walmart-time-series-sales-forecasting

September 4, 2025

[1]:
```python
# Importing all necessary libraries to proceed with this project.

import warnings
import itertools
import numpy as np
import scipy.stats as stats
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import calendar
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Lasso, LinearRegression, LassoCV
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsRegressor
import random
import sqlite3
from itertools import cycle, islice
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
import xgboost as xgb
import catboost as cb
import lightgbm as lgb
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingRegressor
from sklearn.svm import SVR
# Import timedelta from datetime library
from datetime import timedelta


ss = StandardScaler()
```

```
<IPython.core.display.HTML object>
```

[39]:
```
pwd
```

[39]: '/kaggle/input/walmart-recruiting-store-sales-forecasting'

[40]:
```
cd ../input/walmart-recruiting-store-sales-forecasting
```

```
[Errno 2] No such file or directory: '../input/walmart-recruiting-store-sales-
forecasting'
/kaggle/input/walmart-recruiting-store-sales-forecasting
```

[41]:
```
pwd
```

[41]: '/kaggle/input/walmart-recruiting-store-sales-forecasting'

Load and read data

[42]:
```
walmart = pd.read_csv('train.csv.zip')

walmart_feature = pd.read_csv('features.csv.zip')

walmart_store = pd.read_csv('stores.csv')
```

[43]:
```
walmart.head()
```

[43]:

	Store	Dept	Date	Weekly_Sales	IsHoliday
0	1	1	2010-02-05	24924.50	False
1	1	1	2010-02-12	46039.49	True
2	1	1	2010-02-19	41595.55	False
3	1	1	2010-02-26	19403.54	False
4	1	1	2010-03-05	21827.90	False

[44]:
```
walmart.shape
```

[44]: (421570, 5)

[45]:
```
walmart_store_group=walmart.groupby(["Store","Date"])[["Weekly_Sales"]].sum()
walmart_store_group.reset_index(inplace=True)
```

Merging all the datasets into one place for easier test and analysis.

[46]:
```
result = pd.merge(walmart_store_group, walmart_store, how='inner', on='Store',␣
 ↪left_on=None, right_on=None,
        left_index=False, right_index=False, sort=False,
        suffixes=('_x', '_y'), copy=True, indicator=False)

data = pd.merge(result, walmart_feature, how='inner', on=['Store','Date'],␣
 ↪left_on=None, right_on=None,
        left_index=False, right_index=False, sort=False,
        suffixes=('_x', '_y'), copy=True, indicator=False)
```

[47]:
```python
print(data.shape)
```

```
(6435, 15)
```

Dataframe Walmart with 421570 rows has come down to 6435 rows by doing a group by and merge

[48]:
```python
data.head()
```

[48]:
```
   Store        Date  Weekly_Sales Type    Size  Temperature  Fuel_Price  \
0      1  2010-02-05    1643690.90    A  151315        42.31       2.572
1      1  2010-02-12    1641957.44    A  151315        38.51       2.548
2      1  2010-02-19    1611968.17    A  151315        39.93       2.514
3      1  2010-02-26    1409727.59    A  151315        46.63       2.561
4      1  2010-03-05    1554806.68    A  151315        46.50       2.625

   MarkDown1  MarkDown2  MarkDown3  MarkDown4  MarkDown5         CPI  \
0        NaN        NaN        NaN        NaN        NaN  211.096358
1        NaN        NaN        NaN        NaN        NaN  211.242170
2        NaN        NaN        NaN        NaN        NaN  211.289143
3        NaN        NaN        NaN        NaN        NaN  211.319643
4        NaN        NaN        NaN        NaN        NaN  211.350143

   Unemployment  IsHoliday
0         8.106      False
1         8.106       True
2         8.106      False
3         8.106      False
4         8.106      False
```

[49]:
```python
#let's encode the categorical column : IsHoliday

data['IsHoliday'] = data['IsHoliday'].apply(lambda x: 1 if x == True else 0)
```

[50]:
```python
# Want to check the date column is in object format or datetime
data.dtypes
```

[50]:
```
Store             int64
Date             object
Weekly_Sales    float64
Type             object
Size              int64
Temperature     float64
Fuel_Price      float64
MarkDown1       float64
MarkDown2       float64
MarkDown3       float64
MarkDown4       float64
```

```
MarkDown5        float64
CPI              float64
Unemployment     float64
IsHoliday          int64
dtype: object
```

[51]:
```
data["Date"]=pd.to_datetime(data.Date)

# Extracting details from date given. so that can be used for seasonal checks␣
↪or grouping

data["Day"]=data.Date.dt.day
data["Month"]=data.Date.dt.month
data["Year"]=data.Date.dt.year

# Changing the Months value from numbers to real values like Jan, Feb to Dec
data['Month'] = data['Month'].apply(lambda x: calendar.month_abbr[x])
```

[52]:
```
# Lets look into the null values
data.isnull().sum()
```

[52]:
```
Store              0
Date               0
Weekly_Sales       0
Type               0
Size               0
Temperature        0
Fuel_Price         0
MarkDown1       4155
MarkDown2       4798
MarkDown3       4389
MarkDown4       4470
MarkDown5       4140
CPI                0
Unemployment       0
IsHoliday          0
Day                0
Month              0
Year               0
dtype: int64
```

[ ]:
```
#will create this column for later use
#data['MarkdownsSum'] = data['MarkDown1'] + data['MarkDown2'] +␣
↪data['MarkDown3'] + data['MarkDown4'] + data['MarkDown5']
```

[26]:
```
data.fillna(0, inplace = True)
```

[53]:
```
data.describe().T
```

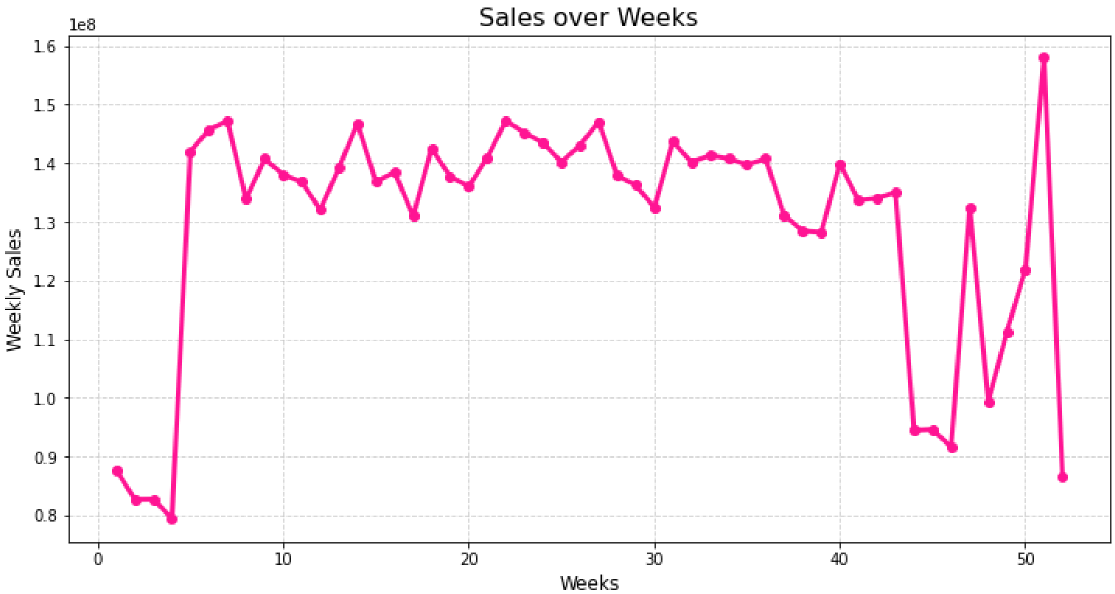
[53]:
```
              count          mean            std         min          25%  \
Store        6435.0  2.300000e+01      12.988182       1.000       12.000
Weekly_Sales 6435.0  1.046965e+06  564366.622054  209986.250   553350.105
Size         6435.0  1.302876e+05   63117.022465   34875.000    70713.000
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
MarkDown1    2280.0  6.855587e+03    8183.310015       0.270     1679.190
MarkDown2    1637.0  3.218966e+03    9268.082387    -265.760       37.200
MarkDown3    2046.0  1.349853e+03    9287.242800     -29.100        4.700
MarkDown4    1965.0  3.303858e+03    6211.203947       0.220      483.270
MarkDown5    2295.0  4.435262e+03    5868.933325     135.160     1702.565
CPI          6435.0  1.715784e+02      39.356712     126.064      131.735
Unemployment 6435.0  7.999151e+00       1.875885       3.879        6.891
IsHoliday    6435.0  6.993007e-02       0.255049       0.000        0.000
Day          6435.0  1.567832e+01       8.755780       1.000        8.000
Year         6435.0  2.010965e+03       0.797019    2010.000     2010.000

                        50%           75%           max
Store             23.000000  3.400000e+01  4.500000e+01
Weekly_Sales  960746.040000  1.420159e+06  3.818686e+06
Size          126512.000000  2.023070e+05  2.196220e+05
Temperature       62.670000  7.494000e+01  1.001400e+02
Fuel_Price         3.445000  3.735000e+00  4.468000e+00
MarkDown1       4972.590000  8.873583e+03  8.864676e+04
MarkDown2        187.040000  1.785290e+03  1.045195e+05
MarkDown3         22.700000  9.998750e+01  1.416306e+05
MarkDown4       1419.420000  3.496080e+03  6.747485e+04
MarkDown5       3186.520000  5.422080e+03  1.085193e+05
CPI              182.616521  2.127433e+02  2.272328e+02
Unemployment       7.874000  8.622000e+00  1.431300e+01
IsHoliday          0.000000  0.000000e+00  1.000000e+00
Day               16.000000  2.300000e+01  3.100000e+01
Year            2011.000000  2.012000e+03  2.012000e+03
```

[54]:
```
#add a 'week' column to the dataset for further analysis
data['Week'] = data.Date.dt.isocalendar().week
```

[55]:
```
data.describe().T.style.bar(subset=['mean'], color='#205ff2')\
                            .background_gradient(subset=['std'], cmap='Reds')\
                            .background_gradient(subset=['50%'],␣
 ↪cmap='coolwarm')
```

[55]: <pandas.io.formats.style.Styler at 0x7b2f34edaf10>

[56]:
```python
import matplotlib.pyplot as plt

df_weeks = data.groupby('Week')['Weekly_Sales'].sum()
plt.figure(figsize=(12,6))
plt.plot(df_weeks.index, df_weeks.values, color='deeppink', linewidth=3, marker='o')

plt.title("Sales over Weeks", fontsize=16)
plt.xlabel("Weeks", fontsize=12)
plt.ylabel("Weekly Sales", fontsize=12)
plt.grid(True, linestyle='--', alpha=0.6)

plt.show()
```

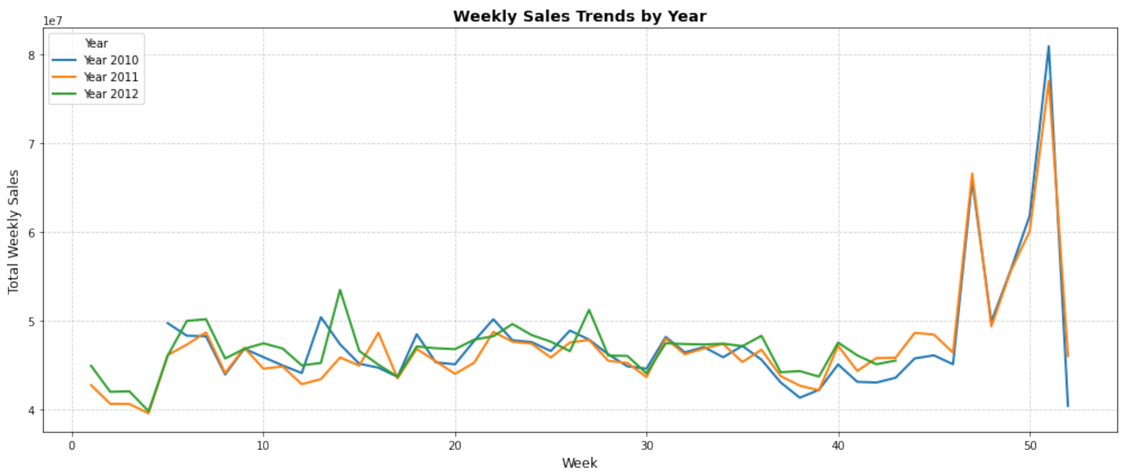


[57]:
```python
df_weeks.head()
```

[57]:
```
Week
1     8.773121e+07
2     8.269676e+07
3     8.273564e+07
4     7.943483e+07
5     1.419895e+08
Name: Weekly_Sales, dtype: float64
```

[35]:
```python
df_year_week = data.groupby(['Year','Week'])['Weekly_Sales'].sum().reset_index()

import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(14,6))

for year in df_year_week['Year'].unique():
    subset = df_year_week[df_year_week['Year'] == year]
    plt.plot(
        subset['Week'],
        subset['Weekly_Sales'],
        label=f"Year {year}",
        linewidth=2
    )

plt.title("Weekly Sales Trends by Year", fontsize=14, fontweight="bold")
plt.xlabel("Week", fontsize=12)
plt.ylabel("Total Weekly Sales", fontsize=12)
plt.legend(title="Year")
plt.grid(True, linestyle="--", alpha=0.6)
plt.tight_layout()
plt.show()
```

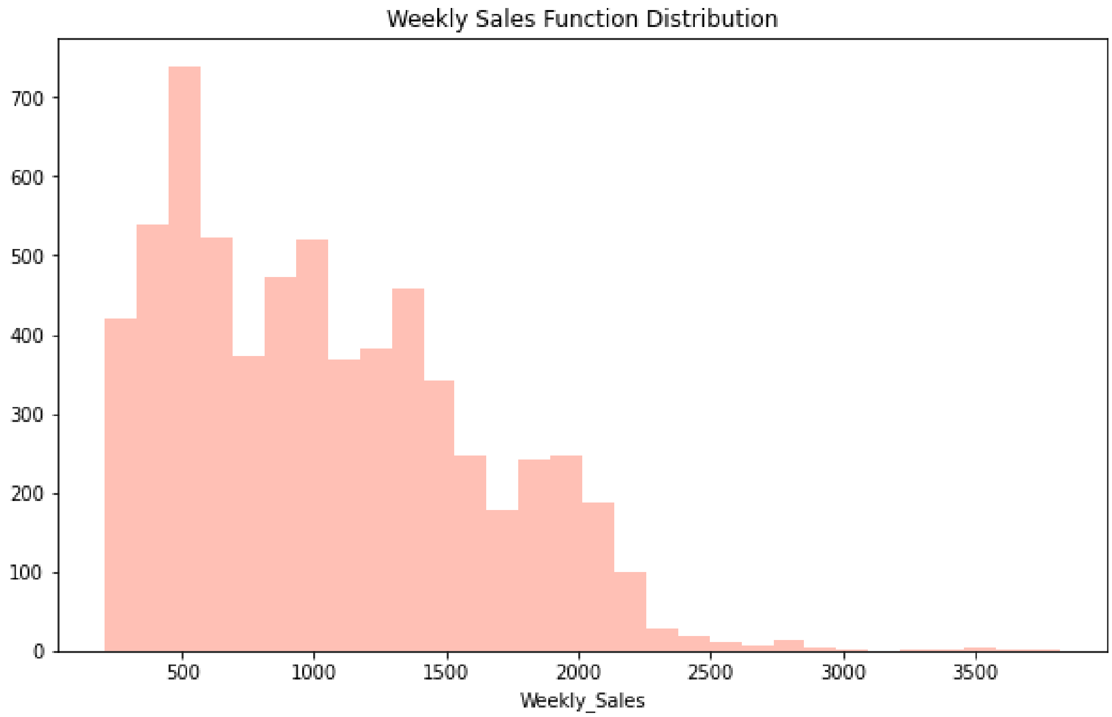


[60]:
```
## setting all missing values in markdown columns to -500 for now. We will␣
↪treat them later while performing Feature scaling
data['MarkDown1'].fillna(-500, inplace=True)
data['MarkDown2'].fillna(-500, inplace=True)
data['MarkDown3'].fillna(-500, inplace=True)
data['MarkDown4'].fillna(-500, inplace=True)
data['MarkDown5'].fillna(-500, inplace=True)
```

[61]:
```
data.isnull().sum()
```

[61]:
```
Store           0
Date            0
Weekly_Sales    0
Type            0
Size            0
Temperature     0
Fuel_Price      0
MarkDown1       0
MarkDown2       0
MarkDown3       0
MarkDown4       0
MarkDown5       0
CPI             0
Unemployment    0
IsHoliday       0
Day             0
Month           0
Year            0
Week            0
dtype: int64
```

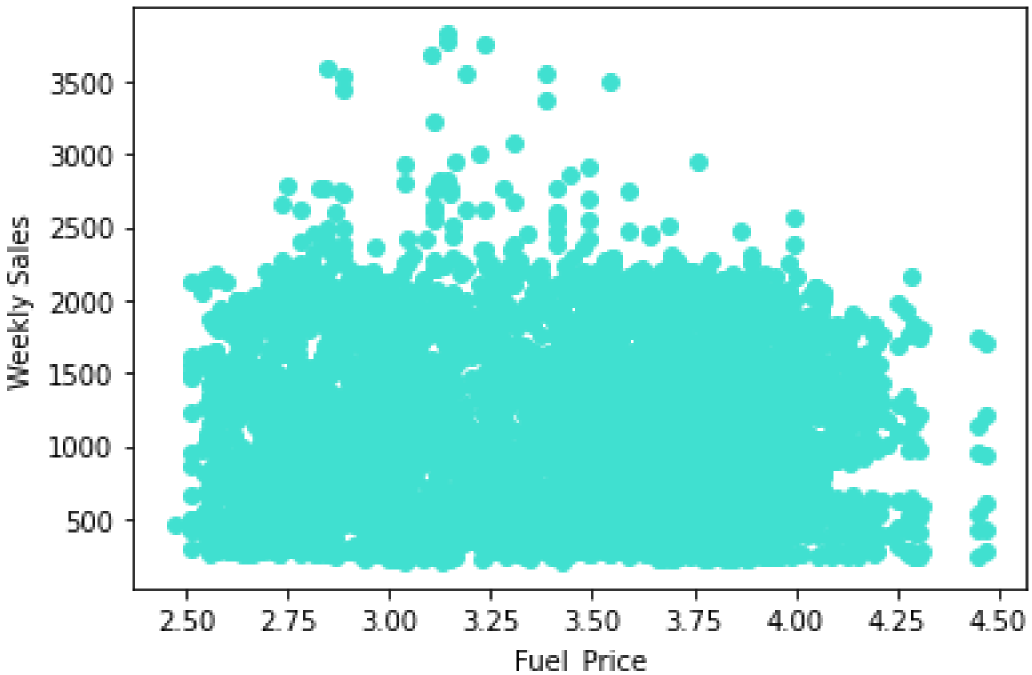
[62]:
```
# From the Describe function we see that weekly sales for each store are very␣
↪high.
# we will scale down the value for ease of use and revert back when we look␣
↪residuals or where necessary

plt.figure(figsize=(10, 6))
data["Weekly_Sales"]=data.Weekly_Sales/1000

sns.distplot(data.Weekly_Sales, kde=False, bins=30, color = 'tomato')
plt.title('Weekly Sales Function Distribution')
plt.show()
```



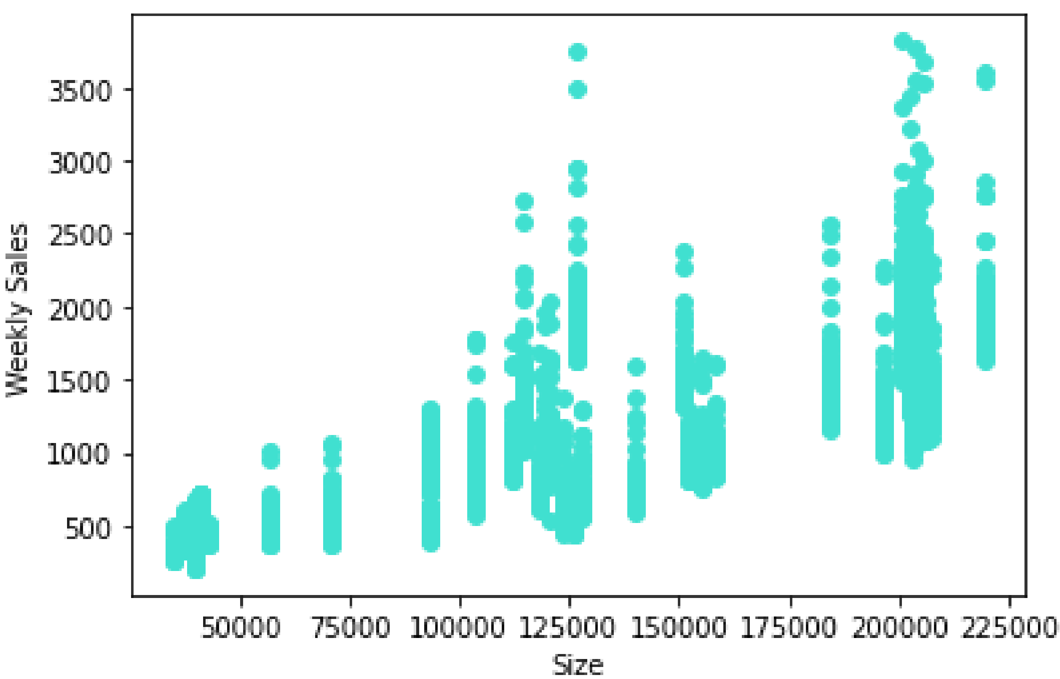
In the Distribution, natural Log of Sales and the square root of Sales look better distributed. We can use Natural Log for predictions later

[63]:
```
def scatter(dataset, column):
    plt.figure()
    plt.scatter(data[column] , data['Weekly_Sales'], color = 'turquoise')
    plt.ylabel('Weekly Sales')
    plt.xlabel(column)
```

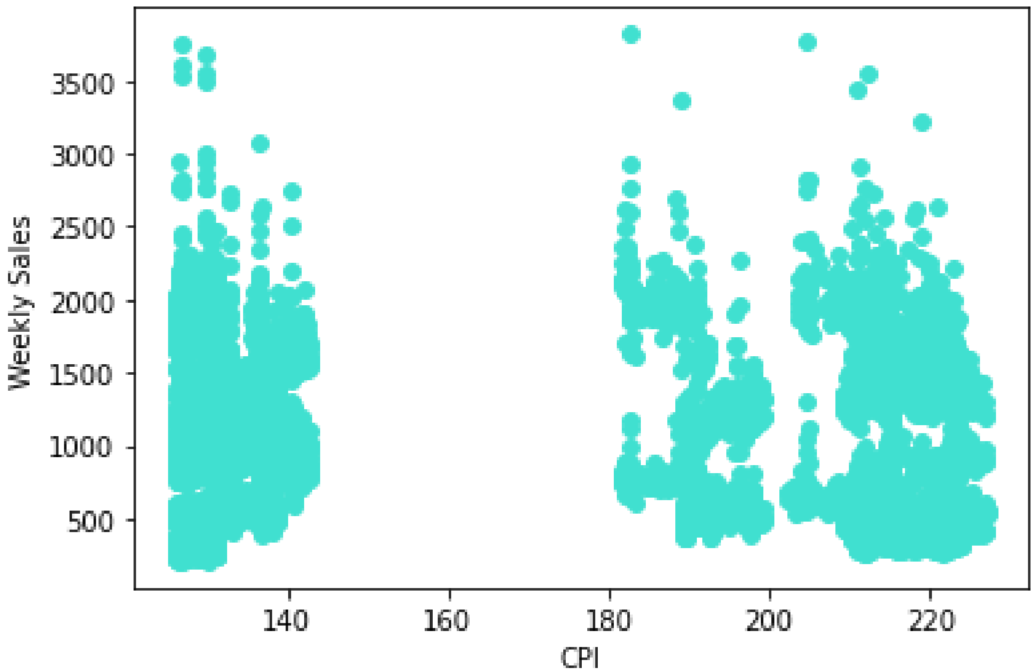
[64]:
```
scatter(data, 'Fuel_Price')
scatter(data, 'Size')
scatter(data, 'CPI')
scatter(data, 'Type')
scatter(data, 'IsHoliday')
scatter(data, 'Unemployment')
scatter(data, 'Temperature')
scatter(data, 'Store')
```

