Capstone Project – Walmart sales

Table of Contents

Problem Statement

A retail store that has multiple outlets across the country are facing issues in managing the inventory - to match the demand with respect to supply. You are a data scientist, who must come up with useful insights using the data and make prediction models to forecast the sales for X number of months/years

Project Objective

The objective of this project is to provide various insights about the sales data collected by the company and to build a machine learning model to forecast the sales data for future.

Data Description

The dataset available is walmart.csv contains 6435 rows and 8 columns.

Feature Name	Description
Store	Store number
Date	Week of Sales
Weekly_Sales	Sales for the given store in that week
Holiday_Flag	If it is a holiday week
Temperature	Temperature on the day of the sale
Fuel_Price	Cost of the fuel in the region
CPI	Consumer Price Index
Unemployment	Unemployment Rate

1.Data information: In the below table we can find the data type of each column and total non-null values.

```
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
```

2.Data Description: The data description showing the total count, mean, standard deviation(std), minimum, etc. for the numeric features is as shown in the Table.

	count	mean	std	min	25%	50%	75%	max
Store	6435.0	2.300000e+01	12.988182	1.000	12.000	23.000000	3.400000e+01	4.500000e+01
Weekly_Sales	6435.0	1.046965e+06	564366.622054	209986.250	553350.105	960746.040000	1.420159e+06	3.818686e+06
Holiday_Flag	6435.0	6.993007e-02	0.255049	0.000	0.000	0.000000	0.000000e+00	1.000000e+00
Temperature	6435.0	6.066378e+01	18.444933	-2.060	47.460	62.670000	7.494000e+01	1.001400e+02
Fuel_Price	6435.0	3.358607e+00	0.459020	2.472	2.933	3.445000	3.735000e+00	4.468000e+00
CPI	6435.0	1.715784e+02	39.356712	126.064	131.735	182.616521	2.127433e+02	2.272328e+02
Unemployment	6435.0	7.999151e+00	1.875885	3.879	6.891	7.874000	8.622000e+00	1.431300e+01

3. check for Null values:

```
1 wdf.isna().sum()
```

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

3. check for Duplicate values:

```
1 wdf.duplicated().sum()
```

```
0
```

4. Correlation of the features:

	Store	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
Store	1.000000e+00	-0.335332	-4.386841e-16	-0.022659	0.060023	-0.209492	0.223531
Weekly_Sales	-3.353320e-01	1.000000	3.689097e-02	-0.063810	0.009464	-0.072634	-0.106176
Holiday_Flag	-4.386841e-16	0.036891	1.000000e+00	-0.155091	-0.078347	-0.002162	0.010960
Temperature	-2.265908e-02	-0.063810	-1.550913e-01	1.000000	0.144982	0.176888	0.101158
Fuel_Price	6.002295e-02	0.009464	-7.834652e-02	0.144982	1.000000	-0.170642	-0.034684
CPI	-2.094919e-01	-0.072634	-2.162091e-03	0.176888	-0.170642	1.000000	-0.302020
Unemployment	2.235313e-01	-0.106176	1.096028e-02	0.101158	-0.034684	-0.302020	1.000000

Data Preprocessing Steps And Inspiration

The preprocessing of the data included the following steps:

1. Load Walmart sales data
2. Check for missing values and take necessary action as per the following:

   a. Remove the rows with missing value if their number is insignificant.

   b. Replace the missing values with the mean or median if the feature is numeric.

```
wdf.isna().sum()
```

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

3. Check for duplicate values and if present remove duplicate data.

```
wdf.duplicated().sum()
```

```
0
```

4. Convert the feature Date to YYYY-MM-DD time record and set it as index.
5. Before performing tsa convert weekly sales feature to stationary.
6. Perform Exploratory Data Analysis.

Choosing the Algorithm for the Project

The given data set consists of Walmart sales dataset which have large amount of sales data. Random Forest Regressor can capture the complex relationships between variables by building many decision trees and combining their predictions. This can be beneficial for sales data, as sales can be influenced by many different factors that may not follow a specific distribution.

Motivation and Reasons for Choosing the Algorithm:

The reasons for choosing the Random Forest Regressor algorithm include:

Non-Parametric Model: Random Forest Regressor is a non-parametric model, which means that it does not make assumptions about the distribution of the data. This makes it a good choice for sales data that may not follow a specific distribution.

Handling Complex Relationships: Random Forest Regressor can capture complex relationships between variables by building many decision trees and combining their predictions. This is important for Walmart sales data, as sales can be influenced by many different factors such as store location, seasonality, promotions, and economic indicators.

Large Amounts of Data: Walmart is a large company with many stores, so there is likely to be a large amount of sales data. Random Forest Regressor can handle large amounts of data, as it can be parallelized and run on multiple processors.

Model Interpretability: Random Forest Regressor is easy to interpret, as it provides information on the importance of each variable in predicting the target variable. This can be valuable for understanding which factors have the greatest impact on sales.

Assumptions:

The following assumptions were made in order to create the random Forest regressor model for Walmart sale project.

1. Random forest regressor assumes that each decision tree in the forest independently contributes to the overall prediction.
2. The model assumes that the individual trees are uncorrelated and the predictions from each tree are combined to make the final prediction.
3. The model also assumes that the distribution of the target variable is roughly the same for each tree, and that the noise in the target variable is random and independent.

Model Evaluation and Technique

Model 1: ARIMA

1. Check the weekly sale feature is stationary or not using Ad fuller test. If it is not stationary convert weekly sale feature to stationary.

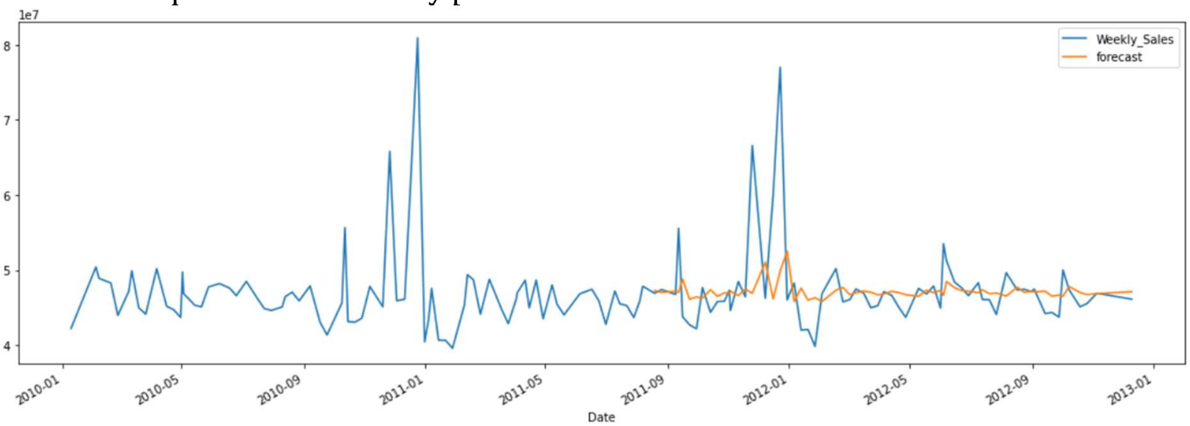
2. After converting feature in to stationary split the data in to train and test datasets.

3. Import auto arima and run the auto arima to find the optimal p,d,q values.

4. Now we created a ARIMA model using the obtained p=0,d=0,q=1 values.

5. Train the Arima model and predict the values up to test size.

6. ARIMA model performance is very poor.




```
np.sqrt(mean_absolute_error(monthly_sales['Weekly_Sales'][80:142],monthly_sales['forecast'][80:142]))
```

```
1719.8566856222462
```

```
r2_score(monthly_sales['Weekly_Sales'][80:142],monthly_sales['forecast'][80:142])
```

