Retail Analysis with Walmart data_Jignesh Kataria

March 27, 2022

0.1 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.

0.2 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

0.3 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

0.4 Problem Statements/Analysis tasks:

0.5 Basic Statistics tasks

- Which store has maximum sales?

- Which store has maximum standard deviation i.e., the sales vary a lot. Also, find out the coefficient of mean to standard deviation?

- Which store/s has good quarterly growth rate in Q3'2012?

- Some holidays have a negative impact on sales. Find out holidays which have higher sales than the mean sales in non-holiday season for all stores together

- Provide a monthly and semester view of sales in units and give insights

0.6 Model Creation:

For Store 1 – Build prediction models to forecast demand

- Linear Regression – Utilize variables like date and restructure dates as 1 for 5 Feb 2010 (starting from the earliest date in order). Hypothesize if CPI, unemployment, and fuel price have any impact on sales.

- Change dates into days by creating new variable.

Select the model which gives best accuracy.

```python
[24]:  # Import necessary libraries
       import pandas as pd
       import seaborn as sns
       import numpy as np
       import matplotlib.pyplot as plt
       from matplotlib import dates
       from datetime import datetime
```

```
[25]: # Load dataset
      data = pd.read_csv(r'C:\Users\admin\Desktop\Data Scientist\Practice and
       →Projects\Walmart Retail Analysis\Walmart_Store_sales.csv')
      data
```

```
[25]:       Store        Date  Weekly_Sales  Holiday_Flag  Temperature  Fuel_Price  \
      0         1  05-02-2010    1643690.90             0        42.31       2.572
      1         1  12-02-2010    1641957.44             1        38.51       2.548
      2         1  19-02-2010    1611968.17             0        39.93       2.514
      3         1  26-02-2010    1409727.59             0        46.63       2.561
      4         1  05-03-2010    1554806.68             0        46.50       2.625
      ...     ...         ...           ...           ...          ...         ...
      6430     45  28-09-2012     713173.95             0        64.88       3.997
      6431     45  05-10-2012     733455.07             0        64.89       3.985
      6432     45  12-10-2012     734464.36             0        54.47       4.000
      6433     45  19-10-2012     718125.53             0        56.47       3.969
      6434     45  26-10-2012     760281.43             0        58.85       3.882

                   CPI  Unemployment
      0     211.096358         8.106
      1     211.242170         8.106
      2     211.289143         8.106
      3     211.319643         8.106
      4     211.350143         8.106
      ...          ...           ...
      6430  192.013558         8.684
      6431  192.170412         8.667
      6432  192.327265         8.667
      6433  192.330854         8.667
      6434  192.308899         8.667

      [6435 rows x 8 columns]
```

1 Data Preparation/Formatting/Cleaning

```
[26]: # Convert date to datetime format and show dataset information
      data['Date'] = pd.to_datetime(data['Date'])
      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   datetime64[ns]
 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: datetime64[ns](1), float64(5), int64(2)
memory usage: 402.3 KB
```

[27]:
```python
# checking for missing values
data.isnull().sum()
```

[27]:
```
Store           0
Date            0
Weekly_Sales    0
Holiday_Flag    0
Temperature     0
Fuel_Price      0
CPI             0
Unemployment    0
dtype: int64
```

[28]:
```python
# Splitting Date and create new columns (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
```

[28]:
```
      Store       Date  Weekly_Sales  Holiday_Flag  Temperature  Fuel_Price  \
0         1 2010-05-02    1643690.90             0        42.31       2.572
1         1 2010-12-02    1641957.44             1        38.51       2.548
2         1 2010-02-19    1611968.17             0        39.93       2.514
3         1 2010-02-26    1409727.59             0        46.63       2.561
4         1 2010-05-03    1554806.68             0        46.50       2.625
...     ...        ...           ...           ...          ...         ...
6430     45 2012-09-28     713173.95             0        64.88       3.997
6431     45 2012-05-10     733455.07             0        64.89       3.985
6432     45 2012-12-10     734464.36             0        54.47       4.000
6433     45 2012-10-19     718125.53             0        56.47       3.969
6434     45 2012-10-26     760281.43             0        58.85       3.882

             CPI  Unemployment  Day  Month  Year
0     211.096358         8.106    2      5  2010
1     211.242170         8.106    2     12  2010
2     211.289143         8.106   19      2  2010
3     211.319643         8.106   26      2  2010
4     211.350143         8.106    3      5  2010
...          ...           ...  ...    ...   ...
6430  192.013558         8.684   28      9  2012
```

```
6431   192.170412                8.667   10       5   2012
6432   192.327265                8.667   10      12   2012
6433   192.330854                8.667   19      10   2012
6434   192.308899                8.667   26      10   2012

[6435 rows x 11 columns]
```

1.1 Task 1 : Which store has maximum sales?

```python
[29]: plt.figure(figsize=(15,7))

      # Sum Weekly_Sales for each store, then sortded by total sales
      total_sales_for_each_store = data.groupby('Store')['Weekly_Sales'].sum().
       ↪sort_values()
      total_sales_for_each_store_array = np.array(total_sales_for_each_store) #␣
       ↪convert to array

      # Assigning a specific color for the stores have the lowest and highest sales
      clrs = ['lightsteelblue' if ((x < max(total_sales_for_each_store_array)) and (x␣
       ↪> min(total_sales_for_each_store_array))) else 'midnightblue' for x in␣
       ↪total_sales_for_each_store_array]


      ax = total_sales_for_each_store.plot(kind='bar',color=clrs);

      # store have minimum sales
      p = ax.patches[0]
      print(type(p.get_height()))
      ax.annotate("The store has minimum sales is 33 with {0:.2f} $".format((p.
       ↪get_height())), xy=(p.get_x(), p.get_height()), xycoords='data',
                  xytext=(0.17, 0.32), textcoords='axes fraction',
                  arrowprops=dict(arrowstyle="->", connectionstyle="arc3"),
                  horizontalalignment='center', verticalalignment='center')


      # store have maximum sales
      p = ax.patches[44]
      ax.annotate("The store has maximum sales is 20 with {0:.2f} $".format((p.
       ↪get_height())), xy=(p.get_x(), p.get_height()), xycoords='data',
                  xytext=(0.82, 0.98), textcoords='axes fraction',
                  arrowprops=dict(arrowstyle="->", connectionstyle="arc3"),
                  horizontalalignment='center', verticalalignment='center')


      # plot properties
      plt.xticks(rotation=0)
      plt.ticklabel_format(useOffset=False, style='plain', axis='y')
```
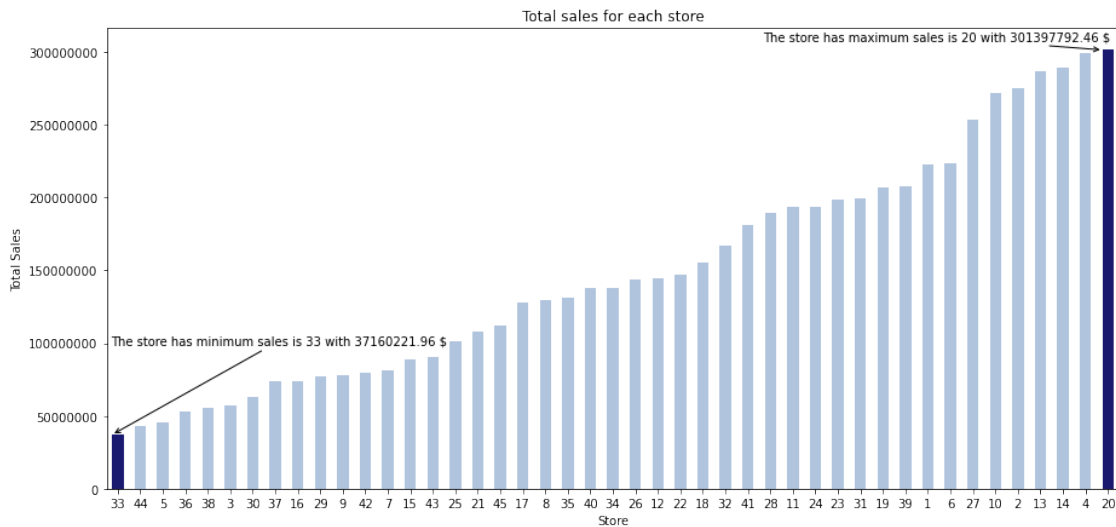
```
plt.title('Total sales for each store')
plt.xlabel('Store')
plt.ylabel('Total Sales');
```