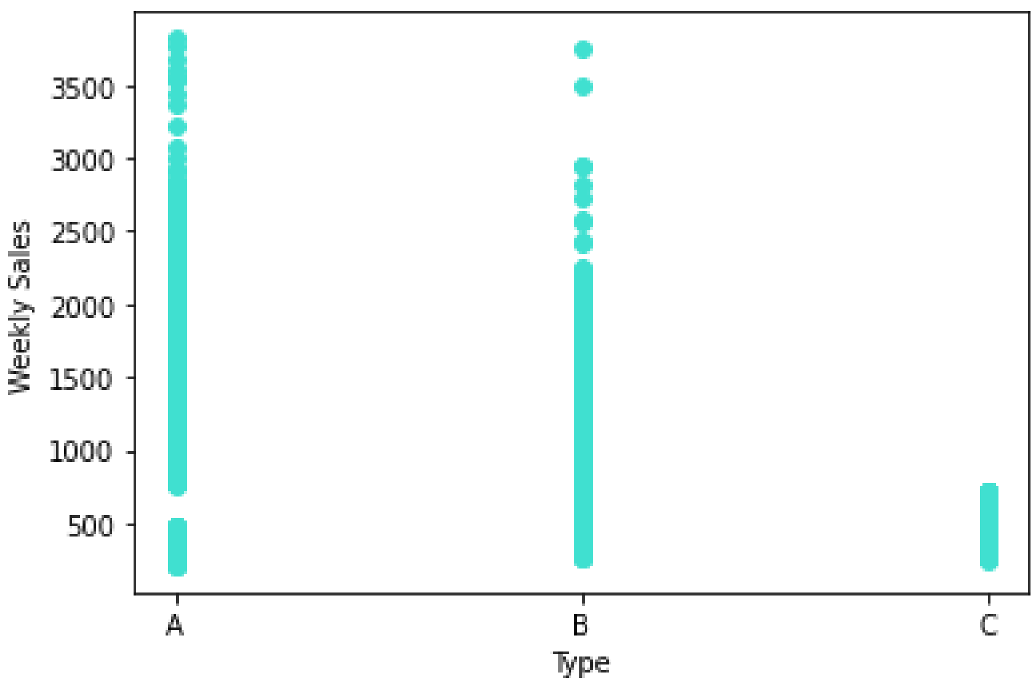


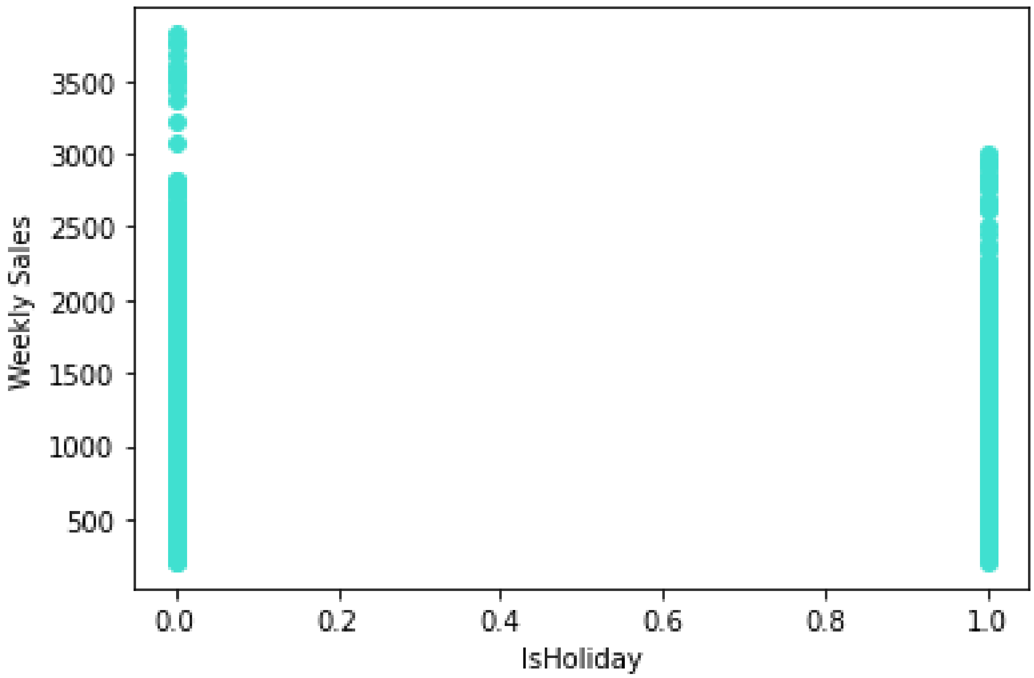
Weekly Sales
Fuel_Price



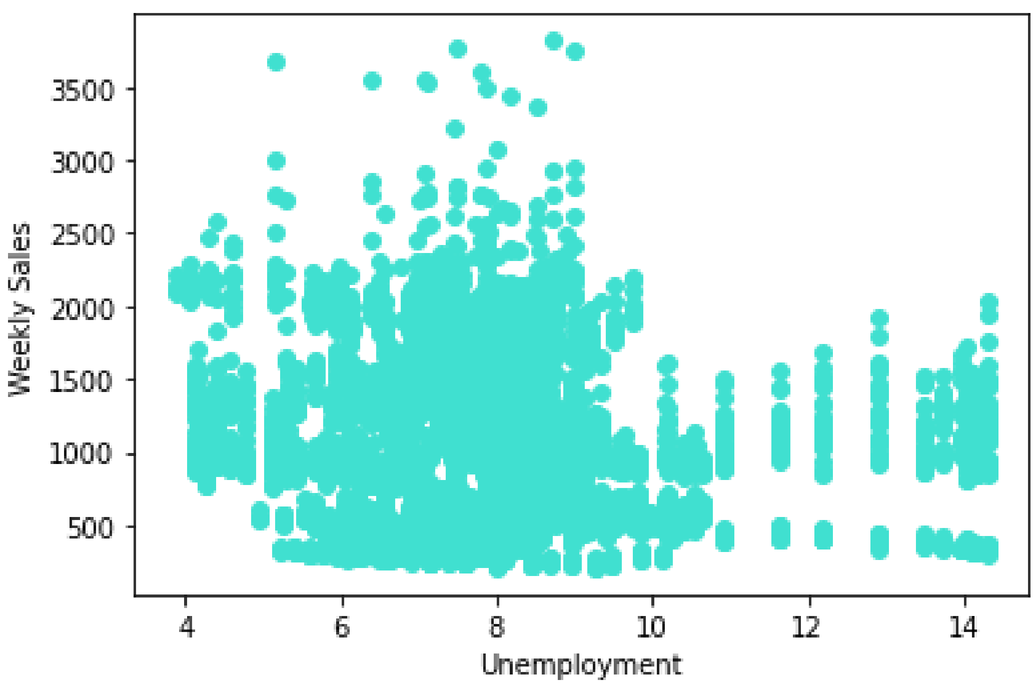
Weekly Sales
Size


Weekly Sales
3500
3000
2500
2000
1500
1000
500
140
160
180
200
220
CPI

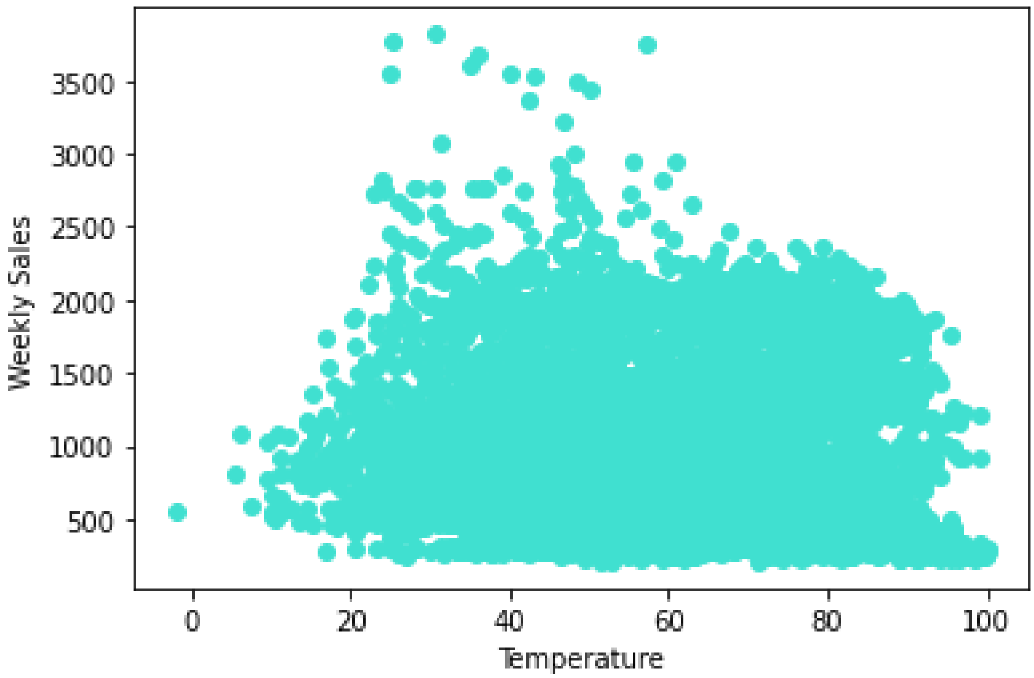

Weekly Sales
3500
3000
2500
2000
1500
1000
500
A
B
C
Type

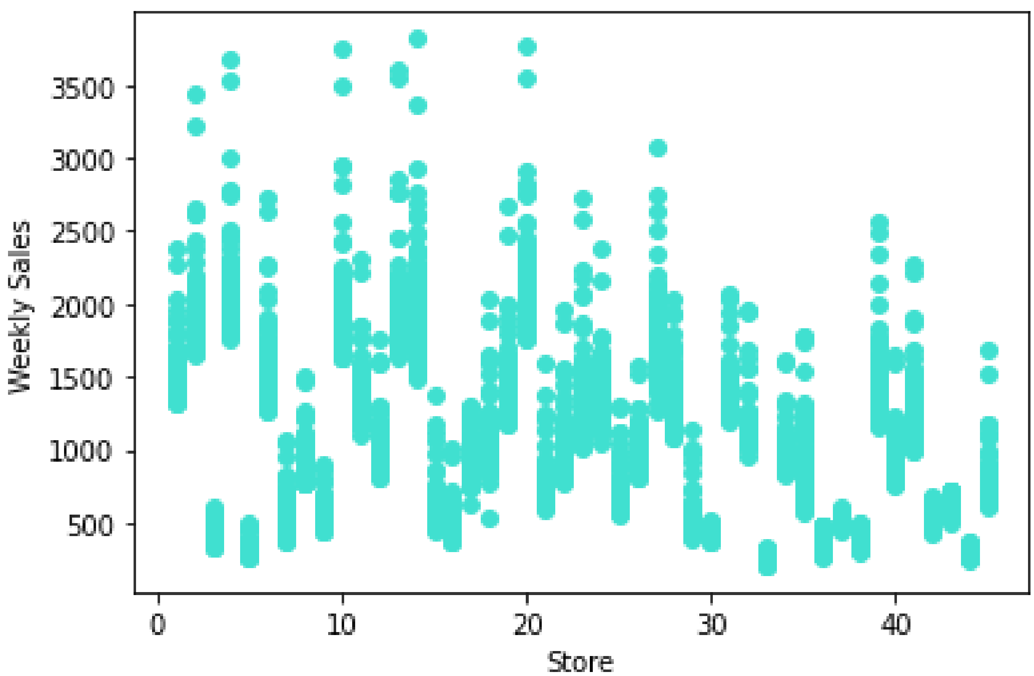


Weekly Sales
3500
3000
2500
2000
1500
1000
500
0.0
0.2
0.4
0.6
0.8
1.0
IsHoliday



Weekly Sales
3500
3000
2500
2000
1500
1000
500
4
6
8
10
12
14
Unemployment


3500
3000
2500
2000
1500
1000
500
Weekly Sales
0
20
40
60
80
100
Temperature


3500
3000
2500
2000
1500
1000
500
Weekly Sales
0
10
20
30
40
Store

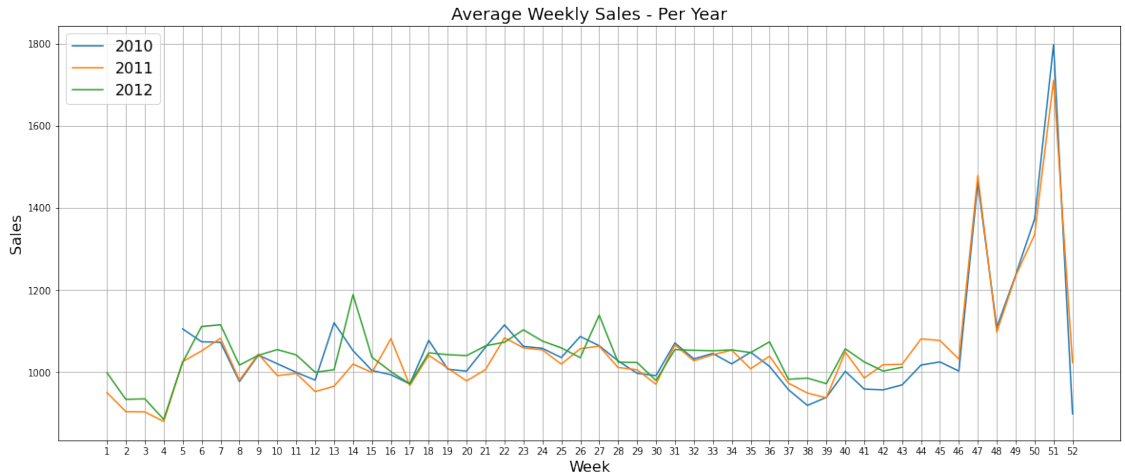
[65]:
```
data.head()
```

[65]:   Store   Date Weekly_Sales Type   Size Temperature Fuel_Price \
0   1 2010-02-05   1643.69090   A 151315   42.31   2.572
1   1 2010-02-12   1641.95744   A 151315   38.51   2.548
2   1 2010-02-19   1611.96817   A 151315   39.93   2.514
3   1 2010-02-26   1409.72759   A 151315   46.63   2.561
4   1 2010-03-05   1554.80668   A 151315   46.50   2.625

   MarkDown1 MarkDown2 MarkDown3 MarkDown4 MarkDown5   CPI \
0   -500.0   -500.0   -500.0   -500.0   -500.0 211.096358
1   -500.0   -500.0   -500.0   -500.0   -500.0 211.242170
2   -500.0   -500.0   -500.0   -500.0   -500.0 211.289143
3   -500.0   -500.0   -500.0   -500.0   -500.0 211.319643
4   -500.0   -500.0   -500.0   -500.0   -500.0 211.350143

   Unemployment IsHoliday Day Month Year Week
0   8.106   0   5   Feb 2010   5
1   8.106   1   12   Feb 2010   6
2   8.106   0   19   Feb 2010   7
3   8.106   0   26   Feb 2010   8
4   8.106   0   5   Mar 2010   9

[66]:
```
weekly_sales_2010 = data[data.Year==2010]['Weekly_Sales'].groupby(data['Week']).
 ↪mean()
weekly_sales_2011 = data[data.Year==2011]['Weekly_Sales'].groupby(data['Week']).
 ↪mean()
weekly_sales_2012 = data[data.Year==2012]['Weekly_Sales'].groupby(data['Week']).
 ↪mean()
plt.figure(figsize=(20,8))
sns.lineplot(weekly_sales_2010.index, weekly_sales_2010.values)
sns.lineplot(weekly_sales_2011.index, weekly_sales_2011.values)
sns.lineplot(weekly_sales_2012.index, weekly_sales_2012.values)
plt.grid()
plt.xticks(np.arange(1, 53, step=1))
plt.legend(['2010', '2011', '2012'], loc='best', fontsize=16)
plt.title('Average Weekly Sales - Per Year', fontsize=18)
plt.ylabel('Sales', fontsize=16)
plt.xlabel('Week', fontsize=16)
plt.show()
```



Note : As we can see, there is one important Holiday not included in 'IsHoliday'. It's the Easter Day. It is always in a Sunday, but can fall on different weeks.

In 2010 is in Week 13

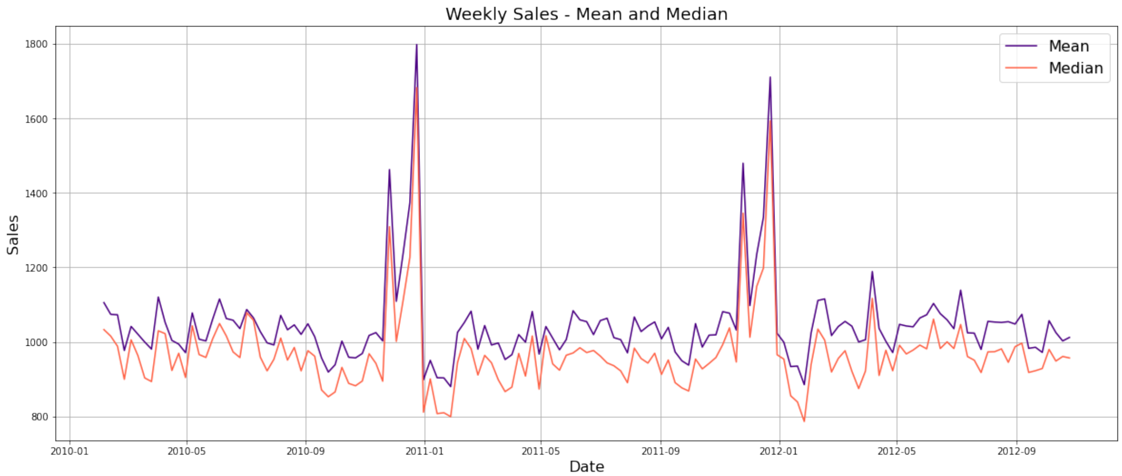
In 2011, Week 16

Week 14 in 2012

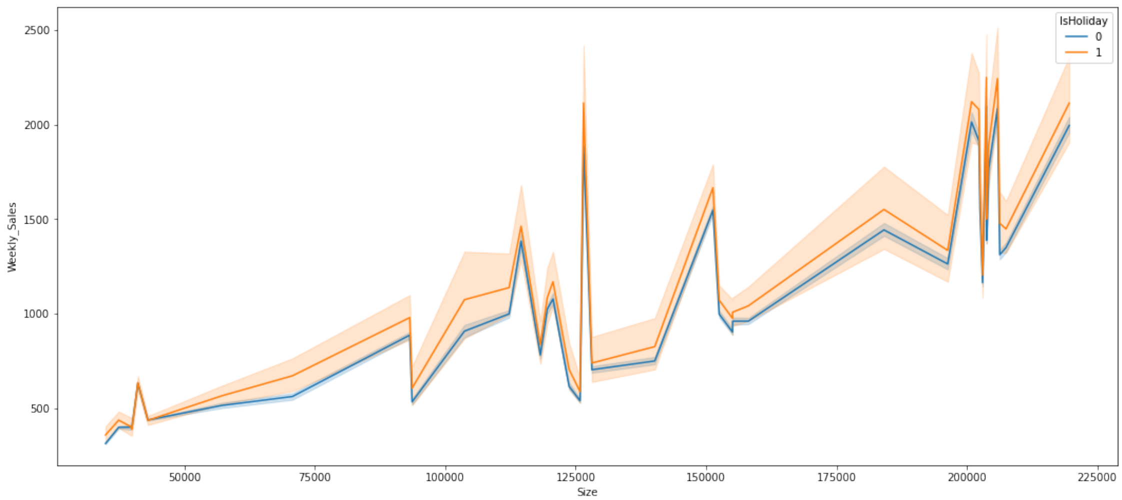
Week 13 in 2013 for **Test set**

So, we can change to 'True' these Weeks in each Year.

[67]:
```
weekly_sales_mean = data['Weekly_Sales'].groupby(data['Date']).mean()
weekly_sales_median = data['Weekly_Sales'].groupby(data['Date']).median()
plt.figure(figsize=(20,8))
sns.lineplot(weekly_sales_mean.index, weekly_sales_mean.values, color = 'indigo')
sns.lineplot(weekly_sales_median.index, weekly_sales_median.values, color = 'tomato')
plt.grid()
plt.legend(['Mean', 'Median'], loc='best', fontsize=16)
plt.title('Weekly Sales - Mean and Median', fontsize=18)
plt.ylabel('Sales', fontsize=16)
plt.xlabel('Date', fontsize=16)
plt.show()
```

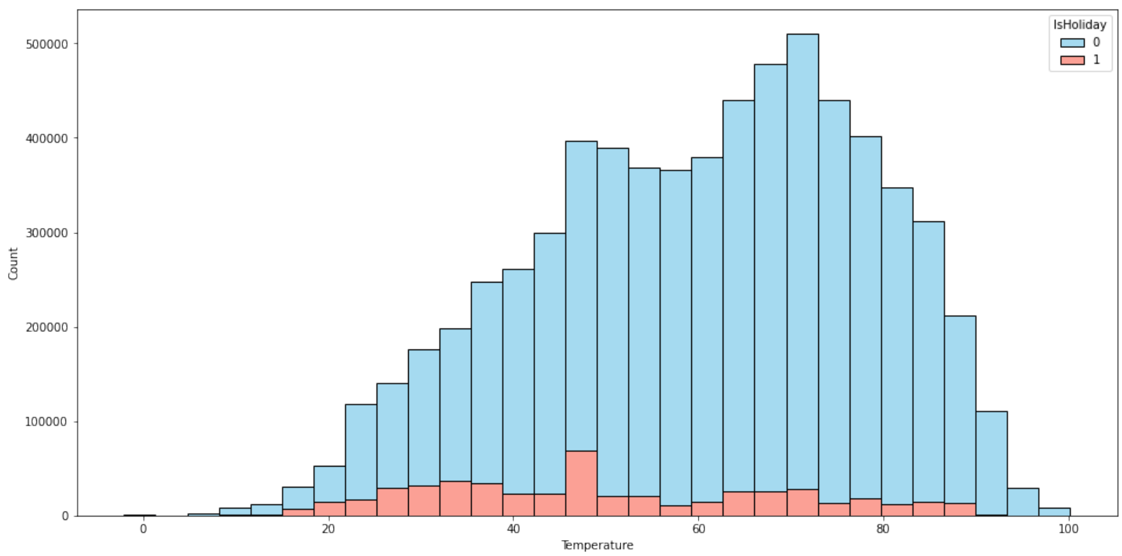


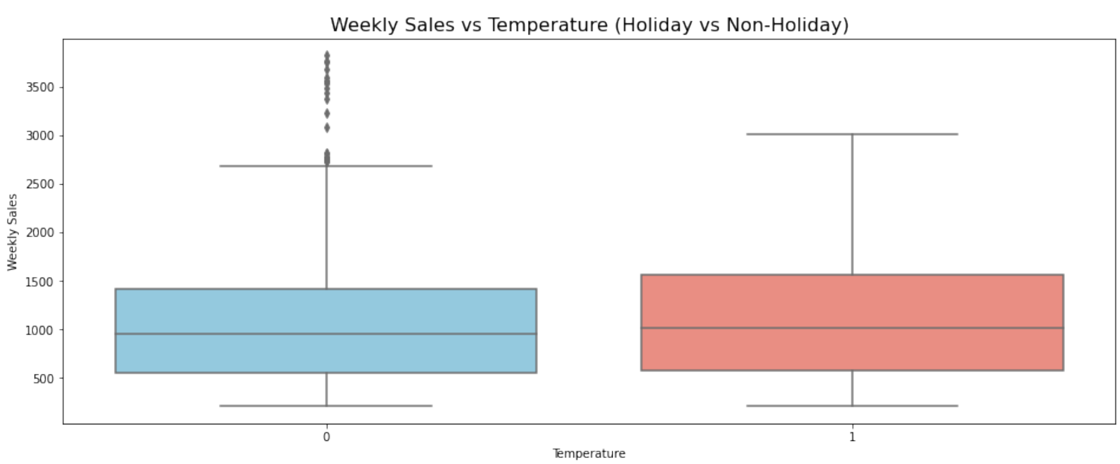
[68]:
```
plt.figure(figsize=(18,8))
sns.lineplot ( data = data, x = 'Size', y =  'Weekly_Sales', hue = 'IsHoliday');
```



