)
memory usage: 8.0 MB
```

In [270…
```
import pandas as pd

# Assuming dfzc61 is already defined and contains the 'Sales' column
total_sales = dfzc61['Sales'].sum()

print("Total Sales Sum:", total_sales)
```

```
Total Sales Sum: 419685989.0
```

In [271…
```
dfzc61.head(7).T
```

Out[271]:

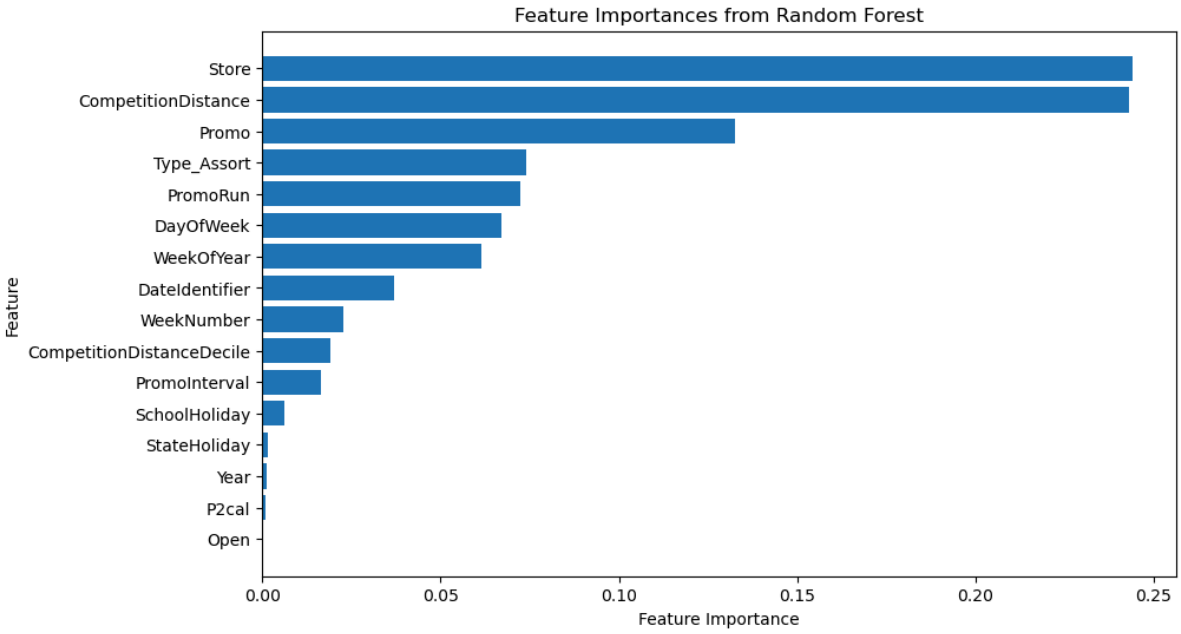
	48	49	50	51	52	54	55
Sales	5263.0	5020.0	4782.0	5011.0	6102.0	4364.0	3706.0
Store	1.0	1.0	1.0	1.0	1.0	1.0	1.0
DayOfWeek	5.0	4.0	3.0	2.0	1.0	6.0	5.0
Open	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Promo	1.0	1.0	1.0	1.0	1.0	0.0	0.0
StateHoliday	3.0	3.0	3.0	3.0	3.0	3.0	3.0
SchoolHoliday	1.0	1.0	1.0	1.0	1.0	0.0	0.0
CompetitionDistance	1270.0	1270.0	1270.0	1270.0	1270.0	1270.0	1270.0
PromoInterval	0.0	0.0	0.0	0.0	0.0	0.0	0.0
CompetitionDistanceDecile	4.0	4.0	4.0	4.0	4.0	4.0	4.0
Type_Assort	5.0	5.0	5.0	5.0	5.0	5.0	5.0
PromoRun	0.0	0.0	0.0	0.0	0.0	0.0	0.0
DateIdentifier	942.0	941.0	940.0	939.0	938.0	936.0	935.0
P2cal	0.0	0.0	0.0	0.0	0.0	0.0	0.0
WeekNumber	135.0	135.0	135.0	135.0	134.0	134.0	134.0
Year	2015.0	2015.0	2015.0	2015.0	2015.0	2015.0	2015.0
WeekOfYear	31.0	31.0	31.0	31.0	30.0	30.0	30.0

In [272…

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import pandas as pd

# Ensure that dfzc881 and dfzc61 are defined and loaded appropriately

# Separate features (X) and target variable (y) for the first DataFrame
X_train = dfzc881.drop(columns=['Sales'])
y_train = dfzc881['Sales']

# Separate features (X) and target variable (y) for the second DataFrame
X_test = dfzc61.drop(columns=['Sales'])
y_test = dfzc61['Sales']

# Initialize the Random Forest regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model on the first DataFrame
rf_model.fit(X_train, y_train)

# Make predictions on the second DataFrame
y_pred = rf_model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Get feature importances
feature_importances = rf_model.feature_importances_

# Create a DataFrame for feature importances
```

```
features_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': feature_importances
})

# Sort by importance
features_df = features_df.sort_values(by='Importance', ascending=False)

# Calculate RMSPE
rmspe = np.sqrt(np.mean(((y_test - y_pred) / y_test) ** 2))

print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Root Mean Squared Percentage Error (RMSPE):", rmspe)
print("\nFeature Importances:\n", features_df)

# Optional: Plotting feature importances
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.barh(features_df['Feature'], features_df['Importance'])
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Feature Importances from Random Forest')
plt.gca().invert_yaxis()
plt.show()
```

```
Mean Squared Error: 1611692.2564780894
R-squared: 0.8334896525808704
Root Mean Squared Percentage Error (RMSPE): 0.18245869416153865

Feature Importances:
                       Feature  Importance
0                        Store    0.244173
6          CompetitionDistance    0.242997
3                        Promo    0.132525
9                  Type_Assort    0.073998
10                    PromoRun    0.072519
1                    DayOfWeek    0.066940
15                  WeekOfYear    0.061597
11              DateIdentifier    0.036846
13                  WeekNumber    0.022695
8    CompetitionDistanceDecile    0.019195
7                PromoInterval    0.016435
5                SchoolHoliday    0.006078
4                 StateHoliday    0.001659
14                        Year    0.001388
12                       P2cal    0.000954
2                         Open    0.000000
```

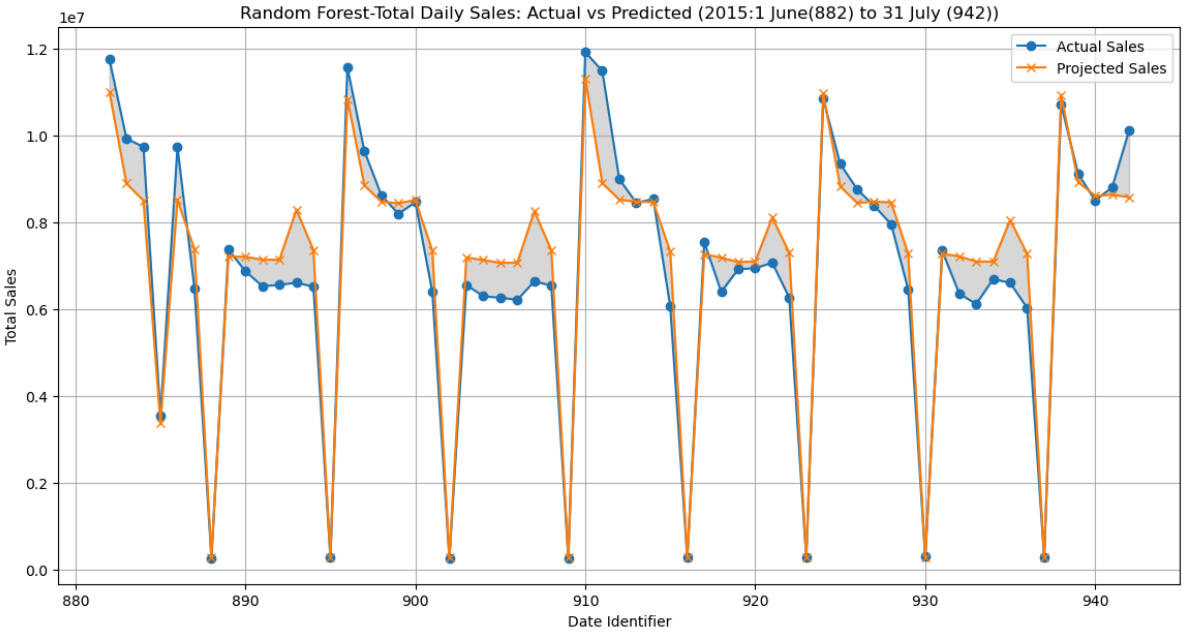


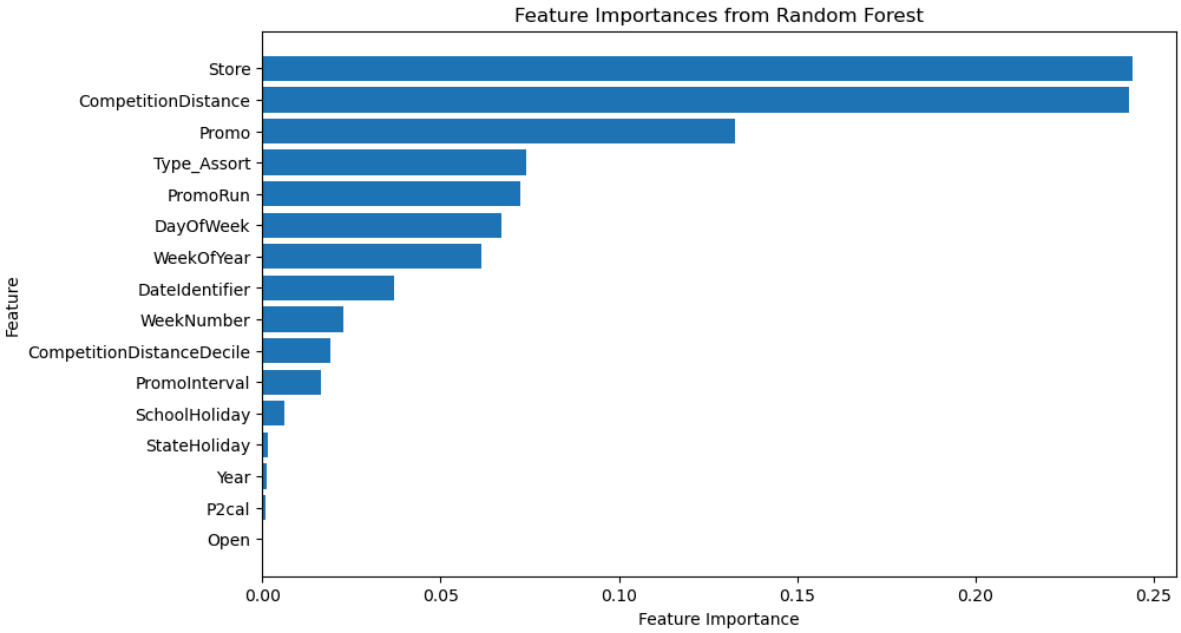
In [273…

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Ensure that dfzc881 and dfzc61 are defined and loaded appropriately

# Separate features (X) and target variable (y) for the first DataFrame
X_train = dfzc881.drop(columns=['Sales'])
y_train = dfzc881['Sales']

# Separate features (X) and target variable (y) for the second DataFrame
X_test = dfzc61.drop(columns=['Sales'])
y_test = dfzc61['Sales']

# Initialize the Random Forest regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model on the first DataFrame
rf_model.fit(X_train, y_train)

# Make predictions on the second DataFrame
y_pred = rf_model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Get feature importances
feature_importances = rf_model.feature_importances_

# Create a DataFrame for feature importances
features_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': feature_importances
})

# Sort by importance
features_df = features_df.sort_values(by='Importance', ascending=False)

# Calculate RMSPE
rmspe = np.sqrt(np.mean(((y_test - y_pred) / y_test) ** 2))
```

```
print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Root Mean Squared Percentage Error (RMSPE):", rmspe)
print("\nFeature Importances:\n", features_df)

# Create a DataFrame to compare actual and predicted sales
comparison_df = dfzc61[['DateIdentifier', 'Store', 'Sales']].copy()
comparison_df['Predicted_Sales'] = y_pred
comparison_df['Difference'] = comparison_df['Sales'] - comparison_df['Predic