<class 'numpy.float64'>

1.2 Task 2: Which store has maximum standard deviation i.e., the sales vary a lot. Also, find out the coefficient of mean to standard deviation

```
[30]: # Which store has maximum standard deviation
      data_std = pd.DataFrame(data.groupby('Store')['Weekly_Sales'].std().
       ↪sort_values(ascending=False))
      print("The store has maximum standard deviation is "+str(data_std.head(1).
       ↪index[0])+" with {0:.0f} $".format(data_std.head(1).Weekly_Sales[data_std.
       ↪head(1).index[0]]))
```

The store has maximum standard deviation is 14 with 317570 $

```
[31]: # Distribution of store has maximum standard deviation
      plt.figure(figsize=(15,7))
      sns.distplot(data[data['Store'] == data_std.head(1).index[0]]['Weekly_Sales'])
      plt.title('The Sales Distribution of Store #'+ str(data_std.head(1).index[0]));
```

C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)

The Sales Distribution of Store #14
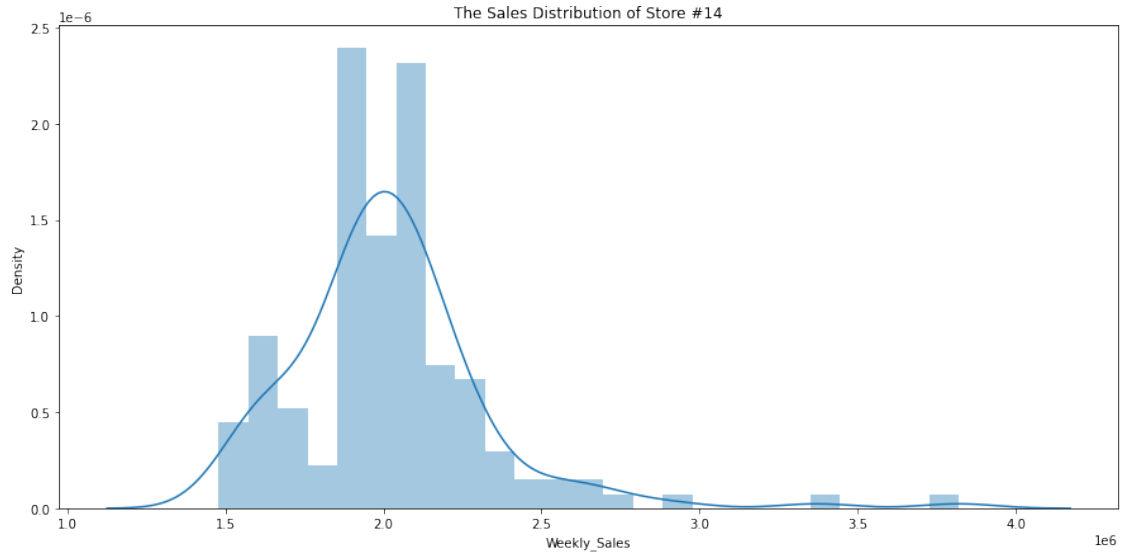
```
[32]: # Coefficient of mean to standard deviation
      coef_mean_std = pd.DataFrame(data.groupby('Store')['Weekly_Sales'].std() / data.
       ↪groupby('Store')['Weekly_Sales'].mean())
      coef_mean_std = coef_mean_std.rename(columns={'Weekly_Sales':'Coefficient of␣
       ↪mean to standard deviation'})
      coef_mean_std
```

[32]:   Coefficient of mean to standard deviation
      Store
      1   0.100292
      2   0.123424
      3   0.115021
      4   0.127083
      5   0.118668
      6   0.135823
      7   0.197305
      8   0.116953
      9   0.126895
      10   0.159133
      11   0.122262
      12   0.137925
      13   0.132514
      14   0.157137
      15   0.193384
      16   0.165181
      17   0.125521
      18   0.162845
      19   0.132680

```
20                              0.130903
21                              0.170292
22                              0.156783
23                              0.179721
24                              0.123637
25                              0.159860
26                              0.110111
27                              0.135155
28                              0.137330
29                              0.183742
30                              0.052008
31                              0.090161
32                              0.118310
33                              0.092868
34                              0.108225
35                              0.229681
36                              0.162579
37                              0.042084
38                              0.110875
39                              0.149908
40                              0.123430
41                              0.148177
42                              0.090335
43                              0.064104
44                              0.081793
45                              0.165613
```

[33]:
```python
# Distribution of store has maximum coefficient of mean to standard deviation
coef_mean_std_max = coef_mean_std.sort_values(by='Coefficient of mean to␣
 ↪standard deviation')
plt.figure(figsize=(15,7))
sns.distplot(data[data['Store'] == coef_mean_std_max.tail(1).
 ↪index[0]]['Weekly_Sales'])
plt.title('The Sales Distribution of Store #'+str(coef_mean_std_max.tail(1).
 ↪index[0]));
```

C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)

The Sales Distribution of Store #35
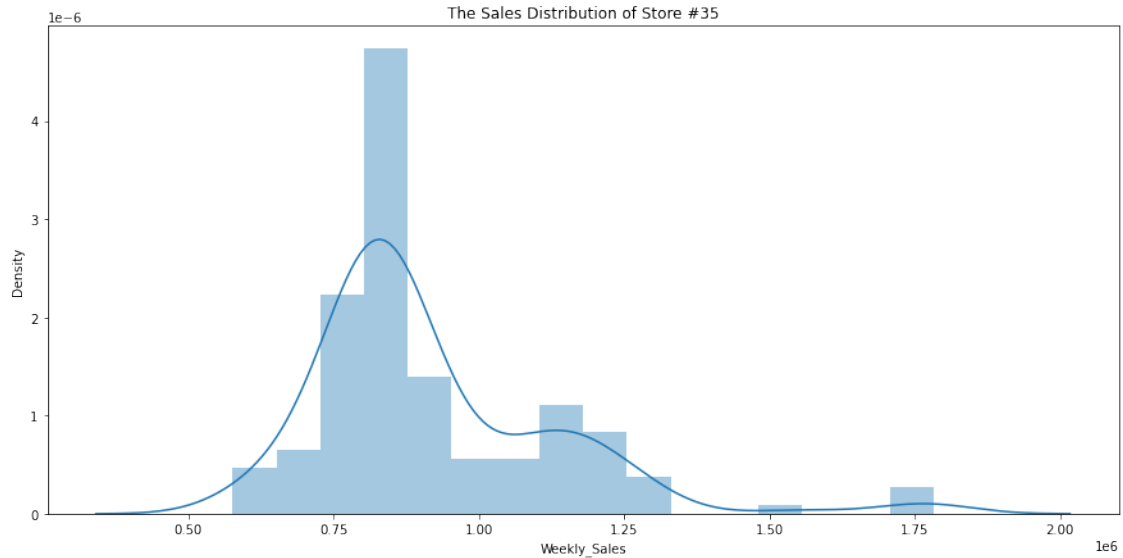
1.3 Task 3 : - Which store/s has good quarterly growth rate in Q3'2012?

```
[36]: plt.figure(figsize=(15,7))

      # Sales for third quarterly in 2012
      Q3 = data[(data['Date'] > '2012-07-01') & (data['Date'] < '2012-09-30')].
       ↪groupby('Store')['Weekly_Sales'].sum()

      # Sales for second quarterly in 2012
      Q2 = data[(data['Date'] > '2012-04-01') & (data['Date'] < '2012-06-30')].
       ↪groupby('Store')['Weekly_Sales'].sum()

      # Plotting the difference between sales for second and third quarterly
      Q2.plot(ax=Q3.plot(kind='bar',legend=True),kind='bar',color='r',alpha=0.
       ↪2,legend=True);
      plt.legend(["Q3' 2012", "Q2' 2012"]);
```
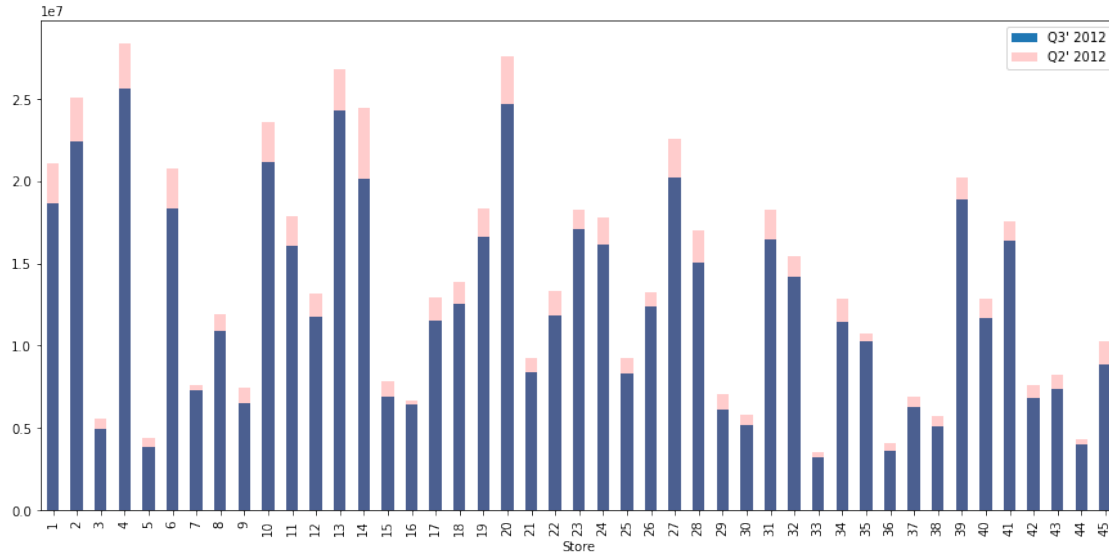
```
[37]:  #  store/s has good quarterly growth rate in Q3'2012 - .
       ↪sort_values(by='Weekly_Sales')
       print('Store have good quarterly growth rate in Q3'2012 is Store '+str(Q3.
       ↪idxmax())+' With '+str(Q3.max())+' $')
```