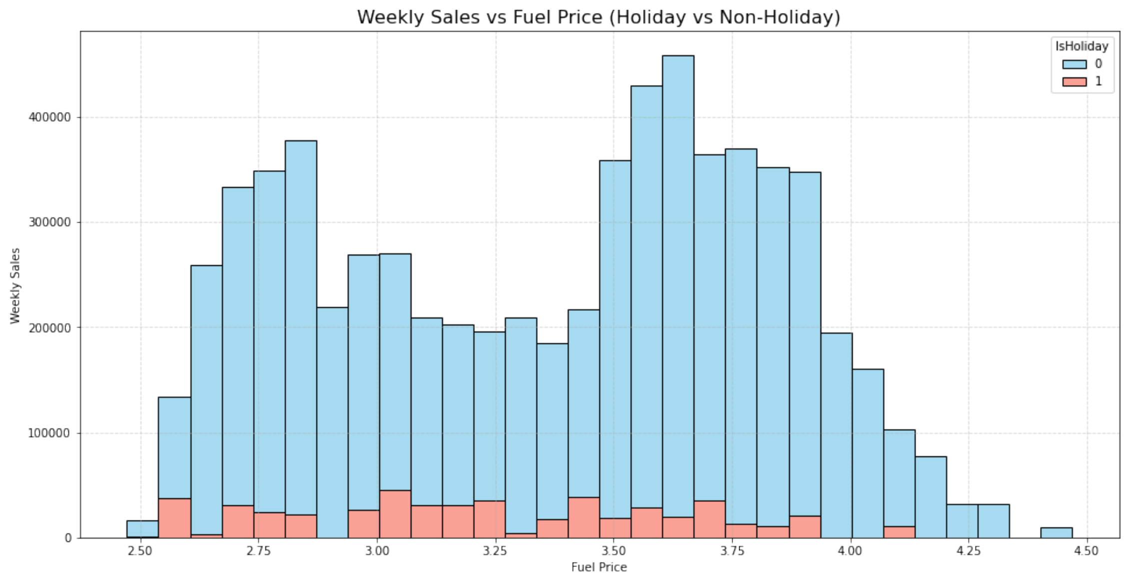
[71]:
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(16,8))
sns.histplot(
    data=data,
    x='Temperature',
    weights='Weekly_Sales',
    hue='IsHoliday',
    multiple='stack',
```

```
    bins=30,
    palette=['skyblue', 'salmon'])
plt.figure(figsize=(16,6))
sns.boxplot(
    data=data,
    x='IsHoliday',
    y='Weekly_Sales',
    palette=['skyblue', 'salmon'])

plt.title('Weekly Sales vs Temperature (Holiday vs Non-Holiday)', fontsize=16)
plt.xlabel('Temperature')
plt.ylabel('Weekly Sales')
plt.show()
```

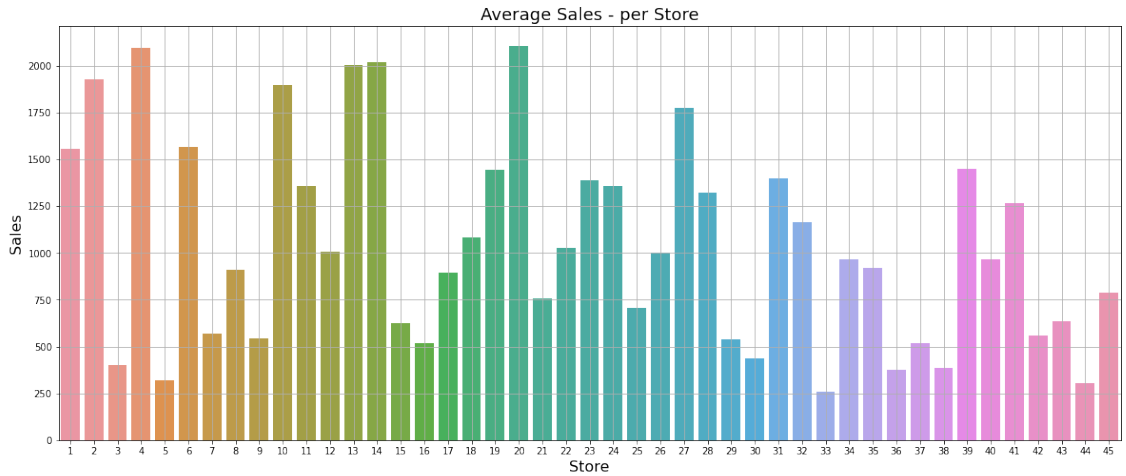




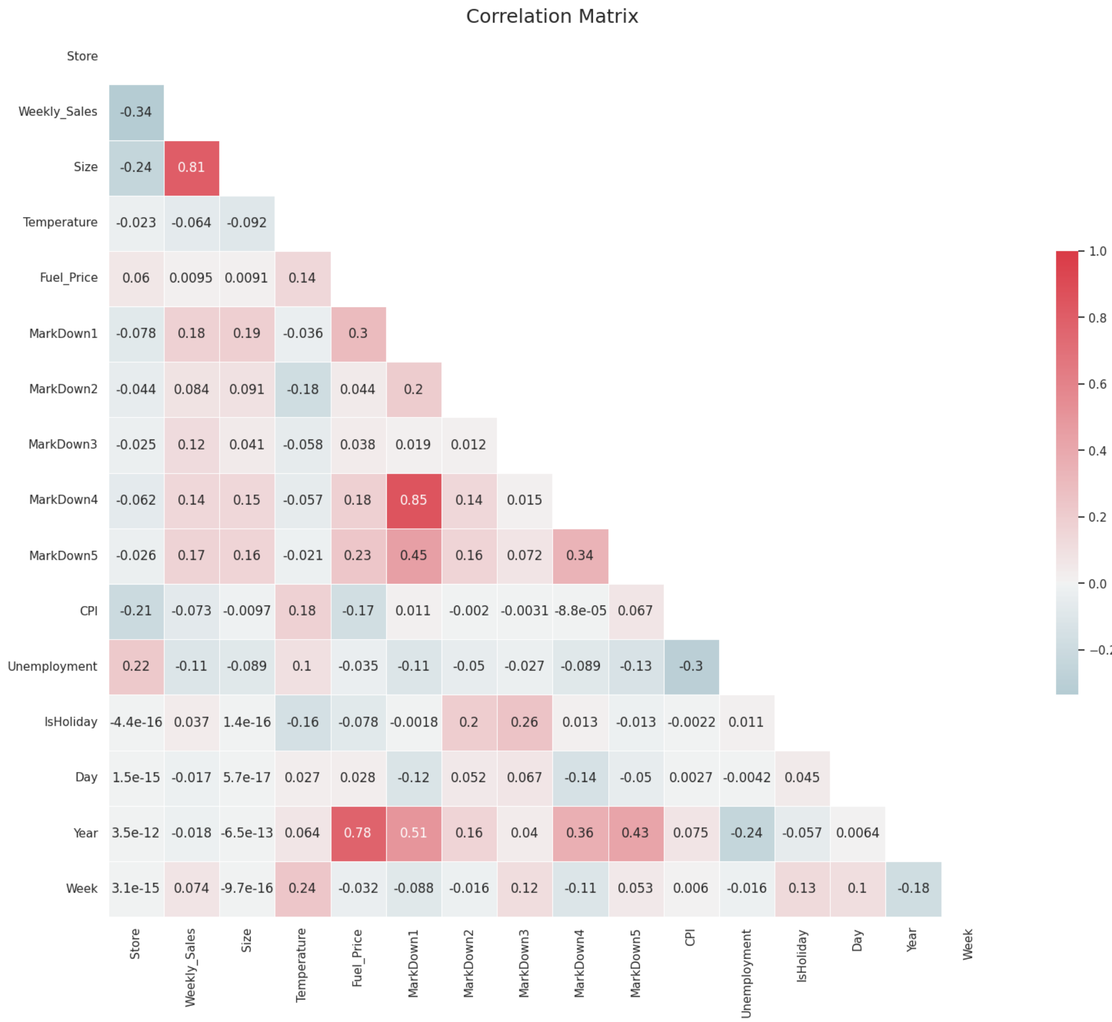
[74]:
```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(16,8))
sns.histplot(
    data=data,
    x='Fuel_Price',
    weights='Weekly_Sales',
    hue='IsHoliday',
    multiple='stack',
    bins=30,
    palette=['skyblue', 'salmon']
)
plt.title('Weekly Sales vs Fuel Price (Holiday vs Non-Holiday)', fontsize=16)
plt.xlabel('Fuel Price')
plt.ylabel('Weekly Sales')
plt.grid(True, linestyle='--', alpha=0.5)
plt.show()
```



[75]:
```
weekly_sales = data['Weekly_Sales'].groupby(data['Store']).mean()
plt.figure(figsize=(20,8))
plt.style.use('default')
sns.barplot(weekly_sales.index, weekly_sales.values)
plt.grid()
plt.title('Average Sales - per Store', fontsize=18)
plt.ylabel('Sales', fontsize=16)
plt.xlabel('Store', fontsize=16)
plt.show()
```



[76]:
```
sns.set(style="white")

corr = data.corr()
mask = np.triu(np.ones_like(corr, dtype=np.bool))
f, ax = plt.subplots(figsize=(20, 15))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
plt.title('Correlation Matrix', fontsize=18)
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True)

plt.show()
```

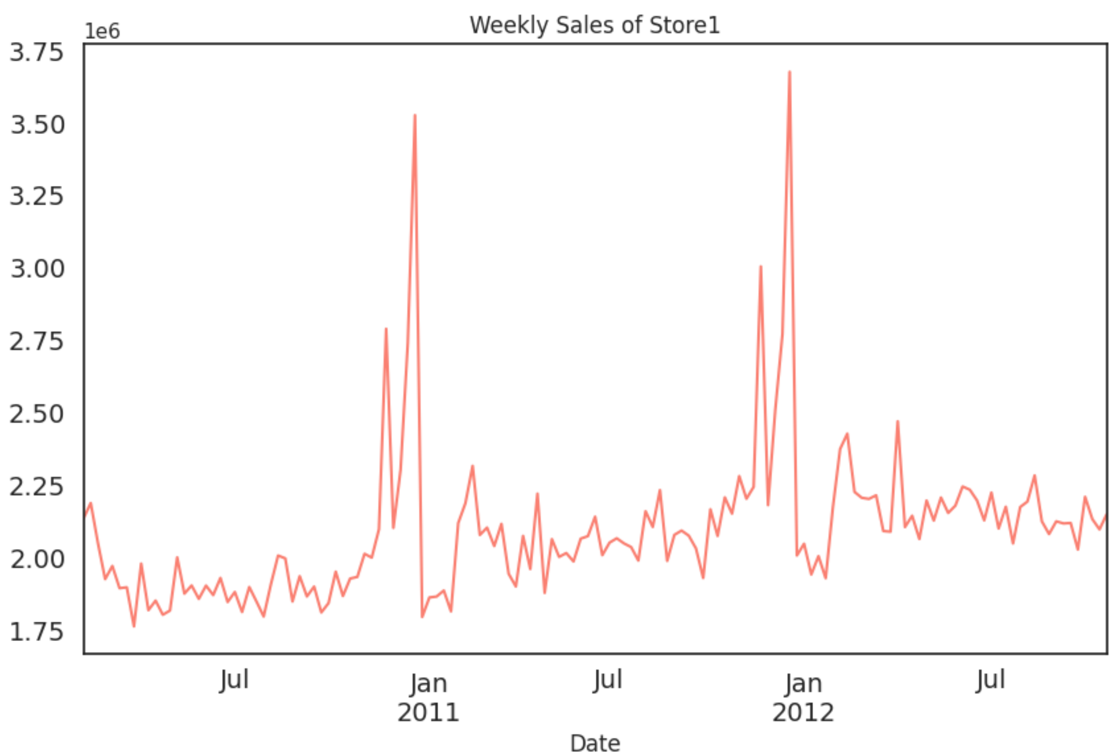


[77]:
```python
data1 = pd.read_csv('train.csv.zip')
data1.set_index('Date', inplace=True)

store4 = data1[data1.Store == 4]
# there are about 45 different stores in this dataset.

sales4 = pd.DataFrame(store4.Weekly_Sales.groupby(store4.index).sum())
sales4.dtypes
sales4.head(20)
# Grouped weekly sales by store 4

#remove date from index to change its dtype because it clearly isnt acceptable.
sales4.reset_index(inplace = True)

#converting 'date' column to a datetime type
```

```
sales4['Date'] = pd.to_datetime(sales4['Date'])
# resetting date back to the index
sales4.set_index('Date',inplace = True)
```

[78]:
```
# Lets take store 6 data for analysis
store6 = data1[data1.Store == 6]
# there are about 45 different stores in this dataset.

sales6 = pd.DataFrame(store6.Weekly_Sales.groupby(store6.index).sum())
sales6.dtypes
# Grouped weekly sales by store 6

#remove date from index to change its dtype because it clearly isnt acceptable.
sales6.reset_index(inplace = True)

#converting 'date' column to a datetime type
sales6['Date'] = pd.to_datetime(sales6['Date'])
# resetting date back to the index
sales6.set_index('Date',inplace = True)
```

[79]:
```
sales4.Weekly_Sales.plot(figsize=(10,6), title= 'Weekly Sales of Store1',␣
 ↪fontsize=14, color = 'salmon')
plt.show()
```

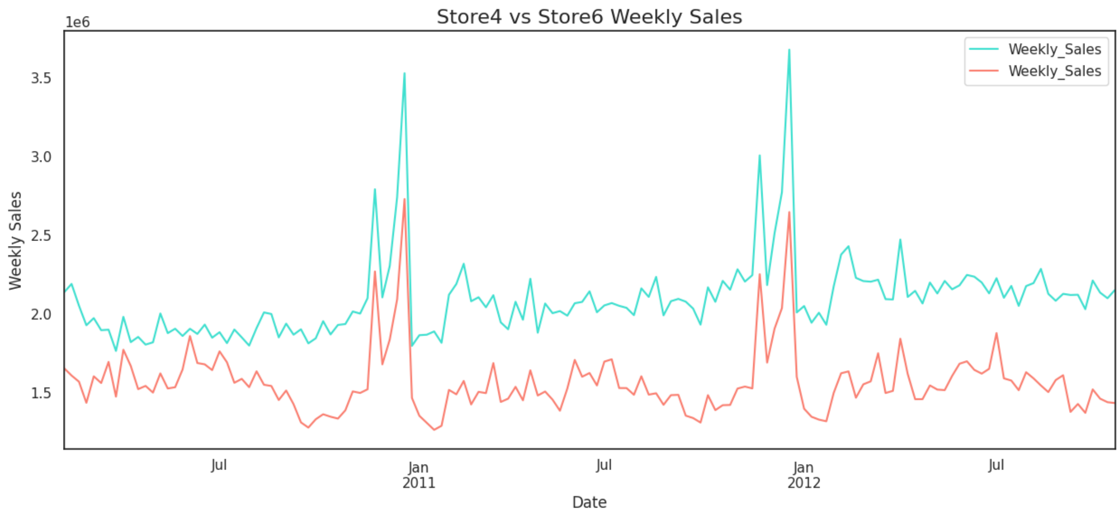


[80]:
```
from statsmodels.tsa.seasonal import seasonal_decompose

decomposition = seasonal_decompose(sales4.Weekly_Sales, period=12)
fig = plt.figure()
fig = decomposition.plot()
fig.set_size_inches(12, 10)
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```

[82]:
```
y1=sales4.Weekly_Sales
y2=sales6.Weekly_Sales
```

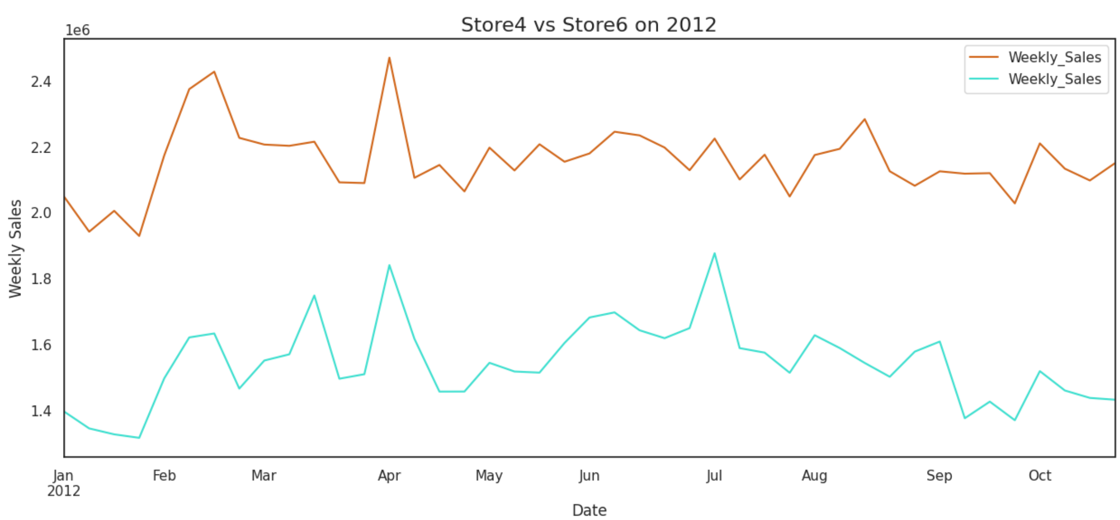
[83]:
```
y1.plot(figsize=(15, 6), legend=True, color = 'turquoise')
y2.plot(figsize=(15, 6), legend=True, color = 'salmon')
plt.ylabel('Weekly Sales')
plt.title('Store4 vs Store6 Weekly Sales', fontsize = '16')
plt.show()
```



This shows an interesting trend during year ends (during both 2011 & 2012). The best thing is both the stores have almost the same trends and spike just the magnitude is different.

`This clearly tells its a timeseries problem and it will be interesting to look more into it`

[84]:
```
# Lets Look into 2012 data for a better view
y1['2012'].plot(figsize=(15, 6),legend=True, color = 'chocolate')
y2['2012'].plot(figsize=(15, 6), legend=True, color = 'turquoise')
plt.ylabel('Weekly Sales')
plt.title('Store4 vs Store6 on 2012', fontsize = '16')
plt.show()
```



[85]:
```python
# Define the p, d and q parameters to take any value between 0 and 2
p = d = q = range(0, 5)

# Generate all different combinations of p, d and q triplets
pdq = list(itertools.product(p, d, q))

# Generate all different combinations of seasonal p, d and q triplets
seasonal_pdq = [(x[0], x[1], x[2], 52) for x in list(itertools.product(p, d,␣
↪q))]
```

[86]:
```python
import statsmodels.api as sm

mod = sm.tsa.statespace.SARIMAX(y1,
                                order=(4, 4, 3),
                                seasonal_order=(1, 1, 0, 52),   ␣
↪#enforce_stationarity=False,
                                enforce_invertibility=False)

results = mod.fit()

print(results.summary().tables[1])
```

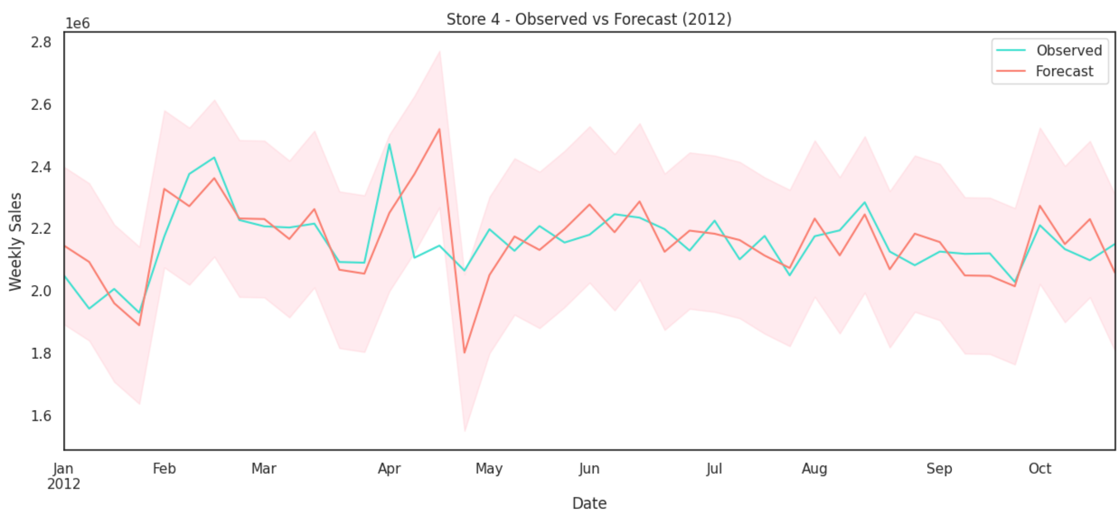
```
/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:471:
ValueWarning:

No frequency information was provided, so inferred frequency W-FRI will be used.

/opt/conda/lib/python3.7/site-packages/statsmodels/tsa/base/tsa_model.py:471:
ValueWarning:

No frequency information was provided, so inferred frequency W-FRI will be used.

 This problem is unconstrained.

RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            9     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  8.32190D+00    |proj g|=  2.11788D-01

At iterate    5    f=  8.19196D+00    |proj g|=  3.64631D-01

At iterate   10    f=  8.03865D+00    |proj g|=  1.09208D+00
```

```
At iterate   15    f=  7.98422D+00    |proj g|=  3.19756D-01

At iterate   20    f=  7.98333D+00    |proj g|=  9.54934D-02

At iterate   25    f=  7.98300D+00    |proj g|=  5.18928D-02

At iterate   30    f=  7.98251D+00    |proj g|=  2.84029D-01

At iterate   35    f=  7.98218D+00    |proj g|=  4.14258D-02

At iterate   40    f=  7.98217D+00    |proj g|=  1.05929D-02

At iterate   45    f=  7.98215D+00    |proj g|=  2.53882D-02

At iterate   50    f=  7.98200D+00    |proj g|=  2.65812D-02

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    9     50     64      1     0     0   2.658D-02   7.982D+00
  F =   7.9820012973704415

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT

/opt/conda/lib/python3.7/site-packages/statsmodels/base/model.py:606:
ConvergenceWarning:

Maximum Likelihood optimization failed to converge. Check mle_retvals

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -1.7384      0.544     -3.197      0.001      -2.804      -0.673
ar.L2         -1.2705      0.586     -2.168      0.030      -2.419      -0.122
ar.L3         -0.5848      0.251     -2.326      0.020      -1.078      -0.092
ar.L4         -0.1873      0.093     -2.011      0.044      -0.370      -0.005
ma.L1         -1.3921      0.493     -2.824      0.005      -2.358      -0.426
```

```
ma.L2         -0.1939      1.062     -0.183      0.855     -2.275      1.887
ma.L3          0.5902      0.593      0.995      0.320     -0.572      1.753
ar.S.L52      -0.0670      0.048     -1.388      0.165     -0.162      0.028
sigma2       1.622e+10   5.92e-11   2.74e+20      0.000   1.62e+10   1.62e+10
==============================================================================
```

