DRAFT 1

DATA70002 Assignment Student ID number:11489360

In [1]:
```
# For traditions sake start with the phrase 'Hello World'
print ('Hello World')
```

```
Hello World
```

In [2]:
```
#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 [3]:
```
# Remove warning messages
import warnings
warnings.filterwarnings('ignore')
```

1. Review Data Provided

In [4]:
```
# Read in the CSV file and name df
store= pd.read_csv("store.csv")
```

In [5]:
```
# Begin review of df
store.shape
```

Out[5]:
```
(1115, 10)
```

In [6]:
```
# Df has 1115 rows and 10 columns(cols)
# Continue review of df
# Display df's first 5 rows
store.head().T
```

Out[6]:

	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 [7]:
```
# Display df's last 10 rows
store.tail(10).T
```

Out[7]:

	1105	1106	1107	1108	1109	1110	1111	1112	1113	11
Store	1106	1107	1108	1109	1110	1111	1112	1113	1114	11
StoreType	a	a	a	c	c	a	c	a	a	
Assortment	c	a	a	a	c	a	c	c	c	
CompetitionDistance	5330.0	1400.0	540.0	3490.0	900.0	1900.0	1880.0	9260.0	870.0	535(
CompetitionOpenSinceMonth	9.0	6.0	4.0	4.0	9.0	6.0	4.0	NaN	NaN	N
CompetitionOpenSinceYear	2011.0	2012.0	2004.0	2011.0	2010.0	2014.0	2006.0	NaN	NaN	N
Promo2	1	1	0	1	0	1	0	0	0	
Promo2SinceWeek	31.0	13.0	NaN	22.0	NaN	31.0	NaN	NaN	NaN	22
Promo2SinceYear	2013.0	2010.0	NaN	2012.0	NaN	2013.0	NaN	NaN	NaN	201
PromoInterval	Jan,Apr,Jul,Oct	Jan,Apr,Jul,Oct	NaN	Jan,Apr,Jul,Oct	NaN	Jan,Apr,Jul,Oct	NaN	NaN	NaN	Mar,Jun,Sept,D

In [8]:
```
# There are numerous columns that are not visible in the df review so change viewing angle
store_transposed = store.T
store.T.head(10)
```

Out[8]:

	0	1	2	3	4	5	6	7	8	9	...	1105
Store	1	2	3	4	5	6	7	8	9	10	...	1106
StoreType	c	a	a	c	a	a	a	a	a	a	...	a
Assortment	a	a	a	c	a	a	c	a	c	a	...	c
CompetitionDistance	1270.0	570.0	14130.0	620.0	29910.0	310.0	24000.0	7520.0	2030.0	3160.0	...	5330.0
CompetitionOpenSinceMonth	9.0	11.0	12.0	9.0	4.0	12.0	4.0	10.0	8.0	9.0	...	9.0
CompetitionOpenSinceYear	2008.0	2007.0	2006.0	2009.0	2015.0	2013.0	2013.0	2014.0	2000.0	2009.0	...	2011.0
Promo2	0	1	1	0	0	0	0	0	0	0	...	1
Promo2SinceWeek	NaN	13.0	14.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	31.0
Promo2SinceYear	NaN	2010.0	2011.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	2013.0
PromoInterval	NaN	Jan,Apr,Jul,Oct	Jan,Apr,Jul,Oct	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	Jan,Apr,Jul,Oc

10 rows × 1115 columns

In [9]:
```
# 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 [10]:
```
# 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 [11]:
```
# Sum up the distances for non-null values
store['CompetitionDistance'].sum()
```

Out[11]:
```
6010250.0
```

In [12]:
```
# Calculate mean long hand
+6010250.0/1112
```

Out[12]:
```
5404.901079136691
```

In [13]:
```
# 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 [14]:
```
# Calculate the mean of CompetitionDistance
competition_distance_mean = round(store['CompetitionDistance'].mean(), 0)

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

In [15]:
```
# Continue review of df
# Review details of one row for reasonableness
store.loc[290]
```

Out[15]:
```
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 [16]:
```
# Continue review of df
# Review details of one row for reasonableness
store.loc[621]
```

Out[16]:
```
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 [17]:
```
store.nunique()
```

Out[17]:
```
Store                      1115
StoreType                     4
Assortment                    3
CompetitionDistance         655
CompetitionOpenSinceMonth    12
CompetitionOpenSinceYear     23
Promo2                        2
Promo2SinceWeek              24
Promo2SinceYear               7
PromoInterval                 3
dtype: int64
```

In [18]:
```
# 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 [19]:
```
# Filter rows for years 2013, 2014, and 2015
competition_years = store[store['CompetitionOpenSinceYear'].isin([2013, 2014, 2015])]

# 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 [20]:
```
# Count Promo2 values
store["Promo2"].value_counts()
```

Out[20]:
```
Promo2
1    571
0    544
Name: count, dtype: int64
```

In [21]:
```
# 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 [22]:
```
# 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 [23]:
```
# 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 [24]:
```
# 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 [25]:
```
# Storetype value count
store["StoreType"].value_counts()
```

Out[25]:
```
StoreType
a    602
d    348
c    148
b     17
Name: count, dtype: int64
```

In [26]:
```
# Assortment value count
store["Assortment"].value_counts()
```

Out[26]:
```
Assortment
a    593
c    513
b      9
Name: count, dtype: int64
```

In [27]:
```
# 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 [28]:
```
# Check no zero count
zero_count = store.loc[store['CompetitionDistance'] == 0, 'CompetitionDistance'].count()

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

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

In [29]:
```
# 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, CompetitionOpenSinceMonth, CompetitionOpenSinceYear, Pro
mo2, Promo2SinceWeek, Promo2SinceYear, PromoInterval]
Index: []
```

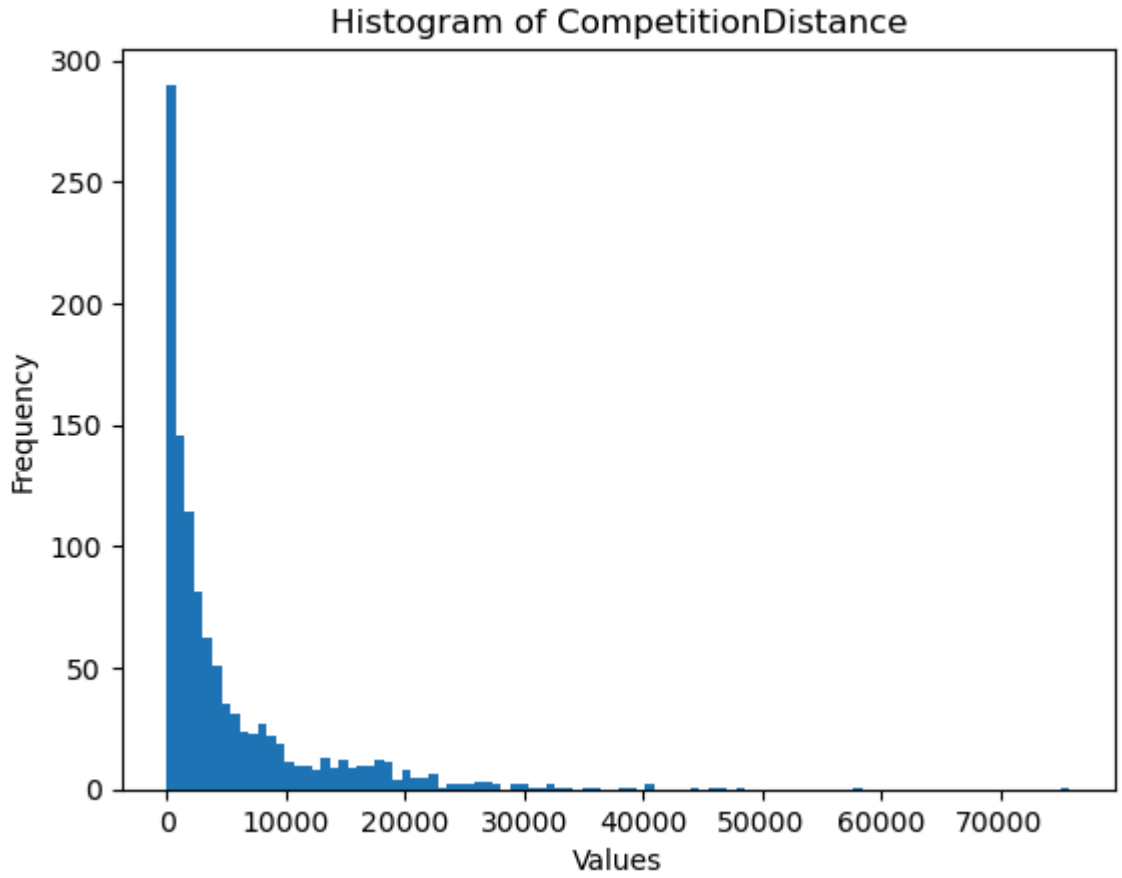
In [30]:
```
# Get statistics
column_stats = store['CompetitionDistance'].describe()

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
```

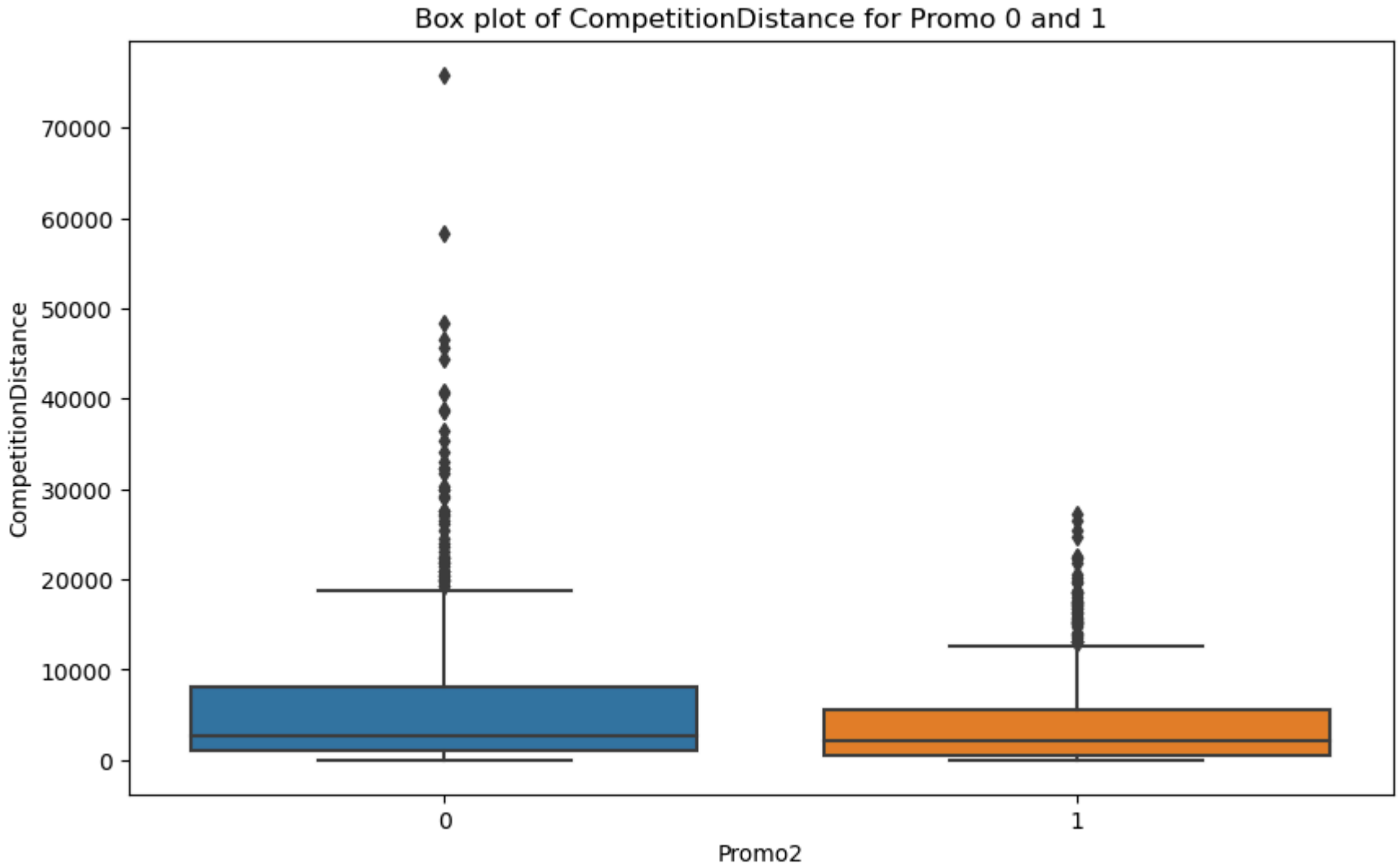
In [31]:
```python
# Calculate the mode for the column
mode_result = store['CompetitionDistance'].mode()

# 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 [32]:
```python
plt.hist(store['CompetitionDistance'], bins=100)  # Adjust the number of bins as needed
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.title('Histogram of ' + 'CompetitionDistance')  # Adjust the title as needed
plt.show()
```



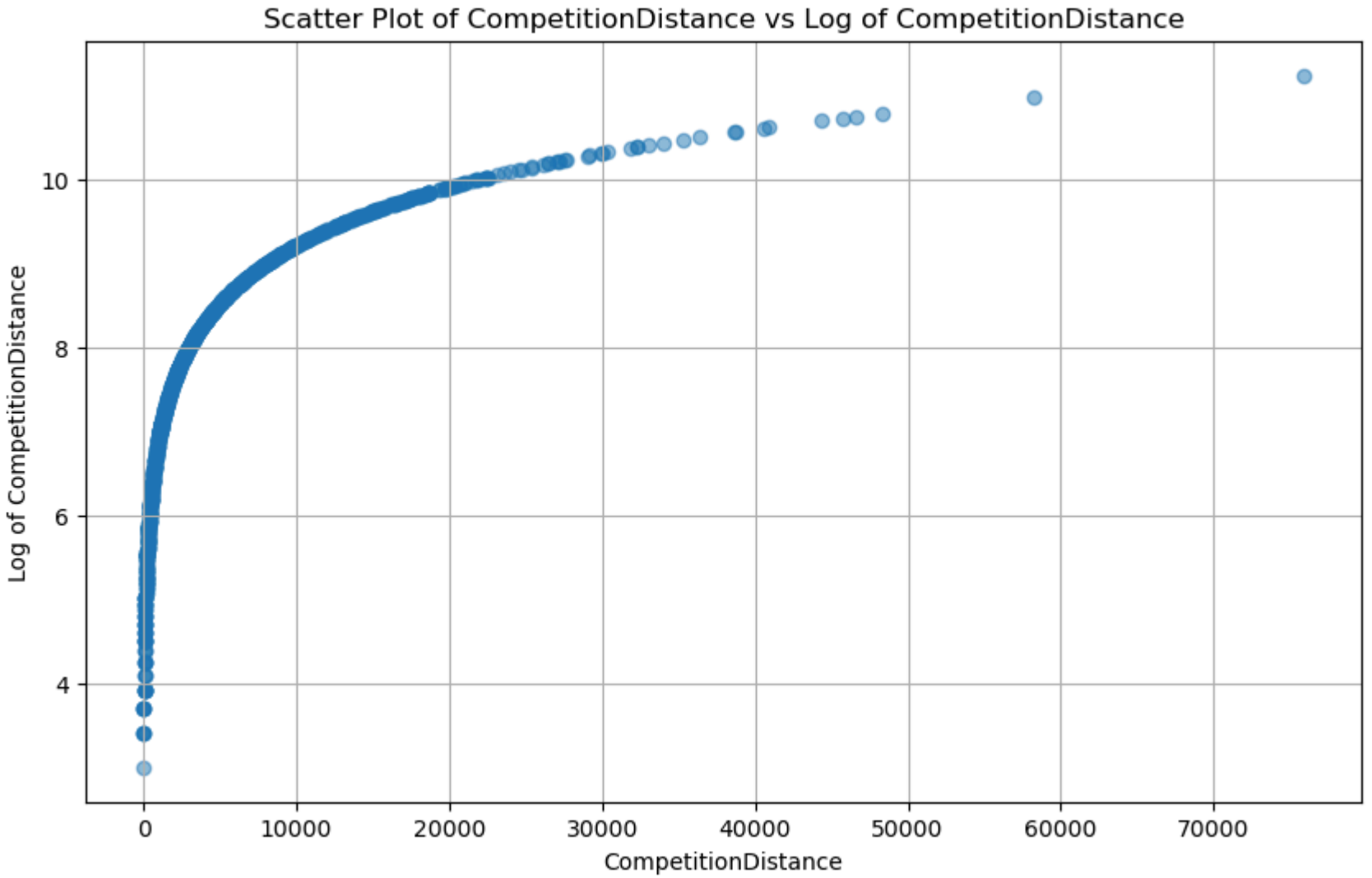
In [33]:
```
# Filter the DataFrame based on 'Promo' values
promo_0 = store[store['Promo2'] == 0]
promo_1 = store[store['Promo2'] == 1]

# 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 [34]:
```
# 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 [35]:
```
import numpy as np
import matplotlib.pyplot as plt

# Scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(store['CompetitionDistance'], np.log(store['CompetitionDistance']), alpha=0.5)
plt.xlabel('CompetitionDistance')
plt.ylabel('Log of CompetitionDistance')
plt.title('Scatter Plot of CompetitionDistance vs Log of CompetitionDistance')
plt.grid(True)
plt.show()
```



In [36]:
```
# 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 [37]:
```
# 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 [38]:
```
# Calculate deciles
deciles = store['CompetitionDistance'].quantile([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])

# 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 value

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

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

# 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 [39]:
```
# 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 [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['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 [41]:
```
store["Promo2"].value_counts()
```

Out[41]:
```
Promo2
1    571
0    544
Name: count, dtype: int64
```

In [42]:
```
# 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 [43]:
```
# 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 [44]:
```
# Create a new column 'NewCategory' by combining 'StoreType' and 'Assortment' categories
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 [45]:
```
store2['Type_Assort'].value_counts()
```

Out[45]:
```
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 [46]:
```
# 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[46]:

	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 [47]:
```
# 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 [48]:
```
# Check the categories to numbers
store2.tail().T
```

Out[48]:

	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 [49]:
```
store2['PromoInterval'].value_counts()
```

Out[49]:
```
PromoInterval
0.0    544
1.0    335
2.0    130
3.0    106
Name: count, dtype: int64
```

In [50]:
```
# Define a mapping for year to number
year_mapping = {0: -31, 2015: 0, 2014: 52, 2013: 104, 2012: 156, 2011: 208, 2010: 260, 2009: 313, 2008: 365, 2007: 417


# 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 [51]:
```
# Calculate the PromoRun
store2['PromoRun'] = store2['Promo2SinceYear'] - store2['Promo2SinceWeek'] + 31

# 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 [52]:
```
store2['PromoRun'].value_counts()
```

Out[52]:

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 [53]:
```
store2['PromoRun'].value_counts().sort_index(ascending=True)
```

Out[53]:

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 [54]:

```python
import pandas as pd

# Assuming store2 is your DataFrame

# 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=['PromoRun', 'Count'])

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

# 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 [55]:
```
import pandas as pd

# Assuming store2 is your DataFrame

# 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=['PromoRun', 'Count'])

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

# 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 [56]:
```
84334.0*7
```

Out[56]:
```
590338.0
```

In [57]:
```
(769-544)* 135 *7
```

Out[57]:
```
212625
```

In [58]:
```
73440*7
```

Out[58]:
```
514080
```

In [59]:
```
store2['PromoRun'].describe()
```

Out[59]:
```
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 [60]:
```
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 [61]:
```
# Specify the columns to drop
columns_to_drop = ['StoreType', 'Assortment', 'CompetitionOpenSinceMonth',
                   'CompetitionOpenSinceYear', 'Promo2SinceWeek', 'Promo2SinceYear']

# 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 [62]:
```
# Read in the CSV file and name df
train= pd.read_csv("train(2).csv")
```

In [63]:
```
# Begin review of df
train.shape
```

Out[63]:
```
(1017209, 9)
```

In [64]:
```
# Df has 1017209 rows and 9 columns(cols)
# Continue review of df
# Display df's head rows
train.head(1116)
```

Out[64]:

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

1116 rows × 9 columns

In [65]:
```
# Display df's first 5 rows
train.tail(130)
```

Out[65]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
1017079	985	2	01/01/2013	0	0	0	0	a	1
1017080	986	2	01/01/2013	0	0	0	0	a	1
1017081	987	2	01/01/2013	0	0	0	0	a	1
1017082	989	2	01/01/2013	0	0	0	0	a	1
1017083	990	2	01/01/2013	0	0	0	0	a	1
...	...	...	...	...	...	...	...	...	...
1017204	1111	2	01/01/2013	0	0	0	0	a	1
1017205	1112	2	01/01/2013	0	0	0	0	a	1
1017206	1113	2	01/01/2013	0	0	0	0	a	1
1017207	1114	2	01/01/2013	0	0	0	0	a	1
1017208	1115	2	01/01/2013	0	0	0	0	a	1

130 rows × 9 columns

In [66]:
```
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 [67]:
```
train.nunique()
```

Out[67]:
```
Store            1115
DayOfWeek           7
Date              942
Sales           21734
Customers        4086
Open                2
Promo               2
StateHoliday        5
SchoolHoliday       2
dtype: int64
```

In [68]:
```
train.count()
```

Out[68]:
```
Store            1017209
DayOfWeek        1017209
Date             1017209
Sales            1017209
Customers        1017209
Open             1017209
Promo            1017209
StateHoliday     1017209
SchoolHoliday    1017209
dtype: int64
```

In [69]:
```
train["Open"].value_counts()
```

Out[69]:
```
Open
1    844392
0    172817
Name: count, dtype: int64
```

In [70]:
```
train["StateHoliday"].value_counts()
```

Out[70]:
```
StateHoliday
0    855087
0    131072
a     20260
b      6690
c      4100
Name: count, dtype: int64
```

In [71]:
```
# Replace 0 and '0' with 'd' in the StateHoliday column of df2
train['StateHoliday'].replace({0: 'd', '0': 'd'}, inplace=True)
```

In [72]:
```
train["StateHoliday"].value_counts()
```

Out[72]:
```
StateHoliday
d    986159
a     20260
b      6690
c      4100
Name: count, dtype: int64
```

In [73]:
```
# 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 [74]:

```
# 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 [75]:

```
# 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 [76]:
```
train["DayOfWeek"].value_counts()
```

Out[76]:
```
DayOfWeek
5    145845
4    145845
3    145665
2    145664
1    144730
7    144730
6    144730
Name: count, dtype: int64
```

In [77]:
```
train["Date"].value_counts()
```

Out[77]:
```
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 [78]:
```
train["Promo"].value_counts()
```

Out[78]:
```
Promo
0    629129
1    388080
Name: count, dtype: int64
```

In [79]:
```
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 [80]:
```
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 [81]:
```
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', '28/08/2014', '27/08/2014', '25/08/2014', '30/12/2014', '24/08/2014', '23/08/2014', '22/08/20
14', '21/08/2014', '20/08/2014', '19/08/2014', '18/08/2014', '17/08/2014', '16/08/2014', '05/09/2014', '06/09/2014',
'07/09/2014', '08/09/2014', '27/09/2014', '26/09/2014', '25/09/2014', '24/09/2014', '23/09/2014', '22/09/2014', '21/0
9/2014', '20/09/2014', '19/09/2014', '18/09/2014', '17/09/2014', '16/09/2014', '15/09/2014', '14/09/2014', '13/09/201
4', '12/09/2014', '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', '16/07/2014', '15/07/2014', '14/07/2014', '13/07/2014', '12/07/2014', '11/0
7/2014', '10/07/2014', '09/07/2014', '08/07/2014', '07/07/2014', '06/07/2014', '05/07/2014', '04/07/2014', '03/07/201
4', '02/07/2014', '01/07/2014', '20/07/2014', '22/07/2014', '12/08/2014', '23/07/2014', '11/08/2014', '10/08/2014',
'09/08/2014', '08/08/2014', '07/08/2014', '06/08/2014', '05/08/2014', '04/08/2014', '03/08/2014', '02/08/2014', '01/0
8/2014', '31/07/2014', '30/07/2014', '29/07/2014', '28/07/2014', '27/07/2014', '26/07/2014', '25/07/2014', '24/07/201
4', '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', '30/11/2014', '29/11/2014', '28/11/2014', '27/11/2014', '26/11/2014', '25/11/2014', '24/11/2014', '23/1
1/2014', '22/11/2014', '21/11/2014', '20/11/2014', '19/11/2014', '18/11/2014', '17/11/2014', '06/12/2014', '09/12/201
4', '15/11/2014', '10/12/2014', '29/12/2014', '28/12/2014', '27/12/2014', '26/12/2014', '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', '16/11/2014', '14/11/2014', '01/10/2014', '22/10/201
4', '20/10/2014', '19/10/2014', '18/10/2014', '17/10/2014', '16/10/2014', '15/10/2014', '14/10/2014', '13/10/2014',
'12/10/2014', '11/10/2014', '10/10/2014', '09/10/2014', '08/10/2014', '07/10/2014', '06/10/2014', '05/10/2014', '04/1
0/2014', '03/10/2014', '02/10/2014', '21/10/2014', '23/10/2014', '13/11/2014', '24/10/2014', '12/11/2014', '11/11/201
4', '10/11/2014', '09/11/2014', '08/11/2014', '07/11/2014', '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', '27/10/2014', '26/10/2014', '25/1
0/2014', '08/12/2014']
```

In [82]:
```
train["Store"].value_counts()
```

Out[82]:
```
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 [83]:
```
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 [84]:
```
train.describe()
```

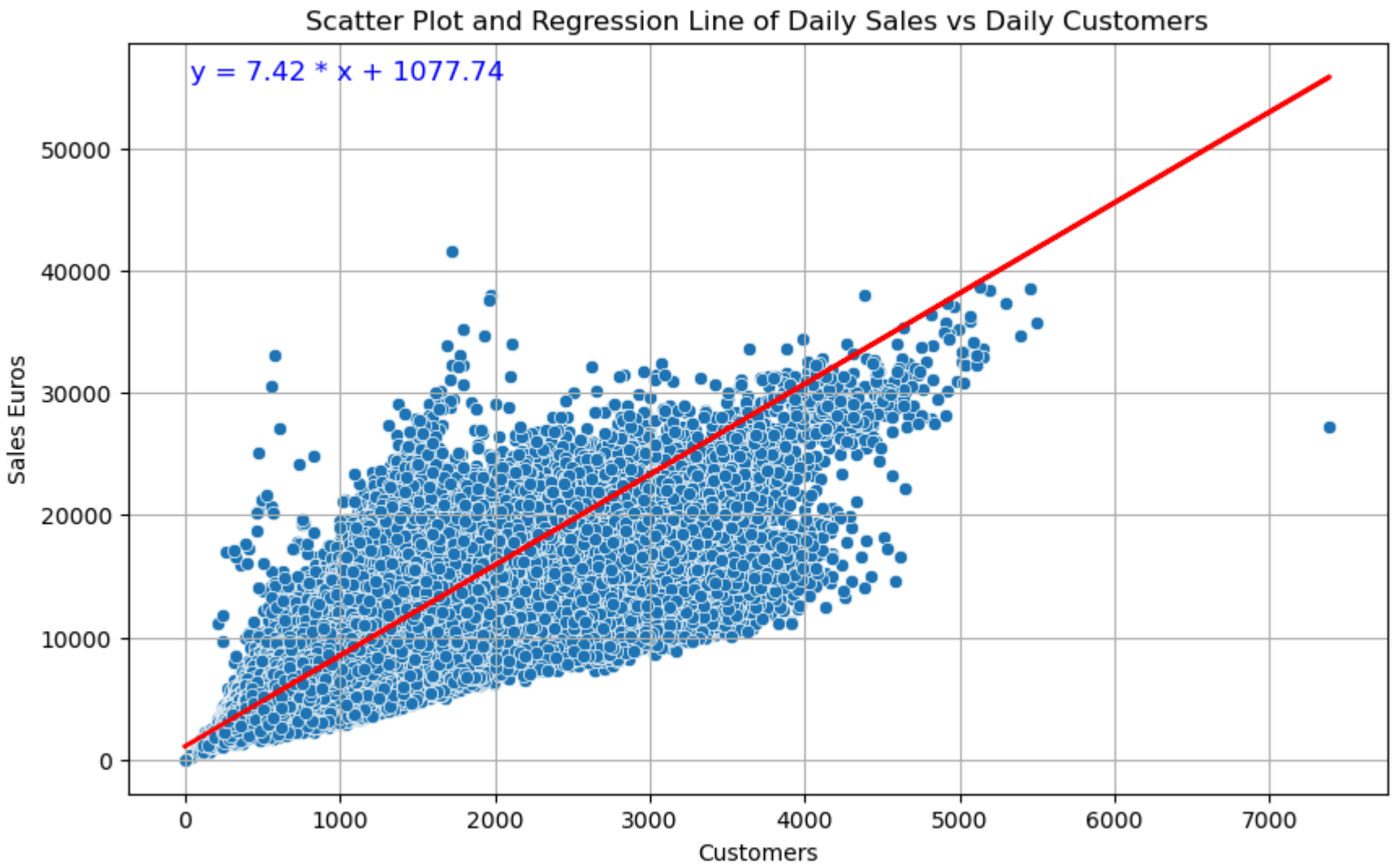
Out[84]:

	Store	DayOfWeek	Sales	Customers	Open	Promo	SchoolHoliday
count	1.017209e+06	1.017209e+06	1.017209e+06	1.017209e+06	1.017209e+06	1.017209e+06	1.017209e+06
mean	5.584297e+02	3.998341e+00	5.773819e+03	6.331459e+02	8.301067e-01	3.815145e-01	1.786467e-01
std	3.219087e+02	1.997391e+00	3.849926e+03	4.644117e+02	3.755392e-01	4.857586e-01	3.830564e-01
min	1.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	2.800000e+02	2.000000e+00	3.727000e+03	4.050000e+02	1.000000e+00	0.000000e+00	0.000000e+00
50%	5.580000e+02	4.000000e+00	5.744000e+03	6.090000e+02	1.000000e+00	0.000000e+00	0.000000e+00
75%	8.380000e+02	6.000000e+00	7.856000e+03	8.370000e+02	1.000000e+00	1.000000e+00	0.000000e+00
max	1.115000e+03	7.000000e+00	4.155100e+04	7.388000e+03	1.000000e+00	1.000000e+00	1.000000e+00

In [85]:

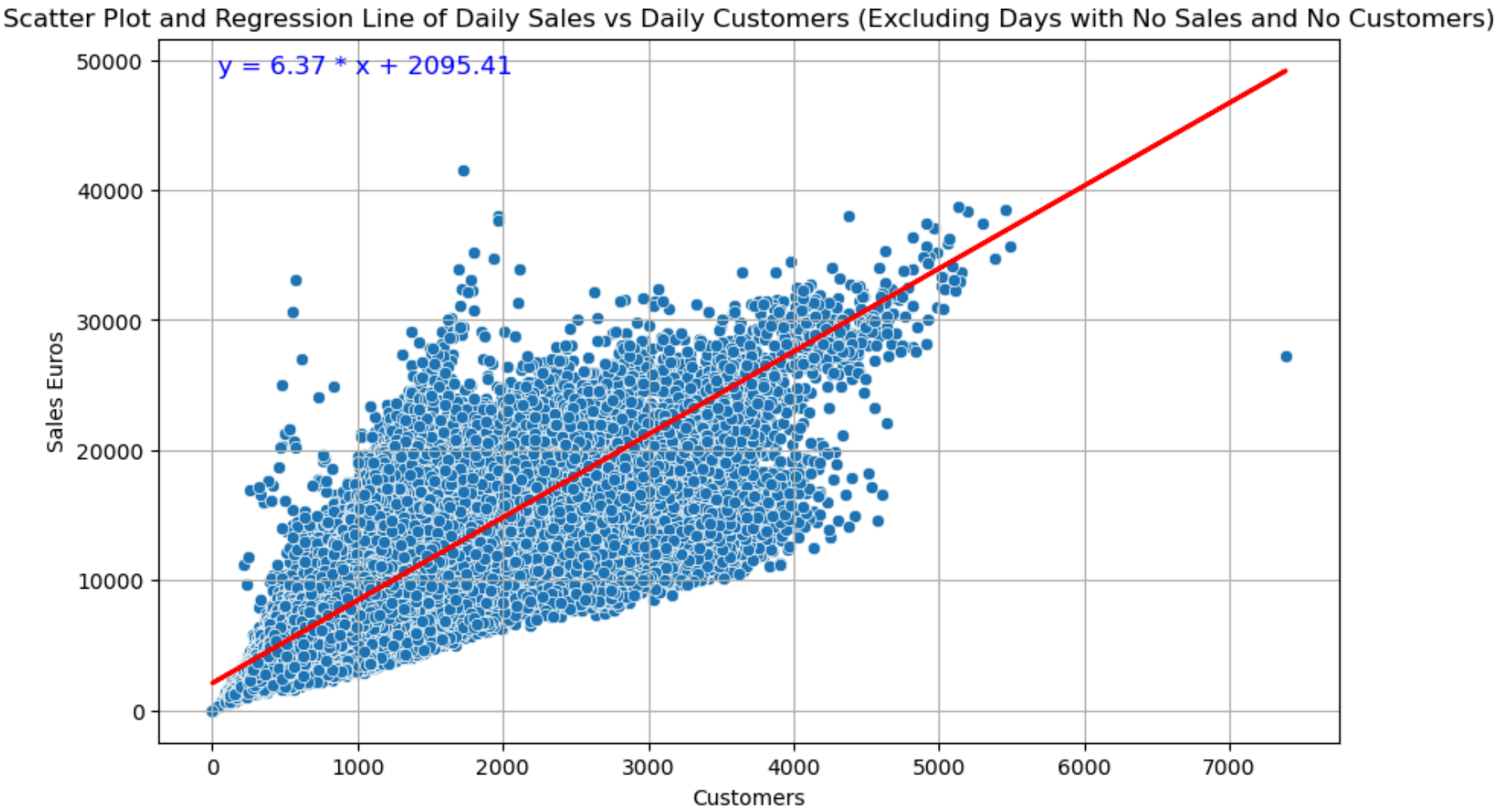
```
import statsmodels.api as sm

# 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=12, color='blue')

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

# Show plot
```

```
plt.grid(True)
plt.show()
```



