Retail Analysis with walmart data

DESCRIPTION

One of the leading retail stores in the US, Walmart, would like to predict the sales and demand accurately. There are certain events and holidays which impact sales on each day. There are sales data available for 45 stores of Walmart. The business is facing a challenge due to unforeseen demands and runs out of stock some times, due to the inappropriate machine learning algorithm. An ideal ML algorithm will predict demand accurately and ingest factors like economic conditions including CPI, Unemployment Index, etc.

Walmart runs several promotional markdown events throughout the year. These markdowns precede prominent holidays, the four largest of all, which are the Super Bowl, Labour Day, Thanksgiving, and Christmas. The weeks including these holidays are weighted five times higher in the evaluation than non-holiday weeks. Part of the challenge presented by this competition is modeling the effects of markdowns on these holiday weeks in the absence of complete/ideal historical data. Historical sales data for 45 Walmart stores located in different regions are available.

Dataset Description

This is the historical data that covers sales from 2010-02-05 to 2012-11-01, in the file Walmart_Store_sales. Within this file you will find the following fields:

Store - the store number

Date - the week of sales

Weekly_Sales - sales for the given store

Holiday_Flag - whether the week is a special holiday week 1 – Holiday week 0 – Non-holiday week

Temperature - Temperature on the day of sale

Fuel_Price - Cost of fuel in the region

CPI – Prevailing consumer price index

Unemployment - Prevailing unemployment rate

Holiday Events

Super Bowl: 12-Feb-10, 11-Feb-11, 10-Feb-12, 8-Feb-13 Labour Day: 10-Sep-10, 9-Sep-11, 7-Sep-12, 6-Sep-13 Thanksgiving: 26-Nov-10, 25-Nov-11, 23-Nov-12, 29-Nov-13 Christmas: 31-Dec-10, 30-Dec-11, 28-Dec-12, 27-Dec-13

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
from datetime import datetime
```

```python
from matplotlib import dates
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [2]:
```python
data = pd.read_csv('walmart-store-sales/Walmart_Store_sales.csv')
data.head()
```

Out[2]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
0	1	05-02-2010	1643690.90	0	42.31	2.572	211.096358	8.106
1	1	12-02-2010	1641957.44	1	38.51	2.548	211.242170	8.106
2	1	19-02-2010	1611968.17	0	39.93	2.514	211.289143	8.106
3	1	26-02-2010	1409727.59	0	46.63	2.561	211.319643	8.106
4	1	05-03-2010	1554806.68	0	46.50	2.625	211.350143	8.106

In [3]:
```python
data.shape
```

Out[3]: (6435, 8)

In [4]:
```python
data.describe().T
```

Out[4]:

	count	mean	std	min	25%	50%	
Store	6435.0	2.300000e+01	12.988182	1.000	12.000	23.000000	3.40
Weekly_Sales	6435.0	1.046965e+06	564366.622054	209986.250	553350.105	960746.040000	1.42
Holiday_Flag	6435.0	6.993007e-02	0.255049	0.000	0.000	0.000000	0.00
Temperature	6435.0	6.066378e+01	18.444933	-2.060	47.460	62.670000	7.49
Fuel_Price	6435.0	3.358607e+00	0.459020	2.472	2.933	3.445000	3.73
CPI	6435.0	1.715784e+02	39.356712	126.064	131.735	182.616521	2.12
Unemployment	6435.0	7.999151e+00	1.875885	3.879	6.891	7.874000	8.62

In [5]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6435 entries, 0 to 6434
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Store         6435 non-null   int64  
 1   Date          6435 non-null   object 
 2   Weekly_Sales  6435 non-null   float64
 3   Holiday_Flag  6435 non-null   int64  
 4   Temperature   6435 non-null   float64
 5   Fuel_Price    6435 non-null   float64
 6   CPI           6435 non-null   float64
 7   Unemployment  6435 non-null   float64
dtypes: float64(5), int64(2), object(1)
memory usage: 402.3+ KB
```

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

Out[6]:
```
Store             int64
Date             object
Weekly_Sales    float64
Holiday_Flag      int64
Temperature     float64
Fuel_Price      float64
CPI             float64
Unemployment    float64
dtype: object
```

In [7]:
```
data['Date'] = pd.to_datetime(data['Date'])
print(data['Date'].dtype)
```

```
datetime64[ns]
```

In [8]:
```
print(data['Date'][:1])
print(pd.DatetimeIndex(data['Date'][:1]).day)
print(pd.DatetimeIndex(data['Date'][:1]).month)
print(pd.DatetimeIndex(data['Date'][:1]).year)
```

```
0   2010-05-02
Name: Date, dtype: datetime64[ns]
Int64Index([2], dtype='int64', name='Date')
Int64Index([5], dtype='int64', name='Date')
Int64Index([2010], dtype='int64', name='Date')
```

In [9]:
```
# Creating new columns for day, month and year
data['Day'] = pd.DatetimeIndex(data['Date']).day
data['Month'] = pd.DatetimeIndex(data['Date']).month
data['Year'] = pd.DatetimeIndex(data['Date']).year
data.sample(5)
```

Out[9]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemploym
500	4	2011-06-17	2141210.62	0	83.51	3.618	129.043200	5
4814	34	2011-02-12	988742.08	0	39.75	3.176	129.845967	10
6030	43	2010-07-23	649035.55	0	86.60	2.608	203.175016	9
2797	20	2011-08-19	1990017.93	0	71.25	3.747	208.842436	7
3011	22	2010-02-04	1177340.99	0	44.96	2.826	135.746499	8

Basic statistical tasks

Which store has maximum sales

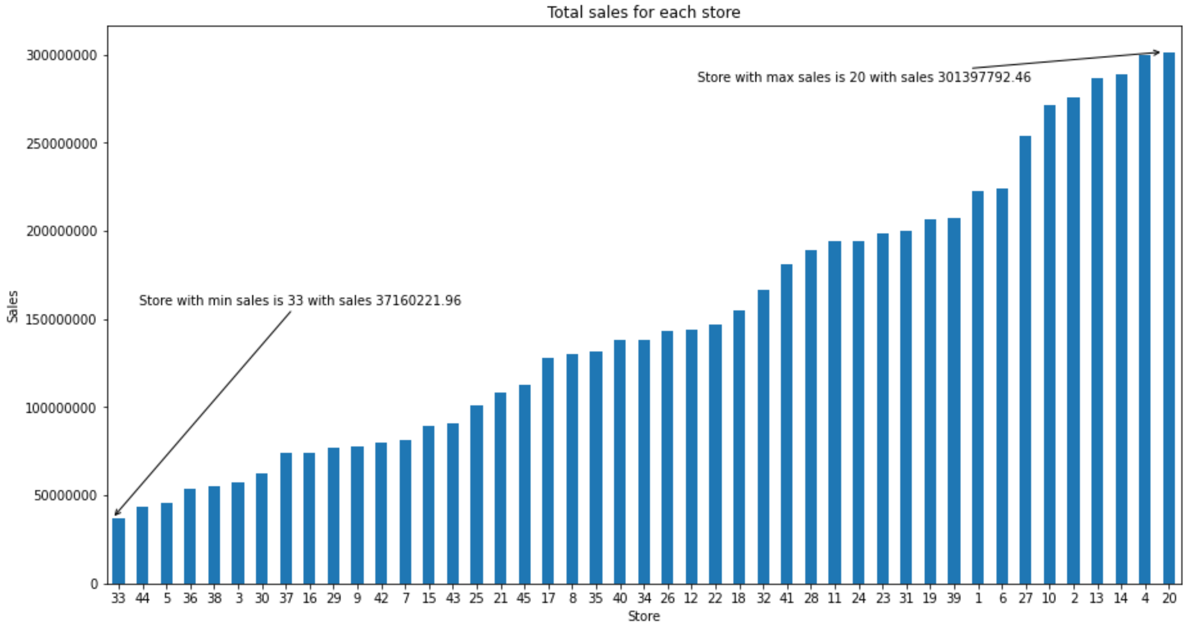
In [10]:

```
# Summing the store sales using group by function
totalsales_eachstore = data.groupby('Store')['Weekly_Sales'].sum().sort_values()
print("Store with minimum sales\n", totalsales_eachstore.head(1))
print("\nStore with maximum sales\n",totalsales_eachstore.tail(1))
```

```
Store with minimum sales
 Store
33    37160221.96
Name: Weekly_Sales, dtype: float64

Store with maximum sales
 Store
20    3.013978e+08
Name: Weekly_Sales, dtype: float64
```

In [11]:

```
plt.figure(figsize=(15,8))
plt.title('Total sales for each store')
ax = totalsales_eachstore.plot(kind='bar')
p = ax.patches[0]
plt.annotate(f"Store with min sales is 33 with sales {p.get_height()}",
             xy=(p.get_x(), p.get_height()),
             xytext=(0.03,0.5),
             textcoords = 'axes fraction',
             arrowprops=dict(arrowstyle='->', connectionstyle='arc3'),
            )
p = ax.patches[44]
plt.annotate(f"Store with max sales is 20 with sales {p.get_height()}",
             xy=(p.get_x(), p.get_height()),
             xytext=(0.55,0.9),
             textcoords = 'axes fraction',
             arrowprops=dict(arrowstyle='->', connectionstyle='arc3'),
            )
plt.xlabel('Store')
plt.ylabel('Sales')
plt.xticks(rotation=0)
plt.ticklabel_format(useOffset=False, style='plain', axis='y')
```

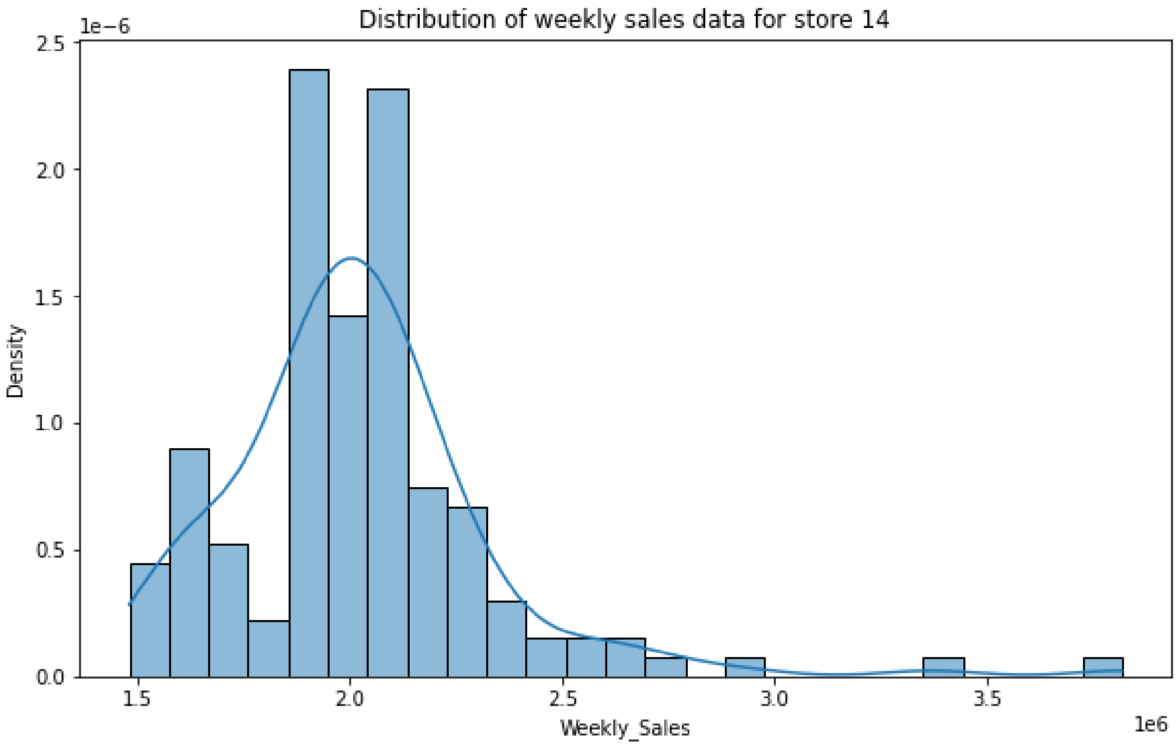


Which store has maximum standard deviation

i.e., Where the sales vary a lot. Also find out the mean to standard deviation

In [12]:
```
data_std = pd.DataFrame(data.groupby('Store')['Weekly_Sales'].std().sort_values(as
max_std_score = data_std.head(1)
print(f'Store with maximum standard deviation is {max_std_score.index[0]} with sal
# Distribution
plt.figure(figsize=(10,6))
plt.title(f"Distribution of weekly sales data for store {max_std_score.index[0]}")
sns.histplot(data[data['Store'] == max_std_score.index[0]]['Weekly_Sales'], kde=Tr
```

Out[12]:
```
Store with maximum standard deviation is 14 with sales 317569.9
<AxesSubplot:title={'center':'Distribution of weekly sales data for store 14'}, xl
abel='Weekly_Sales', ylabel='Density'>
```



In [13]:
```
# Coefficient of mean to standard deviation
mean_to_std = pd.DataFrame(data.groupby('Store')['Weekly_Sales'].std()/data.groupb
mean_to_std.head()
```

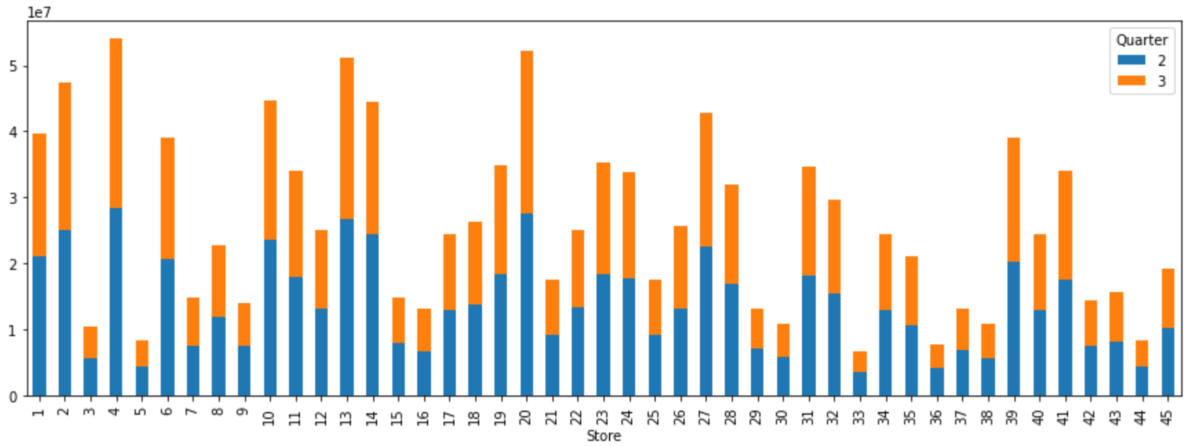
Out[13]:

	Weekly_Sales
Store	
1	0.100292
2	0.123424
3	0.115021
4	0.127083
5	0.118668

Which store has good quarterly growth rate in Q3'2012

In [14]:
```
data['Quarter'] = data['Date'].dt.quarter
data_2012 = data.query('Year==2012')
data_2012_qs = data_2012.groupby(['Store', 'Quarter'])['Weekly_Sales'].sum().unsta
data_2012_qs[[2,3]].sort_index().plot(kind='bar', stacked=True, figsize=(15,5))
```

Out[14]: <AxesSubplot:xlabel='Store'>



In [15]:
```
# Growth rate
growth = (data_2012_qs[3] - data_2012_qs[2])
data_2012_qs['Growth Rate'] = (growth/ data_2012_qs[2]) * 100
data_2012_qs.sort_values(by='Growth Rate',ascending=False)['Growth Rate'].head()
```

Out[15]:
```
Store
16   -2.789294
7    -3.824738
35   -4.663086
26   -6.057624
39   -6.396875
Name: Growth Rate, dtype: float64
```

Growth rates of stores for Quarter 4 compared to Quarter 3

Holidays which have higher sales than the mean sales in non holiday season for all stores

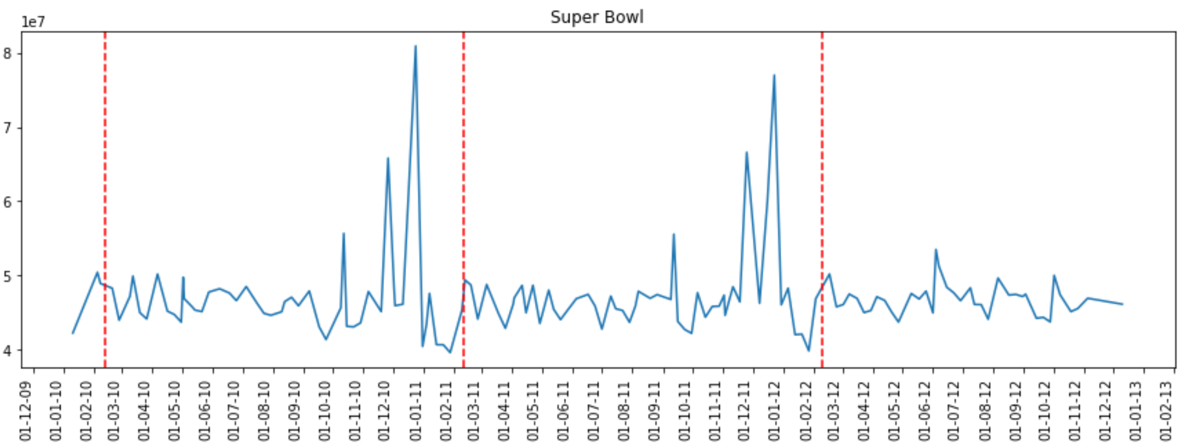
Given holiday events

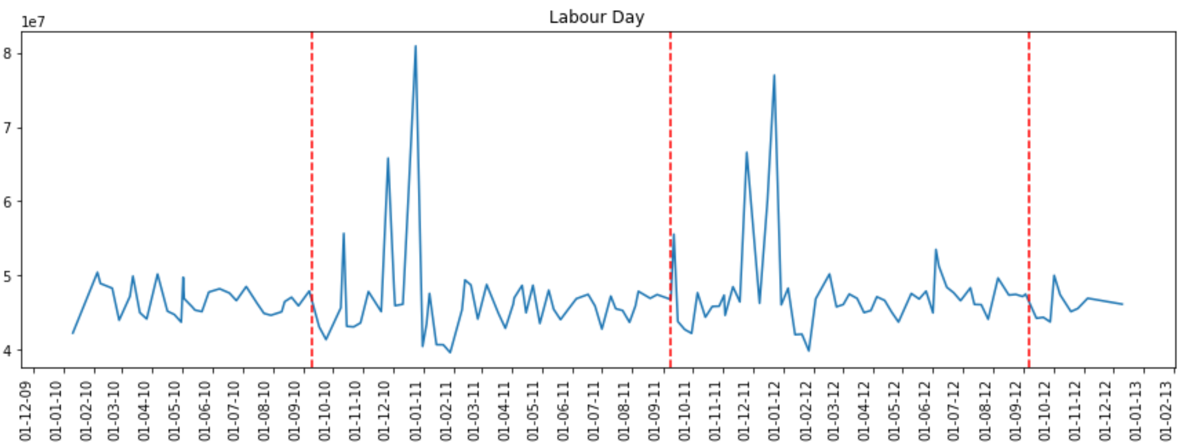
- Super Bowl: 12-Feb-10, 11-Feb-11, 10-Feb-12, 8-Feb-13
- Labour Day: 10-Sep-10, 9-Sep-11, 7-Sep-12, 6-Sep-13
- Thanksgiving: 26-Nov-10, 25-Nov-11, 23-Nov-12, 29-Nov-13

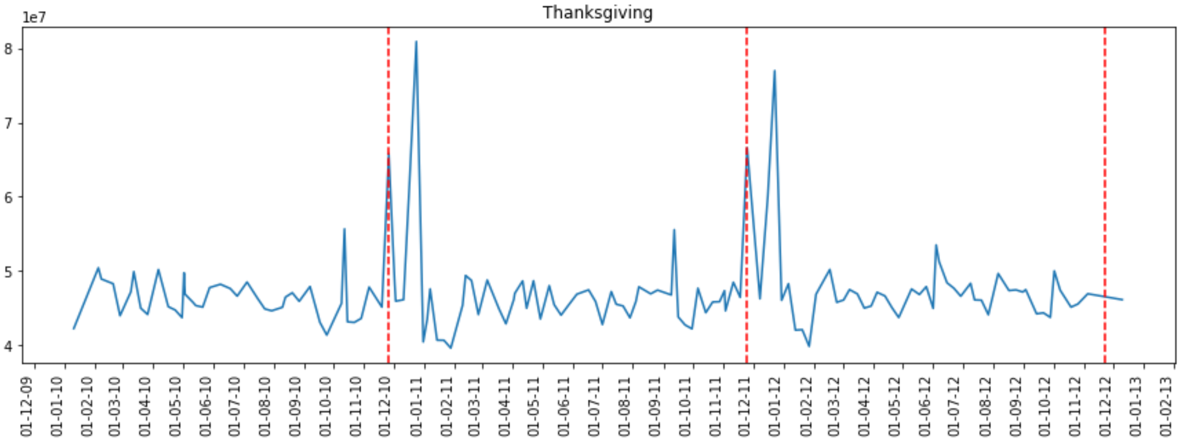
- Christmas: 31-Dec-10, 30-Dec-11, 28-Dec-12, 27-Dec-13

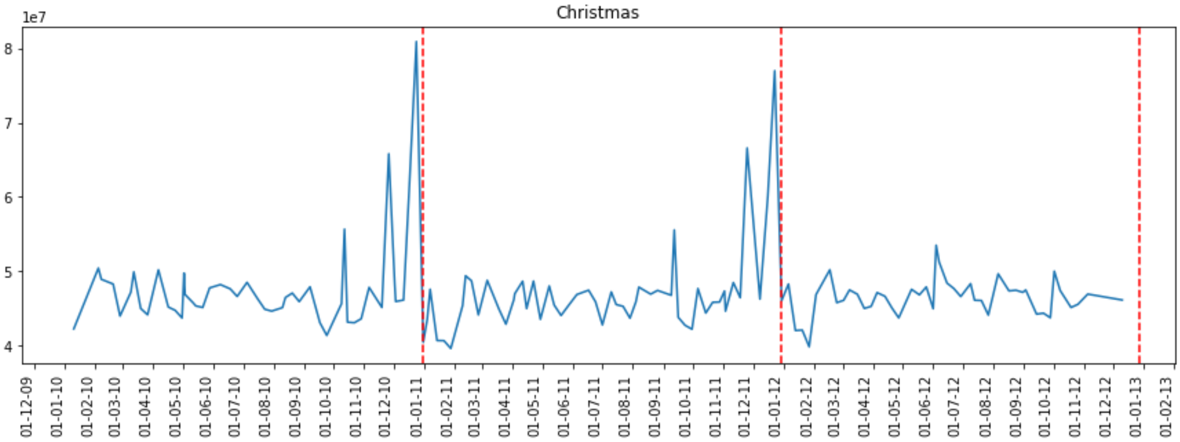
In [16]:

```
total_sales = data.groupby('Date')['Weekly_Sales'].sum().reset_index()
Super_Bowl =['12-2-2010', '11-2-2011', '10-2-2012']
Labour_Day =  ['10-9-2010', '9-9-2011', '7-9-2012']
Thanksgiving =  ['26-11-2010', '25-11-2011', '23-11-2012']
Christmas = ['31-12-2010', '30-12-2011', '28-12-2012']

def plot_data(data, holiday, holiday_dates):
    fig, ax = plt.subplots(figsize=(15,5))
    ax.plot(data['Date'], data['Weekly_Sales'],label=holiday)
    for day in holiday_dates:
        day = datetime.strptime(day, '%d-%m-%Y')
        plt.axvline(x=day, linestyle='--', c='r')

    plt.title(holiday)
    x_dates = data['Date'].dt.strftime('%Y-%m-%d').sort_values().unique()
    xfmt = dates.DateFormatter('%d-%m-%y')
    ax.xaxis.set_major_formatter(xfmt)
    ax.xaxis.set_major_locator(dates.DayLocator(1))
    plt.gcf().autofmt_xdate(rotation=90)
    plt.show()

plot_data(total_sales, 'Super Bowl', Super_Bowl)
plot_data(total_sales, 'Labour Day', Labour_Day)
plot_data(total_sales, 'Thanksgiving', Thanksgiving)
plot_data(total_sales, 'Christmas', Christmas)
```





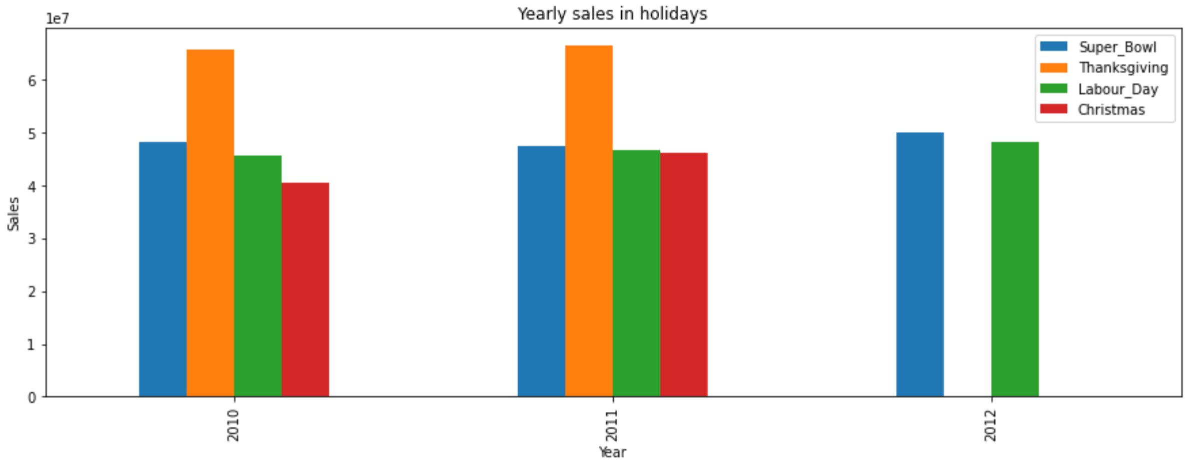




The sales are increased in *Thanksgiving* holidays and decreased in *Christmas* holidays

In [17]:
```
holiday_sales = pd.DataFrame()
holiday_sales['Super_Bowl'] = data[data['Date'].isin(Super_Bowl)].groupby('Year')[
holiday_sales['Thanksgiving'] = data[data['Date'].isin(Thanksgiving)].groupby('Year
holiday_sales['Labour_Day'] = data[data['Date'].isin(Labour_Day)].groupby('Year')[
holiday_sales['Christmas'] = data[data['Date'].isin(Christmas)].groupby('Year')['W

holiday_sales.fillna(0, inplace=True)
holiday_sales.plot(kind='bar', figsize=(15,5), xlabel='Year', ylabel='Sales', titl
```

Out[17]:
```
<AxesSubplot:title={'center':'Yearly sales in holidays'}, xlabel='Year', ylabel='S
ales'>
```

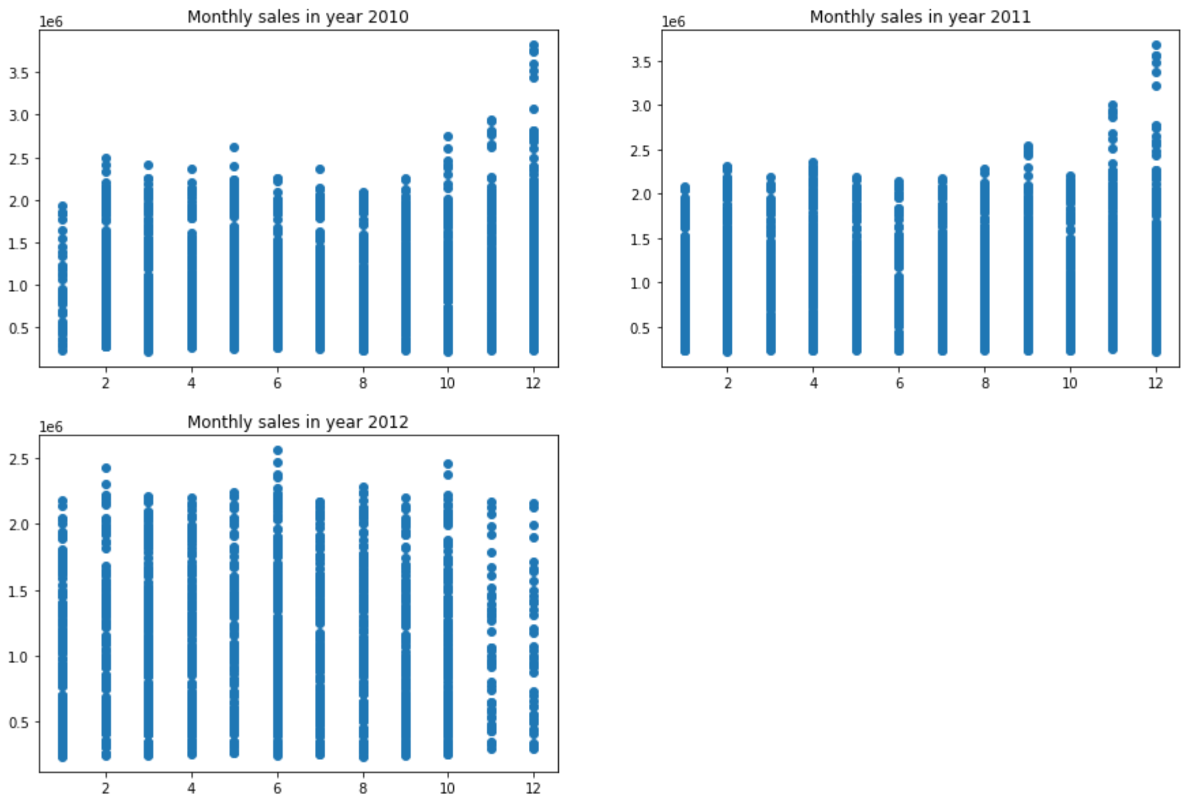


Monthly and semester view of sales in units and insights

In [18]:
```
plt.figure(figsize=(15,10))
plt.subplot(221)
plt.title("Monthly sales in year 2010")
```

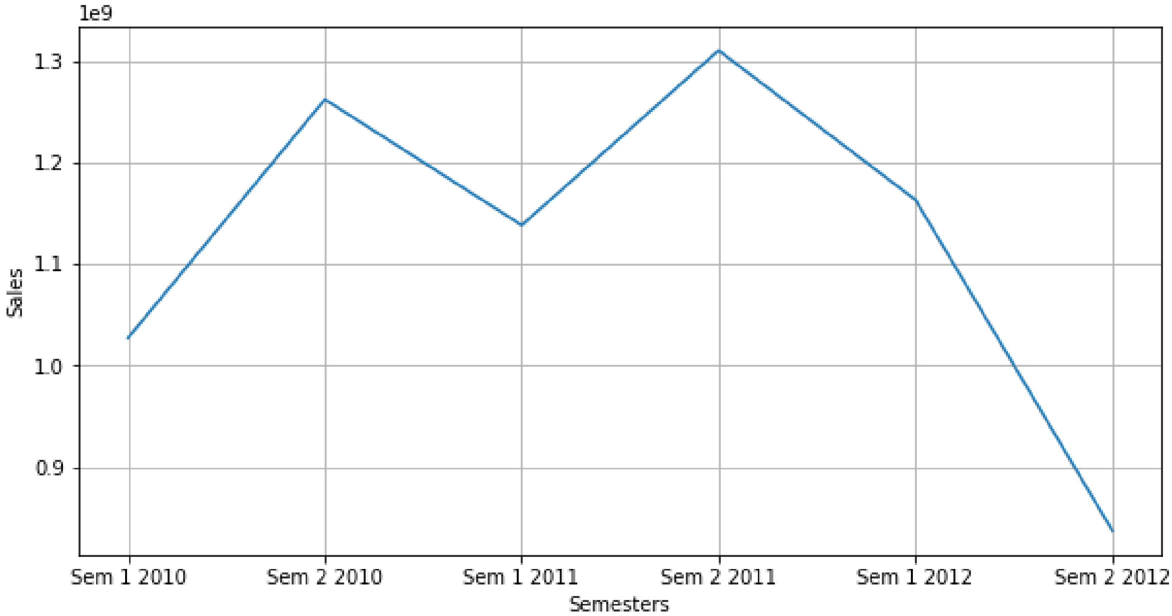
```python
plt.scatter(data[data.Year == 2010]["Month"], data[data.Year == 2010]["Weekly_Sale

plt.subplot(222)
plt.title("Monthly sales in year 2011")
plt.scatter(data[data.Year == 2011]["Month"], data[data.Year == 2011]["Weekly_Sale

plt.subplot(223)
plt.title("Monthly sales in year 2012")
plt.scatter(data[data.Year == 2012]["Month"], data[data.Year == 2012]["Weekly_Sale
```

Out[18]: <matplotlib.collections.PathCollection at 0x22035813100>



In [19]:

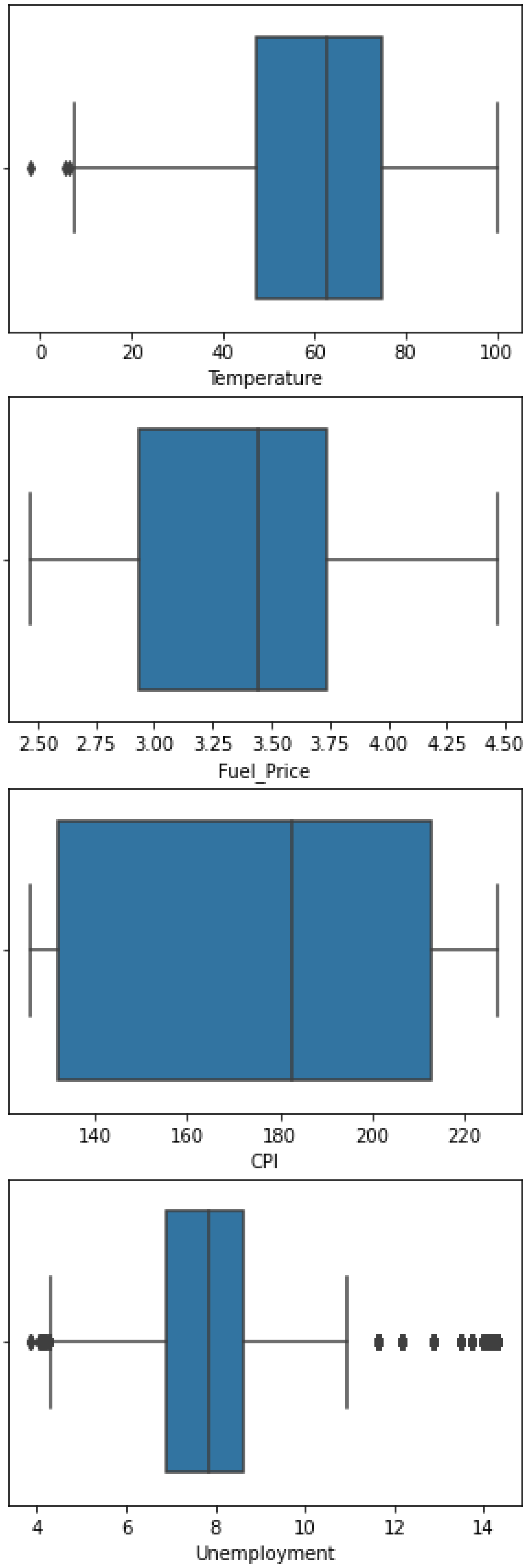
```python
# Semester view of sales
semester_labels = ['Sem 1 2010', 'Sem 2 2010', 'Sem 1 2011', 'Sem 2 2011', 'Sem 1 
semester_sales = []
semester_sales.append(data[data['Year']==2010].loc[data['Month']<7, ['Weekly_Sales
semester_sales.append(data[data['Year']==2010].loc[data['Month']>6, ['Weekly_Sales
semester_sales.append(data[data['Year']==2011].loc[data['Month']<7, ['Weekly_Sales
semester_sales.append(data[data['Year']==2011].loc[data['Month']>6, ['Weekly_Sales
semester_sales.append(data[data['Year']==2012].loc[data['Month']<7, ['Weekly_Sales
semester_sales.append(data[data['Year']==2012].loc[data['Month']>6, ['Weekly_Sales

plt.figure(figsize=(10,5))
plt.plot(semester_labels,semester_sales)
plt.xlabel('Semesters')
plt.ylabel('Sales')
plt.grid()
```



In [20]:
```
data.columns
```

Out[20]:
```
Index(['Store', 'Date', 'Weekly_Sales', 'Holiday_Flag', 'Temperature',
       'Fuel_Price', 'CPI', 'Unemployment', 'Day', 'Month', 'Year', 'Quarter'],
      dtype='object')
```

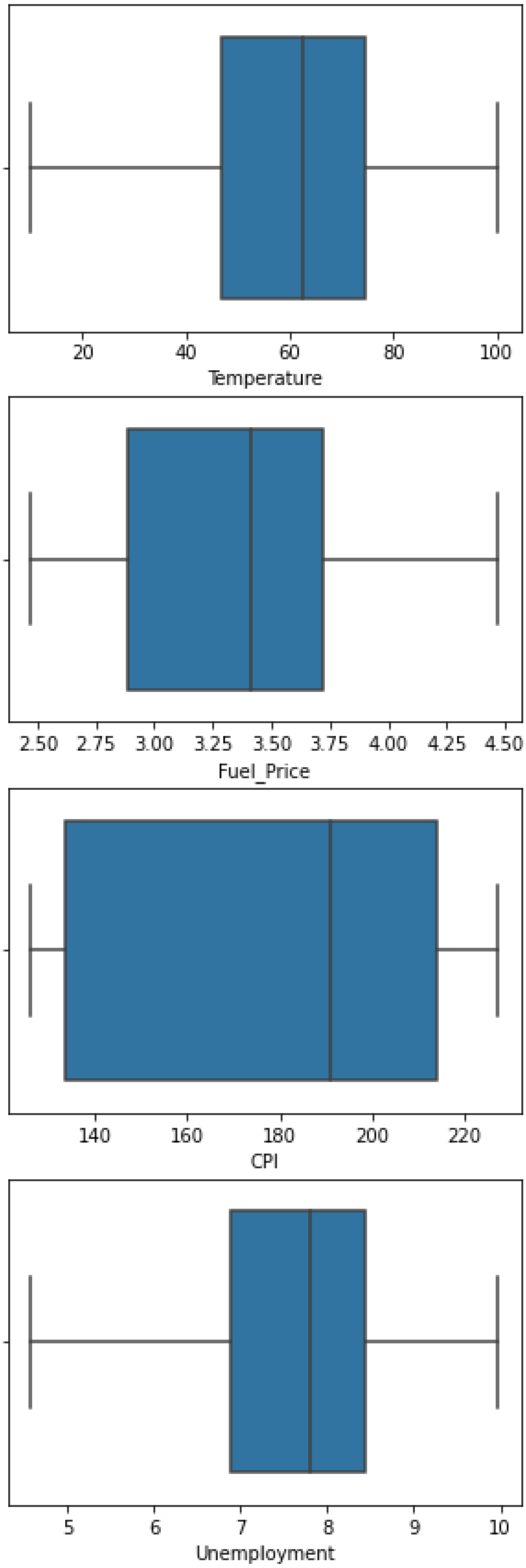
In [21]:
```
# Checking for outliers in the data
fig, axs = plt.subplots(4, figsize=(5,15))
X = data[['Temperature', 'Fuel_Price', 'CPI', 'Unemployment']]
for i, column in enumerate(X):
    sns.boxplot(x=data[column], ax=axs[i])
```



Temperature and Unemployment columns have outliers as seen in boxplot

In [22]:
```
# Removing outliers from the data
data_cleaned = data[(data["Temperature"] > 10) & (data["Unemployment"] > 4.5) & (da

# Checking if outliers are removed from the data
fig, axs = plt.subplots(4, figsize=(5,15))
X = data_cleaned[['Temperature', 'Fuel_Price', 'CPI', 'Unemployment']]
```

```
for i, column in enumerate(X):
    sns.boxplot(x=data_cleaned[column], ax=axs[i])
```



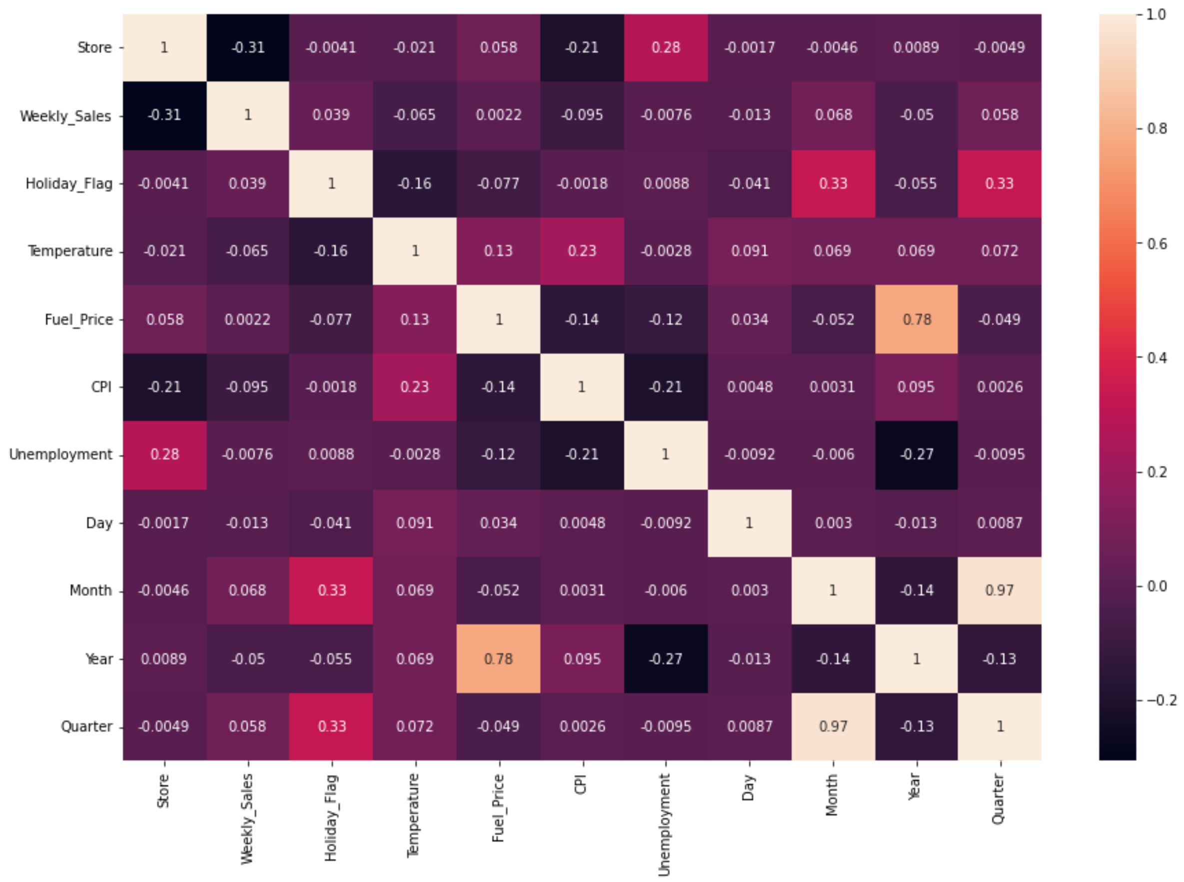
Statistical model

For Store 1 - Build prediction model to forecast demand

In [23]:
```
plt.figure(figsize=(15,10))
```

```python
sns.heatmap(data_cleaned.corr(), annot=True)
```

Out[23]: <AxesSubplot:>



Fuel price and year are more colinear

In [24]:
```python
# Dropping the more colinear values i.e., Fuel Price and Year
data_cleaned = data_cleaned.drop(['Year', 'Fuel_Price', 'Quarter', 'Date'], axis=1
```

In [25]:
```python
data_cleaned
```

Out[25]:

	Store	Weekly_Sales	Holiday_Flag	Temperature	CPI	Unemployment	Day	Month
0	1	1643690.90	0	42.31	211.096358	8.106	2	5
1	1	1641957.44	1	38.51	211.242170	8.106	2	12
2	1	1611968.17	0	39.93	211.289143	8.106	19	2
3	1	1409727.59	0	46.63	211.319643	8.106	26	2
4	1	1554806.68	0	46.50	211.350143	8.106	3	5
...	...	...	...	...	...	...	...	...
6430	45	713173.95	0	64.88	192.013558	8.684	28	9
6431	45	733455.07	0	64.89	192.170412	8.667	10	5
6432	45	734464.36	0	54.47	192.327265	8.667	10	12
6433	45	718125.53	0	56.47	192.330854	8.667	19	10
6434	45	760281.43	0	58.85	192.308899	8.667	26	10

5658 rows × 8 columns

In [26]:
```
store1_data = data_cleaned[data_cleaned['Store']==1]
store1_data.sample(5)
```

Out[26]:

	Store	Weekly_Sales	Holiday_Flag	Temperature	CPI	Unemployment	Day	Month
96	1	1799682.38	0	43.93	218.961846	7.866	12	9
106	1	1819870.00	0	45.32	220.425759	7.348	17	2
78	1	1624383.75	0	91.65	215.544618	7.962	8	5
48	1	1444732.28	0	48.27	211.404742	7.742	1	7
105	1	1802477.43	1	48.02	220.265178	7.348	2	10

In [27]:
```
X = store1_data.drop(['Weekly_Sales', 'Store'], axis=1)
y = store1_data['Weekly_Sales']

# Splitting data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Standardizing the data
sc = StandardScaler()
X_train_sc = sc.fit_transform(X_train)
X_test_sc = sc.fit_transform(X_test)
```

In [28]:
```
# Using linear regression
lr_model = LinearRegression()
lr_model.fit(X_train_sc, y_train)
y_pred = lr_model.predict(X_test_sc)
print('Accuracy:', lr_model.score(X_train_sc, y_train))
print('R2 Score:', metrics.r2_score(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Accuracy: 0.22248341296773333
R2 Score: 0.3640470647313505
RMSE: 102554.76508765195
```

In [29]:
```
# Using Random forest regressor
rf_model = RandomForestRegressor()
rf_model.fit(X_train_sc, y_train)
y_pred = rf_model.predict(X_test_sc)
print('Accuracy:', rf_model.score(X_train_sc, y_train))
print('R2 Score:', metrics.r2_score(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Accuracy: 0.9197775032904872
R2 Score: 0.6142573078813662
RMSE: 79871.56786402348
```

Random forest regressor has more accuracy than linear regression