Store have good quarterly growth rate in Q3'2012 is Store 4 With 25652119.35 $

1.4 Task 4: Some holidays have a negative impact on sales. Find out holidays which have higher sales than the mean sales in non-holiday season for all stores together

1.5 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

```
[39]:  def plot_line(df,holiday_dates,holiday_label):
           fig, ax = plt.subplots(figsize = (15,5))
           ax.plot(df['Date'],df['Weekly_Sales'],label=holiday_label)

           for day in holiday_dates:
               day = datetime.strptime(day, '%d-%m-%Y')
               plt.axvline(x=day, linestyle='--', c='r')


           plt.title(holiday_label)
           x_dates = df['Date'].dt.strftime('%Y-%m-%d').sort_values().unique()
```

10

```
    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()


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']

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

Thanksgiving
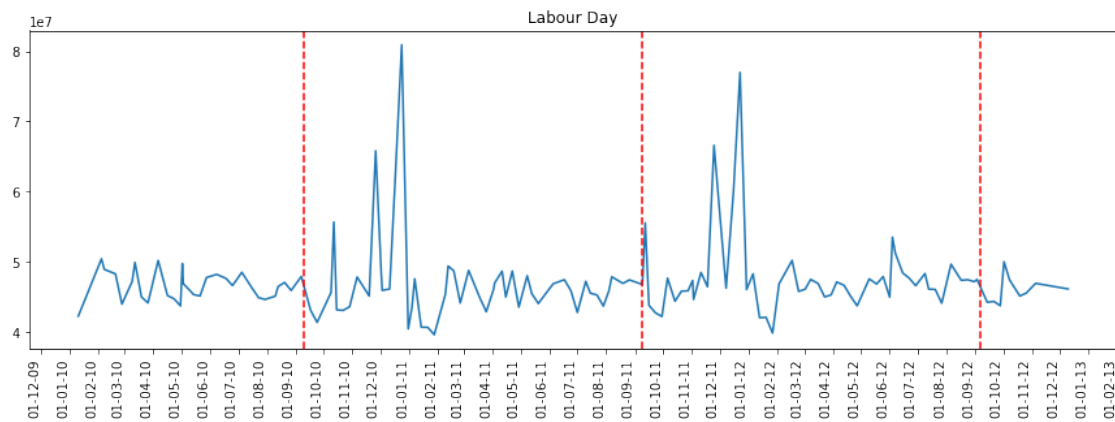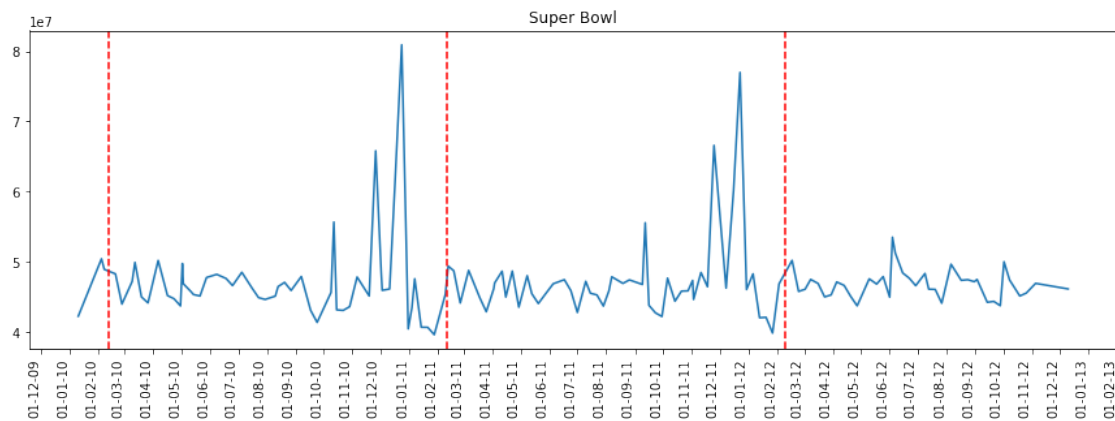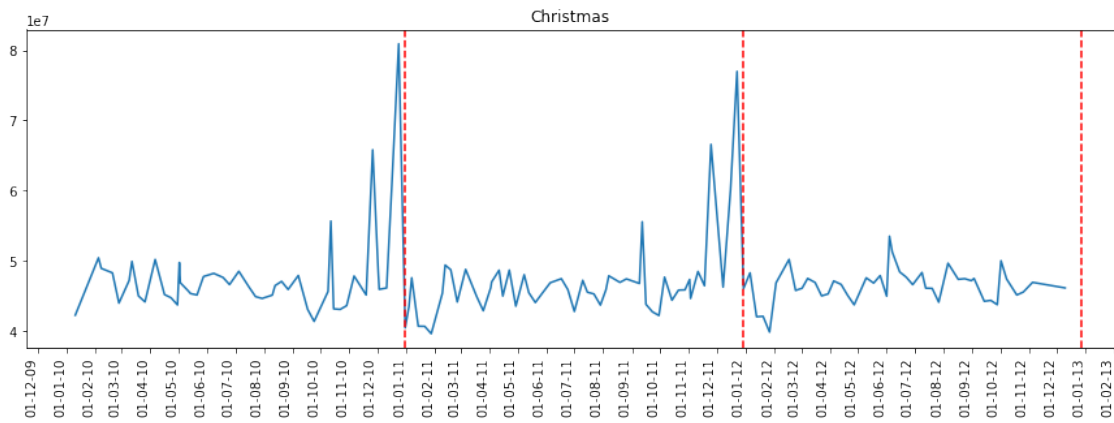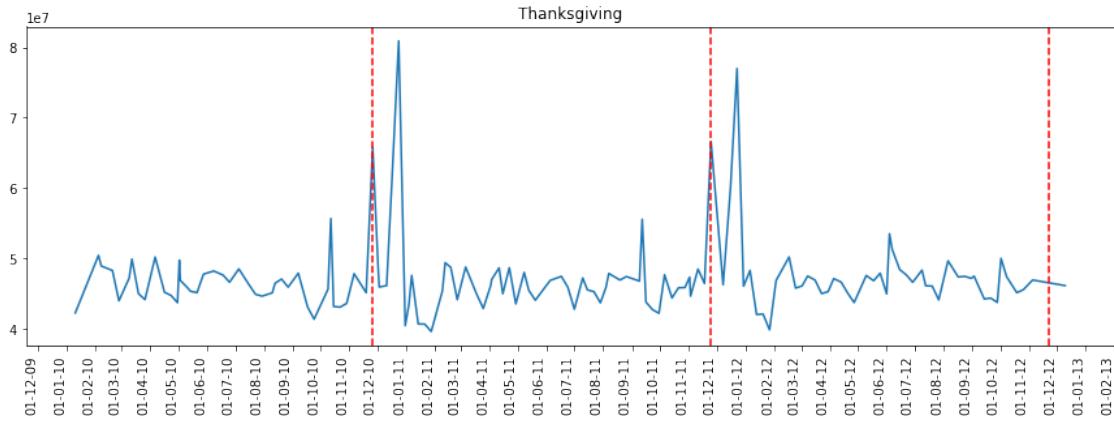
Christmas

The sales increased during thanksgiving. And the sales decreased during christmas.