In [86]:
```
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=12, color='blue')

# Set plot labels and title
plt.xlabel('Customers')
plt.ylabel('Sales Euros')
plt.title('Scatter Plot and Regression Line of Daily Sales vs Daily Customers (Excluding Days with No Sales and No Cus

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



In [87]:
```
# 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 [88]:
```
27190/7388
```

Out[88]:
```
3.680292365998917
```

In [89]:
```
26190/3788
```

Out[89]:
```
6.913938753959873
```

In [90]:
```
6.37*3788+2095
```

Out[90]:
```
26224.56
```

In [91]:
```
# 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 [92]:
```
# 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 [93]:
```
test.shape
```

Out[93]:
```
(41088, 9)
```

In [94]:
```
test.head()
```

Out[94]:

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

In [95]:
```
test.tail()
```

Out[95]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
41083	1111	6	01/08/2015	NaN	NaN	1.0	0	0	0
41084	1112	6	01/08/2015	NaN	NaN	1.0	0	0	0
41085	1113	6	01/08/2015	NaN	NaN	1.0	0	0	0
41086	1114	6	01/08/2015	NaN	NaN	1.0	0	0	0
41087	1115	6	01/08/2015	NaN	NaN	1.0	0	0	1

In [96]:
```
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]:
```
test.nunique()
```

Out[97]:

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

In [98]:
```
856*48
```

Out[98]:
```
41088
```

In [99]:
```
test.count()
```

Out[99]:
```
Store            41088
DayOfWeek        41088
Date             41088
Sales                0
Customers            0
Open             41077
Promo            41088
StateHoliday     41088
SchoolHoliday    41088
dtype: int64
```

In [100…
```
test['Open'].value_counts()
```

Out[100]:
```
Open
1.0    35093
0.0     5984
Name: count, dtype: int64
```

In [101…
```
test['Promo'].value_counts()
```

Out[101]:
```
Promo
0    24824
1    16264
Name: count, dtype: int64
```

In [102…
```
test['StateHoliday'].value_counts()
```

Out[102]:
```
StateHoliday
0    40908
a      180
Name: count, dtype: int64
```

In [103…
```
test['SchoolHoliday'].value_counts()
```

Out[103]:
```
SchoolHoliday
0    22866
1    18222
Name: count, dtype: int64
```

In [104…
```
# 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 [105…
```
test['StateHoliday'].value_counts()
```

Out[105]:
```
StateHoliday
0    40908
a      180
Name: count, dtype: int64
```

In [106…
```
# 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 [107…
```
test['StateHoliday'].value_counts()
```

Out[107]:
```
StateHoliday
d    40908
a      180
Name: count, dtype: int64
```

In [108…
```
test.describe()
```

Out[108]:

	Store	DayOfWeek	Sales	Customers	Open	Promo	SchoolHoliday
count	41088.000000	41088.000000	0.0	0.0	41077.000000	41088.000000	41088.000000
mean	555.899533	3.979167	NaN	NaN	0.854322	0.395833	0.443487
std	320.274496	2.015481	NaN	NaN	0.352787	0.489035	0.496802
min	1.000000	1.000000	NaN	NaN	0.000000	0.000000	0.000000
25%	279.750000	2.000000	NaN	NaN	1.000000	0.000000	0.000000
50%	553.500000	4.000000	NaN	NaN	1.000000	0.000000	0.000000
75%	832.250000	6.000000	NaN	NaN	1.000000	1.000000	1.000000
max	1115.000000	7.000000	NaN	NaN	1.000000	1.000000	1.000000

In [109…

```
# 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 [110…

```
# 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 [111…

```
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 [112…

```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 [113…

```
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 [114…
```
agg_df.head()
```

Out[114]:

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

In [115…
```
agg_df.nunique()
```

Out[115]:
```
Store            1115
DayOfWeek           7
Date              990
Sales           21734
Customers        4085
Open                2
Promo               2
StateHoliday        4
SchoolHoliday       2
dtype: int64
```

In [116…
```
agg_df.count()
```

Out[116]:
```
Store            1058297
DayOfWeek        1058297
Date             1058297
Sales            1017209
Customers        1017209
Open             1058297
Promo            1058297
StateHoliday     1058297
SchoolHoliday    1058297
dtype: int64
```

In [117…
```
agg_df.head()
```

Out[117]:

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

In [118…
```
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 [119…

```
# 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 [120…
```
merged_df.nunique()
```

Out[120]:
```
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 [121…
```
merged_df.head().T
```

Out[121]:

	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 [122…

```
merged_df.count()
```

Out[122]:
```
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 [123…
```python
import numpy as np

# Assuming 'train' is your DataFrame containing sales data
# Filter sales for 14 June 2014 (Saturday)
sales_2014 = train[(train['Date'] == '14/06/2014') & (train['DayOfWeek'] == 6)]['Sales']

# Filter sales for 13 June 2015 (Saturday)
sales_2015 = train[(train['Date'] == '13/06/2015') & (train['DayOfWeek'] == 6)]['Sales']

# Calculate RMSPE
def rmspe(y_true, y_pred):
    rmspe = np.sqrt(np.mean(((y_true - y_pred) / y_true)**2))
    return rmspe

rmspe_value = rmspe(sales_2014, sales_2015)

print("RMSPE between sales on 14 June 2014 and 13 June 2015 (Saturdays):", rmspe_value)
```

```
RMSPE between sales on 14 June 2014 and 13 June 2015 (Saturdays): nan
```

In [124…
```python
import numpy as np

# Assuming 'train' is your DataFrame containing sales data
# Filter sales for 14 June 2014
sales_2014 = train[train['Date'] == '14/06/2014']['Sales']

# Filter sales for 13 June 2015
```

```
sales_2015 = train[train['Date'] == '13/06/2015']['Sales']

# Calculate RMSPE
def rmspe(y_true, y_pred):
    rmspe = np.sqrt(np.mean(((y_true - y_pred) / y_true)**2))
    return rmspe

rmspe_value = rmspe(sales_2014, sales_2015)

print("RMSPE between sales on 14 June 2014 and 13 June 2015:", rmspe_value)
```

```
RMSPE between sales on 14 June 2014 and 13 June 2015: nan
```

In [125…

```
print("Missing values in sales_2014:", sales_2014.isnull().sum())
print("Missing values in sales_2015:", sales_2015.isnull().sum())

print("Zero values in sales_2014:", (sales_2014 == 0).sum())
print("Zero values in sales_2015:", (sales_2015 == 0).sum())
```

```
Missing values in sales_2014: 0
Missing values in sales_2015: 0
Zero values in sales_2014: 2
Zero values in sales_2015: 1
```

In [126…

```
# Calculate RMSPE
def rmspe(y_true, y_pred):
    rmspe = np.sqrt(np.mean(((y_true - y_pred) / y_true)**2))
    return rmspe

# Check if any sales values are zero in y_true
zero_sales_2014 = (sales_2014 == 0).any()
zero_sales_2015 = (sales_2015 == 0).any()

print("Any zero sales values in sales_2014:", zero_sales_2014)
print("Any zero sales values in sales_2015:", zero_sales_2015)
```

```
Any zero sales values in sales_2014: True
Any zero sales values in sales_2015: True
```

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 each date
merged_df['DateIdentifier'] = merged_df['Date'].rank(method='dense').astype(int)
```

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

Out[127]:

	0	1	2	3	4	5	6	7	8	9	10	
Store	1	1	1	1	1	1	1	1	1	1	1	
DayOfWeek	4	3	2	1	7	6	5	4	3	2	1	
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	2015-09-11 00:00:00	2015-09-10 00:00:00	2015-09-09 00:00:00	2015-09-08 00:00:00	2015-09-07 00:00:00	2(09 00:0(
Sales	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
Customers	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
Open	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	
Promo	1	1	1	1	0	0	0	0	0	0	0	
StateHoliday	d	d	d	d	d	d	d	d	d	d	d	
SchoolHoliday	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	1270.0	1270.0	1270.0	12
Promo2	0	0	0	0	0	0	0	0	0	0	0	
PromoInterval	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
CompetitionDistanceDecile	4	4	4	4	4	4	4	4	4	4	4	
Type_Assort	Ca	Ca	Ca	Ca	Ca	Ca	Ca	Ca	Ca	Ca	Ca	
PromoRun	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
DateIdentifier	990	989	988	987	986	985	984	983	982	981	980	

In [128…

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

Out[128]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday	CompetitionDistance	Promo2	PromoInte
1058292	1110	6	2013-01-05	3156.0	428.0	1.0	0	d	0	900.0	0	
1058293	1110	5	2013-01-04	3933.0	537.0	1.0	0	d	1	900.0	0	
1058294	1110	4	2013-01-03	3508.0	491.0	1.0	0	d	1	900.0	0	
1058295	1110	3	2013-01-02	4126.0	507.0	1.0	0	d	1	900.0	0	
1058296	1110	2	2013-01-01	0.0	0.0	0.0	0	a	1	900.0	0	

In [129…
```
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…
```
merged_df["Open"].value_counts()
```

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

In [131…
```
merged_df["Promo2"].value_counts()
```

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

In [132…
```
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…

```
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…
```
merged_df['P2C'] = merged_df['PromoRun'] * 7 + merged_df['DateIdentifier']-943
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…
```
merged_df['P2cal'] = merged_df['PromoRun'] * 7 + merged_df['DateIdentifier']-943
```

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

In [139…
```
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 else row['P2cal'], axis=1)

# 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…

```
# 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[143]:

	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 [144…

```
common_stores_df.tail().T
```

Out[144]:

	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 [145…

```
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 [146…

```
common_stores_df.nunique()
```

Out[146]:
```
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 [147…
```
common_stores_df["CommonP2cal"].value_counts()
```

Out[147]:
```
CommonP2cal
1    138653
0     70673
Name: count, dtype: int64
```

In [148…
```
# 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  PromoInterval  \
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    P2C  \
12734                          6          Da      43.0             942  300.0
12735                          6          Da      43.0             941  299.0
12736                          6          Da      43.0             940  298.0
12737                          6          Da      43.0             939  297.0
12738                          6          Da      43.0             938  296.0

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

In [149…

```
# 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 [150…

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

```
# 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 [151…

```
# Pivot the DataFrame to have 'CommonP2cal' as columns
pivot_sales = grouped_sales.pivot(index='Store', columns='CommonP2cal', values='Sales')

# 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 [152…

```
# 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 [153…
```
1358983.46/225
```

Out[153]:
```
6039.926488888888
```

In [154…
```
1474976.78/225
```

Out[154]:
```
6555.452355555556
```

In [155…
```
# Calculate sales increase by Promo2 (8.54%)
(6555.452355555556-6039.926488888888)/6039.926488888888
```

Out[155]:
```
0.08535300348688589
```

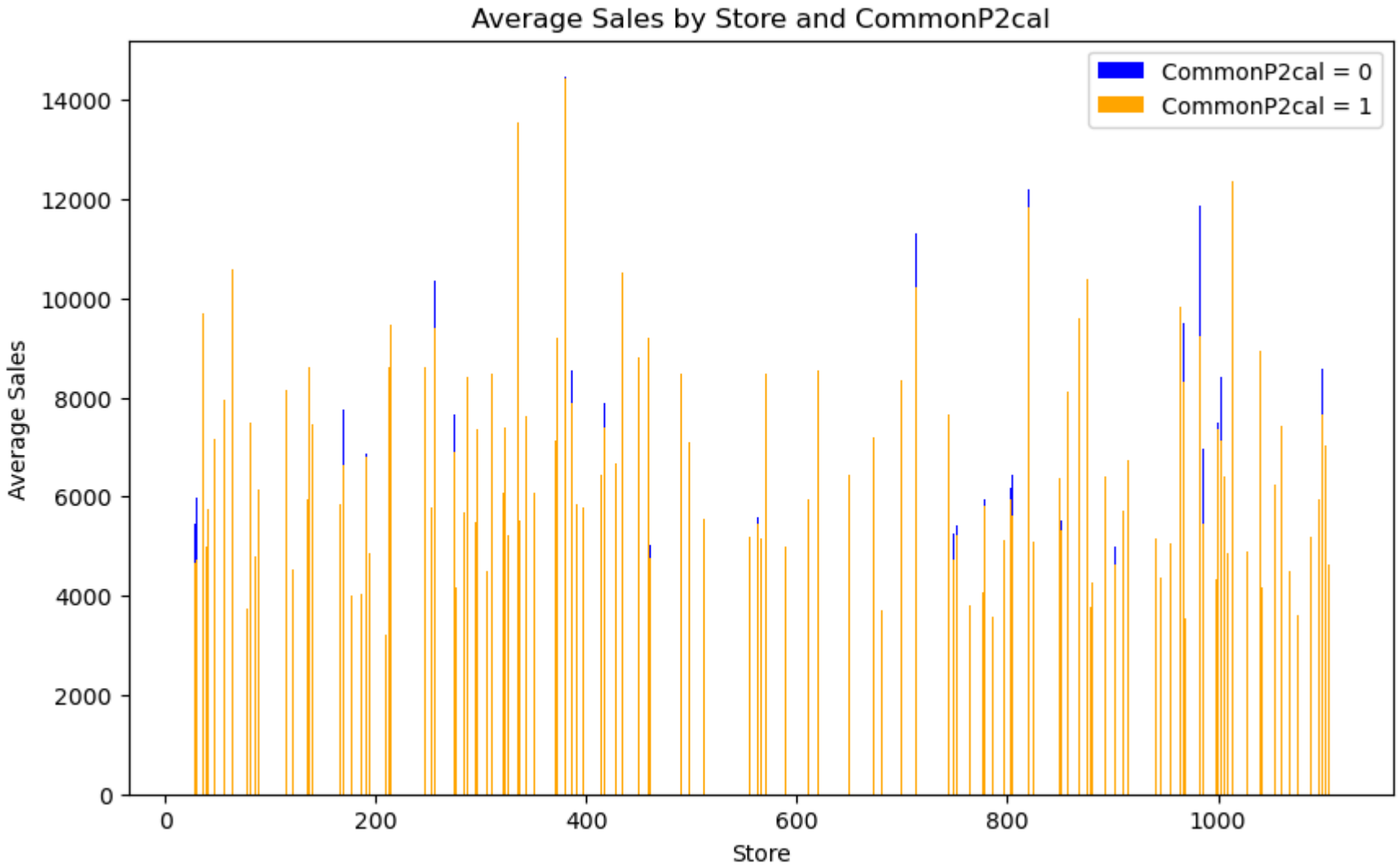
In [156…
```
# Calculate the total difference between average sales at 0 and 1 for all stores
total_difference = pivot_sales['Sales_Difference'].sum()

# 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 [ ]:
```

```

In [157…
```
import matplotlib.pyplot as plt

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

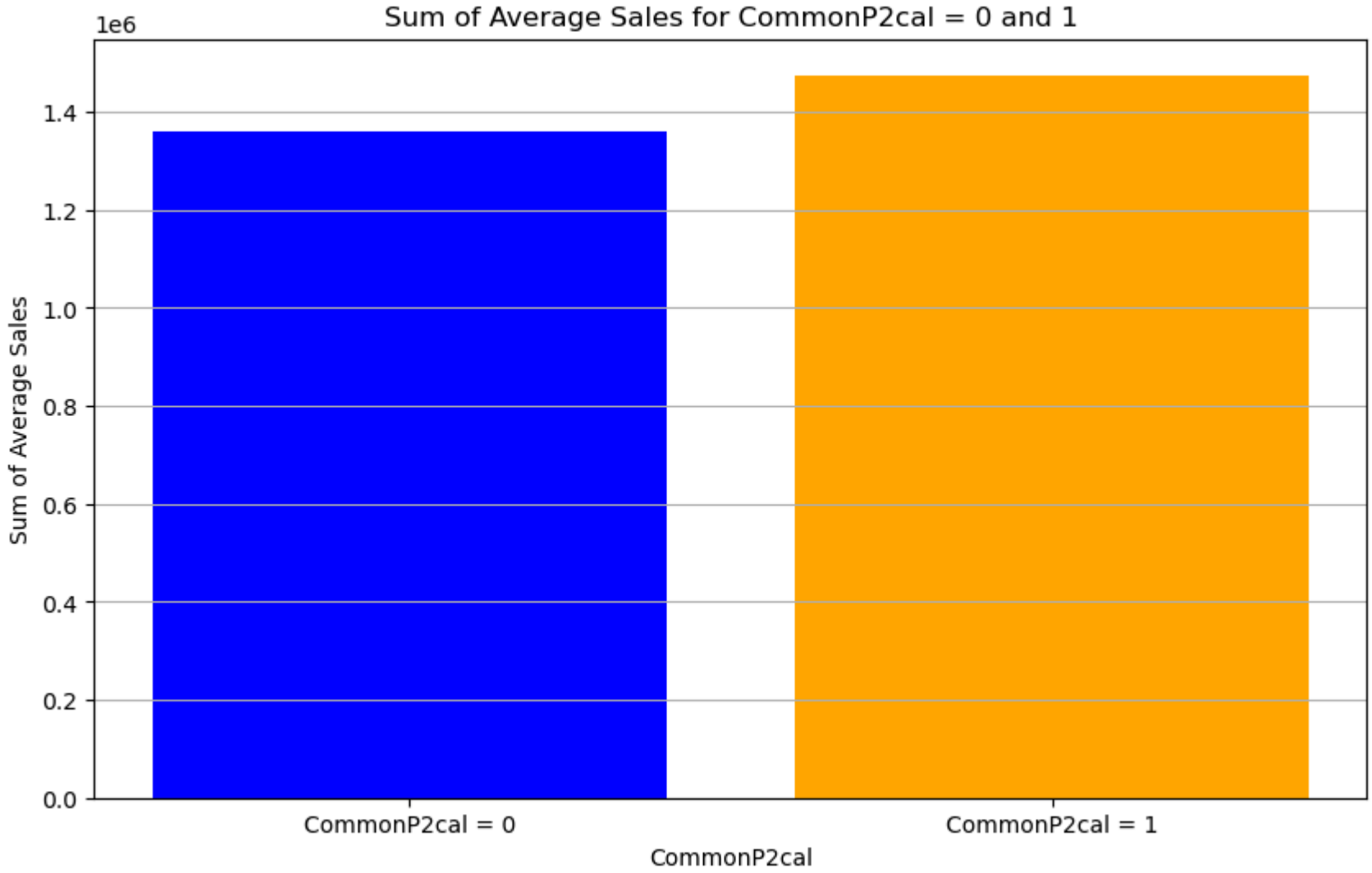
```
# Filter the DataFrame for stores with CommonP2cal equal to 1
common_p2cal_1 = grouped_sales[grouped_sales['CommonP2cal'] == 1]

# 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', label='CommonP2cal = 0')

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

# 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 [158…

```
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_average_sales_1], color=['blue', 'orange'])

# 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 [159…
```
# 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 [160…
```
merged_df.tail().T
```

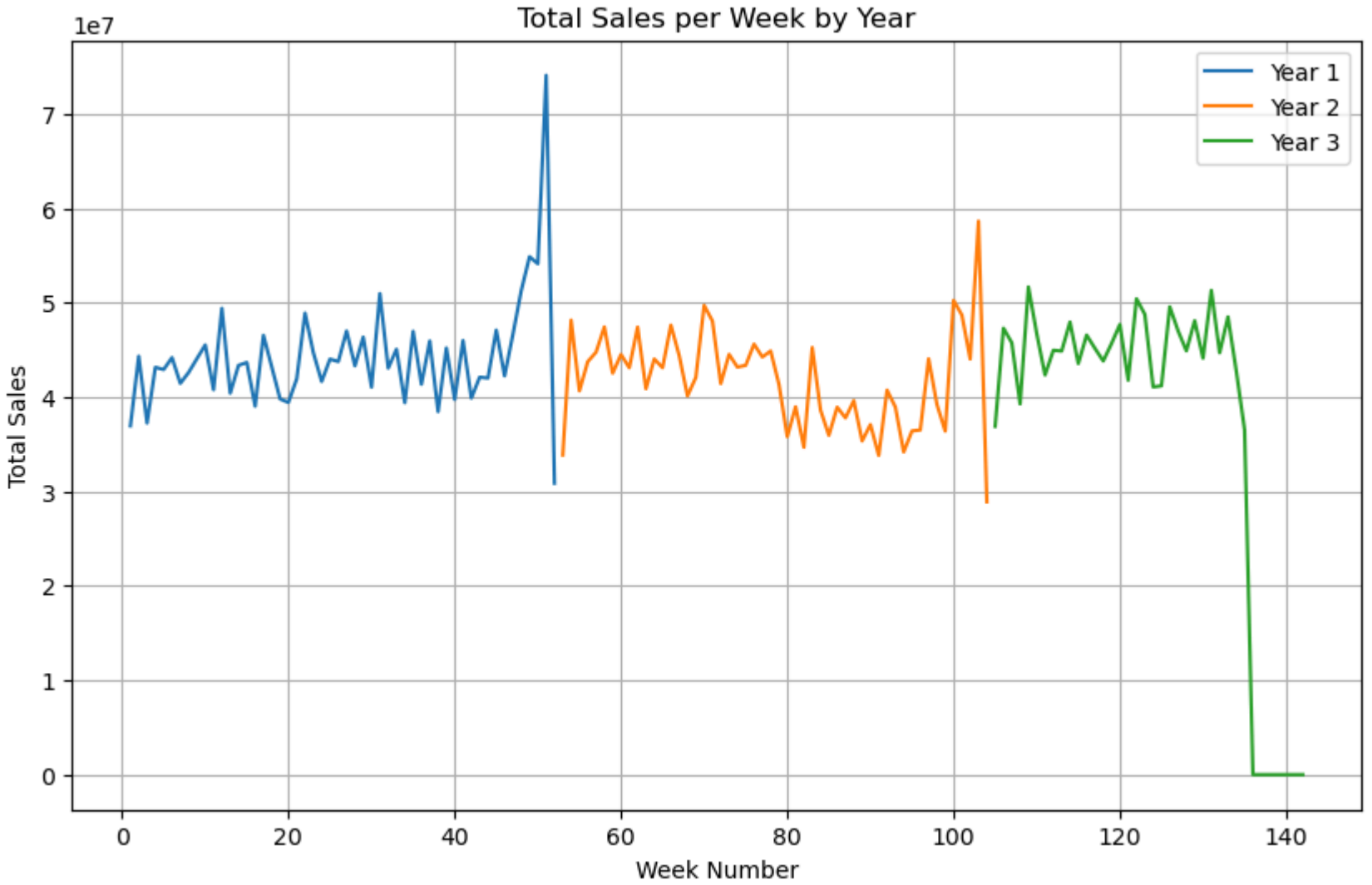
Out[160]:

	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 [161…

```
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'].sum()
```

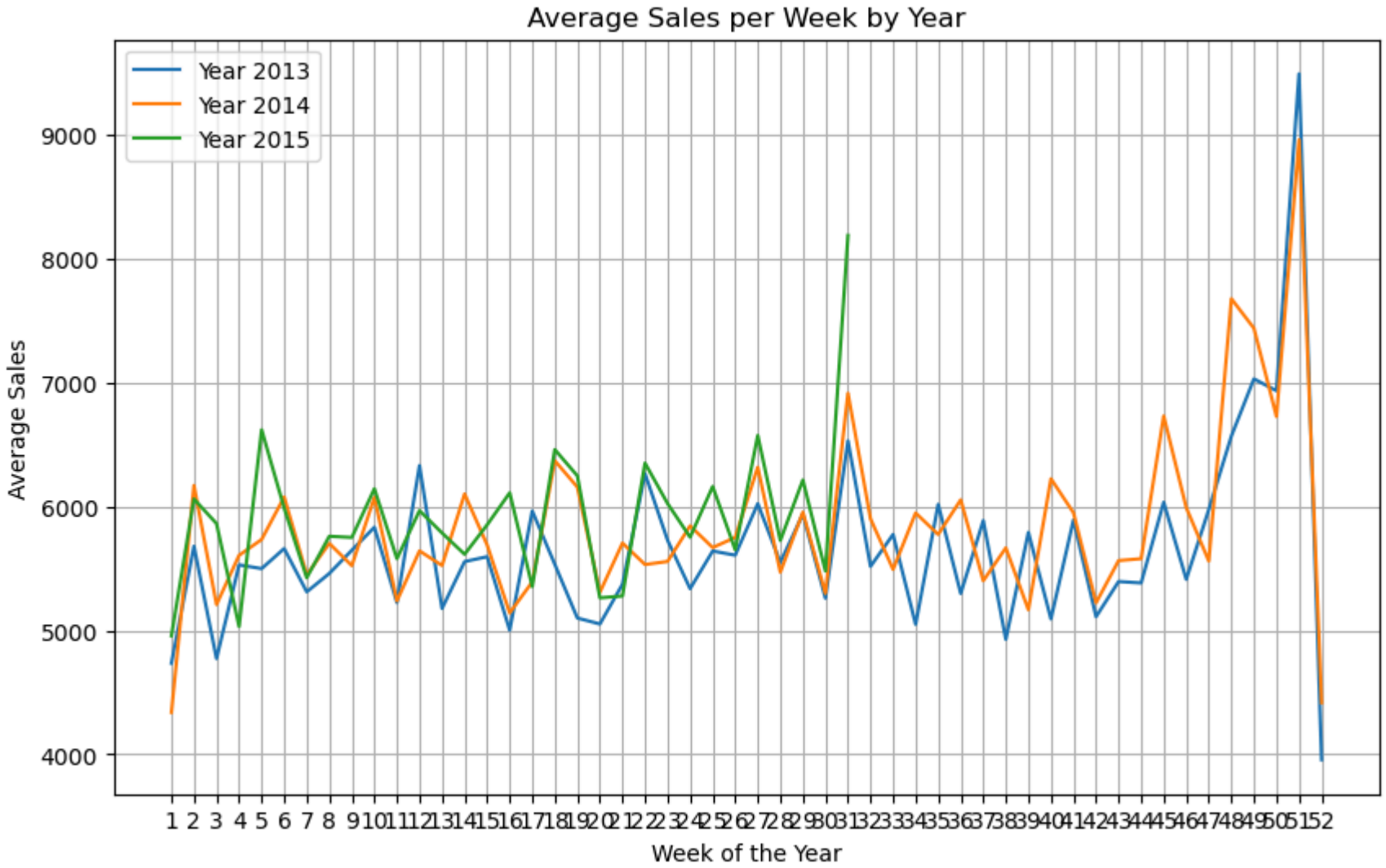
```
# Plot the total sales per week for each year
plt.figure(figsize=(10, 6))

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 [162…

```
# 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'].mean().reset_index()
```

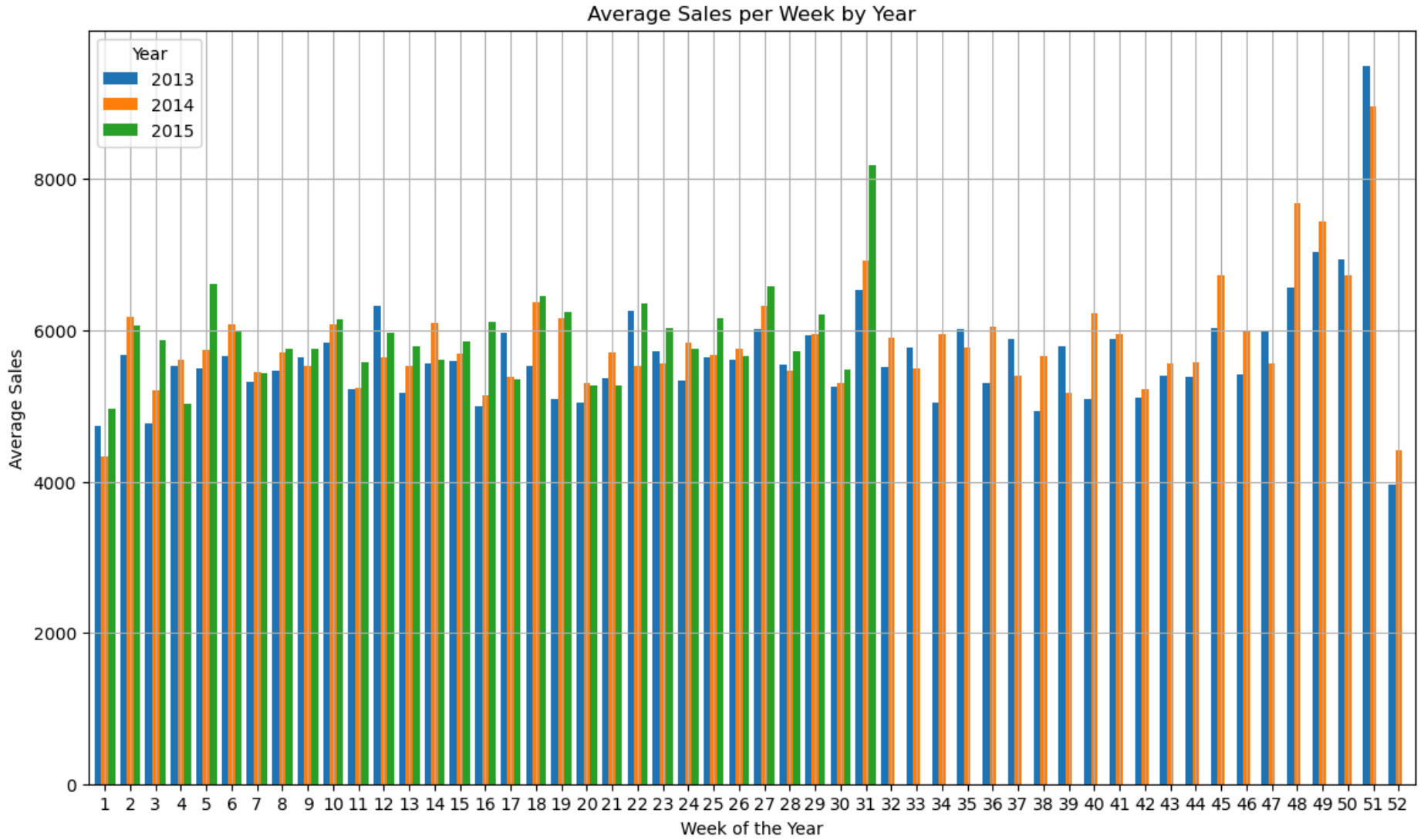
```
# Plot the average sales per week for each year
plt.figure(figsize=(10, 6))

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 {year}')

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 [163…

```
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'].mean().reset_index()

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

# 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 readability
plt.legend(title='Year')

plt.show()
```



In [164…
```
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 [165…

```
merged_df.describe().T
```

Out[165]:

	count	mean	min	25%	50%	75%	max	std
Store	1058297.0	558.331493	1.0	280.0	558.0	837.0	1115.0	321.845581
DayOfWeek	1058297.0	3.997596	1.0	2.0	4.0	6.0	7.0	1.998099
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	2015-01-12 00:00:00	2015-09-17 00:00:00	NaN
Sales	1017209.0	5773.818972	0.0	3727.0	5744.0	7856.0	41551.0	3849.926175
Customers	1017209.0	633.142407	0.0	405.0	609.0	837.0	5494.0	464.373973
Open	1058297.0	0.831048	0.0	1.0	1.0	1.0	1.0	0.374709
Promo	1058297.0	0.38207	0.0	0.0	0.0	1.0	1.0	0.485894
SchoolHoliday	1058297.0	0.188929	0.0	0.0	0.0	0.0	1.0	0.391452
CompetitionDistance	1058297.0	5416.793017	20.0	710.0	2330.0	6870.0	75860.0	7687.199185
Promo2	1058297.0	0.503671	0.0	0.0	1.0	1.0	1.0	0.499987
PromoInterval	1058297.0	0.813569	0.0	0.0	1.0	1.0	3.0	0.981788
CompetitionDistanceDecile	1058297.0	5.483524	1.0	3.0	5.0	8.0	10.0	2.895327
PromoRun	1058297.0	88.900305	0.0	0.0	13.0	186.0	313.0	106.671614
DateIdentifier	1058297.0	485.492282	1.0	238.0	475.0	742.0	990.0	285.926009
P2C	1058297.0	164.794417	-942.0	-479.0	-72.0	784.0	2238.0	799.138669
P2cal	1058297.0	0.436892	0.0	0.0	0.0	1.0	1.0	0.496002
WeekNumber	1058297.0	69.785393	1.0	34.0	68.0	106.0	142.0	40.84703
Year	1058297.0	2013.880448	2013.0	2013.0	2014.0	2015.0	2015.0	0.795714
WeekOfYear	1058297.0	24.002071	1.0	12.0	23.0	35.0	52.0	14.332119

