1st Part-

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

In [2]:
```
df=pd.read_csv('C:\Datasets\\Walmart_Store_sales.csv')
```

In [3]:
```
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

In [5]:
```
#changing the data type of the 'Date' column because it is an object type
from datetime import datetime
df['Date'] = pd.to_datetime(df['Date'])
```

In [5]:
```
df.dtypes
```

Out[5]:
```
Store                  int64
Date           datetime64[ns]
Weekly_Sales          float64
Holiday_Flag            int64
Temperature           float64
Fuel_Price            float64
CPI                   float64
Unemployment          float64
dtype: object
```

In [6]:
```
#Statistical Tasks-
```

In [7]:
```
#Which store has maximum sales?
total_sales=df.groupby('Store')['Weekly_Sales'].sum().round().sort_values(ascendi
```

In [8]:
```
pd.DataFrame(total_sales).head(1)
```

Out[8]:

Store	Weekly_Sales
20	301397792.0

In [9]:
```
#Which store has maximum standard deviation i.e., the sales vary a lot.
#Also, find out the coefficient of mean to standard deviation
```

In [10]:
```
df_std=df.groupby('Store')['Weekly_Sales'].std().round().sort_values(ascending=Fa
```

In [11]:
```
pd.DataFrame(df_std).head()
```

Out[11]:

Store	Weekly_Sales
14	317570.0
10	302262.0
20	275901.0
4	266201.0
13	265507.0

In [12]:
```
#Store which has maximum Standard Deviation
pd.DataFrame(df_std).head(1)
```

Out[12]:

Store	Weekly_Sales
14	317570.0

```
Store 14 has maximum Standard Deviation
```

In [13]:
```
#Coefficient of mean to standard deviation
```

In [14]:
```
store14=df[df.Store==14].Weekly_Sales
```

In [15]:
```
mean_to_stddev=store14.std()/store14.mean()*100
```

In [16]:
```
print(mean_to_stddev,'%')
```

```
15.713673600948338 %
```

In [17]:
```
#Which store/s has a good quarterly growth rate in Q3'2012?
```

In [18]:
```
#Finding the Q2 sales then Q3 sales,then taking out the difference to get the gro
q2_sales=df[(df['Date']>='2012-04-01') & (df['Date']<='2012-06-30')].groupby('Sto
```

In [19]:
```
q3_sales=df[(df['Date']>='2012-07-01') & (df['Date']<='2012-09-30')].groupby('Sto
```

In [20]:
```
#Growth rate = ((present-past)/past)*100
df_2012=pd.DataFrame({'Q2 Sales':q2_sales,'Q3 Sales':q3_sales,'Difference':(q3_sa
df_2012
```

Out[20]:

Store	Q2 Sales	Q3 Sales	Difference	Growth Rate %
16	6626133.0	6441311.0	-184822.0	-2.789289
7	7613594.0	7322394.0	-291200.0	-3.824738
35	10753571.0	10252123.0	-501448.0	-4.663084
26	13218290.0	12417575.0	-800715.0	-6.057629
39	20191586.0	18899955.0	-1291631.0	-6.396877