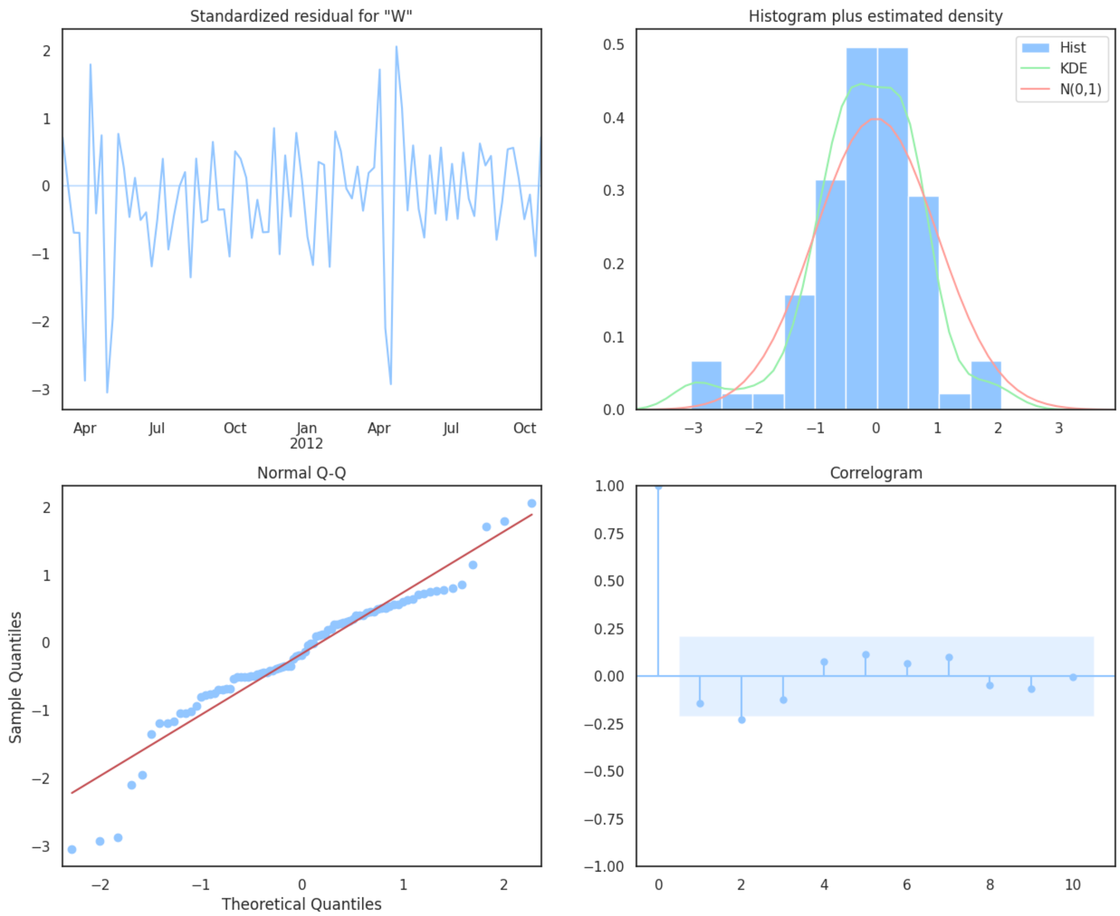
[91]:

```
start_idx = y1['2012'].index[0]
end_idx = y1['2012'].index[-1]
pred = results.get_prediction(start=start_idx, end=end_idx, dynamic=False)
pred_ci = pred.conf_int()

plt.figure(figsize=(15,6))
y1['2012'].plot(label='Observed', color='turquoise')
pred.predicted_mean.plot(label='Forecast', color='salmon')

plt.fill_between(pred_ci.index,
                 pred_ci.iloc[:, 0],
                 pred_ci.iloc[:, 1], color='pink', alpha=0.3)

plt.xlabel('Date')
plt.ylabel('Weekly Sales')
plt.title('Store 4 - Observed vs Forecast (2012)')
plt.legend()
plt.show()
```



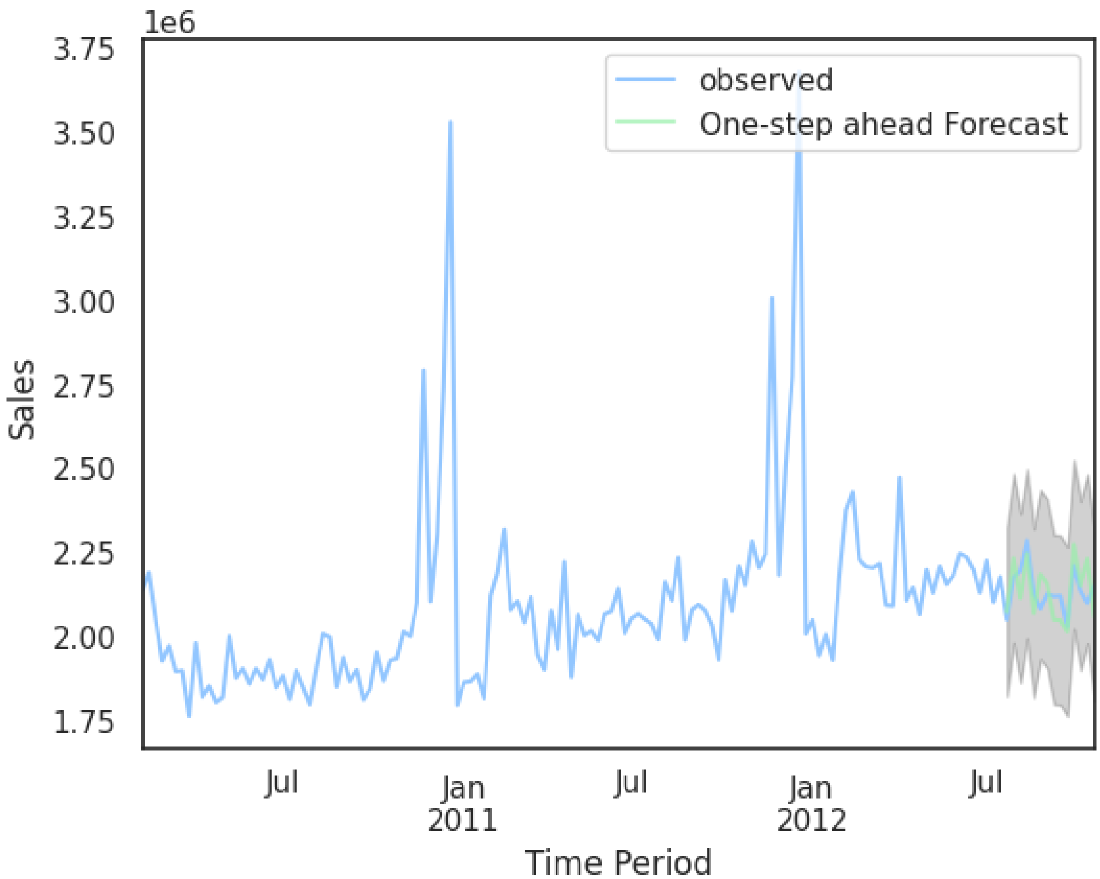
[87]:

```
plt.style.use('seaborn-pastel')
results.plot_diagnostics(figsize=(15, 12))
plt.show()
```




```
[88]: # Will predict for last 90 days. So setting the date according to that
      pred = results.get_prediction(start=pd.to_datetime('2012-07-27'), dynamic=False)
      pred_ci = pred.conf_int()
```

```
[89]: ax = y1['2010':].plot(label='observed')
      pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=.7)

      ax.fill_between(pred_ci.index,
                      pred_ci.iloc[:, 0],
                      pred_ci.iloc[:, 1], color='k', alpha=.2)

      ax.set_xlabel('Time Period')
      ax.set_ylabel('Sales')
      plt.legend()

      plt.show()
```



[92]:
```
y_forecasted = pred.predicted_mean
y_truth = y1['2012-7-27':]

# Compute the mean square error
mse = ((y_forecasted - y_truth) ** 2).mean()
print('The Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))
```

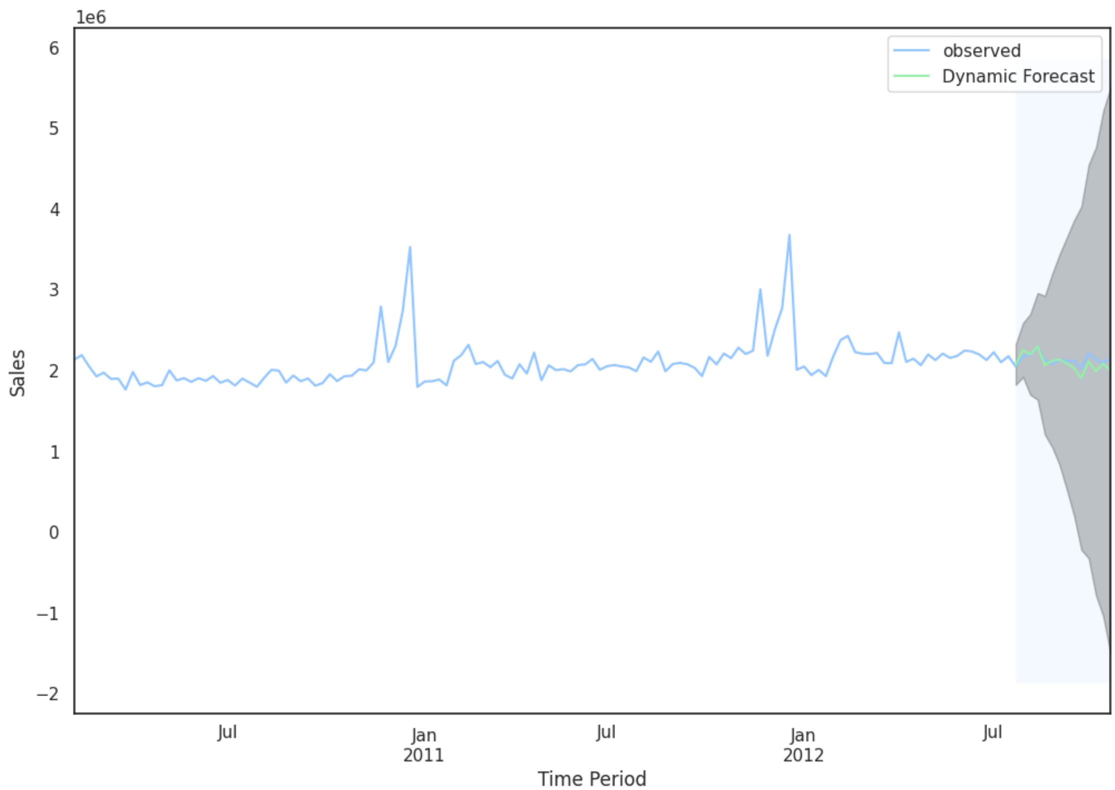
```
The Mean Squared Error of our forecasts is 4759937674.55
```

[93]:
```
import numpy as np

rmse = np.sqrt(mse)
print('Root Mean Squared Error = {}'.format(round(rmse, 2)))
```

```
Root Mean Squared Error = 68992.3
```

[94]:
```
pred_dynamic = results.get_prediction(start=pd.to_datetime('2012-7-27'),␣
 ↪dynamic=True, full_results=True)
pred_dynamic_ci = pred_dynamic.conf_int()
```

[95]:
```
ax = y1['2010':].plot(label='observed', figsize=(12, 8))
pred_dynamic.predicted_mean.plot(label='Dynamic Forecast', ax=ax)

ax.fill_between(pred_dynamic_ci.index,
                pred_dynamic_ci.iloc[:, 0],
                pred_dynamic_ci.iloc[:, 1], color='k', alpha=.25)

ax.fill_betweenx(ax.get_ylim(), pd.to_datetime('2012-7-26'), y1.index[-1],
                 alpha=.1, zorder=-1)

ax.set_xlabel('Time Period')
ax.set_ylabel('Sales')

plt.legend()
plt.show()
```



That looks good. Both the observed and predicted lines go together indicating nearly accurate prediction

[96]:
```
# Extract the predicted and true values of our time series
y_forecasted = pred_dynamic.predicted_mean
```

```
y_truth = y1['2012-7-27':]

# Compute the Root mean square error
rmse = np.sqrt(((y_forecasted - y_truth) ** 2).mean())
print('The Root Mean Squared Error of our forecasts is {}'.format(round(rmse,
 2)))
```

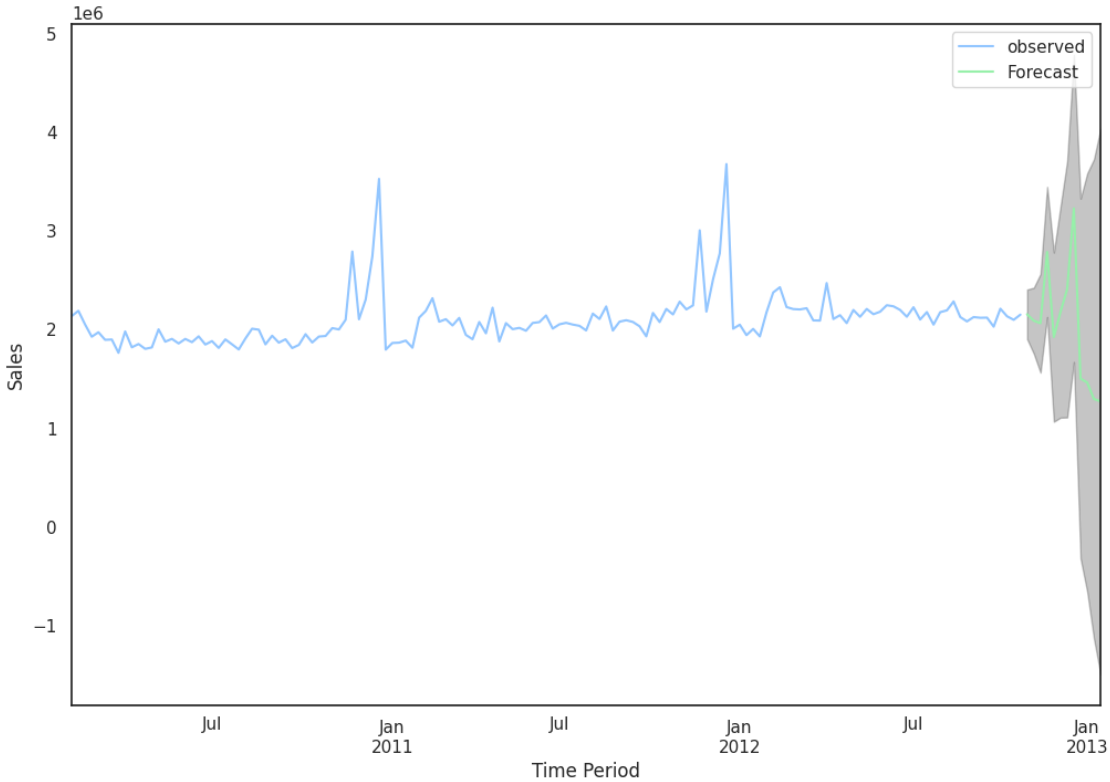
```
The Root Mean Squared Error of our forecasts is 80536.5
```

[97]:
```
Residual= y_forecasted - y_truth
print("Residual for Store1",np.abs(Residual).sum())
```

```
Residual for Store1 886130.9000129956
```

[98]:
```
# Get forecast 12 weeks ahead in future
pred_uc = results.get_forecast(steps=12)

# Get confidence intervals of forecasts
pred_ci = pred_uc.conf_int()
```

[99]:
```
ax = y1.plot(label='observed', figsize=(12, 8))
pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.25)
ax.set_xlabel('Time Period')
ax.set_ylabel('Sales')

plt.legend()
plt.show()
```



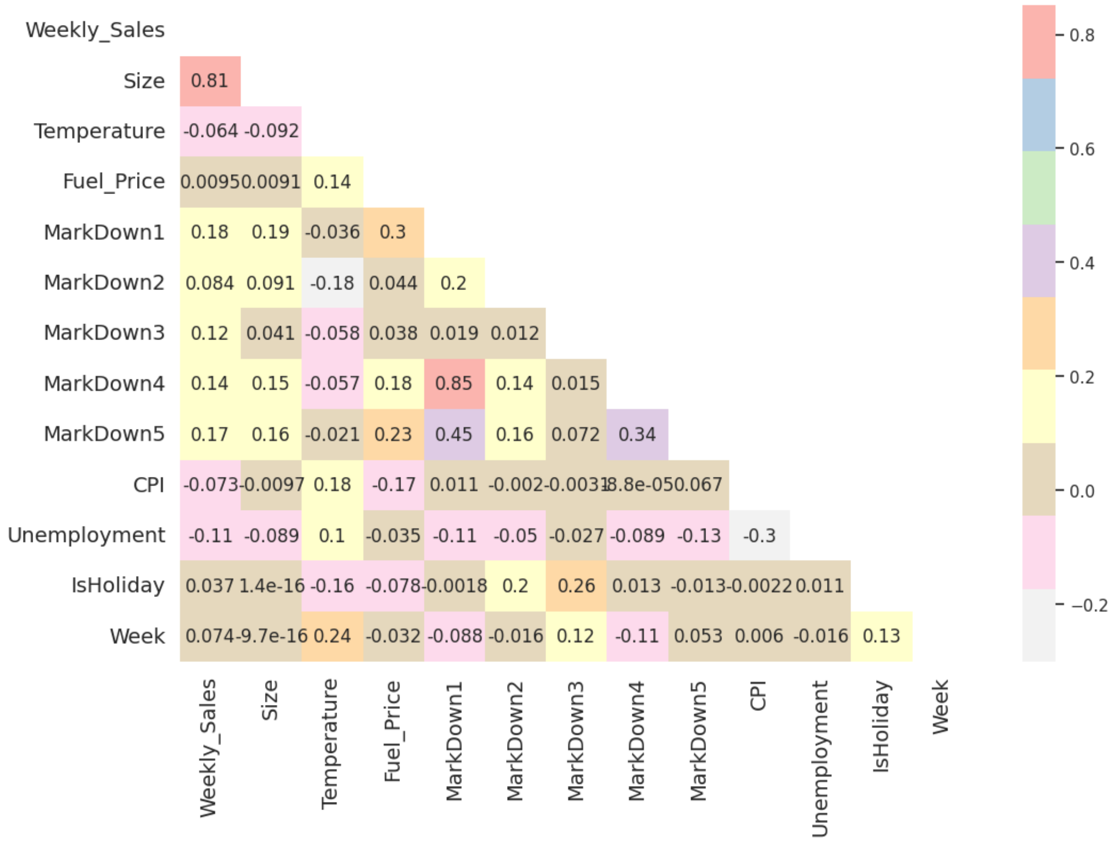
For future prediction the model is not that great because the error interval is way big. But if we just check the green line prediction this is almost like earlier years. If we look for may be first 2 weeks the prediction is way better and error is also low.

[100]:
```
# create dummy variables for 'Type' and keeping all columns to see heatmap then␣
↪will drop 1 column
Type_dummies = pd.get_dummies(data.Type, prefix='Type')

# concatenate two DataFrames (axis=0 for rows, axis=1 for columns)
data = pd.concat([data, Type_dummies], axis=1)

# Not dropping the orginal Type column now so that I can use the field in some␣
↪data analysis
```

[101]:
```
#Create a dataframe for heatmap
data_heatmap_df=data.copy()

# Eliminating all the columns that are not continuous/binary  variables from␣
↪the heatmap section.
data_heatmap_df.
↪drop(['Store','Day','Month','Year','Date','Store','Type','Type_A','Type_B','Type_C'],␣
↪axis=1,inplace=True)
```

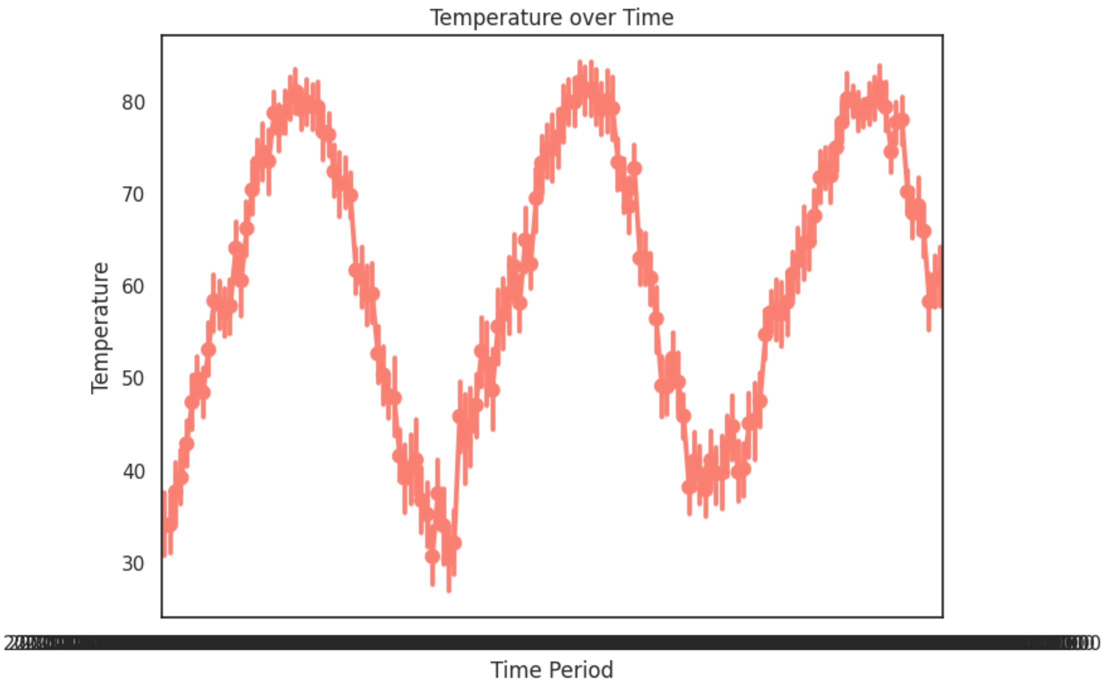
```
# Lets look the correlation matrix and heat map of the

## Correlation Heat map
def correlation_heat_map(df):
    corrs = df.corr()

    # Set the default matplotlib figure size:
    fig, ax = plt.subplots(figsize=(12,8))

    # Generate a mask for the upper triangle (taken from seaborn example␣
 ↪gallery)
    mask = np.zeros_like(corrs, dtype=np.bool)
    mask[np.triu_indices_from(mask)] = True

    # Plot the heatmap with seaborn.
    # Assign the matplotlib axis the function returns. This will let us resize␣
 ↪the labels.
    ax = sns.heatmap(corrs, mask=mask, annot=True, cmap='Pastel1_r')

    # Resize the labels.
    ax.set_xticklabels(ax.xaxis.get_ticklabels(), fontsize=14, rotation=90)
    ax.set_yticklabels(ax.yaxis.get_ticklabels(), fontsize=14, rotation=0)

    # If you put plt.show() at the bottom, it prevents those useless printouts␣
 ↪from matplotlib.
    plt.show()
```

[102]:
```
correlation_heat_map(data_heatmap_df)

#inference: By checking the direct correlation of features there is no much␣
 ↪promising correlations.
#            There are no much correlation within the features as well. In a way␣
 ↪this is good because
#            there won't be multicollinearity that we have to take care while␣
 ↪running models.
```



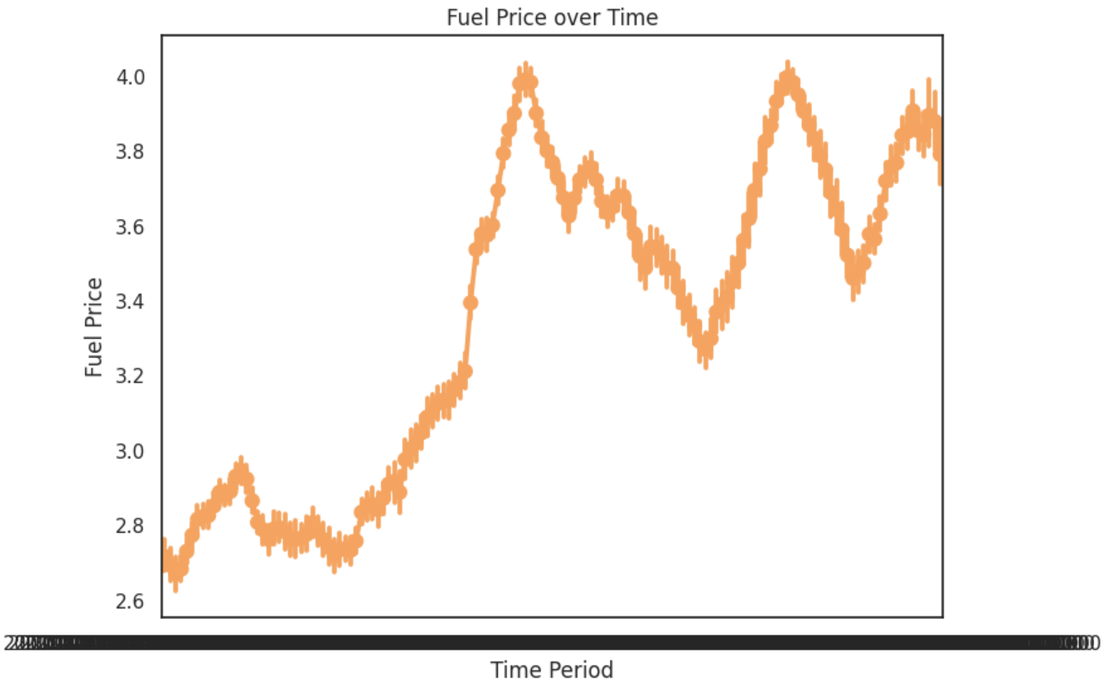
[103]:

```
plt.figure(figsize=(8,6))
sns.pointplot(x="Date", y="Temperature", data=data, color = 'salmon')
plt.xlabel('Time Period')
plt.ylabel('Temperature')
plt.title('Temperature over Time')
plt.show()
# inference: Graph clearly shows Temperature is more of a seasonal and repeated␣
↪in cycles and this would
# be an interesting data point that we can use for studies further
```



