Waltmart data set

Historical sales data for 45 Walmart stores located in different regions are available. There are certain events and holidays which impact sales on each day. The business is facing a challenge due to unforeseen demands and runs out of stock some times. Walmart would like to predict the sales and demand accurately.

Data set contain the folling details of the 45 wallamrat stores for 143 weeks

- Store Number
- Weekly sales
- Holidat status
- Temperature
- Fuel prices
- CPI
- Unemployment

Import Dependencies

In [42]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

# Make NumPy printouts easier to read.
np.set_printoptions(precision=3, suppress=True)
import csv

from sklearn.model_selection import train_test_split
```

Import the dataset

In [6]:
```
data= pd.read_csv("Walmart_Store_sales.csv")
```

In [7]:
```
data
```

Out[7]:

	Store	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
0	1	1643690.90	0	42.31	2.572	211.096358	8.106
1	1	1641957.44	1	38.51	2.548	211.242170	8.106
2	1	1611968.17	0	39.93	2.514	211.289143	8.106
3	1	1409727.59	0	46.63	2.561	211.319643	8.106

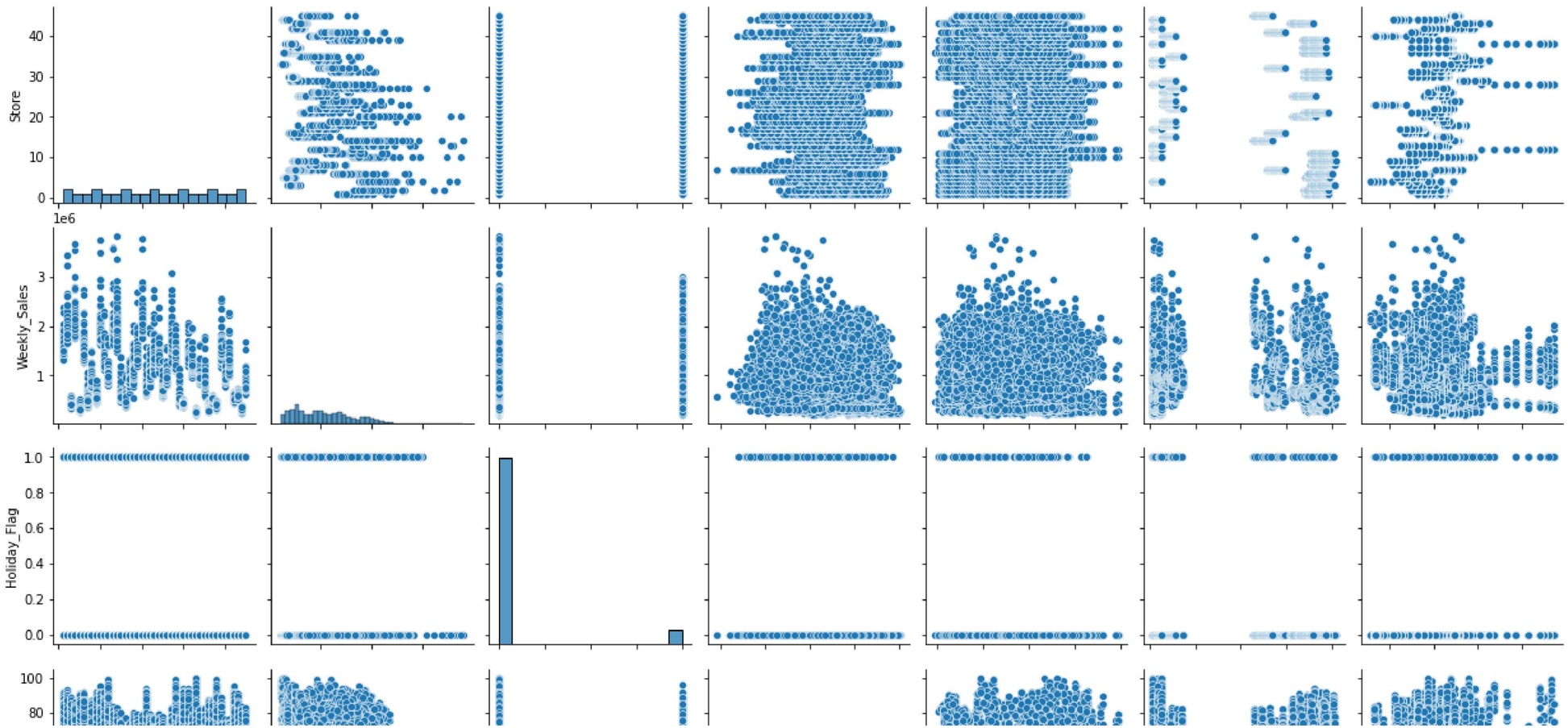
	Store	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
4	1	1554806.68	0	46.50	2.625	211.350143	8.106
...	...	...	...	...	...	...	...
6430	45	713173.95	0	64.88	3.997	192.013558	8.684
6431	45	733455.07	0	64.89	3.985	192.170412	8.667
6432	45	734464.36	0	54.47	4.000	192.327265	8.667
6433	45	718125.53	0	56.47	3.969	192.330854	8.667
6434	45	760281.43	0	58.85	3.882	192.308899	8.667

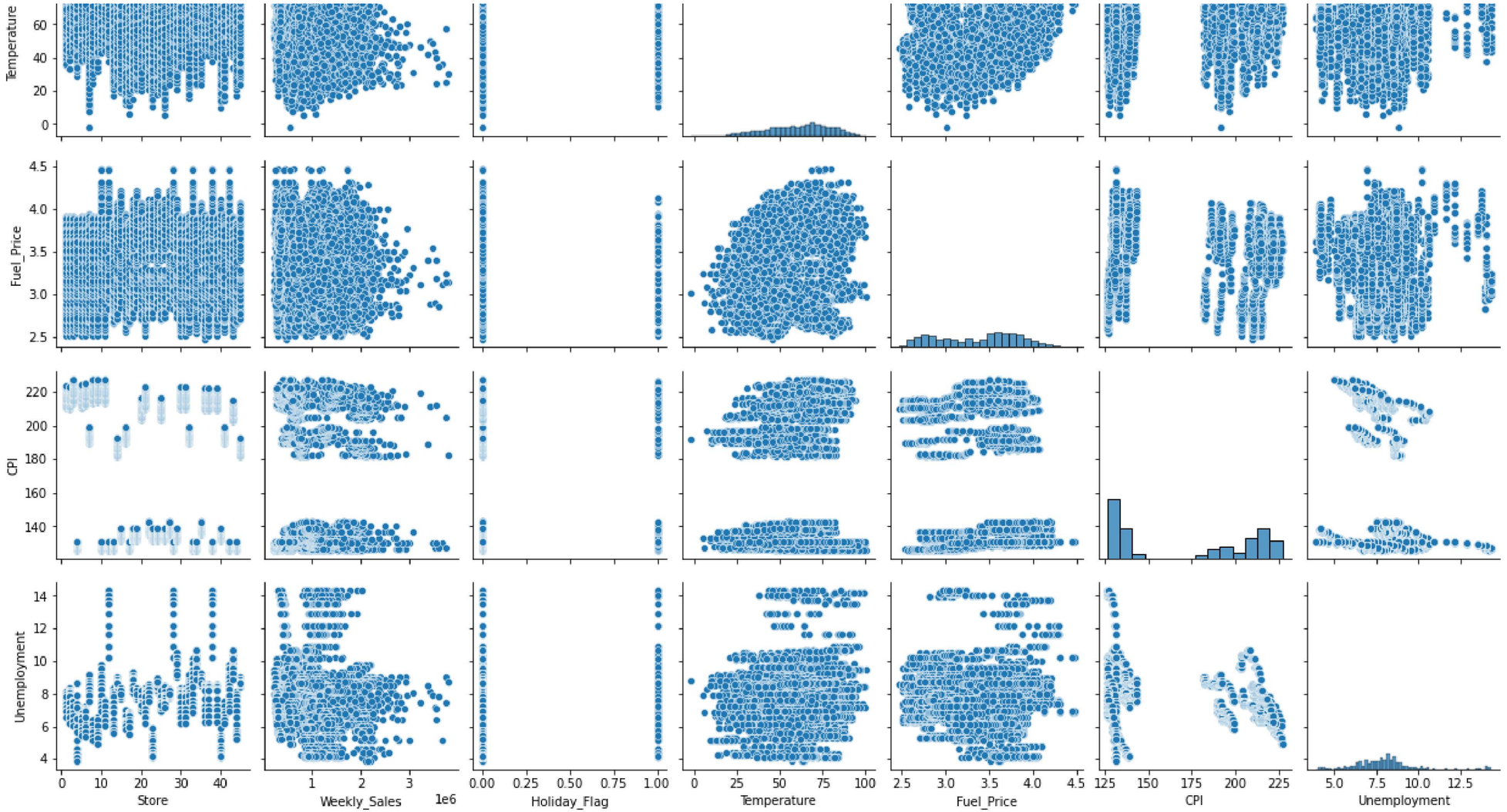
6435 rows × 7 columns

Data Annalysis

In [8]:
```
sns.pairplot(data)
```

Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x2099d15ad60>
```





IT seems weekly sales have some intresting correation with all the avaialabe variables

In [9]:
```
data.describe()
```

Out[9]:

	Store	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
count	6435.000000	6.435000e+03	6435.000000	6435.000000	6435.000000	6435.000000	6435.000000
mean	23.000000	1.046965e+06	0.069930	60.663782	3.358607	171.578394	7.999151
std	12.988182	5.643666e+05	0.255049	18.444933	0.459020	39.356712	1.875885
min	1.000000	2.099862e+05	0.000000	-2.060000	2.472000	126.064000	3.879000
25%	12.000000	5.533501e+05	0.000000	47.460000	2.933000	131.735000	6.891000
50%	23.000000	9.607460e+05	0.000000	62.670000	3.445000	182.616521	7.874000

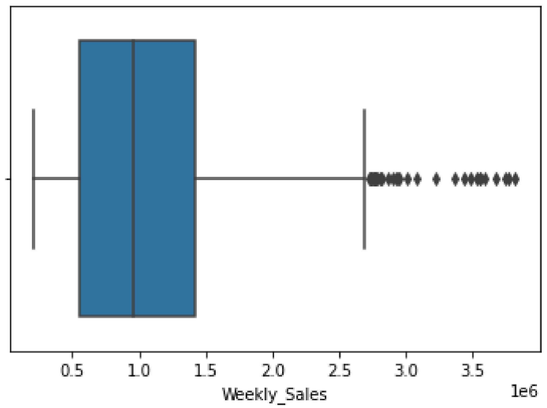
	Store	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
75%	34.000000	1.420159e+06	0.000000	74.940000	3.735000	212.743293	8.622000
max	45.000000	3.818686e+06	1.000000	100.140000	4.468000	227.232807	14.313000

box plots

In [15]:
```
sns.boxplot(data['Weekly_Sales'])
```

```
Z:\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passin
g other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
<AxesSubplot:xlabel='Weekly_Sales'>
```

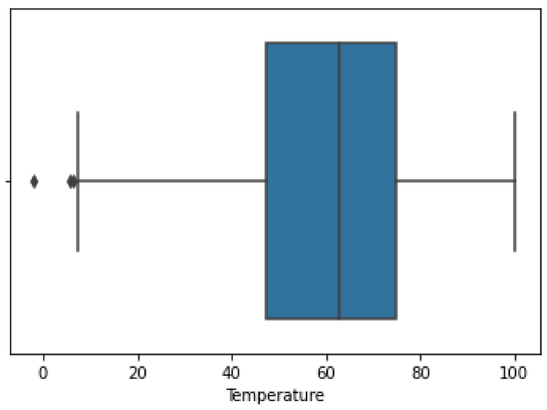
Out[15]:



In [16]:
```
sns.boxplot(data['Temperature'])
```

```
Z:\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passin
g other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
<AxesSubplot:xlabel='Temperature'>
```

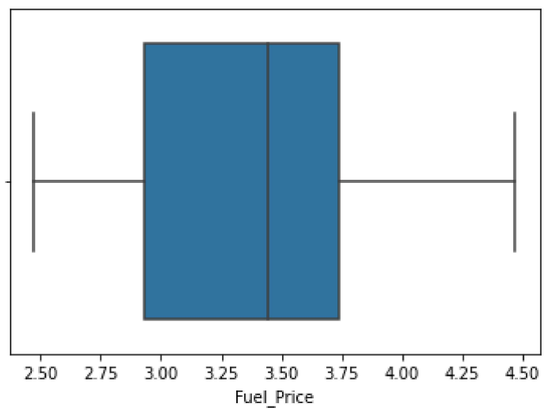
Out[16]:



In [17]:
```
sns.boxplot(data['Fuel_Price'])
```

```
Z:\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passin
g other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

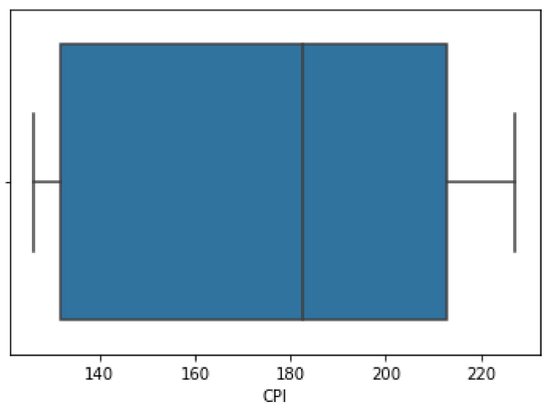
Out[17]:
```
<AxesSubplot:xlabel='Fuel_Price'>
```



In [18]:
```
sns.boxplot(data['CPI'])
```

```
Z:\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passin
g other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

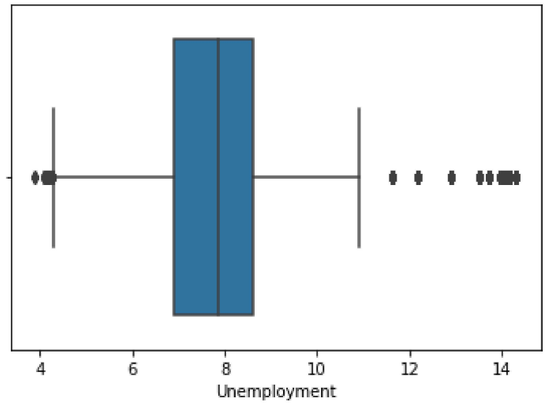
Out[18]:
```
<AxesSubplot:xlabel='CPI'>
```



In [19]:
```
sns.boxplot(data['Unemployment'])
```

```
Z:\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passin
g other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[19]:
```
<AxesSubplot:xlabel='Unemployment'>
```



Basic Info

In [22]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6435 entries, 0 to 6434
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: 352.0 KB
```

Correaltion annalysis

In [35]:
```
corr = data.corr()
```

In [36]:
```
sns.heatmap(corr)
```

Out[36]:
```
<AxesSubplot:>
```

In [37]:
```
corr
```

Out[37]:

	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

Conclusion

There are no null values so we dont have to eliminate null values. Futher eventhough box plots indicates that there are several outliers it is worthy to keep them becuase they might be unique environmental factors.

Intrestingle correaltion seems pretty low.

Thus as the dataset has fewer fields and fewer data it is recormended to use the data set without diemensionality reduction

Further regression models will be sufficient for the application becuase of the size of the data set

Building the regressive models

In [39]:
```
X = data.drop('Weekly_Sales', axis=1)
y = data['Weekly_Sales']
```

In [40]:
```
X
```

Out[40]:

	Store	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment
0	1	0	42.31	2.572	211.096358	8.106
1	1	1	38.51	2.548	211.242170	8.106
2	1	0	39.93	2.514	211.289143	8.106
3	1	0	46.63	2.561	211.319643	8.106
4	1	0	46.50	2.625	211.350143	8.106
...	...	...	...	...	...	...
6430	45	0	64.88	3.997	192.013558	8.684
6431	45	0	64.89	3.985	192.170412	8.667
6432	45	0	54.47	4.000	192.327265	8.667
6433	45	0	56.47	3.969	192.330854	8.667
6434	45	0	58.85	3.882	192.308899	8.667

6435 rows × 6 columns

In [41]:
```
y
```

Out[41]:
```
0       1643690.90
1       1641957.44
2       1611968.17
3       1409727.59
4       1554806.68
           ...
6430     713173.95
6431     733455.07
6432     734464.36
6433     718125.53
6434     760281.43
Name: Weekly_Sales, Length: 6435, dtype: float64
```

In [ ]:
```
# split the data set as train and test
```

In [43]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=1234)
```

In [44]:
```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(4504, 6) (1931, 6) (4504,) (1931,)
```

Importing Dependencies

In [173…
```
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression,Ridge, Lasso, ElasticNet
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
```

In [174…
```
pipelines = {
    'rf':make_pipeline(RandomForestRegressor(random_state=1234)),
    'gb':make_pipeline(GradientBoostingRegressor(random_state=1234)),
    'ridge':make_pipeline(Ridge(random_state=1234)),
    'lasso':make_pipeline(Lasso(random_state=1234)),
    'enet':make_pipeline(ElasticNet(random_state=1234)),
}
```

In [175…
```
LogisticRegression().get_params()
```

Out[175…
```
{'C': 1.0,
 'class_weight': None,
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'l1_ratio': None,
 'max_iter': 100,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': None,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

In [176…
```
Ridge().get_params()
```

Out[176…
```
{'alpha': 1.0,
 'copy_X': True,
 'fit_intercept': True,
 'max_iter': None,
 'normalize': False,
 'random_state': None,
 'solver': 'auto',
 'tol': 0.001}
```

In [177…
```
Lasso().get_params()
```

Out[177…
```
{'alpha': 1.0,
 'copy_X': True,
 'fit_intercept': True,
 'max_iter': 1000,
 'normalize': False,
 'positive': False,
 'precompute': False,
 'random_state': None,
 'selection': 'cyclic',
 'tol': 0.0001,
 'warm_start': False}
```

In [178…
```
ElasticNet().get_params()
```

Out[178…
```
{'alpha': 1.0,
 'copy_X': True,
 'fit_intercept': True,
```

```
 'l1_ratio': 0.5,
 'max_iter': 1000,
 'normalize': False,
 'positive': False,
 'precompute': False,
 'random_state': None,
 'selection': 'cyclic',
 'tol': 0.0001,
 'warm_start': False}
```

In [179…

```
GradientBoostingRegressor().get_params()
```

Out[179…

```
{'alpha': 0.9,
 'ccp_alpha': 0.0,
 'criterion': 'friedman_mse',
 'init': None,
 'learning_rate': 0.1,
 'loss': 'ls',
 'max_depth': 3,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_iter_no_change': None,
 'random_state': None,
 'subsample': 1.0,
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': 0,
 'warm_start': False}
```

In [180…

```
hypergrid = {
    'lr': {
        'lr__max__iter':[15,100,400]
    },
    'gb':{
        'gradientboostingregressor__alpha':[0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 0.99]
    },
    'ridge':{
        'ridge__alpha':[0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 0.99]
    },
    'lasso':{
        'lasso__alpha':[0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 0.99]
    },
    'enet':{
        'elasticnet__alpha':[0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 0.99]
    }
}
```

In [181…

```
fit_models = {}
for algo, pipeline in pipelines.items():
    model = pipeline.fit(X_train, y_train)
    fit_models[algo] = model
```

In [182…
```
fit_models
```

Out[182…
```
{'rf': Pipeline(steps=[('randomforestregressor',
                        RandomForestRegressor(random_state=1234))]),
 'gb': Pipeline(steps=[('gradientboostingregressor',
                        GradientBoostingRegressor(random_state=1234))]),
 'ridge': Pipeline(steps=[('ridge', Ridge(random_state=1234))]),
 'lasso': Pipeline(steps=[('lasso', Lasso(random_state=1234))]),
 'enet': Pipeline(steps=[('elasticnet', ElasticNet(random_state=1234))])}
```

Evaluvating the models

In [183…
```
from sklearn.metrics import r2_score, mean_absolute_error
import pickle
```

In [185…
```
for algo,model in fit_models.items():
    yhat = model.predict(X_test)
    print('{} scores - R2:{} MAE:{}'.format(algo, r2_score(y_test, yhat), mean_absolute_error(y_test, yhat)))
```

```
rf scores - R2:0.9378389540396331 MAE:75916.16422718797
gb scores - R2:0.8760179095039451 MAE:138038.27535453404
ridge scores - R2:0.11641934163096612 MAE:434251.93286359526
lasso scores - R2:0.11640743024507427 MAE:434255.9577096853
enet scores - R2:0.11794992112012459 MAE:433648.9325482862
```

Conclusion

IT seems that rf algorithm is best regression model that can be used fot this implementation

Save the model

In [188…
```
with open('wall_mart.pkl', 'wb') as f:
     pickle.dump(fit_models['rf'], f)
```

Make Detection

In [191…
```
with open('wall_mart.pkl', 'rb') as f:
    model = pickle.load(f)
```

In [192…
```
model
```

Out[192…
```
Pipeline(steps=[('randomforestregressor',
                 RandomForestRegressor(random_state=1234))])
```

Let is make the prediction for the following scenario

- Store : 40
- Holiday status : 1

- Temperature : 40 far
- Fuel price : 2.6
- CPI : 211
- Unemployemnet : 8.1

In [206…
```
x_p1 = pd.DataFrame([40,1,40,2.6,211,8.1]).transpose()
```

In [207…
```
x_p1
```

Out[207…

	0	1	2	3	4	5
0	40.0	1.0	40.0	2.6	211.0	8.1

In [208…
```
model.predict(x_p1)
```

Out[208…
```
array([1250011.488])
```