In [166…

```
dfz = merged_df[merged_df['Sales'] != 0]

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

Out[166]:

	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 [167…

```
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 [168…

```python
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", 'Open', 'Promo2'])

dfzc.head().T
```

Out[168]:

	0	1	2	3	4
Sales	NaN	NaN	NaN	NaN	NaN
Store	1	1	1	1	1
DayOfWeek	4	3	2	1	7
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 [ ]:

In [169…

```
dfzc.describe().T
```

Out[169]:

	count	mean	std	min	25%	50%	75%	max
Sales	844338.0	6955.959134	3103.815515	46.0	4859.0	6369.0	8360.0	41551.0
Store	885426.0	558.304348	321.663682	1.0	280.0	558.0	837.0	1115.0
DayOfWeek	885426.0	3.541641	1.741012	1.0	2.0	4.0	5.0	7.0
Promo	885426.0	0.444011	0.496856	0.0	0.0	0.0	1.0	1.0
SchoolHoliday	885426.0	0.205175	0.403829	0.0	0.0	0.0	0.0	1.0
CompetitionDistance	885426.0	5440.909720	7773.780012	20.0	710.0	2330.0	6880.0	75860.0
PromoInterval	885426.0	0.811643	0.981459	0.0	0.0	1.0	1.0	3.0
CompetitionDistanceDecile	885426.0	5.483130	2.896799	1.0	3.0	5.0	8.0	10.0
PromoRun	885426.0	88.646062	106.564424	0.0	0.0	13.0	186.0	313.0
DateIdentifier	885426.0	489.271051	287.997254	1.0	240.0	479.0	749.0	990.0
P2cal	885426.0	0.436699	0.495977	0.0	0.0	0.0	1.0	1.0
WeekNumber	885426.0	70.370649	41.140541	1.0	35.0	69.0	107.0	142.0
Year	885426.0	2013.889520	0.798811	2013.0	2013.0	2014.0	2015.0	2015.0
WeekOfYear	885426.0	24.115617	14.264523	1.0	12.0	24.0	35.0	52.0

In [170…

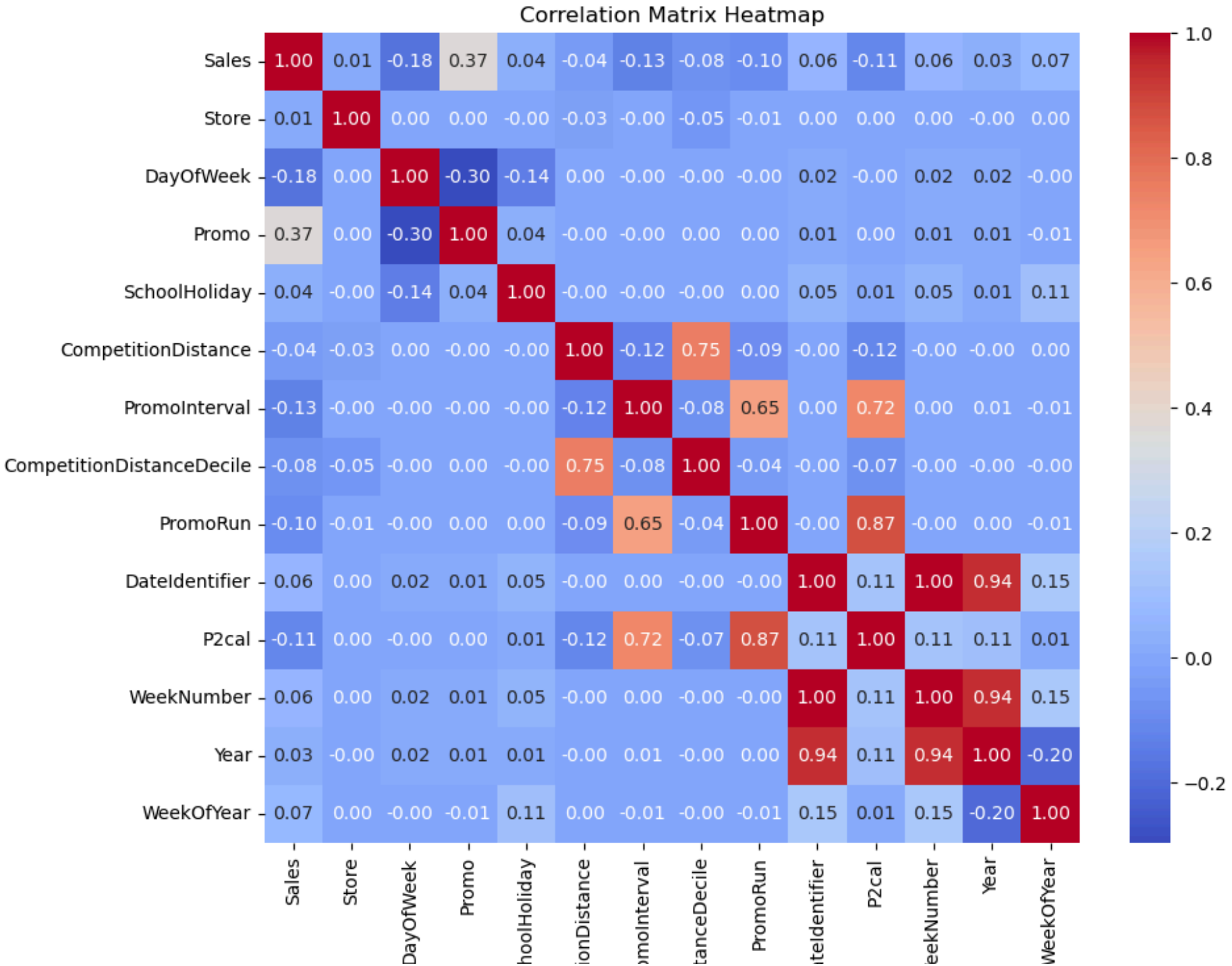
```
import pandas as pd

# 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[170]:

	Sales	Store	DayOfWeek	Promo	SchoolHoliday	CompetitionDistance	PromoInterval	CompetitionDista
Sales	1.000000	0.007723	-0.178753	0.368199	0.038635	-0.036401	-0.131638	
Store	0.007723	1.000000	0.000233	0.000021	-0.000392	-0.029640	-0.000124	
DayOfWeek	-0.178753	0.000233	1.000000	-0.297459	-0.144800	0.004690	-0.001479	
Promo	0.368199	0.000021	-0.297459	1.000000	0.036575	-0.002074	-0.000173	
SchoolHoliday	0.038635	-0.000392	-0.144800	0.036575	1.000000	-0.004055	-0.000920	
CompetitionDistance	-0.036401	-0.029640	0.004690	-0.002074	-0.004055	1.000000	-0.123500	
PromoInterval	-0.131638	-0.000124	-0.001479	-0.000173	-0.000920	-0.123500	1.000000	
CompetitionDistanceDecile	-0.075098	-0.049341	-0.000626	0.000307	-0.000368	0.750234	-0.083868	
PromoRun	-0.098959	-0.009657	-0.002512	0.000173	0.001604	-0.085378	0.647784	
DateIdentifier	0.062734	0.000135	0.015778	0.008914	0.050607	-0.001780	0.003614	
P2cal	-0.114502	0.001870	-0.000315	0.000978	0.007732	-0.118457	0.723856	
WeekNumber	0.062167	0.000134	0.015487	0.009777	0.051133	-0.001803	0.003621	
Year	0.033802	-0.000212	0.016003	0.011920	0.013229	-0.002457	0.005795	
WeekOfYear	0.074203	0.001002	-0.001934	-0.006514	0.108950	0.001956	-0.006432	

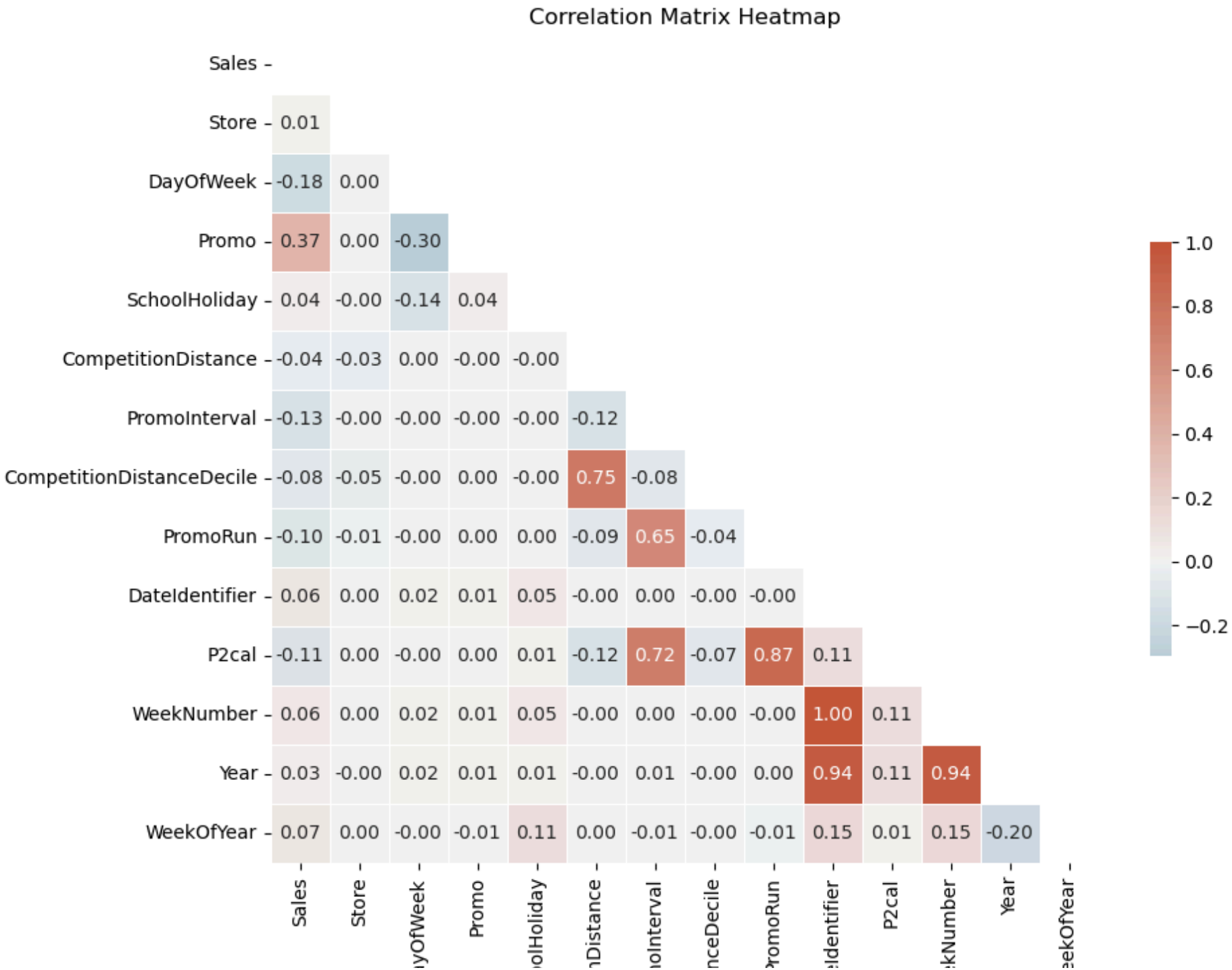
In [171…

```
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()
```


Correlation Matrix Heatmap
Sales 1.00 0.01 -0.18 0.37 0.04 -0.04 -0.13 -0.08 -0.10 0.06 -0.11 0.06 0.03 0.07
Store 0.01 1.00 0.00 0.00 -0.00 -0.03 -0.00 -0.05 -0.01 0.00 0.00 0.00 -0.00 0.00
DayOfWeek -0.18 0.00 1.00 -0.30 -0.14 0.00 -0.00 -0.00 -0.00 0.02 -0.00 0.02 0.02 -0.00
Promo 0.37 0.00 -0.30 1.00 0.04 -0.00 -0.00 0.00 0.00 0.01 0.00 0.01 0.01 -0.01
SchoolHoliday 0.04 -0.00 -0.14 0.04 1.00 -0.00 -0.00 -0.00 0.00 0.05 0.01 0.05 0.01 0.11
CompetitionDistance -0.04 -0.03 0.00 -0.00 -0.00 1.00 -0.12 0.75 -0.09 -0.00 -0.12 -0.00 -0.00 0.00
PromoInterval -0.13 -0.00 -0.00 -0.00 -0.00 -0.12 1.00 -0.08 0.65 0.00 0.72 0.00 0.01 -0.01
CompetitionDistanceDecile -0.08 -0.05 -0.00 0.00 -0.00 0.75 -0.08 1.00 -0.04 -0.00 -0.07 -0.00 -0.00 -0.00
PromoRun -0.10 -0.01 -0.00 0.00 0.00 -0.09 0.65 -0.04 1.00 -0.00 0.87 -0.00 0.00 -0.01
DateIdentifier 0.06 0.00 0.02 0.01 0.05 -0.00 0.00 -0.00 -0.00 1.00 0.11 1.00 0.94 0.15
P2cal -0.11 0.00 -0.00 0.00 0.01 -0.12 0.72 -0.07 0.87 0.11 1.00 0.11 0.11 0.01
WeekNumber 0.06 0.00 0.02 0.01 0.05 -0.00 0.00 -0.00 -0.00 1.00 0.11 1.00 0.94 0.15
Year 0.03 -0.00 0.02 0.01 0.01 -0.00 0.01 -0.00 0.00 0.94 0.11 0.94 1.00 -0.20
WeekOfYear 0.07 0.00 -0.00 -0.01 0.11 0.00 -0.01 -0.00 -0.01 0.15 0.01 0.15 -0.20 1.00
Sales Store DayOfWeek Promo hoolHoliday ionDistance omoInterval tanceDecile PromoRun teIdentifier P2cal eekNumber Year WeekOfYear
1.0 0.8 0.6 0.4 0.2 0.0 −0.2

In [172…

```
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, fmt=".2f")

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


Correlation Matrix Heatmap
Sales
Store 0.01
DayOfWeek -0.18 0.00
Promo 0.37 0.00 -0.30
SchoolHoliday 0.04 -0.00 -0.14 0.04
CompetitionDistance -0.04 -0.03 0.00 -0.00 -0.00
PromoInterval -0.13 -0.00 -0.00 -0.00 -0.00 -0.12
CompetitionDistanceDecile -0.08 -0.05 -0.00 0.00 -0.00 0.75 -0.08
PromoRun -0.10 -0.01 -0.00 0.00 0.00 -0.09 0.65 -0.04
DateIdentifier 0.06 0.00 0.02 0.01 0.05 -0.00 0.00 -0.00 -0.00
P2cal -0.11 0.00 -0.00 0.00 0.01 -0.12 0.72 -0.07 0.87 0.11
WeekNumber 0.06 0.00 0.02 0.01 0.05 -0.00 0.00 -0.00 -0.00 1.00 0.11
Year 0.03 -0.00 0.02 0.01 0.01 -0.00 0.01 -0.00 0.00 0.94 0.11 0.94
WeekOfYear 0.07 0.00 -0.00 -0.01 0.11 0.00 -0.01 -0.00 -0.01 0.15 0.01 0.15 -0.20
Sales
Store
DayOfWeek
Promo
SchoolHoliday
CompetitionDistance
PromoInterval
CompetitionDistanceDecile
PromoRun
DateIdentifier
P2cal
WeekNumber
Year
WeekOfYear
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

In [173…

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

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

   CompetitionDistance  PromoInterval  CompetitionDistanceDecile Type_Assort  \
0               1270.0            0.0                          4          Ca
1               1270.0            0.0                          4          Ca
2               1270.0            0.0                          4          Ca
3               1270.0            0.0                          4          Ca
4               1270.0            0.0                          4          Ca

   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
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 [174…

```
dfzc.nunique()
```

Out[174]:
```
Sales                       21733
Store                        1115
DayOfWeek                       7
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 [175…
```
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.transform(label_encoder.classes_)):
    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 [176…
```
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.transform(label_encoder.classes_)):
    print(f"{category}: {encoded_value}")
```

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

In [177…

```
dfzc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 885426 entries, 0 to 1058295
Data columns (total 16 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   Promo                     885426 non-null  int64  
 4   StateHoliday              885426 non-null  int64  
 5   SchoolHoliday             885426 non-null  int64  
 6   CompetitionDistance       885426 non-null  float64
 7   PromoInterval             885426 non-null  float64
 8   CompetitionDistanceDecile 885426 non-null  int64  
 9   Type_Assort               885426 non-null  int64  
 10  PromoRun                  885426 non-null  float64
 11  DateIdentifier            885426 non-null  int64  
 12  P2cal                     885426 non-null  int64  
 13  WeekNumber                885426 non-null  int64  
 14  Year                      885426 non-null  int64  
 15  WeekOfYear                885426 non-null  int64  
dtypes: float64(4), int64(12)
memory usage: 114.8 MB
```

In [178…
```
import pandas as pd


# Filter DataFrame to include only rows with DateIdentifier between 1 and 943
dfzc942 = dfzc[(dfzc['DateIdentifier'] >= 1) & (dfzc['DateIdentifier'] <= 942)]

dfzc942.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 844338 entries, 48 to 1058295
Data columns (total 16 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   Promo                      844338 non-null  int64  
 4   StateHoliday               844338 non-null  int64  
 5   SchoolHoliday              844338 non-null  int64  
 6   CompetitionDistance        844338 non-null  float64
 7   PromoInterval              844338 non-null  float64
 8   CompetitionDistanceDecile  844338 non-null  int64  
 9   Type_Assort                844338 non-null  int64  
 10  PromoRun                   844338 non-null  float64
 11  DateIdentifier             844338 non-null  int64  
 12  P2cal                      844338 non-null  int64  
 13  WeekNumber                 844338 non-null  int64  
 14  Year                       844338 non-null  int64  
 15  WeekOfYear                 844338 non-null  int64  
dtypes: float64(4), int64(12)
memory usage: 109.5 MB
```

In [179…
```
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 1
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', 'CompetitionDistance', 'CompetitionDistanceDecile',
    'PromoInterval', 'Type_Assort', 'PromoRun', 'P2cal', 'Year', 'WeekOfYear'
]

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'NormalizedAvgSales_{variable}'] = df[variable].map(normalized_avg_sales)

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

Out[179]:

	48	49	50	51	52	54	5
Sales	5263.000000	5020.000000	4782.000000	5011.000000	6102.000000	4364.000000	3706.00000
Store	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000
DayOfWeek	5.000000	4.000000	3.000000	2.000000	1.000000	6.000000	5.00000
Promo	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	0.00000
StateHoliday	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	3.00000
SchoolHoliday	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	0.00000
CompetitionDistance	1270.000000	1270.000000	1270.000000	1270.000000	1270.000000	1270.000000	1270.00000
PromoInterval	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
CompetitionDistanceDecile	4.000000	4.000000	4.000000	4.000000	4.000000	4.000000	4.00000
Type_Assort	5.000000	5.000000	5.000000	5.000000	5.000000	5.000000	5.00000
PromoRun	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
DateIdentifier	942.000000	941.000000	940.000000	939.000000	938.000000	936.000000	935.00000
P2cal	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
WeekNumber	135.000000	135.000000	135.000000	135.000000	134.000000	134.000000	134.00000
Year	2015.000000	2015.000000	2015.000000	2015.000000	2015.000000	2015.000000	2015.00000
WeekOfYear	31.000000	31.000000	31.000000	31.000000	30.000000	30.000000	30.00000
AvgSalesPerStore	4759.096031	4759.096031	4759.096031	4759.096031	4759.096031	4759.096031	4759.09603
StoreRank	155.000000	155.000000	155.000000	155.000000	155.000000	155.000000	155.00000
NormalizedAvgSales_DayOfWeek	1.203903	1.152020	1.145309	1.206520	1.398491	1.000000	1.20390
NormalizedAvgSales_Promo	1.387686	1.387686	1.387686	1.387686	1.387686	1.000000	1.00000
NormalizedAvgSales_StateHoliday	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000
NormalizedAvgSales_SchoolHoliday	1.044004	1.044004	1.044004	1.044004	1.044004	1.000000	1.00000
NormalizedAvgSales_CompetitionDistance	3.087343	3.087343	3.087343	3.087343	3.087343	3.087343	3.08734
NormalizedAvgSales_CompetitionDistanceDecile	1.067718	1.067718	1.067718	1.067718	1.067718	1.067718	1.06771
NormalizedAvgSales_PromoInterval	1.182585	1.182585	1.182585	1.182585	1.182585	1.182585	1.18258

	48	49	50	51	52	54	5
NormalizedAvgSales_Type_Assort	1.062482	1.062482	1.062482	1.062482	1.062482	1.062482	1.06248
NormalizedAvgSales_PromoRun	1.655080	1.655080	1.655080	1.655080	1.655080	1.655080	1.65508
NormalizedAvgSales_P2cal	1.109664	1.109664	1.109664	1.109664	1.109664	1.109664	1.10966
NormalizedAvgSales_Year	1.037487	1.037487	1.037487	1.037487	1.037487	1.037487	1.03748
NormalizedAvgSales_WeekOfYear	1.319977	1.319977	1.319977	1.319977	1.039190	1.039190	1.03919

In [180…

```
df.nunique()
```

Out[180]:

```
Sales                                      21733
Store                                       1115
DayOfWeek                                      7
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
NormalizedAvgSales_DayOfWeek                   7
NormalizedAvgSales_Promo                       2
NormalizedAvgSales_StateHoliday                4
NormalizedAvgSales_SchoolHoliday               2
NormalizedAvgSales_CompetitionDistance       655
NormalizedAvgSales_CompetitionDistanceDecile  10
NormalizedAvgSales_PromoInterval               4
NormalizedAvgSales_Type_Assort                 9
NormalizedAvgSales_PromoRun                   56
NormalizedAvgSales_P2cal                       2
NormalizedAvgSales_Year                        3
NormalizedAvgSales_WeekOfYear                 52
dtype: int64
```

In [181…

```
df.describe()
```

Out[181]:

	Sales	Store	DayOfWeek	Promo	StateHoliday	SchoolHoliday	CompetitionDistance	PromoInterval
count	844338.000000	844338.000000	844338.000000	844338.000000	844338.000000	844338.000000	844338.000000	844338.000000
mean	6955.959134	558.421374	3.520350	0.446356	2.997107	0.193578	5458.019004	0.805774
std	3103.815515	321.730861	1.723712	0.497114	0.090334	0.395102	7799.457551	0.980689
min	46.000000	1.000000	1.000000	0.000000	0.000000	0.000000	20.000000	0.000000
25%	4859.000000	280.000000	2.000000	0.000000	3.000000	0.000000	710.000000	0.000000
50%	6369.000000	558.000000	3.000000	0.000000	3.000000	0.000000	2330.000000	0.000000
75%	8360.000000	837.000000	5.000000	1.000000	3.000000	0.000000	6880.000000	1.000000
max	41551.000000	1115.000000	7.000000	1.000000	3.000000	1.000000	75860.000000	3.000000

8 rows × 30 columns

In [182…

```
# 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[182]:

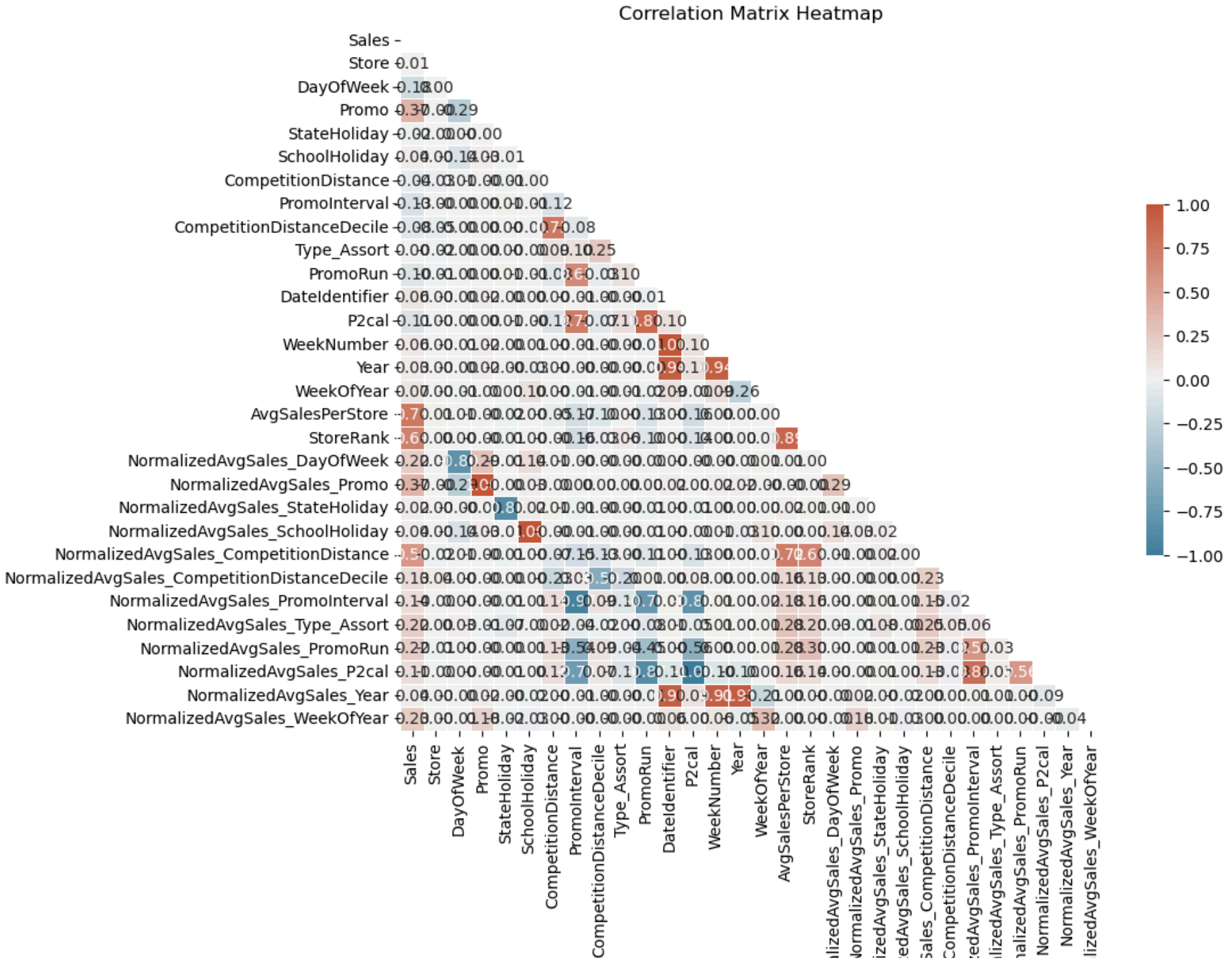
	Sales	Store	DayOfWeek	Promo	StateHoliday	SchoolHoliday	CompetitionDistan
Sales	1.000000	0.007723	-0.178753	0.368199	-0.017897	0.038635	-0.0364
Store	0.007723	1.000000	0.000343	-0.000015	-0.002697	0.000526	-0.0270
DayOfWeek	-0.178753	0.000343	1.000000	-0.289268	0.001424	-0.139319	0.0055
Promo	0.368199	-0.000015	-0.289268	1.000000	-0.004419	0.028971	-0.0023
StateHoliday	-0.017897	-0.002697	0.001424	-0.004419	1.000000	-0.006242	-0.0110
SchoolHoliday	0.038635	0.000526	-0.139319	0.028971	-0.006242	1.000000	-0.0042
CompetitionDistance	-0.036401	-0.027027	0.005535	-0.002389	-0.011068	-0.004273	1.0000
PromoInterval	-0.131638	-0.001496	-0.003161	0.000427	0.009178	-0.006449	-0.1224
CompetitionDistanceDecile	-0.075098	-0.046866	-0.000530	0.000270	0.000768	-0.003327	0.7508
Type_Assort	0.000698	-0.016799	-0.001812	0.000337	0.003837	-0.002429	0.0897
PromoRun	-0.098959	-0.010011	-0.004028	0.000703	0.009579	-0.005190	-0.0841
DateIdentifier	0.062734	0.000812	-0.004828	0.018360	-0.002328	0.004595	0.0020
P2cal	-0.114502	0.000647	-0.004099	0.002465	0.007224	-0.004776	-0.1153
WeekNumber	0.062167	0.000810	-0.005033	0.019213	-0.002303	0.005141	0.0020
Year	0.033802	0.000316	-0.001085	0.019872	-0.002794	-0.029353	0.0006
WeekOfYear	0.074203	0.001315	-0.010648	-0.003548	0.001582	0.096457	0.0036
AvgSalesPerStore	0.775939	0.009953	0.006507	-0.002656	-0.017156	0.003315	-0.0469
StoreRank	0.692339	0.000534	0.003985	-0.001471	-0.011195	0.003339	-0.0041
NormalizedAvgSales_DayOfWeek	0.224254	0.000448	-0.797102	0.290163	-0.006991	0.135767	0.0054
NormalizedAvgSales_Promo	0.368199	-0.000015	-0.289268	1.000000	-0.004419	0.028971	-0.0023
NormalizedAvgSales_StateHoliday	0.020533	0.002218	-0.003556	-0.000550	-0.871607	0.020633	0.0089
NormalizedAvgSales_SchoolHoliday	0.038635	0.000526	-0.139319	0.028971	-0.006242	1.000000	-0.0042
NormalizedAvgSales_CompetitionDistance	0.558298	-0.021343	0.005744	-0.002377	-0.013941	0.003845	-0.0651
NormalizedAvgSales_CompetitionDistanceDecile	0.127542	0.041671	0.001260	-0.000306	-0.001886	0.003193	-0.2304
NormalizedAvgSales_PromoInterval	0.135946	-0.002753	0.003372	-0.000139	-0.009887	0.007488	0.1363

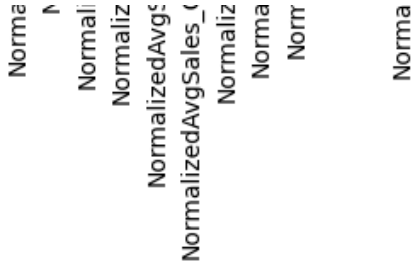
	Sales	Store	DayOfWeek	Promo	StateHoliday	SchoolHoliday	CompetitionDistan
NormalizedAvgSales_Type_Assort	0.219744	0.001491	0.027322	-0.012524	-0.067936	-0.002328	0.0221
NormalizedAvgSales_PromoRun	0.219224	-0.010809	0.001720	-0.000015	-0.004922	0.005299	0.1295
NormalizedAvgSales_P2cal	0.114502	-0.000647	0.004099	-0.002465	-0.007224	0.004776	0.1153
NormalizedAvgSales_Year	0.035408	0.000767	-0.001898	0.018310	-0.002180	-0.022677	0.0017
NormalizedAvgSales_WeekOfYear	0.232775	0.000289	-0.005167	0.179334	-0.016014	-0.026177	0.0008