# Show the first few rows of the comparison
print(comparison_df.head())

# Calculate total sales per day (projected vs actual)
daily_sales_comparison = comparison_df.groupby('DateIdentifier').agg({
    'Sales': 'sum',
    'Predicted_Sales': 'sum'
}).reset_index()
daily_sales_comparison['Difference'] = daily_sales_comparison['Sales'] - da

print("\nDaily Sales Comparison (Actual vs Projected):")
print(daily_sales_comparison.head())

# Optional: Plotting daily sales comparison
plt.figure(figsize=(14, 7))
plt.plot(daily_sales_comparison['DateIdentifier'], daily_sales_comparison['S
plt.plot(daily_sales_comparison['DateIdentifier'], daily_sales_comparison['P
plt.fill_between(daily_sales_comparison['DateIdentifier'], daily_sales_compa
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Random Forest-Total Daily Sales: Actual vs Predicted (2015:1 June
plt.legend()
plt.grid(True)
plt.show()

# Optional: Plotting feature importances
plt.figure(figsize=(10, 6))
plt.barh(features_df['Feature'], features_df['Importance'])
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Feature Importances from Random Forest')
plt.gca().invert_yaxis()
plt.show()


# Calculate total actual and predicted sales
total_actual_sales = comparison_df['Sales'].sum()
total_predicted_sales = comparison_df['Predicted_Sales'].sum()
total_difference = total_actual_sales - total_predicted_sales

print("\nTotal Sales Comparison (Actual vs Projected):")
print("Total Actual Sales:", total_actual_sales)
print("Total Predicted Sales:", total_predicted_sales)
print("Total Difference:", total_difference)
```

```
Mean Squared Error: 1611692.2564780894
R-squared: 0.8334896525808704
Root Mean Squared Percentage Error (RMSPE): 0.18245869416153865

Feature Importances:
                      Feature  Importance
0                       Store    0.244173
6         CompetitionDistance    0.242997
3                       Promo    0.132525
9                  Type_Assort    0.073998
10                    PromoRun    0.072519
1                    DayOfWeek    0.066940
15                  WeekOfYear    0.061597
11             DateIdentifier    0.036846
13                  WeekNumber    0.022695
8   CompetitionDistanceDecile    0.019195
7                PromoInterval    0.016435
5                SchoolHoliday    0.006078
4                 StateHoliday    0.001659
14                        Year    0.001388
12                       P2cal    0.000954
2                         Open    0.000000
    DateIdentifier  Store   Sales  Predicted_Sales  Difference
48             942      1  5263.0          4805.17      457.83
49             941      1  5020.0          4767.73      252.27
50             940      1  4782.0          5098.43     -316.43
51             939      1  5011.0          5187.46     -176.46
52             938      1  6102.0          5645.63      456.37

Daily Sales Comparison (Actual vs Projected):
   DateIdentifier       Sales  Predicted_Sales  Difference
0             882  11743615.0      10993277.86   750337.14
1             883   9925193.0       8887523.14  1037669.86
2             884   9731791.0       8495150.03  1236640.97
3             885   3534231.0       3365738.29   168492.71
4             886   9724893.0       8506200.47  1218692.53
```





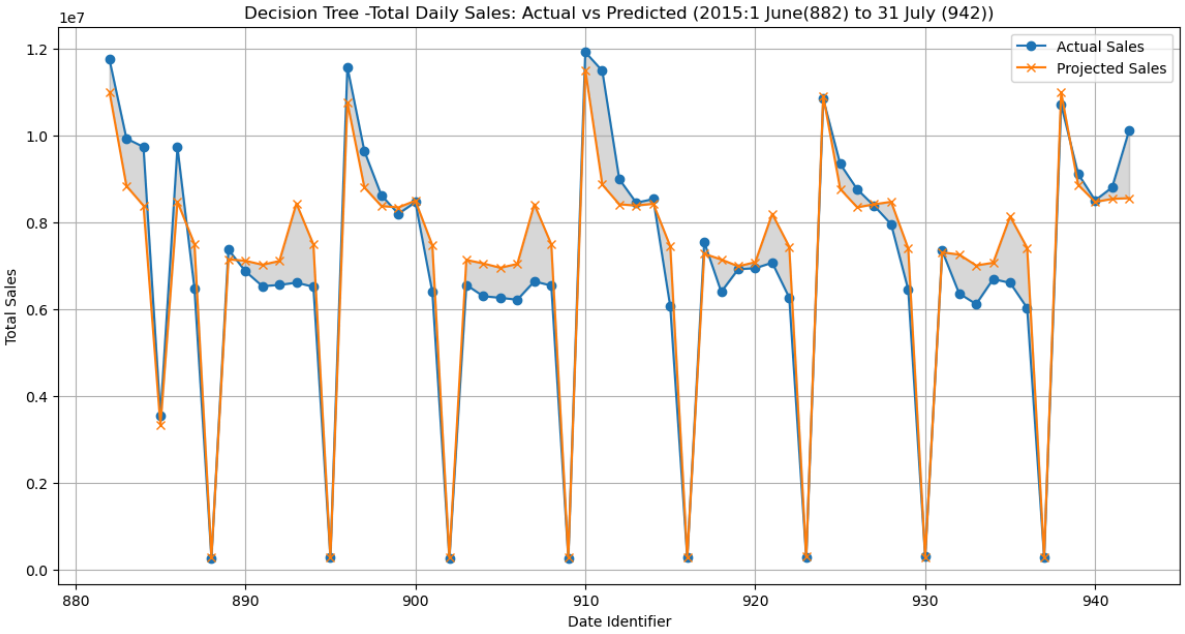

```
Total Sales Comparison (Actual vs Projected):
Total Actual Sales: 419685989.0
Total Predicted Sales: 429732721.65000004
Total Difference: -10046732.650000036
```

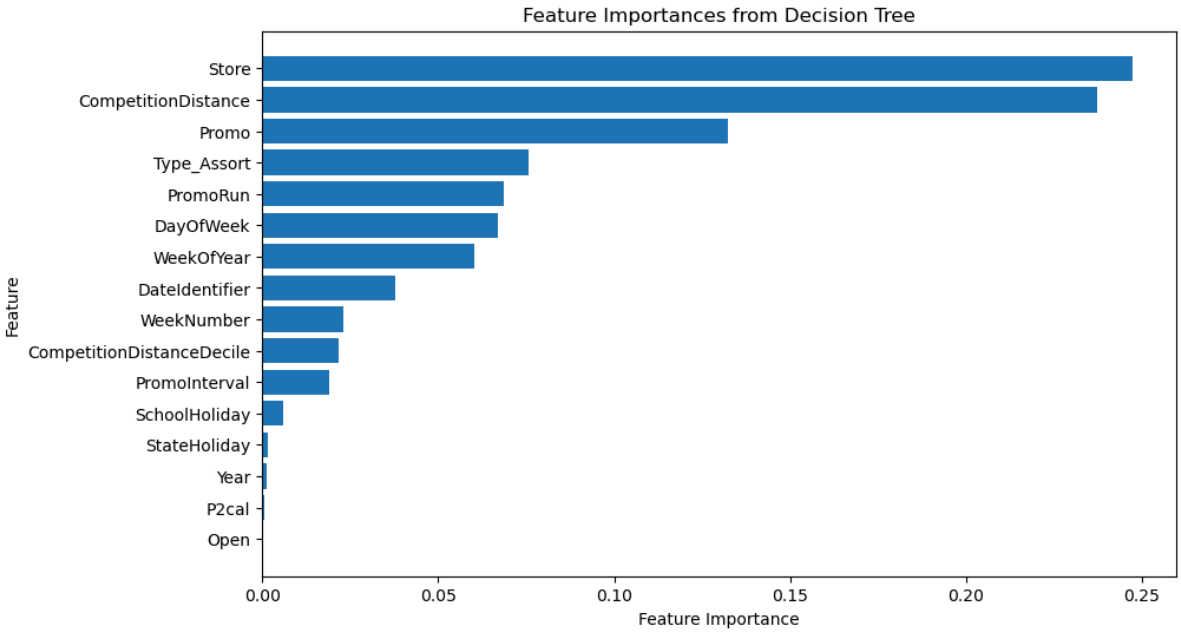
In [303…

```python
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Ensure that dfzc881 and dfzc61 are defined and loaded appropriately

# Separate features (X) and target variable (y) for the first DataFrame
X_train = dfzc881.drop(columns=['Sales'])
y_train = dfzc881['Sales']

# Separate features (X) and target variable (y) for the second DataFrame
X_test = dfzc61.drop(columns=['Sales'])
y_test = dfzc61['Sales']

# Initialize the Decision Tree regressor
dt_model = DecisionTreeRegressor(random_state=42)

# Train the model on the first DataFrame
dt_model.fit(X_train, y_train)

# Make predictions on the second DataFrame
y_pred = dt_model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Get feature importances
feature_importances = dt_model.feature_importances_

# Create a DataFrame for feature importances
features_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': feature_importances
})

# Sort by importance
```

```
features_df = features_df.sort_values(by='Importance', ascending=False)

# Calculate RMSPE
rmspe = np.sqrt(np.mean(((y_test - y_pred) / y_test) ** 2))

print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Root Mean Squared Percentage Error (RMSPE):", rmspe)
print("\nFeature Importances:\n", features_df)

# Create a DataFrame to compare actual and predicted sales
comparison_df = dfzc61[['DateIdentifier', 'Store', 'Sales']].copy()
comparison_df['Predicted_Sales'] = y_pred
comparison_df['Difference'] = comparison_df['Sales'] - comparison_df['Predic

# Show the first few rows of the comparison
print(comparison_df.head())

# Calculate total sales per day (projected vs actual)
daily_sales_comparison = comparison_df.groupby('DateIdentifier').agg({
    'Sales': 'sum',
    'Predicted_Sales': 'sum'
}).reset_index()
daily_sales_comparison['Difference'] = daily_sales_comparison['Sales'] - dai

print("\nDaily Sales Comparison (Actual vs Projected):")
print(daily_sales_comparison.head())

# Optional: Plotting daily sales comparison
plt.figure(figsize=(14, 7))
plt.plot(daily_sales_comparison['DateIdentifier'], daily_sales_comparison['S
plt.plot(daily_sales_comparison['DateIdentifier'], daily_sales_comparison['P
plt.fill_between(daily_sales_comparison['DateIdentifier'], daily_sales_compa
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Decision Tree -Total Daily Sales: Actual vs Predicted (2015:1 Jun
plt.legend()
plt.grid(True)
plt.show()

# Optional: Plotting feature importances
plt.figure(figsize=(10, 6))
plt.barh(features_df['Feature'], features_df['Importance'])
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Feature Importances from Decision Tree')
plt.gca().invert_yaxis()
plt.show()


# Calculate total actual and predicted sales
total_actual_sales = comparison_df['Sales'].sum()
total_predicted_sales = comparison_df['Predicted_Sales'].sum()
total_difference = total_actual_sales - total_predicted_sales

print("\nTotal Sales Comparison (Actual vs Projected):")
print("Total Actual Sales:", total_actual_sales)
print("Total Predicted Sales:", total_predicted_sales)
print("Total Difference:", total_difference)
```

```
Mean Squared Error: 2395733.914998891
R-squared: 0.7524871854990637
Root Mean Squared Percentage Error (RMSPE): 0.21758472290780764

Feature Importances:
                      Feature  Importance
0                       Store    0.247433
6         CompetitionDistance    0.237410
3                       Promo    0.132465
9                  Type_Assort    0.075752
10                    PromoRun    0.068542
1                    DayOfWeek    0.067064
15                  WeekOfYear    0.060197
11             DateIdentifier    0.037913
13                  WeekNumber    0.023094
8   CompetitionDistanceDecile    0.021558
7                PromoInterval    0.019101
5                SchoolHoliday    0.005972
4                 StateHoliday    0.001678
14                        Year    0.001226
12                       P2cal    0.000595
2                         Open    0.000000
    DateIdentifier  Store   Sales  Predicted_Sales  Difference
48             942      1  5263.0           4459.0       804.0
49             941      1  5020.0           4459.0       561.0
50             940      1  4782.0           4459.0       323.0
51             939      1  5011.0           4459.0       552.0
52             938      1  6102.0           6714.0      -612.0