```
No store shown quaterly growth rate in Q3'2012, it can be seen from above
result that there is no growth rate in any of the stores.
```

In [21]:
```
#Some holidays have a negative impact on sales.Find out holidays that have higher
#sales than the mean sales in the non-holiday season for all stores together.
```

We have 4 Holiday Events, (1) Super Bowl: 12-Feb-10, 11-Feb-11, 10-Feb-12, 8-Feb-13, (2) Labour Day: 10-Sep-10, 9-Sep-11, 7-Sep-12, 6-Sep-13, (3) Thanksgiving: 26-Nov-10, 25-Nov-11, 23-Nov-12, 29-Nov-13, (4) Christmas: 31-Dec-10, 30-Dec-11, 28-Dec-12, 27-Dec-13.

In [22]:
```
#Calculating the holiday event sales of each of the events and then find the non-
```

In [6]:
```
#Holiday events
Super_Bowl=['12-02-2010','11-02-2011','10-02-2012','08-02-2013']
Labour_Day=['10-09-2010','09-09-2011','07-09-2012','06-09-2013']
Thanksgiving=['26-11-2010','25-11-2011','23-11-2012','29-11-2013']
Christmas=['31-12-2010','30-12-2011','29-12-2012','27-12-2013']
```

In [9]:
```
Super_Bowl_Sales =round(df[df.Date.isin(Super_Bowl)]['Weekly_Sales'].mean(),2)
Labour_Day_Sales =round(df[df.Date.isin(Labour_Day)]['Weekly_Sales'].mean(),2)
Thanksgiving_Sales =round(df[df.Date.isin(Thanksgiving)]['Weekly_Sales'].mean(),2
Christmas_Sales =round(df[df.Date.isin(Christmas)]['Weekly_Sales'].mean(),2)
```

In [8]:
```
Super_Bowl_Sales,Labour_Day_Sales,Thanksgiving_Sales,Christmas_Sales
```

Out[8]:
```
(1079127.99, 1042427.29, 1471273.43, 960833.11)
```

In [26]:
```
#Non-holiday Sales and Comparison
```

In [27]:
```
non_holiday_sales=round(df[df['Holiday_Flag']==0]['Weekly_Sales'].mean(),2)
non_holiday_sales
```

Out[27]:
```
1041256.38
```

In [28]:
```
pd.DataFrame([{'Super Bowl Sales':Super_Bowl_Sales,'Labour day Sales':Labour_Day
```

Out[28]:

	0
Super Bowl Sales	1079127.99
Labour day Sales	1042427.29
Thanksgiving Sales	1471273.43
Christmas Sales	960833.11
non holiday Sales	1041256.38

Thanksgiving has the highest sales (1,471,273.43) than non-holiday sales (1,041,256.38)

In [29]:
```
#Provide a monthly and semester view of sales in units and give insights.
```

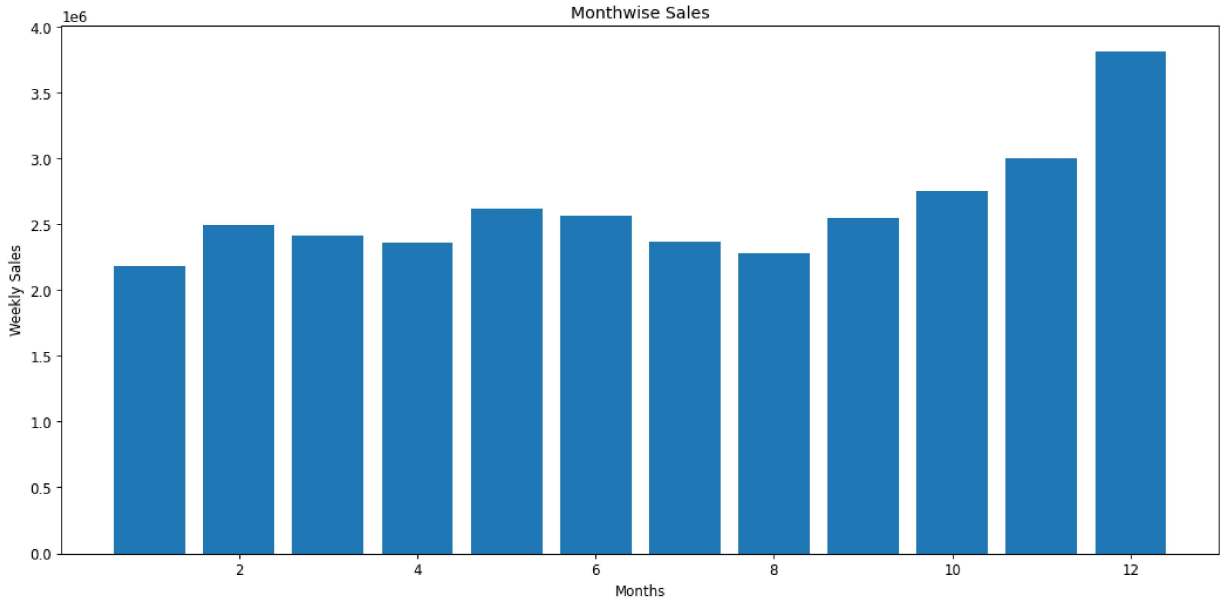
Plotting a month-wise bar graph for weekly sales to get an idea about which month has the maximum sales, then will plot the semester-wise bar graph for weekly sales to get some insights about the semester's weekly sales.

In [30]:
```
df['year'] = pd.DatetimeIndex(df['Date']).year
df['month'] = pd.DatetimeIndex(df['Date']).month
df['day'] = pd.DatetimeIndex(df['Date']).day
```

In [31]:
```
#Monthwise Sales
```

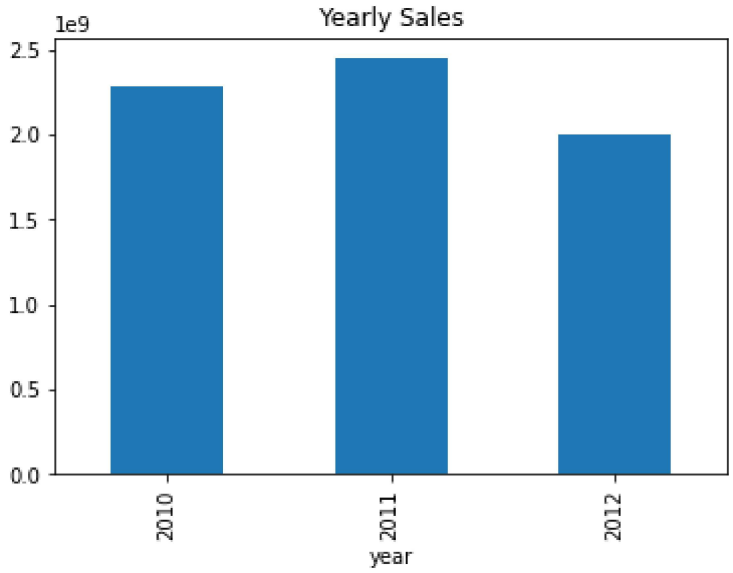
In [32]:
```
plt.figure(figsize=(15,7), dpi=85)
plt.bar(df['month'],df['Weekly_Sales'])
plt.xlabel('Months')
plt.ylabel('Weekly Sales')
plt.title('Monthwise Sales')
```

Out[32]: Text(0.5, 1.0, 'Monthwise Sales')



In [32]:
```
#Yearly Sales
plt.figure(figsize=(15,7), dpi=85)
df.groupby('year')[['Weekly_Sales']].sum().plot(kind='bar',legend=False)
plt.title('Yearly Sales')
```

Out[32]:
```
Text(0.5, 1.0, 'Yearly Sales')

