pythonds

August 11, 2023

[9]:

```
# 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
```

[10]:

```
# Load dataset
data = pd.read_csv('Walmart_Store_sales.csv')
data
```

[10]:

```
      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]
```

0.0.1 Data Preparation

[11]:

```
# 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
```

[14]:

```
# checking for missing values
data.isnull().sum()
```

[14]:

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

[6]:

```
# 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
```

[6]:

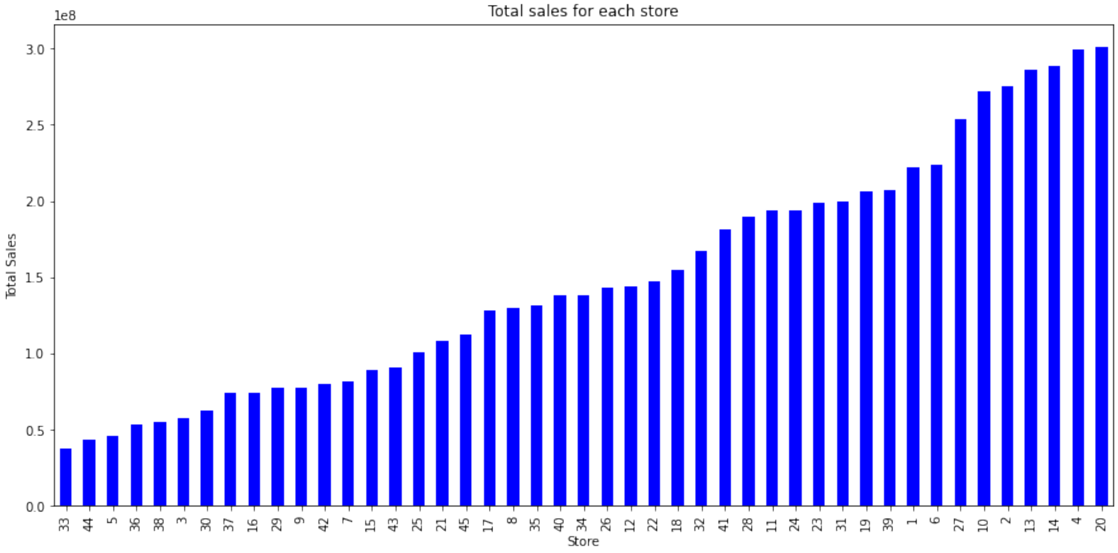
```
      Store       Date  Weekly_Sales  Holiday_Flag  Temperature  Fuel_Price  \
0         1 2010-05-02    1643690.90             0        42.31       2.572
1         1 2010-12-02    1641957.44             1        38.51       2.548
```

```
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]
```

0.0.2 Q1: Which store has minimum and maximum sales?

[24]:

```
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()
ax = total_sales_for_each_store.plot(kind='bar',color="blue");
# plot properties
plt.title('Total sales for each store')
plt.xlabel('Store')
plt.ylabel('Total Sales');
```



0.0.3 Q2: Which store has maximum standard deviation i.e., the sales vary a lot. Also, find out the coefficient of mean to standard deviation?

[33]:
```python
# Which store has maximum standard deviation
data_std = pd.DataFrame(data.groupby('Store')['Weekly_Sales'].std().
 ↪sort_values(ascending=False))
data_std = data_std.rename(columns={'Weekly_Sales':'standard deviation'})
data_std
```

[33]:
```
       standard deviation
Store
14          317569.949476
10          302262.062504
20          275900.562742
4           266201.442297
13          265506.995776
23          249788.038068
27          239930.135688
2           237683.694682
39          217466.454833
6           212525.855862
35          211243.457791
19          191722.638730
41          187907.162766
28          181758.967539
18          176641.510839
24          167745.677567
11          165833.887863
```

```
22           161251.350631
1            155980.767761
12           139166.871880
32           138017.252087
45           130168.526635
21           128752.812853
31           125855.942933
15           120538.652043
40           119002.112858
25           112976.788600
7            112585.469220
17           112162.936087
26           110431.288141
8            106280.829881
34           104630.164676
29            99120.136596
16            85769.680133
9             69028.666585
36            60725.173579
42            50262.925530
3             46319.631557
38            42768.169450
43            40598.413260
5             37737.965745
44            24762.832015
33            24132.927322
30            22809.665590
37            21837.461190
```

[10]:

```
# 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
```

[10]:

```
       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
```

0.0.4 Q3: Which store/s has good quarterly growth rate in Q3'2012

[43]:
```
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()
Q=pd.DataFrame(Q3-Q2).sort_values(by='Weekly_Sales', ascending=False)
Q = Q.rename(columns={'Weekly_Sales':'growth rate'})
Q
```

[43]:

```
       growth rate
Store
16      -184822.33
7       -291200.00
44      -302069.32
33      -335065.62
35      -501448.29
36      -512255.32
5       -546640.33
3       -596172.23
38      -603065.06
30      -604361.01
37      -609253.88
42      -777407.45
26      -800714.31
21      -822771.63
43      -863066.64
9       -903080.57
29      -906631.12
25      -938026.75
15      -958577.86
8      -1060415.27
23     -1179770.54
41     -1186447.44
40     -1202086.08
32     -1273071.37
39     -1291630.46
18     -1327184.36
34     -1381769.00
17     -1384893.64
12     -1415856.54
45     -1427657.73
22     -1510521.06
24     -1642192.12
19     -1670937.25
11     -1784732.70
31     -1794826.89
28     -1930340.28
6      -2387749.05
27     -2402402.62
1      -2403755.60
```

```
10     -2429077.48
13     -2483231.20
2      -2688256.00
4      -2732065.81
20     -2884242.51
14     -4287338.66

<Figure size 1080x504 with 0 Axes>
```

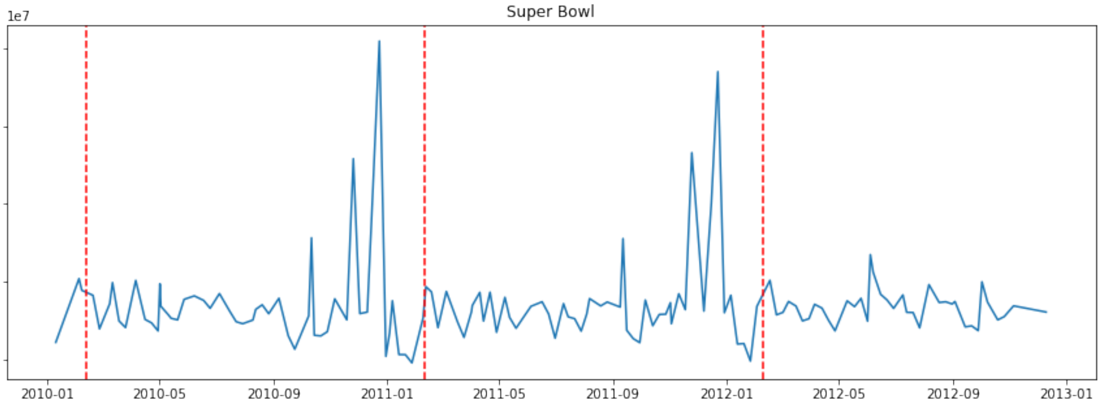
0.0.5 Q4: Some holidays have a negative impact on sales. Find out holidays which have higher sales than the mean sales in non-holiday season for all stores together

Holiday Events:

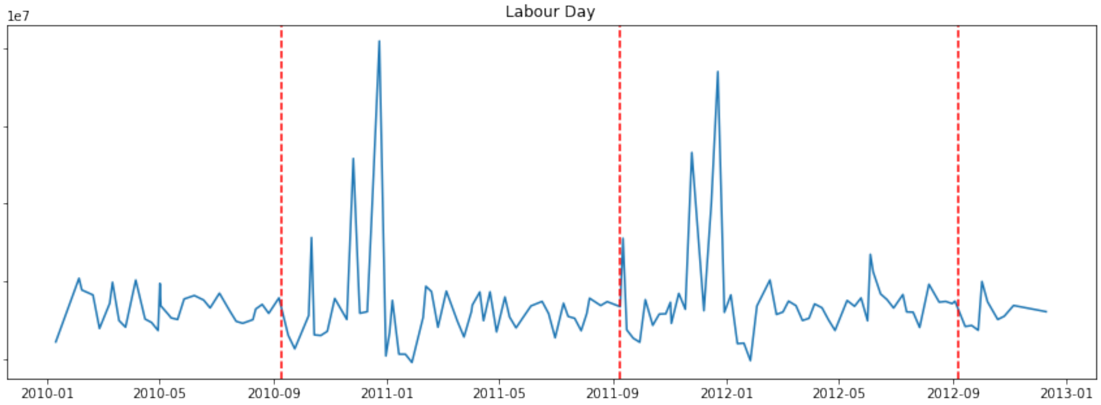
- Super Bowl: 12-Feb-10, 11-Feb-11, 10-Feb-12, 8-Feb-13
- Labour Day: 10-Sep-10, 9-Sep-11, 7-Sep-12, 6-Sep-13
- Thanksgiving: 26-Nov-10, 25-Nov-11, 23-Nov-12, 29-Nov-13
- Christmas: 31-Dec-10, 30-Dec-11, 28-Dec-12, 27-Dec-13

[68]:

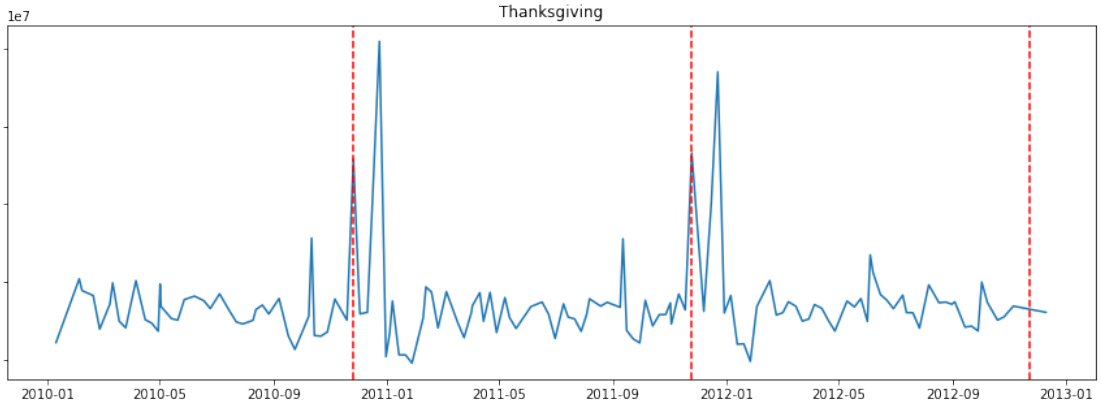
```python
def plot_line(df,holiday_dates,holiday_label):
    fig, ax=plt.subplots(figsize = (15,5))
    ax.plot(df['Date'],df['Weekly_Sales'],label=holiday_label)

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


    plt.title(holiday_label)
    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')
```