Daily Sales Comparison (Actual vs Projected):
   DateIdentifier       Sales  Predicted_Sales  Difference
0             882  11743615.0       10995820.0    747795.0
1             883   9925193.0        8832805.0   1092388.0
2             884   9731791.0        8380923.0   1350868.0
3             885   3534231.0        3318476.0    215755.0
4             886   9724893.0        8468542.0   1256351.0
```





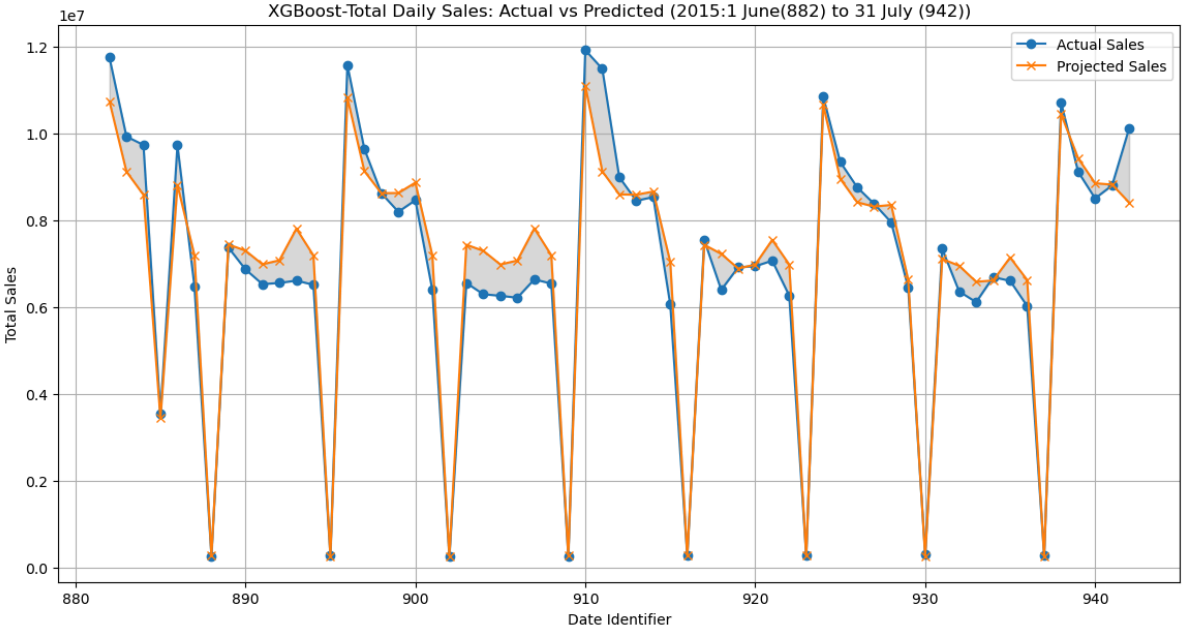

```
Total Sales Comparison (Actual vs Projected):
Total Actual Sales: 419685989.0
Total Predicted Sales: 429286517.0
Total Difference: -9600528.0
```

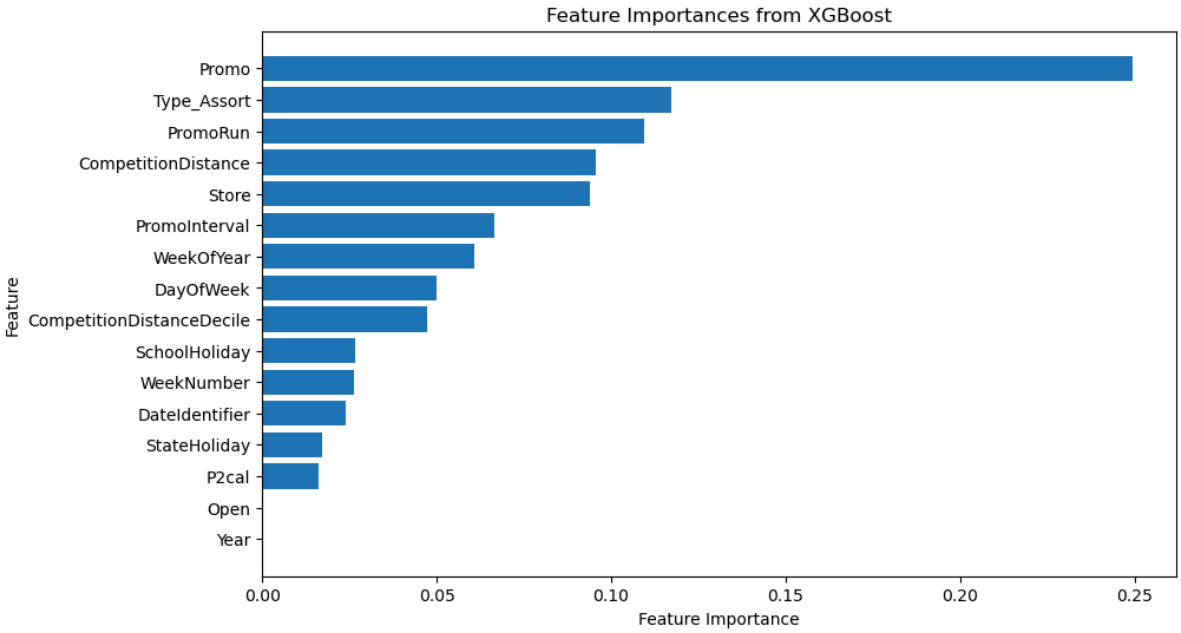
In [305…

```python
import xgboost as xgb
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Separate features (X) and target variable (y) for the first DataFrame
X_train = dfzc881.drop(columns=['Sales'])
y_train = dfzc881['Sales']

# Separate features (X) and target variable (y) for the second DataFrame
X_test = dfzc61.drop(columns=['Sales'])
y_test = dfzc61['Sales']

# Initialize the XGBoost regressor
xgb_model = xgb.XGBRegressor(n_estimators=100, random_state=42)

# Train the model on the first DataFrame
xgb_model.fit(X_train, y_train)

# Make predictions on the second DataFrame
y_pred = xgb_model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Calculate RMSPE
rmspe = np.sqrt(np.mean(((y_test - y_pred) / y_test) ** 2))

print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Root Mean Squared Percentage Error (RMSPE):", rmspe)

# Get feature importances
feature_importances = xgb_model.feature_importances_

# Create a DataFrame for feature importances
features_df = pd.DataFrame({
```

```
    'Feature': X_train.columns,
    'Importance': feature_importances
})

# Sort by importance
features_df = features_df.sort_values(by='Importance', ascending=False)

print("\nFeature Importances:\n", features_df)

# Create a DataFrame to compare actual and predicted sales
comparison_df = dfzc61[['DateIdentifier', 'Store', 'Sales']].copy()
comparison_df['Predicted_Sales'] = y_pred
comparison_df['Difference'] = comparison_df['Sales'] - comparison_df['Predic

# Show the first few rows of the comparison
print(comparison_df.head())

# Calculate total sales per day (projected vs actual)
daily_sales_comparison = comparison_df.groupby('DateIdentifier').agg({
    'Sales': 'sum',
    'Predicted_Sales': 'sum'
}).reset_index()
daily_sales_comparison['Difference'] = daily_sales_comparison['Sales'] - da

print("\nDaily Sales Comparison (Actual vs Projected):")
print(daily_sales_comparison.head())

# Optional: Plotting daily sales comparison
plt.figure(figsize=(14, 7))
plt.plot(daily_sales_comparison['DateIdentifier'], daily_sales_comparison['S
plt.plot(daily_sales_comparison['DateIdentifier'], daily_sales_comparison['P
plt.fill_between(daily_sales_comparison['DateIdentifier'], daily_sales_compa
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('XGBoost-Total Daily Sales: Actual vs Predicted (2015:1 June(882)
plt.legend()
plt.grid(True)
plt.show()

# Optional: Plotting feature importances
plt.figure(figsize=(10, 6))
plt.barh(features_df['Feature'], features_df['Importance'])
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Feature Importances from XGBoost')
plt.gca().invert_yaxis()
plt.show()

# Calculate total actual and predicted sales
total_actual_sales = comparison_df['Sales'].sum()
total_predicted_sales = comparison_df['Predicted_Sales'].sum()
total_difference = total_actual_sales - total_predicted_sales

print("\nTotal Sales Comparison (Actual vs Projected):")
print("Total Actual Sales:", total_actual_sales)
print("Total Predicted Sales:", total_predicted_sales)
print("Total Difference:", total_difference)
```

```
Mean Squared Error: 1867601.1575184863
R-squared: 0.8070506845653512
Root Mean Squared Percentage Error (RMSPE): 0.22730756268742128

Feature Importances:
                       Feature  Importance
3                        Promo    0.249544
9                   Type_Assort    0.117056
10                     PromoRun    0.109469
6          CompetitionDistance    0.095420
0                        Store    0.094050
7                PromoInterval    0.066532
15                  WeekOfYear    0.060700
1                    DayOfWeek    0.049848
8    CompetitionDistanceDecile    0.047268
5                SchoolHoliday    0.026639
13                  WeekNumber    0.026361
11              DateIdentifier    0.024025
4                 StateHoliday    0.017071
12                        P2cal    0.016017
2                          Open    0.000000
14                         Year    0.000000
    DateIdentifier  Store   Sales  Predicted_Sales  Difference
48             942      1  5263.0      5153.643555  109.356445
49             941      1  5020.0      5423.956543 -403.956543
50             940      1  4782.0      5423.956543 -641.956543
51             939      1  5011.0      5729.157227 -718.157227
52             938      1  6102.0      6683.180664 -581.180664

Daily Sales Comparison (Actual vs Projected):
   DateIdentifier       Sales  Predicted_Sales  Difference
0             882  11743615.0      10734241.00  1009374.00
1             883   9925193.0       9117243.00   807950.00
2             884   9731791.0       8582270.00  1149521.00
3             885   3534231.0       3438401.25    95829.75
4             886   9724893.0       8810866.00   914027.00
```






```
Total Sales Comparison (Actual vs Projected):
Total Actual Sales: 419685989.0
Total Predicted Sales: 425177540.0
Total Difference: -5491547.0
```

In [307…

```
# Filter for stores 1111-1115 and DateIdentifier 214
filtered_sales = dfzc942[(dfzc942['Store'].between(1,2)) & (dfzc942['DateIde

# Display the sales for the specified stores and date identifier
print("Sales for Store IDs 1111-1115 on DateIdentifier 214:")
print(filtered_sales[['DateIdentifier', 'Store', 'Sales']])
```

```
Sales for Store IDs 1111-1115 on DateIdentifier 214:
        DateIdentifier  Store   Sales
776                214      1  4494.0
777                213      1  4994.0
778                212      1  5572.0
779                211      1  5773.0
815047             214      2  4776.0
815048             213      2  5640.0
815049             212      2  8295.0
815050             211      2  6802.0
```

In [310…

```
# Filter for stores 1111-1115 and DateIdentifier 214
filtered_sales = dfzc942[(dfzc942['Store'].between(1, 5)) & (dfzc942['DateId

# Display the sales for the specified stores and date identifier
print("Sales for Store IDs 1-5 on DateIdentifier 154:")
print(filtered_sales[['DateIdentifier', 'Store', 'Sales']])
```

```
Sales for Store IDs 1-5 on DateIdentifier 154:
        DateIdentifier  Store    Sales
836                154      1   5422.0
1826               154      3  10025.0
815107             154      2   6959.0
816049             154      4  13444.0
```

In [312…