<Figure size 1275x595 with 0 Axes>
```



In [33]:
```
#Semesterwise Sales
df['semester'] = np.where(df['month']< 7, 1, 2)
```

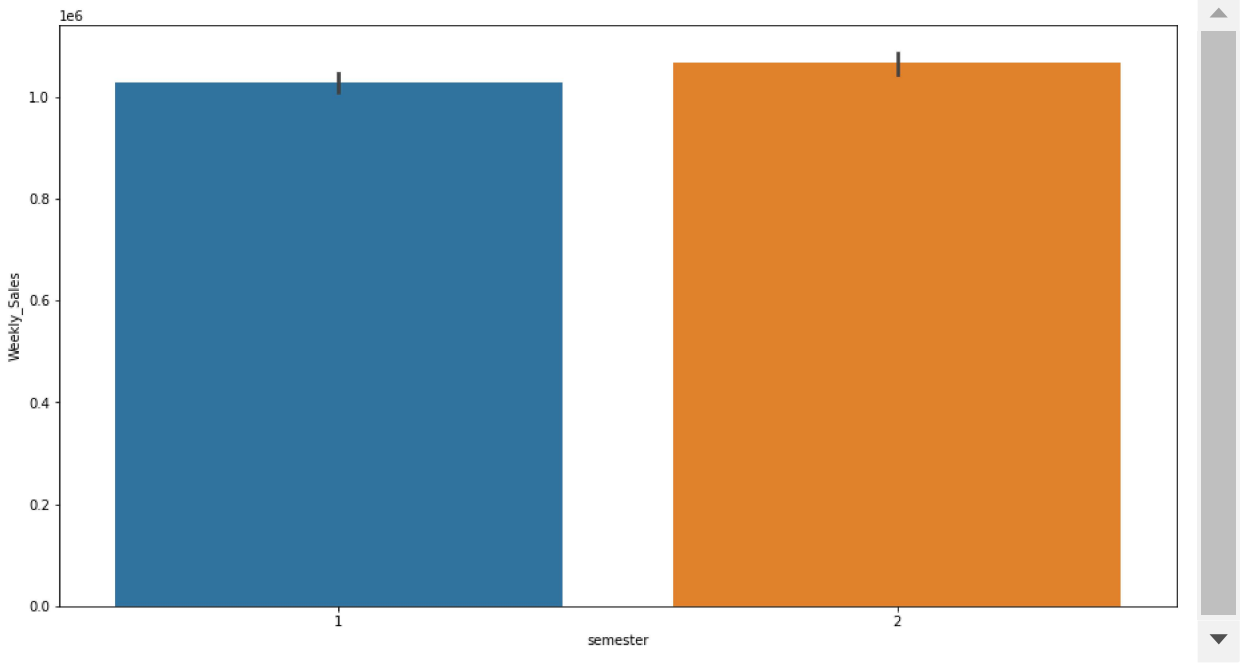
In [34]:
```
#Other method to define semester
if (df['month']<7).all():
    df['semester']==1
else:
    df['semester']==2
```

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

Out[35]:

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

In [36]:
```
plt.figure(figsize =(15,8))
semester=sns.barplot(x='semester',y='Weekly_Sales',data=df)
```



Insights drawn-(1)December month has the highest weekly sales. (2) Semester 2 has the highest weekly sales.

2nd Paert- Model Building: First, define dependent and independent variables. Here, store, fuel price, CPI, unemployment, day, month, and year are the independent variables and weekly sales is the dependent variable. Now, it's time to train the model. Import train_test_spit from sklearn.model_selection and train 80% of the data and test on the rest 20% of the data.

In [36]:
```
#Define independent and dependent variable
# Select features and target
x=df[['Store','Fuel_Price','CPI','Unemployment','day','month','year']]
y=df['Weekly_Sales']
```

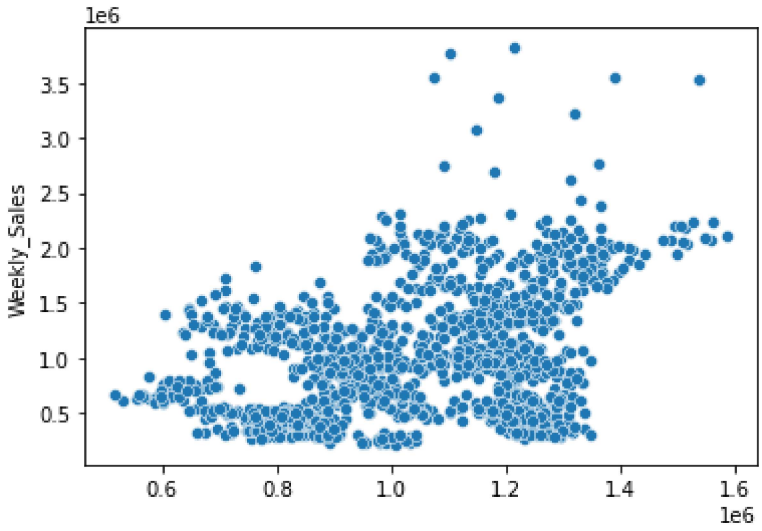
In [37]:
```
from sklearn.model_selection import train_test_split
# Split data to train and test (0.80:0.20)
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2)
```

In [38]:
```
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)
```

In [39]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.linear_model import LinearRegression
```