```
[41]: data.loc[data.Date.isin(Super_Bowl)]
```

```
[41]:        Store        Date  Weekly_Sales  Holiday_Flag  Temperature  Fuel_Price  \
      1          1  2010-12-02    1641957.44             1        38.51       2.548
      53         1  2011-11-02    1649614.93             1        36.39       3.022
      105        1  2012-10-02    1802477.43             1        48.02       3.409
      144        2  2010-12-02    2137809.50             1        38.49       2.548
      196        2  2011-11-02    2168041.61             1        33.19       3.022

      ...      ...         ...           ...           ...          ...         ...
      6202      44  2011-11-02     307486.73             1        30.83       3.034
      6254      44  2012-10-02     325377.97             1        33.73       3.116
      6293      45  2010-12-02     656988.64             1        27.73       2.773
      6345      45  2011-11-02     766456.00             1        30.30       3.239
      6397      45  2012-10-02     803657.12             1        37.00       3.640

             CPI  Unemployment  Day  Month  Year
```

```
1      211.242170         8.106    2    12   2010
53     212.936705         7.742    2    11   2011
105    220.265178         7.348    2    10   2012
144    210.897994         8.324    2    12   2010
196    212.592862         8.028    2    11   2011
...        ...             ...   ...   ...   ...
6202   127.859129         7.224    2    11   2011
6254   130.384903         5.774    2    10   2012
6293   181.982317         8.992    2    12   2010
6345   183.701613         8.549    2    11   2011
6397   189.707605         8.424    2    10   2012

[135 rows x 11 columns]
```

```
[42]:  # Yearly Sales in holidays
       Super_Bowl_df = pd.DataFrame(data.loc[data.Date.isin(Super_Bowl)].
        →groupby('Year')['Weekly_Sales'].sum())
       Thanksgiving_df = pd.DataFrame(data.loc[data.Date.isin(Thanksgiving)].
        →groupby('Year')['Weekly_Sales'].sum())
       Labour_Day_df = pd.DataFrame(data.loc[data.Date.isin(Labour_Day)].
        →groupby('Year')['Weekly_Sales'].sum())
       Christmas_df = pd.DataFrame(data.loc[data.Date.isin(Christmas)].
        →groupby('Year')['Weekly_Sales'].sum())

       Super_Bowl_df.plot(kind='bar',legend=False,title='Yearly Sales in Super Bowl␣
        →holiday')
       Thanksgiving_df.plot(kind='bar',legend=False,title='Yearly Sales in␣
        →Thanksgiving holiday')
       Labour_Day_df.plot(kind='bar',legend=False,title='Yearly Sales in Labour_Day␣
        →holiday')
       Christmas_df.plot(kind='bar',legend=False,title='Yearly Sales in Christmas␣
        →holiday')
```

```
[42]:  <AxesSubplot:title={'center':'Yearly Sales in Christmas holiday'},
       xlabel='Year'>
```

Yearly Sales in Super Bowl holiday

Yearly Sales in Thanksgiving holiday

Yearly Sales in Labour_Day holiday

Yearly Sales in Christmas holiday
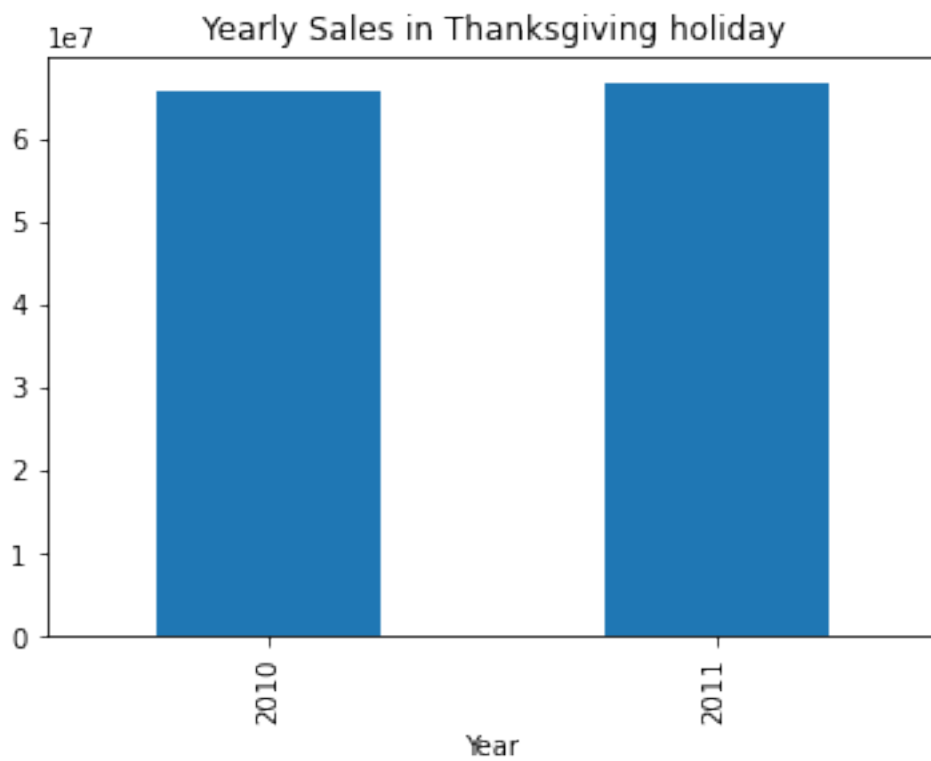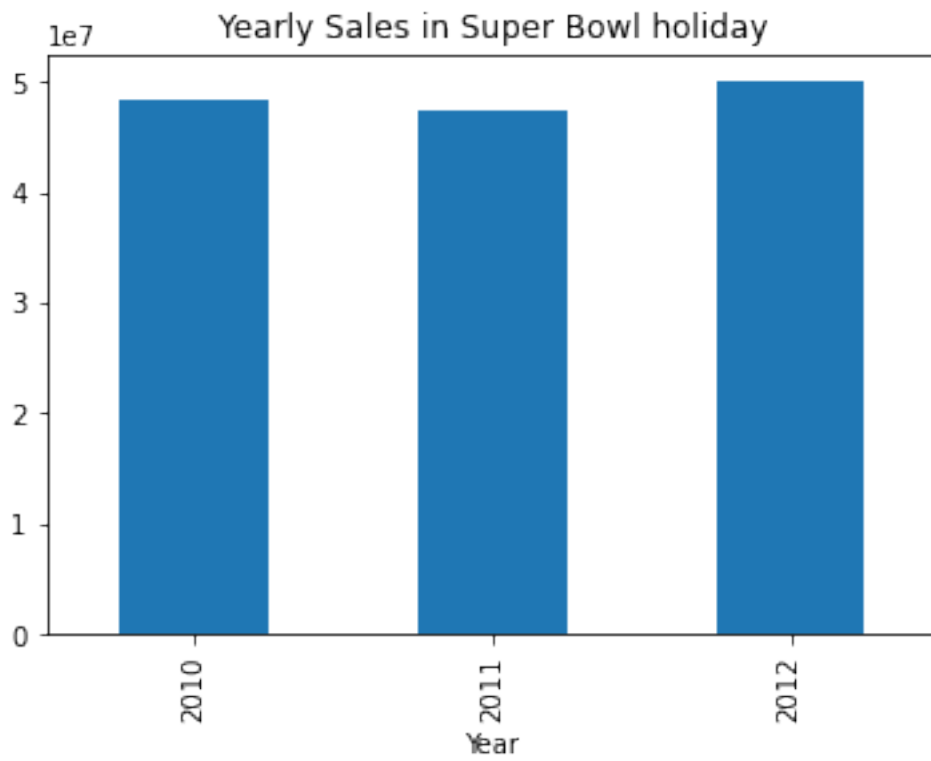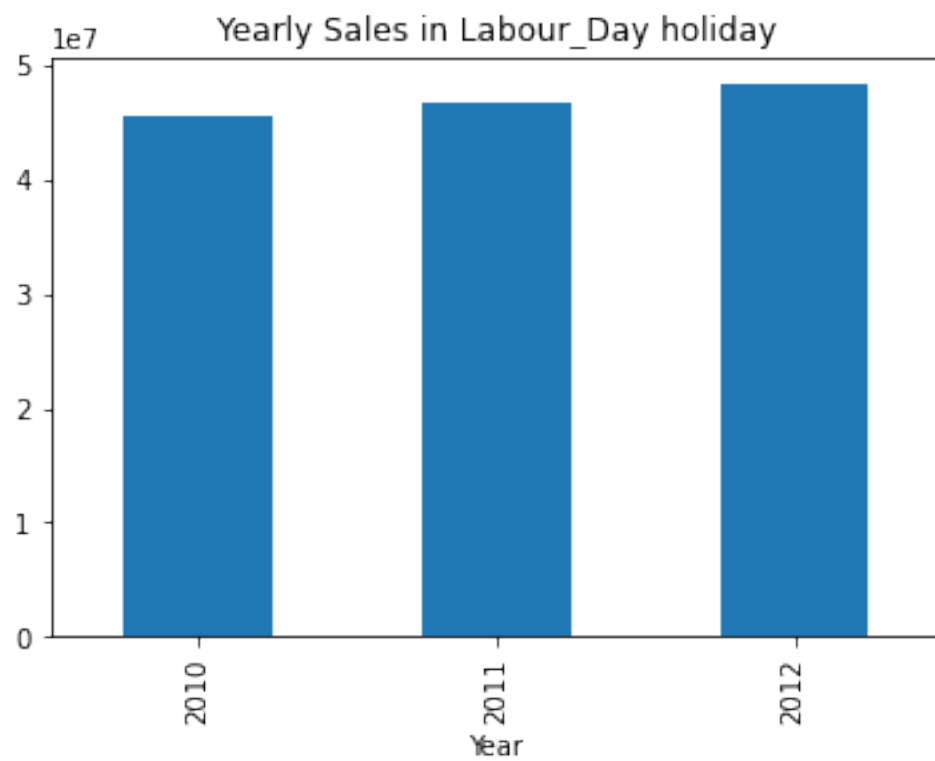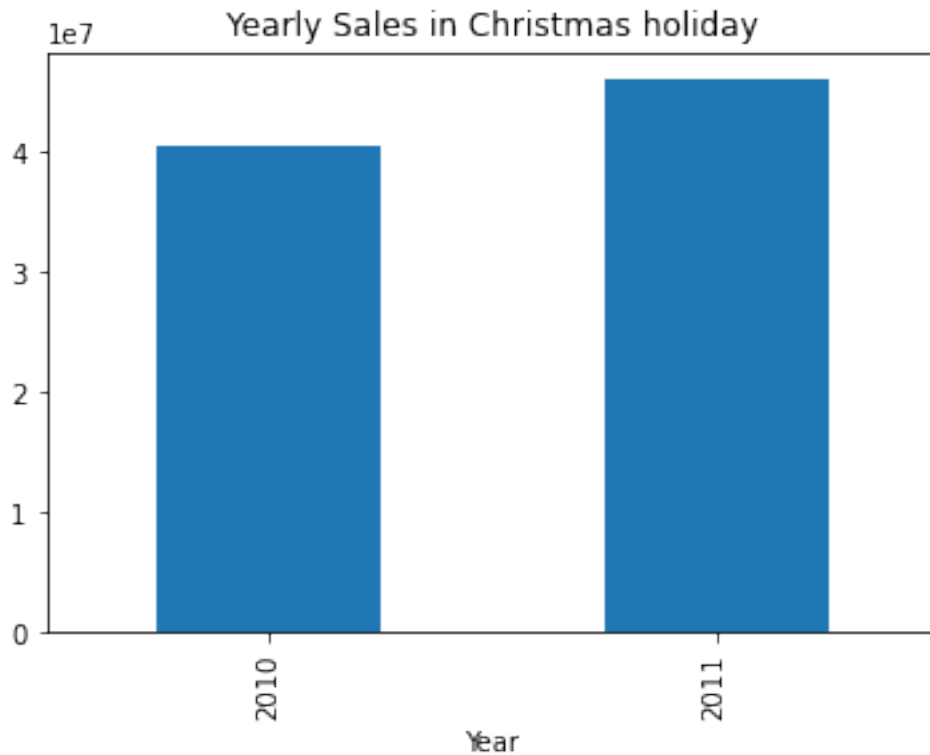
1.6 Task 5: Provide a monthly and semester view of sales in units and give insights