```
agg_df.tail()
```

Out[312]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	S
1058292	1111	2	01/01/2013	0.0	0.0	0.0	0	a	
1058293	1112	2	01/01/2013	0.0	0.0	0.0	0	a	
1058294	1113	2	01/01/2013	0.0	0.0	0.0	0	a	
1058295	1114	2	01/01/2013	0.0	0.0	0.0	0	a	
1058296	1115	2	01/01/2013	0.0	0.0	0.0	0	a	

In [314…
```
agg_df.head()
```

Out[314]:

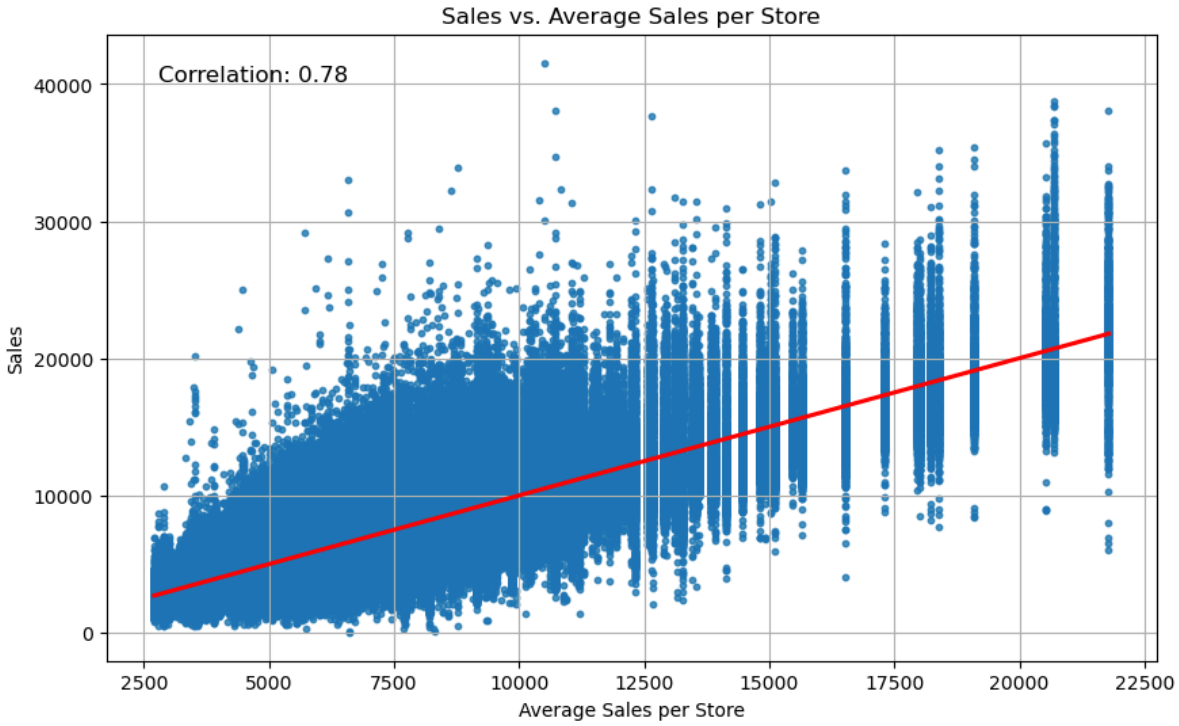
	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHo
0	1	4	17/09/2015	NaN	NaN	1.0	1	d	
1	3	4	17/09/2015	NaN	NaN	1.0	1	d	
2	7	4	17/09/2015	NaN	NaN	1.0	1	d	
3	8	4	17/09/2015	NaN	NaN	1.0	1	d	
4	9	4	17/09/2015	NaN	NaN	1.0	1	d	

In [316…
```
dfzc942.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 844338 entries, 48 to 1058295
Data columns (total 21 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   Sales                           844338 non-null  float64
 1   Store                           844338 non-null  int64  
 2   DayOfWeek                       844338 non-null  int64  
 3   Open                            844338 non-null  float64
 4   CompetitionDistanceDecile       844338 non-null  int64  
 5   Type_Assort                     844338 non-null  int64  
 6   DateIdentifier                  844338 non-null  int64  
 7   AvgSalesPerStore                844338 non-null  float64
 8   StoreRank                       844338 non-null  float64
 9   Index_DayOfWeek                 844338 non-null  float64
 10  Index_Promo                     844338 non-null  float64
 11  Index_StateHoliday              844338 non-null  float64
 12  Index_SchoolHoliday             844338 non-null  float64
 13  Index_CompetitionDistance       844338 non-null  float64
 14  Index_CompetitionDistanceDecile 844338 non-null  float64
 15  Index_PromoInterval             844338 non-null  float64
 16  Index_Type_Assort               844338 non-null  float64
 17  Index_PromoRun                  844338 non-null  float64
 18  Index_P2cal                     844338 non-null  float64
 19  Index_Year                      844338 non-null  float64
 20  Index_WeekOfYear                844338 non-null  float64
dtypes: float64(16), int64(5)
memory usage: 141.7 MB
```

In [318…
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import linregress

# Assuming dfzc942 is your DataFrame
df = dfzc942
```

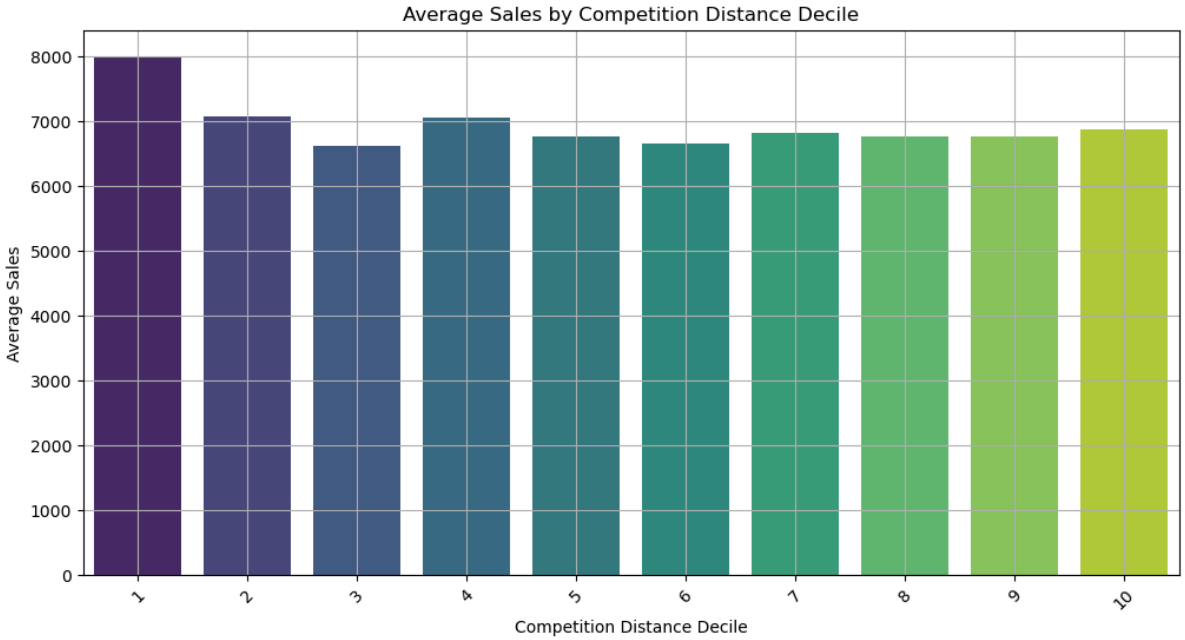
```
# Scatter plot with regression line
plt.figure(figsize=(10, 6))

# Plot using seaborn
sns.regplot(x='AvgSalesPerStore', y='Sales', data=df, scatter_kws={'s':10},

# Calculate the regression line and correlation score
slope, intercept, r_value, p_value, std_err = linregress(df['AvgSalesPerSto

# Add correlation score to the plot
plt.text(0.05, 0.95, f'Correlation: {r_value:.2f}', transform=plt.gca().tra

# Set titles and labels
plt.title('Sales vs. Average Sales per Store')
plt.xlabel('Average Sales per Store')
plt.ylabel('Sales')

# Show plot
plt.grid(True)
plt.show()
```



In [319…

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming dfzc942 is your DataFrame
df = dfzc942

# Calculate the average sales by CompetitionDistanceDecile
avg_sales_by_decile = df.groupby('CompetitionDistanceDecile')['Sales'].mean(

# Plotting the bar plot
plt.figure(figsize=(12, 6))
sns.barplot(x='CompetitionDistanceDecile', y='Sales', data=avg_sales_by_deci

# Set titles and labels
plt.title('Average Sales by Competition Distance Decile')
plt.xlabel('Competition Distance Decile')
plt.ylabel('Average Sales')
```

```
# Show plot
plt.xticks(rotation=45)  # Rotate x labels for better readability
plt.grid(True)
plt.show()
```



In [320…

```
import pandas as pd
import numpy as np

# Assuming dfzc942 contains the data with 'DateIdentifier', 'Store', and 'Sa

# Create the new column '728prior' and initialize with 0
dfzc942['728prior'] = 0

# Create a DataFrame that maps each (Store, DateIdentifier) to the correspor
sales_lookup = dfzc942.set_index(['Store', 'DateIdentifier'])['Sales'].to_di

# Define a function to get the sales from 728 days prior
def get_728_prior(row):
    if row['DateIdentifier'] < 729:
        return 0
    else:
        prior_identifier = row['DateIdentifier'] - 728
        return sales_lookup.get((row['Store'], prior_identifier), 0)

# Apply the function to each row to populate the '728prior' column
dfzc942['728prior'] = dfzc942.apply(get_728_prior, axis=1)

# Display the first few rows to verify the result
dfzc942.head(20).T
```

Out[320]:

	48	49	50	51	
Sales	5263.000000	5020.000000	4782.000000	5011.000000	6
Store	1.000000	1.000000	1.000000	1.000000	
DayOfWeek	5.000000	4.000000	3.000000	2.000000	
Open	1.000000	1.000000	1.000000	1.000000	
CompetitionDistanceDecile	4.000000	4.000000	4.000000	4.000000	
Type_Assort	5.000000	5.000000	5.000000	5.000000	
DateIdentifier	942.000000	941.000000	940.000000	939.000000	
AvgSalesPerStore	4781.312757	4781.312757	4781.312757	4781.312757	
StoreRank	164.000000	164.000000	164.000000	164.000000	
Index_DayOfWeek	1.200379	1.148938	1.139140	1.198516	
Index_Promo	1.383223	1.383223	1.383223	1.383223	
Index_StateHoliday	1.000000	1.000000	1.000000	1.000000	
Index_SchoolHoliday	1.038444	1.038444	1.038444	1.038444	
Index_CompetitionDistance	3.078043	3.078043	3.078043	3.078043	
Index_CompetitionDistanceDecile	1.067662	1.067662	1.067662	1.067662	
Index_PromoInterval	1.182367	1.182367	1.182367	1.182367	
Index_Type_Assort	1.064799	1.064799	1.064799	1.064799	
Index_PromoRun	1.672982	1.672982	1.672982	1.672982	
Index_P2cal	1.110357	1.110357	1.110357	1.110357	
Index_Year	1.032501	1.032501	1.032501	1.032501	
Index_WeekOfYear	1.302267	1.302267	1.302267	1.302267	
728prior	4494.000000	4994.000000	5572.000000	5773.000000	6

In [321…