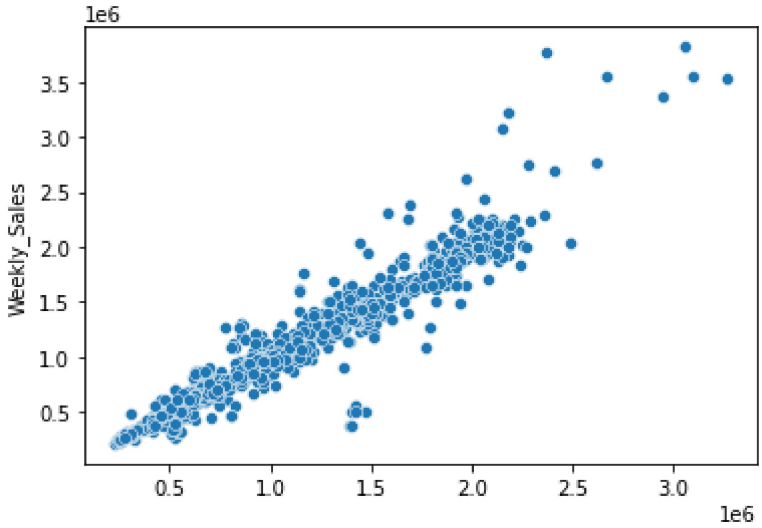
In [40]:
```
# Linear Regression model
print('Linear Regression:')
print()
reg = LinearRegression()
reg.fit(x_train, y_train)
y_pred = reg.predict(x_test)
print('Accuracy:',reg.score(x_train, y_train)*100)


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_pr



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

```
Linear Regression:

Accuracy: 15.04633118568217
Mean Absolute Error: 433358.9422569023
Mean Squared Error: 284394790158.9936
Root Mean Squared Error: 533286.7804089969

C:\Users\112987\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWa
rning: Pass the following variables as keyword args: x, y. From version 0.12, t
he only valid positional argument will be `data`, and passing other arguments w
ithout an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [41]:
```
# 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_pr


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

```
Random Forest Regressor:

Accuracy: 93.46251373530852
Mean Absolute Error: 77095.5425588606
Mean Squared Error: 21208761146.39598
Root Mean Squared Error: 145632.28057816022

C:\Users\112987\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWa
rning: Pass the following variables as keyword args: x, y. From version 0.12, t
he only valid positional argument will be `data`, and passing other arguments w
ithout an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



Here, we have used 2 different algorithms to know which model to use to predict the weekly sales. Linear Regression is not an appropriate model to use as accuracy is very low. However, Random Forest Regression gives an accuracy of almost 93%. so, it is the best model to forecast weekly sales.

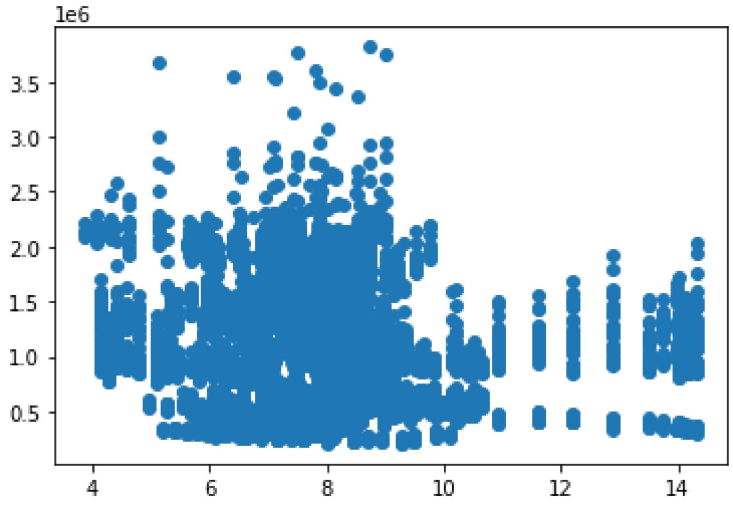
In [85]:
```
experiment_day_start=5
df['Date'] = pd.to_datetime(df['Date'], dayfirst=True)
df['exp_day'] = (df['Date']-df['Date'].min()).dt.days + experiment_day_start
```

In [86]:
```
df.head()
```

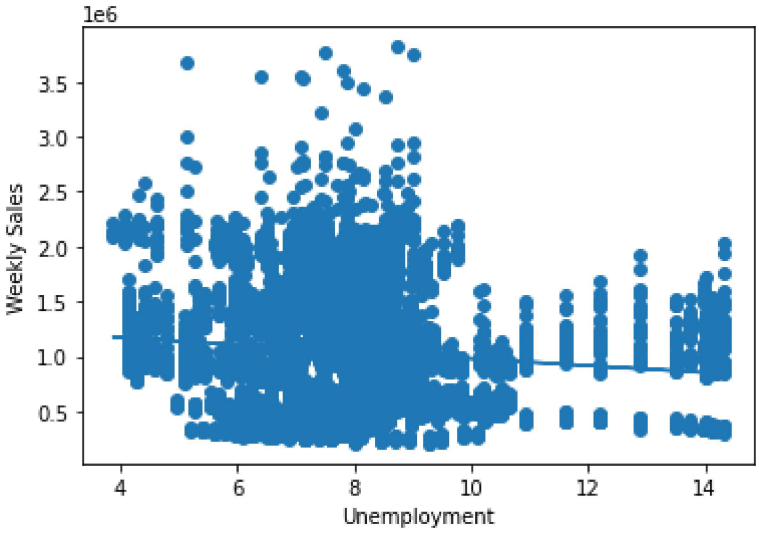
Out[86]:

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

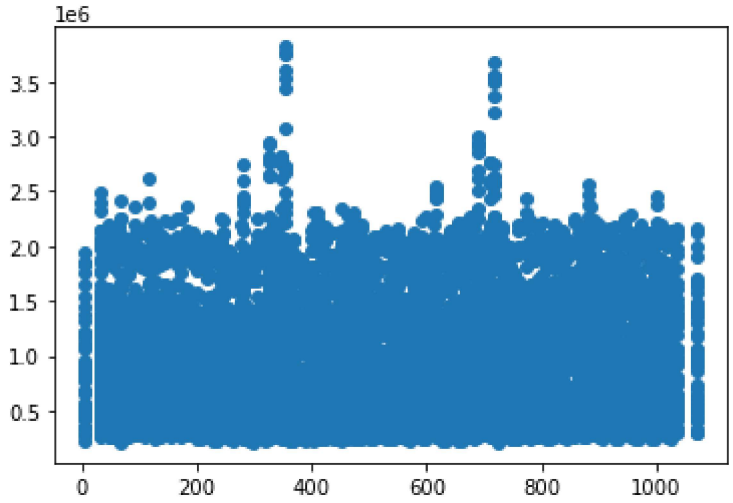
In [51]:
```
from sklearn.linear_model import LinearRegression
from scipy import stats
#Plotting the scatter plot for various relationships.
#Weekly sales vs Unemployment
x = df['Unemployment']
y = df['Weekly_Sales']
plt.scatter(x, y)
plt.show()
slope, intercept, r, p, std_err = stats.linregress(x, y)
print(r)# r should be between -1 to 1
def myfunc(x):
  return slope * x + intercept
mymodel = list(map(myfunc, x))
plt.scatter(x, y)
plt.xlabel('Unemployment')
plt.ylabel('Weekly Sales')
plt.plot(x, mymodel)
plt.show()
```



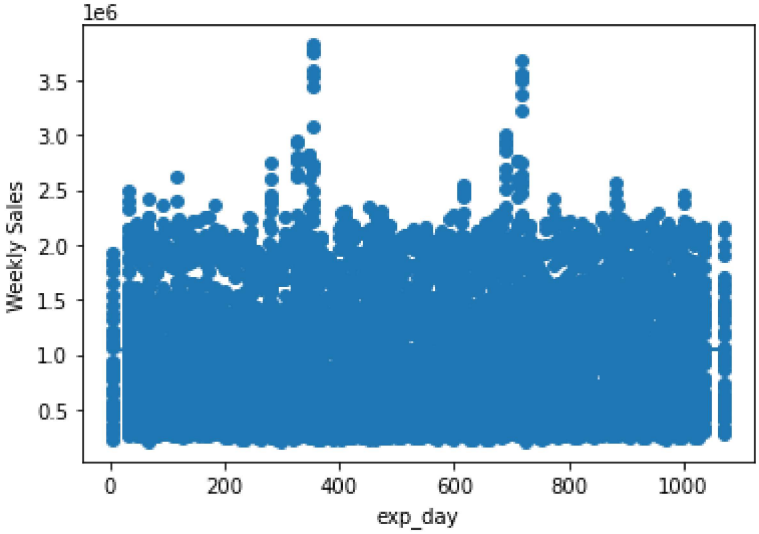

```
-0.10617608965795418
```



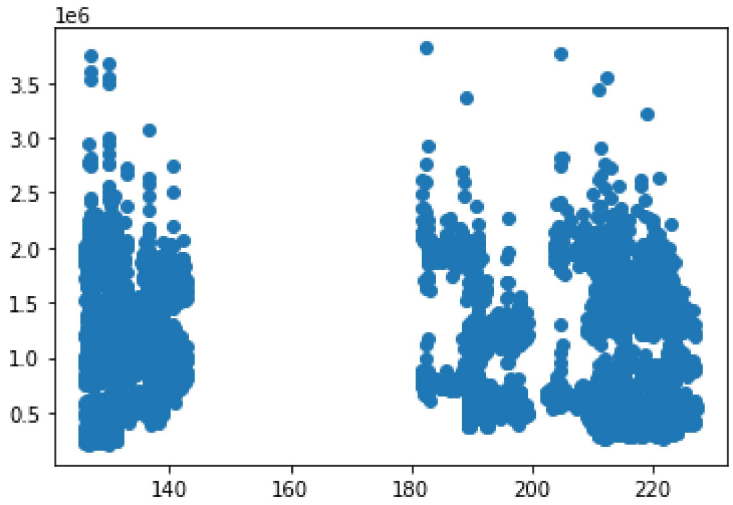
In [52]:
```
# Weekly_Sales vs exp_day
x = df['exp_day']
y = df['Weekly_Sales']
plt.scatter(x, y)
plt.show()
slope, intercept, r, p, std_err = stats.linregress(x, y)
print(r)# r should be between -1 to 1
def myfunc(x):
  return slope * x + intercept
mymodel = list(map(myfunc, x))
plt.scatter(x, y)
plt.xlabel('exp_day')
plt.ylabel('Weekly Sales')
plt.plot(x, mymodel)
plt.show()
```



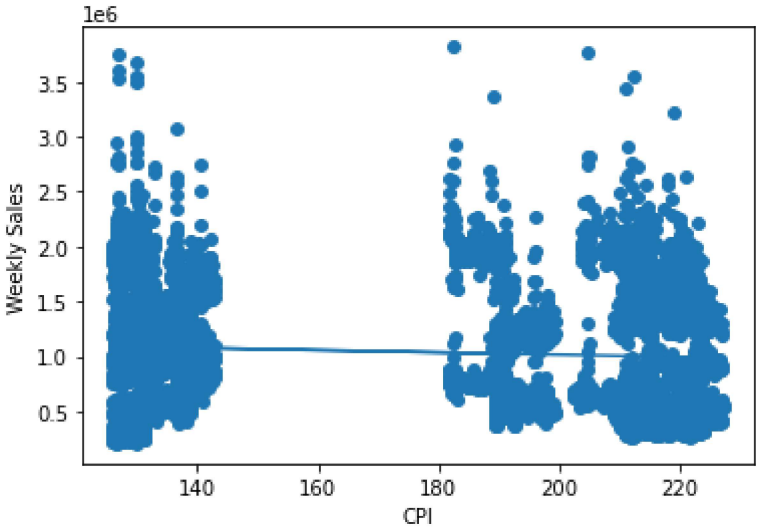

```
0.004591803306455495
```



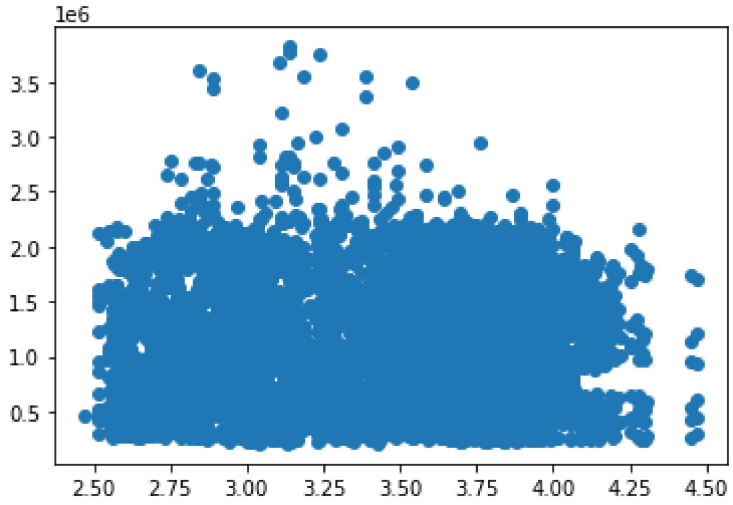
In [53]:
```
#Weekly sales vs CPI
x = df['CPI']
y = df['Weekly_Sales']
plt.scatter(x, y)
plt.show()
slope, intercept, r, p, std_err = stats.linregress(x, y)
print(r)# r should be between -1 to 1
def myfunc(x):
  return slope * x + intercept
mymodel = list(map(myfunc, x))
plt.scatter(x, y)
plt.xlabel('CPI')
plt.ylabel('Weekly Sales')
plt.plot(x, mymodel)
plt.show()
```



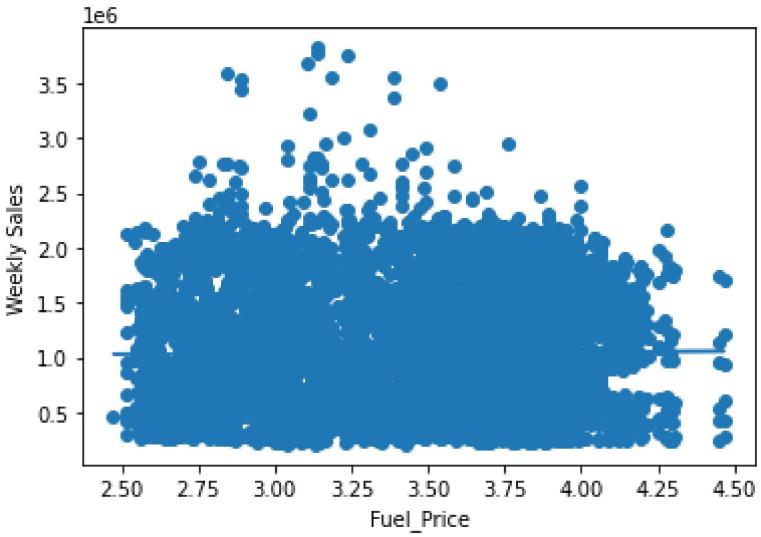

```
-0.07263416204017632
```



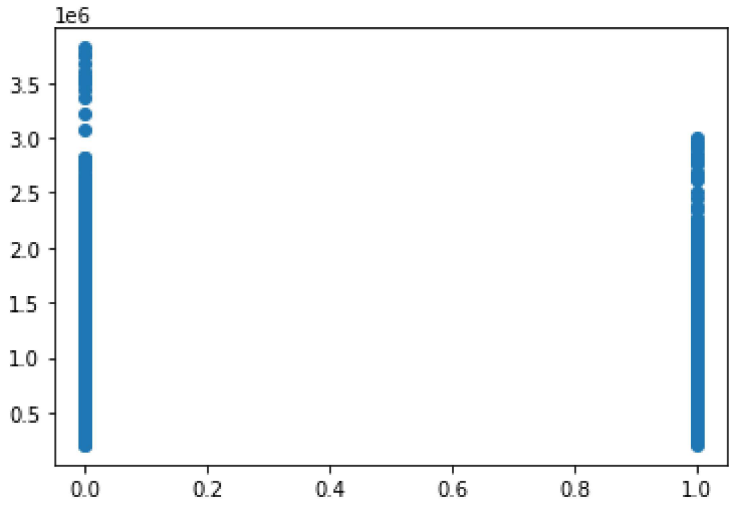
In [54]:
```
#Weekly sales vs Fuel price
x = df['Fuel_Price']
y = df['Weekly_Sales']
plt.scatter(x, y)
plt.show()
slope, intercept, r, p, std_err = stats.linregress(x, y)
print(r)# r should be between -1 to 1
def myfunc(x):
  return slope * x + intercept
mymodel = list(map(myfunc, x))
plt.scatter(x, y)
plt.xlabel('Fuel_Price')
plt.ylabel('Weekly Sales')
plt.plot(x, mymodel)
plt.show()
```



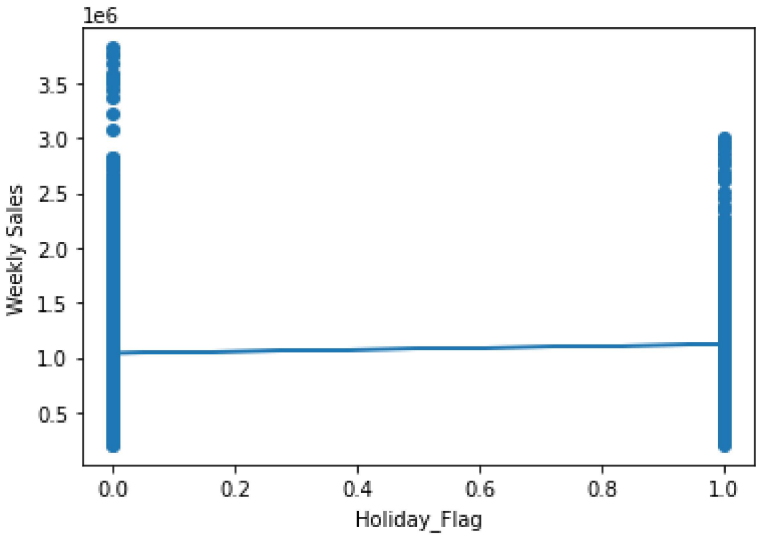

```
0.009463786314475123
```



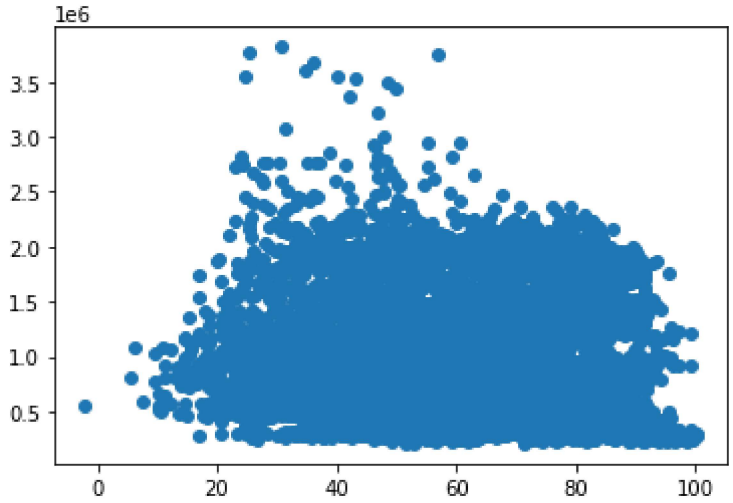
In [55]:
```
#Weekly sales vs Holidays
x = df['Holiday_Flag']
y = df['Weekly_Sales']
plt.scatter(x, y)
plt.show()
slope, intercept, r, p, std_err = stats.linregress(x, y)
print(r)# r should be between -1 to 1
def myfunc(x):
  return slope * x + intercept
mymodel = list(map(myfunc, x))
plt.scatter(x, y)
plt.xlabel('Holiday_Flag')
plt.ylabel('Weekly Sales')
plt.plot(x, mymodel)
plt.show()
```



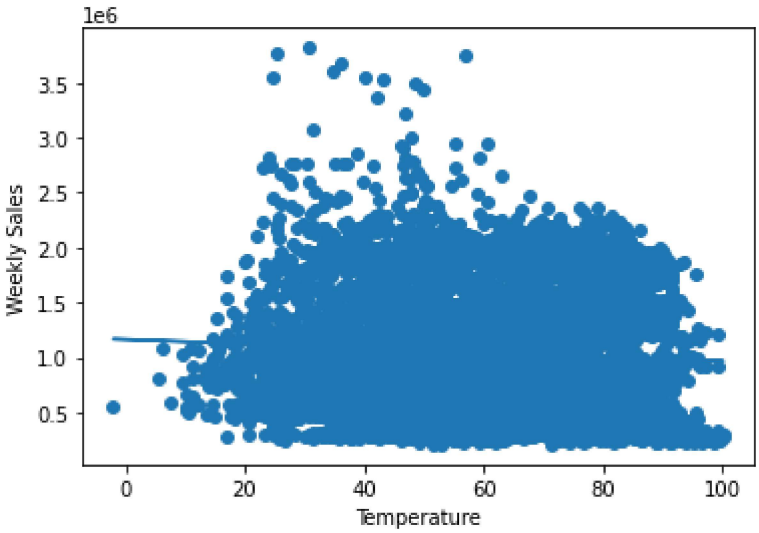

```
0.03689096801041456
```



In [56]:
```
#Weekly sales vs Temperature
x = df['Temperature']
y = df['Weekly_Sales']
plt.scatter(x, y)
plt.show()
slope, intercept, r, p, std_err = stats.linregress(x, y)
print(r)# r should be between -1 to 1
def myfunc(x):
  return slope * x + intercept
mymodel = list(map(myfunc, x))
plt.scatter(x, y)
plt.xlabel('Temperature')
plt.ylabel('Weekly Sales')
plt.plot(x, mymodel)
plt.show()
```




```
-0.06381001317946962
```



In the above diagram, we have seen the relationship between: 1.Weekly sales vs Unemployment 2.Weekly_Sales vs exp_day 3.Weekly sales vs CPI 4.Weekly sales vs Fuel price 5.Weekly sales

vs Holidays 6.Weekly sales vs Temperature

In [57]:
```
#Change dates into days by creating new variable.

df['day'] = pd.to_datetime(df['Date']).dt.day_name()
df.head()
```

Out[57]:

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

In [93]:
```
df['day'].head(10)
```

Out[93]:
```
0      Sunday
1    Thursday
2      Friday
3      Friday
4      Monday
5      Friday
6      Friday
7      Friday
8    Thursday
9    Saturday
Name: day, dtype: object
```

In [94]:
```

```