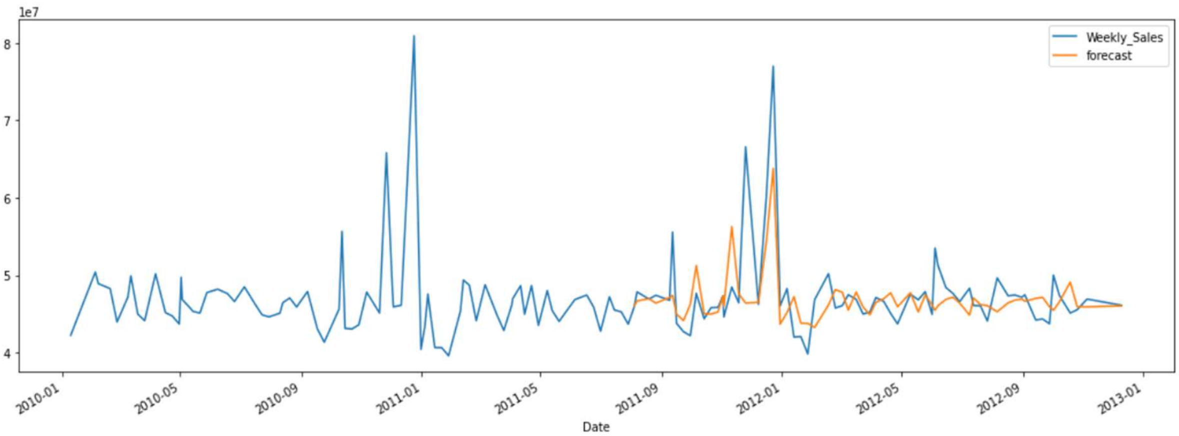
```
0.03314167007046198
```

Model 2; SARIMAX

7. We created a SARIMAX model using the obtained p=0, d=0,q=1,sesonal order=(1,0,1,52).

8. Train the SARIMAX model and predict the values up to test size.

9. SARIMAX model performance is as shown below.




```
In [53]: np.sqrt(mean_absolute_error(monthly_sales1['Weekly_Sales'][78:142],monthly_sales1['forecast'][78:142]))
Out[53]: 1591.3112371285192

In [54]: r2_score(monthly_sales1['Weekly_Sales'][78:142],monthly_sales1['forecast'][78:142])
Out[54]: 0.4086715411189691
```

Model 3: Decision Tree regressor

10. Create a Decision Tree regressor model.
11. Split the dataset in to train and test dataset as 80:20 with random state = 42.
12. Train the model and test.
13. Decision tree regressor r2_score as shown below.

```
RMSE score of DecisionTree for test data:  191441.8215389063
R^2 score of Decision Tree for test data:  0.8862348231017528
```

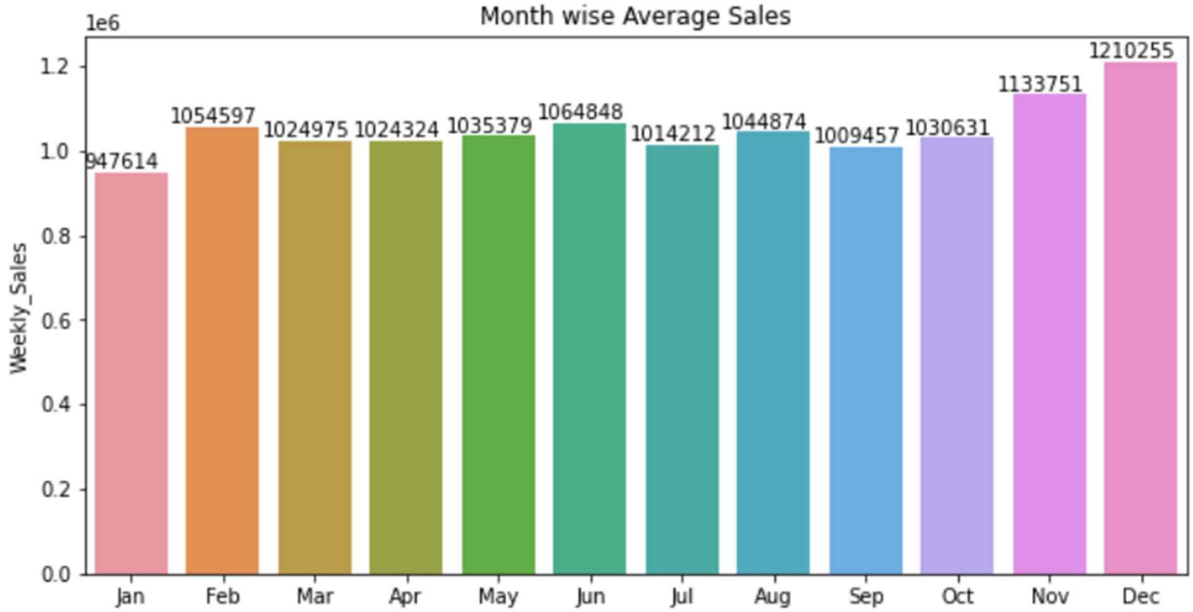
Model 4: Random Forest regressor

14. Created a Random Forest regressor model.
15. Split the dataset in to train and test dataset as 80:20 with random state = 42.
16. Trained the model and tested.
17. Random Forest tree regressor r2_score as below.

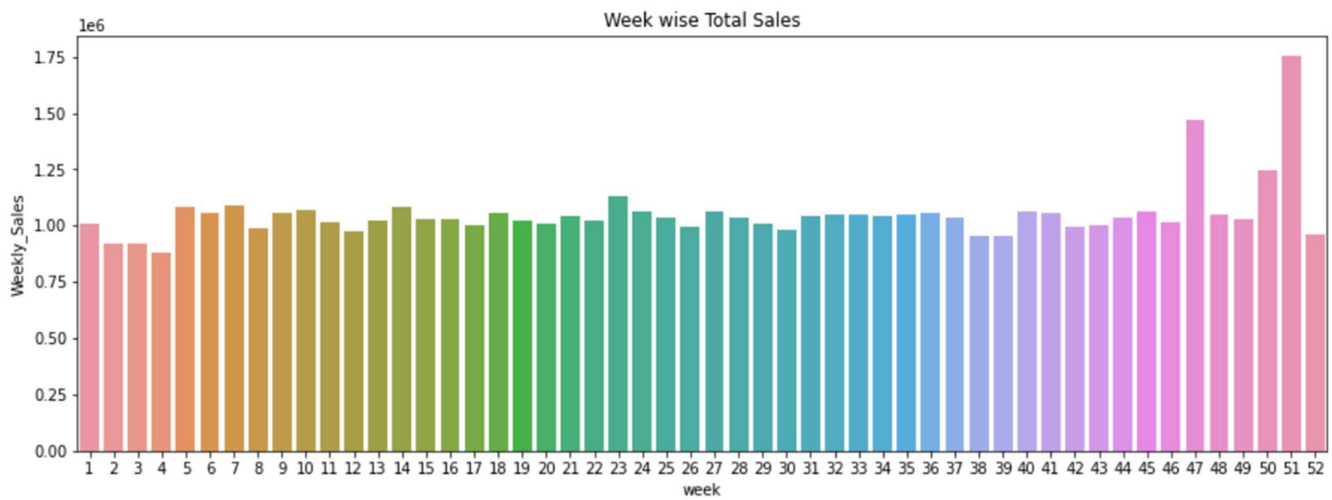
```
RMSE score of Random Forest for test data:  144488.29007378707
R^2 score of Random Forest for test data:  0.9351961193156386
```

Inferences from the Project

Monthly Average sales:



Weekly Average sales:



Top and worst performing store:

```
Top-performing store:  20
Average Sales:  2107676.8703496503
```

```
Worst-performing store:  33
Average Sales:  259861.69202797202
```

The random forest model gives r2-score as 93.5%

```
RMSE score of Random Forest for test data:  144488.29007378707
R^2 score of Random Forest for test data:  0.9351961193156386
```

Conclusion

Random Forest Regressor can be a powerful and flexible model for predicting Walmart sales data. The model is non-parametric and can handle complex relationships between the variables, making it well-suited for sales data that is influenced by a variety of factors. Additionally, Random Forest Regressor can handle large amounts of data, which is important for a company like Walmart that has many stores.