```
dfzc942.tail(20).T
```

Out[321]:

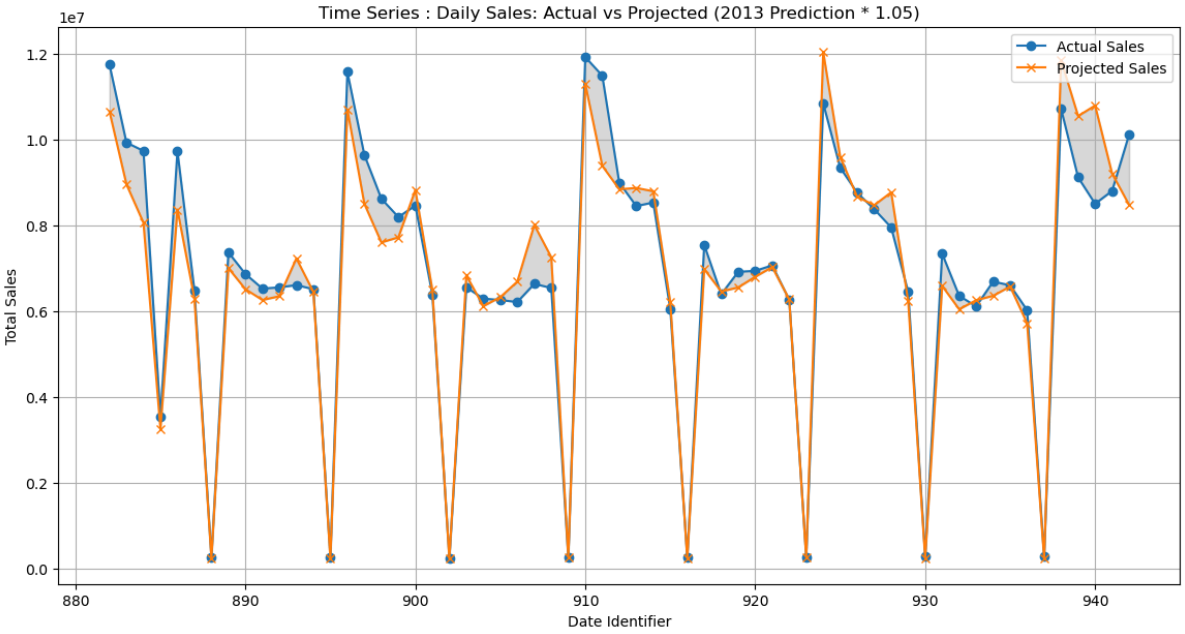
	1058273	1058274	1058275	1058276	
Sales	4689.000000	4270.000000	5079.000000	6520.000000	3
Store	1110.000000	1110.000000	1110.000000	1110.000000	1
DayOfWeek	4.000000	3.000000	2.000000	1.000000	
Open	1.000000	1.000000	1.000000	1.000000	
CompetitionDistanceDecile	3.000000	3.000000	3.000000	3.000000	
Type_Assort	6.000000	6.000000	6.000000	6.000000	
DateIdentifier	24.000000	23.000000	22.000000	21.000000	
AvgSalesPerStore	4520.365753	4520.365753	4520.365753	4520.365753	4
StoreRank	124.000000	124.000000	124.000000	124.000000	
Index_DayOfWeek	1.148938	1.139140	1.198516	1.391329	
Index_Promo	1.383223	1.383223	1.383223	1.383223	
Index_StateHoliday	1.000000	1.000000	1.000000	1.000000	
Index_SchoolHoliday	1.000000	1.000000	1.000000	1.000000	
Index_CompetitionDistance	1.883922	1.883922	1.883922	1.883922	
Index_CompetitionDistanceDecile	1.000000	1.000000	1.000000	1.000000	
Index_PromoInterval	1.182367	1.182367	1.182367	1.182367	
Index_Type_Assort	1.094733	1.094733	1.094733	1.094733	
Index_PromoRun	1.672982	1.672982	1.672982	1.672982	
Index_P2cal	1.110357	1.110357	1.110357	1.110357	
Index_Year	1.000000	1.000000	1.000000	1.000000	
Index_WeekOfYear	1.050478	1.050478	1.050478	1.029845	
728prior	0.000000	0.000000	0.000000	0.000000	

In [322…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score

# Assuming dfcz942 contains the data with 'DateIdentifier', 'Store', 'Sales

# Filter the data for DateIdentifier from 882 to 942
filtered_df = dfzc942[(dfzc942['DateIdentifier'] >= 882) & (dfzc942['DateIde

# Rename columns for clarity
filtered_df = filtered_df.rename(columns={'Sales': 'Sales_2015', 'DateIdenti

# Use 728prior as the predicted sales
filtered_df['Predicted_Sales'] = filtered_df['728prior'] * 1.05

# Create comparison DataFrame
comparison_df = filtered_df[['DateIdentifier_2015', 'Store', 'Sales_2015',
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['P

# Calculate evaluation metrics
actual_sales = comparison_df['Sales_2015']
predicted_sales = comparison_df['Predicted_Sales']
mse = mean_squared_error(actual_sales, predicted_sales)
```

```
r2 = r2_score(actual_sales, predicted_sales)
rmspe = np.sqrt(np.mean(((actual_sales - predicted_sales) / actual_sales) *

print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Root Mean Squared Percentage Error (RMSPE):", rmspe)

# Show the first few rows of the comparison
print("Comparison of Actual and Predicted Sales:")
print(comparison_df.head())

# Calculate total sales per day (projected vs actual)
daily_sales_comparison = comparison_df.groupby('DateIdentifier_2015').agg({
    'Sales_2015': 'sum',
    'Predicted_Sales': 'sum'
}).reset_index()
daily_sales_comparison['Difference'] = daily_sales_comparison['Sales_2015']

print("\nDaily Sales Comparison (Actual vs Projected):")
print(daily_sales_comparison.head())

# Plotting daily sales comparison
plt.figure(figsize=(14, 7))
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparis
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparis
plt.fill_between(daily_sales_comparison['DateIdentifier_2015'], daily_sales_
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Time Series : Daily Sales: Actual vs Projected (2013 Prediction >
plt.legend()
plt.grid(True)
plt.show()

# Calculate total actual and predicted sales
total_actual_sales = comparison_df['Sales_2015'].sum()
total_predicted_sales = comparison_df['Predicted_Sales'].sum()
total_difference = total_actual_sales - total_predicted_sales

print("\nTotal Sales Comparison (Actual vs Projected):")
print("Total Actual Sales:", total_actual_sales)
print("Total Predicted Sales:", total_predicted_sales)
print("Total Difference:", total_difference)
```

```
Mean Squared Error: 2468040.307249578
R-squared: 0.7450169240729801
Root Mean Squared Percentage Error (RMSPE): 0.21017469363375993
Comparison of Actual and Predicted Sales:
    DateIdentifier_2015  Store  Sales_2015  Predicted_Sales  Difference
48                  942      1      5263.0          4718.70      544.30
49                  941      1      5020.0          5243.70     -223.70
50                  940      1      4782.0          5850.60    -1068.60
51                  939      1      5011.0          6061.65    -1050.65
52                  938      1      6102.0          6604.50     -502.50

Daily Sales Comparison (Actual vs Projected):
   DateIdentifier_2015  Sales_2015  Predicted_Sales  Difference
0                  882  11743615.0      10652238.45  1091376.55
1                  883   9925193.0       8953175.70   972017.30
2                  884   9731791.0       8060047.80  1671743.20
3                  885   3534231.0       3261149.85   273081.15
4                  886   9724893.0       8366038.80  1358854.20
```



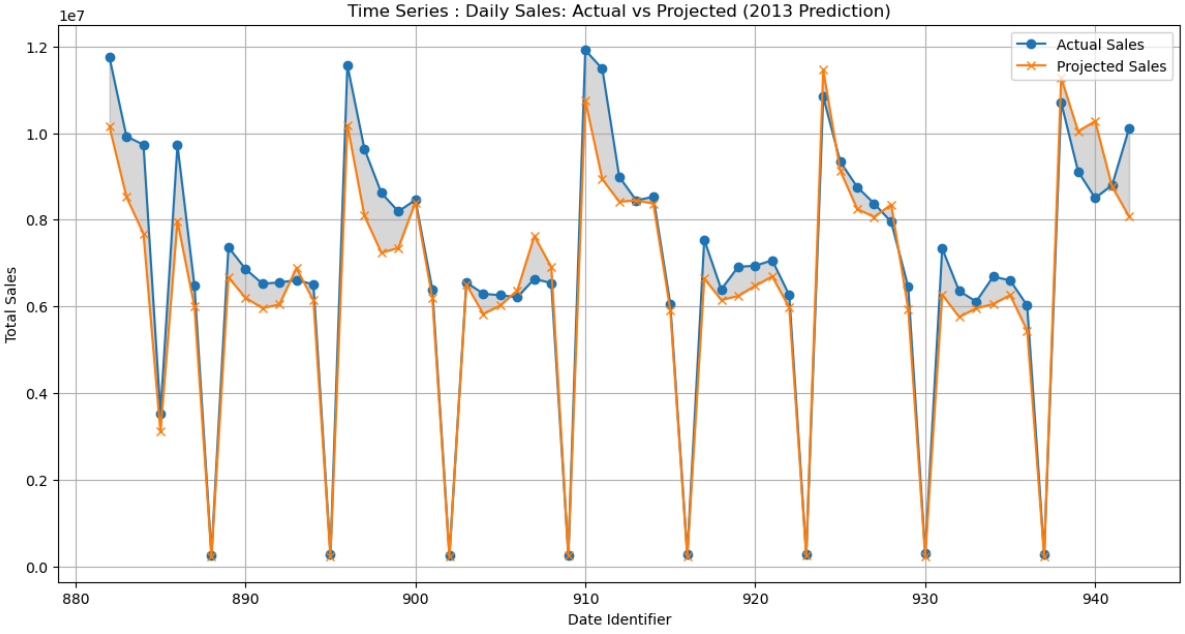

```
Total Sales Comparison (Actual vs Projected):
Total Actual Sales: 419685989.0
Total Predicted Sales: 414023015.7
Total Difference: 5662973.300000012
```

In [323…

```
# Filter the data for DateIdentifier from 882 to 942
filtered_df = dfzc942[(dfzc942['DateIdentifier'] >= 882) & (dfzc942['DateIde

# Rename columns for clarity
filtered_df = filtered_df.rename(columns={'Sales': 'Sales_2015', 'DateIdenti

# Use 728prior as the predicted sales
filtered_df['Predicted_Sales'] = filtered_df['728prior']

# Create comparison DataFrame
comparison_df = filtered_df[['DateIdentifier_2015', 'Store', 'Sales_2015',
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['P

# Calculate evaluation metrics
actual_sales = comparison_df['Sales_2015']
predicted_sales = comparison_df['Predicted_Sales']
mse = mean_squared_error(actual_sales, predicted_sales)
r2 = r2_score(actual_sales, predicted_sales)
rmspe = np.sqrt(np.mean(((actual_sales - predicted_sales) / actual_sales) **

print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Root Mean Squared Percentage Error (RMSPE):", rmspe)

# Show the first few rows of the comparison
print("Comparison of Actual and Predicted Sales:")
print(comparison_df.head())

# Calculate total sales per day (projected vs actual)
daily_sales_comparison = comparison_df.groupby('DateIdentifier_2015').agg({
    'Sales_2015': 'sum',
    'Predicted_Sales': 'sum'
}).reset_index()
daily_sales_comparison['Difference'] = daily_sales_comparison['Sales_2015']

print("\nDaily Sales Comparison (Actual vs Projected):")
print(daily_sales_comparison.head())

# Plotting daily sales comparison
```

```
plt.figure(figsize=(14, 7))
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparis
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparis
plt.fill_between(daily_sales_comparison['DateIdentifier_2015'], daily_sales_
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Time Series : Daily Sales: Actual vs Projected (2013 Prediction)')
plt.legend()
plt.grid(True)
plt.show()

# Calculate total actual and predicted sales
total_actual_sales = comparison_df['Sales_2015'].sum()
total_predicted_sales = comparison_df['Predicted_Sales'].sum()
total_difference = total_actual_sales - total_predicted_sales

print("\nTotal Sales Comparison (Actual vs Projected):")
print("Total Actual Sales:", total_actual_sales)
print("Total Predicted Sales:", total_predicted_sales)
print("Total Difference:", total_difference)
```

```
Mean Squared Error: 2541271.801112419
R-squared: 0.7374510867140729
Root Mean Squared Percentage Error (RMSPE): 0.20475527931238302
Comparison of Actual and Predicted Sales:
    DateIdentifier_2015  Store  Sales_2015  Predicted_Sales  Difference
48                  942      1      5263.0           4494.0       769.0
49                  941      1      5020.0           4994.0        26.0
50                  940      1      4782.0           5572.0      -790.0
51                  939      1      5011.0           5773.0      -762.0
52                  938      1      6102.0           6290.0      -188.0

Daily Sales Comparison (Actual vs Projected):
   DateIdentifier_2015  Sales_2015  Predicted_Sales  Difference
0                  882  11743615.0       10144989.0   1598626.0
1                  883   9925193.0        8526834.0   1398359.0
2                  884   9731791.0        7676236.0   2055555.0
3                  885   3534231.0        3105857.0    428374.0
4                  886   9724893.0        7967656.0   1757237.0
```