```
[43]:  # Monthly view of sales for each years
       plt.scatter(data[data.Year==2010]["Month"],data[data.
        ↪Year==2010]["Weekly_Sales"])
       plt.xlabel("months")
       plt.ylabel("Weekly Sales")
       plt.title("Monthly view of sales in 2010")
       plt.show()

       plt.scatter(data[data.Year==2011]["Month"],data[data.
        ↪Year==2011]["Weekly_Sales"])
       plt.xlabel("months")
       plt.ylabel("Weekly Sales")
       plt.title("Monthly view of sales in 2011")
       plt.show()

       plt.scatter(data[data.Year==2012]["Month"],data[data.
        ↪Year==2012]["Weekly_Sales"])
       plt.xlabel("months")
       plt.ylabel("Weekly Sales")
```
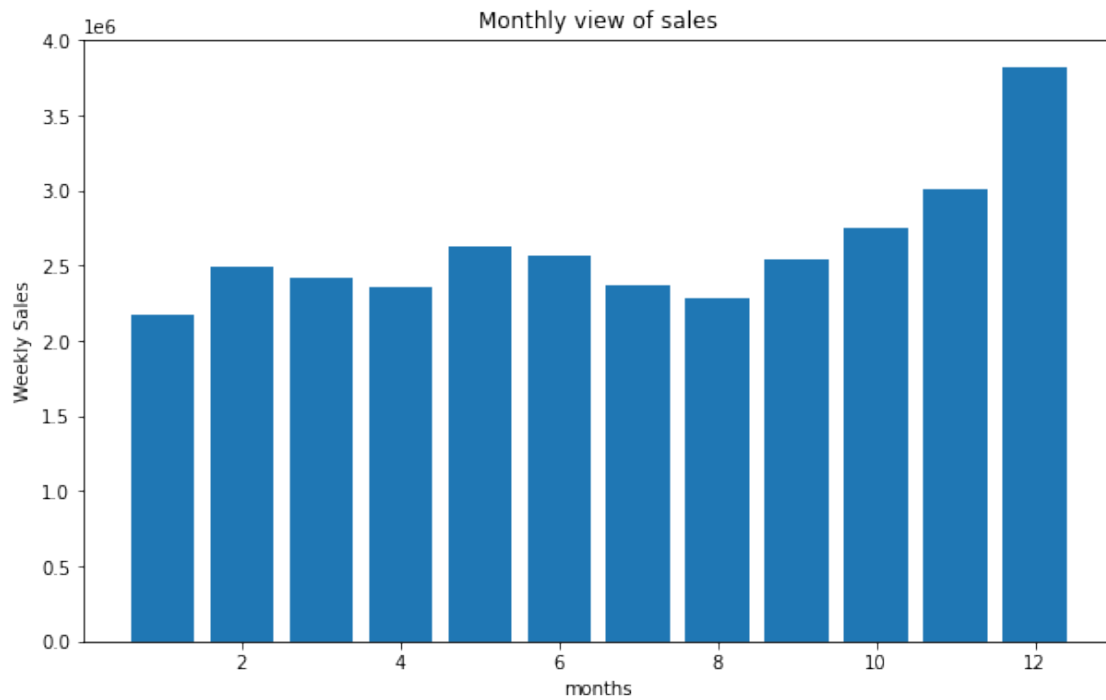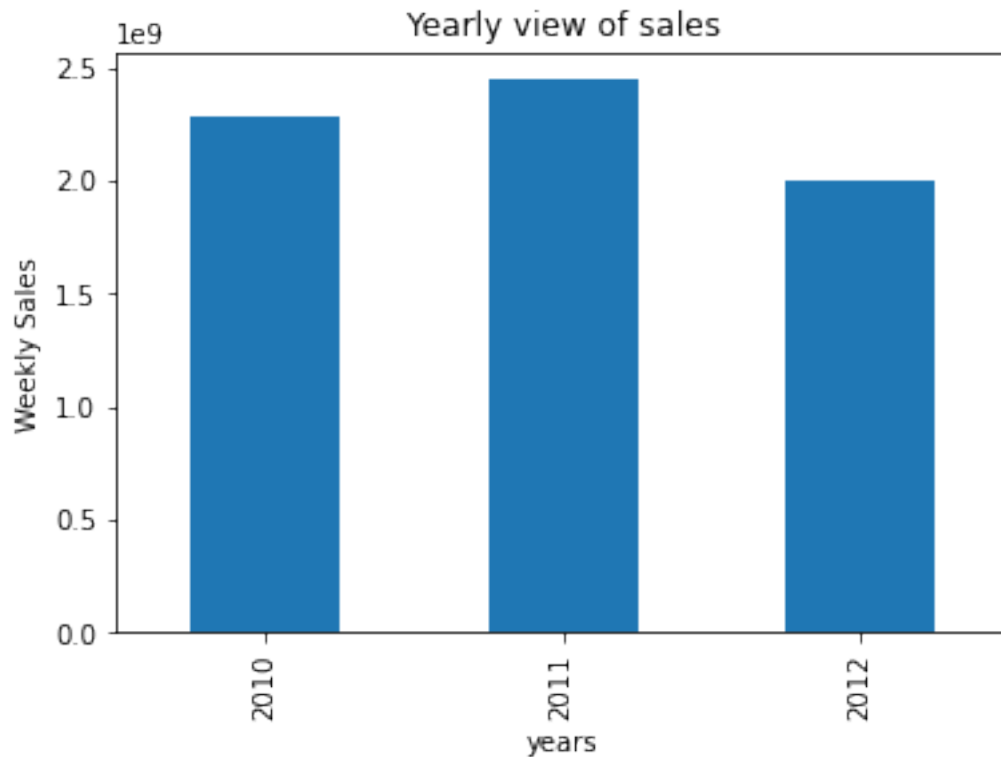
```
plt.title("Monthly view of sales in 2012")
plt.show()
```


Monthly view of sales in 2010

Monthly view of sales in 2011

Monthly view of sales in 2012
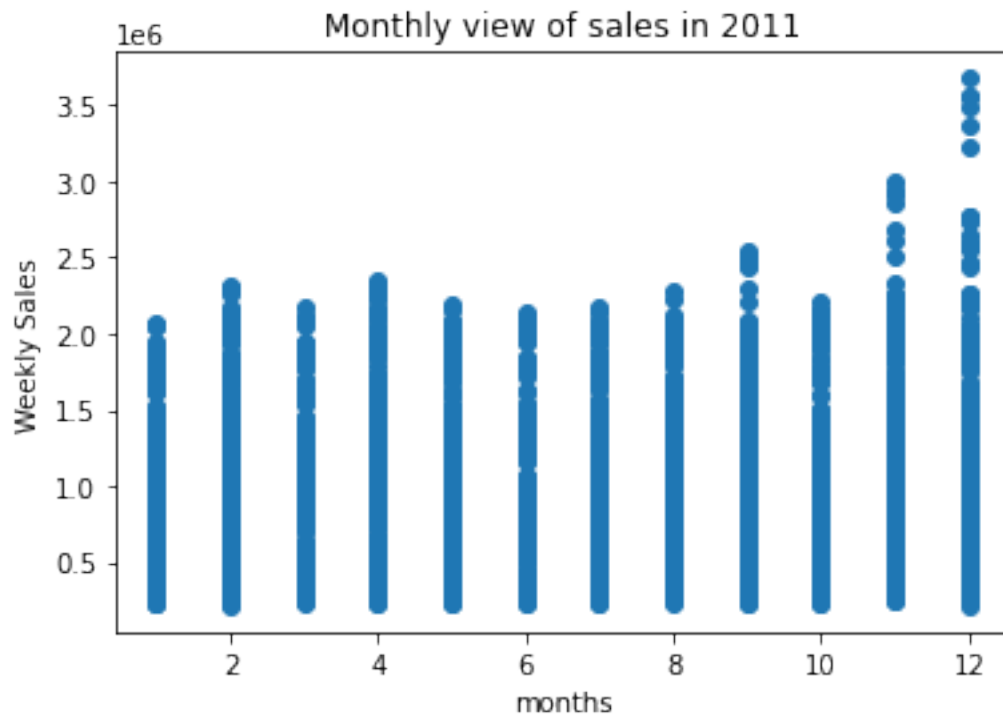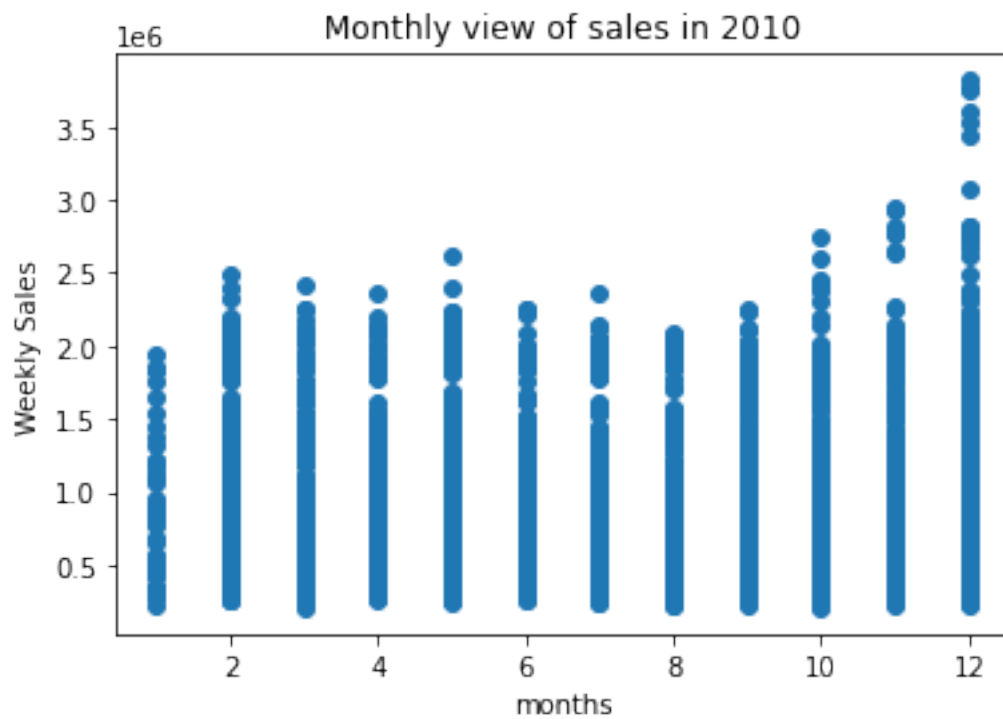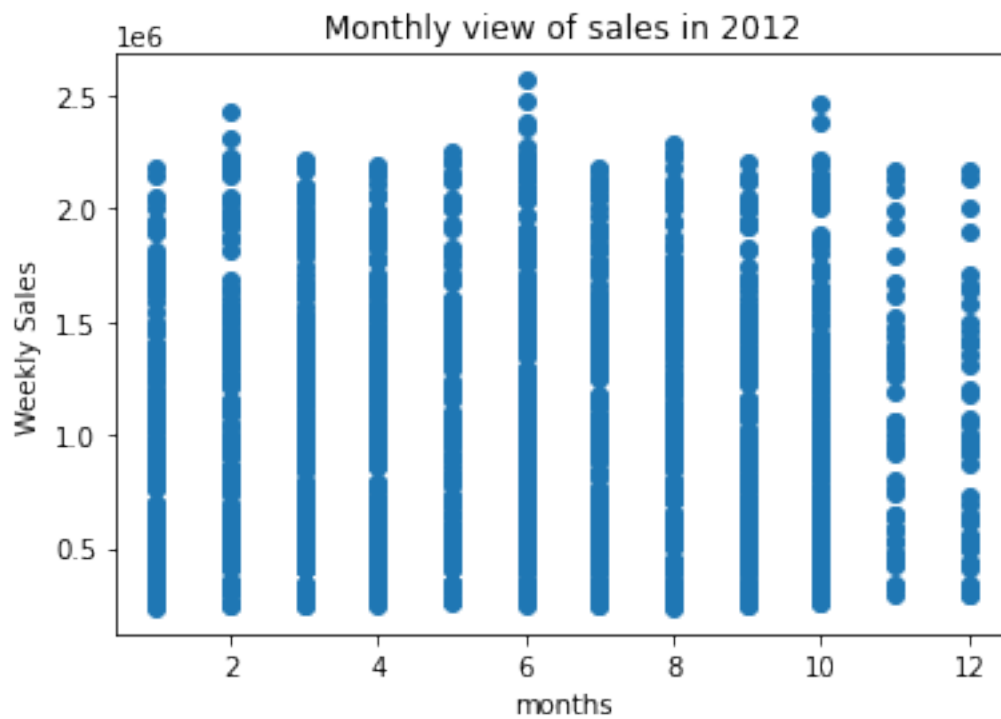
```
[44]: # Monthly view of sales for all years
      plt.figure(figsize=(10,6))
      plt.bar(data["Month"],data["Weekly_Sales"])
      plt.xlabel("months")
      plt.ylabel("Weekly Sales")
      plt.title("Monthly view of sales")
```

[44]: Text(0.5, 1.0, 'Monthly view of sales')

Monthly view of sales

[45]: ```python
# Yearly view of sales
plt.figure(figsize=(10,6))
data.groupby("Year")[["Weekly_Sales"]].sum().plot(kind='bar',legend=False)
plt.xlabel("years")
plt.ylabel("Weekly Sales")
plt.title("Yearly view of sales");
```

<Figure size 720x432 with 0 Axes>

Yearly view of sales

## 1.7 MODEL CREATION - Prediction model to forecast demand

```python
# find outliers
fig, axs = plt.subplots(4,figsize=(6,18))
X = data[['Temperature','Fuel_Price','CPI','Unemployment']]
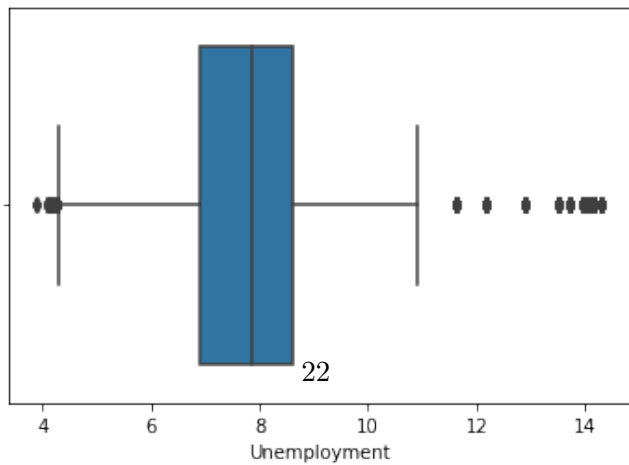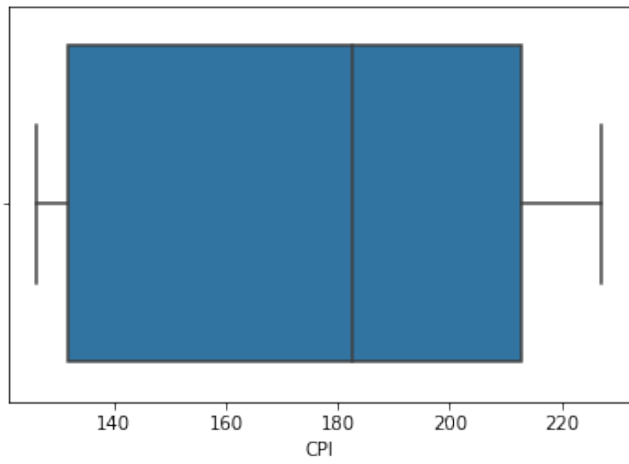for i,column in enumerate(X):
    sns.boxplot(data[column], ax=axs[i])
```

C:\ProgramData\Anaconda3\lib\site-packages\seaborn_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\seaborn_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\seaborn_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other

arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\seaborn_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(

22

```
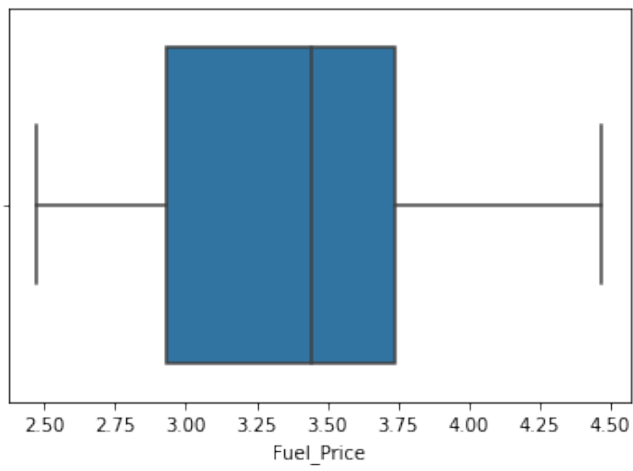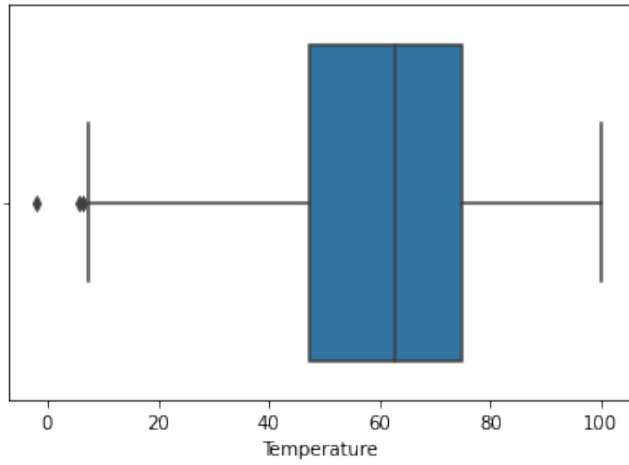[47]: # drop the outliers
      data_new = data[(data['Unemployment']<10) & (data['Unemployment']>4.5) &⏎
       ↪(data['Temperature']>10)]
      data_new
```

```
[47]:         Store         Date   Weekly_Sales   Holiday_Flag   Temperature   Fuel_Price   \
      0            1   2010-05-02    1643690.90              0         42.31        2.572
      1            1   2010-12-02    1641957.44              1         38.51        2.548
      2            1   2010-02-19    1611968.17              0         39.93        2.514
      3            1   2010-02-26    1409727.59              0         46.63        2.561
      4            1   2010-05-03    1554806.68              0         46.50        2.625
      ...        ...          ...           ...            ...           ...          ...
      6430        45   2012-09-28     713173.95              0         64.88        3.997
      6431        45   2012-05-10     733455.07              0         64.89        3.985
      6432        45   2012-12-10     734464.36              0         54.47        4.000
      6433        45   2012-10-19     718125.53              0         56.47        3.969
      6434        45   2012-10-26     760281.43              0         58.85        3.882

                   CPI   Unemployment   Day   Month   Year
      0     211.096358          8.106     2       5   2010
      1     211.242170          8.106     2      12   2010
      2     211.289143          8.106    19       2   2010
      3     211.319643          8.106    26       2   2010
      4     211.350143          8.106     3       5   2010
      ...          ...            ...   ...     ...    ...
      6430  192.013558          8.684    28       9   2012
      6431  192.170412          8.667    10       5   2012
      6432  192.327265          8.667    10      12   2012
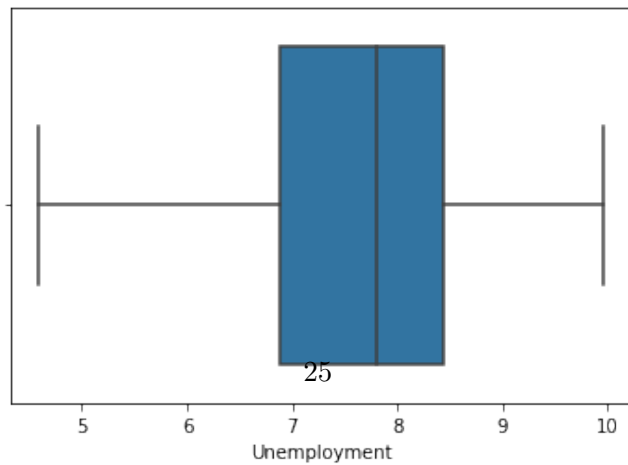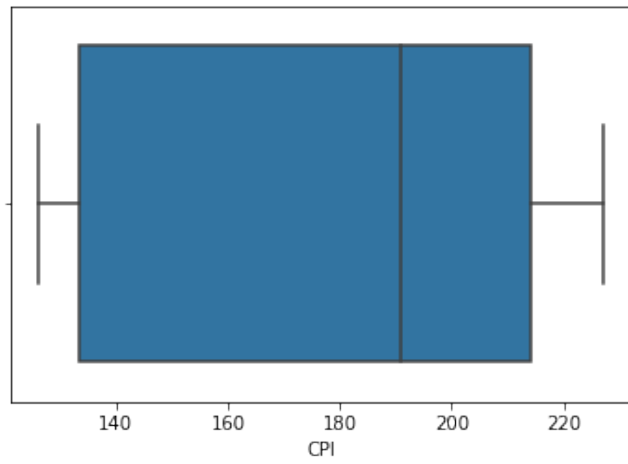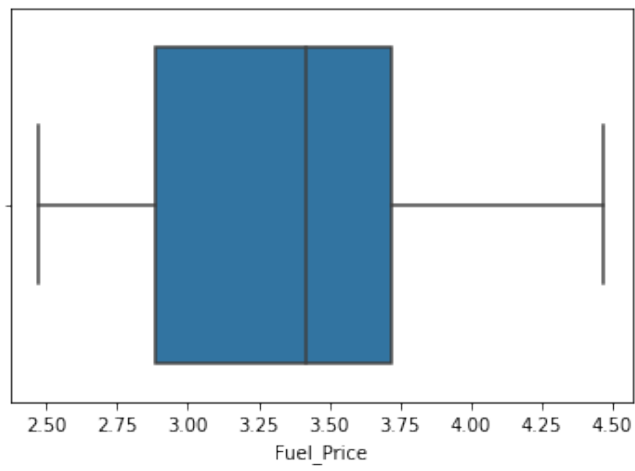      6433  192.330854          8.667    19      10   2012
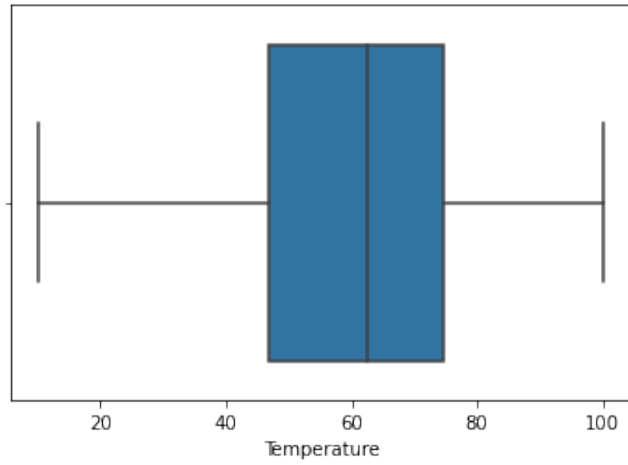      6434  192.308899          8.667    26      10   2012

      [5658 rows x 11 columns]
```

```
[48]: # check outliers
      fig, axs = plt.subplots(4,figsize=(6,18))
      X = data_new[['Temperature','Fuel_Price','CPI','Unemployment']]
      for i,column in enumerate(X):
          sns.boxplot(data_new[column], ax=axs[i])
```