Walmart project

February 11, 2023

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler,MinMaxScaler,LabelEncoder
from sklearn.metrics import *
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
import statsmodels as sm
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import r2_score,␣
↪mean_absolute_error,accuracy_score,classification_report
import warnings
warnings.filterwarnings(action='ignore')
```

[2]:
```
wdf = pd.read_csv(r"Walmart DataSet.csv",index_col='Date',parse_dates=True)
```

[3]:
```
wdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 6435 entries, 2010-05-02 to 2012-10-26
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Store         6435 non-null   int64
 1   Weekly_Sales  6435 non-null   float64
 2   Holiday_Flag  6435 non-null   int64
 3   Temperature   6435 non-null   float64
 4   Fuel_Price    6435 non-null   float64
 5   CPI           6435 non-null   float64
 6   Unemployment  6435 non-null   float64
dtypes: float64(5), int64(2)
memory usage: 402.2 KB
```

[4]:
```
wdf.describe().T
```

```
[4]:                 count          mean            std          min          25%  \
Store          6435.0  2.300000e+01      12.988182        1.000       12.000
Weekly_Sales   6435.0  1.046965e+06  564366.622054   209986.250   553350.105
Holiday_Flag   6435.0  6.993007e-02       0.255049        0.000        0.000
Temperature    6435.0  6.066378e+01      18.444933       -2.060       47.460
Fuel_Price     6435.0  3.358607e+00       0.459020        2.472        2.933
CPI            6435.0  1.715784e+02      39.356712      126.064      131.735
Unemployment   6435.0  7.999151e+00       1.875885        3.879        6.891

                         50%           75%           max
Store              23.000000  3.400000e+01  4.500000e+01
Weekly_Sales   960746.040000  1.420159e+06  3.818686e+06
Holiday_Flag        0.000000  0.000000e+00  1.000000e+00
Temperature        62.670000  7.494000e+01  1.001400e+02
Fuel_Price          3.445000  3.735000e+00  4.468000e+00
CPI               182.616521  2.127433e+02  2.272328e+02
Unemployment        7.874000  8.622000e+00  1.431300e+01
```

```
[5]: wdf.shape
```

```
[5]: (6435, 7)
```

```
[6]: cols = wdf.columns
     cols
```

```
[6]: Index(['Store', 'Weekly_Sales', 'Holiday_Flag', 'Temperature', 'Fuel_Price',
            'CPI', 'Unemployment'],
           dtype='object')
```

```
[7]: wdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 6435 entries, 2010-05-02 to 2012-10-26
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Store         6435 non-null   int64
 1   Weekly_Sales  6435 non-null   float64
 2   Holiday_Flag  6435 non-null   int64
 3   Temperature   6435 non-null   float64
 4   Fuel_Price    6435 non-null   float64
 5   CPI           6435 non-null   float64
 6   Unemployment  6435 non-null   float64
dtypes: float64(5), int64(2)
memory usage: 402.2 KB
```

```
[8]: wdf.isna().sum()
```

[8]:
```
Store           0
Weekly_Sales    0
Holiday_Flag    0
Temperature     0
Fuel_Price      0
CPI             0
Unemployment    0
dtype: int64
```

[9]:
```
wdf.duplicated().sum()
```

[9]: 0

[10]:
```
wdf.describe().T
```

[10]:

	count	mean	std	min	25%
Store	6435.0	2.300000e+01	12.988182	1.000	12.000
Weekly_Sales	6435.0	1.046965e+06	564366.622054	209986.250	553350.105
Holiday_Flag	6435.0	6.993007e-02	0.255049	0.000	0.000
Temperature	6435.0	6.066378e+01	18.444933	-2.060	47.460
Fuel_Price	6435.0	3.358607e+00	0.459020	2.472	2.933
CPI	6435.0	1.715784e+02	39.356712	126.064	131.735
Unemployment	6435.0	7.999151e+00	1.875885	3.879	6.891

	50%	75%	max
Store	23.000000	3.400000e+01	4.500000e+01
Weekly_Sales	960746.040000	1.420159e+06	3.818686e+06
Holiday_Flag	0.000000	0.000000e+00	1.000000e+00
Temperature	62.670000	7.494000e+01	1.001400e+02
Fuel_Price	3.445000	3.735000e+00	4.468000e+00
CPI	182.616521	2.127433e+02	2.272328e+02
Unemployment	7.874000	8.622000e+00	1.431300e+01

[11]:
```
wdf.corr()
```

[11]:

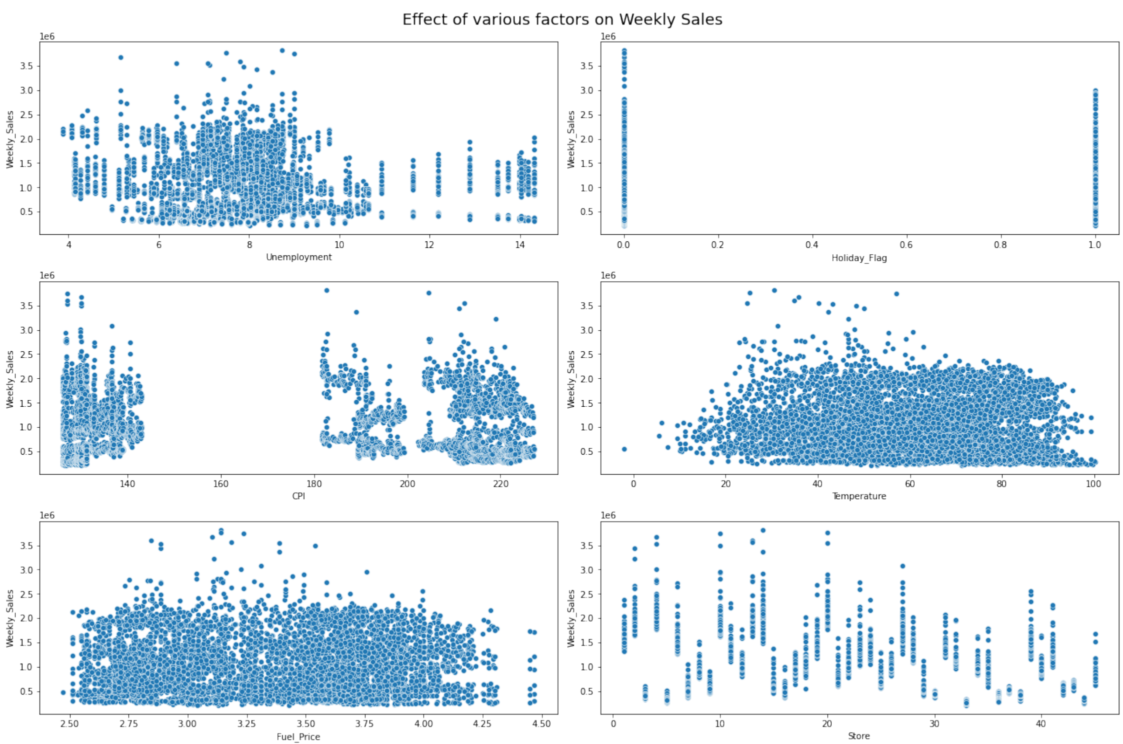
	Store	Weekly_Sales	Holiday_Flag	Temperature
Store	1.000000e+00	-0.335332	-4.386841e-16	-0.022659
Weekly_Sales	-3.353320e-01	1.000000	3.689097e-02	-0.063810
Holiday_Flag	-4.386841e-16	0.036891	1.000000e+00	-0.155091
Temperature	-2.265908e-02	-0.063810	-1.550913e-01	1.000000
Fuel_Price	6.002295e-02	0.009464	-7.834652e-02	0.144982
CPI	-2.094919e-01	-0.072634	-2.162091e-03	0.176888
Unemployment	2.235313e-01	-0.106176	1.096028e-02	0.101158

	Fuel_Price	CPI	Unemployment
Store	0.060023	-0.209492	0.223531
Weekly_Sales	0.009464	-0.072634	-0.106176
Holiday_Flag	-0.078347	-0.002162	0.010960