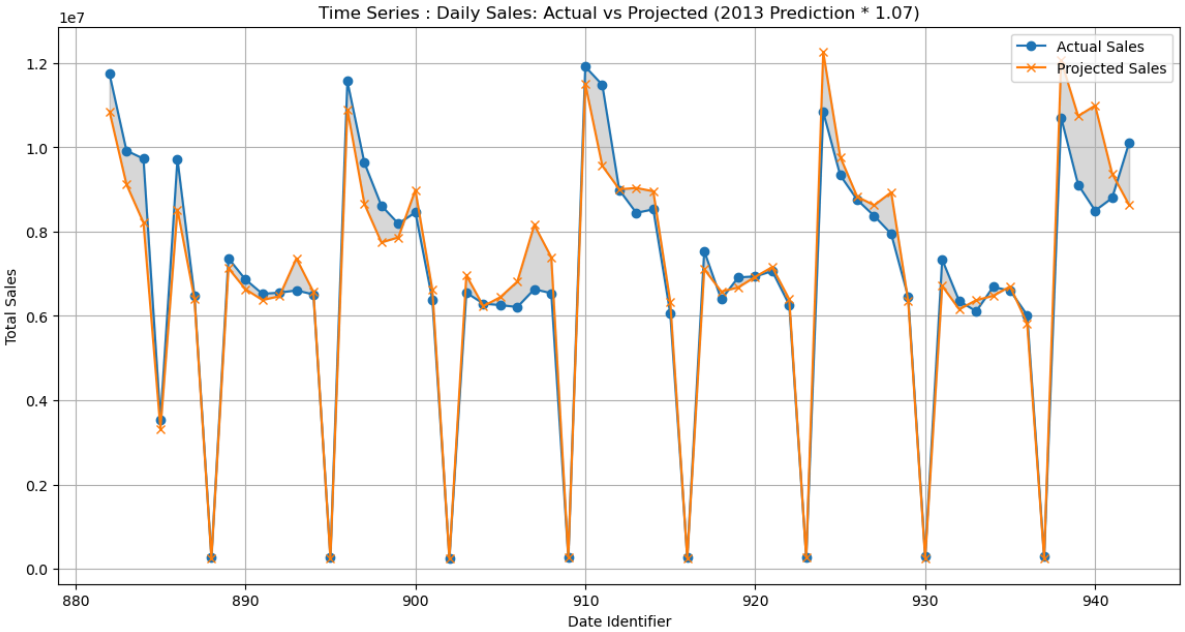



```
Total Sales Comparison (Actual vs Projected):
Total Actual Sales: 419685989.0
Total Predicted Sales: 394307634.0
Total Difference: 25378355.0
```

In [324…

```
# Filter the data for DateIdentifier from 882 to 942
filtered_df = dfzc942[(dfzc942['DateIdentifier'] >= 882) & (dfzc942['DateIde

# Rename columns for clarity
filtered_df = filtered_df.rename(columns={'Sales': 'Sales_2015', 'DateIdent

# Use 728prior as the predicted sales
filtered_df['Predicted_Sales'] = filtered_df['728prior'] * 1.07

# Create comparison DataFrame
comparison_df = filtered_df[['DateIdentifier_2015', 'Store', 'Sales_2015',
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['

# Calculate evaluation metrics
actual_sales = comparison_df['Sales_2015']
predicted_sales = comparison_df['Predicted_Sales']
mse = mean_squared_error(actual_sales, predicted_sales)
r2 = r2_score(actual_sales, predicted_sales)
rmspe = np.sqrt(np.mean(((actual_sales - predicted_sales) / actual_sales) **

print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Root Mean Squared Percentage Error (RMSPE):", rmspe)

# Show the first few rows of the comparison
print("Comparison of Actual and Predicted Sales:")
print(comparison_df.head())

# Calculate total sales per day (projected vs actual)
daily_sales_comparison = comparison_df.groupby('DateIdentifier_2015').agg({
    'Sales_2015': 'sum',
    'Predicted_Sales': 'sum'
}).reset_index()
daily_sales_comparison['Difference'] = daily_sales_comparison['Sales_2015']

print("\nDaily Sales Comparison (Actual vs Projected):")
print(daily_sales_comparison.head())

# Plotting daily sales comparison
plt.figure(figsize=(14, 7))
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparis
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparis
plt.fill_between(daily_sales_comparison['DateIdentifier_2015'], daily_sales_
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Time Series : Daily Sales: Actual vs Projected (2013 Prediction >
plt.legend()
plt.grid(True)
plt.show()

# Calculate total actual and predicted sales
total_actual_sales = comparison_df['Sales_2015'].sum()
total_predicted_sales = comparison_df['Predicted_Sales'].sum()
total_difference = total_actual_sales - total_predicted_sales

print("\nTotal Sales Comparison (Actual vs Projected):")
print("Total Actual Sales:", total_actual_sales)
print("Total Predicted Sales:", total_predicted_sales)
print("Total Difference:", total_difference)
```

```
Mean Squared Error: 2514676.4569052537
R-squared: 0.7401987576704027
Root Mean Squared Percentage Error (RMSPE): 0.21543043251038377
Comparison of Actual and Predicted Sales:
    DateIdentifier_2015  Store  Sales_2015  Predicted_Sales  Difference
48                  942      1      5263.0          4808.58      454.42
49                  941      1      5020.0          5343.58     -323.58
50                  940      1      4782.0          5962.04    -1180.04
51                  939      1      5011.0          6177.11    -1166.11
52                  938      1      6102.0          6730.30     -628.30

Daily Sales Comparison (Actual vs Projected):
   DateIdentifier_2015  Sales_2015  Predicted_Sales  Difference
0                  882  11743615.0      10855138.23   888476.77
1                  883   9925193.0       9123712.38   801480.62
2                  884   9731791.0       8213572.52  1518218.48
3                  885   3534231.0       3323266.99   210964.01
4                  886   9724893.0       8525391.92  1199501.08
```



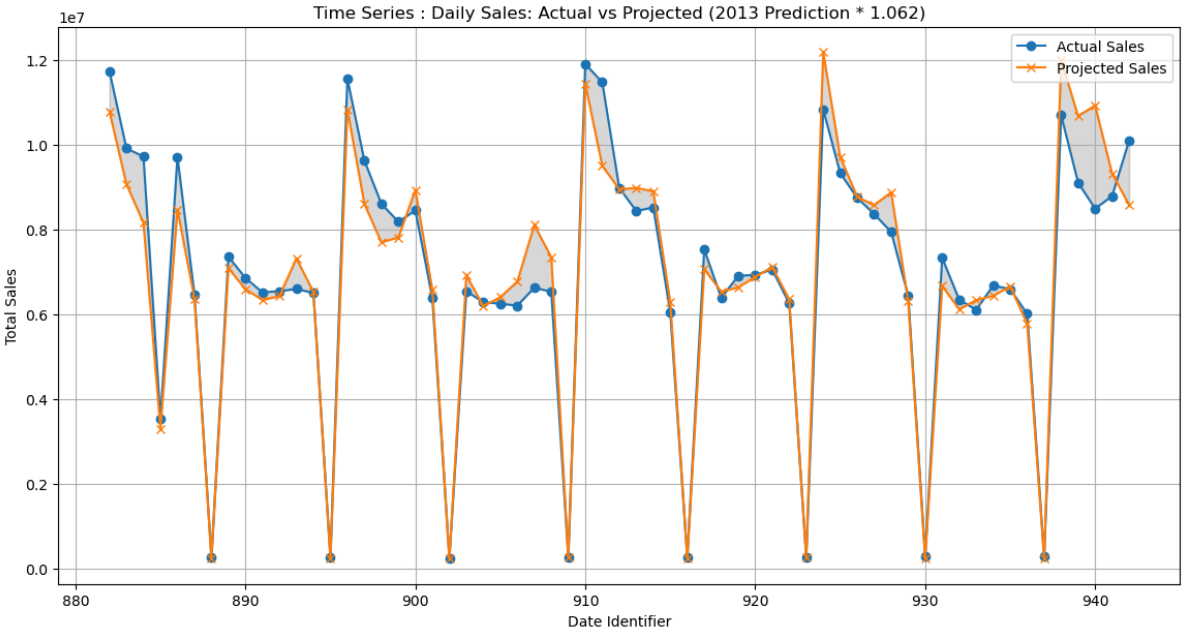

```
Total Sales Comparison (Actual vs Projected):
Total Actual Sales: 419685989.0
Total Predicted Sales: 421909168.38
Total Difference: -2223179.379999995
```

In [325…

```
# Filter the data for DateIdentifier from 882 to 942
filtered_df = dfzc942[(dfzc942['DateIdentifier'] >= 882) & (dfzc942['DateIde

# Rename columns for clarity
filtered_df = filtered_df.rename(columns={'Sales': 'Sales_2015', 'DateIdenti

# Use 728prior as the predicted sales
filtered_df['Predicted_Sales'] = filtered_df['728prior'] * 1.064362

# Create comparison DataFrame
comparison_df = filtered_df[['DateIdentifier_2015', 'Store', 'Sales_2015',
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['P

# Calculate evaluation metrics
actual_sales = comparison_df['Sales_2015']
predicted_sales = comparison_df['Predicted_Sales']
mse = mean_squared_error(actual_sales, predicted_sales)
r2 = r2_score(actual_sales, predicted_sales)
rmspe = np.sqrt(np.mean(((actual_sales - predicted_sales) / actual_sales) **

print("Mean Squared Error:", mse)
```

```
print("R-squared:", r2)
print("Root Mean Squared Percentage Error (RMSPE):", rmspe)

# Show the first few rows of the comparison
print("Comparison of Actual and Predicted Sales:")
print(comparison_df.head())

# Calculate total sales per day (projected vs actual)
daily_sales_comparison = comparison_df.groupby('DateIdentifier_2015').agg({
    'Sales_2015': 'sum',
    'Predicted_Sales': 'sum'
}).reset_index()
daily_sales_comparison['Difference'] = daily_sales_comparison['Sales_2015']

print("\nDaily Sales Comparison (Actual vs Projected):")
print(daily_sales_comparison.head())

# Plotting daily sales comparison
plt.figure(figsize=(14, 7))
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparis
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparis
plt.fill_between(daily_sales_comparison['DateIdentifier_2015'], daily_sales_
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Time Series : Daily Sales: Actual vs Projected (2013 Prediction >
plt.legend()
plt.grid(True)
plt.show()

# Calculate total actual and predicted sales
total_actual_sales = comparison_df['Sales_2015'].sum()
total_predicted_sales = comparison_df['Predicted_Sales'].sum()
total_difference = total_actual_sales - total_predicted_sales

print("\nTotal Sales Comparison (Actual vs Projected):")
print("Total Actual Sales:", total_actual_sales)
print("Total Predicted Sales:", total_predicted_sales)
print("Total Difference:", total_difference)
```

```
Mean Squared Error: 2497138.1726700137
R-squared: 0.7420107076809628
Root Mean Squared Percentage Error (RMSPE): 0.21378106919970866
Comparison of Actual and Predicted Sales:
    DateIdentifier_2015  Store  Sales_2015  Predicted_Sales   Difference
48                  942      1      5263.0      4783.242828   479.757172
49                  941      1      5020.0      5315.423828  -295.423828
50                  940      1      4782.0      5930.625064 -1148.625064
51                  939      1      5011.0      6144.561826 -1133.561826
52                  938      1      6102.0      6694.836980  -592.836980

Daily Sales Comparison (Actual vs Projected):
   DateIdentifier_2015  Sales_2015  Predicted_Sales    Difference
0                  882  11743615.0     1.079794e+07  9.456742e+05
1                  883   9925193.0     9.075638e+06  8.495549e+05
2                  884   9731791.0     8.170294e+06  1.561497e+06
3                  885   3534231.0     3.305756e+06  2.284748e+05
4                  886   9724893.0     8.480470e+06  1.244423e+06
```