30 rows × 30 columns

In [183…

```
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, fmt=".2f")

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


Correlation Matrix Heatmap
Sales
Store
DayOfWeek
Promo
StateHoliday
SchoolHoliday
CompetitionDistance
PromoInterval
CompetitionDistanceDecile
Type_Assort
PromoRun
DateIdentifier
P2cal
WeekNumber
Year
WeekOfYear
AvgSalesPerStore
StoreRank
NormalizedAvgSales_DayOfWeek
NormalizedAvgSales_Promo
NormalizedAvgSales_StateHoliday
NormalizedAvgSales_SchoolHoliday
NormalizedAvgSales_CompetitionDistance
NormalizedAvgSales_CompetitionDistanceDecile
NormalizedAvgSales_PromoInterval
NormalizedAvgSales_Type_Assort
NormalizedAvgSales_PromoRun
NormalizedAvgSales_P2cal
NormalizedAvgSales_Year
NormalizedAvgSales_WeekOfYear
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00



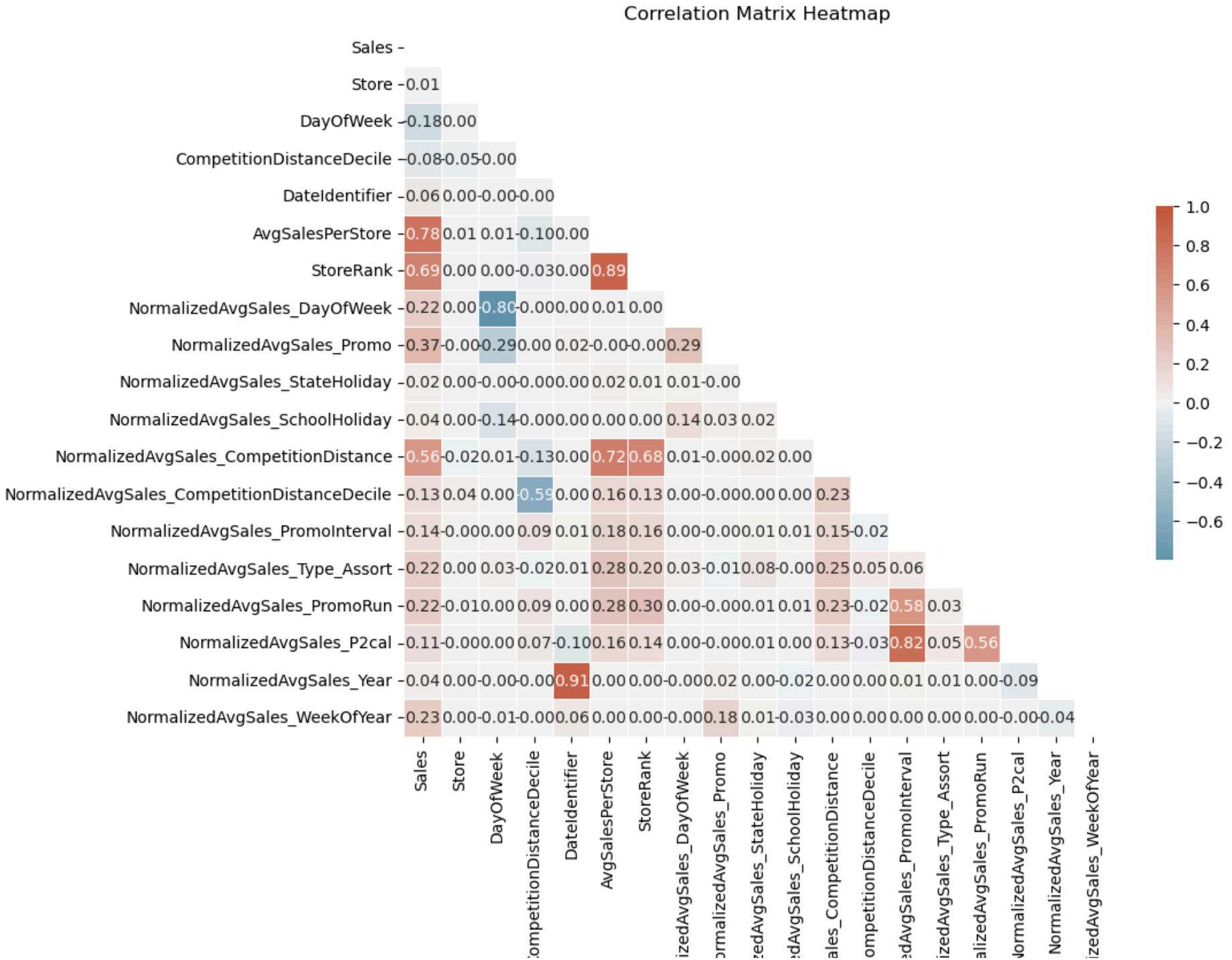
In [184…
```
# List of columns to drop
columns_to_drop = ['Promo', 'StateHoliday', 'SchoolHoliday', 'CompetitionDistance',
                   'PromoInterval', 'Type_Assort', 'PromoRun', 'P2cal',
                   'WeekNumber', 'Year', 'WeekOfYear']

# Drop the columns from the DataFrame
df.drop(columns=columns_to_drop, inplace=True)

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

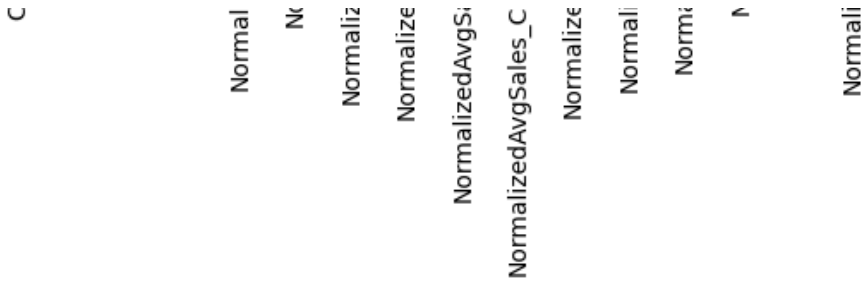
Out[184]:

	Sales	Store	DayOfWeek	CompetitionDistanceDecile	DateIdentifier	AvgSalesPerStore	StoreRank	NormalizedAvgSales_DayOfWeek	Nor
48	5263.0	1	5	4	942	4759.096031	155.0	1.203903	
49	5020.0	1	4	4	941	4759.096031	155.0	1.152020	
50	4782.0	1	3	4	940	4759.096031	155.0	1.145309	
51	5011.0	1	2	4	939	4759.096031	155.0	1.206520	
52	6102.0	1	1	4	938	4759.096031	155.0	1.398491	

In [185…
```
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, fmt=".2f")

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


Correlation Matrix Heatmap
Sales
Store
DayOfWeek
CompetitionDistanceDecile
DateIdentifier
AvgSalesPerStore
StoreRank
NormalizedAvgSales_DayOfWeek
NormalizedAvgSales_Promo
NormalizedAvgSales_StateHoliday
NormalizedAvgSales_SchoolHoliday
NormalizedAvgSales_CompetitionDistance
NormalizedAvgSales_CompetitionDistanceDecile
NormalizedAvgSales_PromoInterval
NormalizedAvgSales_Type_Assort
NormalizedAvgSales_PromoRun
NormalizedAvgSales_P2cal
NormalizedAvgSales_Year
NormalizedAvgSales_WeekOfYear
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6



In [186…

```
# Filter DataFrame to include only rows with DateIdentifier between 1 and 943
dfzc881 = dfzc[(dfzc['DateIdentifier'] >= 1) & (dfzc['DateIdentifier'] <= 881)]

dfzc881.T
```

Out[186]:

	110	111	112	113	114	117	118	119	120	121	...	1058285	1058286	1058287
Sales	5592.0	4656.0	4111.0	4083.0	4211.0	4276.0	4459.0	3755.0	4735.0	5235.0	...	2957.0	4924.0	4682.0
Store	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	...	1110.0	1110.0	1110.0
DayOfWeek	6.0	5.0	4.0	3.0	2.0	6.0	5.0	4.0	3.0	2.0	...	6.0	5.0	4.0
Promo	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	...	0.0	1.0	1.0
StateHoliday	3.0	3.0	3.0	3.0	3.0	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	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	1270.0	1270.0	...	900.0	900.0	900.0
PromoInterval	0.0	0.0	0.0	0.0	0.0	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	4.0	4.0	...	3.0	3.0	3.0
Type_Assort	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	5.0	...	6.0	6.0	6.0
PromoRun	0.0	0.0	0.0	0.0	0.0	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	870.0	869.0	...	12.0	11.0	10.0
P2cal	0.0	0.0	0.0	0.0	0.0	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	125.0	125.0	...	2.0	2.0	2.0
Year	2015.0	2015.0	2015.0	2015.0	2015.0	2015.0	2015.0	2015.0	2015.0	2015.0	...	2013.0	2013.0	2013.0
WeekOfYear	22.0	22.0	22.0	22.0	22.0	21.0	21.0	21.0	21.0	21.0	...	2.0	2.0	2.0

16 rows × 785727 columns

In [187…

```
dfzc881.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 785727 entries, 110 to 1058295
Data columns (total 16 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   Promo                      785727 non-null  int64
 4   StateHoliday               785727 non-null  int64
 5   SchoolHoliday              785727 non-null  int64
 6   CompetitionDistance        785727 non-null  float64
 7   PromoInterval              785727 non-null  float64
 8   CompetitionDistanceDecile  785727 non-null  int64
 9   Type_Assort                785727 non-null  int64
 10  PromoRun                   785727 non-null  float64
 11  DateIdentifier             785727 non-null  int64
 12  P2cal                      785727 non-null  int64
 13  WeekNumber                 785727 non-null  int64
 14  Year                       785727 non-null  int64
 15  WeekOfYear                 785727 non-null  int64
dtypes: float64(4), int64(12)
memory usage: 118.0 MB
```

In [188…

```
# Filter DataFrame to include only rows with DateIdentifier between 882 and 942
dfzc61 = dfzc[(dfzc['DateIdentifier'] >= 882) & (dfzc['DateIdentifier'] <= 942)]

dfzc61
```

Out[188]:

	Sales	Store	DayOfWeek	Promo	StateHoliday	SchoolHoliday	CompetitionDistance	PromoInterval	CompetitionDistanceDecile	Ty
48	5263.0	1	5	1	3	1	1270.0	0.0	4	
49	5020.0	1	4	1	3	1	1270.0	0.0	4	
50	4782.0	1	3	1	3	1	1270.0	0.0	4	
51	5011.0	1	2	1	3	1	1270.0	0.0	4	
52	6102.0	1	1	1	3	1	1270.0	0.0	4	
...	...	...	...	...	...	...	...	...	...	
1057411	5433.0	1110	5	1	3	0	900.0	0.0	3	
1057412	5578.0	1110	4	1	3	0	900.0	0.0	3	
1057413	5794.0	1110	3	1	3	0	900.0	0.0	3	
1057414	6906.0	1110	2	1	3	0	900.0	0.0	3	
1057415	8293.0	1110	1	1	3	0	900.0	0.0	3	

58611 rows × 16 columns

In [189…

```
dfzc61.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 58611 entries, 48 to 1057415
Data columns (total 16 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   Promo                      58611 non-null  int64
 4   StateHoliday               58611 non-null  int64
 5   SchoolHoliday              58611 non-null  int64
 6   CompetitionDistance        58611 non-null  float64
 7   PromoInterval              58611 non-null  float64
 8   CompetitionDistanceDecile  58611 non-null  int64
 9   Type_Assort                58611 non-null  int64
 10  PromoRun                   58611 non-null  float64
 11  DateIdentifier             58611 non-null  int64
 12  P2cal                      58611 non-null  int64
 13  WeekNumber                 58611 non-null  int64
 14  Year                       58611 non-null  int64
 15  WeekOfYear                 58611 non-null  int64
dtypes: float64(4), int64(12)
memory usage: 7.6 MB
```

In [190…

```
dfzc61.head(7).T
```

Out[190]:

	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
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 [191…

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# 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 = model.feature_importances_

# Create a DataFrame for feature importances
features_df = pd.DataFrame({
    'Feature': X.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)
```

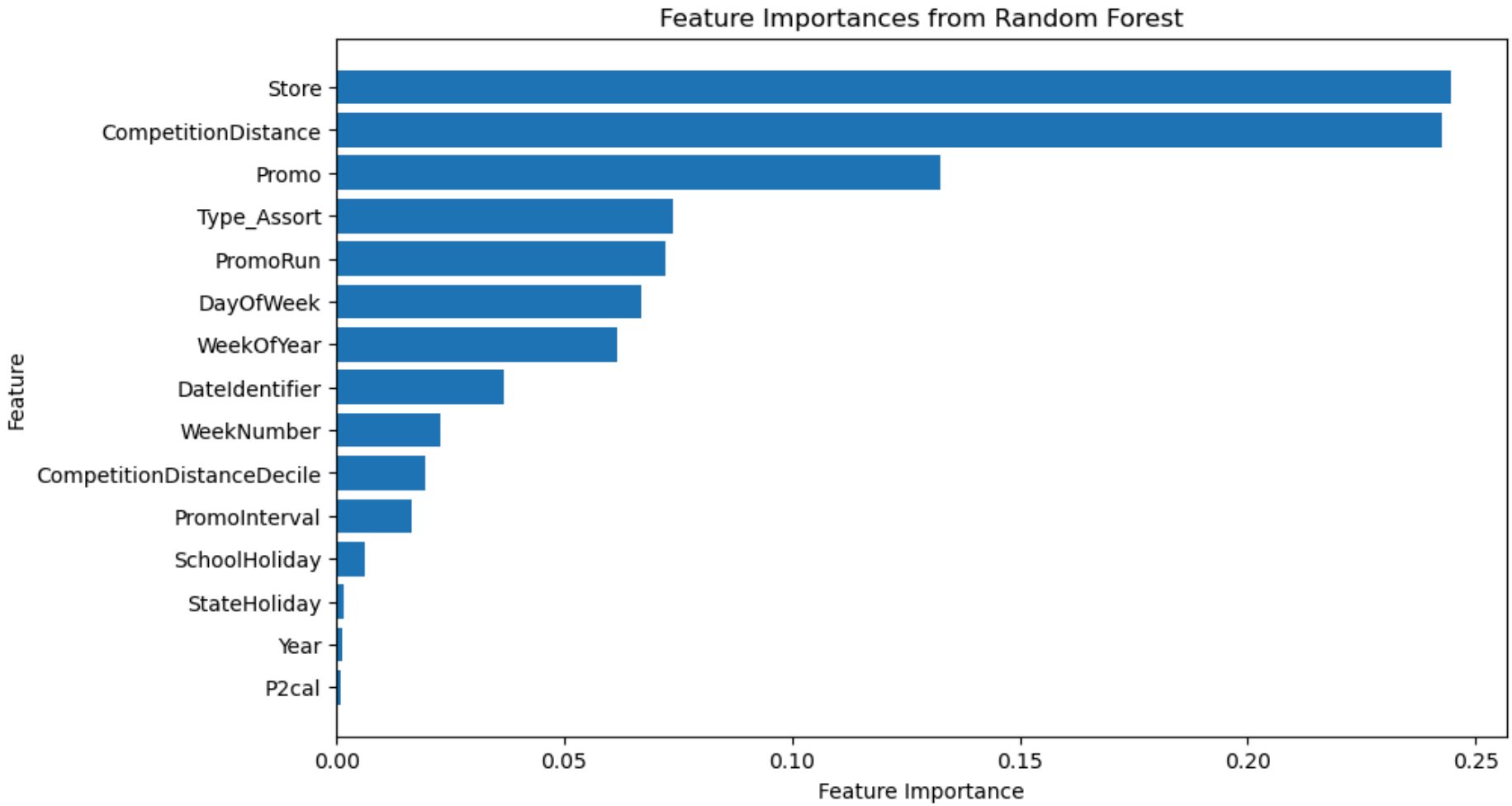
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[191], line 27
     24 r2 = r2_score(y_test, y_pred)
     26 # Get feature importances
---> 27 feature_importances = model.feature_importances_
     29 # Create a DataFrame for feature importances
     30 features_df = pd.DataFrame({
     31     'Feature': X.columns,
     32     'Importance': feature_importances
     33 })

File ~/anaconda3/lib/python3.11/site-packages/statsmodels/base/wrapper.py:34, in ResultsWrapper.__getattribute__(self, attr)
     31 except AttributeError:
     32     pass
---> 34 obj = getattr(results, attr)
     35 data = results.model.data
     36 how = self._wrap_attrs.get(attr)

AttributeError: 'OLSResults' object has no attribute 'feature_importances_'
```

In [192…

```
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: 1610925.020355024
R-squared: 0.8335689188011371
Root Mean Squared Percentage Error (RMSPE): 0.18243901620906622

Feature Importances:
                       Feature  Importance
0                        Store    0.244749
5          CompetitionDistance    0.242546
2                        Promo    0.132525
8                  Type_Assort    0.073846
9                     PromoRun    0.072089
1                    DayOfWeek    0.066959
14                  WeekOfYear    0.061447
10              DateIdentifier    0.036811
12                  WeekNumber    0.022727
7   CompetitionDistanceDecile    0.019496
6                PromoInterval    0.016612
4                SchoolHoliday    0.006210
3                 StateHoliday    0.001661
13                        Year    0.001386
11                       P2cal    0.000935
```



In [ ]:
```
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['Predicted_Sales']

# 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'] - daily_sales_comparison['Predicted_Sales']

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['Sales'], label='Actual Sales', marker='o')
plt.plot(daily_sales_comparison['DateIdentifier'], daily_sales_comparison['Predicted_Sales'], label='Projected Sales',
plt.fill_between(daily_sales_comparison['DateIdentifier'], daily_sales_comparison['Sales'], daily_sales_comparison['Pr
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Daily Sales: Actual vs Projected')
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()
```

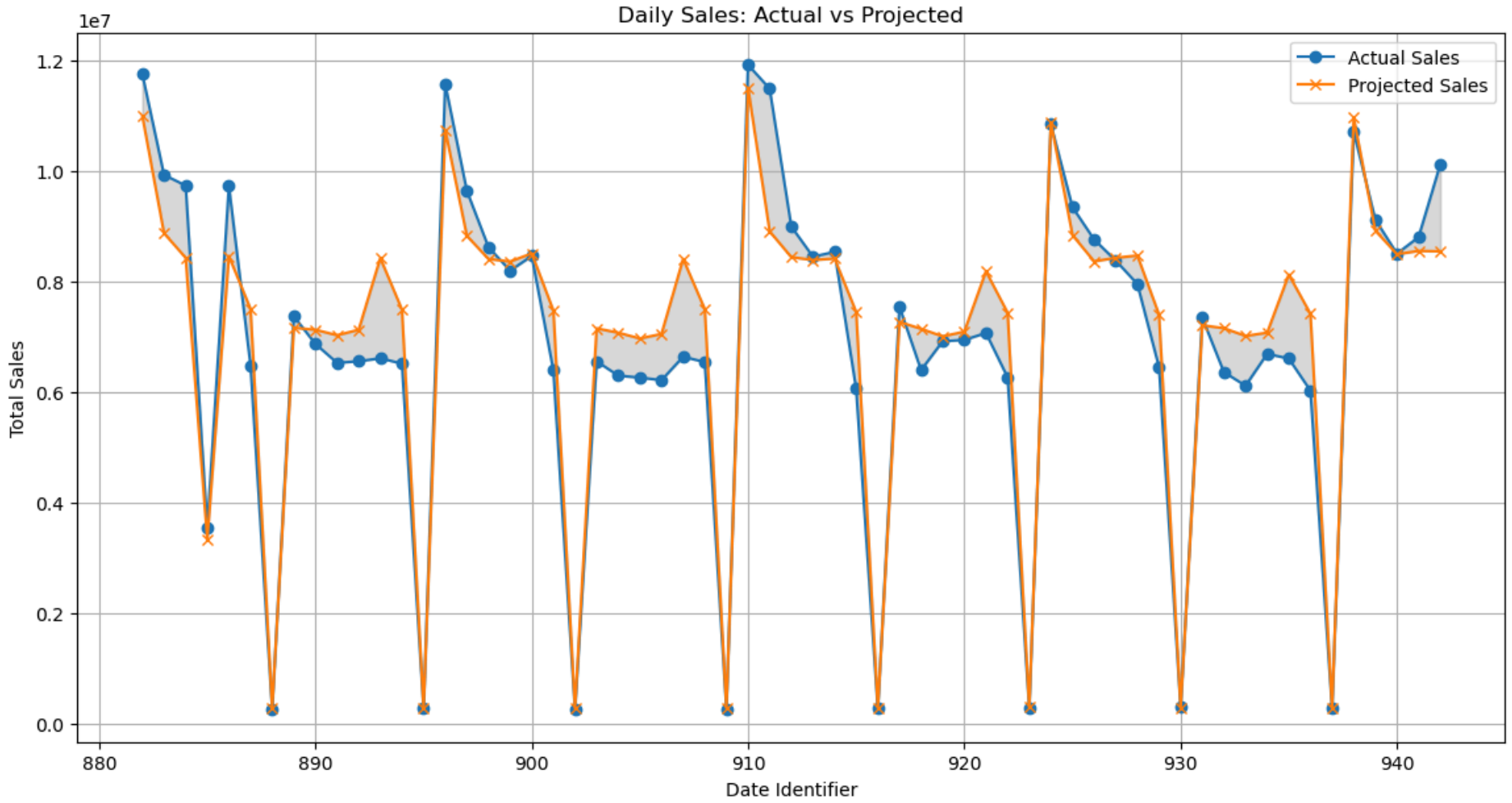
In [193…

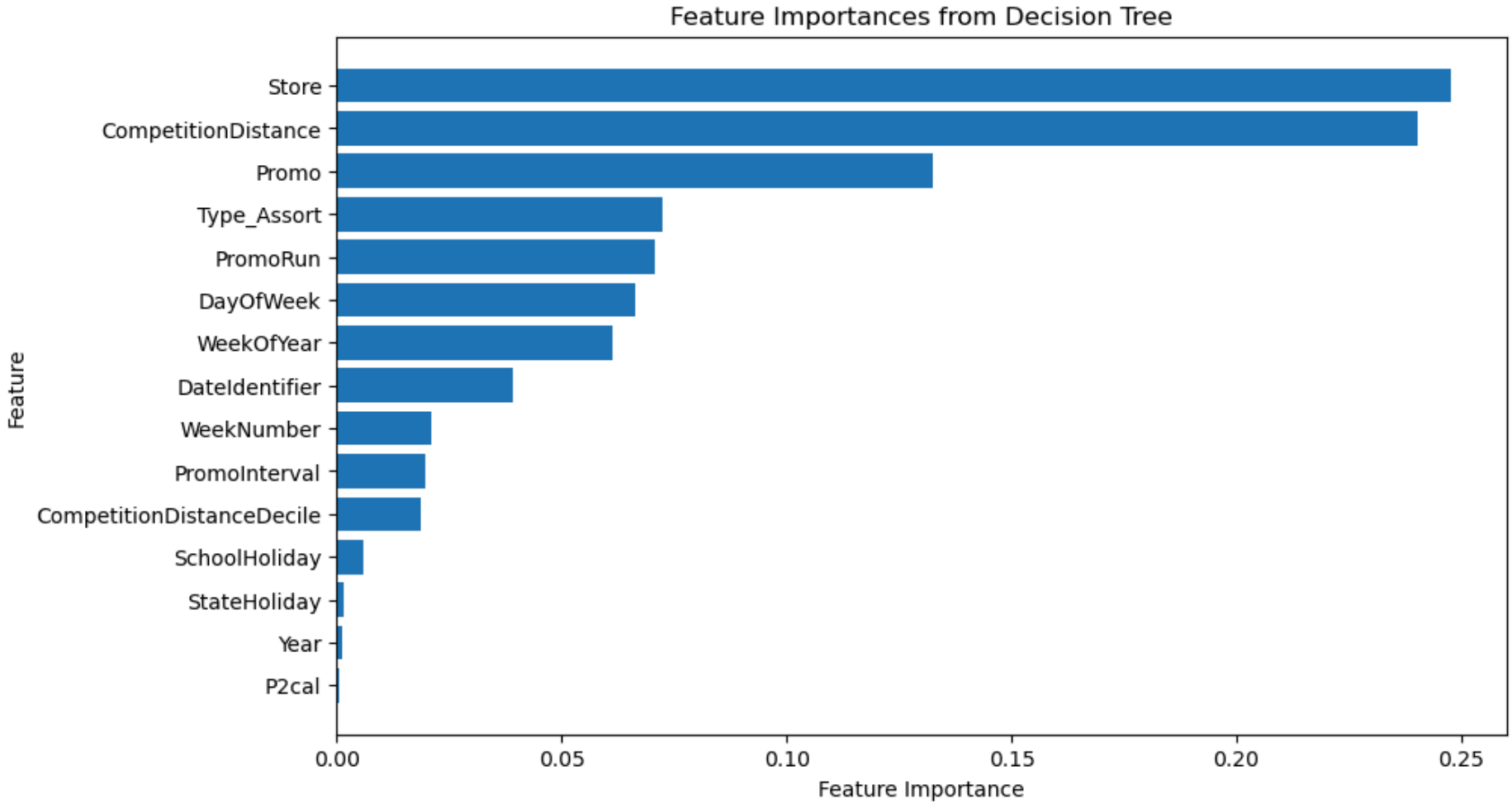
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# 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['Predicted_Sales']

# 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'] - daily_sales_comparison['Predicted_Sales']

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['Sales'], label='Actual Sales', marker='o')
plt.plot(daily_sales_comparison['DateIdentifier'], daily_sales_comparison['Predicted_Sales'], label='Projected Sales',
plt.fill_between(daily_sales_comparison['DateIdentifier'], daily_sales_comparison['Sales'], daily_sales_comparison['Pr
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Daily Sales: Actual vs Projected')
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()
```

```
Mean Squared Error: 2388937.0145194586
R-squared: 0.7531894003640012
Root Mean Squared Percentage Error (RMSPE): 0.21663414641125567

Feature Importances:
                      Feature  Importance
0                       Store    0.247759
5         CompetitionDistance    0.240252
2                       Promo    0.132465
8                 Type_Assort    0.072505
9                    PromoRun    0.070840
1                   DayOfWeek    0.066532
14                 WeekOfYear    0.061233
10             DateIdentifier    0.039302
12                 WeekNumber    0.021175
6                PromoInterval    0.019780
7   CompetitionDistanceDecile    0.018541
4                SchoolHoliday    0.006058
3                StateHoliday    0.001675
13                       Year    0.001173
11                      P2cal    0.000711
    DateIdentifier  Store   Sales  Predicted_Sales  Difference
48             942      1  5263.0           6574.0     -1311.0
49             941      1  5020.0           6574.0     -1554.0
50             940      1  4782.0           6816.0     -2034.0
51             939      1  5011.0           6816.0     -1805.0
52             938      1  6102.0           6714.0      -612.0

Daily Sales Comparison (Actual vs Projected):
   DateIdentifier       Sales  Predicted_Sales  Difference
0             882  11743615.0       10986567.0    757048.0
1             883   9925193.0        8869526.0   1055667.0
2             884   9731791.0        8426048.0   1305743.0
3             885   3534231.0        3329221.0    205010.0
4             886   9724893.0        8455815.0   1269078.0
```


Daily Sales: Actual vs Projected
1e7
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Total Sales
880
890
900
910
920
930
940
Date Identifier
Actual Sales
Projected Sales



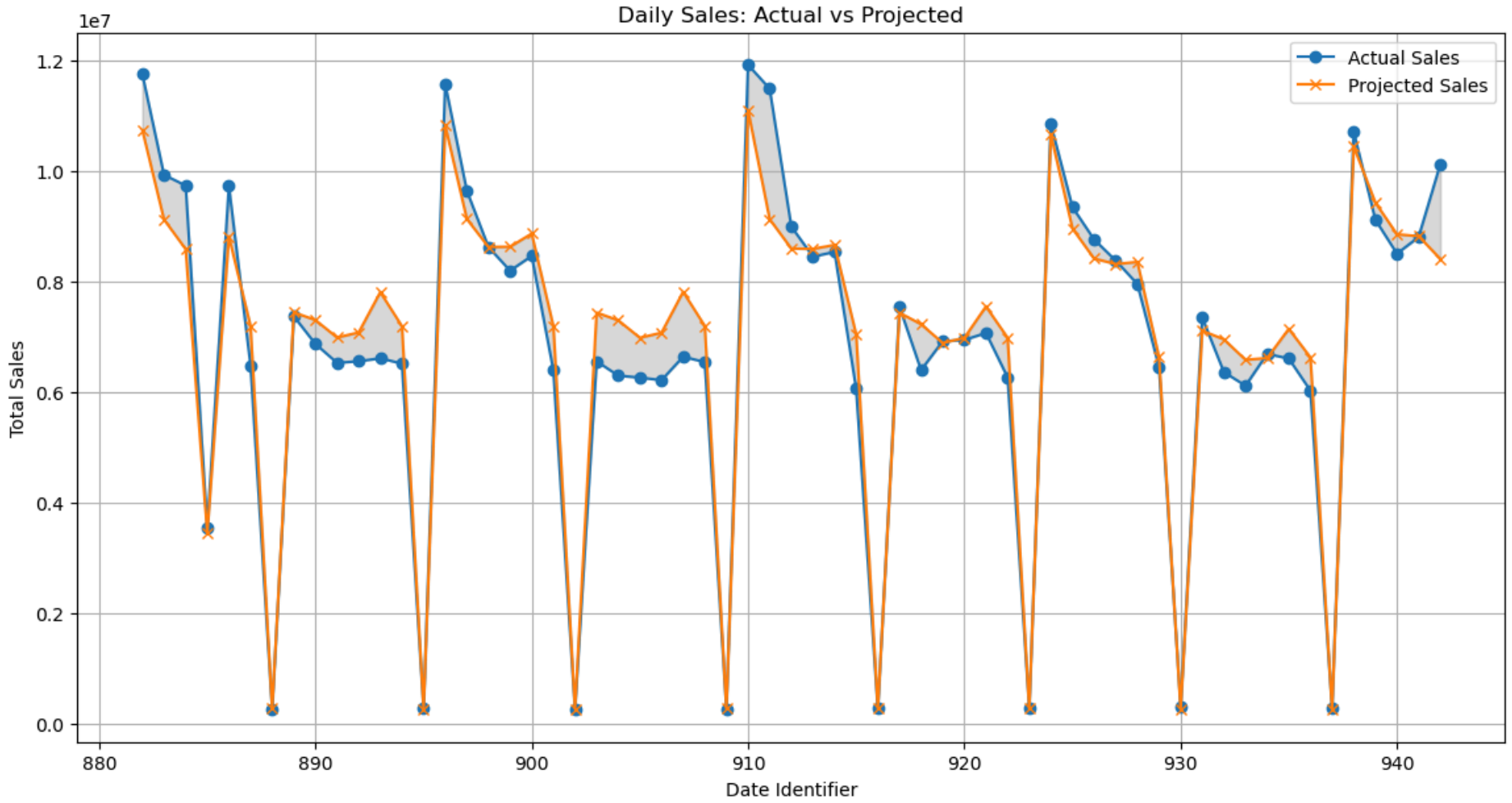
In [204…

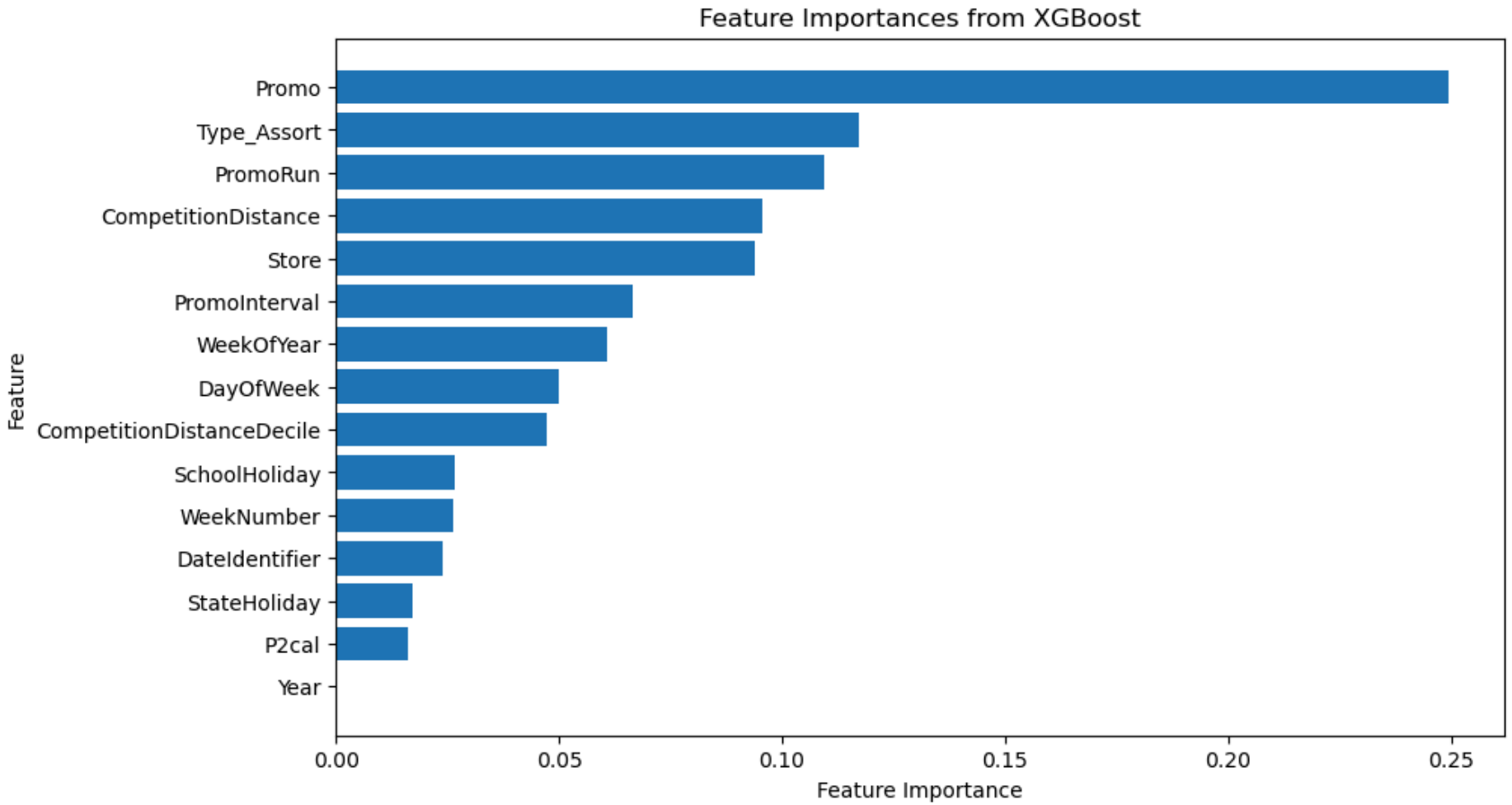
```
import xgboost as xgb
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# 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['Predicted_Sales']

# 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'] - daily_sales_comparison['Predicted_Sales']

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['Sales'], label='Actual Sales', marker='o')
plt.plot(daily_sales_comparison['DateIdentifier'], daily_sales_comparison['Predicted_Sales'], label='Projected Sales',
plt.fill_between(daily_sales_comparison['DateIdentifier'], daily_sales_comparison['Sales'], daily_sales_comparison['Pr
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Daily Sales: Actual vs Projected')
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()
```

```
Mean Squared Error: 1867601.1575184863
R-squared: 0.8070506845653512
Root Mean Squared Percentage Error (RMSPE): 0.22730756268742128

Feature Importances:
                      Feature  Importance
2                       Promo    0.249544
8                  Type_Assort    0.117056
9                     PromoRun    0.109469
5         CompetitionDistance    0.095420
0                       Store    0.094050
6                PromoInterval    0.066532
14                  WeekOfYear    0.060700
1                    DayOfWeek    0.049848
7   CompetitionDistanceDecile    0.047268
4                SchoolHoliday    0.026639
12                  WeekNumber    0.026361
10              DateIdentifier    0.024025
3                 StateHoliday    0.017071
11                       P2cal    0.016017
13                        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
```


