Project 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.

Data 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

Importing the Library

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
%matplotlib inline
import datetime as dt
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
walmart_df = pd.read_csv("Walmart_Store_sales.csv")
```

In [3]:
```
walmart_df.head()
```

Out[3]:

	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

Basic Information about the dataset

In [4]:
```
# shape of the dataset

walmart_df.shape
```

Out[4]:
```
(6435, 8)
```

*The Walmart dataset have 6435 records spread around 8 Features

In [5]:
```
# General info

walmart_df.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
```

- There are No Null value in the dataset
- We have 5 Float variable, 2 int and 1 obj
- Although Date should be the datetime variable, we'll see about that later

In [6]:

```
# Descriptive statistics about the dataset
walmart_df.describe()
```

Out[6]:

	Store	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemploym
count	6435.000000	6.435000e+03	6435.000000	6435.000000	6435.000000	6435.000000	6435.000(
mean	23.000000	1.046965e+06	0.069930	60.663782	3.358607	171.578394	7.999
std	12.988182	5.643666e+05	0.255049	18.444933	0.459020	39.356712	1.875
min	1.000000	2.099862e+05	0.000000	-2.060000	2.472000	126.064000	3.879(
25%	12.000000	5.533501e+05	0.000000	47.460000	2.933000	131.735000	6.891(
50%	23.000000	9.607460e+05	0.000000	62.670000	3.445000	182.616521	7.874(
75%	34.000000	1.420159e+06	0.000000	74.940000	3.735000	212.743293	8.622(
max	45.000000	3.818686e+06	1.000000	100.140000	4.468000	227.232807	14.313(

In [7]:

```
walmart_df['Holiday_Flag'].sum()
```

Out[7]:

```
450
```

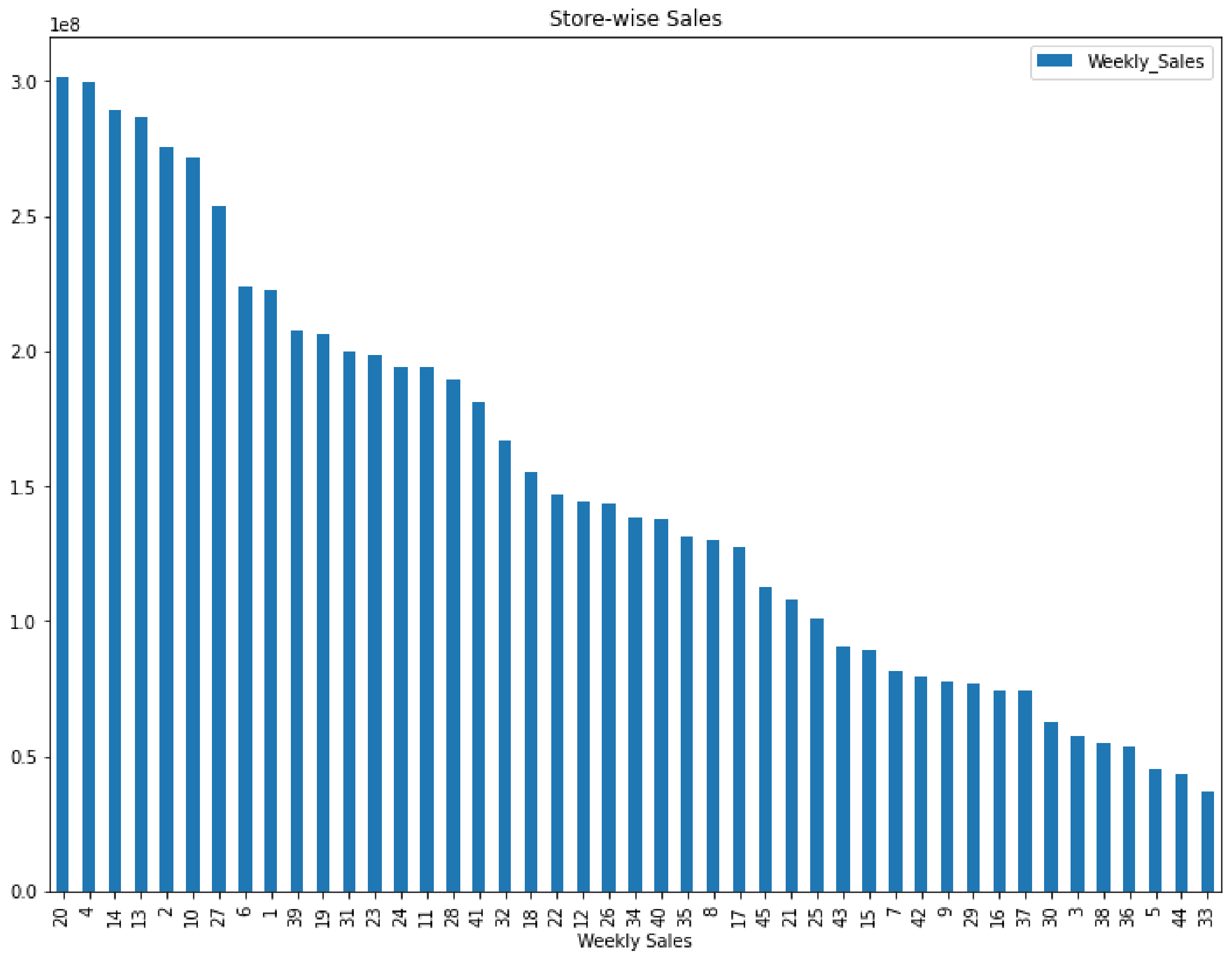
- weekly Sales have a means of 1.04million with min Sales reporting as 209982 and max Sales of $3.8 mn
- Temperature ranges from -2 to 100 with mean temp as 60
- Fuel price ranges from 2.47 to 4.46 with mean fuel price of 3.3

EDA for Walmart dataset

In [8]:

```
# Store with max_Sales
```

```
walmart_df.groupby(by='Store').agg({'Weekly_Sales':'sum'}).sort_values(by='Weekly_Sale

plt.title("Store-wise Sales")
plt.savefig('Store-wise Sales.png')
plt.xlabel("Weekly Sales")
plt.show()
```



- As we can see that store# 20, 4,14, 13 have the highest sales
- And Store# 33,44,5,36,38 have lowest Sales

In [9]:
```
# Which store has maximum standard deviation

Store_sales = walmart_df.groupby(by='Store')['Weekly_Sales'].agg(['std', 'mean']).rese
```

In [10]:
```
Store_sales
```

Out[10]:

	Store	std	mean
0	1	155980.767761	1.555264e+06
1	2	237683.694682	1.925751e+06
2	3	46319.631557	4.027044e+05
3	4	266201.442297	2.094713e+06
4	5	37737.965745	3.180118e+05
5	6	212525.855862	1.564728e+06
6	7	112585.469220	5.706173e+05
7	8	106280.829881	9.087495e+05
8	9	69028.666585	5.439806e+05
9	10	302262.062504	1.899425e+06
10	11	165833.887863	1.356383e+06
11	12	139166.871880	1.009002e+06
12	13	265506.995776	2.003620e+06
13	14	317569.949476	2.020978e+06
14	15	120538.652043	6.233125e+05
15	16	85769.680133	5.192477e+05
16	17	112162.936087	8.935814e+05
17	18	176641.510839	1.084718e+06
18	19	191722.638730	1.444999e+06
19	20	275900.562742	2.107677e+06
20	21	128752.812853	7.560691e+05
21	22	161251.350631	1.028501e+06
22	23	249788.038068	1.389864e+06
23	24	167745.677567	1.356755e+06
24	25	112976.788600	7.067215e+05
25	26	110431.288141	1.002912e+06
26	27	239930.135688	1.775216e+06
27	28	181758.967539	1.323522e+06
28	29	99120.136596	5.394514e+05
29	30	22809.665590	4.385796e+05
30	31	125855.942933	1.395901e+06
31	32	138017.252087	1.166568e+06
32	33	24132.927322	2.598617e+05

	Store	std	mean
33	34	104630.164676	9.667816e+05
34	35	211243.457791	9.197250e+05
35	36	60725.173579	3.735120e+05
36	37	21837.461190	5.189003e+05
37	38	42768.169450	3.857317e+05
38	39	217466.454833	1.450668e+06
39	40	119002.112858	9.641280e+05
40	41	187907.162766	1.268125e+06
41	42	50262.925530	5.564039e+05
42	43	40598.413260	6.333247e+05
43	44	24762.832015	3.027489e+05
44	45	130168.526635	7.859814e+05

In [11]:

```
Store_sales['coef_mean_to_std'] = Store_sales['std']/Store_sales['mean']
```

In [12]:

```
Store_sales.head()
```

Out[12]:

	Store	std	mean	coef_mean_to_std
0	1	155980.767761	1.555264e+06	0.100292
1	2	237683.694682	1.925751e+06	0.123424
2	3	46319.631557	4.027044e+05	0.115021
3	4	266201.442297	2.094713e+06	0.127083
4	5	37737.965745	3.180118e+05	0.118668

In [13]:

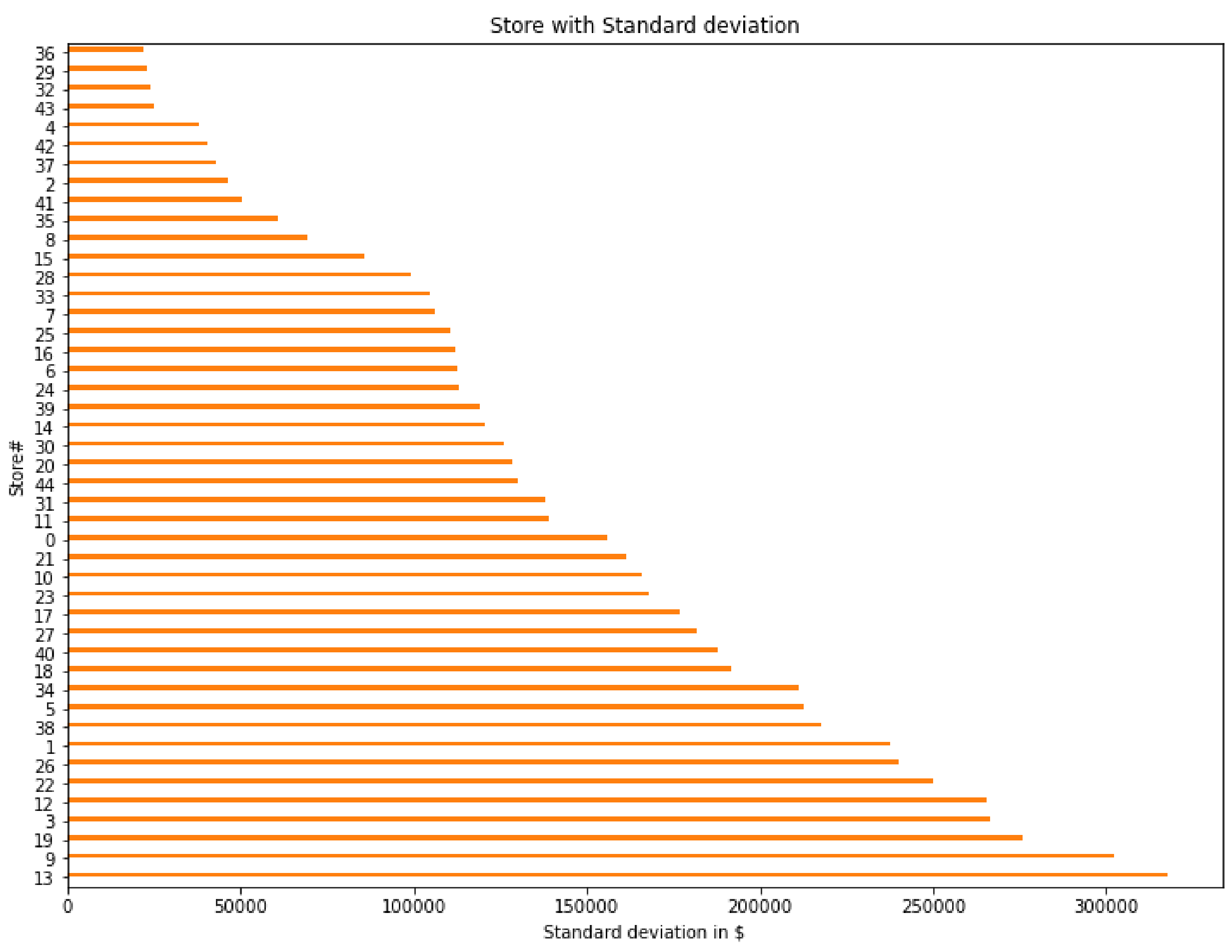
```
Store_sales.sort_values(by=['std'],ascending=False)[['Store','std']]
```

Out[13]:

	Store	std
13	14	317569.949476
9	10	302262.062504
19	20	275900.562742
3	4	266201.442297
12	13	265506.995776
22	23	249788.038068
26	27	239930.135688
1	2	237683.694682
38	39	217466.454833
5	6	212525.855862
34	35	211243.457791
18	19	191722.638730
40	41	187907.162766
27	28	181758.967539
17	18	176641.510839
23	24	167745.677567
10	11	165833.887863
21	22	161251.350631
0	1	155980.767761
11	12	139166.871880
31	32	138017.252087
44	45	130168.526635
20	21	128752.812853
30	31	125855.942933
14	15	120538.652043
39	40	119002.112858
24	25	112976.788600
6	7	112585.469220
16	17	112162.936087
25	26	110431.288141
7	8	106280.829881
33	34	104630.164676
28	29	99120.136596

	Store	std
15	16	85769.680133
8	9	69028.666585
35	36	60725.173579
41	42	50262.925530
2	3	46319.631557
37	38	42768.169450
42	43	40598.413260
4	5	37737.965745
43	44	24762.832015
32	33	24132.927322
29	30	22809.665590
36	37	21837.461190

In [14]:
```
Store_sales.sort_values(by=['std'],ascending=False)[['Store','std']].plot(kind='barh',
plt.title("Store with Standard deviation")
plt.xlabel("Standard deviation in $")
plt.ylabel("Store#")
plt.savefig('Store-wise Std')
plt.show()
```



As we can see that Store 13 have the maximum Standard deviation which we can check with the original dataset

In [15]:
```
# Which store/s has good quarterly growth rate in Q3’2012
```

In [16]:
```
walmart_df['Date'] = pd.to_datetime(walmart_df['Date'])
```

In [17]:
```
from datetime import date
```

In [18]:
```
walmart_df['Quarter'] = pd.PeriodIndex(walmart_df['Date'], freq='Q')
```

In [19]:
```
quarter_wise_sales = walmart_df.groupby(['Store','Quarter']).agg({'Weekly_Sales':'sum'
```

In [20]:
```
Q3_sales = quarter_wise_sales[quarter_wise_sales['Quarter'] == '2012Q3'].groupby('Stor
Q2_sales = quarter_wise_sales[quarter_wise_sales['Quarter'] == '2012Q2'].groupby('Stor
```

In [21]:
```
Q3_sales = pd.merge(Q3_sales,Q2_sales, on=Q3_sales['Store'])
```

In [22]:
```
Q3_sales.head()
```

Out[22]:

	key_0	Store_x	Weekly_Sales_x	Store_y	Weekly_Sales_y
0	1	1	18633209.98	1	21036965.58
1	2	2	22396867.61	2	25085123.61
2	3	3	4966495.93	3	5562668.16
3	4	4	25652119.35	4	28384185.16
4	5	5	3880621.88	5	4427262.21

In [23]:
```
Q3_sales = Q3_sales.drop(['Store_x','Store_y'],axis=1)
#Q3_sales = Q3_sales.rename({'Weekly_Sales_x':'Q3_Sales'},axis=1)
#Q3_sales = Q3_sales.rename({'Weekly_Sales_y':'Q2_Sales'},axis=1)
```

In [24]:
```
Q3_sales = Q3_sales.rename({'Weekly_Sales_x':'Q3_Sales'},axis=1)
Q3_sales = Q3_sales.rename({'Weekly_Sales_y':'Q2_Sales'},axis=1)
```

In [25]:
```
Q3_sales.head()
```

Out[25]:

	key_0	Q3_Sales	Q2_Sales
0	1	18633209.98	21036965.58
1	2	22396867.61	25085123.61
2	3	4966495.93	5562668.16
3	4	25652119.35	28384185.16
4	5	3880621.88	4427262.21

In [26]:
```
Q3_sales['perc_growth'] = np.round((Q3_sales['Q3_Sales'] - Q3_sales['Q2_Sales'])/Q3_sa
```

In [27]:
```
Q3_sales.sort_values(by='perc_growth', ascending=False).head()
```

Out[27]:

	key_0	Q3_Sales	Q2_Sales	perc_growth
15	16	6441311.11	6626133.44	-2.79
6	7	7322393.92	7613593.92	-3.82
34	35	10252122.68	10753570.97	-4.66
25	26	12417575.35	13218289.66	-6.06
38	39	18899955.17	20191585.63	-6.40

As we can see that we saw a dip in 2012Q3 and all the store captures the negetive growth rate over previous quarter but least neg growth rate was done by Store# 15

In [28]:
```
#Some holidays have a negative impact on sales.
#Find out holidays which have higher sales than the mean sales in non-holiday season f

mean_non_holiday_sales = np.round(walmart_df[walmart_df['Holiday_Flag'] == 0]['Weekly_
```

In [29]:
```
mean_non_holiday_sales
```

Out[29]:
```
1041256.38
```

In [30]:
```
holiday_sales = walmart_df[walmart_df['Holiday_Flag']==1]
```

In [31]:
```
holiday_sales
```

Out[31]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
1	1	2010-12-02	1641957.44	1	38.51	2.548	211.242170	8.106
31	1	2010-10-09	1507460.69	1	78.69	2.565	211.495190	7.787
42	1	2010-11-26	1955624.11	1	64.52	2.735	211.748433	7.838
47	1	2010-12-31	1367320.01	1	48.43	2.943	211.404932	7.838
53	1	2011-11-02	1649614.93	1	36.39	3.022	212.936705	7.742
...	...	...	...	...	...	...	...	...
6375	45	2011-09-09	746129.56	1	71.48	3.738	186.673738	8.625
6386	45	2011-11-25	1170672.94	1	48.71	3.492	188.350400	8.523
6391	45	2011-12-30	869403.63	1	37.79	3.389	189.062016	8.523
6397	45	2012-10-02	803657.12	1	37.00	3.640	189.707605	8.424
6427	45	2012-07-09	766512.66	1	75.70	3.911	191.577676	8.684

450 rows × 9 columns

In [32]:

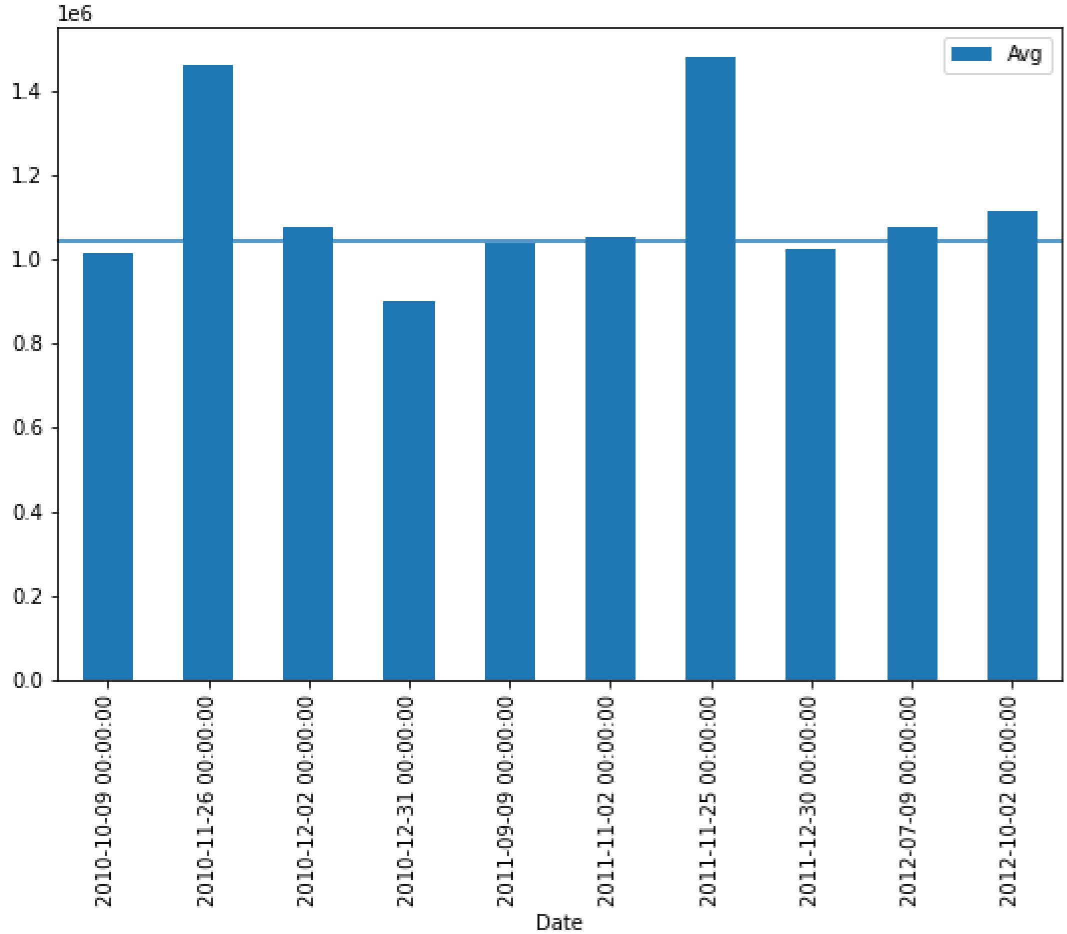
```
holiday_sales.groupby(by='Date')['Weekly_Sales'].agg(Avg='mean').index.tolist()
```

Out[32]:

```
[Timestamp('2010-10-09 00:00:00'),
 Timestamp('2010-11-26 00:00:00'),
 Timestamp('2010-12-02 00:00:00'),
 Timestamp('2010-12-31 00:00:00'),
 Timestamp('2011-09-09 00:00:00'),
 Timestamp('2011-11-02 00:00:00'),
 Timestamp('2011-11-25 00:00:00'),
 Timestamp('2011-12-30 00:00:00'),
 Timestamp('2012-07-09 00:00:00'),
 Timestamp('2012-10-02 00:00:00')]
```

In [33]:

```
holiday_sales.groupby(by='Date')['Weekly_Sales'].agg(Avg='mean').plot(kind='bar',leger
plt.axhline(y=mean_non_holiday_sales)
plt.savefig("holiday season sales.png")
plt.show()
```



As per above graph Following holidays have more sales than the non holiday mean

- ThanksGiving - 2010
- Superbowl - 2010
- ThanksGiving - 2011
- Labour Day - 2012
- Superbowl - 2012

In [34]:
```
#Provide a monthly and semester view of sales in units and give insights

walmart_df['Month'] = walmart_df.Date.dt.month
walmart_df['Year'] = walmart_df.Date.dt.year
```

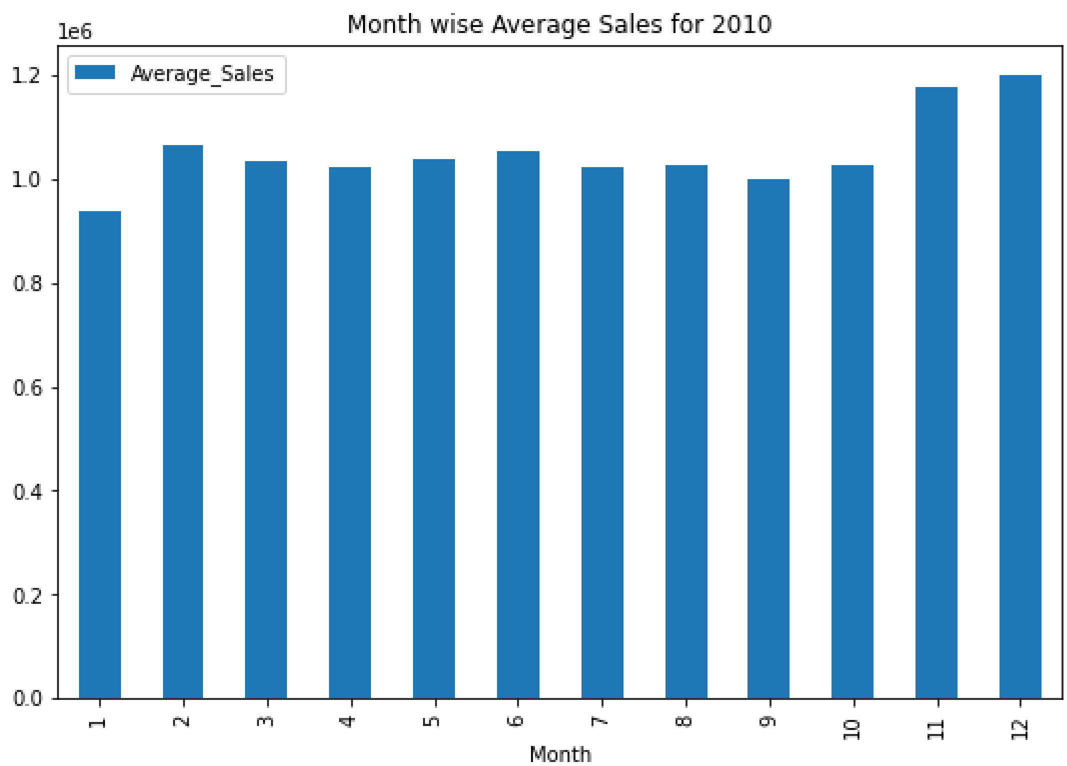
In [35]:
```
walmart_df.head()
```

Out[35]:

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

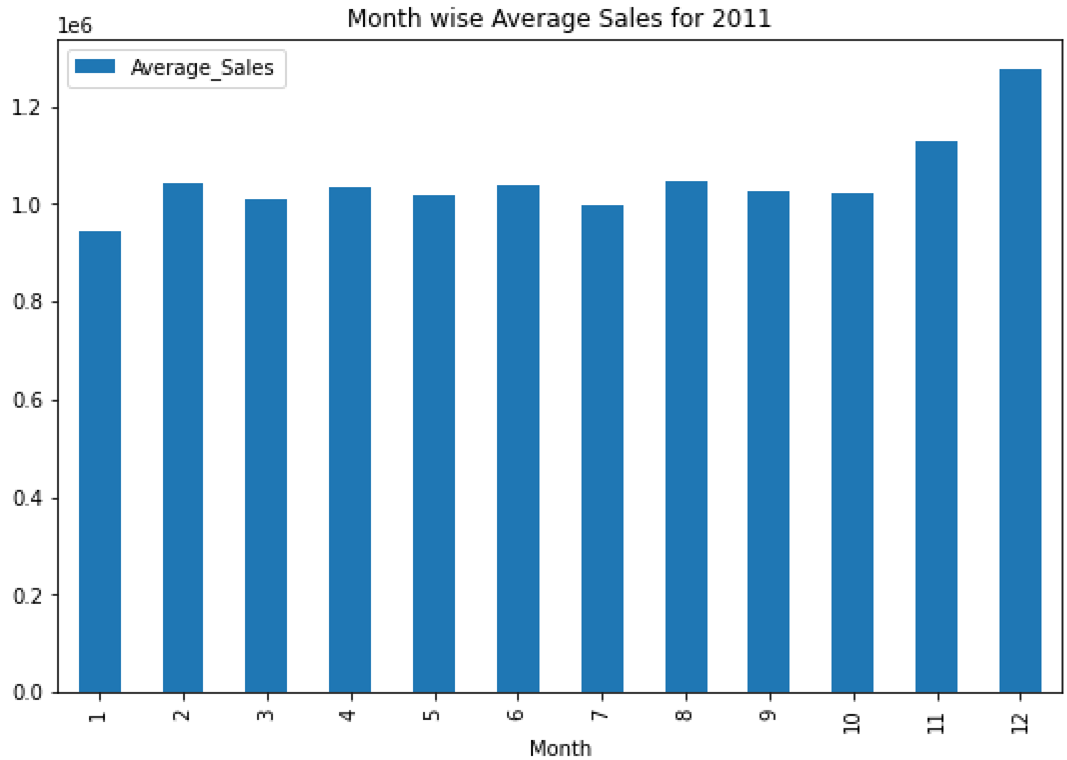
In [36]:

```
walmart_df[walmart_df['Year'] == 2010].groupby('Month')['Weekly_Sales'].agg(Average_Sa
plt.title("Month wise Average Sales for 2010")
plt.savefig("2010_Month_avg_sales.png")
plt.show()
```



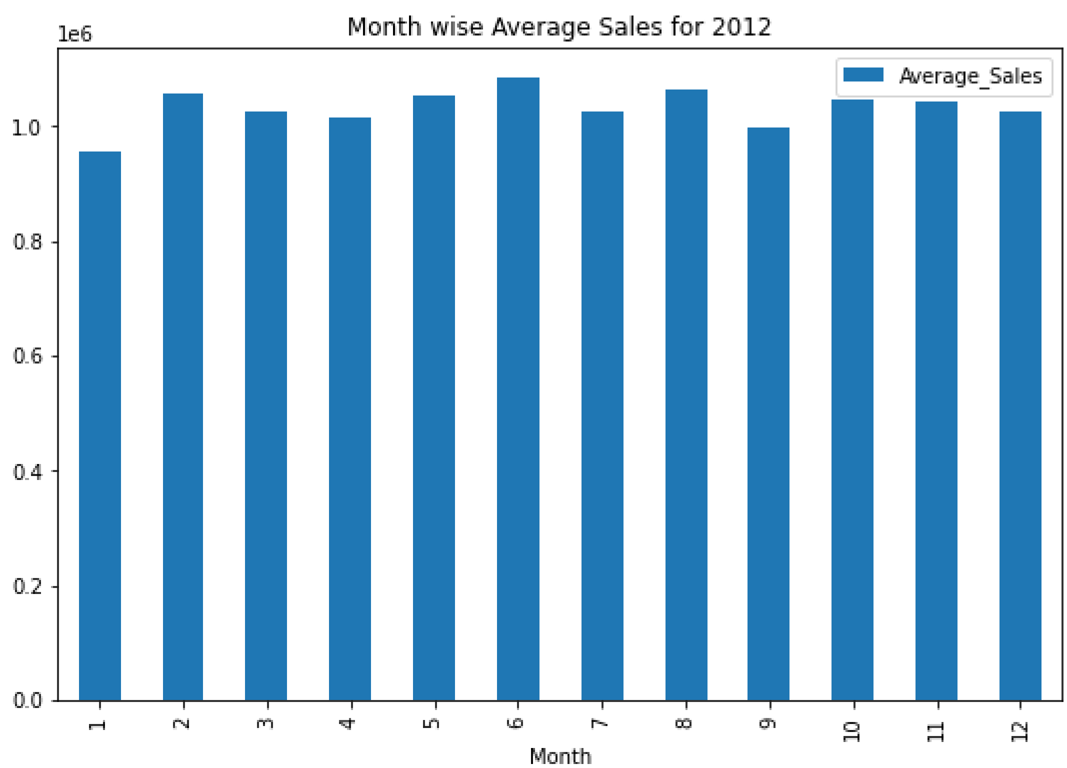
In [37]:

```
walmart_df[walmart_df['Year'] == 2011].groupby('Month')['Weekly_Sales'].agg(Average_Sa
plt.title("Month wise Average Sales for 2011")
plt.savefig("2011_Month_avg_sales.png")
plt.show()
```



In [38]:

```
walmart_df[walmart_df['Year'] == 2012].groupby('Month')['Weekly_Sales'].agg(Average_Sa
plt.title("Month wise Average Sales for 2012")
plt.savefig("2012_Month_avg_sales.png")
plt.show()
```



In 2010 and 2011 we see that Sales peak at the feb and nov and dec during the holiday season and superbowl

Statistical Model

In [39]:
```
walmart_df.head()
```

Out[39]:

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

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.

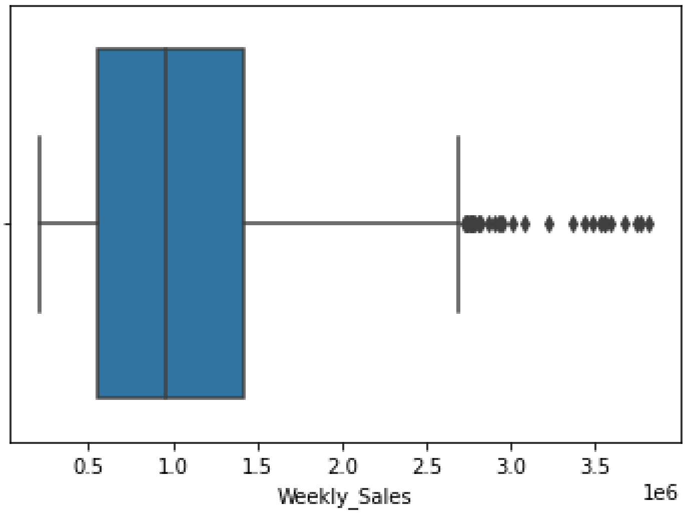
In [40]:
```
walmart_df['days'] = pd.DatetimeIndex(walmart_df['Date']).day
walmart_df.head()
```

Out[40]:

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

In [41]:
```
# Checking for outliers in target variable


sns.boxplot(walmart_df['Weekly_Sales'])
```

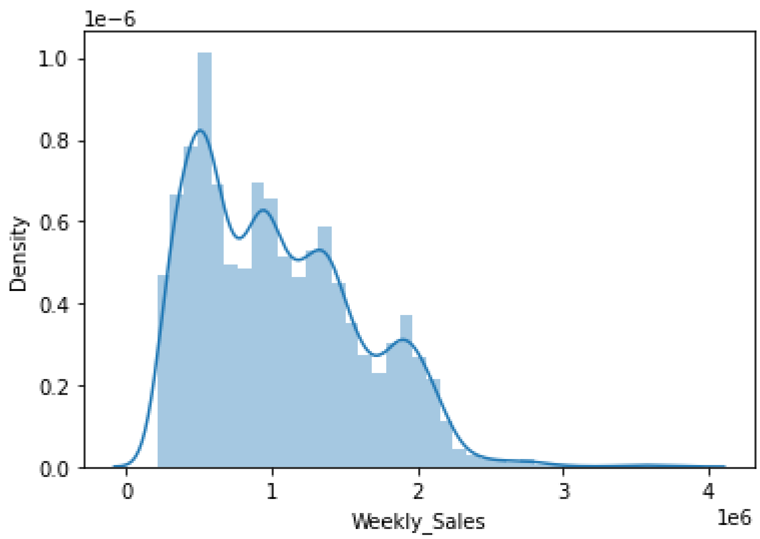
Out[41]:
```
<AxesSubplot:xlabel='Weekly_Sales'>
```



As we can see that there are many outliers that lie outside of the range

In [42]:
```
sns.distplot(walmart_df.Weekly_Sales)
```

Out[42]:
```
<AxesSubplot:xlabel='Weekly_Sales', ylabel='Density'>
```



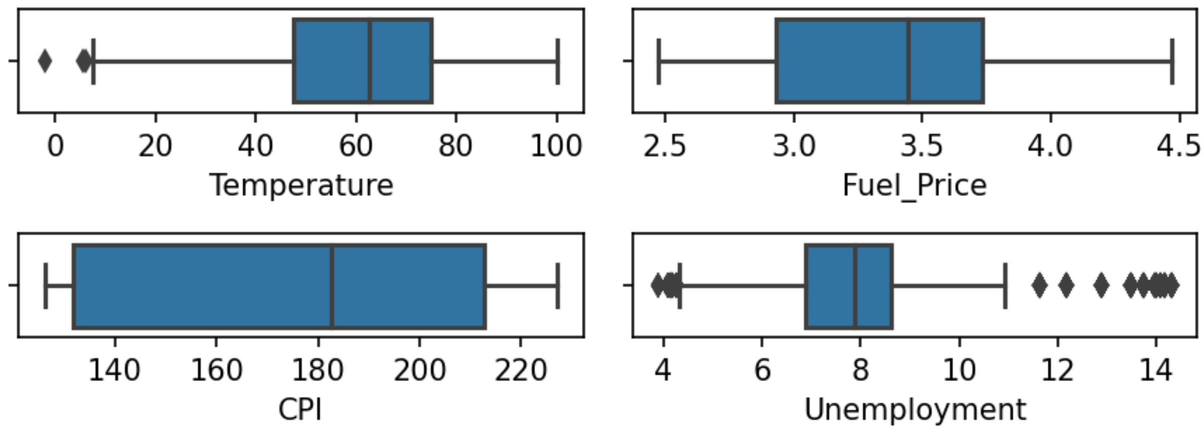
Now let's see the outliers in feature variables

In [43]:
```
walmart_df.columns
```

Out[43]:
```
Index(['Store', 'Date', 'Weekly_Sales', 'Holiday_Flag', 'Temperature',
       'Fuel_Price', 'CPI', 'Unemployment', 'Quarter', 'Month', 'Year',
       'days'],
      dtype='object')
```

In [44]:
```
features_list = 'Temperature, Fuel_Price, CPI, Unemployment'.split(", ")

plt.figure(dpi=150)
count = 1
for feature in features_list:
    plt.subplot(4,2,count)
    sns.boxplot(walmart_df[feature])
    count += 1
```

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



As we can see that there are many outliers in unemployment and temprature data which might affect the predictions Lets remove these outliers

In [45]:
```
# Removing outliers

def remove_out(feature):

    p25 = walmart_df[feature].quantile(0.25)
    p75 = walmart_df[feature].quantile(0.75)
    iqr = p75 - p25

    upper_limit = p75 + 1.5 * iqr
    lower_limit = p25 - 1.5 * iqr

    new_df = walmart_df[(walmart_df[feature] > lower_limit) & (walmart_df[feature] < u

    return new_df
```

In [46]:
```
for feature in features_list:
    walmart_df = remove_out(feature)
```

In [47]:
```
walmart_df.shape
```

Out[47]:
```
(5951, 12)
```

In [48]:
```
walmart_df.head()
```

Out[48]:

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

lets look at the correlation matrix

In [84]:
```
corr_matrix = walmart_df.corr()
```

In [85]:
```
corr_matrix['Weekly_Sales']
```

Out[85]:
```
Store          -0.322210
Weekly_Sales    1.000000
Holiday_Flag    0.036672
Temperature    -0.062210
Fuel_Price      0.011150
CPI            -0.087470
Unemployment   -0.074868
Month           0.066132
Year           -0.034154
days           -0.012670
Name: Weekly_Sales, dtype: float64
```

In [117…
```
features = 'Temperature, Fuel_Price, CPI, Unemployment, days, Holiday_Flag'.split(", '
target = 'Weekly_Sales'
```

lets seperate the dataset into predictor features and predictor

In [118…
```
X = walmart_df[features]
y = walmart_df[target]
```

In [119…
```
X
```

Out[119]:

	Temperature	Fuel_Price	CPI	Unemployment	days	Holiday_Flag
0	42.31	2.572	211.096358	8.106	2	0
1	38.51	2.548	211.242170	8.106	2	1
2	39.93	2.514	211.289143	8.106	19	0
3	46.63	2.561	211.319643	8.106	26	0
4	46.50	2.625	211.350143	8.106	3	0
...	...	...	...	...	...	...
6430	64.88	3.997	192.013558	8.684	28	0
6431	64.89	3.985	192.170412	8.667	10	0
6432	54.47	4.000	192.327265	8.667	10	0
6433	56.47	3.969	192.330854	8.667	19	0
6434	58.85	3.882	192.308899	8.667	26	0

5951 rows × 6 columns

Lets scale the features before putting it into the modeling

In [120…
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

sc.fit(X)

x_sc = sc.transform(X)
```

In [121…
```
x_sc
```

Out[121]:
```
array([[-0.97801381, -1.67807955,  0.92662167,  0.31023429, -1.56294503,
        -0.27485764],
       [-1.18438849, -1.73055089,  0.93035802,  0.31023429, -1.56294503,
         3.63824712],
       [-1.10726953, -1.80488529,  0.93156169,  0.31023429,  0.38166837,
        -0.27485764],
       ...,
       [-0.31761484,  1.44396518,  0.44567293,  0.76170611, -0.64783284,
        -0.27485764],
       [-0.20899659,  1.3761897 ,  0.4457649 ,  0.76170611,  0.38166837,
        -0.27485764],
       [-0.07974087,  1.18598109,  0.4452023 ,  0.76170611,  1.18239153,
        -0.27485764]])
```

Now lets split the data into train/test split

In [122…
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(x_sc, y, test_size=0.2, random_sta
```

In [123…
```
print(X_train.shape)
print(X_test.shape)
```

```
print(y_train.shape)
print(y_test.shape)
```

```
(4760, 6)
(1191, 6)
(4760,)
(1191,)
```

In [124…

```
X_train
```

Out[124]:

```
array([[ 0.85546227,  1.42210212, -1.17511779,  0.77780136,  0.49605739,
        -0.27485764],
       [ 0.65614778,  1.01544923,  0.85318808, -0.34886628,  0.95361348,
        -0.27485764],
       [ 0.61487285, -0.11924349,  1.09359113, -1.14316696, -0.64783284,
        -0.27485764],
       ...,
       [-0.98996182,  0.57818807, -0.98516911, -2.65612065,  1.41116958,
        -0.27485764],
       [ 0.33137921, -0.63302536, -1.24185282,  1.45138766,  0.83922446,
        -0.27485764],
       [-0.413742  , -1.45507636,  0.37113792,  0.99991584,  0.26727934,
        -0.27485764]])
```

In [125…

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

In [126…

```
linreg = LinearRegression()
```

In [127…

```
linreg.fit(X_train, y_train)
linreg_pred = linreg.predict(X_test)
```

In [128…

```
lin_mae = mean_absolute_error(y_test, linreg_pred)
lin_rmse = mean_squared_error(y_test, linreg_pred)
```

In [129…

```
print("Linear Regression MAE: {}" .format(lin_mae))
print("Linear Regression RMSE: {}" .format(lin_rmse))
linreg.score(X_train, y_train)
```

```
Linear Regression MAE: 489419.9102388284
Linear Regression RMSE: 343472748204.1259
```

Out[129]:

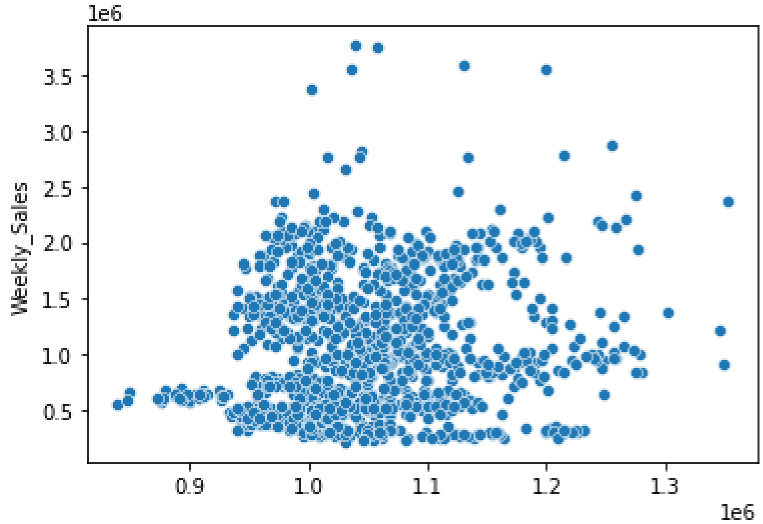
```
0.019367074288056618
```

In [130…

```
sns.scatterplot(linreg_pred, y_test)
```

Out[130]:

```
<AxesSubplot:ylabel='Weekly_Sales'>
```

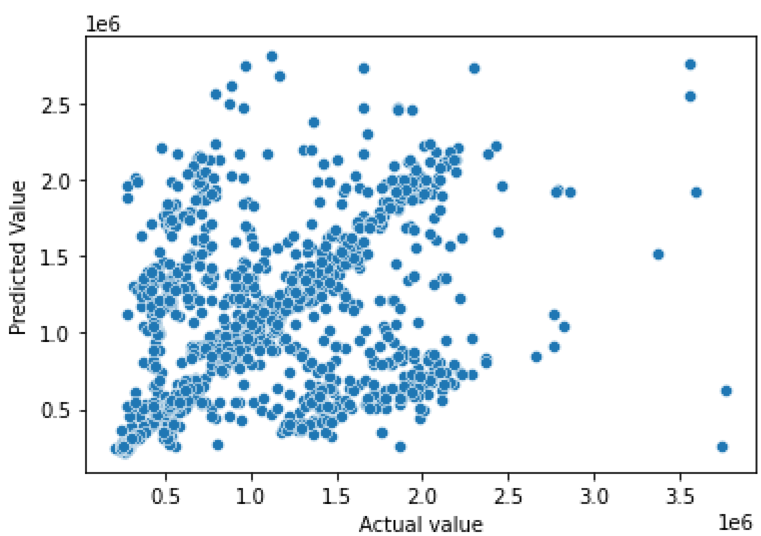


Now lets train the data on Decision Tree regressor

In [131…
```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor()

tree_reg.fit(X_train, y_train)
tree_pred = tree_reg.predict(X_test)

tree_mae = mean_absolute_error(y_test, tree_pred)
tree_rmse = mean_squared_error(y_test, tree_pred)

print("Decision Tree MAE: {}" .format(tree_mae))
print("Decision Tree RMSE: {}" .format(tree_rmse))
```

```
Decision Tree MAE: 416497.7151315421
Decision Tree RMSE: 408181632831.0881
```

In [134…
```
sns.scatterplot(x= y_test, y=tree_pred)
plt.xlabel("Actual value")
plt.ylabel("Predicted Value")
plt.show()
```



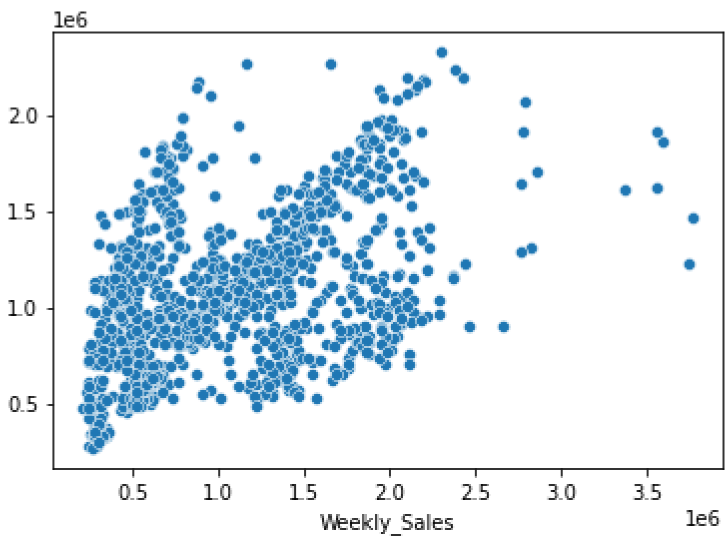
Decision tree performs a little bit better than normal linear regression. Let's try this problem with Random forest regressor

In [137…
```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor()

forest_reg.fit(X_train, y_train)
forest_pred = forest_reg.predict(X_test)

forest_mae = mean_absolute_error(y_test, forest_pred)
forest_rmse = mean_squared_error(y_test, forest_pred)

print("Decision Tree MAE: {}" .format(forest_mae))
print("Decision Tree RMSE: {}" .format(forest_rmse))
```

```
Decision Tree MAE: 383592.7955718071
Decision Tree RMSE: 292778826385.7607
```

In [138…
```
sns.scatterplot(x=y_test, y=forest_pred)
```

Out[138]: <AxesSubplot:xlabel='Weekly_Sales'>



As we saw that none of the regressor performed as well as we would like it to. but Decision tree and random forest does performed better than the Linear regression

In [ ]: