Retail Analysis with Walmart Data

1. Project Motivation
2. Installation
3. Data
4. Implementation
5. Results

1. Project Motivation:

In this project we focused retail analysis with Walmart data and answer the following questions:

1. Which stores have maximum and sales?
2. Which store has maximum standard deviation i.e., the sales vary a lot?. Also, find out the coefficient of mean to standard deviation.
3. Which store/s has good quarterly growth rate in Q3'2012?
4. Find out holidays which have higher sales than the mean sales in non-holiday season for all stores together.
5. Provide a monthly and semester view of sales in units and give insights.
6. Build prediction to forecast demand.

2. Installation:

- Python versions 3.*.
- Python Libraries:
  - Sklearn.
  - Pandas.
  - Numpy.
  - Seaborn.
  - Matplotlib.
  - datetime.

3. Data:

The data is provided by Simplilearn. It consist of sales data for 45 stores of Walmart. This data covers sales from 2010-02-05 to 2012-11-01.

4. Implementation:

In this project, we used RandomForestRegressor and LinearRegression to predict of sales. The data have been split into training and testing with a ratio of 80:20.

5. Result:

The details of the results show in the code.

Data Preparation:

```
#importing data
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import dates
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline

#Reading CSV
walmart = pd.read_csv("Walmart_Store_sales.csv")

walmart.head()

# Convert date to datetime format and show dataset information
walmart['Date'] = pd.to_datetime(walmart['Date'])
walmart.info()

#finding missing values
walmart.isna().sum()


# Splitting Date and create new columns (Day, Month, and Year)
# Change dates into days by creating new variable.

walmart['Day'] = pd.DatetimeIndex(walmart['Date']).day
walmart['Month'] = pd.DatetimeIndex(walmart['Date']).month
walmart['Year'] = pd.DatetimeIndex(walmart['Date']).year

walmart.head()
```

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import dates
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

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

Retail Analysis with Walmart Data

In [3]:
```
walmart.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 [4]:
```
# Convert date to datetime format and show dataset information
walmart['Date'] = pd.to_datetime(walmart['Date'])
walmart.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
```

In [5]:
```
#finding missing values
walmart.isna().sum()
```

Out[5]:

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

In [6]:
```
# Splitting Date and create new columns (Day, Month, and Year)
# Change dates into days by creating new variable.

walmart['Day'] = pd.DatetimeIndex(walmart['Date']).day
walmart['Month'] = pd.DatetimeIndex(walmart['Date']).month
walmart['Year'] = pd.DatetimeIndex(walmart['Date']).year

walmart.head()
```

Out[6]:

	Store	Date	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	Day	Month	Year
0	1	2010-05-02	1643690.90	0	42.31	2.572	211.096358	8.106	2	5	2010
1	1	2010-12-02	1641957.44	1	38.51	2.548	211.242170	8.106	2	12	2010
2	1	2010-02-19	1611968.17	0	39.93	2.514	211.289143	8.106	19	2	2010
3	1	2010-02-26	1409727.59	0	46.63	2.561	211.319643	8.106	26	2	2010
4	1	2010-05-03	1554806.68	0	46.50	2.625	211.350143	8.106	3	5	2010

Q.1: Which store has maximum sales ?

```
total_sales = walmart.groupby('Store')['Weekly_Sales'].sum().sort_values()

plt.figure(figsize=(15,12))
ax = total_sales.plot(kind='bar')


# 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')

plt.xticks(rotation = 0)
plt.xlabel('Stores')
plt.ylabel('Total Sales')
plt.title('Total Number of Sales')
plt.show()


#Insights:
#          We can clearly see that store number 20 has the maximum sales (30,13,97,792.46 $)
```

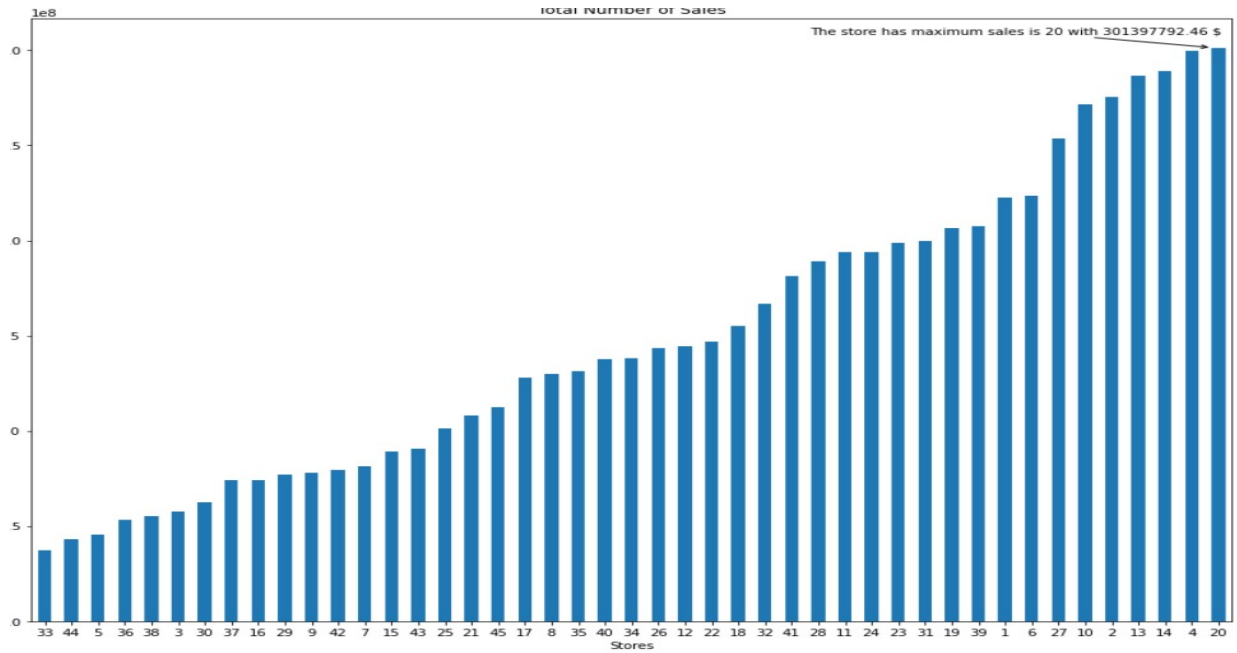

Fig: Show the total number of Sales

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