Daily Sales: Actual vs Projected
1e7
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Total Sales
880
890
900
910
920
930
940
Date Identifier
Actual Sales
Projected Sales



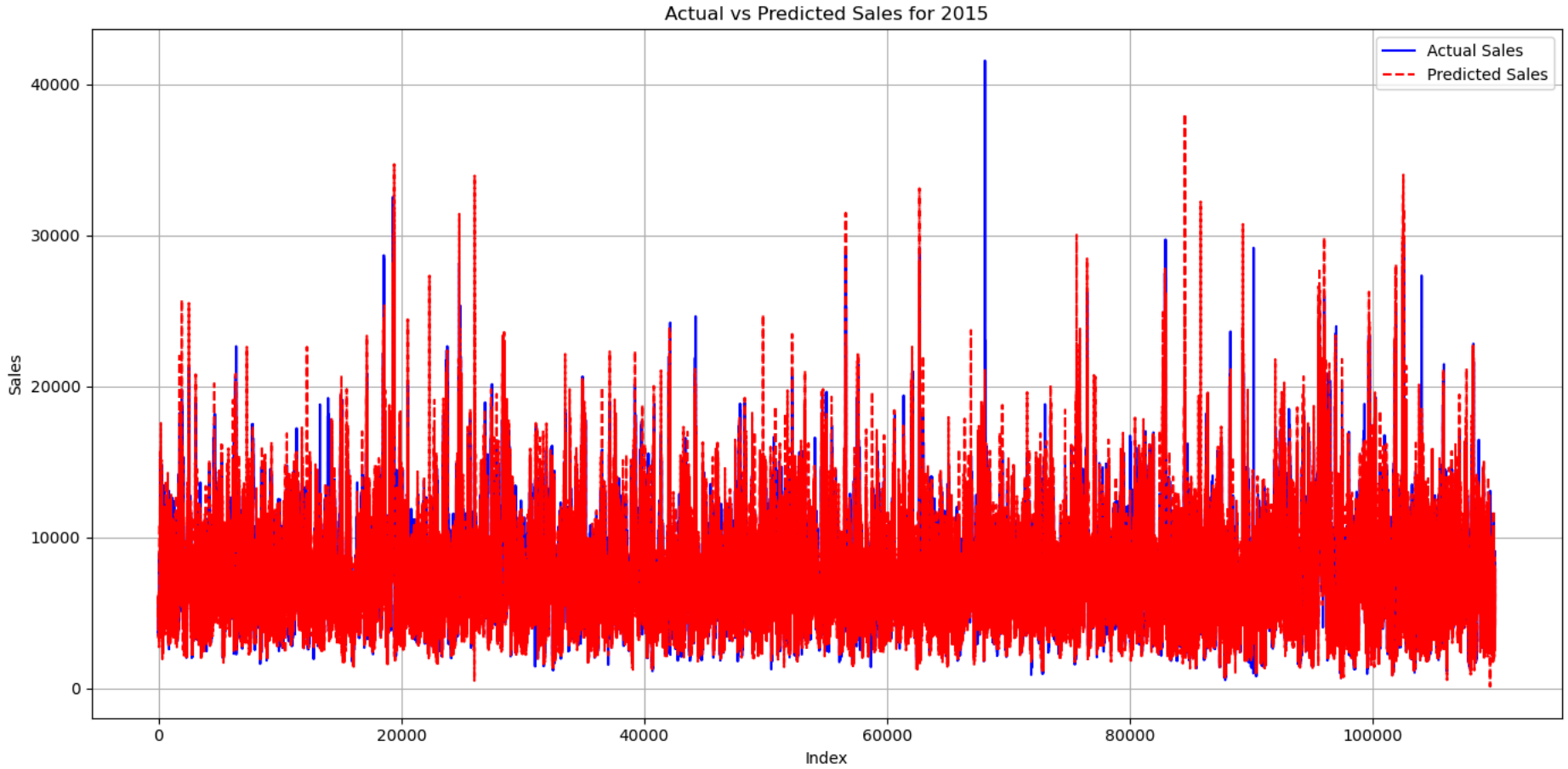
In [194…

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

# Assuming dfzc881 contains data for 2013 and dfzc61 contains data for 2015

# Filter data for 2013 and 2015
dfzc881['Year'] = 2013
dfzc61['Year'] = 2015

# Extract relevant columns
```

```
df_2013 = dfzc881[['Store', 'DayOfWeek', 'Promo', 'StateHoliday', 'SchoolHoliday', 'CompetitionDistance', 'PromoInterv
df_2015 = dfzc61[['Store', 'DayOfWeek', 'Promo', 'StateHoliday', 'SchoolHoliday', 'CompetitionDistance', 'PromoInterva

# Merge 2013 and 2015 dataframes on 'Store', 'WeekOfYear', and 'DayOfWeek'
merged_df = pd.merge(df_2015, df_2013, on=['Store', 'WeekOfYear', 'DayOfWeek'], suffixes=('_2015', '_2013'))

# Use Sales from 2013 as predictions for 2015
merged_df['Predicted_Sales'] = merged_df['Sales_2013']

# Actual sales for 2015
actual_sales = merged_df['Sales_2015']
predicted_sales = merged_df['Predicted_Sales']

# Calculate evaluation metrics
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) ** 2))

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

# Show the first few rows of actual vs predicted sales
output_df = merged_df[['Store', 'DayOfWeek', 'WeekOfYear', 'Sales_2015', 'Predicted_Sales']]
output_df['Difference'] = output_df['Sales_2015'] - output_df['Predicted_Sales']
print(output_df.head())

# Plotting the actual vs predicted sales for each day
plt.figure(figsize=(14, 7))
plt.plot(merged_df.index, actual_sales, label='Actual Sales', color='blue')
plt.plot(merged_df.index, predicted_sales, label='Predicted Sales', color='red', linestyle='--')
plt.xlabel('Index')
plt.ylabel('Sales')
plt.title('Actual vs Predicted Sales for 2015')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Mean Squared Error: 1932466.7040617901
R-squared: 0.8037989456493857
Root Mean Squared Percentage Error (RMSPE): 0.18132074020755526
```

	Store	DayOfWeek	WeekOfYear	Sales_2015	Predicted_Sales	Difference
0	1	5	31	5263.0	5038.0	225.0
1	1	5	31	5263.0	4494.0	769.0
2	1	4	31	5020.0	5106.0	-86.0
3	1	4	31	5020.0	4994.0	26.0
4	1	3	31	4782.0	5487.0	-705.0



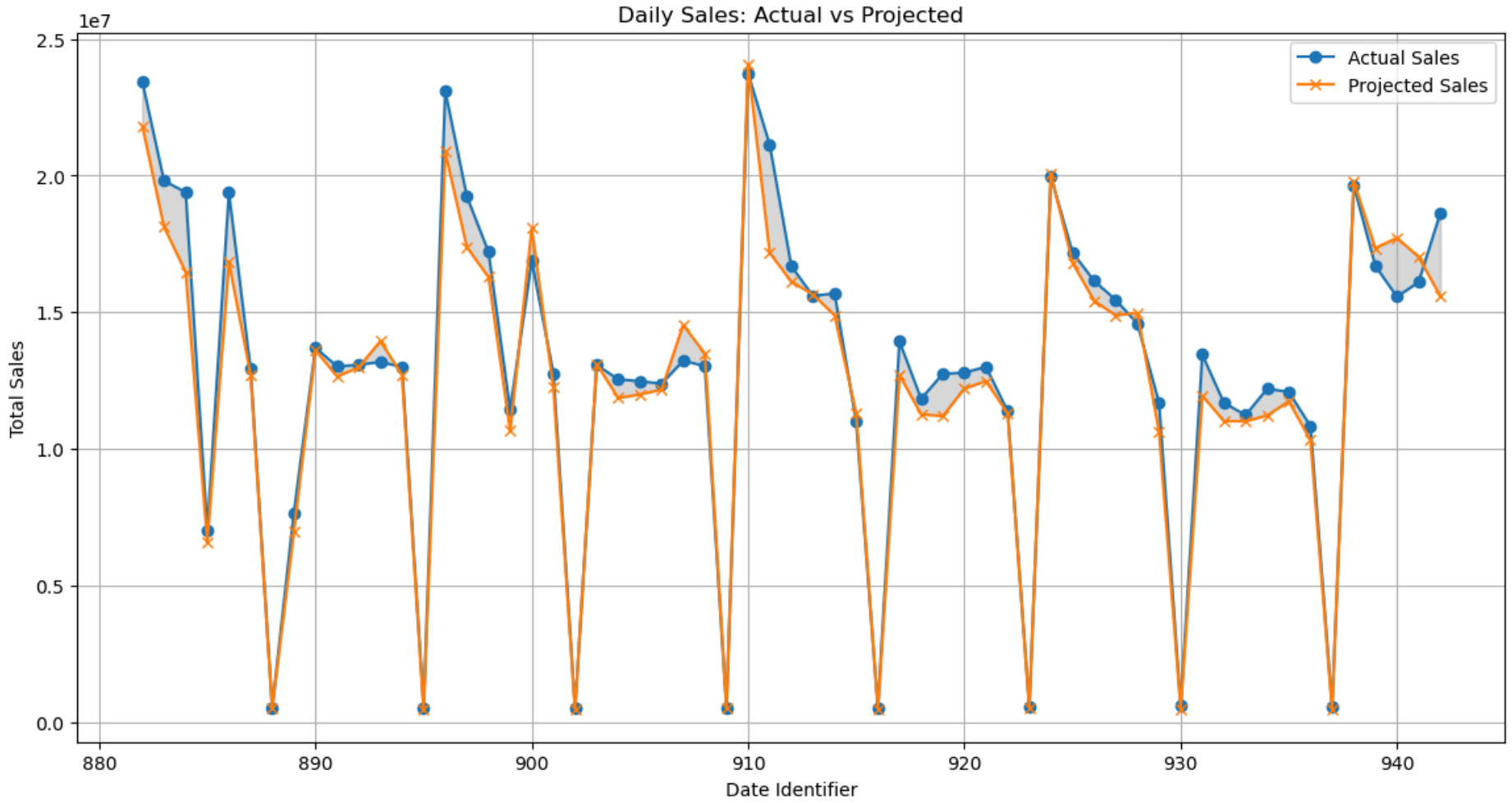
In [205…

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

# Assuming dfzc881 contains data for 2013 and dfzc61 contains data for 2015
```

```
# Merge the dataframes on 'Store', 'DayOfWeek', 'WeekOfYear'
merged_df = pd.merge(dfzc61, dfzc881, on=['Store', 'DayOfWeek', 'WeekOfYear'], suffixes=('_2015', '_2013'))

# Use Sales from 2013 as predictions for 2015
merged_df['Predicted_Sales'] = merged_df['Sales_2013']

# Create DataFrame to compare actual and predicted sales
comparison_df = merged_df[['DateIdentifier_2015', 'Store', 'Sales_2015']].copy()
comparison_df['Predicted_Sales'] = merged_df['Predicted_Sales']
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['Predicted_Sales']

# Calculate evaluation metrics
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) ** 2))

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'] - daily_sales_comparison['Predicted_Sales

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_comparison['Sales_2015'], label='Actual Sales', ma
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Predicted_Sales'], label='Projected Sa
plt.fill_between(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Sales_2015'], daily_sales_comp
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Daily Sales: Actual vs Projected')
plt.legend()
```

```
plt.grid(True)
plt.show()
```

```
Mean Squared Error: 1950508.6708606547
R-squared: 0.8019671661413322
Root Mean Squared Percentage Error (RMSPE): 0.19058677328826445
Comparison of Actual and Predicted Sales:
   DateIdentifier_2015  Store  Sales_2015  Predicted_Sales  Difference
0                  942      1      5263.0           5038.0       225.0
1                  942      1      5263.0           4494.0       769.0
2                  941      1      5020.0           5106.0       -86.0
3                  941      1      5020.0           4994.0        26.0
4                  940      1      4782.0           5487.0      -705.0

Daily Sales Comparison (Actual vs Projected):
   DateIdentifier_2015  Sales_2015  Predicted_Sales  Difference
0                  882  23427426.0       21807677.0   1619749.0
1                  883  19807797.0       18125353.0   1682444.0
2                  884  19396356.0       16475378.0   2920978.0
3                  885   7018299.0        6593673.0    424626.0
4                  886  19382771.0       16824829.0   2557942.0
```

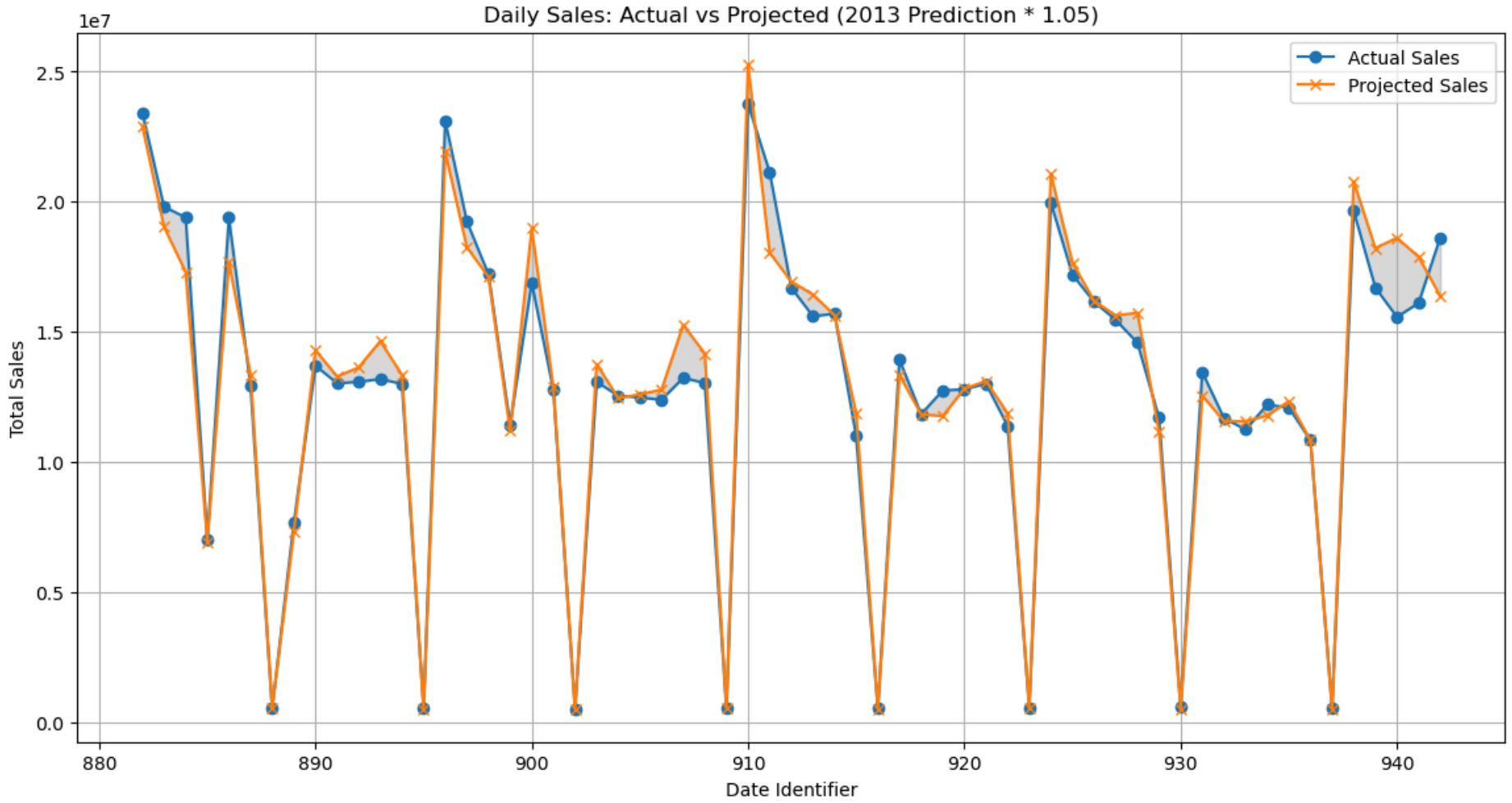


In [195…

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

# Assuming dfzc881 contains data for 2013 and dfzc61 contains data for 2015

# Merge the dataframes on 'Store', 'DayOfWeek', 'WeekOfYear'
merged_df = pd.merge(dfzc61, dfzc881, on=['Store', 'DayOfWeek', 'WeekOfYear'], suffixes=('_2015', '_2013'))

# Use Sales from 2013 multiplied by 1.05 as predictions for 2015
merged_df['Predicted_Sales'] = merged_df['Sales_2013'] * 1.05
```

```
# Create DataFrame to compare actual and predicted sales
comparison_df = merged_df[['DateIdentifier_2015', 'Store', 'Sales_2015']].copy()
comparison_df['Predicted_Sales'] = merged_df['Predicted_Sales']
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['Predicted_Sales']

# 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'] - daily_sales_comparison['Predicted_Sales

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_comparison['Sales_2015'], label='Actual Sales', ma
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Predicted_Sales'], label='Projected Sa
plt.fill_between(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Sales_2015'], daily_sales_comp
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Daily Sales: Actual vs Projected (2013 Prediction * 1.05)')
plt.legend()
plt.grid(True)
plt.show()
```

```
Comparison of Actual and Predicted Sales:
   DateIdentifier_2015  Store  Sales_2015  Predicted_Sales  Difference
0                  942      1      5263.0          5289.90      -26.90
1                  942      1      5263.0          4718.70      544.30
2                  941      1      5020.0          5361.30     -341.30
3                  941      1      5020.0          5243.70     -223.70
4                  940      1      4782.0          5761.35     -979.35

Daily Sales Comparison (Actual vs Projected):
   DateIdentifier_2015  Sales_2015  Predicted_Sales  Difference
0                  882  23427426.0      22898060.85   529365.15
1                  883  19807797.0      19031620.65   776176.35
2                  884  19396356.0      17299146.90  2097209.10
3                  885   7018299.0       6923356.65    94942.35
4                  886  19382771.0      17666070.45  1716700.55
```

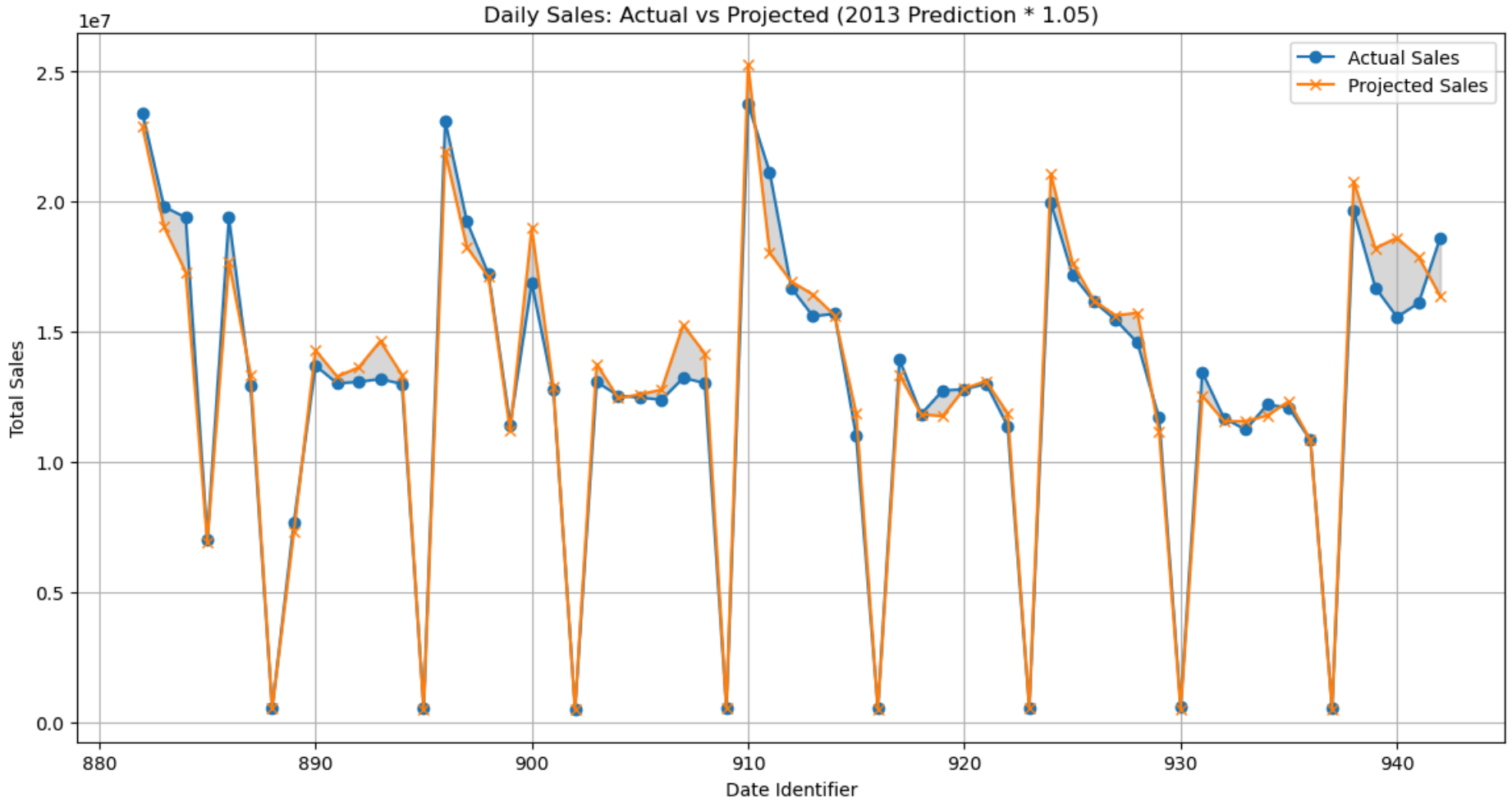


In [206…

```
# Merge the dataframes on 'Store', 'DayOfWeek', 'WeekOfYear'
merged_df = pd.merge(dfzc61, dfzc881, on=['Store', 'DayOfWeek', 'WeekOfYear'], suffixes=('_2015', '_2013'))

# Use Sales from 2013 multiplied by 1.05 as predictions for 2015
merged_df['Predicted_Sales'] = merged_df['Sales_2013'] * 1.05

# Create DataFrame to compare actual and predicted sales
comparison_df = merged_df[['DateIdentifier_2015', 'Store', 'Sales_2015']].copy()
comparison_df['Predicted_Sales'] = merged_df['Predicted_Sales']
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['Predicted_Sales']

# Calculate evaluation metrics
```

```
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) ** 2))

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'] - daily_sales_comparison['Predicted_Sales

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_comparison['Sales_2015'], label='Actual Sales', ma
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Predicted_Sales'], label='Projected Sa
plt.fill_between(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Sales_2015'], daily_sales_comp
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Daily Sales: Actual vs Projected (2013 Prediction * 1.05)')
plt.legend()
plt.grid(True)
plt.show()
```

```
Mean Squared Error: 1950508.6708606547
R-squared: 0.8019671661413322
Root Mean Squared Percentage Error (RMSPE): 0.19058677328826445
Comparison of Actual and Predicted Sales:
   DateIdentifier_2015  Store  Sales_2015  Predicted_Sales  Difference
0                  942      1      5263.0          5289.90      -26.90
1                  942      1      5263.0          4718.70      544.30
2                  941      1      5020.0          5361.30     -341.30
3                  941      1      5020.0          5243.70     -223.70
4                  940      1      4782.0          5761.35     -979.35

Daily Sales Comparison (Actual vs Projected):
   DateIdentifier_2015  Sales_2015  Predicted_Sales  Difference
0                  882  23427426.0      22898060.85   529365.15
1                  883  19807797.0      19031620.65   776176.35
2                  884  19396356.0      17299146.90  2097209.10
3                  885   7018299.0       6923356.65    94942.35
4                  886  19382771.0      17666070.45  1716700.55
```

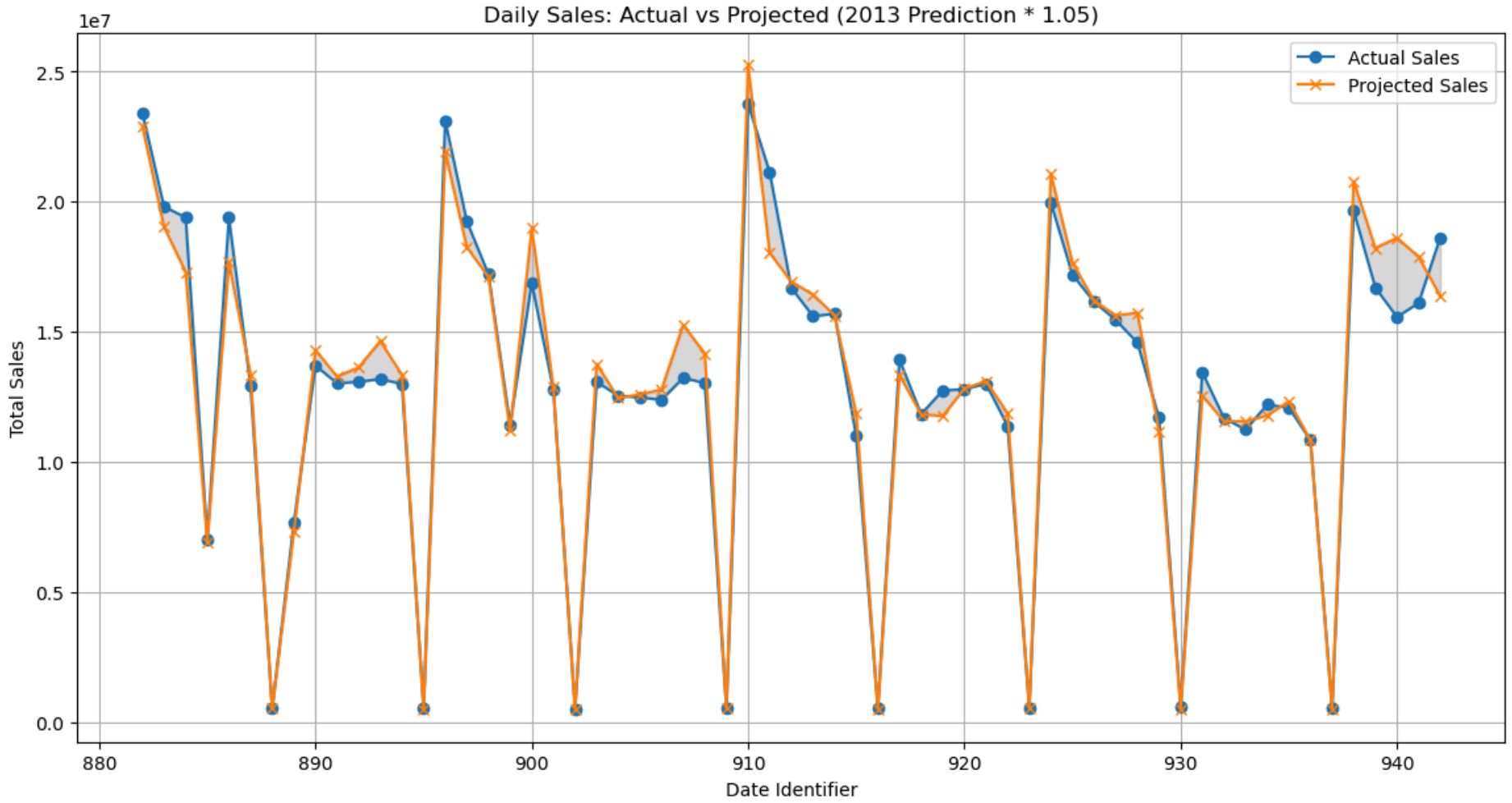


In [196…

```
# Create comparison DataFrame
comparison_df = merged_df[['DateIdentifier_2015', 'Store', 'Sales_2015', 'Predicted_Sales']].copy()
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['Predicted_Sales']

# 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) ** 2))

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'] - daily_sales_comparison['Predicted_Sales

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_comparison['Sales_2015'], label='Actual Sales', ma
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Predicted_Sales'], label='Projected Sa
plt.fill_between(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Sales_2015'], daily_sales_comp
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Daily Sales: Actual vs Projected (2013 Prediction * 1.05)')
plt.legend()
plt.grid(True)
plt.show()
```

```
Mean Squared Error: 1950508.6708606547
R-squared: 0.8019671661413322
Root Mean Squared Percentage Error (RMSPE): 0.19058677328826445
Comparison of Actual and Predicted Sales:
   DateIdentifier_2015  Store  Sales_2015  Predicted_Sales  Difference
0                  942      1      5263.0          5289.90      -26.90
1                  942      1      5263.0          4718.70      544.30
2                  941      1      5020.0          5361.30     -341.30
3                  941      1      5020.0          5243.70     -223.70
4                  940      1      4782.0          5761.35     -979.35

Daily Sales Comparison (Actual vs Projected):
   DateIdentifier_2015  Sales_2015  Predicted_Sales  Difference
0                  882  23427426.0      22898060.85   529365.15
1                  883  19807797.0      19031620.65   776176.35
2                  884  19396356.0      17299146.90  2097209.10
3                  885   7018299.0       6923356.65    94942.35
4                  886  19382771.0      17666070.45  1716700.55
```

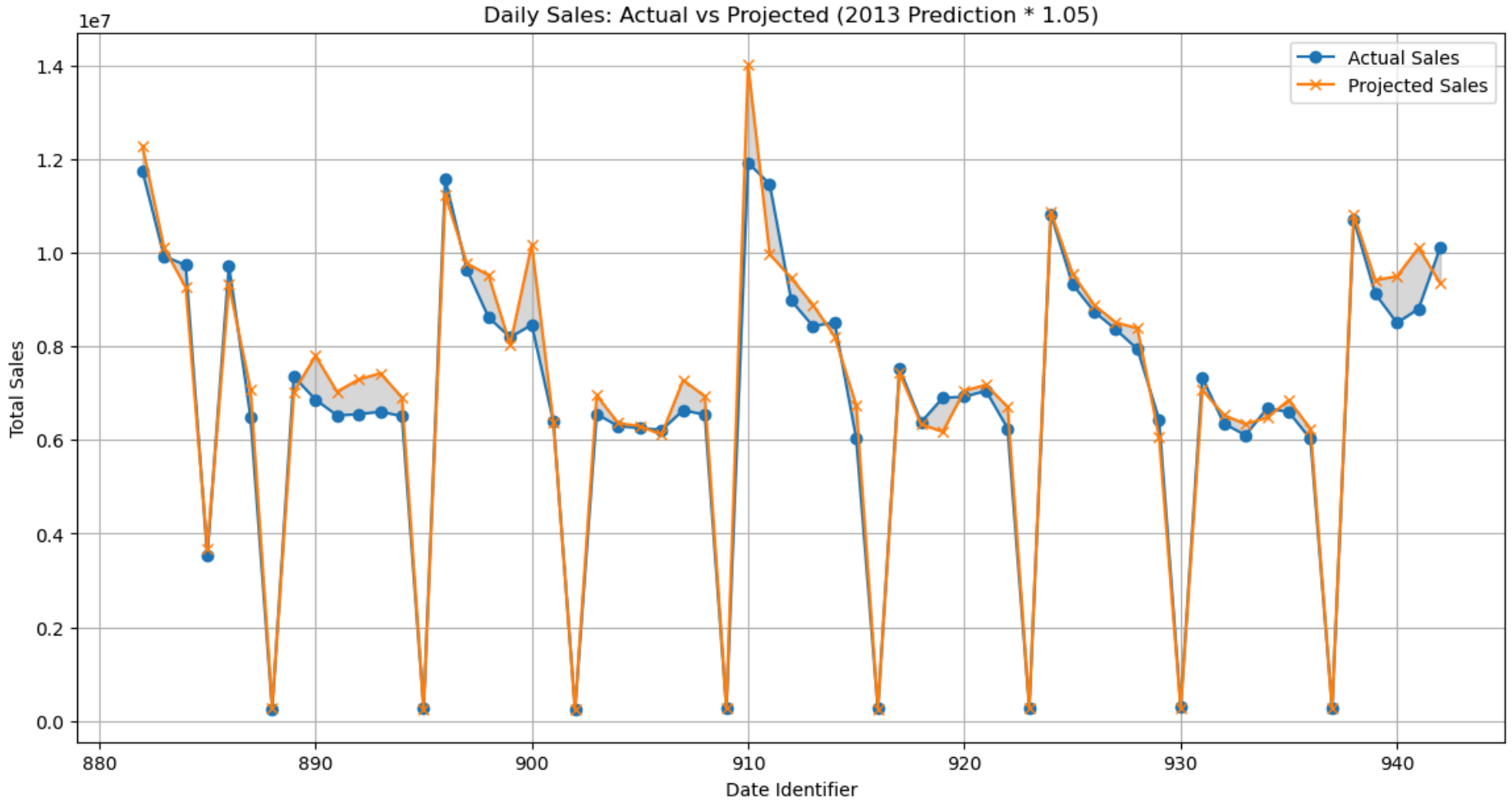


In [207…