C:\ProgramData\Anaconda3\lib\site-packages\seaborn_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

Building Model

```
[49]:  # Import sklearn
       from sklearn.ensemble import RandomForestRegressor
       from sklearn.model_selection import train_test_split
       from sklearn import metrics
       from sklearn.linear_model import LinearRegression
```

```
[50]:  # Select features and target
       X = data_new[['Store','Fuel_Price','CPI','Unemployment','Day','Month','Year']]
       y = data_new['Weekly_Sales']

       # Split data to train and test (0.80:0.20)
       X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
```

```
[51]:  # Linear Regression model
       print('Linear Regression:')
       print()
       reg = LinearRegression()
       reg.fit(X_train, y_train)
       y_pred = reg.predict(X_test)
       print('Accuracy:',reg.score(X_train, y_train)*100)


       print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
       print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
       print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,␣
        ↪y_pred)))


       sns.scatterplot(y_pred, y_test);
```

```
Linear Regression:

Accuracy: 12.763861175625202
Mean Absolute Error: 451923.9305299199
Mean Squared Error: 292836208023.49854
Root Mean Squared Error: 541143.4264809086

C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```
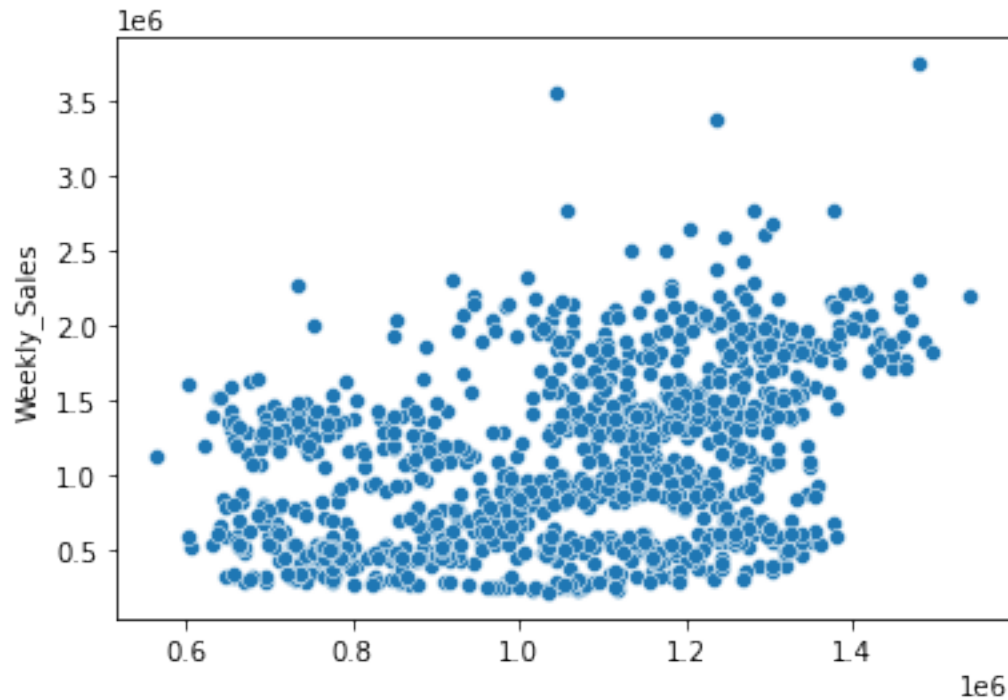
[52]:
```python
# Random Forest Regressor
print('Random Forest Regressor:')
print()
rfr = RandomForestRegressor(n_estimators = 400,max_depth=15,n_jobs=5)
rfr.fit(X_train,y_train)
y_pred=rfr.predict(X_test)
print('Accuracy:',rfr.score(X_test, y_test)*100)

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,
 →y_pred)))


sns.scatterplot(y_pred, y_test);
```

Random Forest Regressor:

Accuracy: 95.42284329295411
Mean Absolute Error: 67684.66541100056
Mean Squared Error: 15424705806.606493
Root Mean Squared Error: 124196.23910008906

C:\ProgramData\Anaconda3\lib\site-packages\seaborn_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version

```
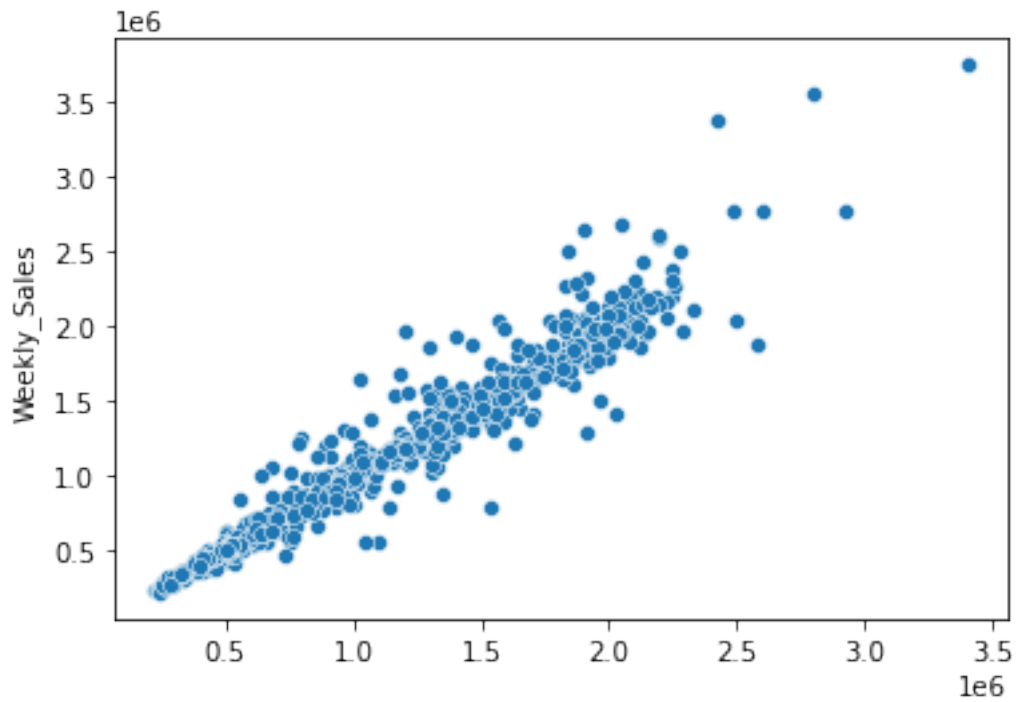0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```


[ ]: