Untitled1

April 25, 2024

[1]:

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import dates
from datetime import datetime
```

[33]:

```
data = pd.read_csv('Walmart_Store_sales.csv')
data
```

[33]:

```
      Store        Date  Weekly_Sales  Holiday_Flag  Temperature  Fuel_Price  \
0         1  05-02-2010    1643690.90             0        42.31       2.572
1         1  12-02-2010    1641957.44             1        38.51       2.548
2         1  19-02-2010    1611968.17             0        39.93       2.514
3         1  26-02-2010    1409727.59             0        46.63       2.561
4         1  05-03-2010    1554806.68             0        46.50       2.625
…         …           …             …             …            …           …
6430     45  28-09-2012     713173.95             0        64.88       3.997
6431     45  05-10-2012     733455.07             0        64.89       3.985
6432     45  12-10-2012     734464.36             0        54.47       4.000
6433     45  19-10-2012     718125.53             0        56.47       3.969
6434     45  26-10-2012     760281.43             0        58.85       3.882

             CPI  Unemployment
0     211.096358         8.106
1     211.242170         8.106
2     211.289143         8.106
3     211.319643         8.106
4     211.350143         8.106
…            …            …
6430  192.013558         8.684
6431  192.170412         8.667
6432  192.327265         8.667
6433  192.330854         8.667
6434  192.308899         8.667

[6435 rows x 8 columns]
```

[34]:
```
# Convert date to datetime format and show dataset information
data['Date'] =  pd.to_datetime(data['Date'])
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6435 entries, 0 to 6434
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Store         6435 non-null   int64
 1   Date          6435 non-null   datetime64[ns]
 2   Weekly_Sales  6435 non-null   float64
 3   Holiday_Flag  6435 non-null   int64
 4   Temperature   6435 non-null   float64
 5   Fuel_Price    6435 non-null   float64
 6   CPI           6435 non-null   float64
 7   Unemployment  6435 non-null   float64
dtypes: datetime64[ns](1), float64(5), int64(2)
memory usage: 402.3 KB
```

```
/tmp/ipykernel_115/236554556.py:2: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  data['Date'] =  pd.to_datetime(data['Date'])
```

[35]:
```
# checking for missing values
data.isnull().sum()
```

[35]:
```
Store           0
Date            0
Weekly_Sales    0
Holiday_Flag    0
Temperature     0
Fuel_Price      0
CPI             0
Unemployment    0
dtype: int64
```

[36]:
```
# Splitting Date and create new columns (Day, Month, and Year)
data["Day"]= pd.DatetimeIndex(data['Date']).day
data['Month'] = pd.DatetimeIndex(data['Date']).month
data['Year'] = pd.DatetimeIndex(data['Date']).year
data
```

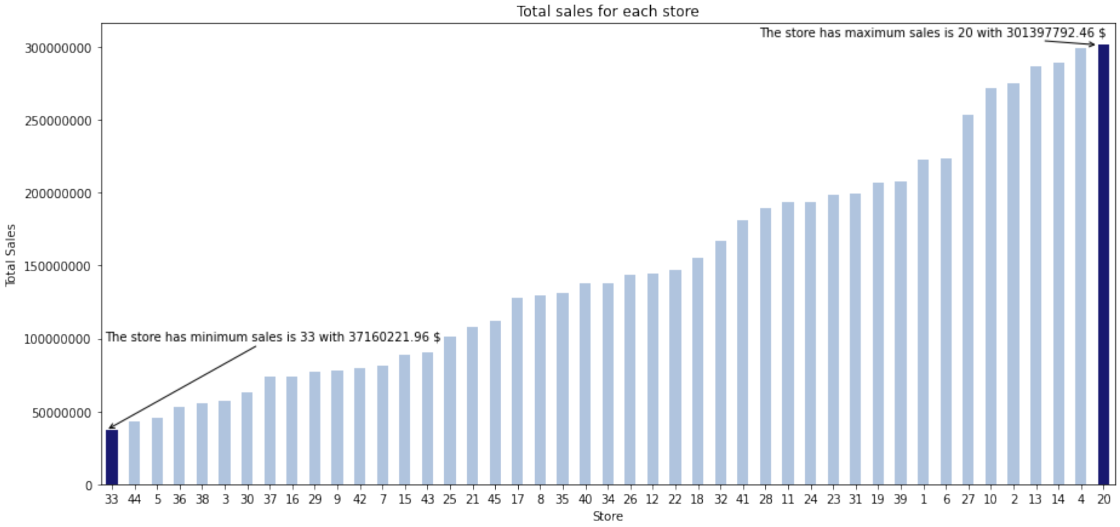
[36]:
```
      Store       Date  Weekly_Sales  Holiday_Flag  Temperature  Fuel_Price  \
0         1 2010-05-02    1643690.90             0        42.31       2.572
1         1 2010-12-02    1641957.44             1        38.51       2.548
2         1 2010-02-19    1611968.17             0        39.93       2.514
```

```
3          1 2010-02-26    1409727.59             0        46.63        2.561
4          1 2010-05-03    1554806.68             0        46.50        2.625
…      …         …             …            …           …          …
6430      45 2012-09-28     713173.95             0        64.88        3.997
6431      45 2012-05-10     733455.07             0        64.89        3.985
6432      45 2012-12-10     734464.36             0        54.47        4.000
6433      45 2012-10-19     718125.53             0        56.47        3.969
6434      45 2012-10-26     760281.43             0        58.85        3.882

             CPI  Unemployment  Day  Month  Year
0     211.096358         8.106    2      5  2010
1     211.242170         8.106    2     12  2010
2     211.289143         8.106   19      2  2010
3     211.319643         8.106   26      2  2010
4     211.350143         8.106    3      5  2010
…            …             …   …    …    …
6430  192.013558         8.684   28      9  2012
6431  192.170412         8.667   10      5  2012
6432  192.327265         8.667   10     12  2012
6433  192.330854         8.667   19     10  2012
6434  192.308899         8.667   26     10  2012

[6435 rows x 11 columns]
```

[37]:

```
plt.figure(figsize=(15,7))

# Sum Weekly_Sales for each store, then sortded by total sales
total_sales_for_each_store = data.groupby('Store')['Weekly_Sales'].sum().
 ↪sort_values()
total_sales_for_each_store_array = np.array(total_sales_for_each_store) #␣
 ↪convert to array

# Assigning a specific color for the stores have the lowest and highest sales
clrs = ['lightsteelblue' if ((x < max(total_sales_for_each_store_array)) and (x␣
 ↪> min(total_sales_for_each_store_array))) else 'midnightblue' for x in␣
 ↪total_sales_for_each_store_array]



ax = total_sales_for_each_store.plot(kind='bar',color=clrs);

# store have minimum sales
p = ax.patches[0]
print(type(p.get_height()))
ax.annotate("The store has minimum sales is 33 with {0:.2f} $".format((p.
 ↪get_height())), xy=(p.get_x(), p.get_height()), xycoords='data',
            xytext=(0.17, 0.32), textcoords='axes fraction',
            arrowprops=dict(arrowstyle="->", connectionstyle="arc3"),
            horizontalalignment='center', verticalalignment='center')


# 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')


# plot properties
plt.xticks(rotation=0)
plt.ticklabel_format(useOffset=False, style='plain', axis='y')
plt.title('Total sales for each store')
plt.xlabel('Store')
plt.ylabel('Total Sales');
```

```
<class 'numpy.float64'>
```



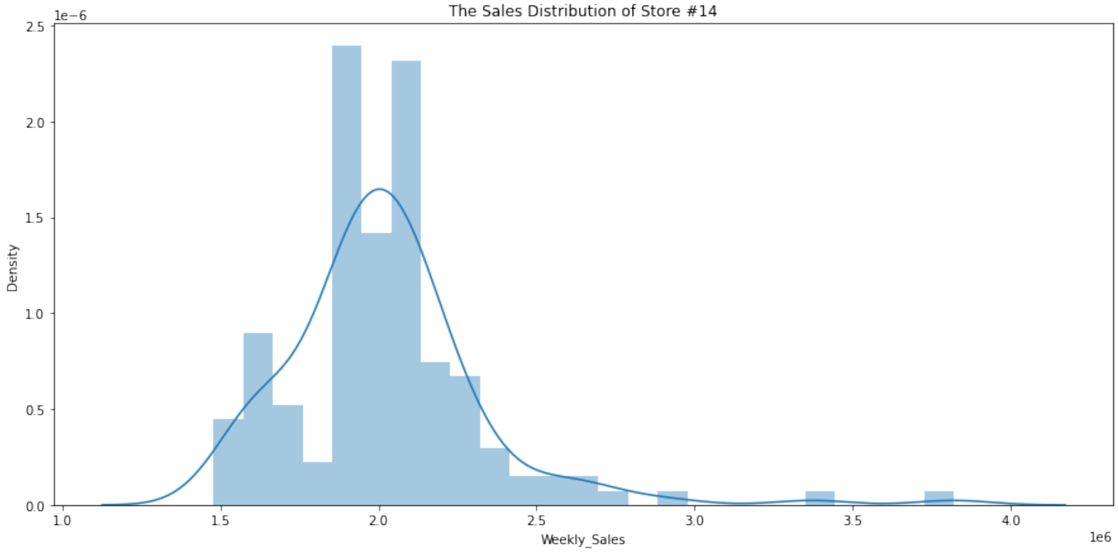
[38]:
```
# Which store has maximum standard deviation
data_std = pd.DataFrame(data.groupby('Store')['Weekly_Sales'].std().
 ↪sort_values(ascending=False))
print("The store has maximum standard deviation is "+str(data_std.head(1).
 ↪index[0])+" with {0:.0f} $".format(data_std.head(1).Weekly_Sales[data_std.
 ↪head(1).index[0]]))
```

```
The store has maximum standard deviation is 14 with 317570 $
```

[39]:

```
# Distribution of store has maximum standard deviation
plt.figure(figsize=(15,7))
sns.distplot(data[data['Store'] == data_std.head(1).index[0]]['Weekly_Sales'])
plt.title('The Sales Distribution of Store #'+ str(data_std.head(1).index[0]));
```

```
/tmp/ipykernel_115/3470610508.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data[data['Store'] == data_std.head(1).index[0]]['Weekly_Sales'])
```



[40]:

```
# Coefficient of mean to standard deviation
coef_mean_std = pd.DataFrame(data.groupby('Store')['Weekly_Sales'].std() / data.
 ↪groupby('Store')['Weekly_Sales'].mean())
coef_mean_std = coef_mean_std.rename(columns={'Weekly_Sales':'Coefficient of␣
 ↪mean to standard deviation'})
coef_mean_std
```

[40]:

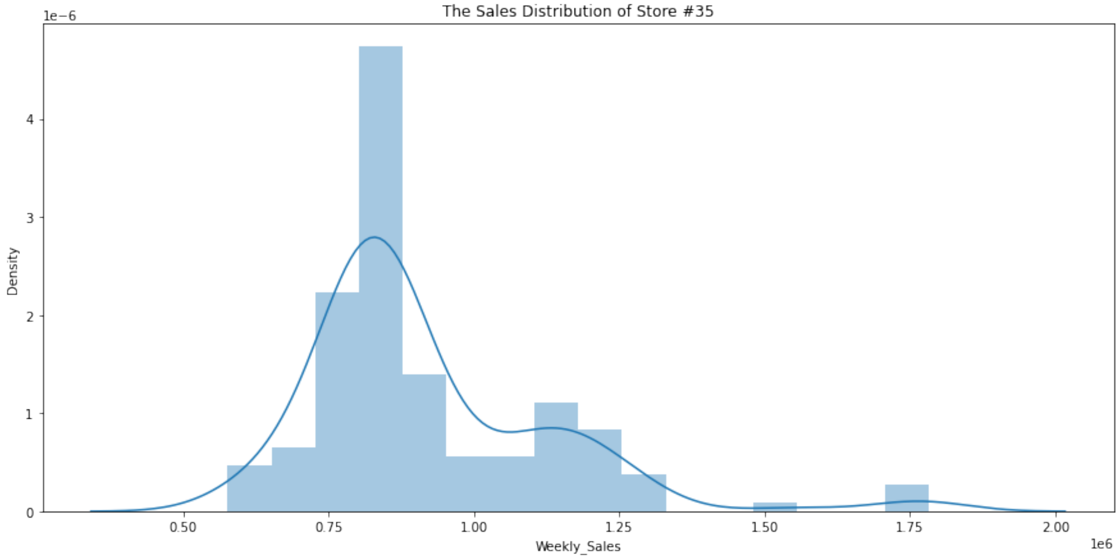
```
       Coefficient of mean to standard deviation
Store
1                                       0.100292
2                                       0.123424
```

```
3                                        0.115021
4                                        0.127083
5                                        0.118668
6                                        0.135823
7                                        0.197305
8                                        0.116953
9                                        0.126895
10                                       0.159133
11                                       0.122262
12                                       0.137925
13                                       0.132514
14                                       0.157137
15                                       0.193384
16                                       0.165181
17                                       0.125521
18                                       0.162845
19                                       0.132680
20                                       0.130903
21                                       0.170292
22                                       0.156783
23                                       0.179721
24                                       0.123637
25                                       0.159860
26                                       0.110111
27                                       0.135155
28                                       0.137330
29                                       0.183742
30                                       0.052008
31                                       0.090161
32                                       0.118310
33                                       0.092868
34                                       0.108225
35                                       0.229681
36                                       0.162579
37                                       0.042084
38                                       0.110875
39                                       0.149908
40                                       0.123430
41                                       0.148177
42                                       0.090335
43                                       0.064104
44                                       0.081793
45                                       0.165613
```

[41]:
```
# Distribution of store has maximum coefficient of mean to standard deviation
coef_mean_std_max = coef_mean_std.sort_values(by='Coefficient of mean to␣
 ↪standard deviation')
```

```
plt.figure(figsize=(15,7))
sns.distplot(data[data['Store'] == coef_mean_std_max.tail(1).
 ↪index[0]]['Weekly_Sales'])
plt.title('The Sales Distribution of Store #'+str(coef_mean_std_max.tail(1).
 ↪index[0]));
```

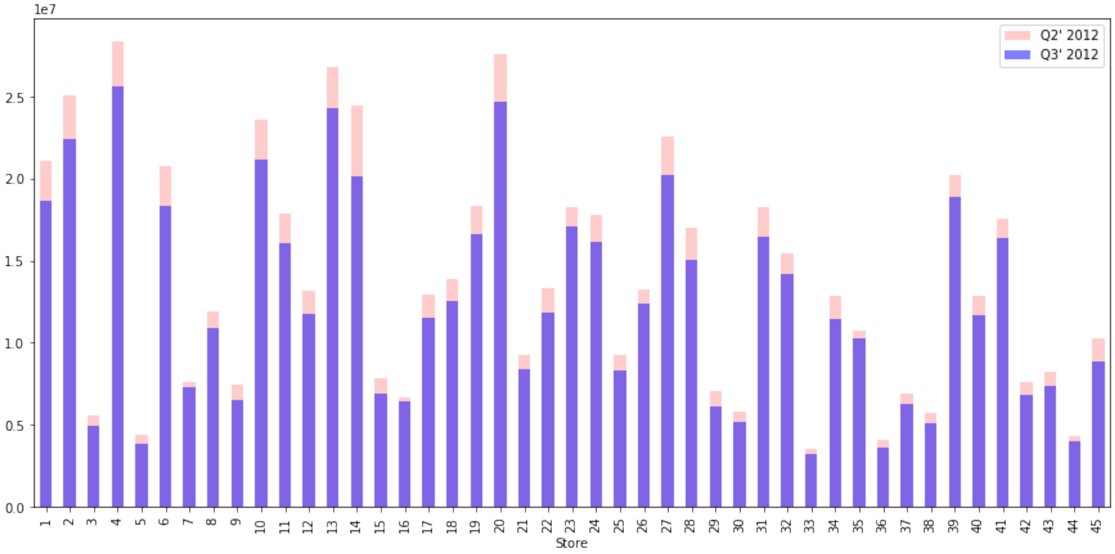
```
/tmp/ipykernel_115/1932089423.py:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data[data['Store'] ==
coef_mean_std_max.tail(1).index[0]]['Weekly_Sales'])
```



[42]:

```
plt.figure(figsize=(15,7))

# Sales for third quarter in 2012
Q3 = data[(data['Date'] > '2012-07-01') & (data['Date'] < '2012-09-30')].
 ↪groupby('Store')['Weekly_Sales'].sum()

# Sales for second quarter in 2012
Q2 = data[(data['Date'] > '2012-04-01') & (data['Date'] < '2012-06-30')].
 ↪groupby('Store')['Weekly_Sales'].sum()
```

```
# Plotting the difference between sales for second and third quarterly
ax = Q2.plot(kind='bar', color='r', alpha=0.2)
Q3.plot(ax=ax, kind='bar', color='b', alpha=0.5)

plt.legend(["Q2' 2012", "Q3' 2012"])
plt.show()
```



[43]:
```
#  store/s has good quarterly growth rate in Q3’2012 - .
 ↪sort_values(by='Weekly_Sales')
print('Store have good quarterly growth rate in Q3’2012 is Store '+str(Q3.
 ↪idxmax())+' With '+str(Q3.max())+' $')
```

```
Store have good quarterly growth rate in Q3’2012 is Store 4 With 25652119.35 $
```

[44]:
```
def get_sales_on_holidays(df, holiday_dates):
    holiday_sales = []
    for day in holiday_dates:
        day = datetime.strptime(day, '%d-%m-%Y')
        sales_on_day = df[df['Date'] == day]['Weekly_Sales'].values
        if len(sales_on_day) > 0:
            holiday_sales.append(sales_on_day[0])
        else:
            holiday_sales.append(0)
    return holiday_sales

def plot_line(df, holiday_dates, holiday_label):
    holiday_sales = get_sales_on_holidays(df, holiday_dates)
    return holiday_sales

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

super_bowl_sales = plot_line(total_sales, Super_Bowl, 'Super Bowl')
labour_day_sales = plot_line(total_sales, Labour_Day, 'Labour Day')
thanksgiving_sales = plot_line(total_sales, Thanksgiving, 'Thanksgiving')
christmas_sales = plot_line(total_sales, Christmas, 'Christmas')

print("Super Bowl Sales:", super_bowl_sales)
print("Labour Day Sales:", labour_day_sales)
print("Thanksgiving Sales:", thanksgiving_sales)
print("Christmas Sales:", christmas_sales)
```

```
Super Bowl Sales: [0, 0, 0]
Labour Day Sales: [0, 46763227.53, 0]
Thanksgiving Sales: [65821003.24, 66593605.26, 0]
Christmas Sales: [40432519.0, 46042461.04, 0]
```

[45]:

```
data.loc[data.Date.isin(Super_Bowl)]
```

[45]:

```
      Store       Date  Weekly_Sales  Holiday_Flag  Temperature  Fuel_Price  \
1         1 2010-12-02    1641957.44             1        38.51       2.548
53        1 2011-11-02    1649614.93             1        36.39       3.022
105       1 2012-10-02    1802477.43             1        48.02       3.409
144       2 2010-12-02    2137809.50             1        38.49       2.548
196       2 2011-11-02    2168041.61             1        33.19       3.022
…     …     …           …             …            …           …
6202     44 2011-11-02     307486.73             1        30.83       3.034
6254     44 2012-10-02     325377.97             1        33.73       3.116
6293     45 2010-12-02     656988.64             1        27.73       2.773
6345     45 2011-11-02     766456.00             1        30.30       3.239
6397     45 2012-10-02     803657.12             1        37.00       3.640

             CPI  Unemployment  Day  Month  Year
1     211.242170         8.106    2     12  2010
53    212.936705         7.742    2     11  2011
105   220.265178         7.348    2     10  2012
144   210.897994         8.324    2     12  2010
196   212.592862         8.028    2     11  2011
…          …            …  …    …    …
6202  127.859129         7.224    2     11  2011
6254  130.384903         5.774    2     10  2012
```

```
6293   181.982317         8.992    2     12   2010
6345   183.701613         8.549    2     11   2011
6397   189.707605         8.424    2     10   2012

[135 rows x 11 columns]
```

[46]:

```
Super_Bowl_df = pd.DataFrame(data.loc[data.Date.isin(Super_Bowl)].
 ↪groupby('Year')['Weekly_Sales'].sum())
Thanksgiving_df = pd.DataFrame(data.loc[data.Date.isin(Thanksgiving)].
 ↪groupby('Year')['Weekly_Sales'].sum())
Labour_Day_df = pd.DataFrame(data.loc[data.Date.isin(Labour_Day)].
 ↪groupby('Year')['Weekly_Sales'].sum())
Christmas_df = pd.DataFrame(data.loc[data.Date.isin(Christmas)].
 ↪groupby('Year')['Weekly_Sales'].sum())

print("Yearly Sales in Super Bowl holiday:")
print(Super_Bowl_df)

print("\nYearly Sales in Thanksgiving holiday:")
print(Thanksgiving_df)

print("\nYearly Sales in Labour Day holiday:")
print(Labour_Day_df)

print("\nYearly Sales in Christmas holiday:")
print(Christmas_df)
```

```
Yearly Sales in Super Bowl holiday:
      Weekly_Sales
Year
2010   48336677.63
2011   47336192.79
2012   50009407.92

Yearly Sales in Thanksgiving holiday:
      Weekly_Sales
Year
2010   65821003.24
2011   66593605.26

Yearly Sales in Labour Day holiday:
      Weekly_Sales
Year
2010   45634397.84
2011   46763227.53
2012   48330059.31
```

```
Yearly Sales in Christmas holiday:
      Weekly_Sales
Year
2010   40432519.00
2011   46042461.04
```

```
/tmp/ipykernel_115/1994519508.py:2: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  Thanksgiving_df = pd.DataFrame(data.loc[data.Date.isin(Thanksgiving)].groupby(
'Year')['Weekly_Sales'].sum())
/tmp/ipykernel_115/1994519508.py:4: UserWarning: Parsing dates in DD/MM/YYYY
format when dayfirst=False (the default) was specified. This may lead to
inconsistently parsed dates! Specify a format to ensure consistent parsing.
  Christmas_df = pd.DataFrame(data.loc[data.Date.isin(Christmas)].groupby('Year'
)['Weekly_Sales'].sum())
```

[47]:

```
# Monthly view of sales for each year
sales_2010 = data[data.Year == 2010].groupby('Month')['Weekly_Sales'].sum()
sales_2011 = data[data.Year == 2011].groupby('Month')['Weekly_Sales'].sum()
sales_2012 = data[data.Year == 2012].groupby('Month')['Weekly_Sales'].sum()

print("Monthly view of sales in 2010:")
print(sales_2010)

print("\nMonthly view of sales in 2011:")
print(sales_2011)

print("\nMonthly view of sales in 2012:")
print(sales_2012)
```

```
Monthly view of sales in 2010:
Month
1     4.223988e+07
2     1.915869e+08
3     1.862262e+08
4     1.838118e+08
5     2.806119e+08
6     1.424361e+08
7     1.842664e+08
8     1.845381e+08
9     1.797041e+08
10    2.311201e+08
11    1.587731e+08
12    3.235716e+08
Name: Weekly_Sales, dtype: float64

Monthly view of sales in 2011:
```

```
Month
1     2.119657e+08
2     1.876092e+08
3     1.365205e+08
4     2.789693e+08
5     1.828017e+08
6     1.401936e+08
7     2.244611e+08
8     1.880810e+08
9     2.310323e+08
10    1.837193e+08
11    2.534703e+08
12    2.293760e+08
Name: Weekly_Sales, dtype: float64

Monthly view of sales in 2012:
Month
1     1.722207e+08
2     1.428296e+08
3     2.307397e+08
4     1.825428e+08
5     1.422830e+08
6     2.923883e+08
7     1.845865e+08
8     1.916126e+08
9     1.797959e+08
10    1.880794e+08
11    4.692588e+07
12    4.612851e+07
Name: Weekly_Sales, dtype: float64
```

[48]:

```python
# Monthly view of sales for all years
monthly_sales = data.groupby('Month')['Weekly_Sales'].sum()

print("Monthly view of sales:")
print(monthly_sales)
```

```
Monthly view of sales:
Month
1     4.264263e+08
2     5.220257e+08
3     5.534864e+08
4     6.453239e+08
5     6.056966e+08
6     5.750180e+08
7     5.933139e+08
8     5.642317e+08
9     5.905323e+08
```

```
10    6.029189e+08
11    4.591693e+08
12    5.990761e+08
Name: Weekly_Sales, dtype: float64
```

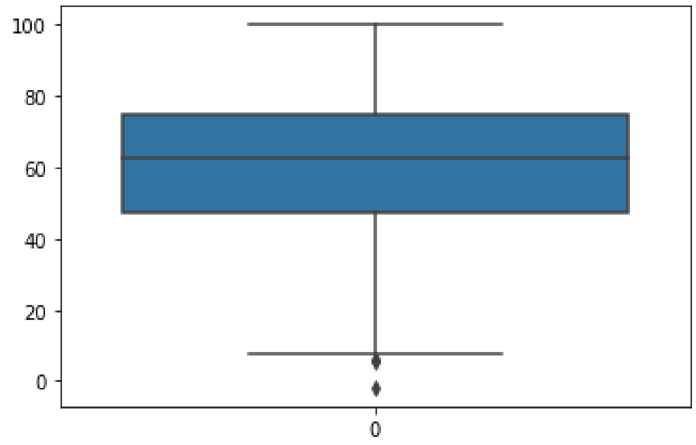
[49]:

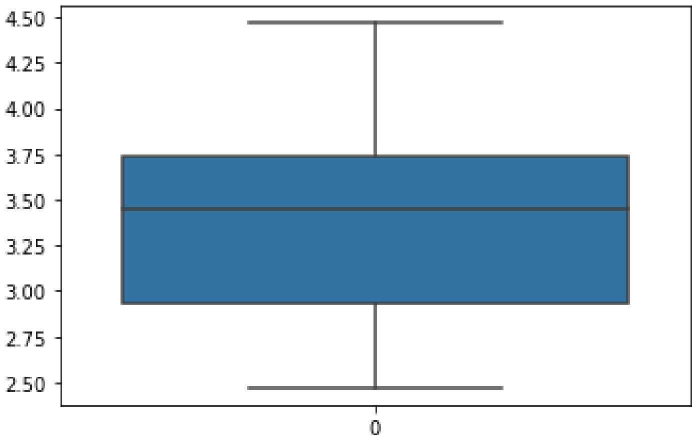
```
# Yearly view of sales
yearly_sales = data.groupby("Year")[["Weekly_Sales"]].sum()

print("Yearly view of sales:")
print(yearly_sales)
```

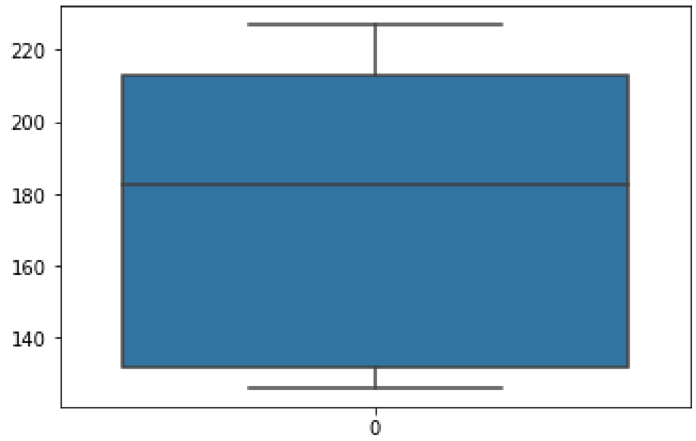
```
Yearly view of sales:
      Weekly_Sales
Year
2010  2.288886e+09
2011  2.448200e+09
2012  2.000133e+09
```

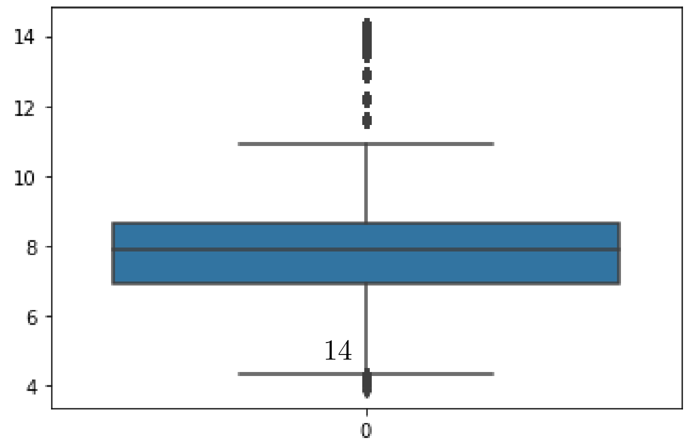
[50]:

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


100
80
60
40
20
0
0


4.50
4.25
4.00
3.75
3.50
3.25
3.00
2.75
2.50
0

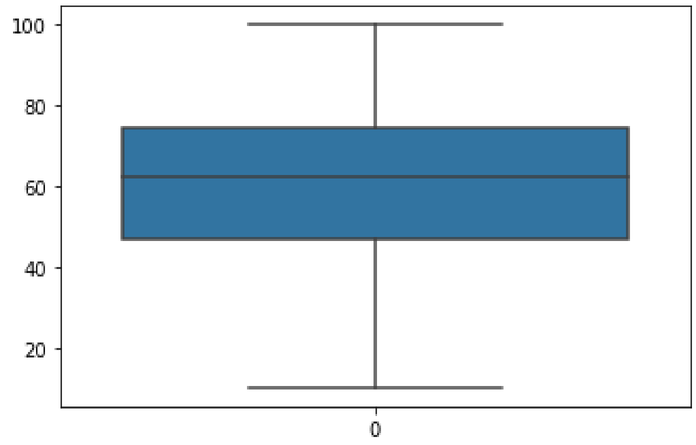

220
200
180
160
140
0

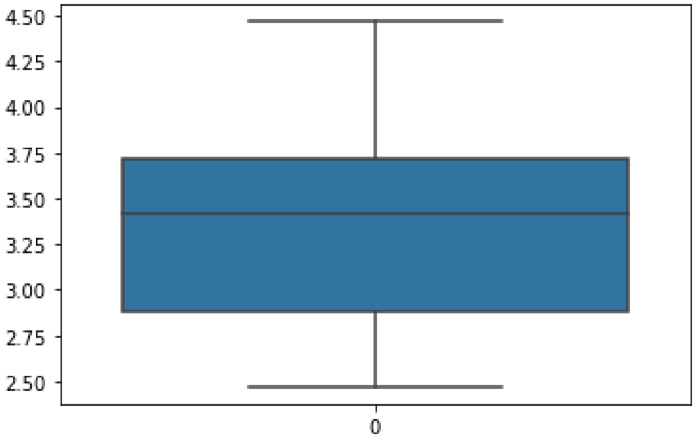

14
12
10
8
6
4
0

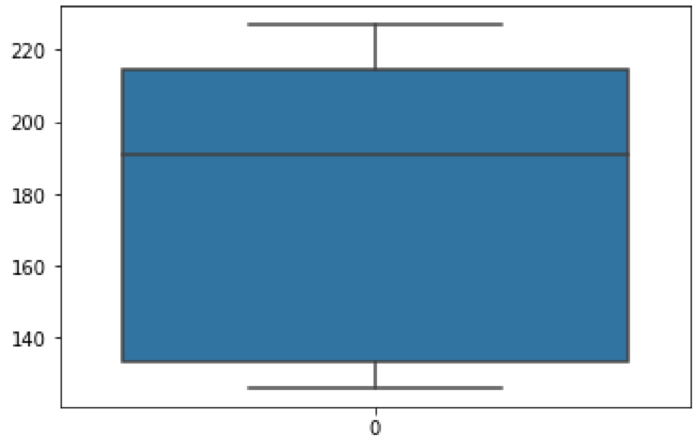
[51]:
```
# drop the outliers
data_new = data[(data['Unemployment']<10) & (data['Unemployment']>4.5) &␣
 ↪(data['Temperature']>10)]
data_new
```

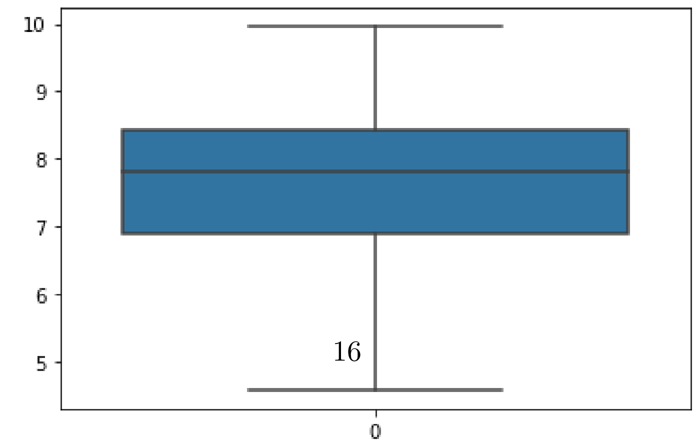
[51]:
```
      Store        Date  Weekly_Sales  Holiday_Flag  Temperature  Fuel_Price  \
0         1  2010-05-02    1643690.90             0        42.31       2.572
1         1  2010-12-02    1641957.44             1        38.51       2.548
2         1  2010-02-19    1611968.17             0        39.93       2.514
3         1  2010-02-26    1409727.59             0        46.63       2.561
4         1  2010-05-03    1554806.68             0        46.50       2.625
…       …         …             …           …            …           …
6430     45  2012-09-28     713173.95             0        64.88       3.997
6431     45  2012-05-10     733455.07             0        64.89       3.985
6432     45  2012-12-10     734464.36             0        54.47       4.000
6433     45  2012-10-19     718125.53             0        56.47       3.969
6434     45  2012-10-26     760281.43             0        58.85       3.882

             CPI  Unemployment  Day  Month  Year
0     211.096358         8.106    2      5  2010
1     211.242170         8.106    2     12  2010
2     211.289143         8.106   19      2  2010
3     211.319643         8.106   26      2  2010
4     211.350143         8.106    3      5  2010
…            …            …   …    …    …
6430  192.013558         8.684   28      9  2012
6431  192.170412         8.667   10      5  2012
6432  192.327265         8.667   10     12  2012
6433  192.330854         8.667   19     10  2012
6434  192.308899         8.667   26     10  2012

[5658 rows x 11 columns]
```

[52]:
```
# check outliers
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])
```


100
80
60
40
20
0


4.50
4.25
4.00
3.75
3.50
3.25
3.00
2.75
2.50
0


220
200
180
160
140
0


10
9
8
7
6
5
0

[53]:

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

[54]:

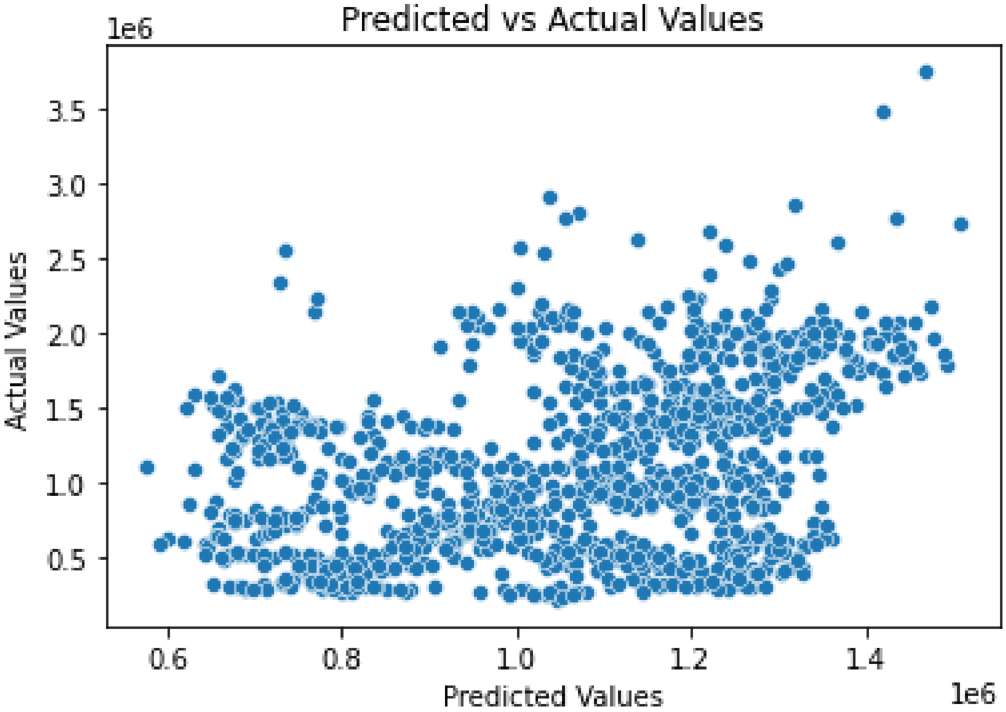
```python
# Select features and target
X = data_new[['Store','Fuel_Price','CPI','Unemployment','Day','Month','Year']]
y = data_new['Weekly_Sales']

# Split data to train and test (0.80:0.20)
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
```

[55]:

```python
# Linear Regression model
print('Linear Regression:')
print()

# Initialize the Linear Regression model
reg = LinearRegression()

# Fit the model on the training data
reg.fit(X_train, y_train)

# Make predictions on the test data
y_pred = reg.predict(X_test)

# Evaluate the model
print('Accuracy:', reg.score(X_train, y_train) * 100)
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,␣
 ↪y_pred)))

# Visualize predicted vs actual values
sns.scatterplot(x=y_pred, y=y_test)
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.title('Predicted vs Actual Values')
plt.show()
```

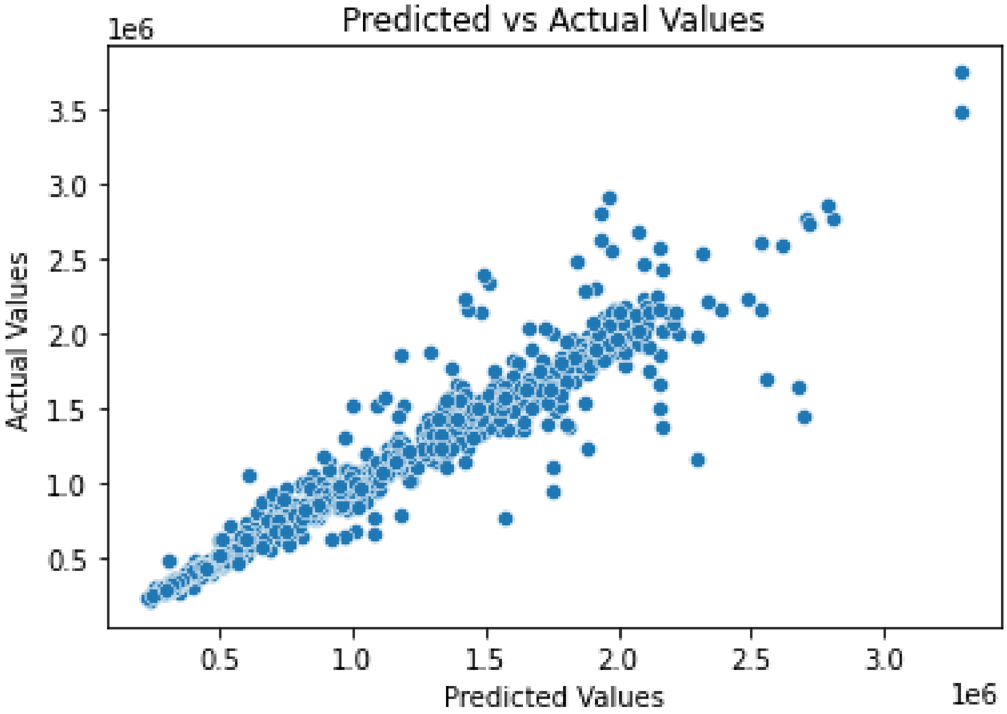
```
Linear Regression:

Accuracy: 13.082119799395276
Mean Absolute Error: 449517.68201199826
Mean Squared Error: 298143089401.3515
```

```
Root Mean Squared Error: 546024.8065805724
```



```
[56]: # Random Forest Regressor
      print('Random Forest Regressor:')
      print()

      # Initialize the 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)

      print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
      print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
      print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,␣
       ↪y_pred)))

      # Visualize predicted vs actual values
      sns.scatterplot(x=y_pred, y=y_test)
      plt.xlabel('Predicted Values')
      plt.ylabel('Actual Values')
      plt.title('Predicted vs Actual Values')
      plt.show()
```

```
Random Forest Regressor:

Accuracy: 93.37186765440984
Mean Absolute Error: 74764.93965668576
Mean Squared Error: 22429407129.753437
Root Mean Squared Error: 149764.50557376217
```



```
[ ]:
```

```
[ ]:
```

```
[ ]:
```