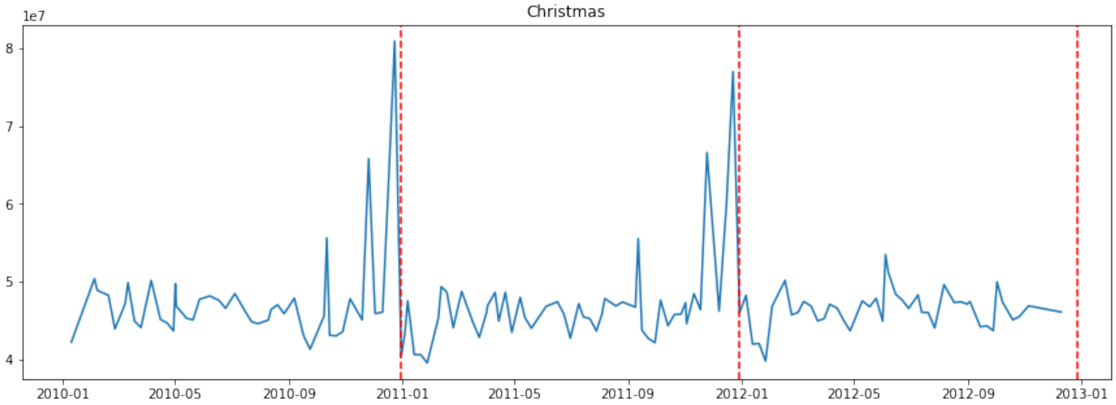
Super Bowl



Labour Day



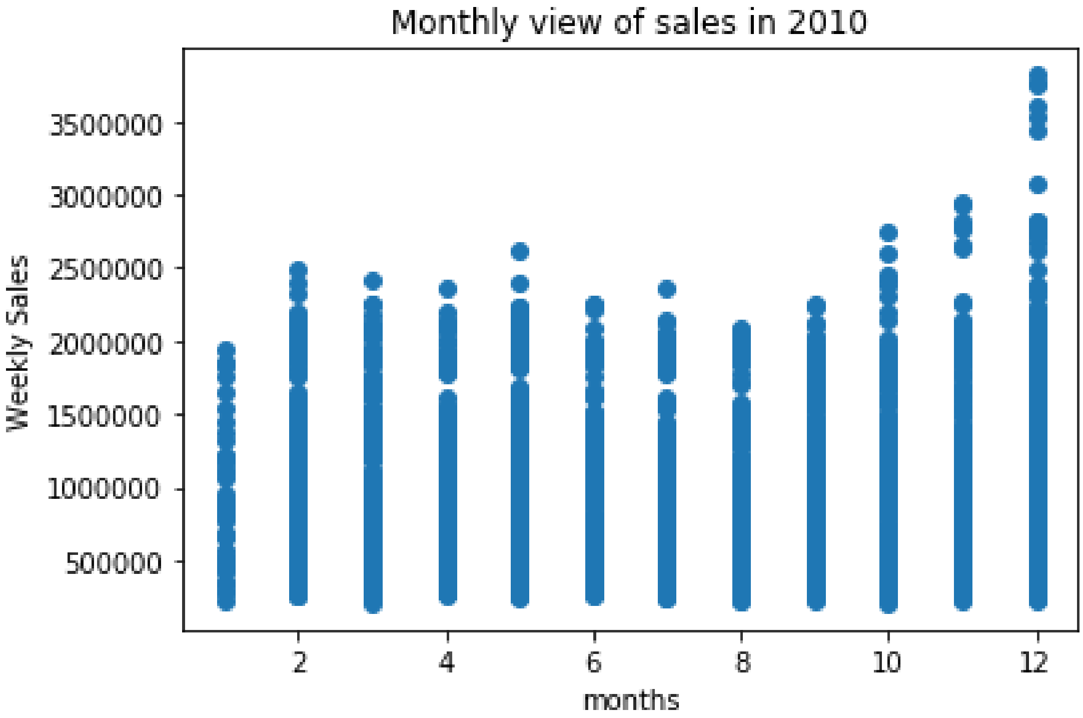
Thanksgiving

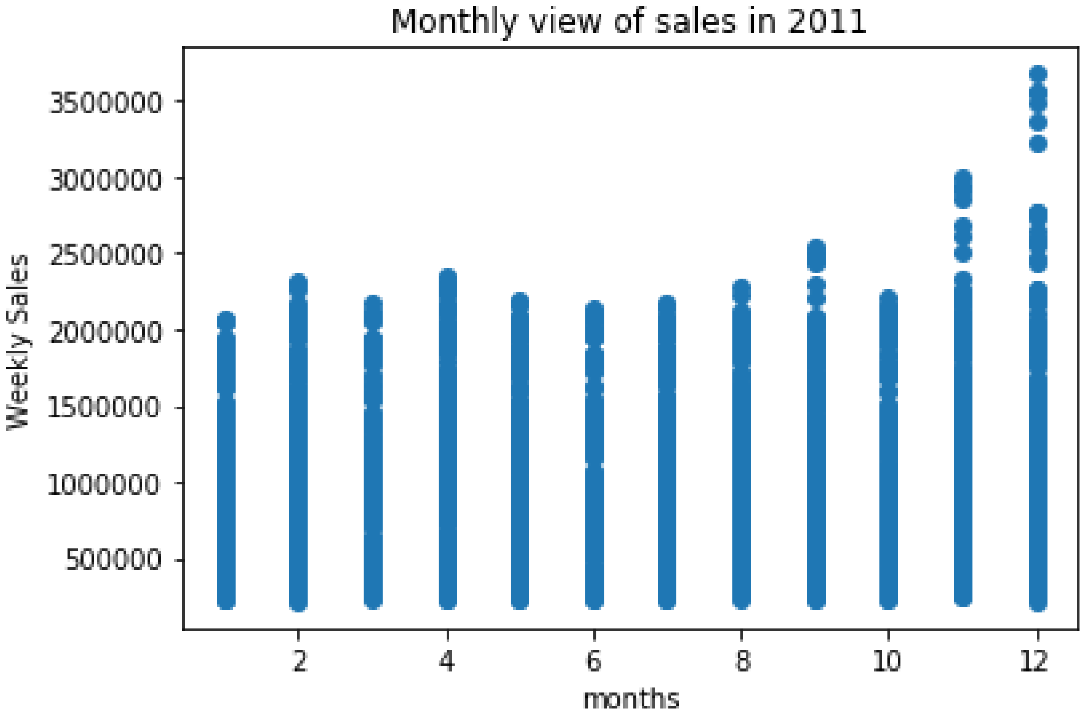


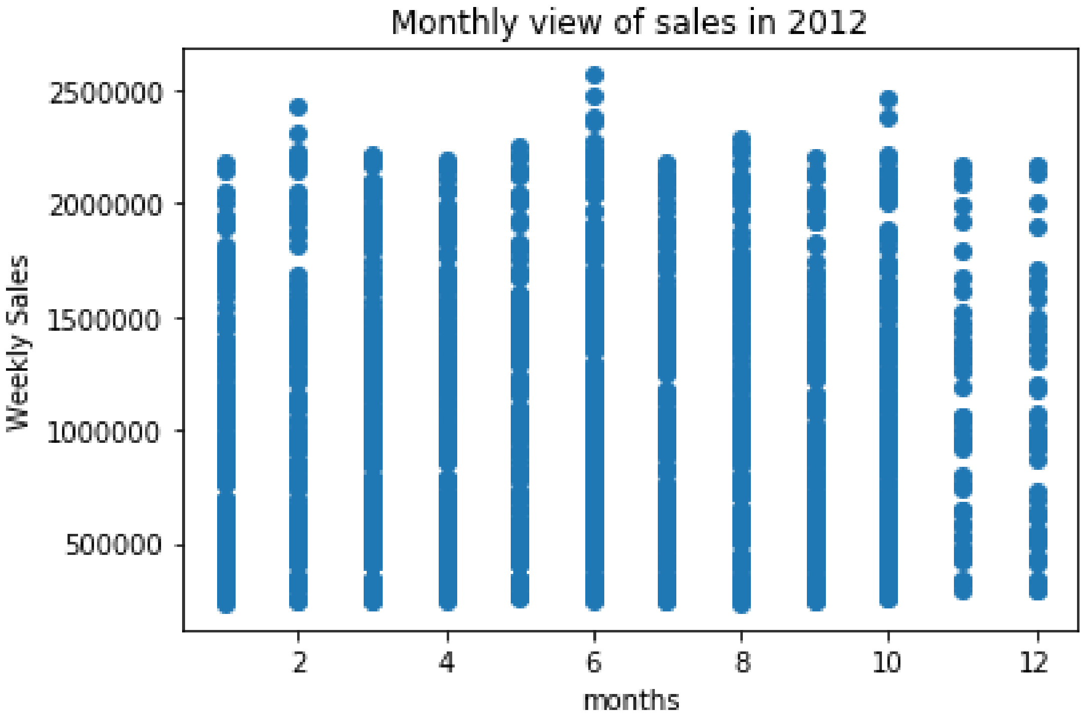
The sales increased during thanksgiving. And the sales decreased during christmas.