```
walmart_std = pd.DataFrame(walmart.groupby('Store')['Weekly_Sales'].std().sort_values(ascending = False))
walmart_std['Weekly_Sales'] = round(walmart_std['Weekly_Sales'],0)

print(f'The maximum standard deviation is in store number {walmart_std.index[0]} with {(walmart_std.head(1).Weekly_Sales[walmart_std.head(1).index[0]])}$')

walmart_std.head()

plt.figure(figsize=(10,6))

sns.distplot(walmart[walmart['Store'] == walmart_std.index[0]]['Weekly_Sales'])
plt.title("Sales distribution for store no. 14")
plt.xlabel("Weekly Sales")
plt.show()
```

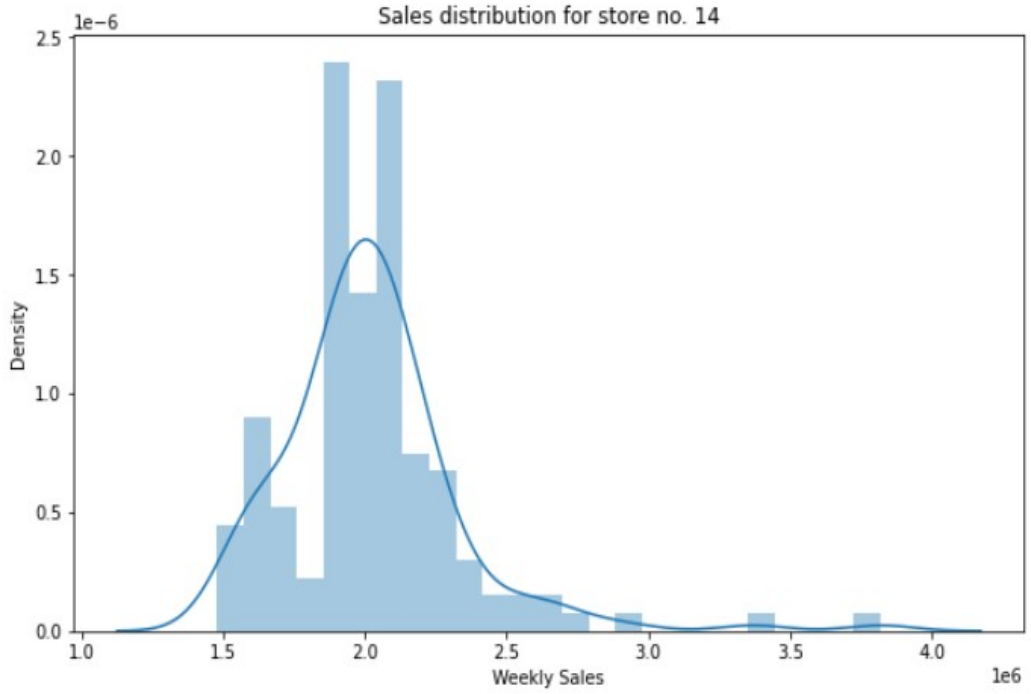

Fig: Sales distribution of store number 14

```
#finding out the coefficient of mean to standard deviation
walmart_coeff =
pd.DataFrame(walmart.groupby('Store')['Weekly_Sales'].std() /
walmart.groupby('Store')['Weekly_Sales'].mean())
walmart_coeff =
walmart_coeff.rename(columns={'Weekly_Sales':"Coefficient"})
walmart_coeff

#Distribution of Store which has maximum co-efficient of mean to standard
deviation
walmart_coeff_max =
walmart_coeff.sort_values(by='Coefficient',ascending=False)

#plot properties
plt.figure(figsize=(15,7))
sns.distplot(walmart[walmart['Store'] ==
walmart_coeff_max.head(1).index[0]]['Weekly_Sales'])
plt.title('The Sales Distribution of Store
#'+str(walmart_coeff_max.head(1).index[0]));
plt.xlabel('Weekly-Sales')
plt.show()
```

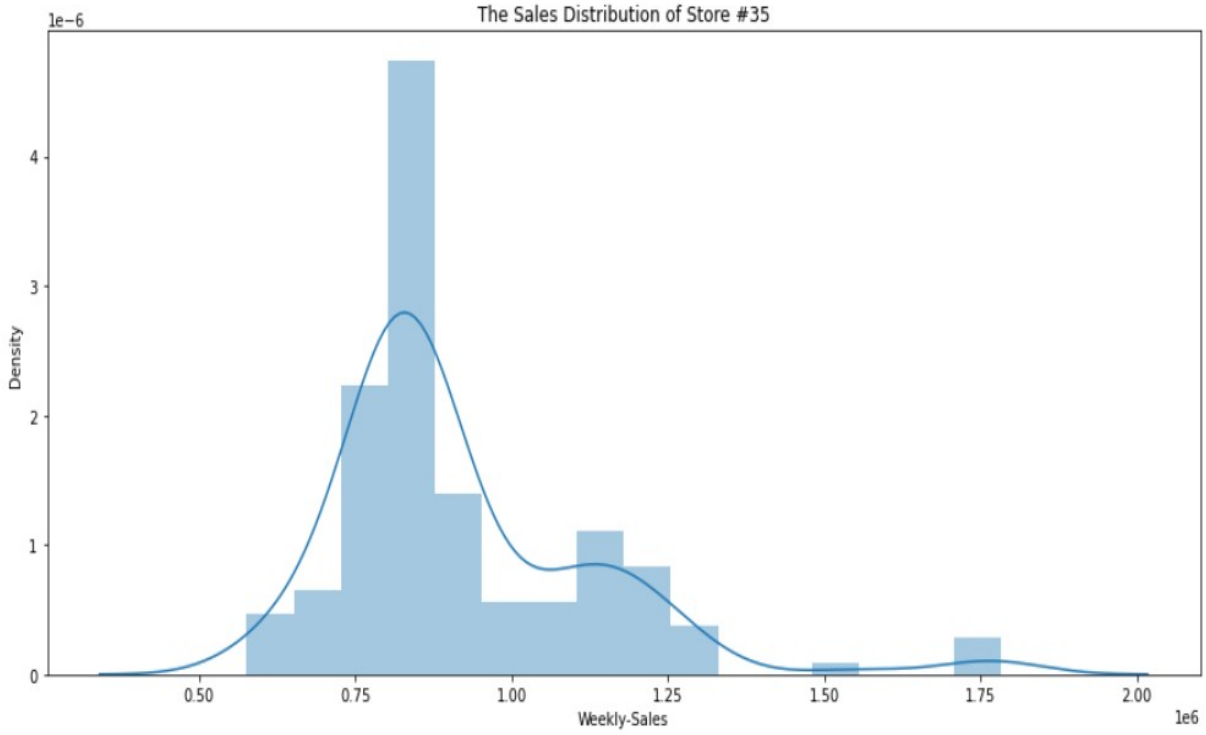

Fig: The Sales distribution of Store number 35

Insight:
Store with good quarterly growth rate in Q3’2012 4 with 25652119.35$

Q.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

```
def plot_line(walmart,holiday_dates,holiday_label):
    fig, ax = plt.subplots(figsize = (15,5))
    ax.plot(walmart['Date'],walmart['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 = walmart['Date'].dt.strftime('%Y-%m-
%d').sort_values().unique()
    xfmt = dates.DateFormatter('%d-%m-%y')
    ax.xaxis.set_major_formatter(xfmt)
    ax.xaxis.set_major_locator(dates.DayLocator(1))
    plt.gcf().autofmt_xdate(rotation=90)
    plt.show()


total_sales = walmart.groupby('Date')['Weekly_Sales'].sum().reset_index()
Super_Bowl = ['12-2-2010', '11-2-2011', '10-2-2012', '8-2-2013']
Labour_Day = ['10-9-2010', '9-9-2011', '7-9-2012', '6-9-2013']
Thanksgiving = ['26-11-2010', '25-11-2011', '23-11-2012', '29-11-2013']
Christmas = ['31-12-2010', '30-12-2011', '28-12-2012', '27-12-2013']


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')
```

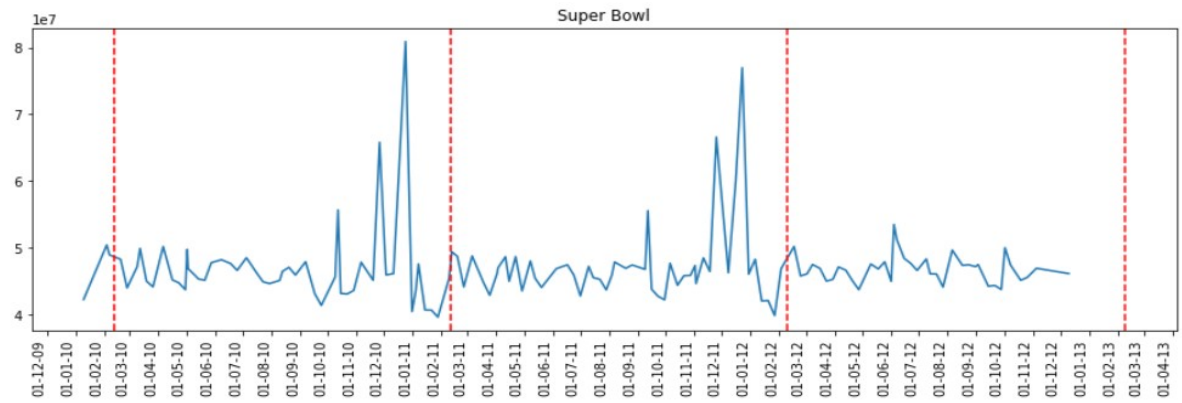

Fig: Sales during Super Bowl

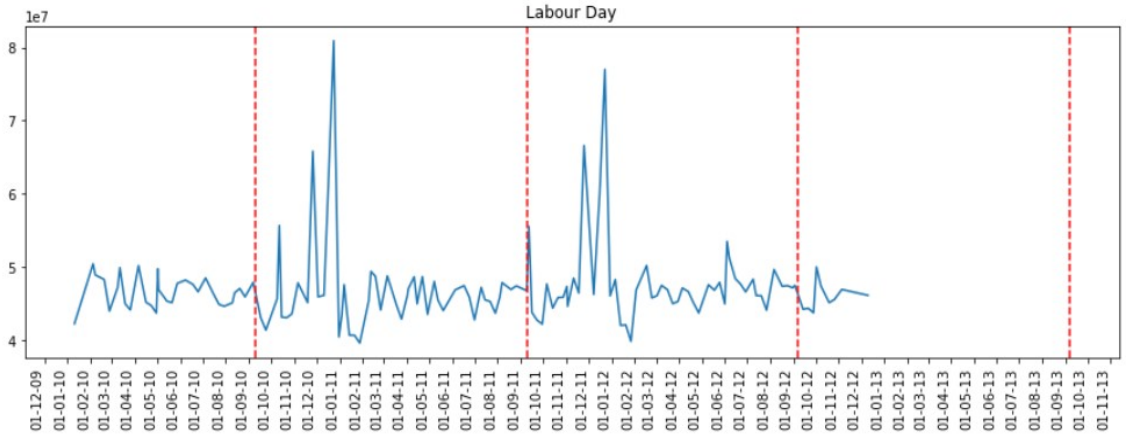

Fig: Sales during Labour Day

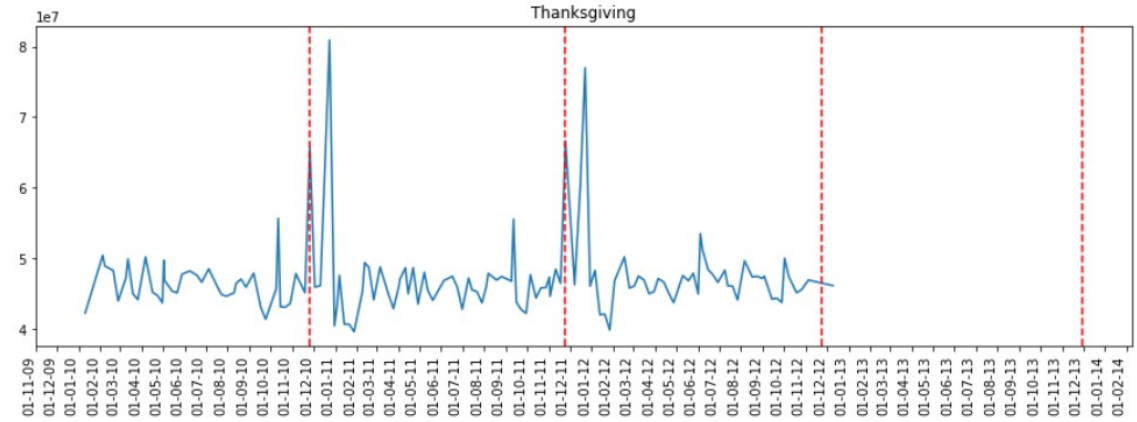

Fig: Sales during Thanksgiving

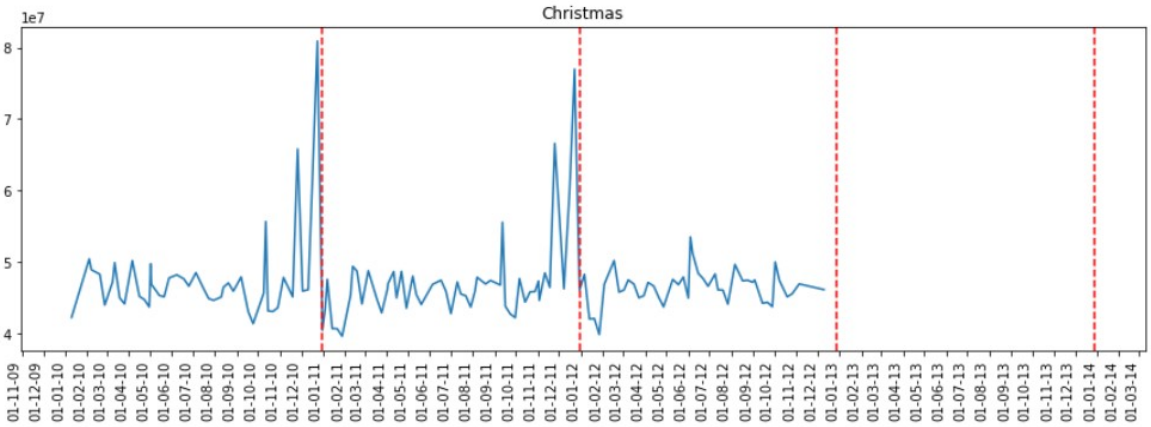

Fig: Sales During Christmas

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

```
Super_bowl_df =
pd.DataFrame(walmart.loc[walmart.Date.isin(Super_Bowl)].groupby('Year')['
Weekly_Sales'].sum())
Labour_Day_df =
pd.DataFrame(walmart.loc[walmart.Date.isin(Labour_Day)].groupby('Year')['
Weekly_Sales'].sum())
Thanksgiving_df =
pd.DataFrame(walmart.loc[walmart.Date.isin(Thanksgiving)].groupby('Year')['
Weekly_Sales'].sum())
Christmas_df =
pd.DataFrame(walmart.loc[walmart.Date.isin(Christmas)].groupby('Year')['We
ekly_Sales'].sum())

Super_bowl_df.plot(kind='bar',legend = False,title = "Yearly sales during
Super bowl holiday")
Thanksgiving_df.plot(kind='bar',legend = False,title = "Yearly sales during
Labour Day holiday")
Labour_Day_df.plot(kind='bar',legend = False,title = "Yearly sales during
Thanksgiving holiday")
Christmas_df.plot(kind='bar',legend = False,title = "Yearly sales during
Christmas holiday")
```

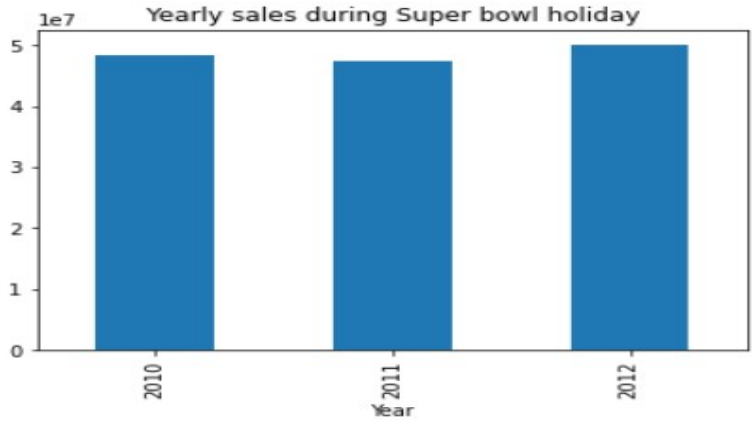

Fig: Yearly sales during Super bowl holiday

Fig: Yearly sales during labour day holiday

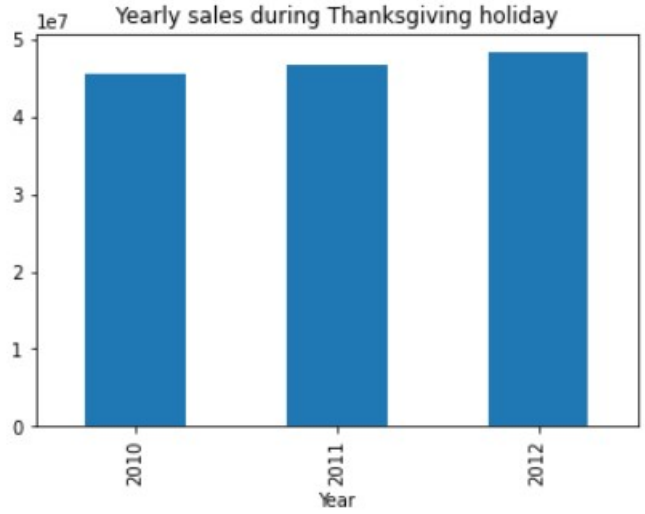

Fig: Yearly sales during Thanksgiving holiday

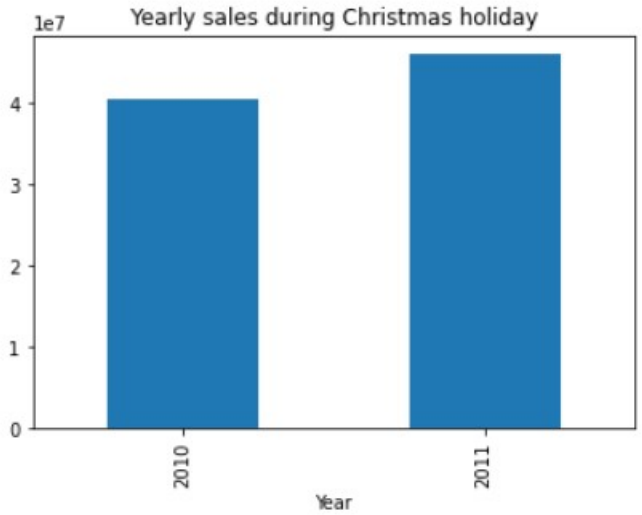

Fig: Yearly sales during Christmas Holiday

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

```
plt.figure(figsize=(8,5))
plt.scatter(walmart[walmart.Year == 2010]['Month'],walmart[walmart.Year
== 2010]["Weekly_Sales"])
plt.title('Montly view of Sales in 2010')
plt.xlabel('Months')
plt.ylabel('Weekly sales')
plt.show()

plt.figure(figsize=(8,5))
plt.scatter(walmart[walmart.Year == 2011]['Month'],walmart[walmart.Year
== 2011]['Weekly_Sales'])
plt.title('Monthly view of sales in 2011')
plt.xlabel('Months')
plt.ylabel('Weekly Sales')
plt.show()


plt.figure(figsize=(8,5))
plt.scatter(walmart[walmart.Year == 2012]['Month'], walmart[walmart.Year
== 2012]['Weekly_Sales'])
plt.title("Monthly view of sales in 2012")
plt.xlabel("Months")
plt.ylabel("Weekly sales")
plt.show()
```

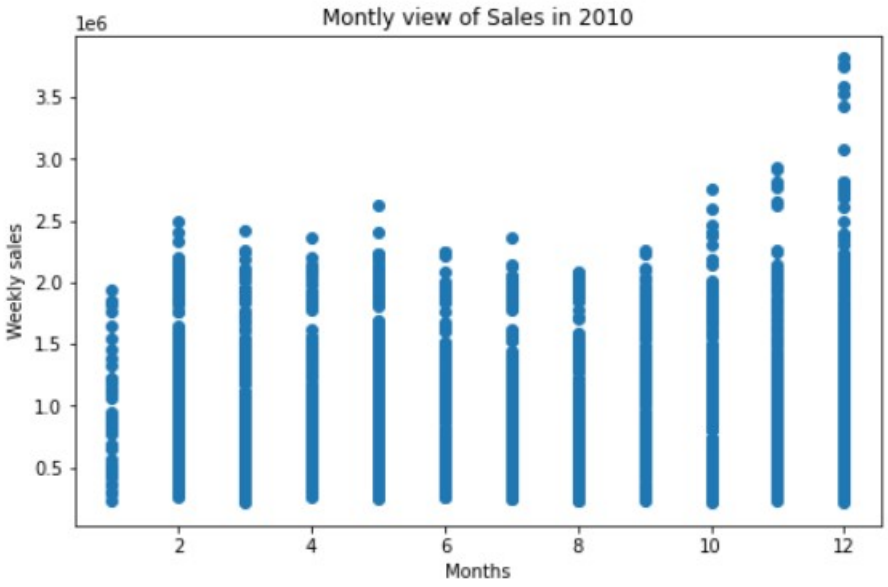

Fig: Montly view of Sales in 2010

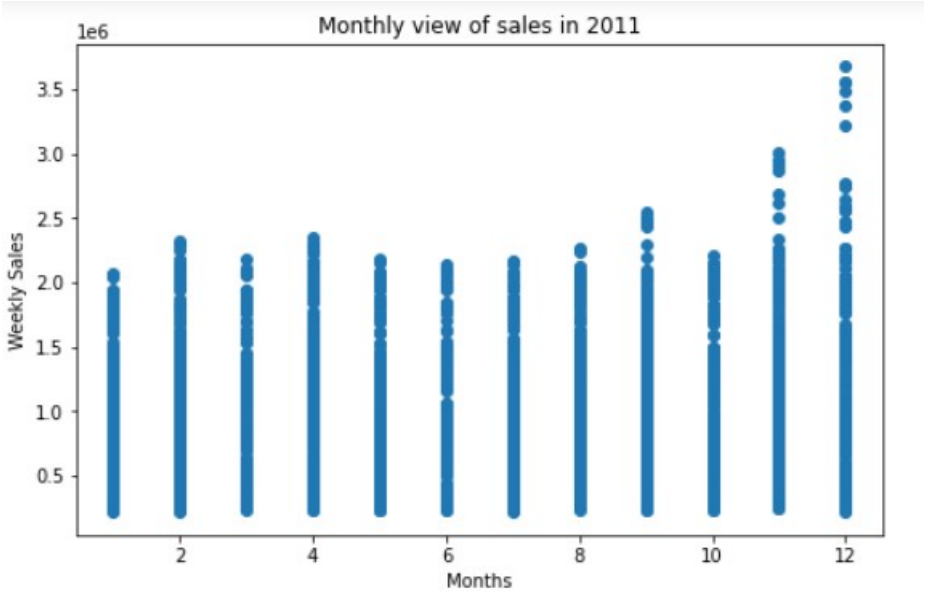

Fig: Monthly view of Sales in 2011

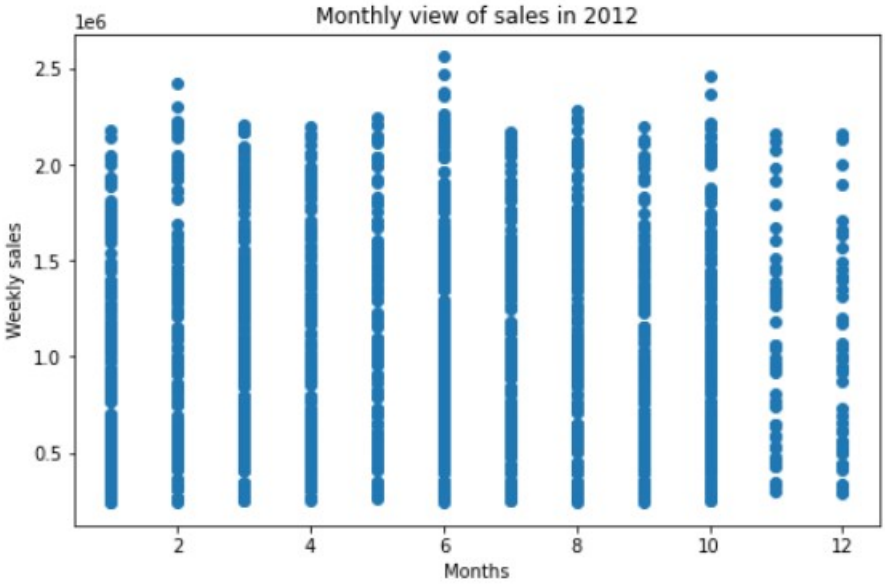

Fig: Monthly view of sales in 2012

```
#Monthly view of Sales in all year

plt.figure(figsize=(15,8))
plt.bar(walmart['Month'],walmart['Weekly_Sales'])
plt.xlabel("Months")
plt.ylabel('Weekly Sales')
plt.title('Monthly view of Sales in all year')
plt.show()
```

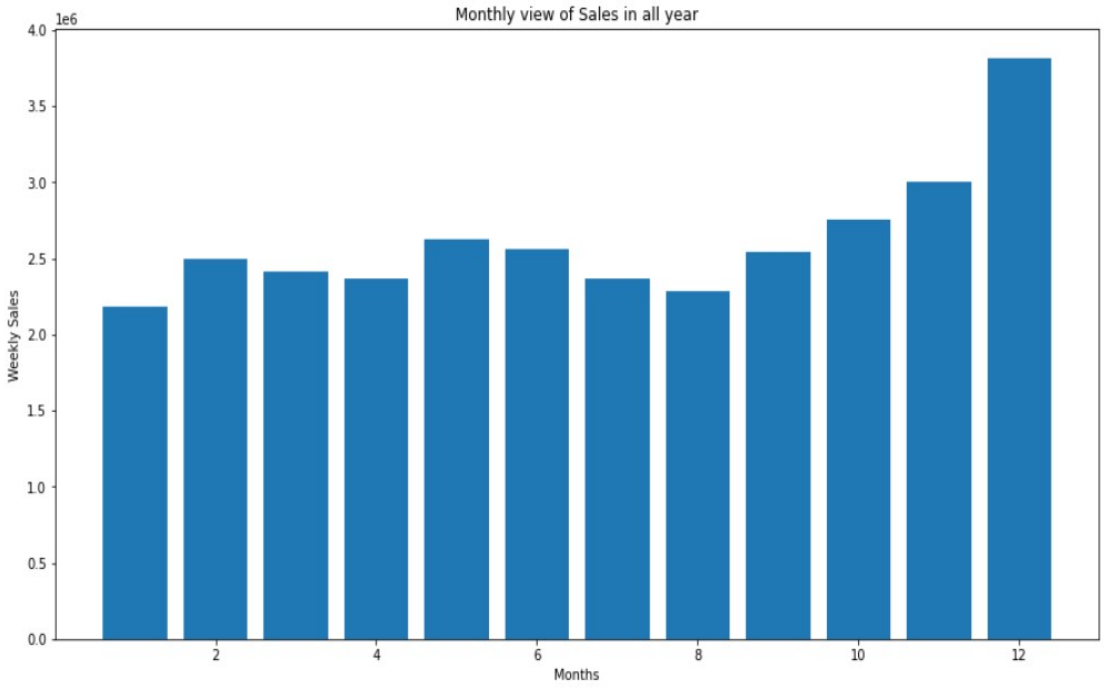

Fig: Monthly view of sales in all year

```
plt.figure(figsize=(12,8))
walmart.groupby('Year')[['Weekly_Sales']].sum().plot(kind='bar',legend=False
)
plt.title('Yearly view of Sales')
plt.xlabel('Years')
plt.ylabel('Weekly Sales')
plt.show()
```

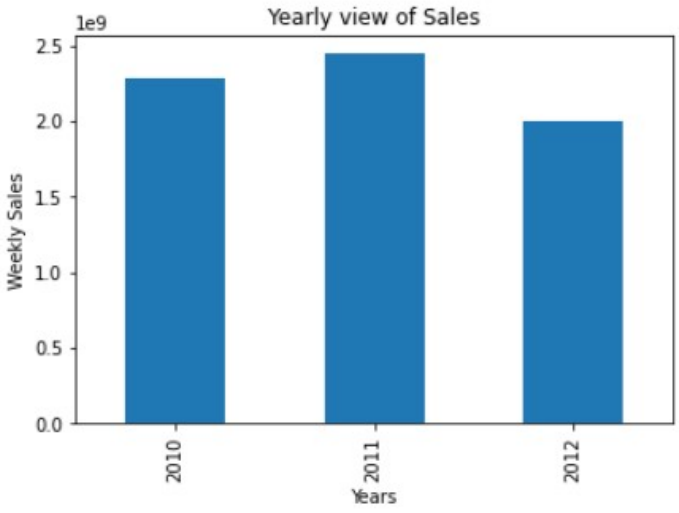

Fig: Yearly view of Sales

Build prediction models to forecast demand (Modeling)

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

Note:	Box plots are useful as they show **outliers** within a data set. An **outlier** is an observation that is numerically distant from the rest of the data.

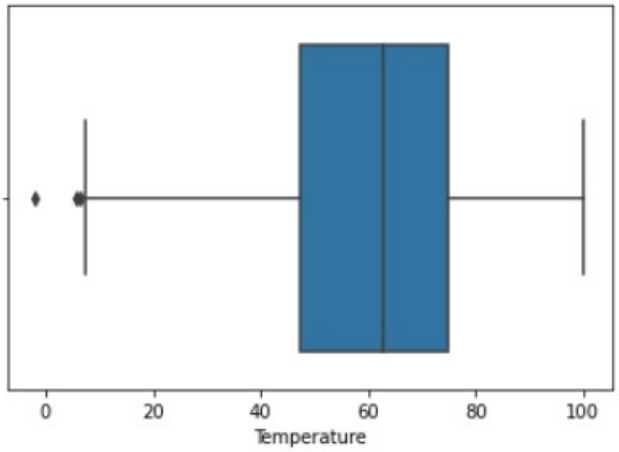

Fig: Boxplot of Temperature (with Outliers)

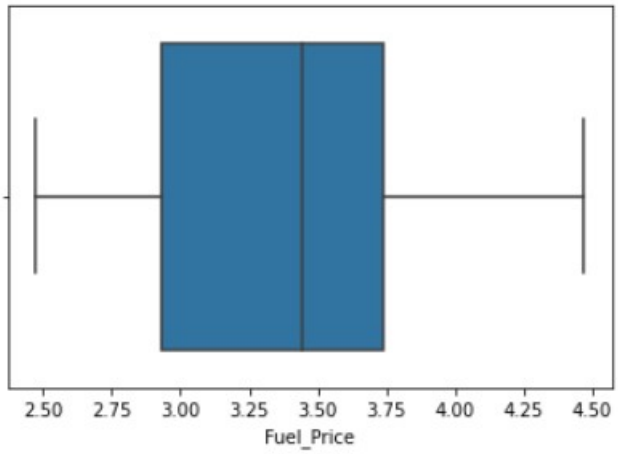

Fig: Boxplot of Fuel price

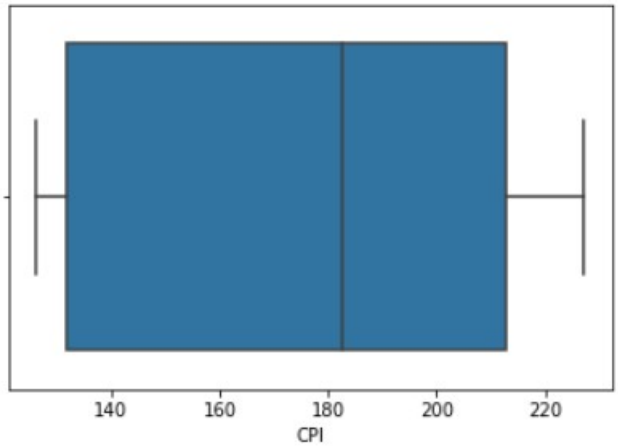

Fig: Boxplot of CPI

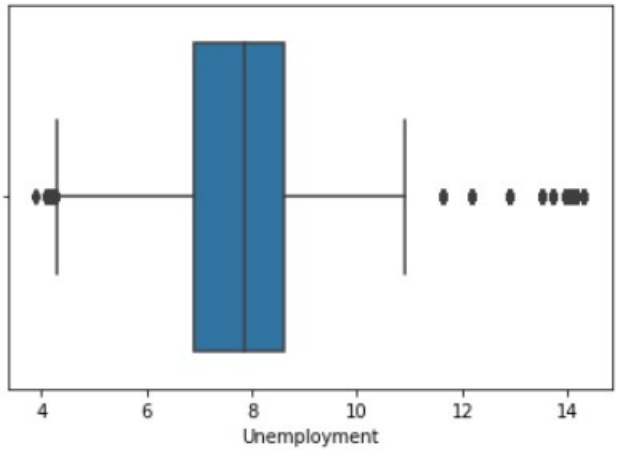

Fig: Boxplot of Unemployement (with outiers)

```
#To avoid misleading interpretations, we are removing outliers

data_new = walmart[(walmart['Temperature'] > 10) &
(walmart['Unemployment']>4.5) & (walmart['Unemployment'] < 10)]
data_new.head()

#againing checking if any outliers are left
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])
```

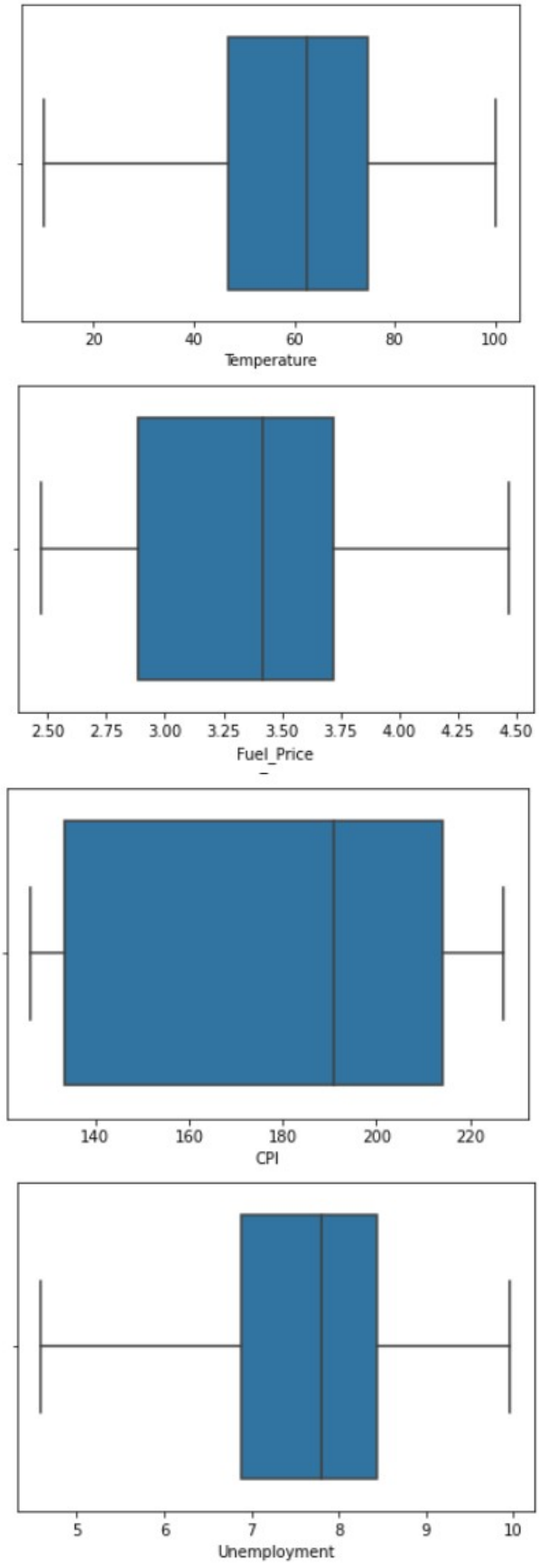

Fig: Boxplot of Temperature, Fuel Price, CPI and Unemployement after removing Outliers

Build Model

Building Linear regression model:

```
#importing libraries.
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor

X = data_new[['Store','Fuel_Price','CPI','Unemployment','Day','Month','Year']]
y = data_new['Weekly_Sales']

#spliting train and test data in the ratio of 80-20
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)

#Linear Regression Model

reg = LinearRegression()
reg.fit(X_train,y_train)
y_pred = reg.predict(X_test)
print('Accuracy:',reg.score(X_train, y_train)*100)

sns.scatterplot(y_pred,y_test)
```

```
Accuracy: 13.480535543049399

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

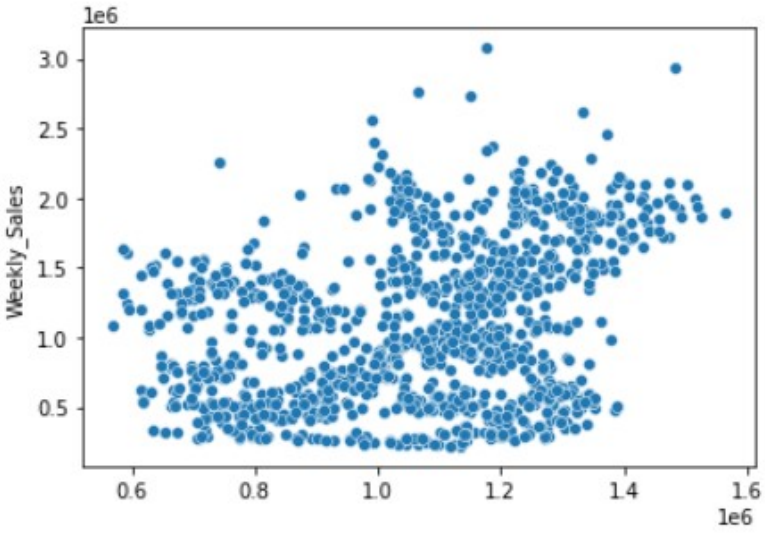

Fig: output of Linear regression model

Building Random Forest Regressor Model:

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

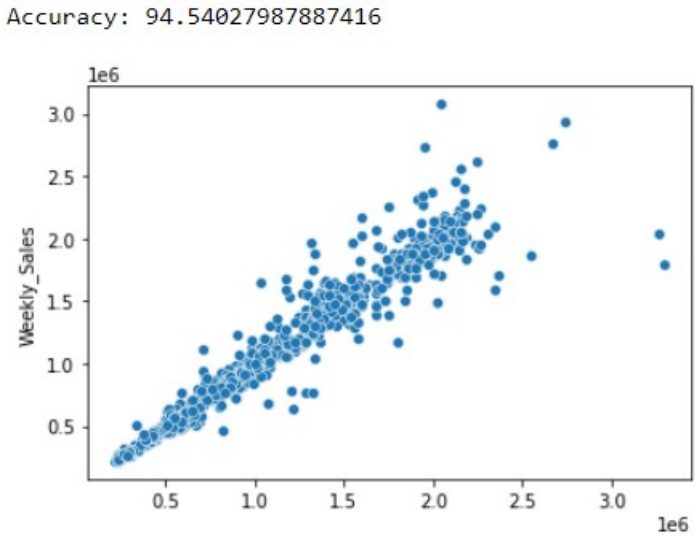

Fig: output of Random Forest Regression Model

****

End of Assessment