```
Temperature     0.144982  0.176888      0.101158
Fuel_Price      1.000000 -0.170642     -0.034684
CPI            -0.170642  1.000000     -0.302020
Unemployment   -0.034684 -0.302020      1.000000
```

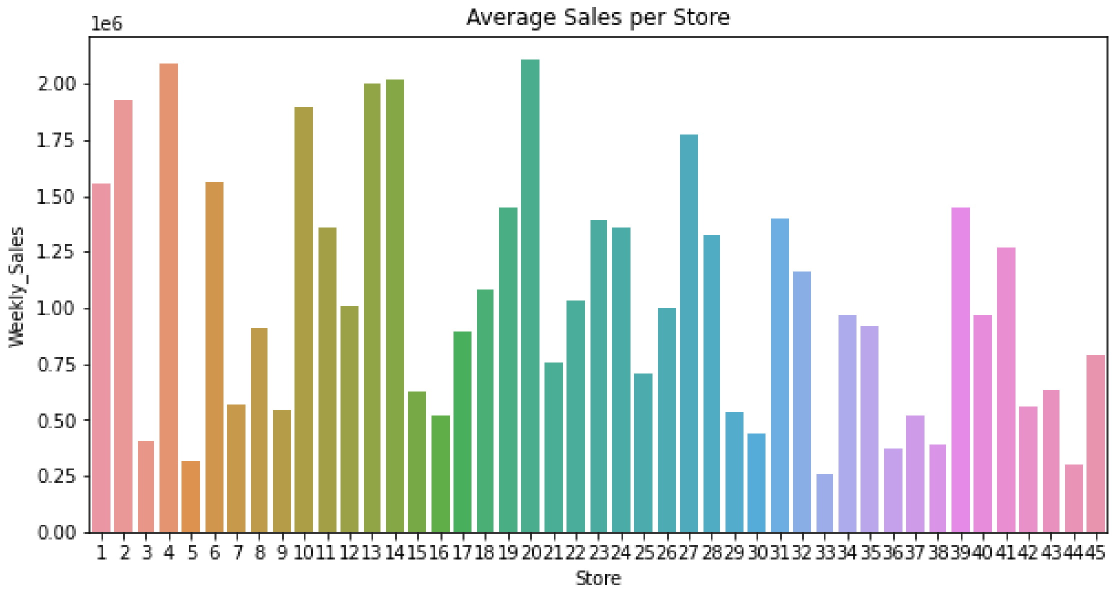
0.1 each feature vs weekly sales

[12]:
```
fig, axes = plt.subplots(3,2,figsize=(18,12))
ax_index = [(i,j) for i in range(3) for j in range(2)]
index_number = 0
fig.suptitle('Effect of various factors on Weekly Sales',fontsize=18, color =␣
 ↪'Black')
for i in␣
 ↪['Unemployment','Holiday_Flag','CPI','Temperature','Fuel_Price','Store']:
    sns.scatterplot(x=i, y='Weekly_Sales', data=wdf,␣
 ↪ax=axes[ax_index[index_number]], palette='afmhot_r')
    index_number += 1
    plt.tight_layout()
```

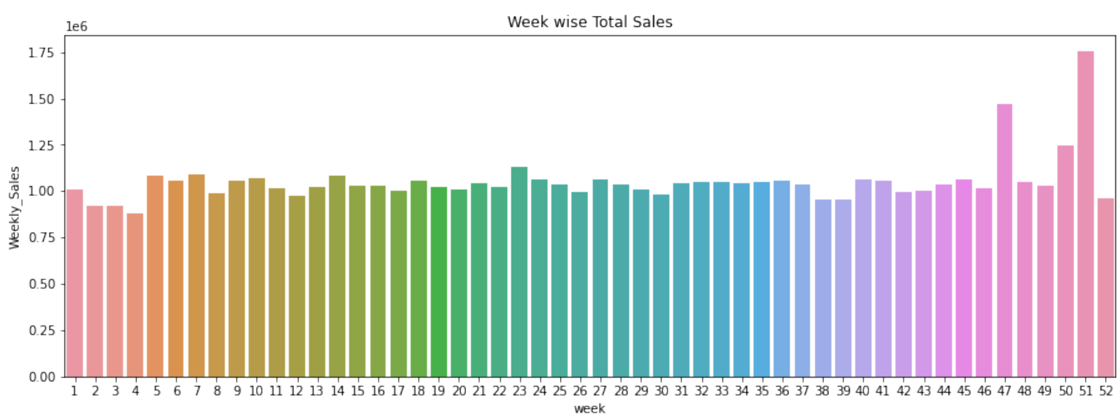


[13]:
```
avg_sales_per_store = wdf.groupby(by='Store')['Weekly_Sales'].mean()
plt.figure(figsize=(10,5))
sns.barplot(x = avg_sales_per_store.index, y=avg_sales_per_store)
plt.title('Average Sales per Store')
```

```
plt.show()
```

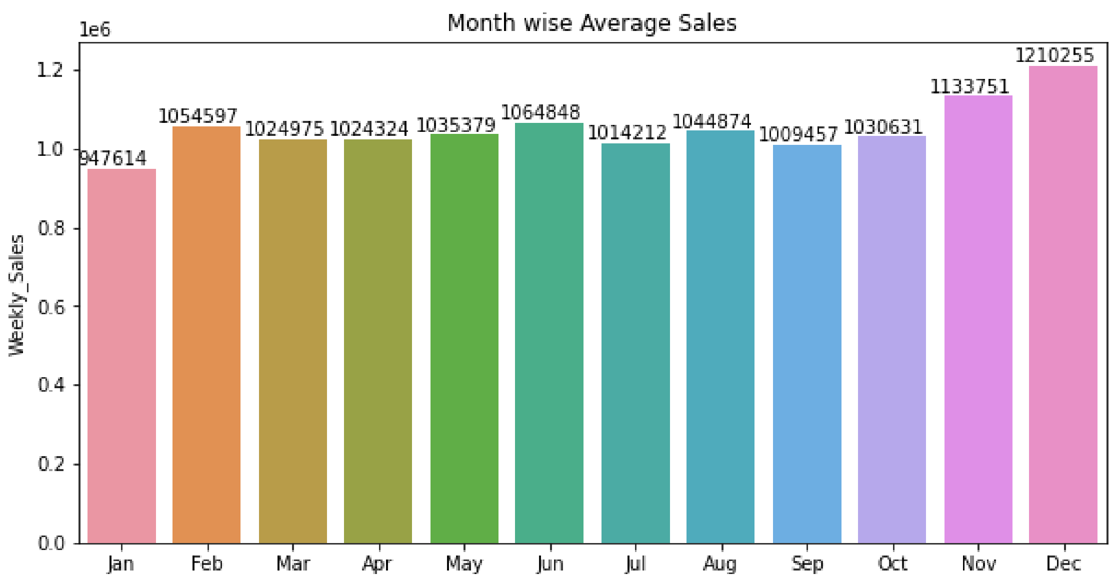


[14]:
```
plt.figure(figsize=(15,5))
ax=sns.barplot(x=wdf.index.isocalendar().week, y="Weekly_Sales",␣
↪data=wdf,ci=None)
plt.title('Week wise Total Sales')
plt.show()
```



[15]:
```
plt.figure(figsize=(10,5))
month_wise_avg_sales=wdf.groupby(wdf.index.month)['Weekly_Sales'].mean()
plt.title('Month wise Average Sales')
```

```
g = sns.
↪barplot(x=['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec'],␣
↪y=month_wise_avg_sales)
for p in g.patches:
    g.annotate('{:.0f}'.format(p.get_height()), (p.get_x()+0.3, p.
↪get_height()),ha='center', va='bottom',color= 'black')
```



[16]:
```
wdf['Store'].unique()
```

[16]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45], dtype=int64)
```

[17]:
```
from sklearn.linear_model import LinearRegression
```

[18]:
```
# extract the unemployment rate and weekly sales columns
X = wdf.loc[:,['Unemployment']]
y = wdf.loc[:,['Weekly_Sales']]
print(X.shape)
print(y.shape)
```

```
(6435, 1)
(6435, 1)
```

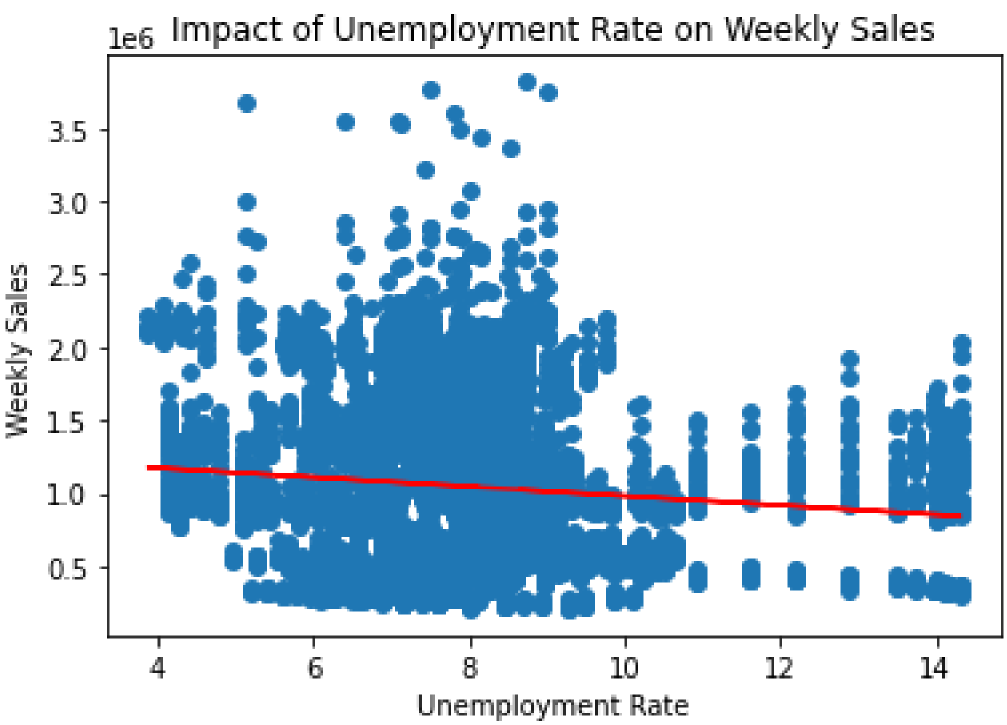
[19]:
```
# create a Linear Regression model
model = LinearRegression()
```

[20]:
```python
# fit the model to the data
model.fit(X, y)

# make predictions using the model
predictions = model.predict(X)
```

[21]:
```python
# calculate the correlation between unemployment rate and weekly sales
correlation = np.corrcoef(wdf['Unemployment'], wdf['Weekly_Sales'])[0][1]

# print the correlation
print("Correlation between unemployment rate and weekly sales:", correlation)
```

```
Correlation between unemployment rate and weekly sales: -0.10617608965795416
```

[59]:
```python
# plot the actual vs predicted values
plt.scatter(wdf['Unemployment'], wdf['Weekly_Sales'])
plt.plot(wdf['Unemployment'], predictions, color='red')
plt.xlabel("Unemployment Rate")
plt.ylabel("Weekly Sales")
plt.title("Impact of Unemployment Rate on Weekly Sales")
plt.show()
```



the regression line slopes downwards, it means that there is a negative relationship between Unemployment Rate and weeklysales

[23]:
```
# calculate the store-wise correlation between unemployment rate and weekly␣
↪sales
store_wise_correlation = {}
for store in wdf['Store'].unique():
    store_data = wdf[wdf['Store'] == store]
    correlation = np.corrcoef(store_data['Unemployment'],␣
↪store_data['Weekly_Sales'])[0][1]
    store_wise_correlation[store] = correlation
store_wise_correlation_df = pd.DataFrame(list(store_wise_correlation.items()),␣
↪columns=['Store', 'Correlation'])
store_wise_correlation_df = store_wise_correlation_df.
↪sort_values(by='Correlation', ascending=False)
store_wise_correlation_df.head()
```

[23]:

	Store	Correlation
35	36	0.833734
34	35	0.483865
20	21	0.218367
13	14	0.210786
29	30	0.201862

[24]:
```
# determine the store that is most affected by unemployment rate
max_correlation = max(store_wise_correlation.values())
for store, corr in store_wise_correlation.items():
    if corr == max_correlation:
        most_affected_store = store
        break

print("Store most affected by unemployment rate:", most_affected_store)
```

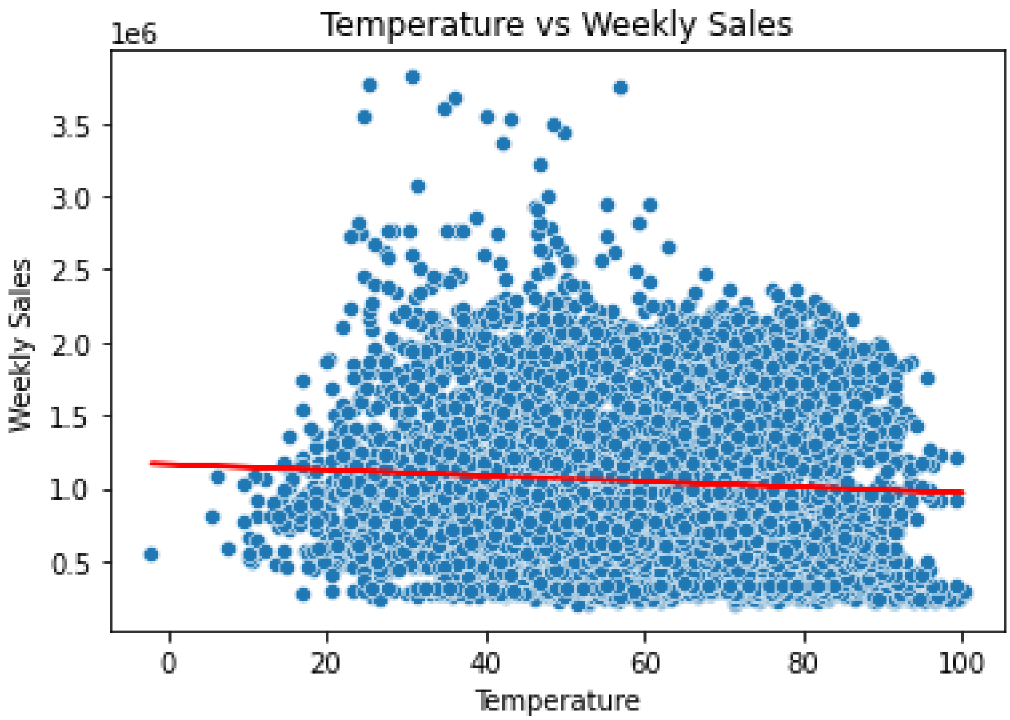
```
Store most affected by unemployment rate: 36
```

[25]:
```
T = wdf.loc[:, ['Temperature']]
W = wdf.loc[:, ['Weekly_Sales']]
```

[26]:
```
reg1 = LinearRegression().fit(T, W)
```

[27]:
```
y_pred = reg1.predict(T)
y_pred =y_pred.flatten()
```

[28]:
```
# plot the scatter plot of the temperature vs weekly sales
sns.scatterplot(x=wdf['Temperature'], y=wdf['Weekly_Sales'])
# plot the regression line
plt.plot(wdf['Temperature'], y_pred, color='red')
plt.xlabel('Temperature')
plt.ylabel('Weekly Sales')
plt.title('Temperature vs Weekly Sales')
plt.show()
```

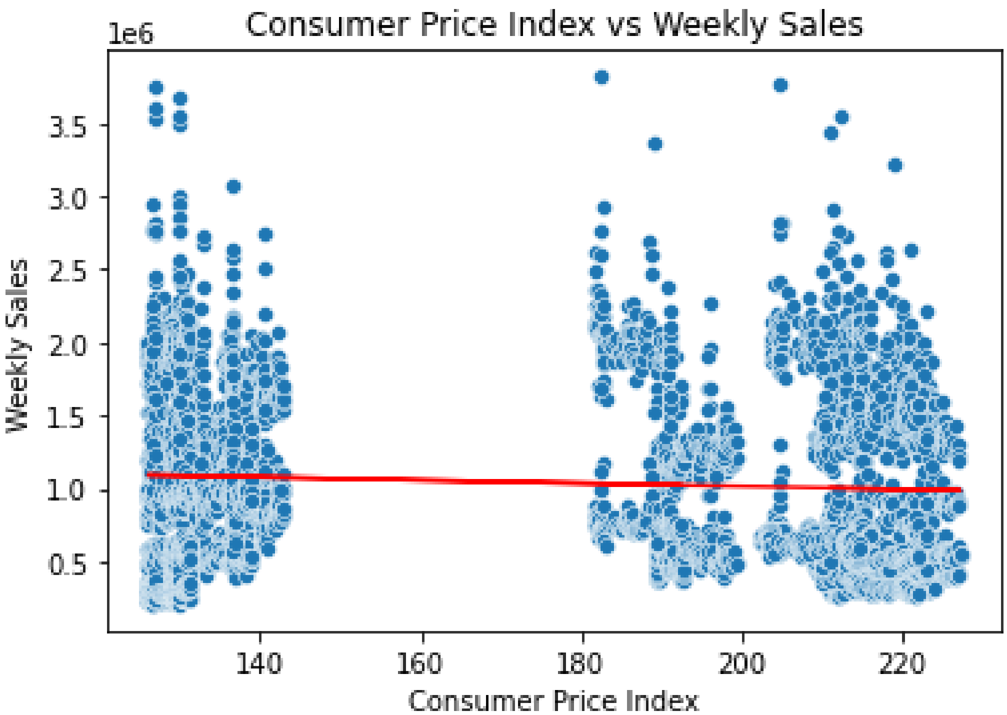


the regression line slopes downwards, it means that there is a slight negative relationship between temperature and weekly sales, i.e., higher temperature results in lower sales.

[29]:
```
C = wdf.loc[:, ['CPI']]
W = wdf.loc[:, ['Weekly_Sales']]
```

[30]:
```
reg = LinearRegression().fit(C, W)
# predict the weekly sales using the Consumer Price Index data
C_pred = reg.predict(C)
C_pred = C_pred.flatten()
```

[31]:
```
sns.scatterplot(wdf['CPI'], wdf['Weekly_Sales'])

# plot the regression line
plt.plot(wdf['CPI'], C_pred, color='red')

plt.xlabel('Consumer Price Index')
plt.ylabel('Weekly Sales')
plt.title('Consumer Price Index vs Weekly Sales')
plt.show()
```



From the scatter plot and the regression line, you can determine that regression line is slightly slope downward which means there is a slightly negative relation ship

[32]:
```
# group the data by store and calculate the average sales for each store
store_wise_avg_sales = wdf.groupby('Store').mean()['Weekly_Sales']
```

[33]:
```
# sort the store_wise_avg_sales dataframe in descending order
store_wise_avg_sales = store_wise_avg_sales.sort_values(ascending=False)
```

[34]:
```
# print the top-performing store
print("Top-performing store: ", store_wise_avg_sales.index[0])
print("Average Sales: ", store_wise_avg_sales.values[0])
```

```
Top-performing store:  20
Average Sales:  2107676.8703496503
```

[35]:
```
# print the worst-performing store
print("Worst-performing store: ", store_wise_avg_sales.index[-1])
print("Average Sales: ", store_wise_avg_sales.values[-1])
```

```
Worst-performing store:  33
Average Sales:  259861.69202797202
```

[36]:
```
# print the difference between the highest and lowest performing stores
difference = store_wise_avg_sales.values[0] - store_wise_avg_sales.values[-1]
```

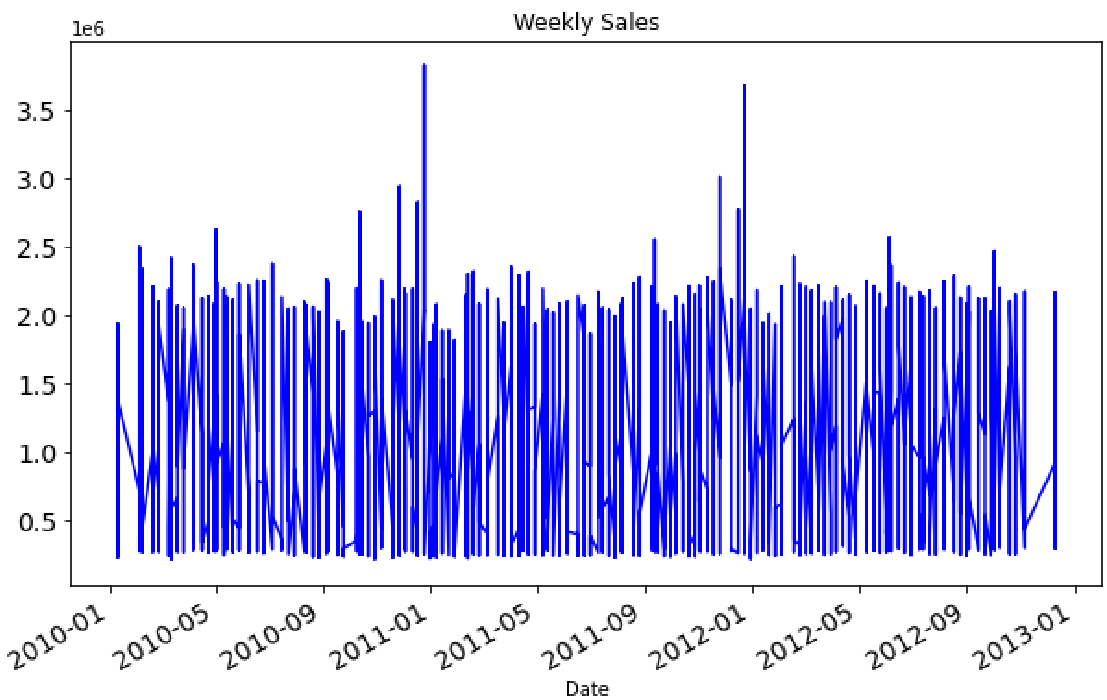
```
print("Difference between highest and lowest performing stores: ", difference)
```

```
Difference between highest and lowest performing stores:  1847815.1783216782
```

[37]:
```
wdf.index
```

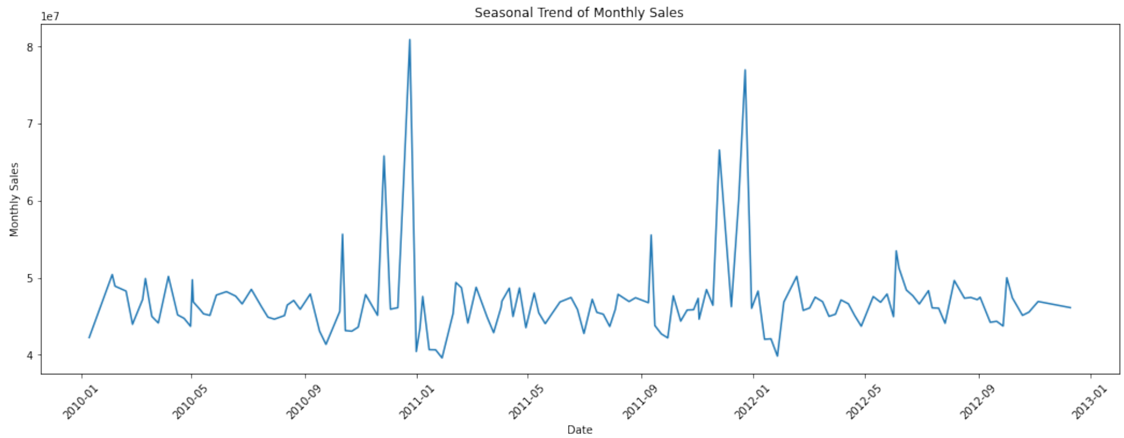
[37]:
```
DatetimeIndex(['2010-05-02', '2010-12-02', '2010-02-19', '2010-02-26',
               '2010-05-03', '2010-12-03', '2010-03-19', '2010-03-26',
               '2010-02-04', '2010-09-04',
               …
               '2012-08-24', '2012-08-31', '2012-07-09', '2012-09-14',
               '2012-09-21', '2012-09-28', '2012-05-10', '2012-12-10',
               '2012-10-19', '2012-10-26'],
              dtype='datetime64[ns]', name='Date', length=6435, freq=None)
```

[38]:
```
wdf['Weekly_Sales'].plot(figsize=(10,6), title= 'Weekly Sales', fontsize=14,␣
 ↪color = 'blue')
plt.show()
```



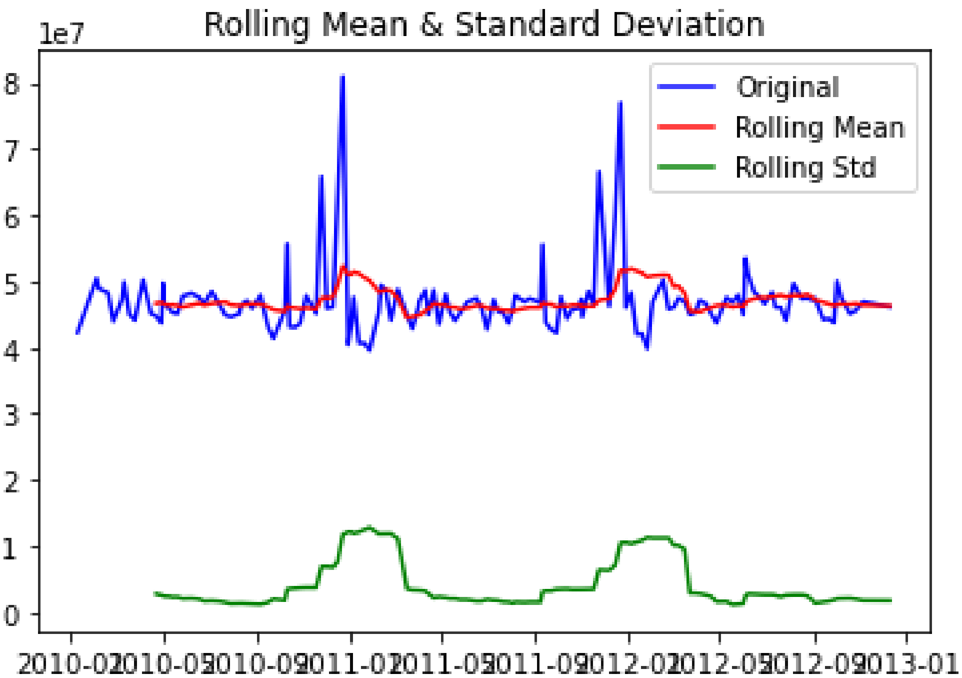
[39]:
```
monthly_sales = wdf.groupby(wdf.index).sum()

plt.figure(figsize=(18,6))
sns.lineplot(monthly_sales.index, monthly_sales['Weekly_Sales'])
plt.xlabel('Date')
plt.xticks(rotation=45)
```

```
plt.ylabel('Monthly Sales')
plt.title('Seasonal Trend of Monthly Sales')
plt.show()
```



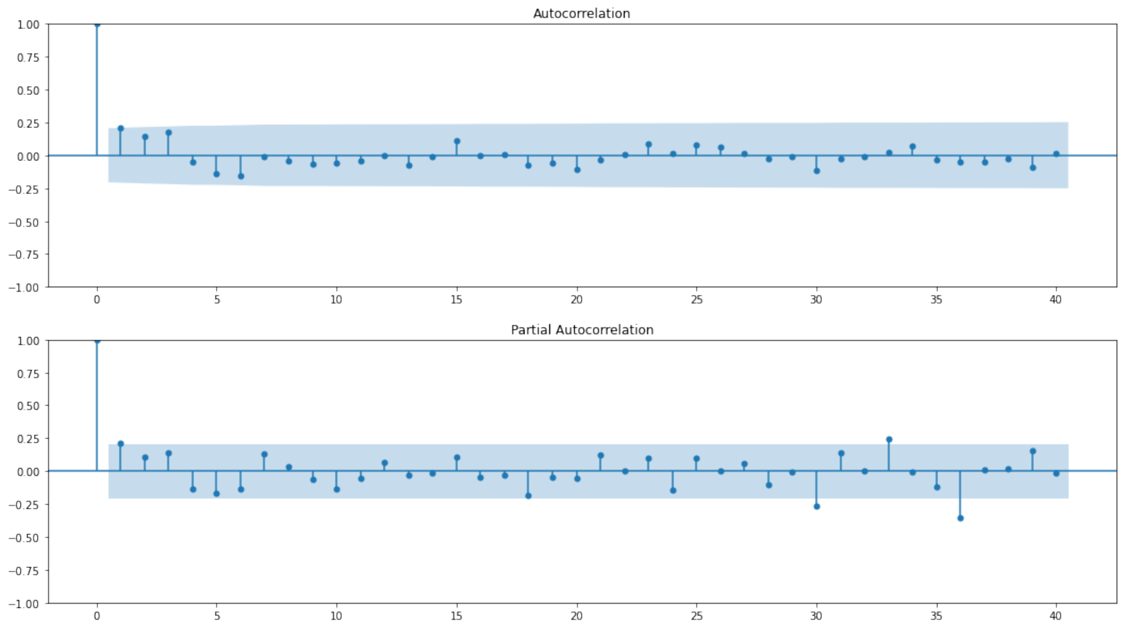
[40]:
```
def check_stationarity(timeseries):
    rolmean = timeseries.rolling(window=12).mean()
    rolstd = timeseries.rolling(window=12).std()
    plt.plot(timeseries, color='blue',label='Original')
    plt.plot(rolmean, color='red', label='Rolling Mean')
    plt.plot(rolstd, color='green', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)
    print("Results of Dickey-Fuller Test:")
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags␣
 ↪Used','Number of Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)
```

[41]:
```
check_stationarity(monthly_sales['Weekly_Sales'])
```



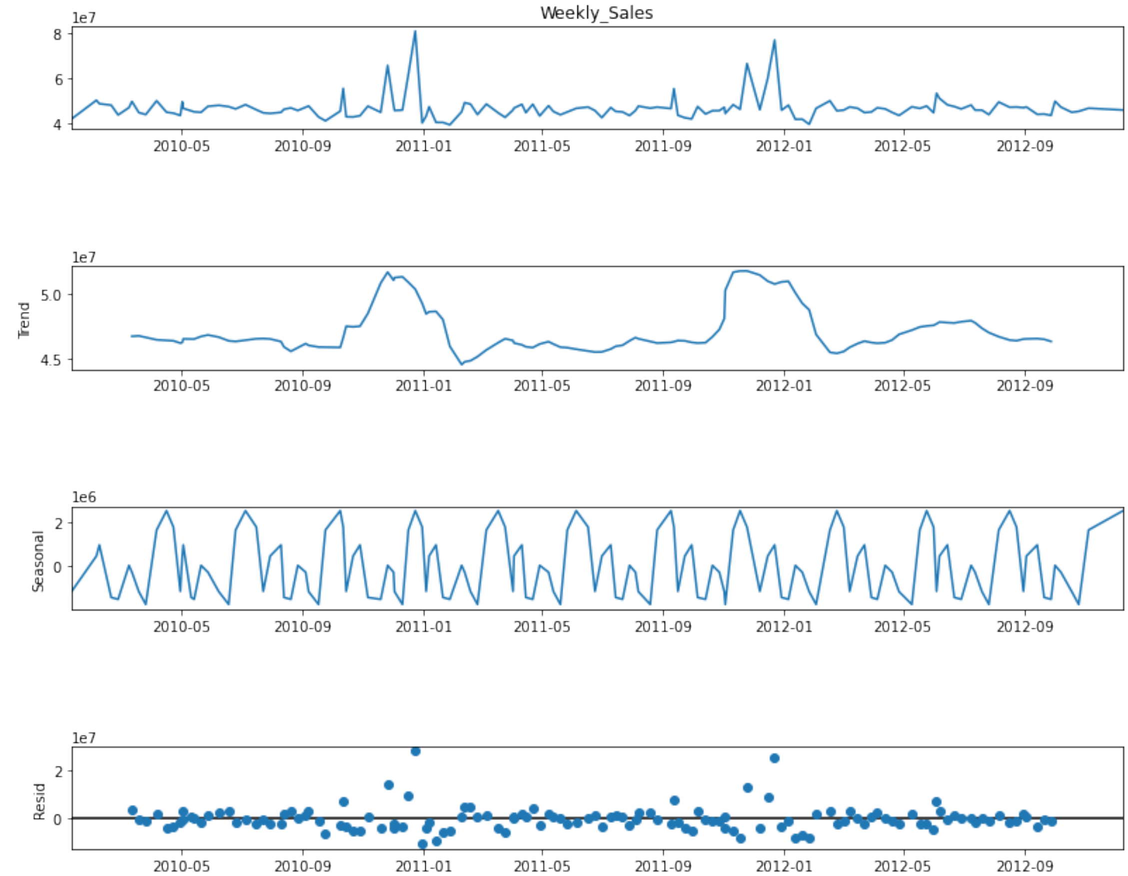

```
Results of Dickey-Fuller Test:
Test Statistic                -9.837722e+00
p-value                        4.845103e-17
#Lags Used                     0.000000e+00
Number of Observations Used    1.420000e+02
Critical Value (1%)           -3.477262e+00
Critical Value (5%)           -2.882118e+00
Critical Value (10%)          -2.577743e+00
dtype: float64
```

[42]:
```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
fig = plt.figure(figsize=(18,10))
ax1 = fig.add_subplot(211)
fig = plot_acf(monthly_sales['Weekly_Sales'].iloc[52:],lags=40,ax=ax1)
ax2 = fig.add_subplot(212)
fig = plot_pacf(monthly_sales['Weekly_Sales'].iloc[52:],lags=40,ax=ax2)
```



[43]:
```
from statsmodels.tsa.seasonal import seasonal_decompose

decomposition = seasonal_decompose(monthly_sales['Weekly_Sales'], period=12)
fig = plt.figure()
fig = decomposition.plot()
fig.set_size_inches(12, 10)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



[44]:
```
from pmdarima import auto_arima
stepwise_fit = auto_arima(monthly_sales['Weekly_Sales'],␣
↪trace=True,seasonal=False,suppress_warnings=True,m=12)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0]             : AIC=inf, Time=0.18 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=5462.768, Time=0.01 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=4917.955, Time=0.01 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=5366.050, Time=0.02 sec
 ARIMA(2,0,0)(0,0,0)[0]             : AIC=inf, Time=0.01 sec
 ARIMA(1,0,1)(0,0,0)[0]             : AIC=inf, Time=0.05 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=inf, Time=0.11 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=4841.703, Time=0.03 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=4844.691, Time=0.01 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=4843.516, Time=0.03 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=4843.733, Time=0.02 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=4841.471, Time=0.01 sec
 ARIMA(0,0,2)(0,0,0)[0] intercept   : AIC=4843.486, Time=0.02 sec
 ARIMA(1,0,2)(0,0,0)[0] intercept   : AIC=4845.479, Time=0.03 sec
```

```
Best model:  ARIMA(0,0,1)(0,0,0)[0] intercept
Total fit time: 0.542 seconds
```

1 ARIMA

[45]:
```
import warnings
import matplotlib.pyplot as plt
warnings.filterwarnings(action='ignore')
from statsmodels.tsa.arima.model import ARIMA

model=ARIMA(monthly_sales['Weekly_Sales'],order=(0, 0, 1))
model_fit=model.fit()
```

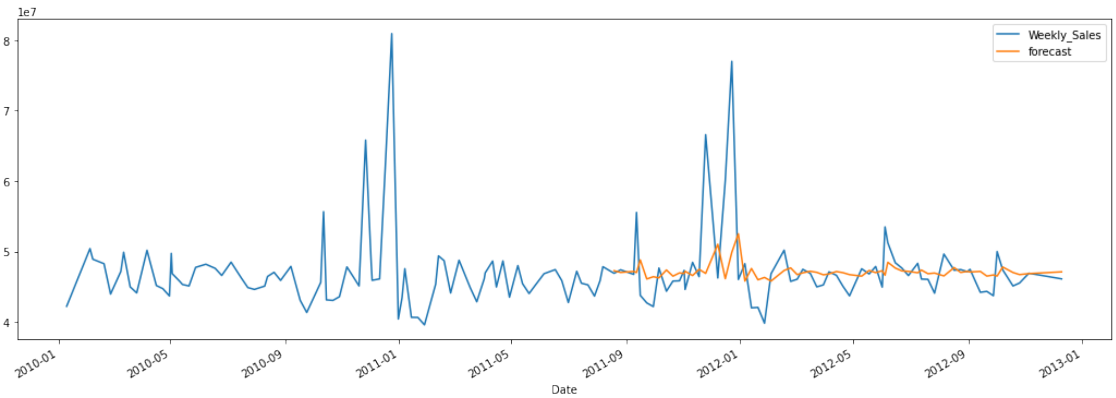
[46]:
```
model_fit.summary()
```

[46]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==============================================================================
Dep. Variable:           Weekly_Sales   No. Observations:                  143
Model:                 ARIMA(0, 0, 1)   Log Likelihood               -2417.698
Date:                Sat, 11 Feb 2023   AIC                           4841.397
Time:                        21:52:22   BIC                           4850.285
Sample:                             0   HQIC                          4845.008
                                - 143
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const       4.711e+07   6.61e+05     71.232      0.000    4.58e+07    4.84e+07
ma.L1          0.1996      0.046      4.372      0.000       0.110       0.289
sigma2      2.886e+13      0.186   1.55e+14      0.000    2.89e+13    2.89e+13
================================================================================
===
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
1282.42
Prob(Q):                              0.95   Prob(JB):
0.00
Heteroskedasticity (H):               0.81   Skew:
3.09
Prob(H) (two-sided):                  0.46   Kurtosis:
16.31
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
```

```
step).
[2] Covariance matrix is singular or near-singular, with condition number
6.3e+28. Standard errors may be unstable.
"""
```

[47]:
```python
monthly_sales['forecast']=model_fit.predict(start=80,end=142)
monthly_sales[['Weekly_Sales','forecast']].plot(figsize=(18,6))
```

[47]: `<AxesSubplot:xlabel='Date'>`



[48]:
```python
np.sqrt(mean_absolute_error(monthly_sales['Weekly_Sales'][80:
↪142],monthly_sales['forecast'][80:142]))
```

[48]: 1719.8566856222462

[49]:
```python
r2_score(monthly_sales['Weekly_Sales'][80:142],monthly_sales['forecast'][80:
↪142])
```

[49]: 0.03314167007046198

2 SARIMAX

[50]:
```python
model1=SARIMAX(monthly_sales['Weekly_Sales'],order=(1,0,1),seasonal_order=(1,0,1,52))
results1=model1.fit()
```

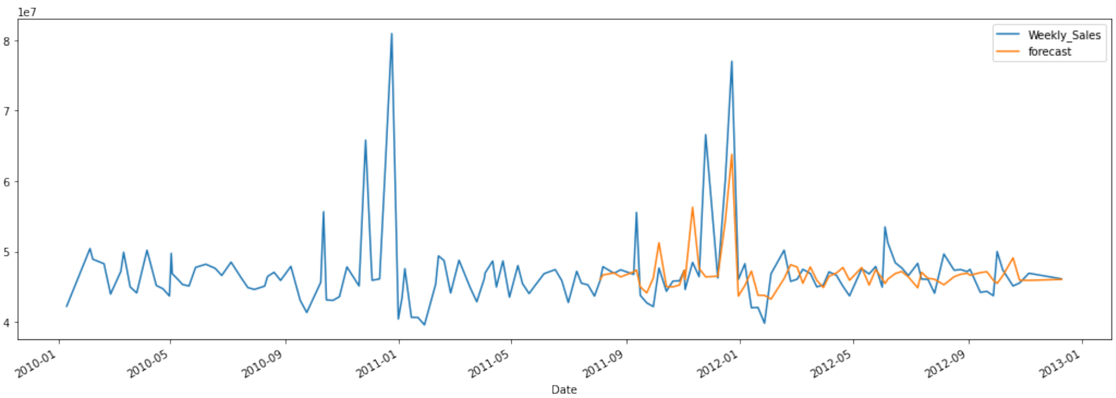
[51]:
```python
results1.summary()
```

[51]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                     SARIMAX Results
================================================================================
==========
Dep. Variable:                       Weekly_Sales   No. Observations:
```

```
143
Model:             SARIMAX(1, 0, 1)x(1, 0, 1, 52)   Log Likelihood
-2416.653
Date:                            Sat, 11 Feb 2023   AIC
4843.307
Time:                                    21:52:24   BIC
4858.121
Sample:                                         0   HQIC
4849.326
                                            - 143
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.0000   4.24e-05   2.36e+04      0.000       1.000       1.000
ma.L1         -0.9924      0.076    -13.026      0.000      -1.142      -0.843
ar.S.L52       0.5100      1.316      0.388      0.698      -2.068       3.088
ma.S.L52      -0.0177      1.749     -0.010      0.992      -3.445       3.410
sigma2      3.743e+13   5.64e-14   6.64e+26      0.000    3.74e+13    3.74e+13
================================================================================
===
Ljung-Box (L1) (Q):                   3.64   Jarque-Bera (JB):
1319.91
Prob(Q):                              0.06   Prob(JB):
0.00
Heteroskedasticity (H):               0.60   Skew:
2.96
Prob(H) (two-sided):                  0.08   Kurtosis:
16.66
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
[2] Covariance matrix is singular or near-singular, with condition number
9.73e+42. Standard errors may be unstable.
"""
```

[52]:
```python
monthly_sales['forecast']=results1.predict(start=78,end=142,dynamic=True)
monthly_sales[['Weekly_Sales','forecast']].plot(figsize=(18,6))
```

[52]: <AxesSubplot:xlabel='Date'>



[53]:
```
np.sqrt(mean_absolute_error(monthly_sales['Weekly_Sales'][78:
↪142],monthly_sales['forecast'][78:142]))
```

[53]: 1591.3112371285192

[54]:
```
r2_score(monthly_sales['Weekly_Sales'][78:142],monthly_sales['forecast'][78:
↪142])
```

[54]: 0.4086715411189691

[55]:
```
wd = wdf
inp= wd.drop('Weekly_Sales',1)
out = wd['Weekly_Sales']
```

3 DecisionTreeRegressor

[56]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from statsmodels.tools.eval_measures import rmse

x_train,x_test,y_train,y_test = train_test_split(inp,out,test_size=0.
↪2,random_state=42)
```

[57]:
```
dtree=  DecisionTreeRegressor()
dtree.fit(x_train,y_train)

_
ytrain_pred = dtree.predict(x_train)
ytest_pred = dtree.predict(x_test)

print('RMSE score of DecisionTree for train data: ', rmse(y_train, ytrain_pred)␣
↪)
```

```
print('R^2 score of Decision Tree for train data: ',r2_score(y_train,␣
 ↪ytrain_pred) )

print('RMSE score of DecisionTree for test data: ', rmse(y_test, ytest_pred) )
print('R^2 score of Decision Tree for test data: ', r2_score(y_test,␣
 ↪ytest_pred) )
```

```
RMSE score of DecisionTree for train data:  0.0
R^2 score of Decision Tree for train data:  1.0
RMSE score of DecisionTree for test data:  195889.7158754299
R^2 score of Decision Tree for test data:  0.8808870492666036
```

4 RandomForestRegressor

[58]:
```
from sklearn.ensemble import RandomForestRegressor
rf1 = RandomForestRegressor()
rf1.fit(x_train, y_train)


ytrain_pred =  rf1.predict(x_train)
ytest_pred = rf1.predict(x_test)

print('RMSE score of train data: ', rmse(y_train, ytrain_pred) )
print('R^2 score of train data: ',r2_score(y_train, ytrain_pred) )

print('RMSE score of Random Forest for test data: ', rmse(y_test, ytest_pred) )
print('R^2 score of Random Forest for test data: ', r2_score(y_test,␣
 ↪ytest_pred) )
```

```
RMSE score of train data:  55549.939438202506
R^2 score of train data:  0.9902814852690749
RMSE score of Random Forest for test data:  148754.63215634137
R^2 score of Random Forest for test data:  0.9313126585273552
```