0.0.6 Q5: Provide a monthly and semester view of sales in units and give insights

[101]:
```
# 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()
```

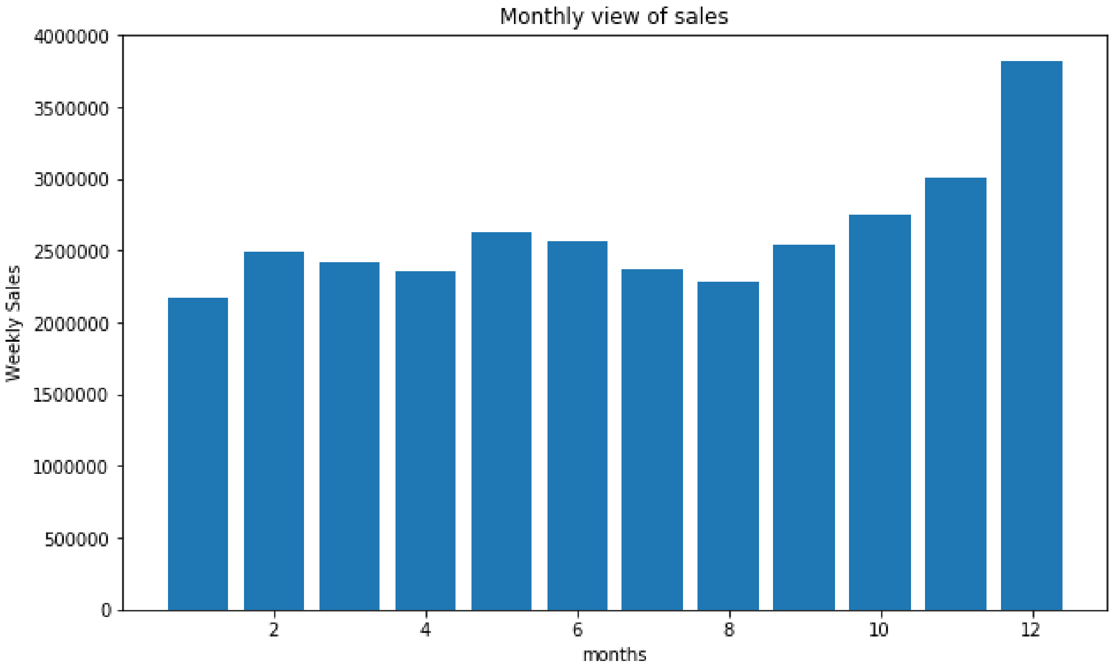

Monthly view of sales in 2010
Weekly Sales
3500000
3000000
2500000
2000000
1500000
1000000
500000
2
4
6
8
10
12
months


Monthly view of sales in 2011
Weekly Sales
3500000
3000000
2500000
2000000
1500000
1000000
500000
2
4
6
8
10
12
months



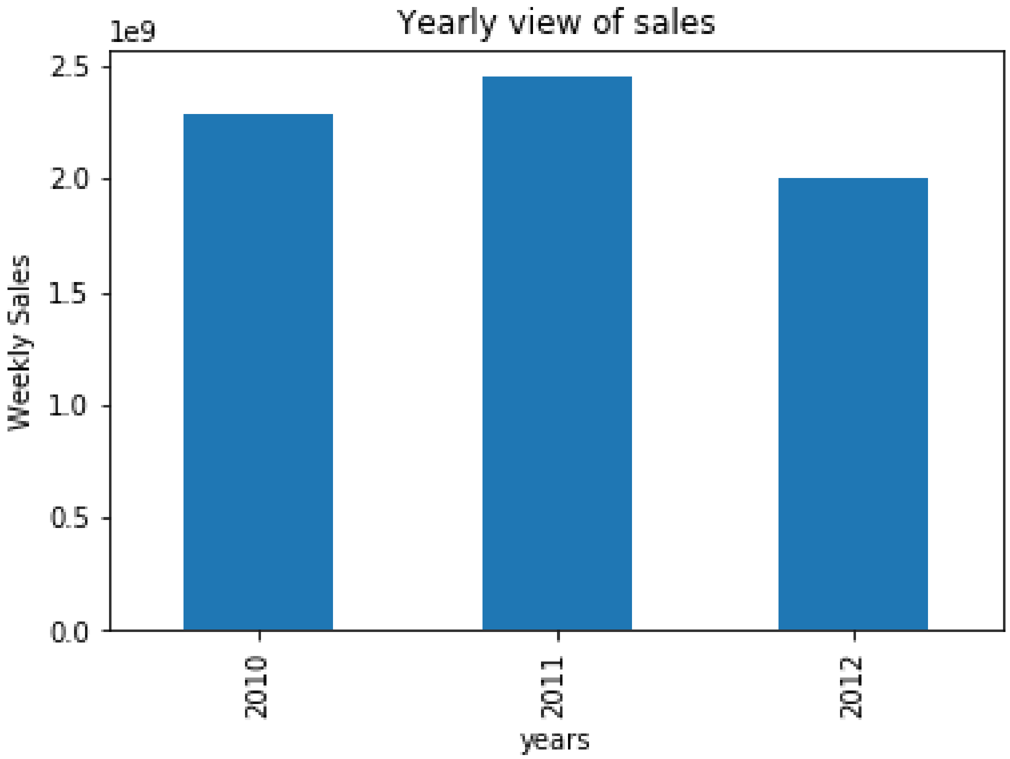
```
[102]: # Monthly view of sales for all years
       plt.figure(figsize=(10,6))
       plt.bar(data["Month"],data["Weekly_Sales"])
       plt.xlabel("months")
       plt.ylabel("Weekly Sales")
       plt.title("Monthly view of sales")
```

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



[110]:
```
# 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>
```



0.0.7 Build Model

[72]:

```
# Import sklearn
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.linear_model import LinearRegression
```

[78]:

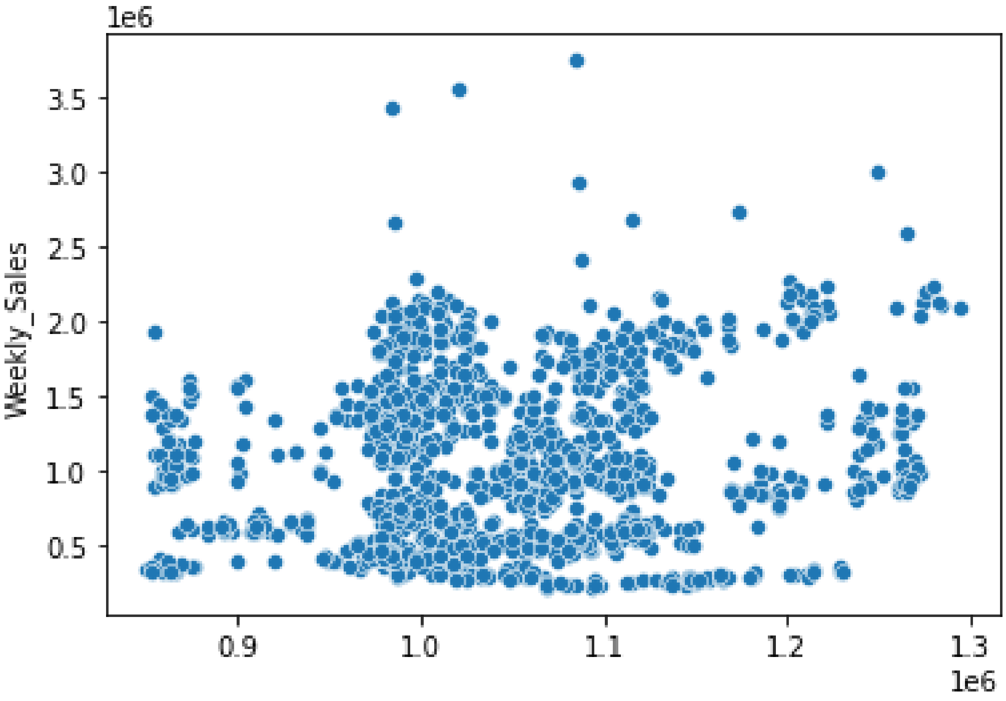
```
# Select features and target
X = data[['Fuel_Price','CPI','Unemployment']]
y = data['Weekly_Sales']

# 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)
```

[79]:

```
# 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: 2.388431611588482
Mean Absolute Error: 461609.5158228802
Mean Squared Error: 301119944808.93097
Root Mean Squared Error: 548743.9701800203
```

```
/opt/anaconda3/lib/python3.9/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(
```



```
[ ]:
```