```
# Aggregate data before creating comparison DataFrame
agg_df = merged_df.groupby(['DateIdentifier_2015', 'Store']).agg({
    'Sales_2015': 'mean',  # Calculate average sales for each store on each date
    'Predicted_Sales': 'first'  # Take the first value of Predicted_Sales (assumed to be the same for each store)
}).reset_index()

# Create comparison DataFrame
comparison_df = agg_df[['DateIdentifier_2015', 'Store', 'Sales_2015']].copy()
comparison_df['Predicted_Sales'] = agg_df['Predicted_Sales']
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['Predicted_Sales']

# Calculate evaluation metrics
```

```
actual_sales = agg_df['Sales_2015']
predicted_sales = agg_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) ** 2))

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'] - daily_sales_comparison['Predicted_Sales

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_comparison['Sales_2015'], label='Actual Sales', ma
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Predicted_Sales'], label='Projected Sa
plt.fill_between(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Sales_2015'], daily_sales_comp
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Daily Sales: Actual vs Projected (2013 Prediction * 1.05)')
plt.legend()
plt.grid(True)
plt.show()
```

```
Mean Squared Error: 1616712.7791536187
R-squared: 0.8330092863068538
Root Mean Squared Percentage Error (RMSPE): 0.1777053570701032
Comparison of Actual and Predicted Sales:
   DateIdentifier_2015  Store  Sales_2015  Predicted_Sales  Difference
0                  882      1      5774.0          6374.55     -600.55
1                  882      2      6629.0          7686.00    -1057.00
2                  882      3     11594.0         11457.60      136.40
3                  882      4     13727.0         13741.35      -14.35
4                  882      5      8396.0          8253.00      143.00

Daily Sales Comparison (Actual vs Projected):
   DateIdentifier_2015  Sales_2015  Predicted_Sales  Difference
0                  882  11743615.0      12274324.65  -530709.65
1                  883   9925193.0      10101890.40  -176697.40
2                  884   9731791.0       9257794.35   473996.65
3                  885   3534231.0       3680657.40  -146426.40
4                  886   9724893.0       9311124.90   413768.10
```



In [ ]:
```
# Aggregate data before creating comparison DataFrame
agg_df = merged_df.groupby(['DateIdentifier_2015', 'Store']).agg({
    'Sales_2015': 'mean',  # Calculate average sales for each store on each date
    'Predicted_Sales': 'first'  # Take the first value of Predicted_Sales (assumed to be the same for each store)
}).reset_index()

# Create comparison DataFrame
comparison_df = agg_df[['DateIdentifier_2015', 'Store', 'Sales_2015']].copy()
comparison_df['Predicted_Sales'] = agg_df['Predicted_Sales']
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['Predicted_Sales']

# Calculate evaluation metrics
```

```
actual_sales = agg_df['Sales_2015']
predicted_sales = agg_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) ** 2))

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'] - daily_sales_comparison['Predicted_Sales

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_comparison['Sales_2015'], label='Actual Sales', ma
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Predicted_Sales'], label='Projected Sa
plt.fill_between(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Sales_2015'], daily_sales_comp
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Daily Sales: Actual vs Projected (2013 Prediction')
plt.legend()
plt.grid(True)
plt.show()
```

In [214…

```
# Filter for stores 1111-1115 and DateIdentifier 214
filtered_sales = dfzc942[(dfzc942['Store'].between(1,2)) & (dfzc942['DateIdentifier'].between(211,214))]

# 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 [ ]:

In [198…
```
# Filter for stores 1111-1115 and DateIdentifier 214
filtered_sales = dfzc942[(dfzc942['Store'].between(1, 5)) & (dfzc942['DateIdentifier'] == 154)]

# 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 [199…
```
agg_df.tail()
```

Out[199]:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
1058292	1111	2	01/01/2013	0.0	0.0	0.0	0	a	1
1058293	1112	2	01/01/2013	0.0	0.0	0.0	0	a	1
1058294	1113	2	01/01/2013	0.0	0.0	0.0	0	a	1
1058295	1114	2	01/01/2013	0.0	0.0	0.0	0	a	1
1058296	1115	2	01/01/2013	0.0	0.0	0.0	0	a	1

In [200…
```
agg_df.head()
```

Out[200]:

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

In [201…

```
dfzc942.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 844338 entries, 48 to 1058295
Data columns (total 19 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   CompetitionDistanceDecile                 844338 non-null  int64  
 4   DateIdentifier                            844338 non-null  int64  
 5   AvgSalesPerStore                          844338 non-null  float64
 6   StoreRank                                 844338 non-null  float64
 7   NormalizedAvgSales_DayOfWeek              844338 non-null  float64
 8   NormalizedAvgSales_Promo                  844338 non-null  float64
 9   NormalizedAvgSales_StateHoliday           844338 non-null  float64
 10  NormalizedAvgSales_SchoolHoliday          844338 non-null  float64
 11  NormalizedAvgSales_CompetitionDistance    844338 non-null  float64
 12  NormalizedAvgSales_CompetitionDistanceDecile  844338 non-null  float64
 13  NormalizedAvgSales_PromoInterval          844338 non-null  float64
 14  NormalizedAvgSales_Type_Assort            844338 non-null  float64
 15  NormalizedAvgSales_PromoRun               844338 non-null  float64
 16  NormalizedAvgSales_P2cal                  844338 non-null  float64
 17  NormalizedAvgSales_Year                   844338 non-null  float64
 18  NormalizedAvgSales_WeekOfYear             844338 non-null  float64
dtypes: float64(15), int64(4)
memory usage: 128.8 MB
```

In [210…

```
import pandas as pd
import numpy as np
```

```
# Assuming dfzc942 contains the data with 'DateIdentifier', 'Store', and 'Sales' columns

# Create the new column '728prior' and initialize with 0
dfzc942['728prior'] = 0

# Create a DataFrame that maps each (Store, DateIdentifier) to the corresponding Sales value
sales_lookup = dfzc942.set_index(['Store', 'DateIdentifier'])['Sales'].to_dict()

# 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[210]:

	48	49	50	51	52	54	5
Sales	5263.000000	5020.000000	4782.000000	5011.000000	6102.000000	4364.000000	3706.0000
Store	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0000
DayOfWeek	5.000000	4.000000	3.000000	2.000000	1.000000	6.000000	5.0000
CompetitionDistanceDecile	4.000000	4.000000	4.000000	4.000000	4.000000	4.000000	4.0000
DateIdentifier	942.000000	941.000000	940.000000	939.000000	938.000000	936.000000	935.0000
AvgSalesPerStore	4759.096031	4759.096031	4759.096031	4759.096031	4759.096031	4759.096031	4759.0960
StoreRank	155.000000	155.000000	155.000000	155.000000	155.000000	155.000000	155.0000
NormalizedAvgSales_DayOfWeek	1.203903	1.152020	1.145309	1.206520	1.398491	1.000000	1.2039
NormalizedAvgSales_Promo	1.387686	1.387686	1.387686	1.387686	1.387686	1.000000	1.0000
NormalizedAvgSales_StateHoliday	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0000
NormalizedAvgSales_SchoolHoliday	1.044004	1.044004	1.044004	1.044004	1.044004	1.000000	1.0000
NormalizedAvgSales_CompetitionDistance	3.087343	3.087343	3.087343	3.087343	3.087343	3.087343	3.0873
NormalizedAvgSales_CompetitionDistanceDecile	1.067718	1.067718	1.067718	1.067718	1.067718	1.067718	1.0677
NormalizedAvgSales_PromoInterval	1.182585	1.182585	1.182585	1.182585	1.182585	1.182585	1.1825
NormalizedAvgSales_Type_Assort	1.062482	1.062482	1.062482	1.062482	1.062482	1.062482	1.0624
NormalizedAvgSales_PromoRun	1.655080	1.655080	1.655080	1.655080	1.655080	1.655080	1.6550
NormalizedAvgSales_P2cal	1.109664	1.109664	1.109664	1.109664	1.109664	1.109664	1.1096
NormalizedAvgSales_Year	1.037487	1.037487	1.037487	1.037487	1.037487	1.037487	1.0374
NormalizedAvgSales_WeekOfYear	1.319977	1.319977	1.319977	1.319977	1.039190	1.039190	1.0391
728prior	4494.000000	4994.000000	5572.000000	5773.000000	6290.000000	3352.000000	4221.0000
Predicted_DateIdentifier	214.000000	213.000000	212.000000	211.000000	210.000000	208.000000	207.0000

In [209…

```
dfzc942.tail(20).T
```

Out[209]:

	1058273	1058274	1058275	1058276	1058278	1058279	105828
Sales	4689.000000	4270.000000	5079.000000	6520.000000	3158.000000	3429.000000	3519.00000
Store	1110.000000	1110.000000	1110.000000	1110.000000	1110.000000	1110.000000	1110.00000
DayOfWeek	4.000000	3.000000	2.000000	1.000000	6.000000	5.000000	4.00000
CompetitionDistanceDecile	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	3.00000
DateIdentifier	24.000000	23.000000	22.000000	21.000000	19.000000	18.000000	17.00000
AvgSalesPerStore	4531.910600	4531.910600	4531.910600	4531.910600	4531.910600	4531.910600	4531.91060
StoreRank	115.000000	115.000000	115.000000	115.000000	115.000000	115.000000	115.00000
NormalizedAvgSales_DayOfWeek	1.152020	1.145309	1.206520	1.398491	1.000000	1.203903	1.15202
NormalizedAvgSales_Promo	1.387686	1.387686	1.387686	1.387686	1.000000	1.000000	1.00000
NormalizedAvgSales_StateHoliday	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000
NormalizedAvgSales_SchoolHoliday	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000
NormalizedAvgSales_CompetitionDistance	1.898796	1.898796	1.898796	1.898796	1.898796	1.898796	1.89879
NormalizedAvgSales_CompetitionDistanceDecile	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000
NormalizedAvgSales_PromoInterval	1.182585	1.182585	1.182585	1.182585	1.182585	1.182585	1.18258
NormalizedAvgSales_Type_Assort	1.093904	1.093904	1.093904	1.093904	1.093904	1.093904	1.09390
NormalizedAvgSales_PromoRun	1.655080	1.655080	1.655080	1.655080	1.655080	1.655080	1.65508
NormalizedAvgSales_P2cal	1.109664	1.109664	1.109664	1.109664	1.109664	1.109664	1.10966
NormalizedAvgSales_Year	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000
NormalizedAvgSales_WeekOfYear	1.050478	1.050478	1.050478	1.029845	1.029845	1.029845	1.02984
728prior	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
Predicted_DateIdentifier	-704.000000	-705.000000	-706.000000	-707.000000	-709.000000	-710.000000	-711.00000

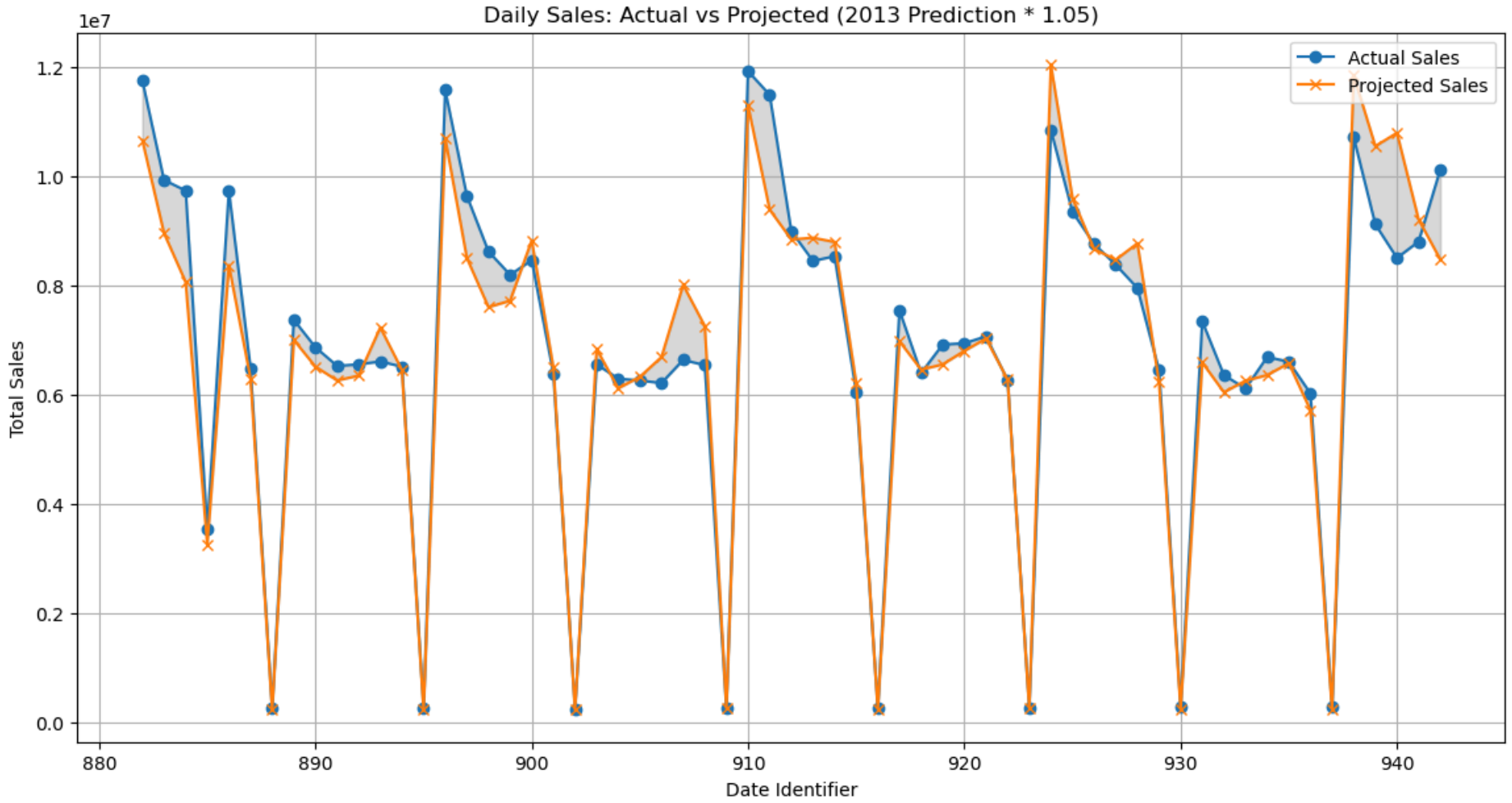
In [216…

```
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', and '728prior' columns

# Filter the data for DateIdentifier from 882 to 942
filtered_df = dfzc942[(dfzc942['DateIdentifier'] >= 882) & (dfzc942['DateIdentifier'] <= 942)]

# Rename columns for clarity
filtered_df = filtered_df.rename(columns={'Sales': 'Sales_2015', 'DateIdentifier': 'DateIdentifier_2015'})

# 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', 'Predicted_Sales']].copy()
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['Predicted_Sales']

# 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) ** 2))

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'] - daily_sales_comparison['Predicted_Sales

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_comparison['Sales_2015'], label='Actual Sales', ma
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Predicted_Sales'], label='Projected Sa
```

```
plt.fill_between(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Sales_2015'], daily_sales_comp
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Daily Sales: Actual vs Projected (2013 Prediction * 1.05)')
plt.legend()
plt.grid(True)
plt.show()
```

```
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
```



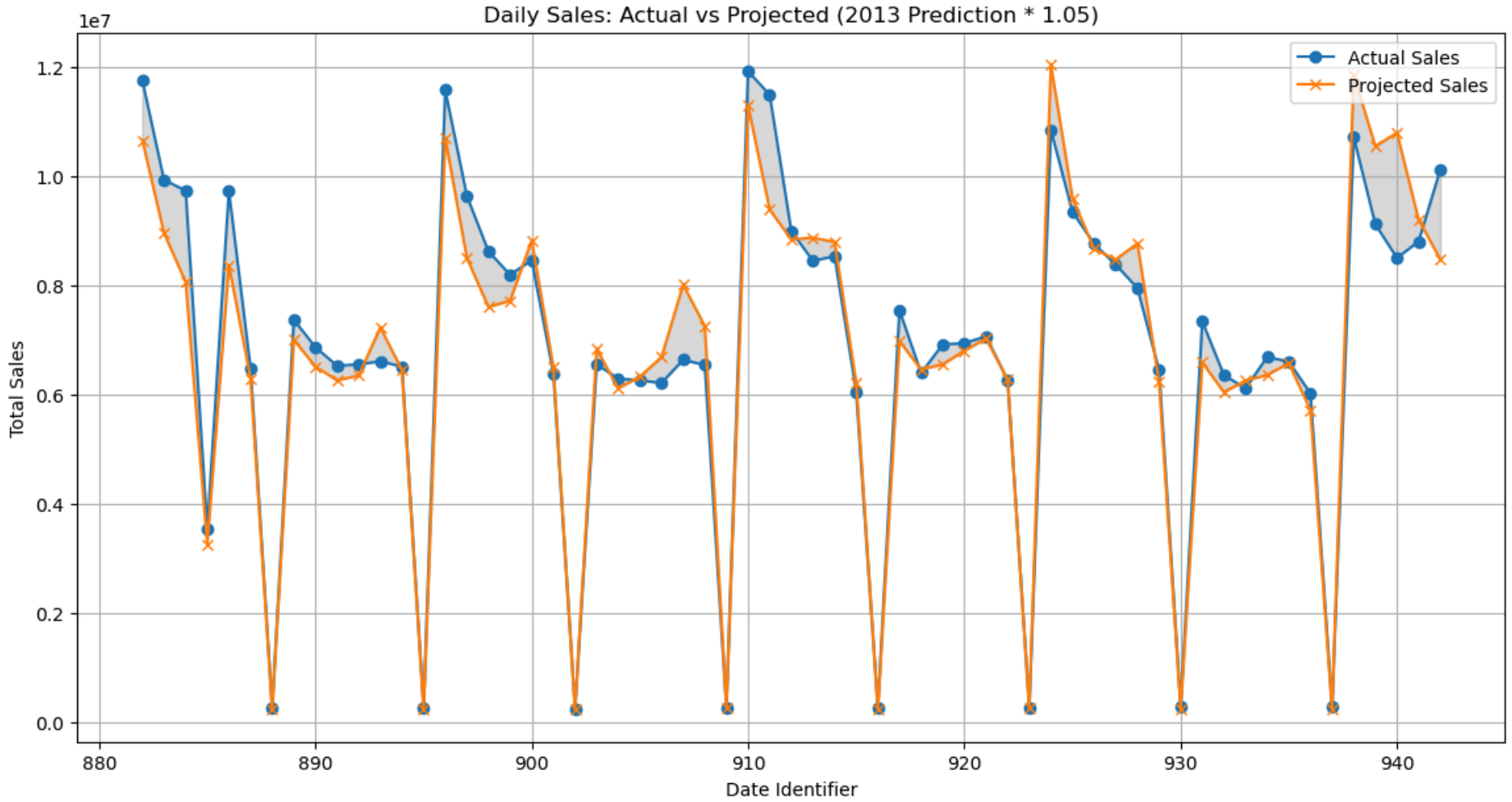
In [217…

```
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', and '728prior' columns

# Filter the data for DateIdentifier from 882 to 942
filtered_df = dfzc942[(dfzc942['DateIdentifier'] >= 882) & (dfzc942['DateIdentifier'] <= 942)]

# Rename columns for clarity
filtered_df = filtered_df.rename(columns={'Sales': 'Sales_2015', 'DateIdentifier': 'DateIdentifier_2015'})
```

```
# 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', 'Predicted_Sales']].copy()
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['Predicted_Sales']

# 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) ** 2))

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'] - daily_sales_comparison['Predicted_Sales

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_comparison['Sales_2015'], label='Actual Sales', ma
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Predicted_Sales'], label='Projected Sa
plt.fill_between(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Sales_2015'], daily_sales_comp
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Daily Sales: Actual vs Projected (2013 Prediction * 1.05)')
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
```

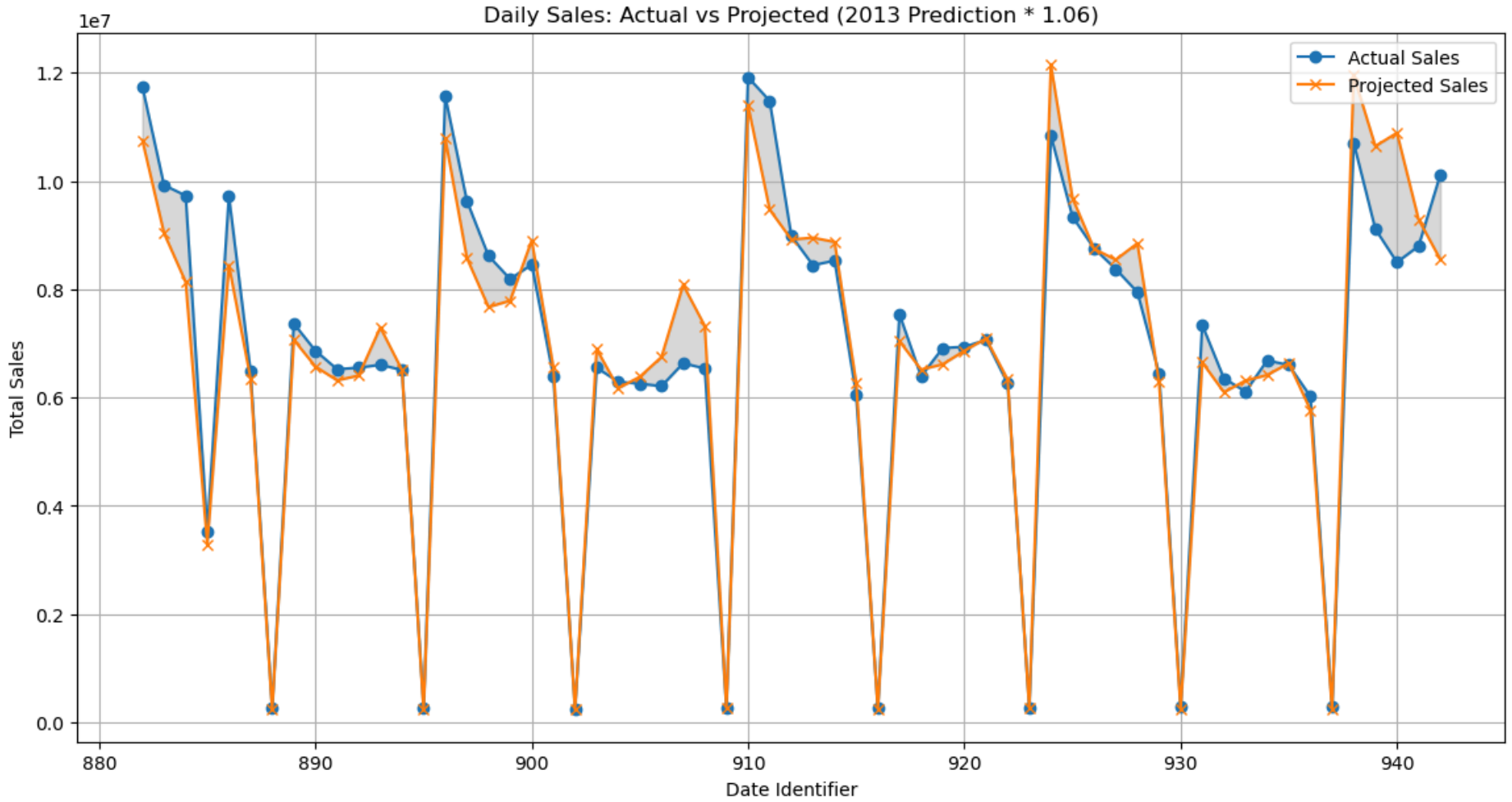
In [219…

```
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', and '728prior' columns

# Filter the data for DateIdentifier from 882 to 942
```

```
filtered_df = dfzc942[(dfzc942['DateIdentifier'] >= 882) & (dfzc942['DateIdentifier'] <= 942)]

# Rename columns for clarity
filtered_df = filtered_df.rename(columns={'Sales': 'Sales_2015', 'DateIdentifier': 'DateIdentifier_2015'})

# 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', 'Predicted_Sales']].copy()
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['Predicted_Sales']

# 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) ** 2))

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'] - daily_sales_comparison['Predicted_Sales

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_comparison['Sales_2015'], label='Actual Sales', ma
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Predicted_Sales'], label='Projected Sa
plt.fill_between(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Sales_2015'], daily_sales_comp
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
```

```
plt.title('Daily Sales: Actual vs Projected (2013 Prediction * 1.06)')
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
```

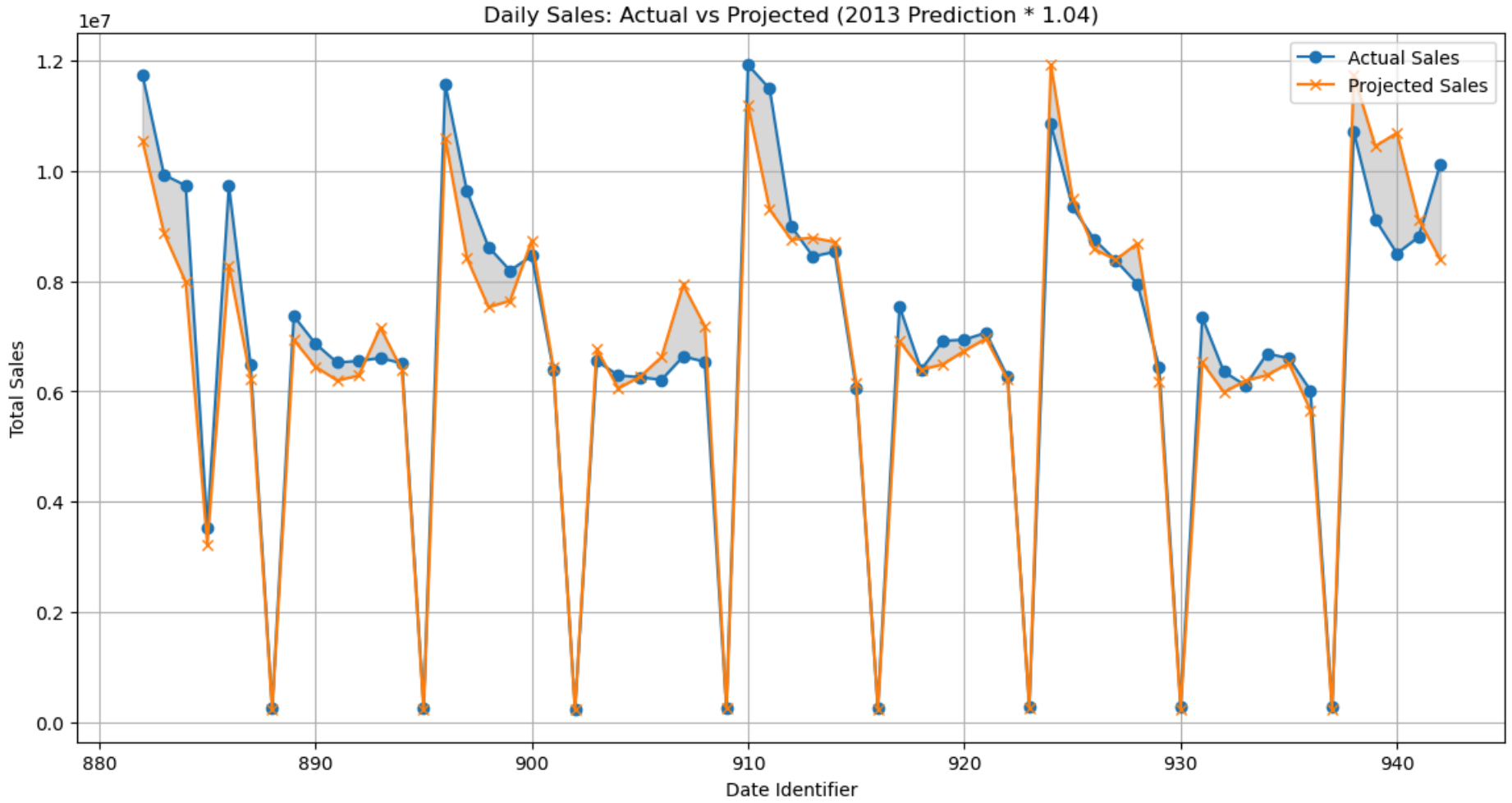
In [220…

```
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', and '728prior' columns

# Filter the data for DateIdentifier from 882 to 942
```

```
filtered_df = dfzc942[(dfzc942['DateIdentifier'] >= 882) & (dfzc942['DateIdentifier'] <= 942)]

# Rename columns for clarity
filtered_df = filtered_df.rename(columns={'Sales': 'Sales_2015', 'DateIdentifier': 'DateIdentifier_2015'})

# 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', 'Predicted_Sales']].copy()
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['Predicted_Sales']

# 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) ** 2))

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'] - daily_sales_comparison['Predicted_Sales

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_comparison['Sales_2015'], label='Actual Sales', ma
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Predicted_Sales'], label='Projected Sa
plt.fill_between(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Sales_2015'], daily_sales_comp
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
```

```
plt.title('Daily Sales: Actual vs Projected (2013 Prediction * 1.04)')
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 [203…

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

# Assuming dfzc942 contains the merged data for both years with 'DateIdentifier' and 'Store'

# Create a new column for the predicted DateIdentifier (728 days earlier)
```

```
dfzc942['Predicted_DateIdentifier'] = dfzc942['DateIdentifier'] - 728

# Merge the dataframe with itself on 'Store' and 'Predicted_DateIdentifier' to get the sales from 728 days earlier
predicted_df = pd.merge(
    dfzc942[['DateIdentifier', 'Store', 'Sales']],
    dfzc942[['DateIdentifier', 'Store', 'Sales']],
    left_on=['Store', 'Predicted_DateIdentifier'],
    right_on=['Store', 'DateIdentifier'],
    suffixes=('_2015', '_2013')
)

# Use Sales from 2013 (728 days earlier) as predictions for 2015
predicted_df['Predicted_Sales'] = predicted_df['Sales_2013'] * 1.05

# Create DataFrame to compare actual and predicted sales
comparison_df = predicted_df[['DateIdentifier_2015', 'Store', 'Sales_2015']].copy()
comparison_df['Predicted_Sales'] = predicted_df['Predicted_Sales']
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['Predicted_Sales']

# 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) ** 2))

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'] - daily_sales_comparison['Predicted_Sales

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_comparison['Sales_2015'], label='Actual Sales', ma
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Predicted_Sales'], label='Projected Sa
plt.fill_between(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Sales_2015'], daily_sales_comp
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Daily Sales: Actual vs Projected (2013 Prediction * 1.05)')
plt.legend()
plt.grid(True)
plt.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/var/folders/5f/7mc4blcs2mxgh06z6sgk8sc80000gp/T/ipykernel_2695/4204817300.py in ?()
      8 # Create a new column for the predicted DateIdentifier (728 days earlier)
      9 dfzc942['Predicted_DateIdentifier'] = dfzc942['DateIdentifier'] - 728
     10
     11 # Merge the dataframe with itself on 'Store' and 'Predicted_DateIdentifier' to get the sales from 728 days ea
rlier
---> 12 predicted_df = pd.merge(
     13     dfzc942[['DateIdentifier', 'Store', 'Sales']],
     14     dfzc942[['DateIdentifier', 'Store', 'Sales']],
     15     left_on=['Store', 'Predicted_DateIdentifier'],

~/anaconda3/lib/python3.11/site-packages/pandas/core/reshape/merge.py in ?(left, right, how, on, left_on, right_on, l
eft_index, right_index, sort, suffixes, copy, indicator, validate)
    165             validate=validate,
    166             copy=copy,
    167         )
    168     else:
--> 169         op = _MergeOperation(
    170             left_df,
    171             right_df,
    172             how=how,

~/anaconda3/lib/python3.11/site-packages/pandas/core/reshape/merge.py in ?(self, left, right, how, on, left_on, right
_on, left_index, right_index, sort, suffixes, indicator, validate)
    787             self.right_join_keys,
    788             self.join_names,
    789             left_drop,
    790             right_drop,
--> 791         ) = self._get_merge_keys()
    792
    793         if left_drop:
    794             self.left = self.left._drop_labels_or_levels(left_drop)

~/anaconda3/lib/python3.11/site-packages/pandas/core/reshape/merge.py in ?(self)
   1283                     if lk is not None:
   1284                         # Then we're either Hashable or a wrong-length arraylike,
   1285                         #  the latter of which will raise
   1286                         lk = cast(Hashable, lk)
-> 1287                         left_keys.append(left._get_label_or_level_values(lk))
   1288                         join_names.append(lk)
   1289                     else:
```

```
   1290                         # work-around for merge_asof(left_index=True)

~/anaconda3/lib/python3.11/site-packages/pandas/core/generic.py in ?(self, key, axis)
   1840             values = self.xs(key, axis=other_axes[0])._values
   1841         elif self._is_level_reference(key, axis=axis):
   1842             values = self.axes[axis].get_level_values(key)._values
   1843         else:
-> 1844             raise KeyError(key)
   1845
   1846         # Check for duplicates
   1847         if values.ndim > 1:

KeyError: 'Predicted_DateIdentifier'
```

In [ ]:

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

# Assuming dfzc942 contains the data for all DateIdentifiers from 1 to 942 with 'Store' and 'Sales' columns

# Filter the DataFrame to get the relevant DateIdentifiers for prediction (882 to 942)
df_for_prediction = dfzc942[(dfzc942['DateIdentifier'] >= 882) & (dfzc942['DateIdentifier'] <= 942)].copy()

# Calculate the corresponding DateIdentifier from two years earlier
df_for_prediction['Predicted_DateIdentifier'] = df_for_prediction['DateIdentifier'] - 728

# Merge the dataframe with itself on 'Store' and 'Predicted_DateIdentifier' to get the sales from 728 days earlier
predicted_df = pd.merge(
    df_for_prediction[['DateIdentifier', 'Store', 'Sales']],
    dfzc942[['DateIdentifier', 'Store', 'Sales']],
    left_on=['Store', 'Predicted_DateIdentifier'],
    right_on=['Store', 'DateIdentifier'],
    suffixes=('_2015', '_2013')
)

# Use Sales from 2013 (728 days earlier) as predictions for 2015 and multiply by 1.05
predicted_df['Predicted_Sales'] = predicted_df['Sales_2013'] * 1.05

# Create DataFrame to compare actual and predicted sales
comparison_df = predicted_df[['DateIdentifier_2015', 'Store', 'Sales_2015', 'Predicted_Sales']].copy()
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['Predicted_Sales']

# 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) ** 2))

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'] - daily_sales_comparison['Predicted_Sales

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_comparison['Sales_2015'], label='Actual Sales', ma
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Predicted_Sales'], label='Projected Sa
plt.fill_between(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Sales_2015'], daily_sales_comp
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Daily Sales: Actual vs Projected (2013 Prediction * 1.05)')
plt.legend()
plt.grid(True)
plt.show()
```

In [ ]:

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

# Assuming merged_df contains the merged data for both years with 'DateIdentifier' and 'Store'
```

```
# Create a new column for the predicted DateIdentifier (728 days earlier)
dfzc942['Predicted_DateIdentifier'] = dfzc942['DateIdentifier'] - 728

# Merge the dataframe with itself on 'Store' and 'Predicted_DateIdentifier' to get the sales from 728 days earlier
predicted_df = pd.merge(
    dfzc942[['DateIdentifier', 'Store', 'Sales']],
    dfzc942[['DateIdentifier', 'Store', 'Sales']],
    left_on=['Store', 'Predicted_DateIdentifier'],
    right_on=['Store', 'DateIdentifier'],
    suffixes=('_2015', '_2013')
)

# Use Sales from 2013 (728 days earlier) as predictions for 2015
predicted_df['Predicted_Sales'] = predicted_df['Sales_2013'] * 1.05

# Create DataFrame to compare actual and predicted sales
comparison_df = predicted_df[['DateIdentifier_2015', 'Store', 'Sales_2015']].copy()
comparison_df['Predicted_Sales'] = predicted_df['Predicted_Sales']
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['Predicted_Sales']

# 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) ** 2))

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'] - daily_sales_comparison['Predicted_Sales

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_comparison['Sales_2015'], label='Actual Sales', ma
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Predicted_Sales'], label='Projected Sa
plt.fill_between(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Sales_2015'], daily_sales_comp
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Daily Sales: Actual vs Projected (2013 Prediction * 1.05)')
plt.legend()
plt.grid(True)
plt.show()
```

In [ ]:

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

# Assuming dfcz942 contains data for DateIdentifier 1 to 942

# Filter data for DateIdentifier 882 to 942
filtered_df = dfzc942[dfzc942['DateIdentifier'].between(882, 942)]

# Calculate DateIdentifier 2 years earlier
filtered_df['Predicted_DateIdentifier'] = filtered_df['DateIdentifier'] - 728

# Merge with itself using the predicted DateIdentifier
merged_df = pd.merge(filtered_df, dfzc942, left_on=['Store', 'Predicted_DateIdentifier'], right_on=['Store', 'DateIden

# Use Sales from two years ago as predictions for current sales
merged_df['Predicted_Sales'] = merged_df['Sales_predicted']

# Calculate evaluation metrics
actual_sales = merged_df['Sales_actual']
predicted_sales = merged_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) ** 2))

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

```
# Filter for stores 1111-1115 and DateIdentifier 214
comparison_df = merged_df[(merged_df['Store'].between(1111, 1115)) & (merged_df['DateIdentifier_actual'] == 214)]

# Show the first few rows of the comparison
print("Comparison of Actual and Predicted Sales for Store IDs 1111-1115 on DateIdentifier 214:")
print(comparison_df[['DateIdentifier_actual', 'Store', 'Sales_actual', 'Predicted_Sales']])

# Plotting the comparison
plt.figure(figsize=(14, 7))
plt.plot(comparison_df['Store'], comparison_df['Sales_actual'], label='Actual Sales', marker='o', linestyle='')
plt.plot(comparison_df['Store'], comparison_df['Predicted_Sales'], label='Projected Sales', marker='x', linestyle='')
plt.fill_between(comparison_df['Store'], comparison_df['Sales_actual'], comparison_df['Predicted_Sales'], color='grey
plt.xlabel('Store ID')
plt.ylabel('Sales')
plt.title('Sales Comparison for Store IDs 1111-1115 on DateIdentifier 214')
plt.legend()
plt.grid(True)
plt.show()
```

In [ ]:

```
# Aggregate data before creating comparison DataFrame
agg_df = merged_df.groupby(['DateIdentifier_2015', 'Store']).agg({
    'Sales_2015': 'mean',  # Calculate average sales for each store on each date
    'Predicted_Sales': 'first'  # Take the first value of Predicted_Sales (assumed to be the same for each store)
}).reset_index()

# Create comparison DataFrame
comparison_df = agg_df[['DateIdentifier_2015', 'Store', 'Sales_2015']].copy()
comparison_df['Predicted_Sales'] = agg_df['Predicted_Sales']
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['Predicted_Sales']

# Calculate evaluation metrics
actual_sales = agg_df['Sales_2015']
predicted_sales = agg_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) ** 2))

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'] - daily_sales_comparison['Predicted_Sales

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_comparison['Sales_2015'], label='Actual Sales', ma
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Predicted_Sales'], label='Projected Sa
plt.fill_between(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Sales_2015'], daily_sales_comp
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Daily Sales: Actual vs Projected (2013 Prediction * 1.05)')
plt.legend()
plt.grid(True)
plt.show()
```

In [ ]:

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

# Assuming dfzc881 contains data for 2013 and dfzc61 contains data for 2015

# Merge the dataframes on 'DateIdentifier'
merged_df = pd.merge(dfzc61, dfzc881, on='DateIdentifier', suffixes=('_2015', '_2013'))

# Use Sales from 2013 as predictions for 2015
merged_df['Predicted_Sales'] = merged_df['Sales_2013']

# Create DataFrame to compare actual and predicted sales
comparison_df = merged_df[['DateIdentifier', 'Store_2015', 'Sales_2015']].copy()
comparison_df['Predicted_Sales'] = merged_df['Predicted_Sales']
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['Predicted_Sales']

# Calculate evaluation metrics
```

```
mse = mean_squared_error(comparison_df['Sales_2015'], comparison_df['Predicted_Sales'])
r2 = r2_score(comparison_df['Sales_2015'], comparison_df['Predicted_Sales'])
rmspe = np.sqrt(np.mean(((comparison_df['Sales_2015'] - comparison_df['Predicted_Sales']) / comparison_df['Sales_2015

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_2015': 'sum',
    'Predicted_Sales': 'sum'
}).reset_index()
daily_sales_comparison['Difference'] = daily_sales_comparison['Sales_2015'] - daily_sales_comparison['Predicted_Sales

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['Sales_2015'], label='Actual Sales', marker=
plt.plot(daily_sales_comparison['DateIdentifier'], daily_sales_comparison['Predicted_Sales'], label='Projected Sales',
plt.fill_between(daily_sales_comparison['DateIdentifier'], daily_sales_comparison['Sales_2015'], daily_sales_comparisc
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Daily Sales: Actual vs Projected')
plt.legend()
plt.grid(True)
plt.show()
```

In [ ]:

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

# Assuming dfzc881 contains data for 2013 and dfzc61 contains data for 2015

# Merge the dataframes on 'Store' and 'DateIdentifier'
merged_df = pd.merge(dfzc61, dfzc881, on=['Store', 'DateIdentifier'], suffixes=('_2015', '_2013'))
```

```
# Adjust DateIdentifier for 2013
merged_df['DateIdentifier_2013'] = merged_df['DateIdentifier_2015'] - 728  # Assuming 2 years earlier

# Use Sales from 2013 as predictions for 2015
merged_df['Predicted_Sales'] = merged_df['Sales_2013']

# Create DataFrame to compare actual and predicted sales
comparison_df = merged_df[['DateIdentifier_2015', 'Store', 'Sales_2015']].copy()
comparison_df['Predicted_Sales'] = merged_df['Predicted_Sales']
comparison_df['Difference'] = comparison_df['Sales_2015'] - comparison_df['Predicted_Sales']

# Calculate evaluation metrics
mse = mean_squared_error(comparison_df['Sales_2015'], comparison_df['Predicted_Sales'])
r2 = r2_score(comparison_df['Sales_2015'], comparison_df['Predicted_Sales'])
rmspe = np.sqrt(np.mean(((comparison_df['Sales_2015'] - comparison_df['Predicted_Sales']) / comparison_df['Sales_2015

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'] - daily_sales_comparison['Predicted_Sales

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_comparison['Sales_2015'], label='Actual Sales', ma
plt.plot(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Predicted_Sales'], label='Projected Sa
plt.fill_between(daily_sales_comparison['DateIdentifier_2015'], daily_sales_comparison['Sales_2015'], daily_sales_comp
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Daily Sales: Actual vs Projected')
plt.legend()
```

```
plt.grid(True)
plt.show()
```

In [ ]:

```
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score

# Assume dfzc881 contains data for 2013 and dfzc61 contains data for 2015



# Extract week number and day of the week
df_2013['WeekOfYear'] = df_2013['Date'].dt.isocalendar().week
df_2013['DayOfWeek'] = df_2013['Date'].dt.dayofweek + 1  # Monday=1, Sunday=7

df_2015['WeekOfYear'] = df_2015['Date'].dt.isocalendar().week
df_2015['DayOfWeek'] = df_2015['Date'].dt.dayofweek + 1

# Merge 2013 and 2015 dataframes on 'WeekOfYear' and 'DayOfWeek'
merged_df = pd.merge(df_2015, df_2013, on=['WeekOfYear', 'DayOfWeek'], suffixes=('_2015', '_2013'))

# Use Sales from 2013 as predictions for 2015
merged_df['Predicted_Sales'] = merged_df['Sales_2013']

# Actual sales for 2015
actual_sales = merged_df['Sales_2015']
predicted_sales = merged_df['Predicted_Sales']

# Calculate evaluation metrics
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) ** 2))

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

# Show the first few rows of actual vs predicted sales
output_df = merged_df[['Date_2015', 'Store_2015', 'Sales_2015', 'Predicted_Sales']]
output_df['Difference'] = output_df['Sales_2015'] - output_df['Predicted_Sales']
print(output_df.head())

# Plotting the actual vs predicted sales for each day
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(14, 7))
plt.plot(merged_df['Date_2015'], actual_sales, label='Actual Sales', color='blue')
plt.plot(merged_df['Date_2015'], predicted_sales, label='Predicted Sales', color='red', linestyle='--')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.title('Actual vs Predicted Sales for 2015')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```

In [ ]:

```
from fbprophet import Prophet

# Train Prophet model
model = Prophet()
model.fit(dfzc881.rename(columns={'DateIdentifier': 'ds', 'Sales': 'y'}))

# Make future dataframe
future = model.make_future_dataframe(periods=len(dfcz61))
forecast = model.predict(future)

# Extract forecasted values
predicted_sales = forecast['yhat'].tail(len(dfcz61))

# Calculate evaluation metrics
mse = mean_squared_error(dfcz61['Sales'], predicted_sales)
r2 = r2_score(dfcz61['Sales'], predicted_sales)
rmspe = np.sqrt(np.mean(((dfcz61['Sales'] - predicted_sales) / dfcz61['Sales']) ** 2))

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

# Create output DataFrame
output_df = dfcz61[['Store', 'DayOfWeek', 'WeekOfYear', 'Sales']].copy()
output_df['Predicted_Sales'] = predicted_sales
output_df['Difference'] = output_df['Sales'] - output_df['Predicted_Sales']

# Plotting the actual vs predicted sales for each day
plt.figure(figsize=(14, 7))
plt.plot(output_df.index, output_df['Sales'], label='Actual Sales', color='blue')
```

```
plt.plot(output_df.index, output_df['Predicted_Sales'], label='Predicted Sales', color='red', linestyle='--')
plt.xlabel('Index')
plt.ylabel('Sales')
plt.title('Actual vs Predicted Sales for 2015')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

In [ ]:
```
dfzc942.info()
```

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

# Assuming dfcz942 contains data for DateIdentifier 1 to 942

# Filter data for DateIdentifier 882 to 942
merged_df = dfzc942[(dfzc942['DateIdentifier'] >= 882) & (dfzc942['DateIdentifier'] <= 942)].copy()

# Use Sales from two years ago as predictions for current sales
merged_df['Predicted_Sales'] = merged_df['Sales']  # Assuming Sales from two years ago is the same as current sales

# Calculate evaluation metrics
actual_sales = merged_df['Sales']
predicted_sales = merged_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) ** 2))

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(merged_df[['DateIdentifier', 'Store', 'Sales', 'Predicted_Sales']].head())

# Calculate total sales per day (projected vs actual)
daily_sales_comparison = merged_df.groupby('DateIdentifier').agg({
    'Sales': 'sum',
    'Predicted_Sales': 'sum'
```

```
}).reset_index()
daily_sales_comparison['Difference'] = daily_sales_comparison['Sales'] - daily_sales_comparison['Predicted_Sales']

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['Sales'], label='Actual Sales', marker='o')
plt.plot(daily_sales_comparison['DateIdentifier'], daily_sales_comparison['Predicted_Sales'], label='Projected Sales',
plt.fill_between(daily_sales_comparison['DateIdentifier'], daily_sales_comparison['Sales'], daily_sales_comparison['Pr
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Daily Sales: Actual vs Projected (Sales from Two Years Ago)')
plt.legend()
plt.grid(True)
plt.show()
```

In [ ]:

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

# Assuming dfcz942 contains data for DateIdentifier 1 to 942

# Filter data for DateIdentifier 882 to 942
merged_df = dfzc942[(dfzc942['DateIdentifier'] >= 882) & (dfzc942['DateIdentifier'] <= 942)].copy()

# Calculate DateIdentifier 2 years earlier
merged_df['Predicted_DateIdentifier'] = merged_df['DateIdentifier'] - 728

# Merge with itself using the predicted DateIdentifier
merged_df = pd.merge(merged_df, dfzc942, left_on='Predicted_DateIdentifier', right_on='DateIdentifier', suffixes=('_ac

# Use Sales from two years ago as predictions for current sales
merged_df['Predicted_Sales'] = merged_df['Sales_predicted']

# Calculate evaluation metrics
actual_sales = merged_df['Sales_actual']
predicted_sales = merged_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) ** 2))
```

```
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(merged_df[['DateIdentifier_actual', 'Store_actual', 'Sales_actual', 'Predicted_Sales']].head())

# Calculate total sales per day (projected vs actual)
daily_sales_comparison = merged_df.groupby('DateIdentifier_actual').agg({
    'Sales_actual': 'sum',
    'Predicted_Sales': 'sum'
}).reset_index()
daily_sales_comparison['Difference'] = daily_sales_comparison['Sales_actual'] - daily_sales_comparison['Predicted_Sale

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_actual'], daily_sales_comparison['Sales_actual'], label='Actual Sales
plt.plot(daily_sales_comparison['DateIdentifier_actual'], daily_sales_comparison['Predicted_Sales'], label='Projected
plt.fill_between(daily_sales_comparison['DateIdentifier_actual'], daily_sales_comparison['Sales_actual'], daily_sales_
plt.xlabel('Date Identifier')
plt.ylabel('Total Sales')
plt.title('Daily Sales: Actual vs Projected (Sales from Two Years Ago)')
plt.legend()
plt.grid(True)
plt.show()
```

In [ ]:

```
import seaborn as sns

# Calculate the correlation matrix
correlation_matrix = dfzc942.corr()

# Create a heatmap to visualize the correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Matrix')
plt.show()
```

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Sample DataFrame creation for demonstration purposes
# Assuming dfz942 is your existing DataFrame and it has all the columns properly encoded
# For this example, we need to encode 'StateHoliday' and 'Type_Assort' if they are categorical


# Split data into features and target
X = dfzc942.drop(columns=['Sales'])  # Features
y = dfzc942['Sales']  # Target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the decision tree model
model = DecisionTreeRegressor(random_state=42)
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# Get feature importances
feature_importances = model.feature_importances_

# Create a DataFrame for feature importances
features_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': feature_importances
})

# Sort by importance
features_df = features_df.sort_values(by='Importance', ascending=False)
```

```
# Print feature importances
print("Feature Importances:")
print(features_df)

# Plot feature importances
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature', data=features_df)
plt.title('Feature Importances in Decision Tree Regressor')
plt.show()
```

In [ ]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Sample DataFrame creation for demonstration purposes
# Assuming dfz942 is your existing DataFrame and it has all the columns properly encoded
# For this example, we need to encode 'StateHoliday' and 'Type_Assort' if they are categorical

# Drop any rows with NaN values in the 'Sales' column
dfzc942 = dfzc942.dropna(subset=['Sales'])

# Split data into features and target
X = dfzc942.drop(columns=['Sales'])  # Features
y = dfzc942['Sales']  # Target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict on the test set
y_pred = 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("Root Mean Squared Percentage Error (RMSPE):", rmspe)
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# Get the coefficients and intercept
coefficients = model.coef_
intercept = model.intercept_

# Print the linear regression equation
equation = "Sales = " + " + ".join(f"{coef:.4f}*{col}" for coef, col in zip(coefficients, X.columns))
equation = f"Sales = {intercept:.4f} + " + " + ".join(f"{coef:.4f}*{col}" for coef, col in zip(coefficients, X.columns
print("Linear Regression Equation:")
print(equation)
```

In [ ]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming dfz942 is your existing DataFrame and it has all the columns properly encoded
# For this example, we need to encode 'StateHoliday' and 'Type_Assort' if they are categorical



# Drop any rows with NaN values in the 'Sales' column
dfzc942 = dfzc942.dropna(subset=['Sales'])

# Split data into features and target
X = dfzc942.drop(columns=['Sales'])  # Features
y = dfzc942['Sales']  # Target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the Random Forest Regressor model
model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)
```

```
# Predict on the test set
y_pred = 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("Root Mean Squared Percentage Error (RMSPE):", rmspe)
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# Get feature importances
feature_importances = model.feature_importances_

# Create a DataFrame for feature importances
features_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': feature_importances
})

# Sort the DataFrame by importance
features_df = features_df.sort_values(by='Importance', ascending=False)

# Print the feature importances
print(features_df)

# Plot the feature importances
plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature', data=features_df)
plt.title('Feature Importances in Random Forest')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```

In [ ]:

```
Mean Squared Error: 491759.5518689154
R-squared: 0.9492452934943248
Feature Importances:
                     Feature  Importance
2                  Customers    0.735940
```

```
11               Type_Assort    0.075755
7         CompetitionDistance    0.060811
0                       Store    0.036013
4                       Promo    0.034276
12                   PromoRun    0.018360
1                   DayOfWeek    0.011186
17                 WeekOfYear    0.007335
13             DateIdentifier    0.005295
9                PromoInterval    0.004090
15                  WeekNumber    0.003971
10  CompetitionDistanceDecile    0.003703
8                       Promo2    0.000988
14                      P2cal    0.000910
6               SchoolHoliday    0.000775
16                       Year    0.000325
5                StateHoliday    0.000268
3                        Open    0.000000

Mean Squared Error: 246233.24901414473
R-squared: 0.97458616464458
                      Feature  Importance
2                   Customers    0.737000
11                Type_Assort    0.074593
7         CompetitionDistance    0.060144
0                       Store    0.035604
4                       Promo    0.034452
12                   PromoRun    0.018698
1                   DayOfWeek    0.010988
17                 WeekOfYear    0.007903
13             DateIdentifier    0.005495
9                PromoInterval    0.004060
15                  WeekNumber    0.004022
10  CompetitionDistanceDecile    0.003533
8                       Promo2    0.001064
14                      P2cal    0.000955
6               SchoolHoliday    0.000839
16                       Year    0.000386
5                StateHoliday    0.000264
3                        Open    0.000000
```

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
```

```
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns

# Filter the DataFrame for rows with DateIdentifier from 1 to 942
selected_dfz942 = dfz942[dfz942['DateIdentifier'].between(1, 942)]

# Split data into features and target
X_selected = selected_dfz942.drop(columns=['Sales'])  # Features
y_selected = selected_dfz942['Sales']  # Target

# Split the selected data into training and testing sets
X_train_selected, X_test_selected, y_train_selected, y_test_selected = train_test_split(
    X_selected, y_selected, test_size=0.2, random_state=42)

# Train the Random Forest model using only the selected data
model_selected = RandomForestRegressor(n_estimators=100, random_state=42)
model_selected.fit(X_train_selected, y_train_selected)

# Predict on the test set
y_pred_selected = model_selected.predict(X_test_selected)

# Evaluate the model
mse_selected = mean_squared_error(y_test_selected, y_pred_selected)
r2_selected = r2_score(y_test_selected, y_pred_selected)

print(f"Mean Squared Error (Selected Data): {mse_selected}")
print(f"R-squared (Selected Data): {r2_selected}")

# Get feature importances from the Random Forest model
importances = model_selected.feature_importances_
feature_names = X_selected.columns

# Create a DataFrame for feature importances
feature_importances = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
feature_importances = feature_importances.sort_values(by='Importance', ascending=False)

# Plot the feature importances as a horizontal bar chart
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importances)
plt.title('Feature Importances from Random Forest')
plt.show()
```

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score



# Split data into features and target
X_selected = selected_dfz942.drop(columns=['Sales'])  # Features
y_selected = selected_dfz942['Sales']  # Target

# Split the selected data into training and testing sets
X_train_selected, X_test_selected, y_train_selected, y_test_selected = train_test_split(
    X_selected, y_selected, test_size=0.2, random_state=42)

# Train the decision tree model using only the selected data
model_selected = DecisionTreeRegressor()
model_selected.fit(X_train_selected, y_train_selected)

# Predict on the test set
y_pred_selected = model_selected.predict(X_test_selected)

# Evaluate the model
mse_selected = mean_squared_error(y_test_selected, y_pred_selected)
r2_selected = r2_score(y_test_selected, y_pred_selected)

print(f"Mean Squared Error (Selected Data): {mse_selected}")
print(f"R-squared (Selected Data): {r2_selected}")
```

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming dfz942 is your existing DataFrame and it has all the columns properly encoded
# Ensure 'StateHoliday', 'SchoolHoliday', 'Type_Assort', and 'PromoInterval' are in the DataFrame

# Calculate average sales for each category in the specified columns
categories = ['StateHoliday', 'SchoolHoliday', 'Type_Assort', 'PromoInterval']
average_sales_by_category = {}

for category in categories:
    average_sales = dfz942.groupby(category)['Sales'].mean().reset_index()
    average_sales_by_category[category] = average_sales
```

```
# Plot the bar charts for each category
plt.figure(figsize=(16, 20))

for i, category in enumerate(categories, 1):
    plt.subplot(len(categories), 1, i)
    sns.barplot(x='Sales', y=category, data=average_sales_by_category[category], orient='h')
    plt.title(f'Average Sales by {category}')
    plt.xlabel('Average Sales')
    plt.ylabel(category)

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

In [ ]:

In [ ]:

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Assuming 'df' contains features and target variable 'Sales'
X = dfz.drop(columns=['Sales'])  # Features
y = dfz['Sales']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize Random Forest Regressor
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model
rf_regressor.fit(X_train, y_train)

# Predictions
y_pred_train = rf_regressor.predict(X_train)
y_pred_test = rf_regressor.predict(X_test)

# Evaluate the model
train_rmse = mean_squared_error(y_train, y_pred_train, squared=False)
test_rmse = mean_squared_error(y_test, y_pred_test, squared=False)
```

```
print("Train RMSE:", train_rmse)
print("Test RMSE:", test_rmse)
```

In [ ]:

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor

# Assuming df is your DataFrame
# Select rows with DateIdentifier from 1 to 943
selected_dfz = dfz[dfz['DateIdentifier'].between(1, 943)]

# Split data into features and target
X_selected = selected_dfz.drop(columns=['Sales'])  # Features
y_selected = selected_dfz['Sales']  # Target

# Split the selected data into training and testing sets
X_train_selected, X_test_selected, y_train_selected, y_test_selected = train_test_split(
    X_selected, y_selected, test_size=0.2, random_state=42)

# Train the decision tree model using only the selected data
model_selected = DecisionTreeRegressor()
model_selected.fit(X_train_selected, y_train_selected)

# Predict on the test set
y_pred_selected = model_selected.predict(X_test_selected)

# Evaluate the model
from sklearn.metrics import mean_squared_error, r2_score

mse_selected = mean_squared_error(y_test_selected, y_pred_selected)
r2_selected = r2_score(y_test_selected, y_pred_selected)

print(f"Mean Squared Error (Selected Data): {mse_selected}")
print(f"R-squared (Selected Data): {r2_selected}")
```

In [ ]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

# Assuming df is your DataFrame
```

```
# Replace 'Sales' with the actual name of your target column if different
target_column = 'Sales'

# Prepare the feature matrix (X) and target vector (y)
X = dfz.drop(columns=[target_column])
y = dfz[target_column]

# Handle any missing values if necessary
# X = X.fillna(X.mean())
# y = y.fillna(y.mean())

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the decision tree regressor
model = DecisionTreeRegressor(random_state=42)
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# Visualize the decision tree
plt.figure(figsize=(20, 10))
plot_tree(model, feature_names=X.columns, filled=True, rounded=True)
plt.title("Decision Tree Regressor for Sales Prediction")
plt.show()
```

In [ ]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn import preprocessing

# Sample DataFrame
# dfz should be your DataFrame
# Check if 'Sales' column exists
if 'Sales' not in dfz.columns:
```

```
    raise KeyError("'Sales' column not found in the DataFrame.")

# Handle missing data if any
dfz = dfz.dropna()

# Convert categorical variables to numerical (if necessary)
# This example assumes 'StateHoliday' and 'Type_Assort' are categorical
label_encoder = preprocessing.LabelEncoder()
categorical_columns = ['StateHoliday', 'Type_Assort']
for col in categorical_columns:
    dfz[col] = label_encoder.fit_transform(dfz[col])

# Split data into features and target
X = dfz.drop(columns=['Sales'])  # Features
y = dfz['Sales']  # Target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the decision tree model
model = DecisionTreeRegressor()
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Evaluate the model (Optional)
from sklearn.metrics import mean_squared_error, r2_score

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# Display feature importances (Optional)
import matplotlib.pyplot as plt
import seaborn as sns

feature_importances = pd.Series(model.feature_importances_, index=X.columns)
feature_importances.nlargest(10).plot(kind='barh')
plt.title('Feature Importances')
plt.show()
```

In [ ]:
```
# Update P2Calc to 0 where Promo2 is 0
merged_df.loc[merged_df['Promo2'] == 0, 'P2cal'] = 0
```

In [ ]:
```

```

In [ ]:
```
merged_df["P2cal"].value_counts()
```

In [ ]:
```
print(merged_df['P2cal'].describe())
```

In [ ]:
```
merged_df.tail().T
```

In [ ]:
```
print(merged_df['P2calc'].describe())
```

In [ ]:
```
merged_df['P2calc'] = merged_df['P2calc'].apply(lambda x: 1 if x > 0 else 0)
```

In [ ]:
```
merged_df["P2calc"].value_counts()
```

In [ ]:
```
# Count of P2Calc values for Promo2 = 0
p2calc_promo2_0 = merged_df.loc[merged_df['Promo2'] == 0, 'P2calc'].value_counts()

# Count of P2Calc values for Promo2 = 1
p2calc_promo2_1 = merged_df.loc[merged_df['Promo2'] == 1, 'P2calc'].value_counts()

print("P2Calc counts for Promo2 = 0:")
print(p2calc_promo2_0)

print("\nP2Calc counts for Promo2 = 1:")
print(p2calc_promo2_1)
```

In [ ]:
```

```

In [ ]:
```
merged_df["P2calc"].value_counts()
merged_df["Promo2"].value_counts()
```

In [ ]:
```
# Create the new column P2
merged_df['P2New'] = np.where((merged_df['Promo2'] == 0) | (merged_df['P2calc'] == 0), 0, 1)
```

In [ ]:
```
merged_df["P2New"].value_counts()
```

In [ ]:
```
# Assuming 'train' is your DataFrame

# Filter sales for identifier 576
sales_576 = train[train['DateIdentifier'] == 506]['Sales']

# Filter sales for identifier 941
sales_941 = train[train['DateIdentifier'] == 513]['Sales']

# Calculate RMSPE
def rmspe(y_true, y_pred):
    rmspe = np.sqrt(np.mean(((y_true - y_pred) / y_true)**2))
    return rmspe

rmspe_value = rmspe(sales_576, sales_941)

print("RMSPE between sales of identifier 576 and 941:", rmspe_value)
```

In [ ]:
```
# Filter sales for identifier 576
sales_576 = train[train['DateIdentifier'] == 942]['Sales']

# Filter sales for identifier 941
sales_941 = train[train['DateIdentifier'] == 914]['Sales']

print("Sales for identifier 576:")
print(sales_576)

print("\nSales for identifier 941:")
print(sales_941)
```

In [ ]:
```
import numpy as np

# Assuming 'train' is your DataFrame

# Filter sales data for the two specific date identifiers
date_identifier_1 = 2  # First date identifier
date_identifier_2 = 366  # Second date identifier (1 week later)

sales_date_1 = train[train['DateIdentifier'] == date_identifier_1].set_index('Store')['Sales']
sales_date_2 = train[train['DateIdentifier'] == date_identifier_2].set_index('Store')['Sales']

# Calculate RMSPE
def rmspe(y_true, y_pred):
```

```
    rmspe = np.sqrt(np.mean(((y_true - y_pred) / y_true)**2))
    return rmspe

# Remove NaN and infinity values before calculating RMSPE
sales_date_1_clean = sales_date_1.replace([np.inf, -np.inf], np.nan).dropna()
sales_date_2_clean = sales_date_2.replace([np.inf, -np.inf], np.nan).dropna()

# Calculate RMSPE for the common stores
rmspe_value = rmspe(sales_date_1_clean, sales_date_2_clean)

print("RMSPE between sales on DateIdentifier", date_identifier_1, "and", date_identifier_2, "for all stores:", rmspe_v
```

In [ ]:

```
import numpy as np

# Define the prediction and actual period identifiers
prediction_period_start = 366
prediction_period_end = 415
actual_period_start = 2
actual_period_end = 49

# Initialize a list to store RMSPE values for each day
rmspe_values = []

# Calculate RMSPE for each day in the prediction period
for i in range(prediction_period_start, prediction_period_end + 1):
    # Filter sales data for the prediction day
    sales_prediction = train[train['DateIdentifier'] == i].set_index('Store')['Sales']

    # Filter sales data for the actual day (48 days earlier)
    actual_day = i - (prediction_period_end - prediction_period_start)
    sales_actual = train[train['DateIdentifier'] == actual_day].set_index('Store')['Sales']

    # Remove NaN and infinity values before calculating RMSPE
    sales_prediction_clean = sales_prediction.replace([np.inf, -np.inf], np.nan).dropna()
    sales_actual_clean = sales_actual.replace([np.inf, -np.inf], np.nan).dropna()

    # Calculate RMSPE for the current day
    rmspe_value = rmspe(sales_actual_clean, sales_prediction_clean)
    rmspe_values.append(rmspe_value)

# Average RMSPE values for all days
average_rmspe = np.mean(rmspe_values)
```

```
print("Average RMSPE for the prediction period:", average_rmspe)
```

In [ ]:
```
# Assuming 'train' is your DataFrame

# Calculate the week number
train['WeekNumber'] = (train['DateIdentifier'] - 1) // 7 + 1

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

In [ ]:
```
print(train.tail())
```

In [ ]:
```
import matplotlib.pyplot as plt

# Aggregate sales data by week and calculate total sales per week
weekly_sales = train.groupby('WeekNumber')['Sales'].sum()

# Plot the total sales per week
plt.figure(figsize=(10, 6))
plt.plot(weekly_sales.index, weekly_sales.values, marker='o', linestyle='-')
plt.title('Total Sales per Week')
plt.xlabel('Week Number')
plt.ylabel('Total Sales')
plt.grid(True)
plt.show()
```

In [ ]:
```
import matplotlib.pyplot as plt

# Assuming 'train' is your DataFrame

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

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

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

for year in range(1, train['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 [ ]:

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

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

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

for year in range(1, train['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()

# Set the x-axis limits to show only weeks 1-52
plt.xlim(1, 52)

plt.show()
```

In [ ]:

```
import matplotlib.pyplot as plt

# Assuming 'train' is your DataFrame

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

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

```
# Group data by year and week, and calculate total sales per week
weekly_sales = train.groupby(['Year', 'WeekOfYear'])['Sales'].sum().reset_index()

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

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

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

plt.show()
```

In [ ]:

```
import matplotlib.pyplot as plt

# Assuming 'train' is your DataFrame

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

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

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

# 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 {year}')

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 [ ]:
```
train.info()
```

In [ ]:
```
#Merge # Merge the train and stores datasets on the "Store" column
merged_train_stores = pd.merge(train, store, on="Store", how="outer")

# Display the final dataset
print(merged_train_stores)
```

In [ ]:
```
merged_train_stores.info()
```

In [ ]:
```
#Merge # Merge the train and stores datasets on the "Store" column
merged_train_stores3 = pd.merge(train, store3, on="Store", how="outer")

# Display the final dataset
print(merged_train_stores3)
```

In [ ]:
```
merged_train_stores3.info()
```

In [ ]:
```
# Rearrange the columns
merged_train_stores3 = merged_train_stores3[['Sales', 'Customers'] + [col for col in merged_train_stores3.columns if c

# Display the DataFrame after rearranging columns
print(merged_train_stores3)
```

In [ ]:
```
merged_train_stores3.describe().T
```

In [ ]:
```
merged_train_stores3.numeric= merged_train_stores3.select_dtypes(include=np.number)
merged_train_stores3.numeric.corr()
```

In [ ]:
```
# Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(merged_train_stores3.numeric.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Numeric Columns')
plt.show()
```

In [ ]:
```
# Filter out rows where Sales is not equal to 0
mdf_nz = merged_train_stores3[merged_train_stores['Sales'] != 0].copy()

# Now mdf_no_zero_sales DataFrame contains rows where Sales is not equal to 0
mdf_nz.info()
```

In [ ]:
```
mdf_nz['Promo2'].value_counts()
```

In [ ]:
```
# Assuming 'df' is your DataFrame and 'store_id' is the ID of the store you're interested in
PromoRun = 6  # Assuming Promo2 has been running for 6 weeks
DateIdentifier = 942  # Assuming Promo2 ends on day 942

# Calculate the first day when Promo2 is not running
promo2_last_day = promo2_end_day - promo2_run_weeks * 7  # Assuming 7 days in a week

# Iterate over each row and update the Promo value based on Promo2 run weeks
for index, row in mdf_nz.iterrows():
    if row['Promo2'] == 1 and row['DayOfWeek'] > promo2_last_day:
        mdf_nz.at[index, 'Promo'] = 0
```

In [ ]:
```
mdf_nz['Promo2'].value_counts()
```

In [ ]:
```
# List all columns in the DataFrame
columns_list = mdf_nz.columns.tolist()

# Print the list of columns
print(columns_list)
```

In [ ]:
```
mdf_nz.describe().T
```

In [ ]:
```
# Sort the DataFrame by the 'Customers' column in descending order
top_customers = mdf_nz.sort_values(by='Customers', ascending=False)

# Take the top 10 rows
top_30_customers = top_customers.head(30)

# Display the top 10 rows
print(top_30_customers)
```

In [ ]:
```
mdf_nz.head().T
```

In [ ]:
```
# Filter the DataFrame for Store 817
store_817_data = mdf_nz[mdf_nz['Store'] == 817]

# Group by Date and get the sales and customers figures
grouped_date = store_817_data.groupby('Date').agg({'Sales': 'sum', 'Customers': 'sum'})

# Display the grouped data
print(grouped_date.head(30))
```

In [ ]:
```
# Calculate total sales and total customers for Store 817
total_sales = grouped_date['Sales'].sum()
total_customers = grouped_date['Customers'].sum()

# Calculate average sales per customer
average_sales_per_customer = total_sales / total_customers

# Display the result
print("Average Sales per Customer for Store 817:", average_sales_per_customer)
```

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

In [ ]:
```
+27190/3788
```

In [ ]:
```
# Group the data by date and calculate the sum of sales and customers
grouped_by_date =  mdf_nz[ mdf_nz['Store'] == 817].groupby('Date')['Sales', 'Customers'].reset_index()

# Sort the grouped data by the number of customers in descending order
sorted_by_customers = grouped_by_date.sort_values(by='Customers', ascending=False)

# Display the sorted data
print(sorted_by_customers)
```

In [ ]:
```
# Get the lowest 10 sales
lowest_sales = mdf_nz.nsmallest(10, 'Sales')

# Print the lowest 10 sales
print(lowest_sales)
```

In [ ]:
```
# Get the lowest 10 sales
largest_sales = mdf_nz.nlargest(10, 'Sales')
```

```
# Print the lowest 10 sales
print(largest_sales)
```

In [ ]:
```
mdf_nz.numeric= mdf_nz.select_dtypes(include=np.number)
mdf_nz.numeric.corr()
```

In [ ]:
```
# Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(mdf_nz.numeric.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Numeric Columns')
plt.show()
```

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import statsmodels.api as sm

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

# Fit and plot regression line
X = mdf_nz['CompetitionDistance']
y = mdf_nz['Sales']
X = sm.add_constant(X)  # Add constant for intercept
model = sm.OLS(y, X).fit()
plt.plot(X['CompetitionDistance'], model.predict(X), color='red', linewidth=2)

# Set plot labels and title
plt.xlabel('Competition Distance')
plt.ylabel('Sales')
plt.title('Scatter Plot and Regression Line of Sales vs Competition Distance')

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

In [ ]:
```
# Create scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='Customers', y='Sales', data=mdf_nz)

# Fit and plot regression line
```

```
X = mdf_nz['Customers']
y = mdf_nz['Sales']
X = sm.add_constant(X)  # Add constant for intercept
model = sm.OLS(y, X).fit()
plt.plot(X['Customers'], model.predict(X), color='red', linewidth=2)

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

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

In [ ]:

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

# Fit linear regression model
X = mdf_nz['Customers']
y = mdf_nz['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=12, color='blue')

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

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

In [ ]:
```
# Group the data by 'Store' and calculate the mean sales
average_sales_per_store = mdf_nz.groupby('Store')['Sales'].mean()

# Create a new column 'AverageSales' with the average sales for each store
mdf_nz['AverageSales'] = mdf_nz['Store'].map(average_sales_per_store)

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

In [ ]:
```
# Calculate the average sales per store
average_sales_per_store = mdf_nz.groupby('Store')['Sales'].mean()

# Calculate the ranking of each store based on average sales
store_ranking = average_sales_per_store.rank(ascending=False)

# Map the store ranking to all rows corresponding to each store
mdf_nz['StoreRank'] = mdf_nz['Store'].map(store_ranking)

# Display the DataFrame with the new 'StoreRank' column
print(mdf_nz)
```

In [ ]:
```
# Create scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='AverageSales', y='Sales', data=mdf_nz)

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

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

# Annotate regression equation
slope = model.params['AverageSales']
intercept = model.params['const']
equation = f'y = {slope:.2f} * x + {intercept:.2f}'
plt.annotate(equation, xy=(0.05, 0.95), xycoords='axes fraction', fontsize=12, color='blue')

# Set plot labels and title
plt.xlabel('AverageSales')
```

```
plt.ylabel('Sales')
plt.title('Scatter Plot and Regression Line of Sales vs AverageSales')

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

In [ ]:
```
# Create scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='StoreRank', y='AverageSales', hue="Type_Assort", data=mdf_nz)

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

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

# Annotate regression equation
slope = model.params['StoreRank']
intercept = model.params['const']
equation = f'y = {slope:.2f} * x + {intercept:.2f}'
plt.annotate(equation, xy=(0.05, 0.95), xycoords='axes fraction', fontsize=12, color='blue')

# Set plot labels and title
plt.xlabel('StoreRank')
plt.ylabel('Sales')
plt.title('Scatter Plot and Regression Line of StoreRank vs AverageSales')

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

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming df is your DataFrame
plt.figure(figsize=(10, 6))
sns.boxplot(x='Type_Assort', y='Sales', data=mdf_nz)
plt.title('Box plot of Sales by Type_Assort')
plt.xlabel('Type_Assort')
```

```
plt.ylabel('Sales')
plt.show()
```

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming df is your DataFrame
plt.figure(figsize=(10, 6))
sns.boxplot(x='Type_Assort', y='AverageSales', data=mdf_nz)
plt.title('Box plot of Aveargs Sales by StoreType/Assortment')
plt.xlabel('StoreType/Assortment')
plt.ylabel('Average Sales')
plt.show()
```

In [ ]:
```
# Create scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='StoreRank', y='Sales', data=mdf_nz)

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

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

# Annotate regression equation
slope = model.params['StoreRank']
intercept = model.params['const']
equation = f'y = {slope:.2f} * x + {intercept:.2f}'
plt.annotate(equation, xy=(0.05, 0.95), xycoords='axes fraction', fontsize=12, color='blue')

# Set plot labels and title
plt.xlabel('StoreRank')
plt.ylabel('Sales')
plt.title('Scatter Plot and Regression Line of Sales vs AverageSales')

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

In [ ]:
```
# Create scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='StoreRank', y='Sales', hue="StoreType", data=mdf_nz)

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

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

# Annotate regression equation
slope = model.params['StoreRank']
intercept = model.params['const']
equation = f'y = {slope:.2f} * x + {intercept:.2f}'
plt.annotate(equation, xy=(0.05, 0.95), xycoords='axes fraction', fontsize=12, color='blue')

# Set plot labels and title
plt.xlabel('StoreRank')
plt.ylabel('Sales')
plt.title('Scatter Plot and Regression Line of Sales vs AverageSales')

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

In [ ]:
```
# Compute the correlation between 'AverageSales' and 'Sales'
correlation = mdf_nz['AverageSales'].corr(mdf_nz['Sales'])

print("Correlation between AverageSales and Sales:", correlation)
```

In [ ]:
```
# Compute the correlation between 'AverageSales' and 'Sales'
correlation = mdf_nz['StoreRank'].corr(mdf_nz['Sales'])

print("Correlation between AverageSales and Sales:", correlation)
```

In [ ]:
```
# Compute the correlation between 'AverageSales' and 'Sales'
correlation = mdf_nz['Customers'].corr(mdf_nz['Sales'])

print("Customers and Sales:", correlation)
```

In [ ]:
```
# Calculate average sales per store
average_sales_per_store = mdf_nz.groupby('Store')['Sales'].mean()

# Merge average sales with mdf_no_zero_sales
mdf_avg_sales = mdf_nz.merge(average_sales_per_store, on='Store', suffixes=('', '_avg'))

# Create scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='CompetitionDistance', y='Sales_avg', data=mdf_avg_sales)

# Fit and plot regression line
X = mdf_avg_sales['CompetitionDistance']
y = mdf_avg_sales['Sales_avg']
X = sm.add_constant(X)  # Add constant for intercept
model = sm.OLS(y, X).fit()
plt.plot(X['CompetitionDistance'], model.predict(X), color='red', linewidth=2)

# Set plot labels and title
plt.xlabel('Competition Distance')
plt.ylabel('Average Sales per Store')
plt.title('Scatter Plot and Regression Line of Average Sales per Store vs Competition Distance')

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

In [ ]:
```
# Calculate average sales per store
average_sales_per_store = mdf_nz.groupby('Store')['Sales'].mean()

# Merge average sales with mdf_no_zero_sales
mdf_avg_sales = mdf_nz.merge(average_sales_per_store, on='Store', suffixes=('', '_avg'))

# Create scatter plot with color based on StoreType
plt.figure(figsize=(10, 6))
sns.scatterplot(x='CompetitionDistance', y='Sales_avg', hue='Assortment', data=mdf_avg_sales)

# Fit and plot regression line
```

```
X = mdf_avg_sales['CompetitionDistance']
y = mdf_avg_sales['Sales_avg']
X = sm.add_constant(X)  # Add constant for intercept
model = sm.OLS(y, X).fit()
plt.plot(X['CompetitionDistance'], model.predict(X), color='black', linewidth=2, label='Regression Line')

# Set plot labels and title
plt.xlabel('Competition Distance')
plt.ylabel('Average Sales per Store')
plt.title('Scatter Plot of Average Sales per Store vs Competition Distance')

# Show legend
plt.legend()

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

In [ ]:
```
# Create box plot for sales by assortment
plt.figure(figsize=(10, 6))
sns.boxplot(x='Assortment', y='Sales', data=mdf_nz)

# Set plot labels and title
plt.xlabel('Assortment')
plt.ylabel('Sales')
plt.title('Box Plot of Sales by Assortment')

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

In [ ]:
```
# Select only numerical features
numerical_features = merged_train_stores.select_dtypes(include=['float64', 'int64'])

# Compute the correlation matrix
correlation_matrix = numerical_features.corr()

# Print the correlation matrix
print("Correlation Matrix:")
correlation_matrix
```

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

```
# Define features (X) and target variable (y)
X = mdf_nz[['StoreType']]

# Create Random Forest model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the model
rf_model.fit(X, y)

# Get feature importances
feature_importances = rf_model.feature_importances_

# Create a DataFrame to store feature importances
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importances})

# Sort feature importances in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Print the top 10 most important features
print(feature_importance_df.head(10))
```

In [ ]:

```
# Define features (X) and target variable (y)
X = mdf_nz[['Store', 'DayOfWeek', 'Customers', 'Promo', 'Open', 'DateIdentifier', 'WeekNumber']]  # Replace 'Feature1
y = mdf_nz['Sales']  # Replace 'Sales' with your target variable

# Create Random Forest model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the model
rf_model.fit(X, y)

# Get feature importances
feature_importances = rf_model.feature_importances_

# Create a DataFrame to store feature importances
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importances})

# Sort feature importances in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Print the top 10 most important features
print(feature_importance_df.head(10))
```

In [ ]:
```
# Define features (X) and target variable (y)
X = mdf_nz[['Store', 'DayOfWeek', 'Promo', 'Open', 'DateIdentifier', 'StateHoliday', 'SchoolHoliday', 'WeekNumber', 'Y
            'CompetitionDistance', 'CompetitionOpenSinceMonth', 'CompetitionOpenSinceYear',
            'Promo2', 'Promo2SinceWeek', 'Promo2SinceYear', 'PromoInterval']]  # Replace 'Feature1', 'Feature2', ... w
y = mdf_nz['Sales']  # Replace 'Sales' with your target variable

# Create Random Forest model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the model
rf_model.fit(X, y)

# Get feature importances
feature_importances = rf_model.feature_importances_

# Create a DataFrame to store feature importances
feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importances})

# Sort feature importances in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Print the top 10 most important features
print(feature_importance_df.head(10))
```

In [ ]:
```
import numpy as np
import statsmodels.api as sm

# Define features (X) using the top features from Random Forest
X = mdf_nz[['Customers', 'Store', 'Promo', 'DayOfWeek', 'DateIdentifier', 'WeekNumber']]

# Add constant for intercept
X = sm.add_constant(X)

# Define target variable (y)
y = mdf_nz['Sales']

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

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

In [ ]:
```
# Assuming your DataFrame is named df
average_sales_per_combination = mdf_nz.groupby(['StoreType', 'Assortment'])['Sales'].mean()

print(average_sales_per_combination)
```

In [ ]:
```
# Assuming your DataFrame is named mdf_nz
sales_count_per_combination = mdf_nz.groupby(['StoreType', 'Assortment']).size()

print(sales_count_per_combination)
```

In [ ]:
```
store_count_per_combination = mdf_nz.groupby(['StoreType', 'Assortment'])['Store'].nunique()

print(store_count_per_combination)
```

In [ ]:
```
store_count_per_combination = mdf_nz.groupby(['StoreType', 'Assortment'])['Store'].nunique().sum()

print(store_count_per_combination)
```

In [ ]:
```
total_stores_per_combination = mdf_nz.groupby(['StoreType', 'Assortment']).size().sum()

print(total_stores_per_combination)
```

In [ ]:
```
total_stores_per_combination = mdf_nz.groupby(['StoreType', 'Assortment']).size().sum()

print(total_stores_per_combination)
```

In [ ]:
```
# Compute the correlation matrix
correlation_matrix = numerical_features.corr()

# Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Numeric Columns')
plt.show()
```

In [ ]:
```
merged_train_stores.info()
```

In [ ]:
```
import matplotlib.pyplot as plt

# Plot histogram of sales per store
```

```
plt.figure(figsize=(10, 6))
plt.hist(merged_train_stores['Sales'], bins=50, color='skyblue', edgecolor='black')
plt.title('Histogram of Sales per Store')
plt.xlabel('Sales')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

In [ ]:
```
import matplotlib.pyplot as plt

# Calculate average sales per store
average_sales_per_store = merged_train_stores.groupby('Store')['Sales'].mean()

# Plot histogram of average sales per store
plt.figure(figsize=(10, 6))
plt.hist(average_sales_per_store, bins=50, color='skyblue', edgecolor='black')
plt.title('Histogram of Average Sales per Store')
plt.xlabel('Average Sales')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

In [ ]:
```
# Select only non-object columns
non_object_df = merged_train_stores.select_dtypes(exclude=['object'])

# Print the non-object columns
print("Non-object columns:")
print(non_object_df.columns)

# Display the new DataFrame
print("New DataFrame with only non-object columns:")
non_object_df.head()
```

In [ ]:
```
# Select only numerical features
nf =non_object_df.select_dtypes(include=['float64', 'int64'])

# Compute the correlation matrix
cm = nf.corr()

# Print the correlation matrix
print("Correlation Matrix:")
cm
```

In [ ]:
```
# Check for constant columns
constant_columns = [col for col in merged_train_stores.columns if merged_train_stores[col].nunique() == 1]
merged_train_stores = merged_train_stores.drop(columns=constant_columns)

# Convert object columns to numeric
merged_train_stores = merged_train_stores.apply(pd.to_numeric, errors='ignore')

# Compute the correlation matrix
correlation_matrix = merged_train_stores.corr()

# Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Numeric Columns')
plt.show()
```

In [ ]:
```
# Identify the missing DateIdentifier for each store
missing_date_identifiers = train[train['Sales'].isnull()]['DateIdentifier'].unique()

# Create a DataFrame to store the missing rows
missing_rows = pd.DataFrame(columns=train.columns)

# Iterate over each missing DateIdentifier
for date_identifier in missing_date_identifiers:
    # Iterate over each store
    for store in train['Store'].unique():
        # Find the prior year row for the same DateIdentifier and store
        prior_year_row = prior_year_sales[(prior_year_sales['Store'] == store) &
                                          (prior_year_sales['DateIdentifier'] == date_identifier)]

        # If prior year row exists, use its sales value as the estimate for the missing value
        if not prior_year_row.empty:
            # Add the row to the missing_rows DataFrame
            missing_rows = missing_rows.append(prior_year_row)

# Concatenate the missing_rows DataFrame with the original train DataFrame
train = pd.concat([train, missing_rows], ignore_index=True)


# Now, 'train' DataFrame contains the missing rows added with sales estimates using prior year data
```

In [ ]:
```
# Identify the missing DateIdentifier values
missing_date_identifiers = train[train['Sales'].isnull()]['DateIdentifier'].unique()

# Create a DataFrame to store the missing rows
missing_rows = pd.DataFrame(columns=train.columns)

# Iterate over each missing DateIdentifier
for date_identifier in missing_date_identifiers:
    # Find the corresponding DateIdentifier from the prior year
    prior_year_date_identifier = date_identifier - 364

    # Find the rows from the prior year with the corresponding DateIdentifier
    prior_year_rows = train[train['DateIdentifier'] == prior_year_date_identifier]

    # Add the rows to the missing_rows DataFrame
    missing_rows = missing_rows.append(prior_year_rows, ignore_index=True)

# Concatenate the missing_rows DataFrame with the original train DataFrame
train = pd.concat([train, missing_rows], ignore_index=True)
```

In [ ]:
```
# Identify the unique store IDs in the original train data
unique_stores = train['Store'].unique()

# Identify the missing DateIdentifier values
missing_date_identifiers = train[train['Sales'].isnull()]['DateIdentifier'].unique()

# Create a DataFrame to store the missing rows
missing_rows = pd.DataFrame(columns=train.columns)

# Iterate over each missing DateIdentifier and each store ID
for date_identifier in missing_date_identifiers:
    for store_id in unique_stores:
        # Find the corresponding DateIdentifier from the prior year
        prior_year_date_identifier = date_identifier - 364

        # Find the row from the prior year with the corresponding DateIdentifier and store ID
        prior_year_row = train[(train['Store'] == store_id) & (train['DateIdentifier'] == prior_year_date_identifier)]

        # Add the row to the missing_rows DataFrame
        missing_rows = missing_rows.append(prior_year_row, ignore_index=True)

# Concatenate the missing_rows DataFrame with the original train DataFrame
```

```
train = pd.concat([train, missing_rows], ignore_index=True)
train.info()
```

In [ ]:
```
# Print information about the missing rows DataFrame
print("Information about the missing rows DataFrame:")
print(missing_rows.info())

# Concatenate the missing_rows DataFrame with the original train DataFrame
train = pd.concat([train, missing_rows], ignore_index=True)
```

In [ ]:
```
train.info()
```

In [ ]:
```
import pandas as pd

# Assuming 'train' is your DataFrame containing sales data
# Assuming 'prior_year_sales' is your DataFrame containing prior year sales data by week and day

# Sort the DataFrame by store, date, and year
train = train.sort_values(by=['Store', 'Date', 'Year'])

# Identify missing rows
missing_rows = train[train['Sales'].isnull()]

# Iterate over missing rows
for index, row in missing_rows.iterrows():
    store = row['Store']
    week_number = row['WeekNumber']
    day_of_week = row['DayOfWeek']

    # Find corresponding row from prior year sales data
    prior_year_row = prior_year_sales[(prior_year_sales['Store'] == store) &
                                      (prior_year_sales['WeekNumber'] == week_number) &
                                      (prior_year_sales['DayOfWeek'] == day_of_week)]

    # If corresponding row exists, use its sales value as the estimate for the missing value
    if not prior_year_row.empty:
        estimated_sales = prior_year_row['Sales'].values[0]
        train.at[index, 'Sales'] = estimated_sales

# Display the created rows
print("Created rows after estimating missing values:")
created_rows = train[train['Sales'].isnull()]
print(created_rows)
```

```
# Now, 'train' DataFrame contains estimated sales values
```

In [ ]:
```
# Identify the missing dates for each store
missing_dates = train[train['Sales'].isnull()]['Date'].unique()

# Create a DataFrame to store the missing rows
missing_rows = pd.DataFrame(columns=train.columns)

# Iterate over each missing date
for date in missing_dates:
    # Iterate over each store
    for store in train['Store'].unique():
        # Find the prior year row for the same week and day of the week
        prior_year_row = prior_year_sales[(prior_year_sales['Store'] == store) & (prior_year_sales['Date'] == date)]

        # If prior year row exists, use its sales value as the estimate for the missing value
        if not prior_year_row.empty:
            # Add the row to the missing_rows DataFrame
            missing_rows = missing_rows.append(prior_year_row)

# Concatenate the missing_rows DataFrame with the original train DataFrame
train = pd.concat([train, missing_rows], ignore_index=True)

# Now, 'train' DataFrame contains the missing rows added with sales estimates using prior year data
```

In [ ]:
```
train.info()
```

In [ ]:
```
# Get the store IDs that do not have 942 DateIdentifier values
stores_not_942 = train["Store"].value_counts()[train["Store"].value_counts() != 942].index

# Create a DataFrame with one column containing the IDs of stores not equal to 942
stores_not_942_df = pd.DataFrame({'Store': stores_not_942})

# Print the first few rows of the stores_not_942_df DataFrame
print("DataFrame with store IDs not equal to 942:")
print(stores_not_942_df.head())
```

In [ ]:
```
print(stores_not_942_df.tail())
```

In [ ]:
```
stores_not_942_df.info()
```

In [ ]:
```
# Assuming 'train' is your DataFrame
train = train[train['Open'] != 0]

# Verify that rows with 'Open' equal to 0 are removed
print(train['Open'].value_counts())
```

In [ ]:
```
# Calculate the year based on the week number
train['Year'] = ((train['WeekNumber'] - 1) // 52) + 2013

# Calculate the week within each year
train['WeekInYear'] = (train['WeekNumber'] - 1) % 52 + 1

# Combine year and week number into a single column
train['YearWeek'] = train['Year'].astype(str) + '-' + train['WeekInYear'].astype(str)

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

# Group data by year and week, and calculate total sales per week
weekly_sales = train.groupby('YearWeek')['Sales'].sum()

for year in range(2013, 2016):
    year_data = weekly_sales[f'{year}-']
    plt.plot(year_data.index, year_data.values, label=f'Year {year}')

plt.title('Total Sales per Week by Year')
plt.xlabel('Year-Week')
plt.ylabel('Total Sales')
plt.grid(True)
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.legend()

plt.show()
```

In [ ]:
```
train.info()
```

In [ ]:
```
# Assuming your DataFrame is named df
average_sales_per_combination = l_nz.groupby(['StoreType', 'Assortment'])['Sales'].mean()

print(average_sales_per_combination)
```

In [ ]:
```
import matplotlib.pyplot as plt

# Assuming 'train' is your DataFrame

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

# Filter data for each year
year_1_data = train[train['Year'] == 1]
year_2_data = train[train['Year'] == 2]
year_3_data = train[train['Year'] == 3]

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

# Plot Year 1 data
plt.plot(year_1_data['WeekNumber'], year_1_data['Sales'], label='Year 1')

# Plot Year 2 data
plt.plot(year_2_data['WeekNumber'], year_2_data['Sales'], label='Year 2')

# Plot Year 3 data
plt.plot(year_3_data['WeekNumber'], year_3_data['Sales'], label='Year 3')

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

plt.show()
```

In [ ]:
```
import numpy as np

# Assuming 'train' is your DataFrame containing sales data
# Filter sales for 14 June 2014 (Saturday)
sales_2014 = train[(train['Date'] == '14/06/2014')['Sales']

# Filter sales for 13 June 2015 (Saturday)
sales_2015 = train[(train['Date'] == '13/06/2015')['Sales']

# Calculate RMSPE
def rmspe(y_true, y_pred):
```

```
    rmspe = np.sqrt(np.mean(((y_true - y_pred) / y_true)**2))
    return rmspe

rmspe_value = rmspe(sales_2014, sales_2015)

print("RMSPE between sales on 14 June 2014 and 13 June 2015 (Saturdays):", rmspe_value)
```

In [ ]:
```
# Group the data by the "StateHoliday" column and calculate the average sales
average_sales_per_DayOfWeek = train.groupby("DayOfWeek")["Sales"].mean()

print(average_sales_per_DayOfWeek)
```

In [ ]:
```
 train["DayOfWeek"].value_counts()
```

In [ ]:
```
# Group the data by the "StateHoliday" column and calculate the average sales
average_sales_per_stateholiday = train.groupby("StateHoliday")["Sales"].mean()

print(average_sales_per_stateholiday)
```

In [ ]:
```
# Filter the DataFrame for rows where Sales is zero and group by StateHoliday
zero_sales_per_store = train[train["Sales"] == 0].groupby("Store").size()

print(zero_sales_per_store)
```

In [ ]:
```
# Group the data by the "StateHoliday" column and calculate the average sales
average_sales_per_store = train.groupby("Store")["Sales"].mean()

print(average_sales_per_store)
```

In [ ]:
```
# Group the data by the "StateHoliday" column and calculate the average sales
average_sales_per_store = train.groupby("Store")["Sales"].describe()

print(average_sales_per_store)
```

In [ ]:
```
sns.histplot(train,x="Sales")
```

In [ ]:
```
sns.histplot(train,x="Sales")
```

In [ ]:
```
 train["StateHoliday"].value_counts()
```

In [ ]:
```
train["StateHoliday"].nunique()
```

In [ ]:
```
# Now you can retrieve the actual names of the unique values
unique_names = train["StateHoliday"].unique()

print(unique_names)
```

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

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

In [ ]:
```
# Group the data by the "StateHoliday" column and calculate the average sales
average_sales_per_stateholiday = train.groupby("StateHoliday")["Sales"].mean()

print(average_sales_per_stateholiday)
```

In [ ]:
```
# Group the data by the "StateHoliday" column and describe the statistics
statistics_per_stateholiday = train.groupby("StateHoliday")["Sales"].describe()

print(statistics_per_stateholiday)
```

In [ ]:
```
# Filter the DataFrame for rows where Sales is zero and group by StateHoliday
zero_sales_per_stateholiday = train[train["Sales"] == 0].groupby("StateHoliday").size()

print(zero_sales_per_stateholiday)
```

In [ ]:
```
# Define a mapping dictionary
holiday_mapping = {'0': 0, 0 : 0, 'a': 1, 'b': 2, 'c': 3}

# Map the values in the "StateHoliday" column using the mapping dictionary
train["StateHoliday"] = train["StateHoliday"].map(holiday_mapping)



# Convert the column to integer type
train["StateHoliday"] = train["StateHoliday"].astype(int)
train["StateHoliday"].value_counts()
```

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

In [ ]:
```
train["Store"].nunique()
```

In [ ]:
```
# Get the store IDs that do not have 942 DateIdentifier values
stores_not_942 = train["Store"].value_counts()[train["Store"].value_counts() != 942].index

# Create a DataFrame with one column containing the IDs of stores not equal to 942
stores_not_942_df = pd.DataFrame({'Store': stores_not_942})

# Print the first few rows of the stores_not_942_df DataFrame
print("DataFrame with store IDs not equal to 942:")
print(stores_not_942_df.head())
print("DataFrame with store IDs not equal to 942:")
print(stores_not_942_df.tail())
```

In [ ]:
```
# Convert the "Date" column to datetime if it's not already
test["Date"] = pd.to_datetime(test["Date"])

# Filter the DataFrame for rows where the date is July 31, 2015
stores_open_on_31_July_2015 = test[test["Date"] == "31/07/2015"]

# Count the unique values in the "Store" column for that date
intersection_number = stores_open_on_31_July_2015["Store"].nunique()

print("Number of stores open on July 31, 2015:", intersection_number)
```

In [ ]:
```
# Adjust StateHoliday variable to integer
# Define a mapping dictionary
holiday_mapping = {'0': 0, 0 : 0,'a': 1, 'b': 2, 'c': 3}

# Map the values in the "StateHoliday" column using the mapping dictionary
train["StateHoliday"] = train["StateHoliday"].map(holiday_mapping)

# Convert the column to integer type
train["StateHoliday"] = train["StateHoliday"].astype(int)
```

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

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

In [ ]:
```
# Df has 195 rows and 35 columns(cols)
# Continue review of df
# Display df's first 5 rows
test.head(860)
```

In [ ]:
```
# Display df's first 5 rows
test.tail(20)
```

In [ ]:
```
# Display df's first 5 rows
test.info()
```

In [ ]:
```
# Filter rows with null values in the 'Open' column
rows_with_null_open = test[test['Open'].isnull()]


# Display the filtered rows
print(rows_with_null_open)
```

In [ ]:
```
test["DayOfWeek"].value_counts()
```

In [ ]:
```

```

In [ ]:
```
test["Store"].value_counts()
```

In [ ]:
```

```

In [ ]:
```
# Get the value counts of the "Store" column
store_counts = test["Store"].value_counts()

# Find the IDs of stores that don't have the maximum count
ids_not_48 = store_counts[store_counts != 48].index

print(ids_not_48)
```

In [ ]:
```
test["Store"].nunique()
```

In [ ]:
```
test["StateHoliday"].value_counts()
```

In [ ]:
```
test["Open"].value_counts()
```

In [ ]:
```
unique_names = test["Open"].unique()

print(unique_names)
```

In [ ]:
```
closed_days = test[test['Open'] == 0.0]

# Now, you can use value_counts to see the split by day of the week
closed_days_by_day = closed_days['DayOfWeek'].value_counts()

print(closed_days_by_day)
```

In [ ]:
```
closed_days = test[test['Open'] == 0.0]

# Now, you can use value_counts to see the split by day of the week
closed_days_by_day = closed_days['Date'].value_counts()

print(closed_days_by_day)
```

In [ ]:
```
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
```

In [ ]:
```
# Merge the train and stores datasets on the "Store" column
merged_train_stores = pd.merge(train, store, on="Store", how="inner")

# Merge the test and stores datasets on the "Store" column
merged_test_stores = pd.merge(test, store, on="Store", how="inner")

# Merge the resulting datasets on the "Store" column
final_dataset = pd.merge(merged_train_stores, merged_test_stores, on="Store", how="inner")

# Display the final dataset
print(final_dataset)
```

In [ ]:
```
closed_days = test[test['Open'] == 0.0]

# Now, you can use value_counts to see the split by day of the week
closed_days_by_day = closed_days['StateHoliday'].value_counts()

print(closed_days_by_day)
```

In [ ]:
```
# Display the final dataset
final_dataset.info
```

In [ ]:
```
final_dataset["Store"].nunique()
```

In [ ]:
```
# Assuming your DataFrame is named df
average_sales_per_combination = final_dataset.groupby(['StoreType_x', 'Assortment_x'])['Sales_x'].mean()

print(average_sales_per_combination)
```

In [ ]:
```
final_dataset.describe().
```

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming df is your DataFrame
final_dataset.drop(columns=['Date_x', 'Date_y'], inplace=True)
```

```
# Compute the correlation matrix
correlation_matrix = final_dataset.corr()
```

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

In [ ]:
```
# Assuming your DataFrame is named df
average_sales_per_combination = final_dataset.groupby(['Store', 'Assortment'])['Sales'].mean()

print(average_sales_per_combination)
```