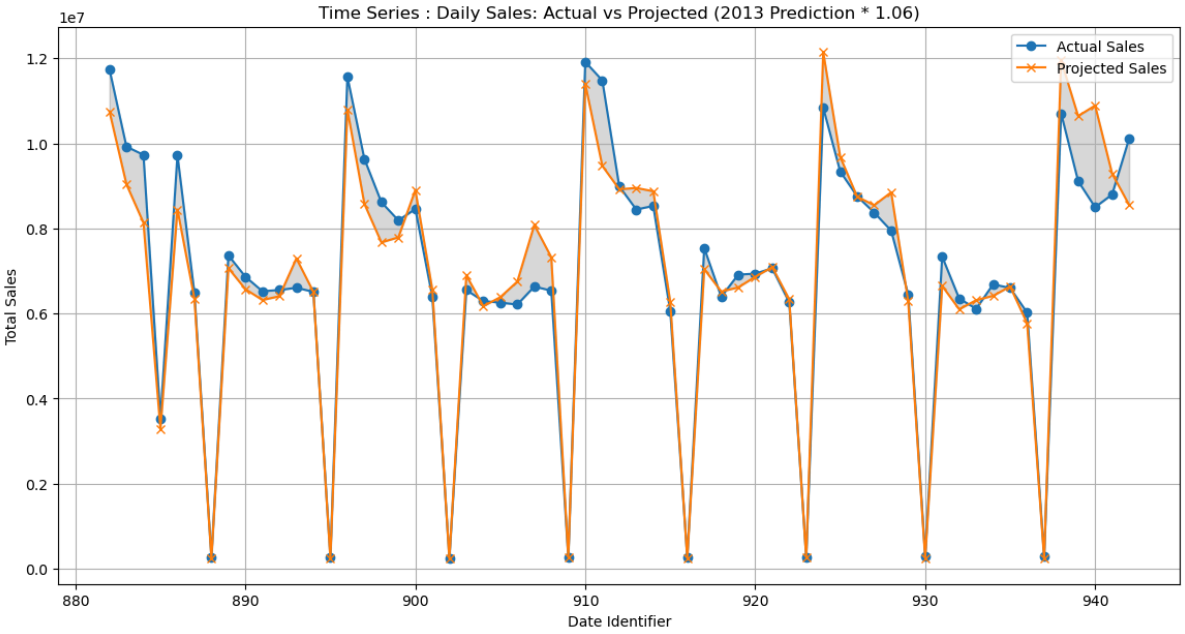

```
Total Sales Comparison (Actual vs Projected):
Total Actual Sales: 419685989.0
Total Predicted Sales: 419686061.93950796
Total Difference: -72.9395079612732
```

In [334…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score

# Assuming dfcz942 contains the data with 'DateIdentifier', 'Store', 'Sales

# Filter the data for DateIdentifier from 882 to 942
filtered_df = dfzc942[(dfzc942['DateIdentifier'] >= 882) & (dfzc942['DateIde

# Rename columns for clarity
filtered_df = filtered_df.rename(columns={'Sales': 'Sales_2015', 'DateIdenti

# Use 728prior as the predicted sales
filtered_df['Predicted_Sales'] = filtered_df['728prior'] * 1.06

# Create comparison DataFrame
comparison_df = filtered_df[['DateIdentifier_2015', 'Store', 'Sales_2015',
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['P

# Calculate evaluation metrics
actual_sales = comparison_df['Sales_2015']
predicted_sales = comparison_df['Predicted_Sales']
mse = mean_squared_error(actual_sales, predicted_sales)
r2 = r2_score(actual_sales, predicted_sales)
rmspe = np.sqrt(np.mean(((actual_sales - predicted_sales) / actual_sales) **

print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Root Mean Squared Percentage Error (RMSPE):", rmspe)

# Show the first few rows of the comparison
print("Comparison of Actual and Predicted Sales:")
print(comparison_df.head())

# Calculate total sales per day (projected vs actual)
daily_sales_comparison = comparison_df.groupby('DateIdentifier_2015').agg({
    'Sales_2015': 'sum',
    'Predicted_Sales': 'sum'
```

```
}).reset_index()
daily_sales_comparison['Difference'] = daily_sales_comparison['Sales_2015']

print("\nDaily Sales Comparison (Actual vs Projected):")
print(daily_sales_comparison.head())

# Plotting daily sales comparison
plt.figure(figsize=(14, 7))
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparis
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparis
plt.fill_between(daily_sales_comparison['DateIdentifier_2015'], daily_sales_
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Time Series : Daily Sales: Actual vs Projected (2013 Prediction >
plt.legend()
plt.grid(True)
plt.show()

# Calculate total actual and predicted sales
total_actual_sales = comparison_df['Sales_2015'].sum()
total_predicted_sales = comparison_df['Predicted_Sales'].sum()
total_difference = total_actual_sales - total_predicted_sales

print("\nTotal Sales Comparison (Actual vs Projected):")
print("Total Actual Sales:", total_actual_sales)
print("Total Predicted Sales:", total_predicted_sales)
print("Total Difference:", total_difference)
```

```
Mean Squared Error: 2485934.9001345006
R-squared: 0.7431681623964157
Root Mean Squared Percentage Error (RMSPE): 0.21259423343159178
Comparison of Actual and Predicted Sales:
    DateIdentifier_2015  Store  Sales_2015  Predicted_Sales  Difference
48                  942      1      5263.0          4763.64      499.36
49                  941      1      5020.0          5293.64     -273.64
50                  940      1      4782.0          5906.32    -1124.32
51                  939      1      5011.0          6119.38    -1108.38
52                  938      1      6102.0          6667.40     -565.40

Daily Sales Comparison (Actual vs Projected):
   DateIdentifier_2015  Sales_2015  Predicted_Sales  Difference
0                  882  11743615.0      10753688.34   989926.66
1                  883   9925193.0       9038444.04   886748.96
2                  884   9731791.0       8136810.16  1594980.84
3                  885   3534231.0       3292208.42   242022.58
4                  886   9724893.0       8445715.36  1279177.64
```



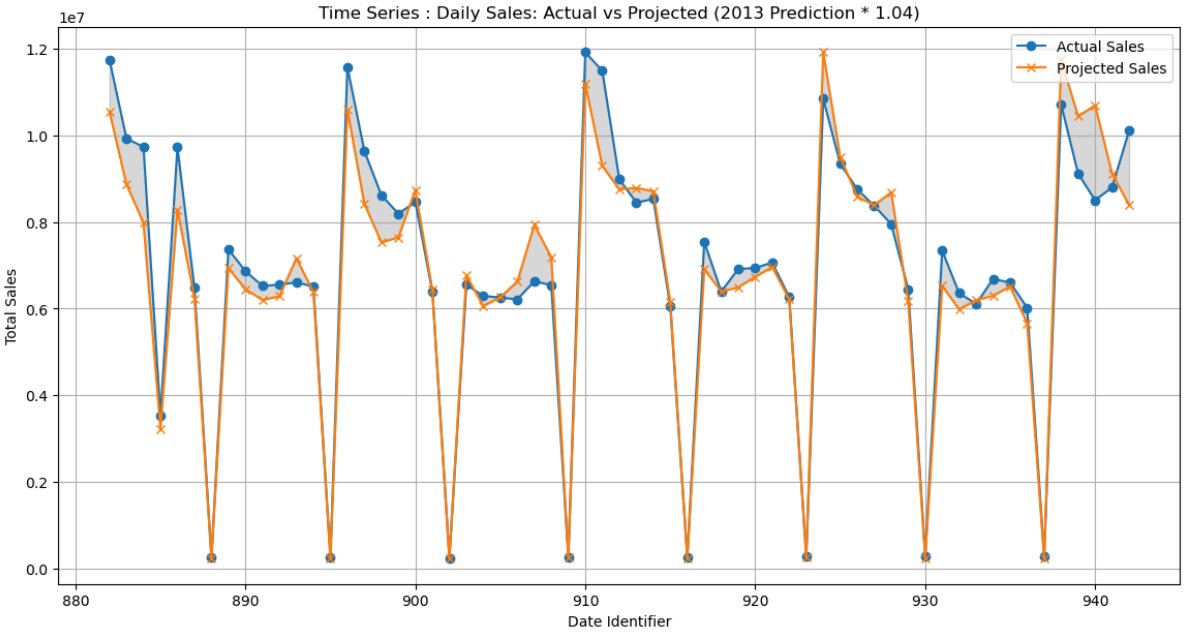

```
Total Sales Comparison (Actual vs Projected):
Total Actual Sales: 419685989.0
Total Predicted Sales: 417966092.0400001
Total Difference: 1719896.959999919
```

In [336…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score

# Assuming dfcz942 contains the data with 'DateIdentifier', 'Store', 'Sales

# Filter the data for DateIdentifier from 882 to 942
filtered_df = dfzc942[(dfzc942['DateIdentifier'] >= 882) & (dfzc942['DateIde

# Rename columns for clarity
filtered_df = filtered_df.rename(columns={'Sales': 'Sales_2015', 'DateIdenti

# Use 728prior as the predicted sales
filtered_df['Predicted_Sales'] = filtered_df['728prior'] * 1.04

# Create comparison DataFrame
comparison_df = filtered_df[['DateIdentifier_2015', 'Store', 'Sales_2015',
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['P

# Calculate evaluation metrics
actual_sales = comparison_df['Sales_2015']
predicted_sales = comparison_df['Predicted_Sales']
mse = mean_squared_error(actual_sales, predicted_sales)
r2 = r2_score(actual_sales, predicted_sales)
rmspe = np.sqrt(np.mean(((actual_sales - predicted_sales) / actual_sales) **

print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("Root Mean Squared Percentage Error (RMSPE):", rmspe)

# Show the first few rows of the comparison
print("Comparison of Actual and Predicted Sales:")
print(comparison_df.head())

# Calculate total sales per day (projected vs actual)
daily_sales_comparison = comparison_df.groupby('DateIdentifier_2015').agg({
    'Sales_2015': 'sum',
    'Predicted_Sales': 'sum'
```

```
}).reset_index()
daily_sales_comparison['Difference'] = daily_sales_comparison['Sales_2015']

print("\nDaily Sales Comparison (Actual vs Projected):")
print(daily_sales_comparison.head())

# Plotting daily sales comparison
plt.figure(figsize=(14, 7))
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparis
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparis
plt.fill_between(daily_sales_comparison['DateIdentifier_2015'], daily_sales_
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Time Series : Daily Sales: Actual vs Projected (2013 Prediction >
plt.legend()
plt.grid(True)
plt.show()

# Calculate total actual and predicted sales
total_actual_sales = comparison_df['Sales_2015'].sum()
total_predicted_sales = comparison_df['Predicted_Sales'].sum()
total_difference = total_actual_sales - total_predicted_sales

print("\nTotal Sales Comparison (Actual vs Projected):")
print("Total Actual Sales:", total_actual_sales)
print("Total Predicted Sales:", total_predicted_sales)
print("Total Difference:", total_difference)
```

```
Mean Squared Error: 2460992.6782504856
R-squared: 0.745745042700096
Root Mean Squared Percentage Error (RMSPE): 0.208186340855127
Comparison of Actual and Predicted Sales:
    DateIdentifier_2015  Store  Sales_2015  Predicted_Sales  Difference
48                  942      1      5263.0          4673.76      589.24
49                  941      1      5020.0          5193.76     -173.76
50                  940      1      4782.0          5794.88    -1012.88
51                  939      1      5011.0          6003.92     -992.92
52                  938      1      6102.0          6541.60     -439.60

Daily Sales Comparison (Actual vs Projected):
   DateIdentifier_2015  Sales_2015  Predicted_Sales  Difference
0                  882  11743615.0      10550788.56  1192826.44
1                  883   9925193.0       8867907.36  1057285.64
2                  884   9731791.0       7983285.44  1748505.56
3                  885   3534231.0       3230091.28   304139.72
4                  886   9724893.0       8286362.24  1438530.76
```




```
Total Sales Comparison (Actual vs Projected):
Total Actual Sales: 419685989.0
Total Predicted Sales: 410079939.36
Total Difference: 9606049.639999986
```

In [338…
```
dfzc942.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 844338 entries, 48 to 1058295
Data columns (total 22 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   Sales                           844338 non-null  float64
 1   Store                           844338 non-null  int64  
 2   DayOfWeek                       844338 non-null  int64  
 3   Open                            844338 non-null  float64
 4   CompetitionDistanceDecile       844338 non-null  int64  
 5   Type_Assort                     844338 non-null  int64  
 6   DateIdentifier                  844338 non-null  int64  
 7   AvgSalesPerStore                844338 non-null  float64
 8   StoreRank                       844338 non-null  float64
 9   Index_DayOfWeek                 844338 non-null  float64
 10  Index_Promo                     844338 non-null  float64
 11  Index_StateHoliday              844338 non-null  float64
 12  Index_SchoolHoliday             844338 non-null  float64
 13  Index_CompetitionDistance       844338 non-null  float64
 14  Index_CompetitionDistanceDecile 844338 non-null  float64
 15  Index_PromoInterval             844338 non-null  float64
 16  Index_Type_Assort               844338 non-null  float64
 17  Index_PromoRun                  844338 non-null  float64
 18  Index_P2cal                     844338 non-null  float64
 19  Index_Year                      844338 non-null  float64
 20  Index_WeekOfYear                844338 non-null  float64
 21  728prior                        844338 non-null  float64
dtypes: float64(17), int64(5)
memory usage: 148.2 MB
```

In [340…
```
dfzc.info()
```

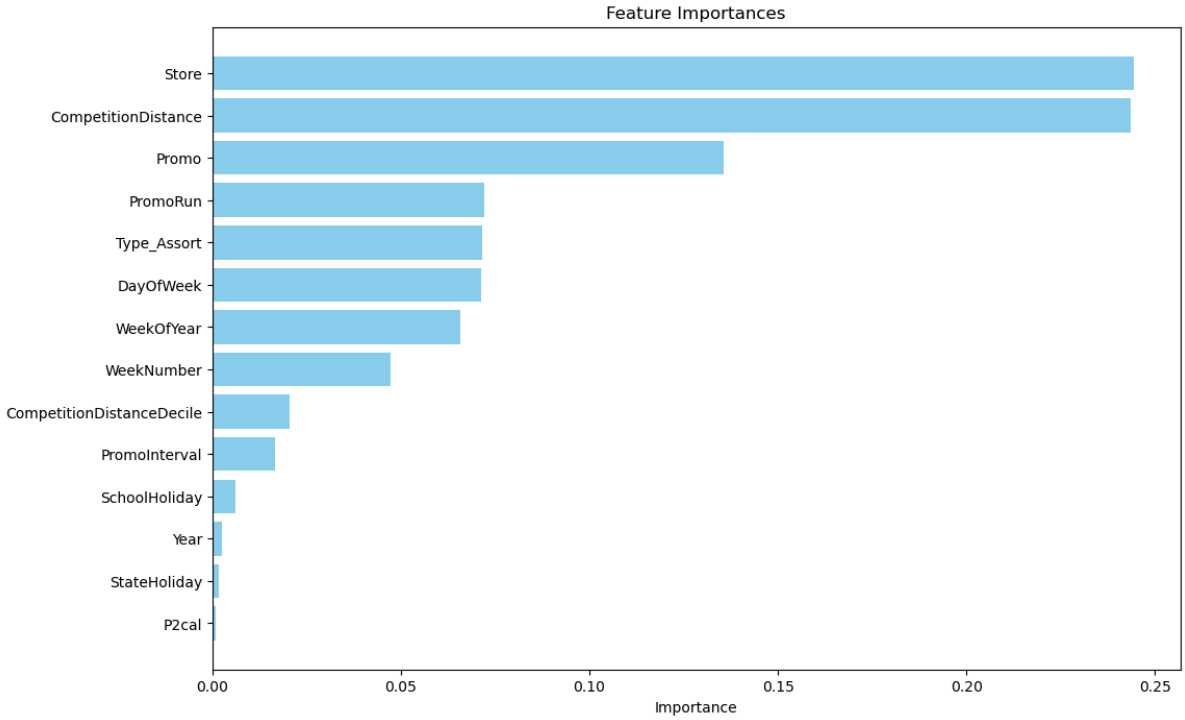
```
<class 'pandas.core.frame.DataFrame'>
Index: 879442 entries, 0 to 1058295
Data columns (total 17 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   Sales                      844338 non-null  float64
 1   Store                      879442 non-null  int64
 2   DayOfWeek                  879442 non-null  int64
 3   Open                       879442 non-null  float64
 4   Promo                      879442 non-null  int64
 5   StateHoliday               879442 non-null  int64
 6   SchoolHoliday              879442 non-null  int64
 7   CompetitionDistance        879442 non-null  float64
 8   PromoInterval              879442 non-null  float64
 9   CompetitionDistanceDecile  879442 non-null  int64
 10  Type_Assort                879442 non-null  int64
 11  PromoRun                   879442 non-null  float64
 12  DateIdentifier             879442 non-null  int64
 13  P2cal                      879442 non-null  int64
 14  WeekNumber                 879442 non-null  int64
 15  Year                       879442 non-null  int64
 16  WeekOfYear                 879442 non-null  int64
dtypes: float64(5), int64(12)
memory usage: 120.8 MB
```

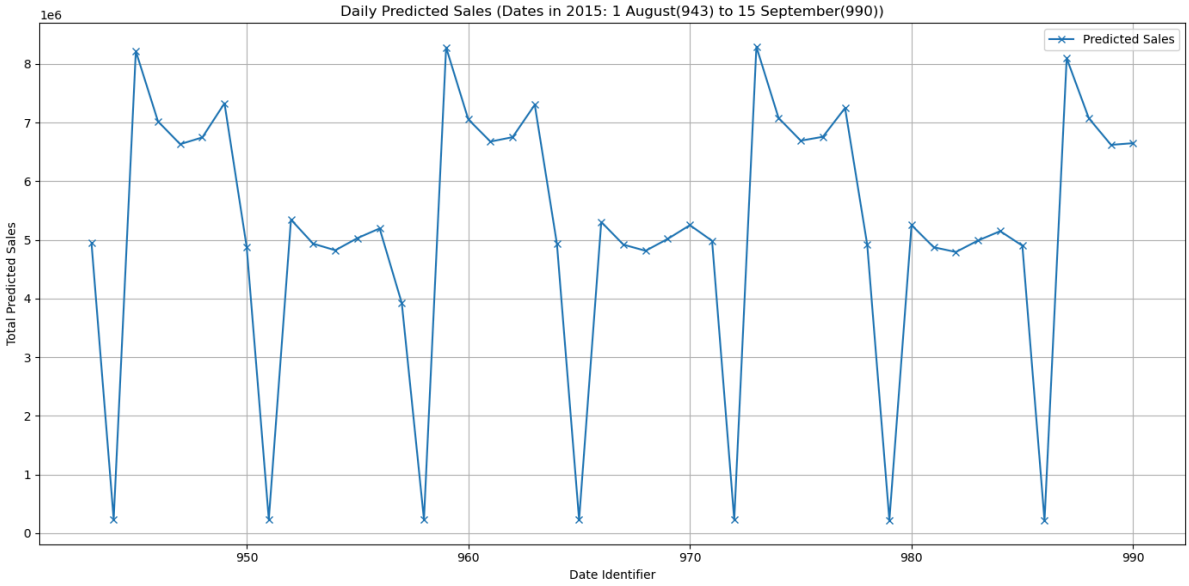
In [342…

```
# Remove rows where Open = 0 and assign to a new DataFrame dfzc0
dfzc0 = dfzc[dfzc['Open'] != 0]

# Filter the DataFrame to include only DateIdentifiers 1-942 for training
train_df = dfzc0[dfzc0['DateIdentifier'] <= 942].dropna(subset=['Sales'])

# Filter the DataFrame to include only DateIdentifiers 943-990 for testing
test_df = dfzc0[(dfzc0['DateIdentifier'] >= 943) & (dfzc0['DateIdentifier']

# Features to use for prediction
features = ['Store', 'DayOfWeek', 'Promo', 'StateHoliday', 'SchoolHoliday',
            'CompetitionDistance', 'PromoInterval', 'CompetitionDistanceDec
            'Type_Assort', 'PromoRun', 'P2cal', 'WeekNumber', 'Year', 'Week(

# Separate features and target variable for training
X_train = train_df[features]
y_train = train_df['Sales']

# Fit the Random Forest model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predict sales for the test set
X_test = test_df[features]
test_df['Predicted_Sales'] = rf_model.predict(X_test)

# Show feature importances in a table
feature_importances = pd.DataFrame({
    'Feature': features,
    'Importance': rf_model.feature_importances_
}).sort_values(by='Importance', ascending=False)

# Print feature importances
print("Feature Importances:")
print(feature_importances)

# Plot feature importances as a horizontal bar chart
plt.figure(figsize=(12, 8))
plt.barh(feature_importances['Feature'], feature_importances['Importance'],
```

```
plt.xlabel('Importance')
plt.title('Feature Importances')
plt.gca().invert_yaxis()
plt.show()

# Calculate total predicted sales for each day
daily_sales_prediction = test_df.groupby('DateIdentifier').agg({
    'Predicted_Sales': 'sum'
}).reset_index()

# Calculate total predicted sales for the entire prediction period
total_predicted_sales = daily_sales_prediction['Predicted_Sales'].sum()

# Plotting the predicted sales for each day
plt.figure(figsize=(14, 7))
plt.plot(daily_sales_prediction['DateIdentifier'], daily_sales_prediction['P
plt.xlabel('Date Identifier')
plt.ylabel('Total Predicted Sales')
plt.title('Daily Predicted Sales (Dates in 2015: 1 August(943) to 15 Septemb
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Print the total predicted sales for the entire prediction period
print("\nTotal Predicted Sales for the entire prediction period (DateIdentif
print(total_predicted_sales)

# Print the daily sales prediction table
print("\nDaily Predicted Sales for each DateIdentifier (943-990):")
print(daily_sales_prediction)
```

```
Feature Importances:
                      Feature  Importance
0                       Store    0.244465
5         CompetitionDistance    0.243527
2                       Promo    0.135518
9                     PromoRun    0.071999
8                  Type_Assort    0.071614
1                    DayOfWeek    0.071325
13                  WeekOfYear    0.065632
11                  WeekNumber    0.047363
7   CompetitionDistanceDecile    0.020601
6                PromoInterval    0.016632
4                SchoolHoliday    0.006086
12                        Year    0.002640
3                StateHoliday    0.001649
10                       P2cal    0.000949
```


Feature Importances
Store
CompetitionDistance
Promo
PromoRun
Type_Assort
DayOfWeek
WeekOfYear
WeekNumber
CompetitionDistanceDecile
PromoInterval
SchoolHoliday
Year
StateHoliday
P2cal
0.00
0.05
0.10
0.15
0.20
0.25
Importance


Daily Predicted Sales (Dates in 2015: 1 August(943) to 15 September(990))
1e6
Predicted Sales
Total Predicted Sales
0
1
2
3
4
5
6
7
8
950
960
970
980
990
Date Identifier

```
Total Predicted Sales for the entire prediction period (DateIdentifier 943-
990):
247306853.77

Daily Predicted Sales for each DateIdentifier (943-990):
    DateIdentifier  Predicted_Sales
0              943       4943651.01
1              944        237473.12
2              945       8216755.75
3              946       7020315.48
4              947       6631753.66
5              948       6747087.14
6              949       7329149.82
7              950       4882197.43
8              951        235757.68
9              952       5345355.40
10             953       4935322.07
11             954       4823067.33
12             955       5030460.83
13             956       5195136.97
14             957       3921599.06
15             958        234367.71
16             959       8281347.69
17             960       7055294.81
18             961       6675810.42
19             962       6751562.00
20             963       7303388.65
21             964       4930941.38
22             965        230743.05
23             966       5306343.67
24             967       4918921.10
25             968       4814472.44
26             969       5017131.33
27             970       5251235.52
28             971       4984132.69
29             972        229314.48
30             973       8292467.82
31             974       7078944.47
32             975       6691521.11
33             976       6757660.81
34             977       7248738.55
35             978       4918770.23
36             979        222552.25
37             980       5252496.54
38             981       4875075.38
39             982       4793543.58
40             983       4989567.90
41             984       5149026.18
42             985       4901239.01
43             986        215951.28
44             987       8097176.32
45             988       7073041.54
46             989       6620173.15
47             990       6648817.96
```

Note German Holidays Date Day Holiday States 1 Jan Mon New Year's Day National 6 Jan Sat Epiphany BW, BY & ST 8 Mar Fri International Women's Day BE & MV 29 Mar Fri Good Friday National 31 Mar Sun Easter Sunday BB 1 Apr Mon Easter Monday National 1 May Wed Labour Day National 9 May Thu Ascension Day National 19 May Sun Whit Sunday BB 20 May Mon Whit Monday National 30 May Thu Corpus Christi BW, BY, HE, NW, RP, SL, SN & TH 15 Aug Thu Assumption Day BY & SL 20 Sep Fri Children's Day TH 3 Oct Thu Day of German Unity National 31 Oct Thu Reformation Day BB, HH, MV, NI, SH, SN, ST & TH 1 Nov Fri All Saints' Day BW, BY, NW, RP & SL 20 Nov Wed Repentance Day SN 25 Dec Wed Christmas Day National 26 Dec Thu 2nd Day of Christmas National