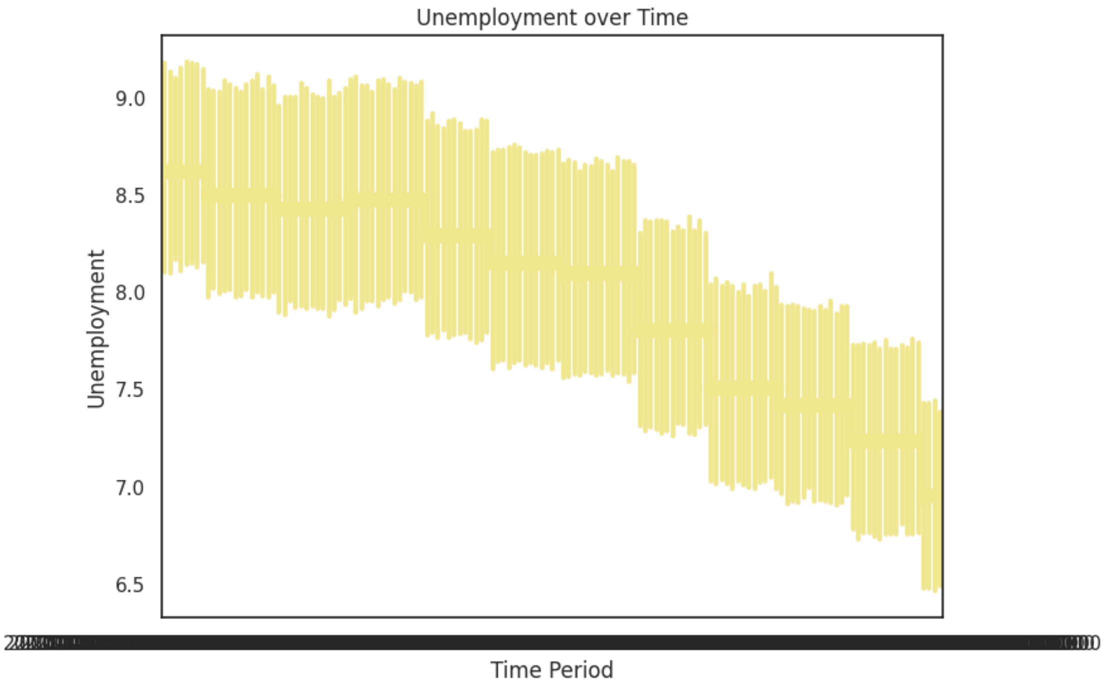
[104]:

```
plt.figure(figsize=(8,6))
sns.pointplot(x="Date", y="Fuel_Price", data=data, color = 'sandybrown')
plt.xlabel('Time Period')
plt.ylabel('Fuel Price')
plt.title('Fuel Price over Time')
plt.show()
# inference: Fuel price varies over time and there are high and lows
```



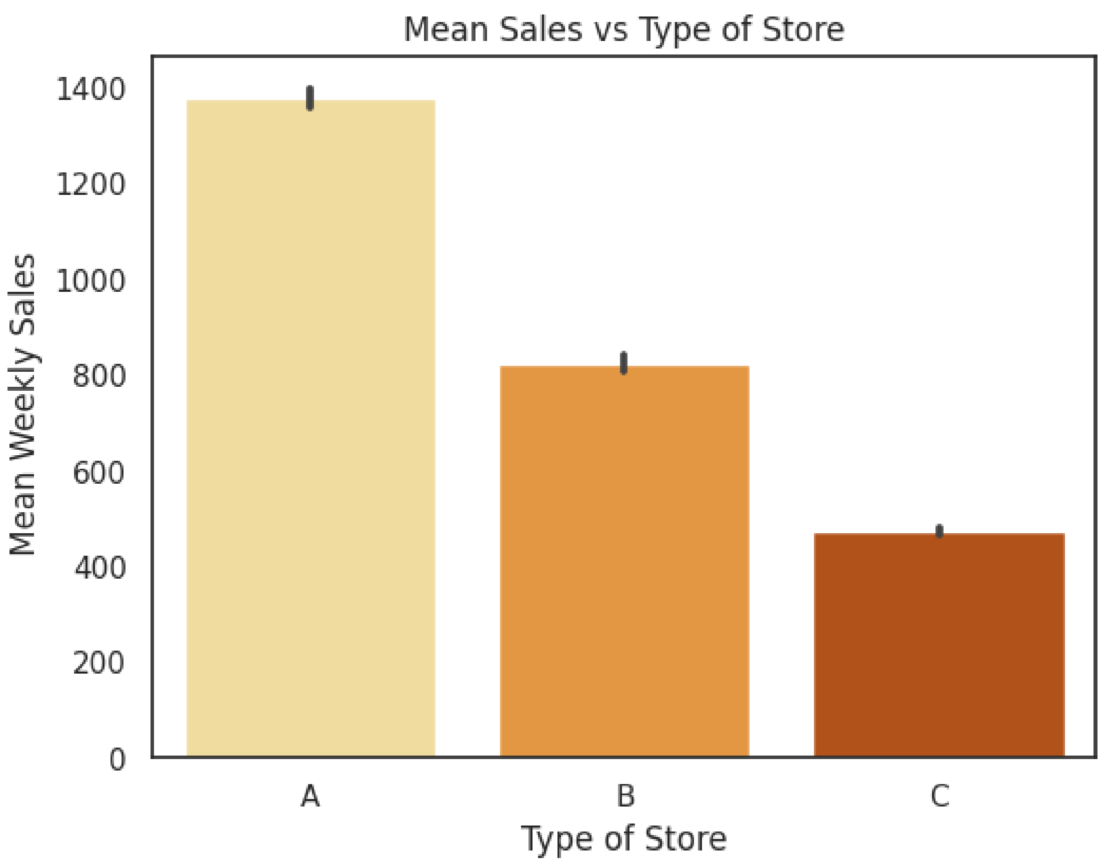
[105]:
```
plt.figure(figsize=(8,6))
sns.pointplot(x="Date", y="CPI", data=data, color = 'turquoise')
plt.xlabel('Time Period')
plt.ylabel('Consumer Price Index')
plt.title('Consumer Price Index over Time')
plt.show()
# inference: over time CPI have increased. but the change is not much
```

[106]:
```
plt.figure(figsize=(8,6))
sns.pointplot(x="Date", y="Unemployment", data=data, color='khaki')
plt.xlabel('Time Period')
plt.ylabel('Unemployment')
plt.title('Unemployment over Time')
plt.show()
# inference:  Over time unemployment have came down we can see this factor also␣
↪whether it have affected the Sales
```

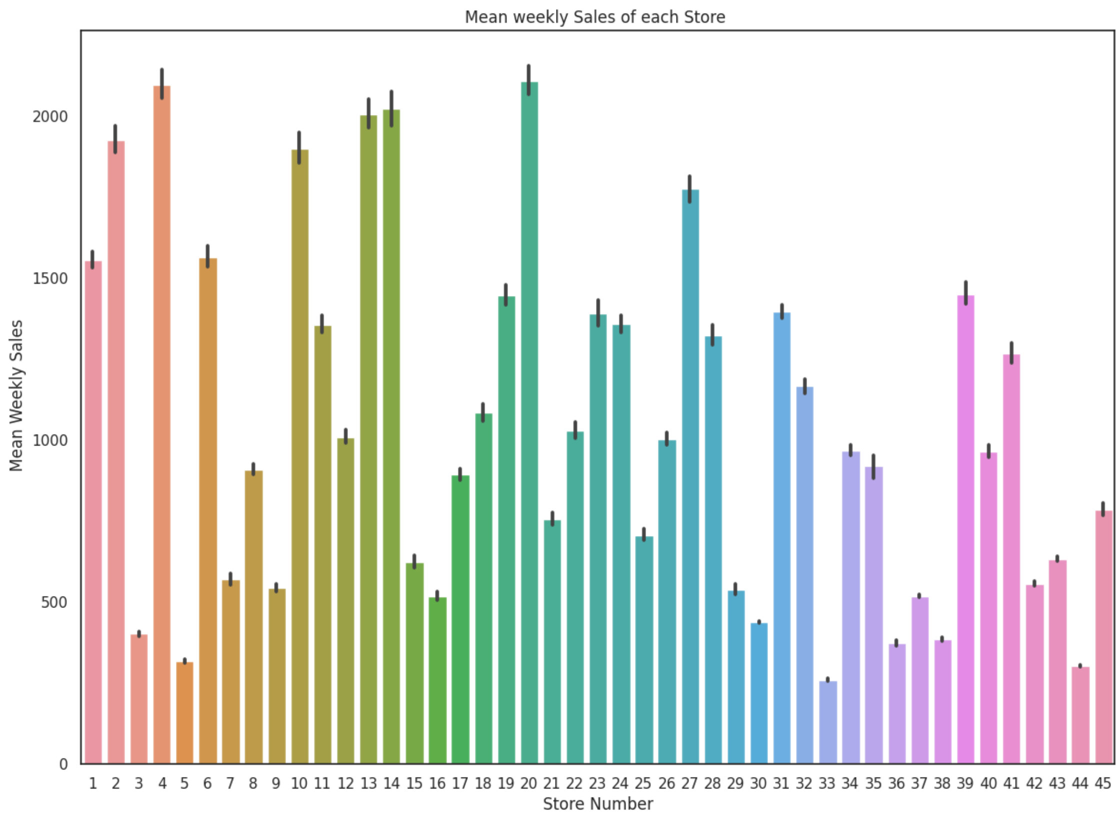


This is interesting. Features over time changes quite a bit. We will see whether these have any effects on Sales while we model

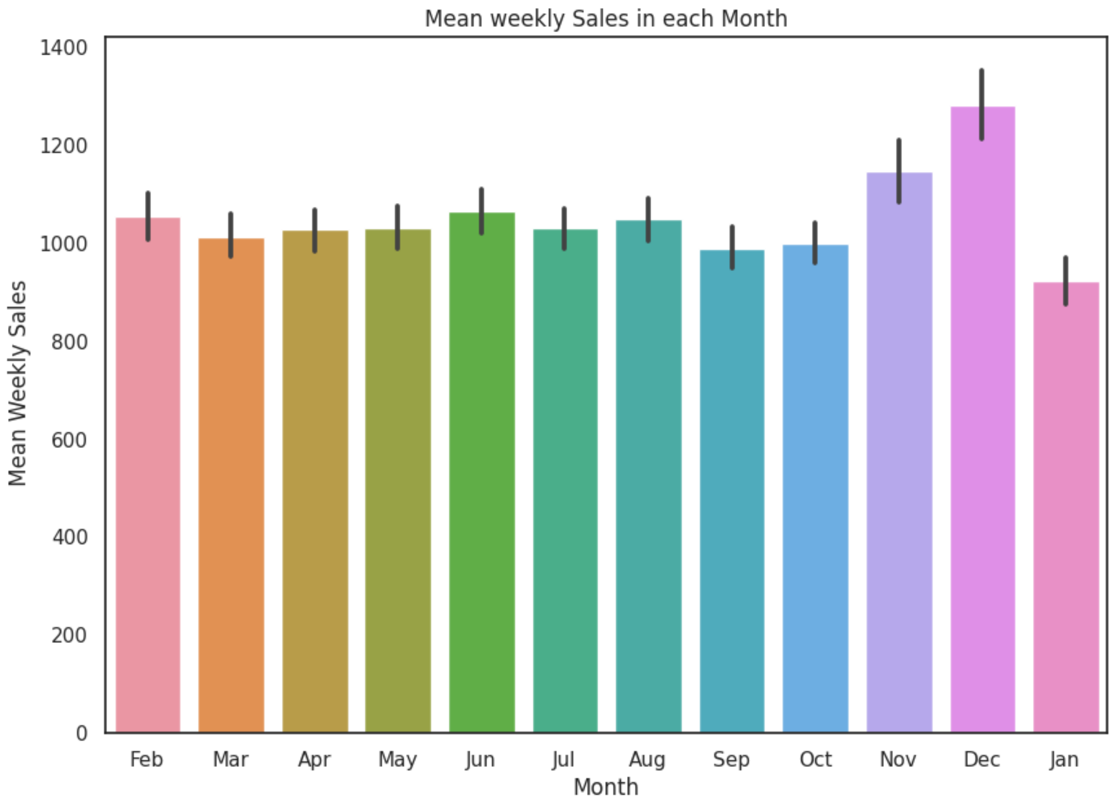
[107]:
```
# Checking how the Type of the store have effect on the sales.
col=['coral', 'greenyellow', 'turquoise']
sns.barplot(x="Type", y="Weekly_Sales", data=data,orient='v', palette ='YlOrBr')
plt.xlabel('Type of Store')
plt.ylabel(' Mean Weekly Sales')
plt.title('Mean Sales vs Type of Store')
#plt.savefig('./images/Type_vs_Sales.png')
plt.show()

# inference: From the graph its clear that Type A > Type B > Type C in mean␣
↪weekly sales.
```



[108]:
```
plt.subplots(figsize=(14,10))
sns.barplot(x="Store", y="Weekly_Sales", data=data,orient='v')
plt.xlabel('Store Number')
plt.ylabel(' Mean Weekly Sales')
plt.title('Mean weekly Sales of each Store ')
#plt.savefig('./images/Mean_Weekly_Sales_vs_Stores.png')
plt.show()

# inference : From the chart we can see that there are stores that have a␣
↪weekly sales from $250,000
#             to $2,200,000
```



[109]:
```
plt.subplots(figsize=(10,7))
sns.barplot(x="Month", y="Weekly_Sales", data=data,orient='v')
plt.xlabel('Month')
plt.ylabel(' Mean Weekly Sales')
plt.title('Mean weekly Sales in each Month')
#plt.savefig('./images/Mean_Weekly_Sales_vs_Months.png')
plt.show()
# inference: Graph shows sales in each month and from this we can see December␣
↪seems to have a very high sales
#            compared to every other month and January have the least sales.
```



With this we come to an end of EDA & Time series analysis. We will now move forward with Machine Learning & Modelling

[110]:
```
# Create Week column which says which week of the month it is.
data["Week"]= round(np.floor(((data.Day-1)/7)+1))

# Create dummies for the columns that are required for later studies
Store_dummies = pd.get_dummies(data.Store, prefix='Store')
Month_dummies = pd.get_dummies(data.Month, prefix='Month')
Year_dummies = pd.get_dummies(data.Year, prefix='Year')
Week_dummies = pd.get_dummies(data.Week, prefix='Week')

# concatenate DataFrames (axis=0 for rows, axis=1 for columns)
data = pd.concat([data, Store_dummies,Month_dummies,Year_dummies,Week_dummies],
 ↪axis=1)
```

[111]:
```
data_decision=data.iloc[:,:18]

data_decision["Week"]= round(np.floor(((data_decision.Day-1)/7)+1))
```

[112]:
```
# Drop the columns that we have created dummies
data.drop(['Type', 'Store','Month','Year','Day','Week'], axis=1, inplace=True)
```

[113]:
```
# drop each column from the list of dummies to make it perfect to use in models
data.drop(['Type_C', 'Store_1','Month_Jan','Year_2010','Week_5.0'], axis=1,␣
↪inplace=True)
```

[114]:
```
data.iloc[:,5:10].describe().T

# Inference: more than 50% is missing values with (-500) so imputing with KNN␣
↪might not be a good idea.
# But what are the other methods? imputing with random values in the range of␣
↪that particular columns?
# Lets try that first.
```

[114]:
```
            count         mean          std    min    25%    50%       75%  \
MarkDown1  6435.0  2106.175500  6008.334618 -500.0 -500.0 -500.0  2302.300
MarkDown2  6435.0   446.067837  4946.234382 -500.0 -500.0 -500.0     0.090
MarkDown3  6435.0    88.158396  5306.320800 -500.0 -500.0 -500.0     3.705
MarkDown4  6435.0   661.551088  3853.055534 -500.0 -500.0 -500.0   314.320
MarkDown5  6435.0  1260.128491  4227.342723 -500.0 -500.0 -500.0  1983.265

                 max
MarkDown1   88646.76
MarkDown2  104519.54
MarkDown3  141630.61
MarkDown4   67474.85
MarkDown5  108519.28
```

[115]:
```
data.MarkDown1=data.MarkDown1.map(lambda x: np.nan if x==-500 else x)
data.MarkDown2=data.MarkDown2.map(lambda x: np.nan if x==-500 else x)
data.MarkDown3=data.MarkDown3.map(lambda x: np.nan if x==-500 else x)
data.MarkDown4=data.MarkDown4.map(lambda x: np.nan if x==-500 else x)
data.MarkDown5=data.MarkDown5.map(lambda x: np.nan if x==-500 else x)
```

[116]:
```
missing_cols = ['MarkDown1','MarkDown2','MarkDown3','MarkDown4','MarkDown5']

# Not including our actual y(Weekly Sales) and Size of store for Markdown since␣
↪by including weekly sales
# It can be a bad method to use those MarkDown again for predicting weekly␣
↪sales.

impute_cols = [c for c in data.columns if not c in␣
↪['Weekly_Sales','Date','Sqrt_Sales','lnSales']+missing_cols]

data_imputed=data.copy()
```

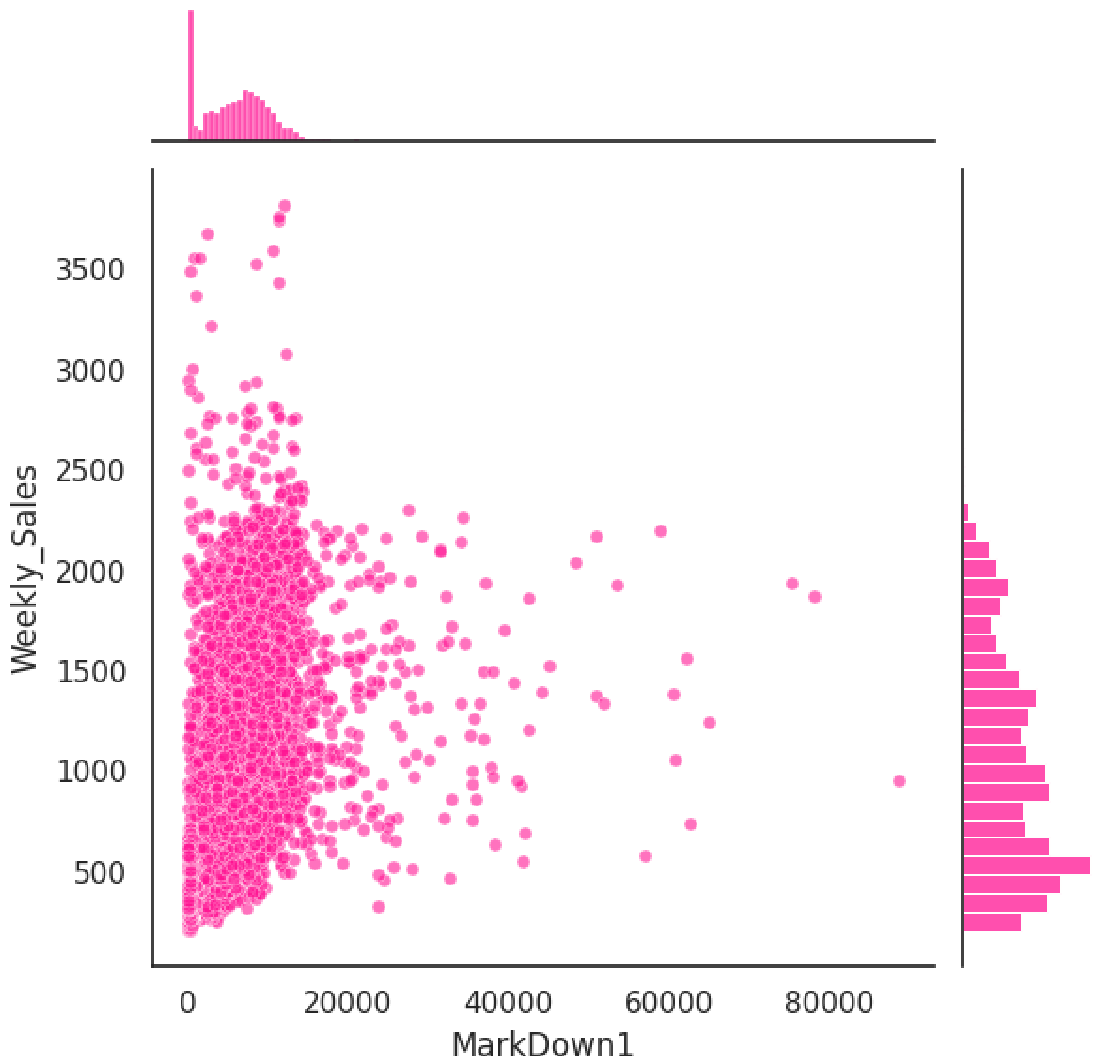
[117]:
```
def find_best_k_reg(X, y, k_min=1, k_max=51, step=2, cv=10):
    k_range = range(k_min, k_max+1, step)
    r2s = []
```

```
    for k in k_range:
        knn = KNeighborsRegressor(n_neighbors=k)
        scores = cross_val_score(knn, X, y, cv=cv)
        r2s.append(np.mean(scores))
    print ("Best R2 value:",np.max(r2s),"\nBest k: ",np.argmax(k_range))
    return np.argmax(k_range)
```

[118]:
```
impute_missing = data.loc[data.MarkDown1.isnull(), :]
impute_valid = data.loc[~data.MarkDown1.isnull(), :]

y = impute_valid.MarkDown1.values
X = impute_valid[impute_cols]

Xs = ss.fit_transform(X)
```

[119]:
```
best_k = find_best_k_reg(Xs, y)
knn = KNeighborsRegressor(n_neighbors=best_k)
knn.fit(Xs, y)

X_miss = impute_missing[impute_cols]
X_miss_s = ss.transform(X_miss)

MarkDown1_impute = knn.predict(X_miss_s)

data_imputed.loc[data.MarkDown1.isnull(), 'MarkDown1'] = MarkDown1_impute

#Lets look how the MarkDown1 vs Weekly_Sales appear
sns.jointplot(data_imputed.MarkDown1, data_imputed.Weekly_Sales,␣
 ↪joint_kws=dict(s=25, alpha=0.6), color='deeppink')
plt.show()
```

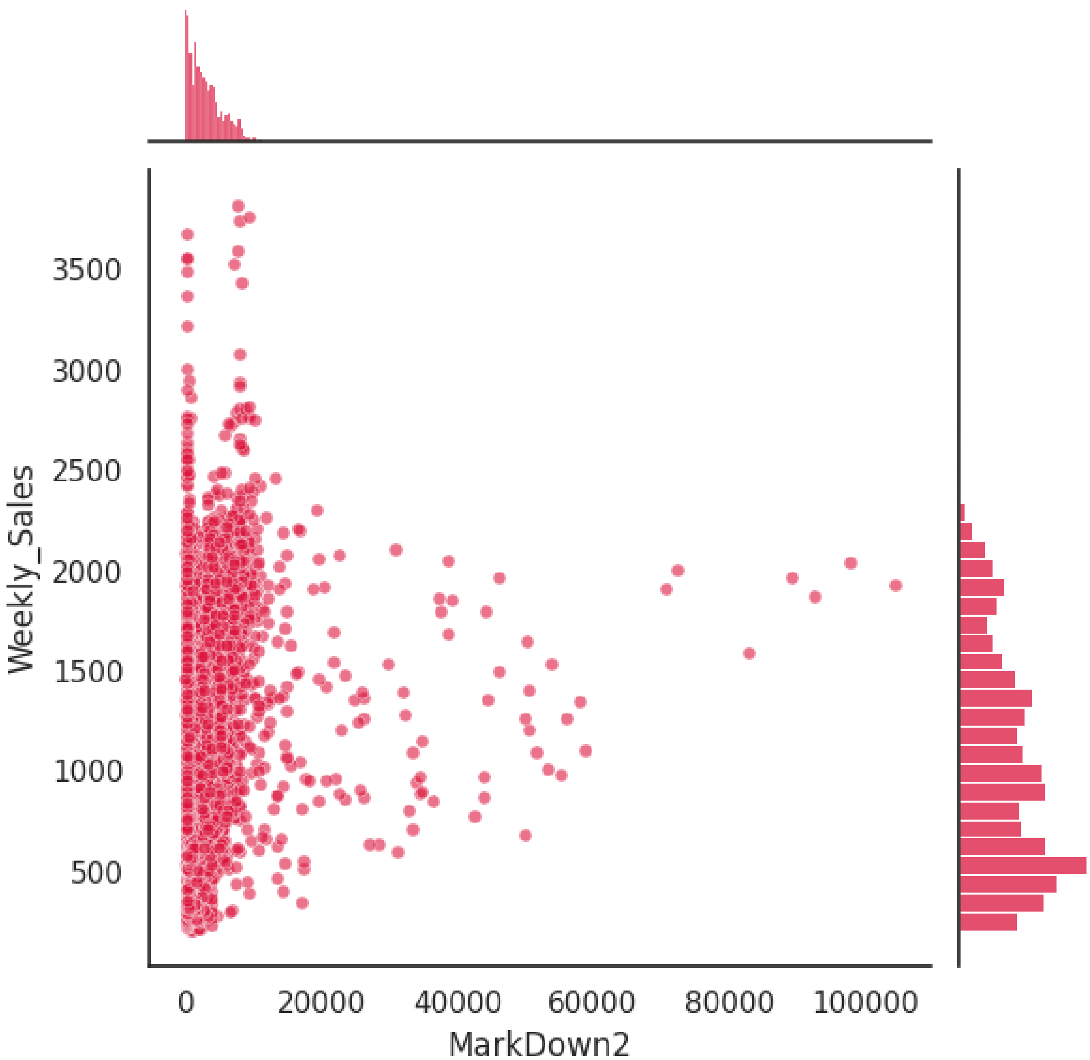
```
Best R2 value: 0.14140639091506765
Best k:  25
```



[120]:
```
impute_missing = data.loc[data.MarkDown2.isnull(), :]
impute_valid = data.loc[~data.MarkDown2.isnull(), :]

y = impute_valid.MarkDown2.values
X = impute_valid[impute_cols]

ss = StandardScaler()
Xs = ss.fit_transform(X)
best_k = find_best_k_reg(Xs, y)
knn = KNeighborsRegressor(n_neighbors=best_k)
knn.fit(Xs, y)

X_miss = impute_missing[impute_cols]
```

```
X_miss_s = ss.transform(X_miss)

MarkDown2_impute = knn.predict(X_miss_s)

data_imputed.loc[data.MarkDown2.isnull(), 'MarkDown2'] = MarkDown2_impute

#Lets look how the MarkDown1 vs Weekly_Sales appear
sns.jointplot(data_imputed.MarkDown2, data_imputed.Weekly_Sales,␣
 ↪joint_kws=dict(s=25, alpha=0.6), color = 'crimson')
plt.show()
```

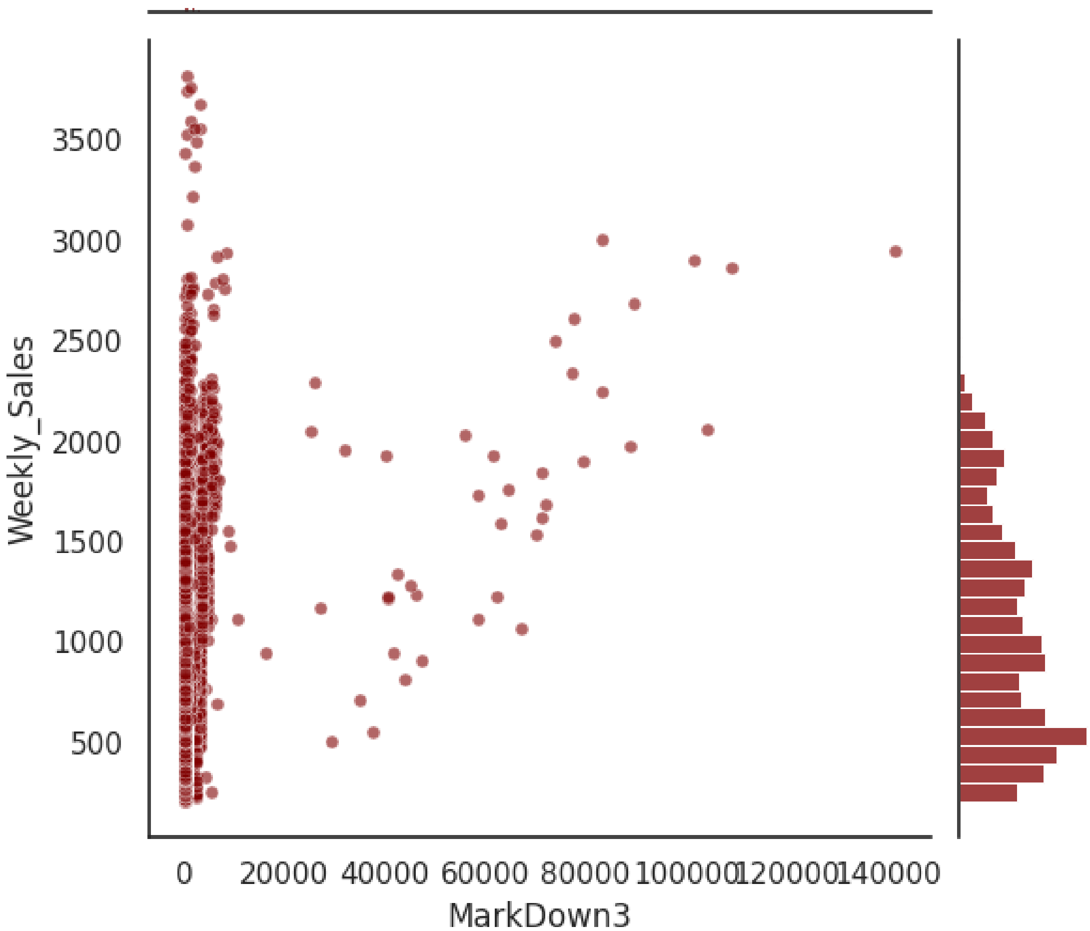
```
Best R2 value: 0.41503402148479146
Best k:  25
```



[121]:

```
impute_missing = data.loc[data.MarkDown3.isnull(), :]
impute_valid = data.loc[~data.MarkDown3.isnull(), :]

y = impute_valid.MarkDown3.values
X = impute_valid[impute_cols]

ss = StandardScaler()
Xs = ss.fit_transform(X)
best_k = find_best_k_reg(Xs, y)
knn = KNeighborsRegressor(n_neighbors=best_k)
knn.fit(Xs, y)

X_miss = impute_missing[impute_cols]
X_miss_s = ss.transform(X_miss)

MarkDown3_impute = knn.predict(X_miss_s)

data_imputed.loc[data.MarkDown3.isnull(), 'MarkDown3'] = MarkDown3_impute

sns.jointplot(data_imputed.MarkDown3, data_imputed.Weekly_Sales,␣
 ↪joint_kws=dict(s=25, alpha=0.6), color= 'maroon')
plt.show()
```

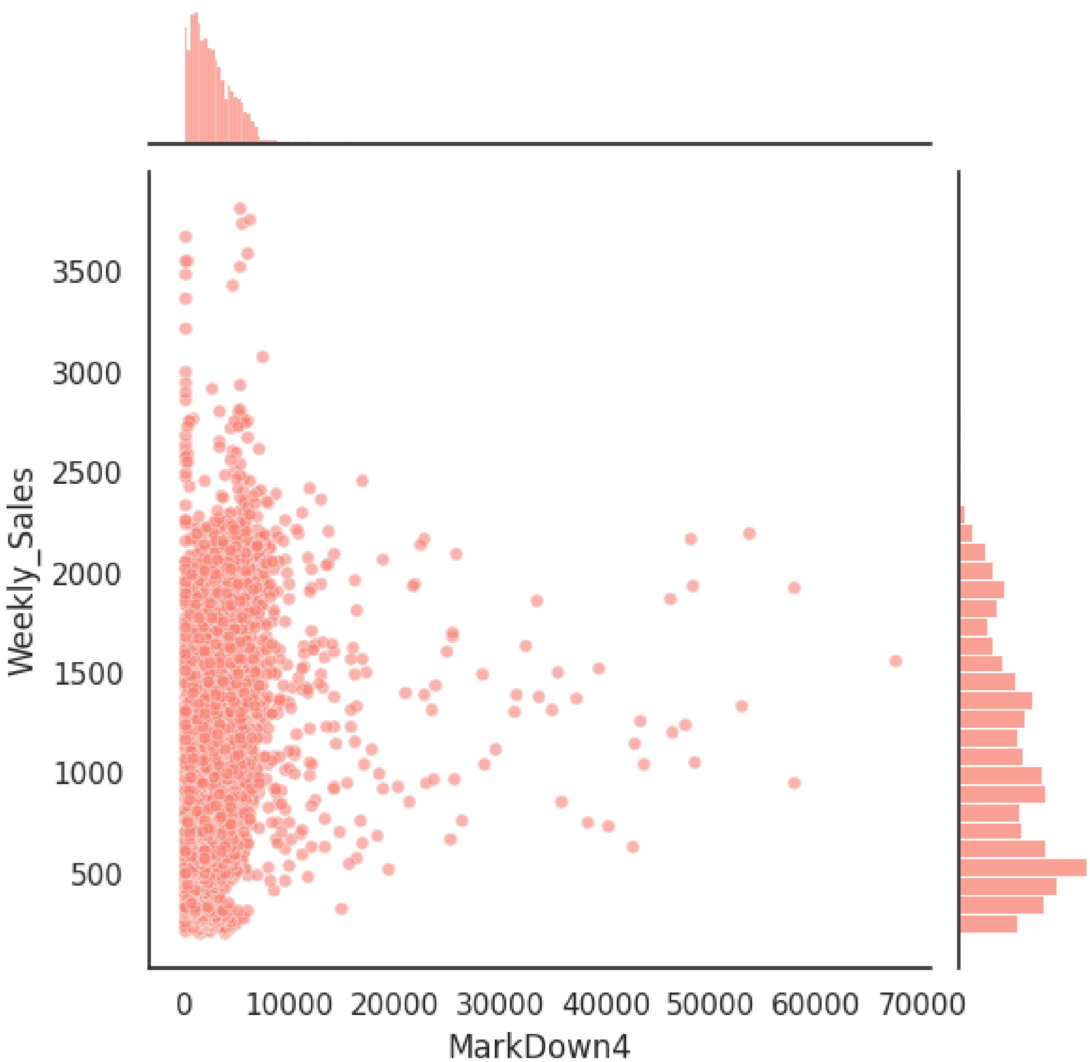
```
Best R2 value: 0.20431863554303203
Best k:  25
```



[122]:
```
impute_missing = data.loc[data.MarkDown4.isnull(), :]
impute_valid = data.loc[~data.MarkDown4.isnull(), :]

y = impute_valid.MarkDown4.values
X = impute_valid[impute_cols]

ss = StandardScaler()
Xs = ss.fit_transform(X)
best_k = find_best_k_reg(Xs, y)
knn = KNeighborsRegressor(n_neighbors=best_k)
knn.fit(Xs, y)

X_miss = impute_missing[impute_cols]
```

```
X_miss_s = ss.transform(X_miss)

MarkDown4_impute = knn.predict(X_miss_s)

data_imputed.loc[data.MarkDown4.isnull(), 'MarkDown4'] = MarkDown4_impute

sns.jointplot(data_imputed.MarkDown4, data_imputed.Weekly_Sales,␣
↪joint_kws=dict(s=25, alpha=0.6), color = 'salmon')
plt.show()
```

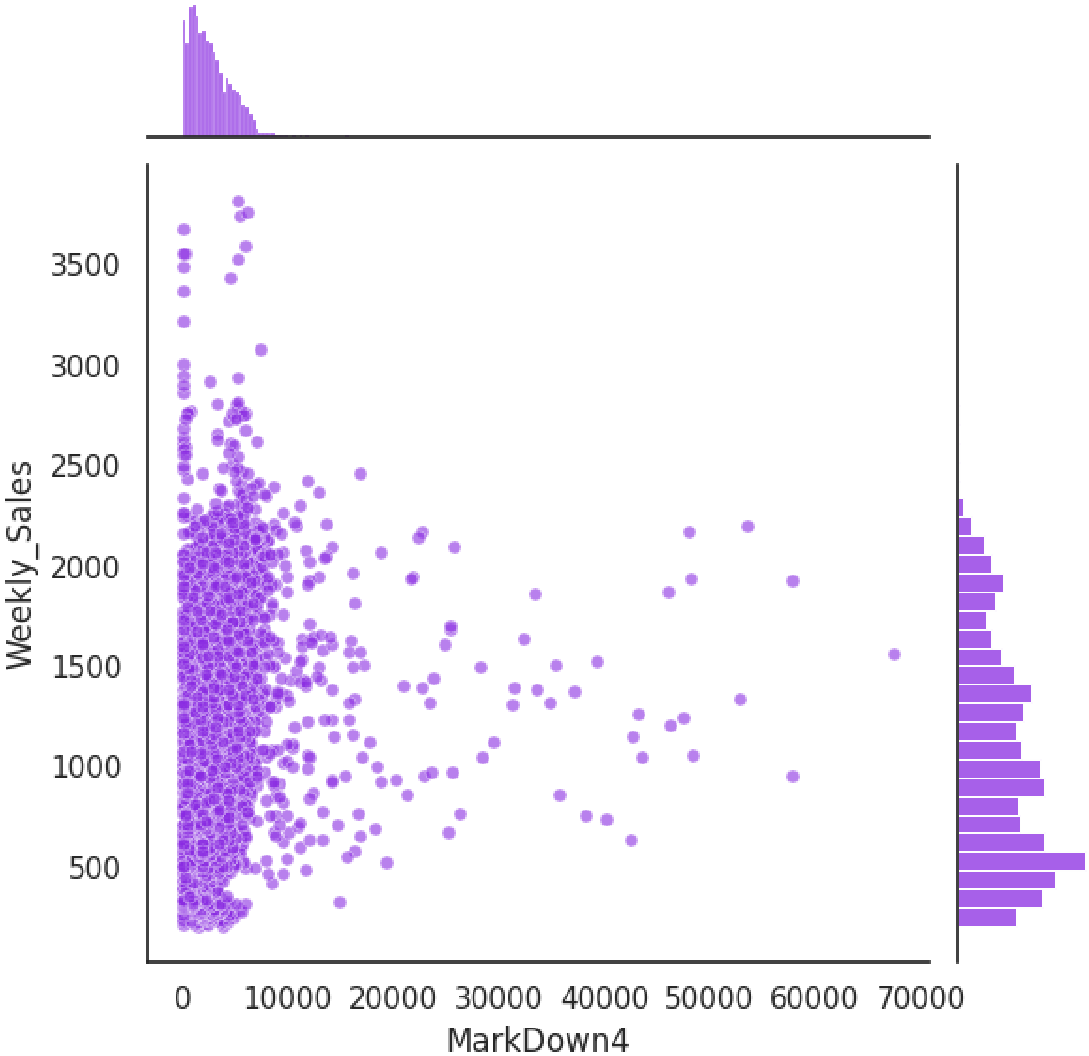
```
Best R2 value: 0.3470822603116795
Best k:  25
```



[123]:

```
impute_missing = data.loc[data.MarkDown5.isnull(), :]
impute_valid = data.loc[~data.MarkDown5.isnull(), :]

y = impute_valid.MarkDown5.values
X = impute_valid[impute_cols]

ss = StandardScaler()
Xs = ss.fit_transform(X)
best_k = find_best_k_reg(Xs, y)
knn = KNeighborsRegressor(n_neighbors=best_k)
knn.fit(Xs, y)

X_miss = impute_missing[impute_cols]
X_miss_s = ss.transform(X_miss)

MarkDown5_impute = knn.predict(X_miss_s)

data_imputed.loc[data.MarkDown5.isnull(), 'MarkDown5'] = MarkDown5_impute

sns.jointplot(data_imputed.MarkDown4, data_imputed.Weekly_Sales,␣
 ↪joint_kws=dict(s=25, alpha=0.6), color = 'blueviolet')
plt.show()
```

```
Best R2 value: 0.0815206539204145
Best k:  25
```



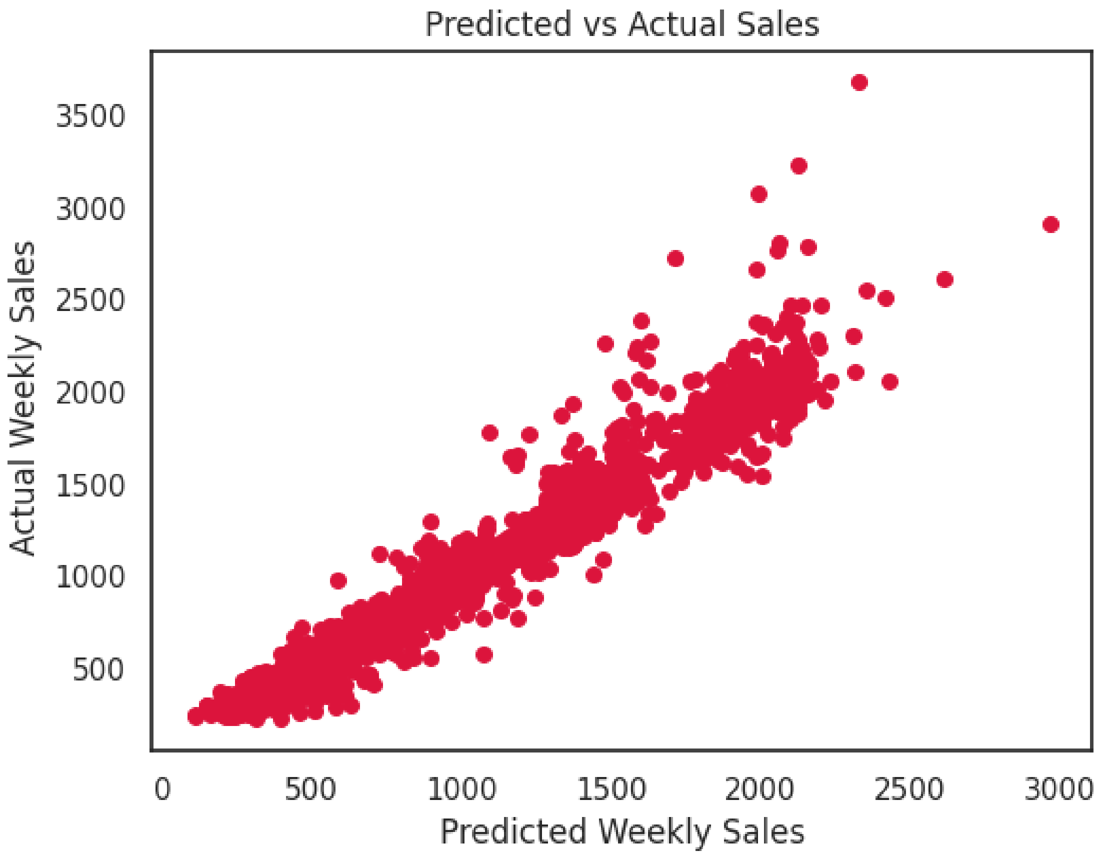
[124]:
```
walmart_data=data_imputed.copy()

# The accuracy and R2 are very bad. This means that we likely imputing crap␣
↪with these models.
# This doesn't necessarily mean that imputation is a bad idea, but we may want␣
↪to consider
# using a different method.
```

[125]:
```
predictors=[col for col in data_imputed.columns if col not in␣
↪['Date','Weekly_Sales']]
X=data_imputed[predictors]
y=data_imputed.Weekly_Sales.values
```

```
Xs = ss.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(Xs, y, test_size=0.33)

mlr = LinearRegression()
mlr.fit(X_train, y_train)
r2=mlr.score(X_test, y_test)
print(mlr.score(X_test, y_test))
print(mlr.score(X_train, y_train))
adj_r2 = 1 - (len(y)-1)/(len(y)-X.shape[1]-1)*(1-r2)
print("Adjusted R^2",adj_r2)


# Perform 10-fold cross validation
scores = cross_val_score(mlr, X_train, y_train, cv=10)
print ("Cross-validated scores:", scores)
print ("Mean Cross validation",scores.mean())



# Make cross validated predictions on the test sets
predictions = cross_val_predict(mlr, X_test, y_test, cv=10)

plt.scatter(predictions, y_test, s=30, c='crimson', zorder=10)
plt.xlabel('Predicted Weekly Sales')
plt.ylabel(' Actual Weekly Sales')
plt.title('Predicted vs Actual Sales')
```

```
0.9489188739348253
0.9421334223906142
Adjusted R^2 0.948324533788784
Cross-validated scores: [0.9357271  0.93327463 0.9454755  0.93791479 0.93442197
0.920653
 0.95233081 0.94636528 0.94963853 0.94596254]
Mean Cross validation 0.9401764144060213
```

[125]: Text(0.5, 1.0, 'Predicted vs Actual Sales')



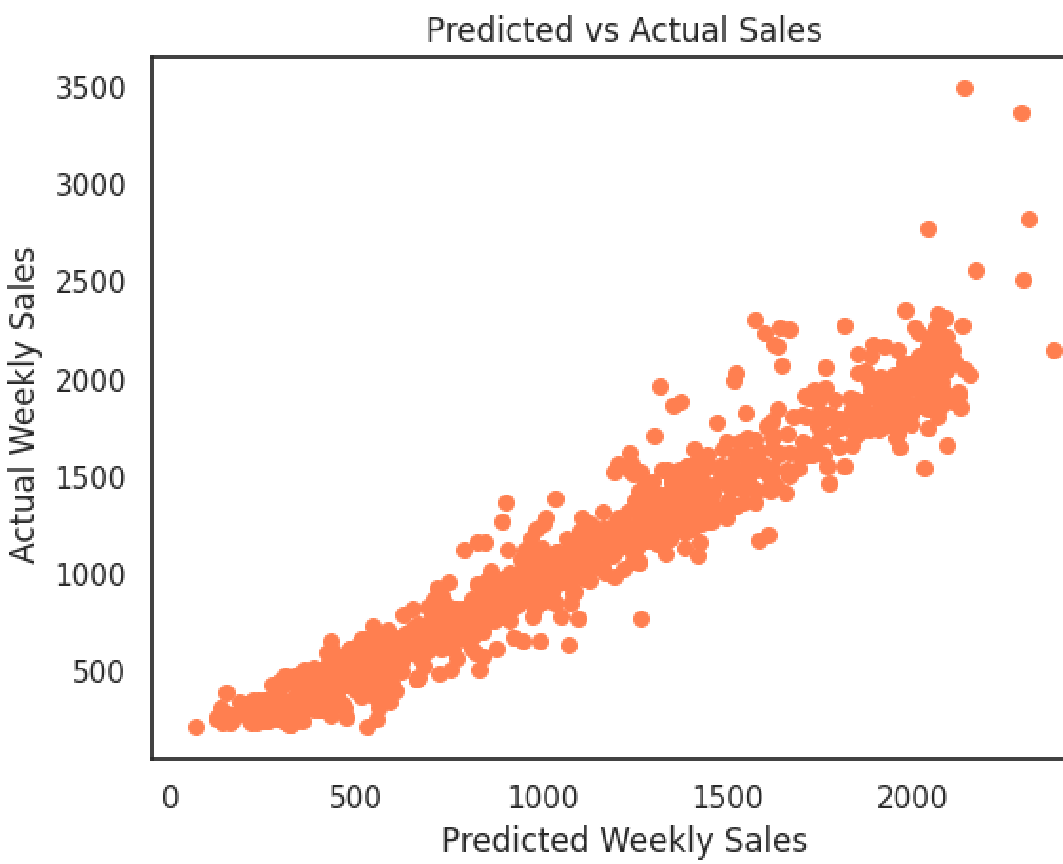
Now let us look the same model without MarkDowns to check whether data with MarkDown or without MarkDown is good.

[126]:
```
predictors=[col for col in data.columns if col not in ['Date','Weekly_Sales']]
predictors=[col for col in predictors if 'MarkDown' not in col]
X=data[predictors]
y=data.Weekly_Sales.values
Xs = ss.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(Xs, y, test_size=0.2)

lr = LinearRegression()
lr.fit(X_train, y_train)

print(lr.score(X_test, y_test))
print(lr.score(X_train, y_train))


# Perform 10-fold cross validation
scores = cross_val_score(lr, X, y, cv=10)
```

```
print ("Cross-validated scores:", scores)
print ("Mean Cross validation",scores.mean())



# Make cross validated predictions on the test sets
predictions = cross_val_predict(lr, X_test, y_test, cv=10)

plt.scatter(predictions, y_test, s=30, color = 'coral', zorder=10)
plt.xlabel('Predicted Weekly Sales')
plt.ylabel(' Actual Weekly Sales')
plt.title('Predicted vs Actual Sales')
```

```
0.945265790613953
0.9389945374708548
Cross-validated scores: [ 7.00623669e-01 -1.24124887e+14 -3.24124426e+13
-1.90030137e+13
 -1.38598264e+13 -5.65271030e+13 -7.19320597e+10 -2.59063572e+13
 -2.93677442e+12 -7.47144847e+13]
Mean Cross validation -34955682152777.6
```

[126]: Text(0.5, 1.0, 'Predicted vs Actual Sales')



[127]:
```
data=data_imputed.copy()
```

We will divide our train and test datasets first and then deal with that seperately

[128]:
```
# Setting the offset to finalize the test data.
offset = timedelta(days=90)
split_date=data.Date.max()-offset
```

[129]:
```
data_train=data[data.Date < split_date]
data_test=data[data.Date > split_date]
```

Before we start lets shuffle the dataframe a bit because while we use crossvalidation for regressors it won't take a random sample as test and train, instead it takes section by section. Here my Dataframe have data for each store in order. So if we take section by section model might not have enough data to learn about certain stores and which intern will give terrible answers

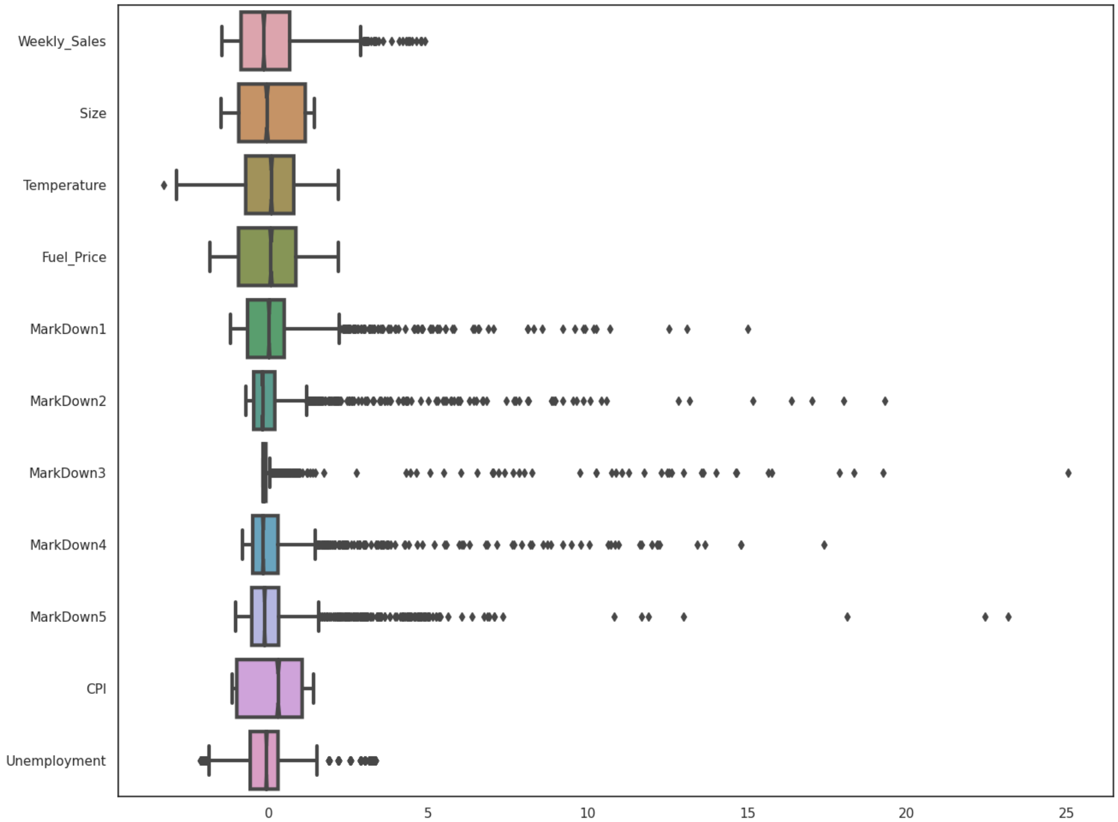
[130]:
```
data_train = data_train.reindex(np.random.permutation(data_imputed.index))##␣
 ↪Identify outliers
```

[131]:
```
data_train.columns
```

[131]:
```
Index(['Date', 'Weekly_Sales', 'Size', 'Temperature', 'Fuel_Price',
       'MarkDown1', 'MarkDown2', 'MarkDown3', 'MarkDown4', 'MarkDown5', 'CPI',
       'Unemployment', 'IsHoliday', 'Type_A', 'Type_B', 'Store_2', 'Store_3',
       'Store_4', 'Store_5', 'Store_6', 'Store_7', 'Store_8', 'Store_9',
       'Store_10', 'Store_11', 'Store_12', 'Store_13', 'Store_14', 'Store_15',
       'Store_16', 'Store_17', 'Store_18', 'Store_19', 'Store_20', 'Store_21',
       'Store_22', 'Store_23', 'Store_24', 'Store_25', 'Store_26', 'Store_27',
       'Store_28', 'Store_29', 'Store_30', 'Store_31', 'Store_32', 'Store_33',
       'Store_34', 'Store_35', 'Store_36', 'Store_37', 'Store_38', 'Store_39',
       'Store_40', 'Store_41', 'Store_42', 'Store_43', 'Store_44', 'Store_45',
       'Month_Apr', 'Month_Aug', 'Month_Dec', 'Month_Feb', 'Month_Jul',
       'Month_Jun', 'Month_Mar', 'Month_May', 'Month_Nov', 'Month_Oct',
       'Month_Sep', 'Year_2011', 'Year_2012', 'Week_1.0', 'Week_2.0',
       'Week_3.0', 'Week_4.0'],
      dtype='object')
```

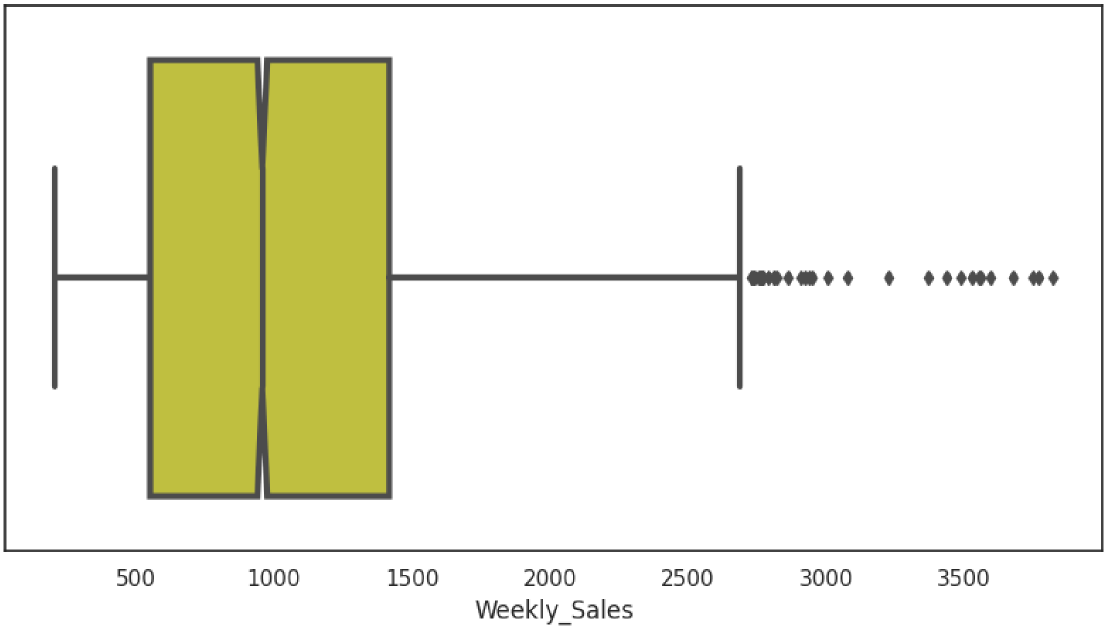
[132]:
```
data_box=data_train.iloc[:, 1:12]
data_norm = (data_box - data_box.mean()) / data_box.std()

fig = plt.figure(figsize=(15, 12))
ax = fig.gca()

ax = sns.boxplot(data=data_norm, orient='h', fliersize=5,
                 linewidth=3, notch=True, saturation=0.5, ax=ax)
```

```
plt.show()
```



There are quite a lot of outliers in MarkDown, But Lets first deal the outliers in weekly sales data because we might just drop MarkDowns Later because the percentage of missing values are really high in MarkDowns

[133]:
```
fig = plt.figure(figsize=(10, 5))
ax = fig.gca()

ax = sns.boxplot(data_train.Weekly_Sales, orient='h', fliersize=5,
                 linewidth=3, notch=True, saturation=0.5, ax=ax, color =␣
 ↪'yellow')
plt.show()
```



[134]:
```
# Lets consider 3,000,000 as upper limit
data_train[data_train.Weekly_Sales>3000].shape
```

[134]: (14, 76)

[135]:
```
# there is only 14 outliers. Lets drop it and proceed.
data_train=data_train[data_train.Weekly_Sales<3000]
```

[136]:
```
predictors=[col for col in data.columns if col not in␣
↪['Weekly_Sales','Sqrt_Sales','lnSales','Date']] # Date

predictors=[col for col in predictors if 'Month' not in col]
predictors=[col for col in predictors if 'Week' not in col]
predictors=[col for col in predictors if 'Year' not in col]
```

[137]:
```
X_train = data_train[predictors]
y_train = data_train.Weekly_Sales.values

X_test = data_test[predictors]
y_test = data_test.Weekly_Sales.values
```

[138]:
```
X_train_s=ss.fit_transform(X_train)
X_test_s=ss.fit_transform(X_test)
```

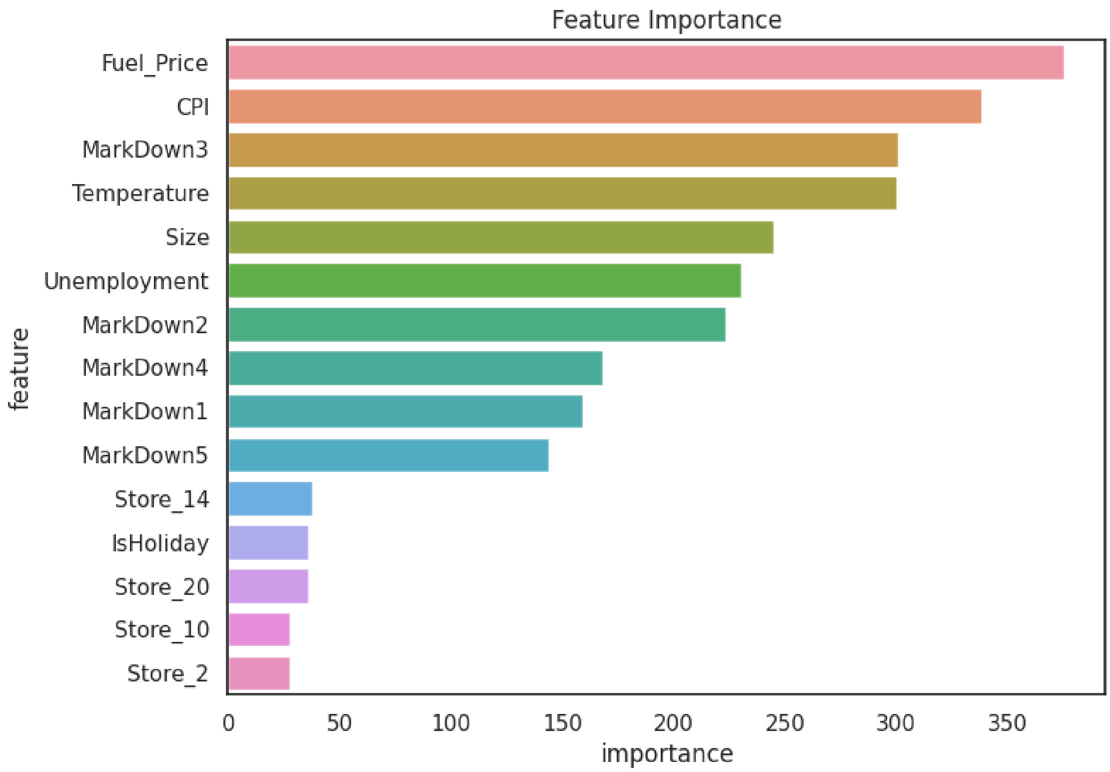
[139]:
```
lgbm_features = lgb.LGBMRegressor()
```

[140]:
```
lgbm_features.fit(X_train, y_train)
```

[140]:
```
LGBMRegressor()
```

[141]:
```
importance_df = pd.DataFrame({
    'feature': X_train.columns,
    'importance': lgbm_features.feature_importances_
}).sort_values('importance', ascending=False)
```

[142]:
```
plt.figure(figsize=(8,6))
plt.title('Feature Importance')
sns.barplot(data=importance_df.head(15), x='importance', y='feature');
```



Clearly we have the top features listed above. Top five :

```
Fuel_Price
CPI
Markdown3
Temperature
Size
```

[143]:
```
lasso_cv = LassoCV(n_alphas=1000,max_iter=2000, cv=10, verbose=1)
lasso_cv.fit(X_train_s, y_train)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
```

...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...
...

```
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
…
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:    3.0s finished
```

[143]:
```
LassoCV(cv=10, max_iter=2000, n_alphas=1000, verbose=1)
```

[144]:
```python
# Put the features and coefs into a dataframe
# sort by magnitude
lasso_feat = pd.DataFrame(dict(feature=X_train.columns, coef=lasso_cv.coef_,␣
 ↪abscoef=np.abs(lasso_cv.coef_)))
lasso_feat.sort_values('abscoef', inplace=True, ascending=False)
# main_features
lasso_feat[lasso_feat.coef != 0.]
```

[144]:

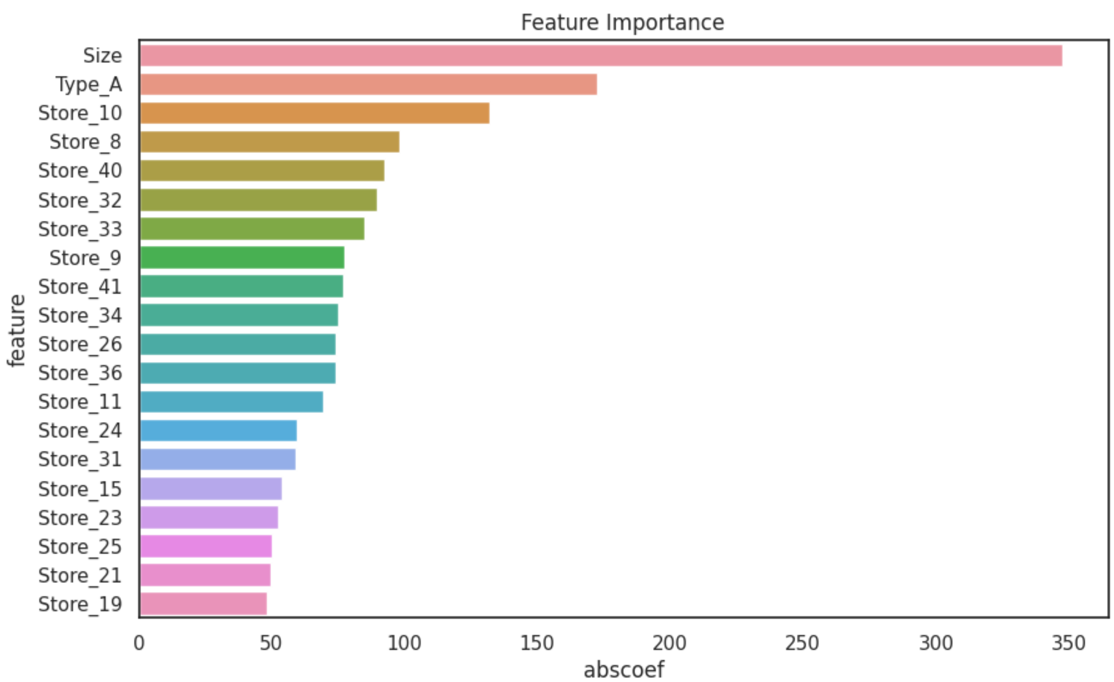
	feature	coef	abscoef
0	Size	347.440079	347.440079
11	Type_A	172.384215	172.384215
21	Store_10	131.966692	131.966692
19	Store_8	-98.376268	98.376268

51	Store_40	-92.706010	92.706010
43	Store_32	-89.647948	89.647948
44	Store_33	-85.179793	85.179793
20	Store_9	-77.209609	77.209609
52	Store_41	-77.163473	77.163473
45	Store_34	-75.254886	75.254886
37	Store_26	-74.236073	74.236073
47	Store_36	-74.104517	74.104517
22	Store_11	-69.370005	69.370005
35	Store_24	-59.758579	59.758579
42	Store_31	-58.984018	58.984018
26	Store_15	-53.699262	53.699262
34	Store_23	52.273264	52.273264
36	Store_25	-49.965329	49.965329
32	Store_21	-49.712121	49.712121
30	Store_19	-48.335226	48.335226
9	Unemployment	-46.894832	46.894832
39	Store_28	-46.295713	46.295713
25	Store_14	40.361164	40.361164
31	Store_20	39.469250	39.469250
5	MarkDown3	38.995491	38.995491
50	Store_39	-38.738596	38.738596
55	Store_44	-37.788295	37.788295
17	Store_6	-36.175284	36.175284
16	Store_5	-35.272260	35.272260
40	Store_29	-34.947106	34.947106
23	Store_12	33.918002	33.918002
15	Store_4	32.747364	32.747364
27	Store_16	-24.775158	24.775158
56	Store_45	-23.330999	23.330999
14	Store_3	-21.384872	21.384872
18	Store_7	-20.438032	20.438032
54	Store_43	20.253820	20.253820
29	Store_18	19.693823	19.693823
41	Store_30	-18.265050	18.265050
13	Store_2	16.940394	16.940394
4	MarkDown2	-13.557513	13.557513
24	Store_13	11.418004	11.418004
1	Temperature	-11.400903	11.400903
2	Fuel_Price	-10.287153	10.287153
10	IsHoliday	9.922931	9.922931
33	Store_22	9.539345	9.539345
8	CPI	8.564592	8.564592
46	Store_35	8.259022	8.259022
53	Store_42	7.978034	7.978034
7	MarkDown5	4.842193	4.842193
48	Store_37	-2.774176	2.774176

```
28       Store_17    1.988387    1.988387
6       MarkDown4   -1.807744    1.807744
```

[145]:
```python
importance_df = pd.DataFrame({
    'feature': X_train.columns,
    'importance': lgbm_features.feature_importances_
}).sort_values('importance', ascending=False)
```

[146]:
```python
plt.figure(figsize=(10,6))
plt.title('Feature Importance')
sns.barplot(data=lasso_feat.head(20), x='abscoef', y='feature');
```



The list of features that are seleted and their magnitude of effect on weekly sales can be seen above (remember the target is scaled down)

We will set the predictors that we got from Lasso as our actual predictors and use in further models

[147]:
```python
actual_predictors=lasso_feat[lasso_feat.coef != 0.].feature.values
```

[148]:
```python
# Lets see the best alpha score
lasso_cv.alpha_

#best alpha value is 0.4538419729195478 which could be used later to run model
```

[148]: 0.4538419729195478

[149]:
```
# We will assign the best alpha score and according to that we will train and␣
↪test our model
best_lasso = Lasso(alpha=lasso_cv.alpha_)
best_lasso.fit(X_train_s, y_train)
```

[149]:
```
Lasso(alpha=0.4538419729195478)
```

[150]:
```
lasso_scores = cross_val_score(best_lasso, X_train_s, y_train, cv=10)

print (lasso_scores)
print (np.mean(lasso_scores))
```

```
[0.90136933 0.94299048 0.9231935  0.93398535 0.93401581 0.94849042
 0.93820935 0.94061142 0.93660939 0.94071723]
0.9340192276810741
```

Thats great. getting a cross validated score of .933 is good. Now lets use this to predict our last 90 days data which the model don't know about. So if this works well in this test data give a good score and residual is small or comparable to train data we can assume its not overfitting

[151]:
```
lasso_yhat=best_lasso.predict(X_test_s)
lasso_score=best_lasso.score(X_test_s, y_test)
print("R2: ",lasso_score)
lasso_adj_r2 = 1 - (len(y_test)-1)/(len(y_test)-X_test.
↪shape[1]-1)*(1-lasso_score)
print("Adjusted R2: ",lasso_adj_r2)
```

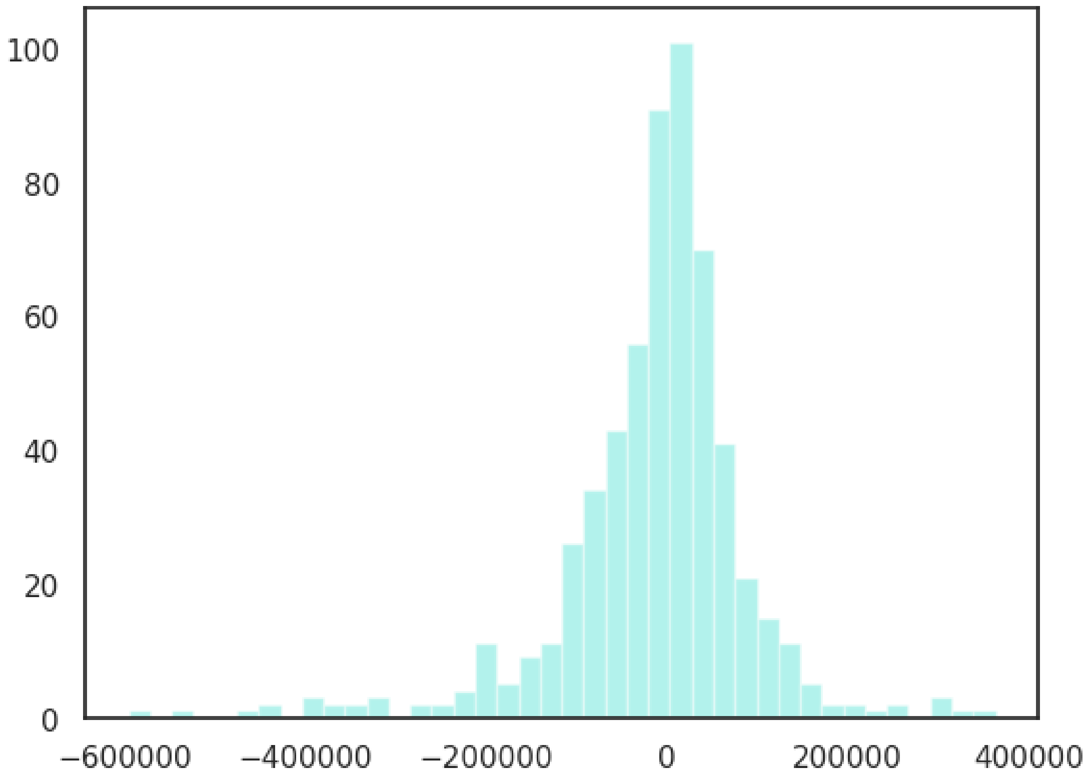
```
R2:  0.9602241138737961
Adjusted R2:  0.9559219781827266
```

[152]:
```
# converting the residuals into the actual dimenssion

train_resids = y_train*1000 - best_lasso.predict(X_train_s)*1000
test_resids = y_test*1000 - lasso_yhat*1000
lasso_residue=np.abs(test_resids).sum()
# Let me look at the actual Residuals.
print("Train Residual",np.abs(train_resids).sum())
print("Test Residual",lasso_residue)
print("Residual ratio of Test to Train",np.abs(test_resids).sum()/np.
↪abs(train_resids).sum())
# The Residual looks quite big. But this can be because our base values (␣
↪Weekly Sales) are quite big
# and in terms of millions
```

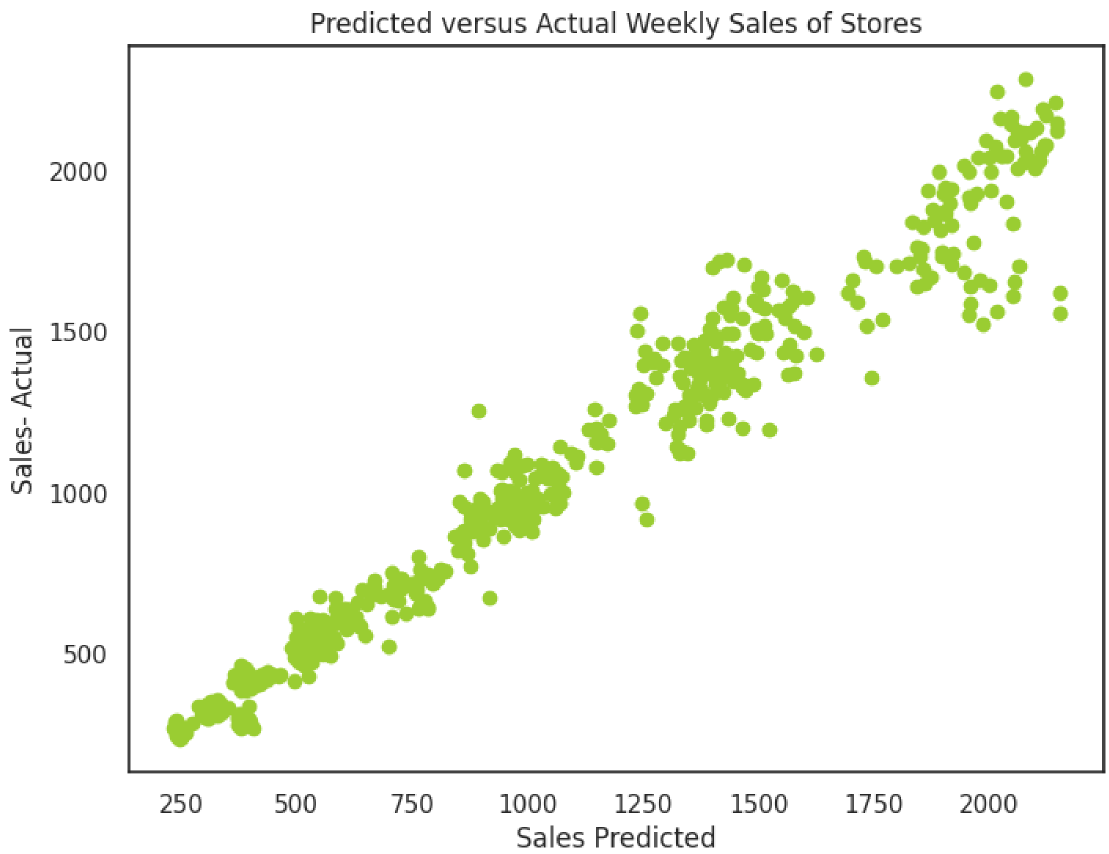
```
Train Residual 500899948.58931714
Test Residual 39751906.32790464
Residual ratio of Test to Train 0.07936097106789052
```

The residuals seems to be in same ratio,Train dataset have a higher ratio because its compartively bigger in size.

[153]:
```
sns.distplot(test_resids, kde=False, bins=40, color = 'turquoise')
plt.show()

# The residuals looks ok and almost like a normal distribution
```



[154]:
```
fig = plt.subplots(figsize=(8,6))
plt.scatter(lasso_yhat,y_test, c='yellowgreen')
plt.xlabel('Sales Predicted')
plt.ylabel('Sales- Actual')
plt.title('Predicted versus Actual Weekly Sales of Stores')
#plt.savefig('./images/Actual_vs_Predicted_Sales.png')
plt.show()
```



[155]:
```
X_train = X_train[actual_predictors]
X_test = X_test[actual_predictors]

X_train_s=ss.fit_transform(X_train)
X_test_s=ss.fit_transform(X_test)
```

Decision Trees are likely to overfit and result in over learning. So we will go for Random Forest Regressor which is a ensemble method of decision tree and check how it works.

[156]:
```
rfr=RandomForestRegressor(n_estimators=100, max_depth=None, max_features='auto')
```

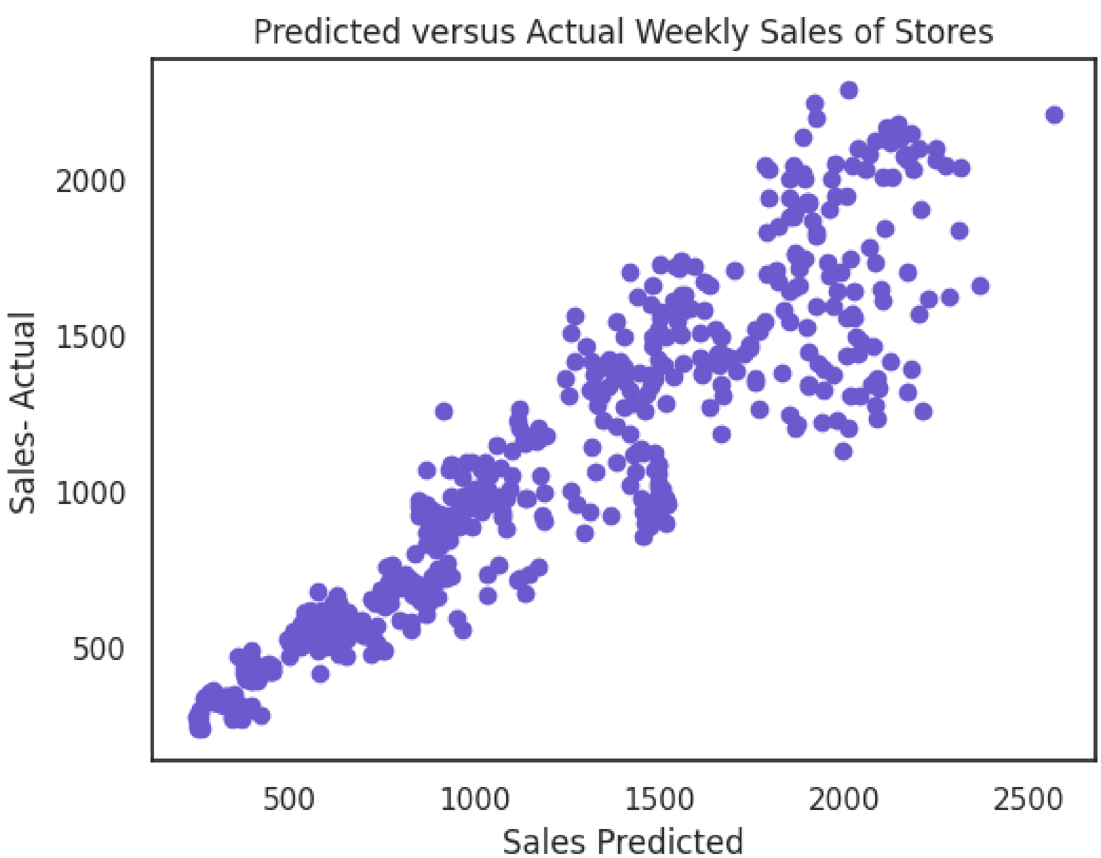
[157]:
```
# Fit and crossvalidate on train data
rfr.fit(X_train_s, y_train)
rfr_scores = cross_val_score(rfr, X_train_s, y_train, cv=10)
np.mean(rfr_scores)
```

[157]: 0.9507595120212111

[158]:
```
rfr_yhat = rfr.predict(X_test_s)
rfr_score=rfr.score(X_test_s, y_test)

print("R2: ",rfr_score)
rfr_adj_r2 = 1 - (len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)*(1-rfr_score)
print("Adjusted R2: ",rfr_adj_r2)
```

```
R2:  0.7765841728931955
Adjusted R2:  0.7542846647262262
```

[159]:
```
plt.scatter(rfr_yhat, y_test, color='slateblue')
plt.xlabel('Sales Predicted')
plt.ylabel('Sales- Actual')
plt.title('Predicted versus Actual Weekly Sales of Stores')
plt.show()
```



[160]:
```
train_resids = y_train*1000 - rfr.predict(X_train_s)*1000
test_resids = y_test*1000 - rfr_yhat*1000
rfr_residue=np.abs(test_resids).sum()
# Let me look at the actual Residuals.
```

```
print("Train Residual",np.abs(train_resids).sum())
print("Test Residual",rfr_residue)
print("Residual ratio of Test to Train",np.abs(test_resids).sum()/np.
 ↪abs(train_resids).sum())
```

```
Train Residual 149290328.54549992
Test Residual 94983880.79579999
Residual ratio of Test to Train 0.6362359954673908
```

[161]:
```
gb = GradientBoostingRegressor(n_estimators=100,max_depth=10,learning_rate=0.1)
```

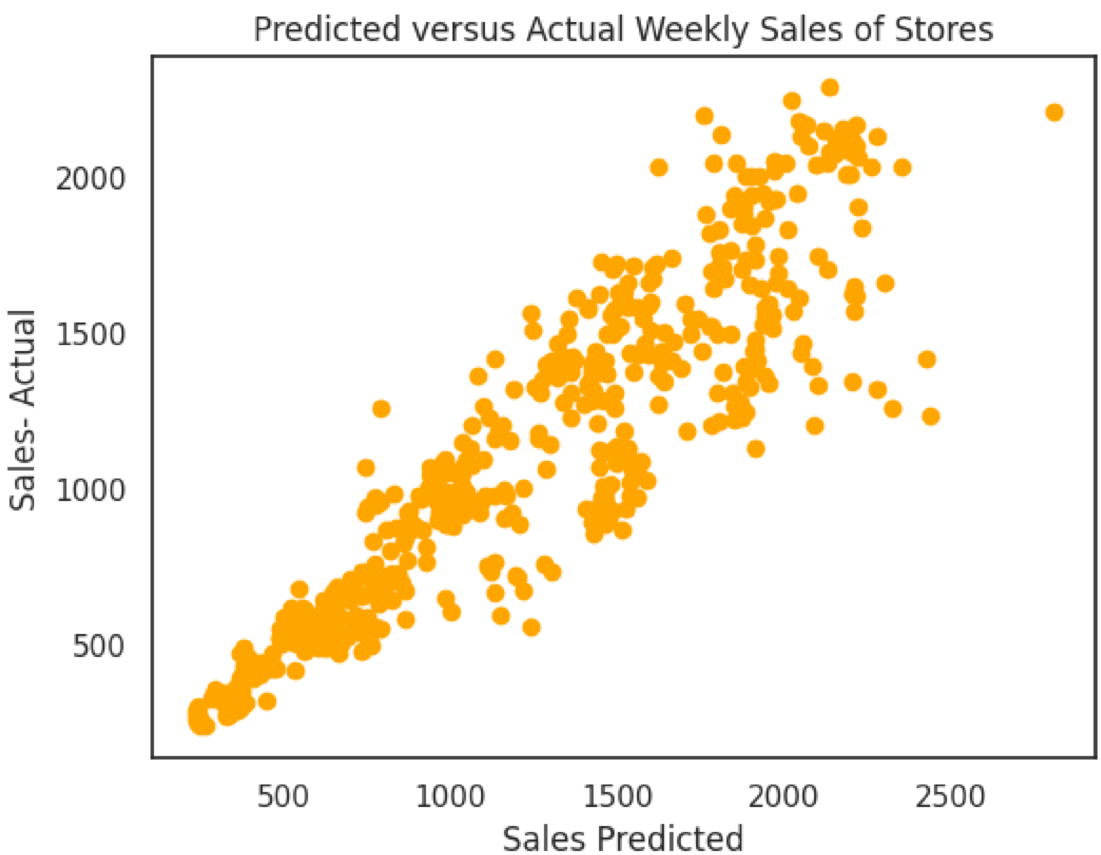
[162]:
```
gb.fit(X_train_s, y_train)
gb_scores = cross_val_score(gb, X_train_s, y_train, cv=6)
np.mean(gb_scores)
```

[162]:
```
0.9484298448650424
```

[163]:
```
gb_yhat=gb.predict(X_test_s)
gb_score=gb.score(X_test_s,y_test)

print("R2: ",gb_score)
gb_adj_r2 = 1 - (len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)*(1-gb_score)
print("Adjusted R2: ",gb_adj_r2)
```

```
R2:  0.7695219770891821
Adjusted R2:  0.746517579322189
```

[164]:
```
plt.scatter(gb_yhat, y_test, c='orange')
plt.xlabel('Sales Predicted')
plt.ylabel('Sales- Actual')
plt.title('Predicted versus Actual Weekly Sales of Stores')

plt.show()
```



[165]:
```
train_resids = y_train*1000 - gb.predict(X_train_s)*1000
test_resids = y_test*1000 - gb_yhat*1000
gb_residue=np.abs(test_resids).sum()
# Let me look at the actual Residuals.
print("Train Residual",np.abs(train_resids).sum())
print("Test Residual",gb_residue)
print("Residual ratio of Test to Train",np.abs(test_resids).sum()/np.
 ↪abs(train_resids).sum())
```

```
Train Residual 70612589.50101233
Test Residual 95307286.901544
Residual ratio of Test to Train 1.349720886530831
```

[166]:
```
svr=SVR(C=50000.0, max_iter=500)

svr.fit(X_train_s, y_train)
```

[166]: SVR(C=50000.0, max_iter=500)

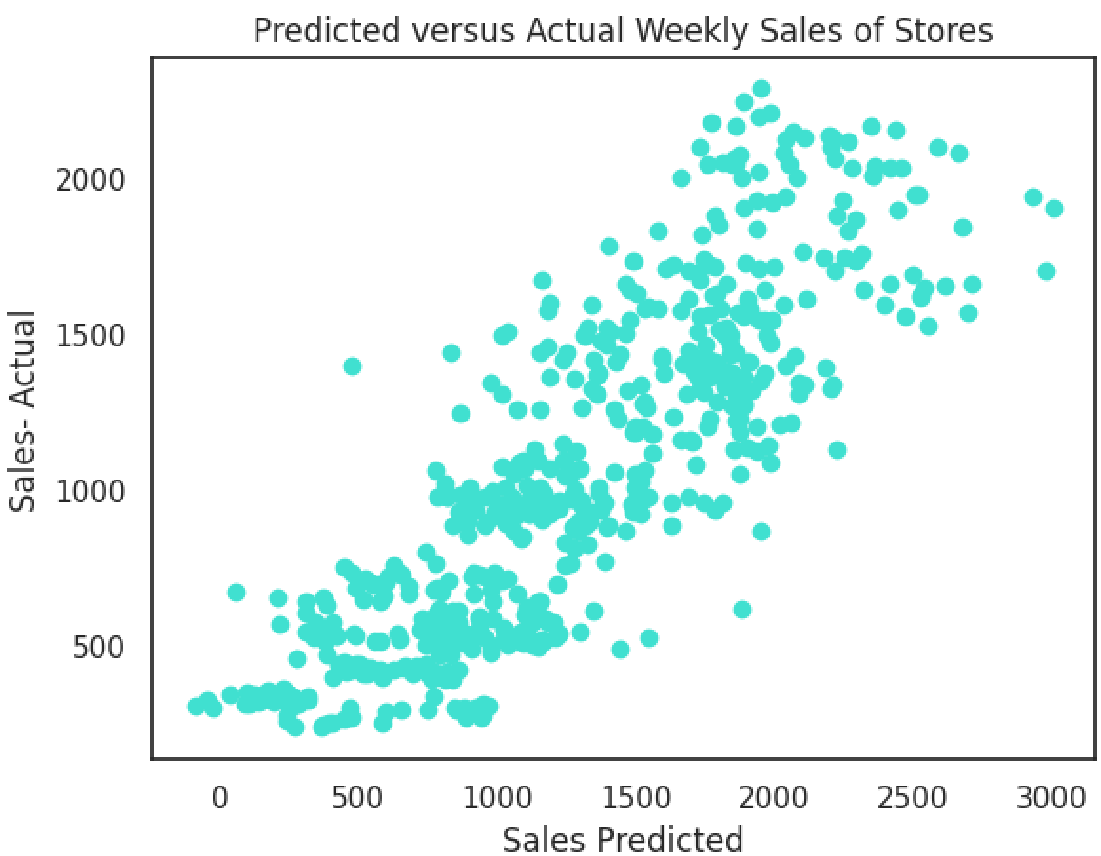
[167]:
```
svr_scores = cross_val_score(svr, X_train_s, y_train, cv=10)
np.mean(svr_scores)
```

[167]: 0.5112334434617309

[168]:
```
svr_yhat=svr.predict(X_test_s)
svr_score=svr.score(X_test_s,y_test)
print("R2: ",svr_score)
svr_adj_r2 = 1 - (len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)*(1-svr_score)
print("Adjusted R2: ",svr_adj_r2)
```

```
R2:  0.40465210542665286
Adjusted R2:  0.3452294342168838
```

[169]:
```
plt.scatter(svr_yhat, y_test, c='turquoise')
plt.xlabel('Sales Predicted')
plt.ylabel('Sales- Actual')
plt.title('Predicted versus Actual Weekly Sales of Stores')

plt.show()
```



[170]:
```
train_resids = y_train*1000 - rfr.predict(X_train_s)*1000
test_resids = y_test*1000 - svr_yhat*1000
svr_residue=np.abs(test_resids).sum()
# Let me look at the actual Residuals.
print("Train Residual", np.abs(train_resids).sum())
print("Test Residual",svr_residue)
print("Residual ratio of Test to Train",np.abs(test_resids).sum()/np.
 ↪abs(train_resids).sum())
```

```
Train Residual 149290328.54549992
Test Residual 192235311.08143502
Residual ratio of Test to Train 1.287660848189818
```

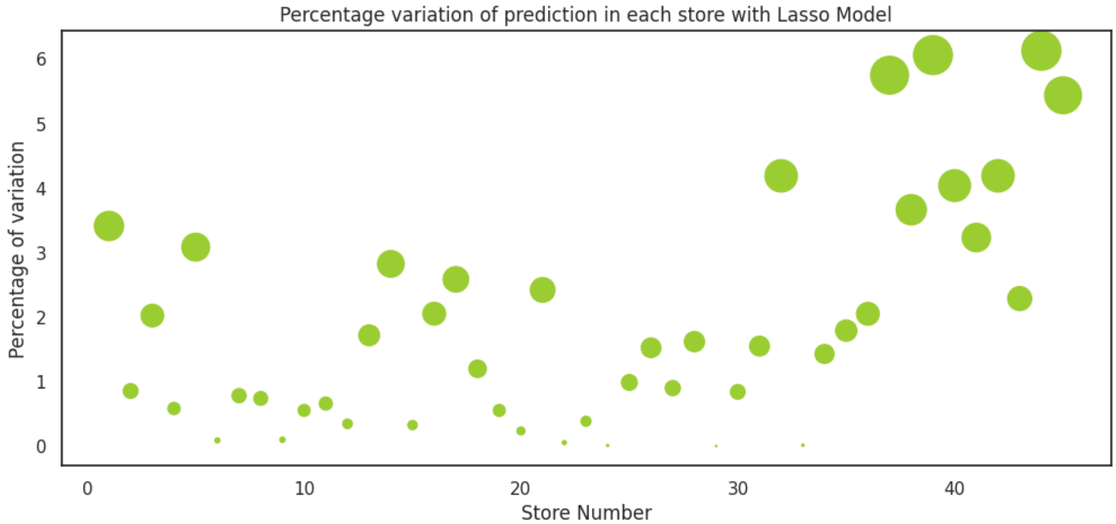
So out of all models the best comes with Lasso regression

[171]:
```
Residual_graph=pd.DataFrame()
Residual_graph["Store"]=range(1,46)
Residual_graph['actual_y']=0
Residual_graph['predicted_lasso_y']=0

count=0
for x in y_test:
    count+=1
    Residual_graph['actual_y'][count%45]+=x

count=0
for x in lasso_yhat:
    count+=1
    Residual_graph['predicted_lasso_y'][count%45]+=x

Residual_graph["actual_y"]=Residual_graph["actual_y"]/13
Residual_graph["predicted_lasso_y"]=Residual_graph["predicted_lasso_y"]/13

Residual_graph["Residual_lasso"]=np.abs(Residual_graph["actual_y"] -␣
 ↪Residual_graph["predicted_lasso_y"])
Residual_graph["Residual_lasso_percentage"]=(Residual_graph["Residual_lasso"]/
 ↪Residual_graph["actual_y"])*100
```

[172]:
```
# Setting the size of bubble according to the percentage change in prediction
s=Residual_graph.Residual_lasso_percentage.values
s=s*100
```

[173]:
```
fig = plt.subplots(figsize=(12,5))
plt.scatter(Residual_graph.Store, Residual_graph.Residual_lasso_percentage,␣
 ↪s=s, color = 'yellowgreen')
plt.xlabel('Store Number')
plt.ylabel('Percentage of variation')
plt.title('Percentage variation of prediction in each store with Lasso Model')
```

```
#plt.savefig('./images/percentage_prediction_variation.png')
plt.show()
```

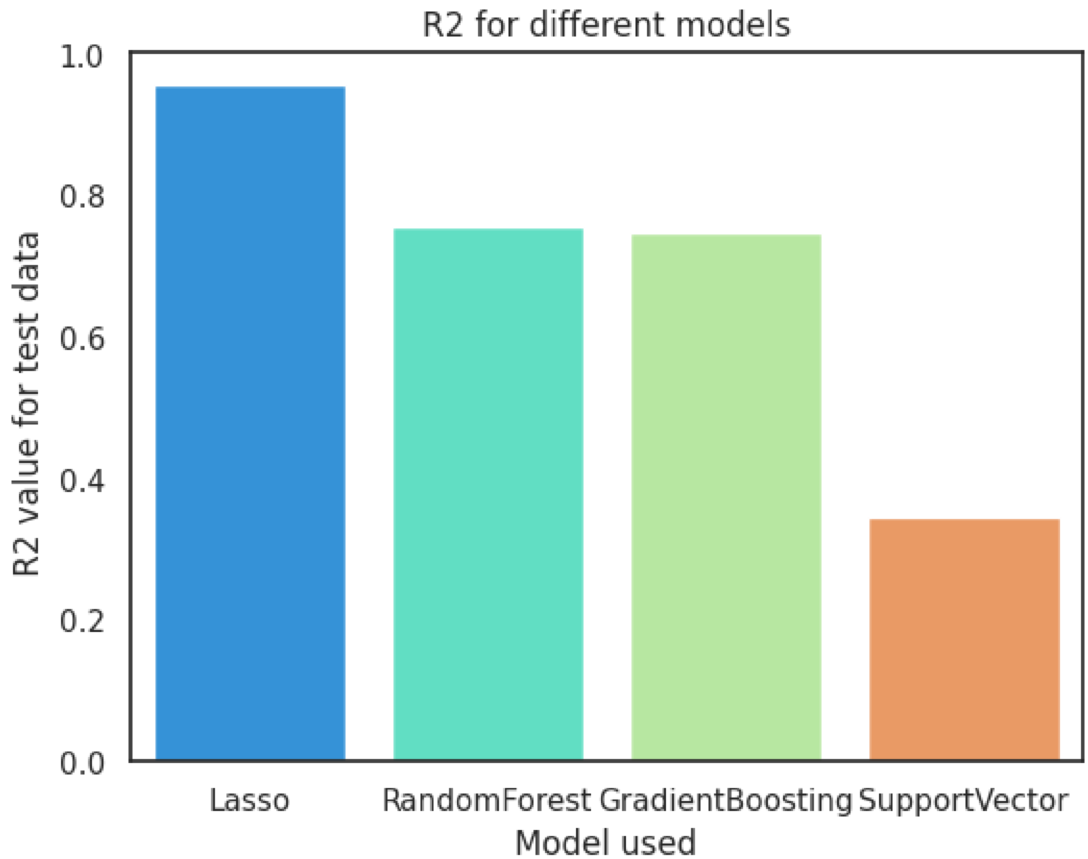


[174]:

```
# Create a dataframe to compare different models
Score=pd.DataFrame()
Score["Model_Name"]=('Lasso','RandomForest','GradientBoosting','SupportVector')
Score["Test_Score"]=(lasso_score,rfr_score,gb_score,svr_score)
Score["Adj_R2"]=(lasso_adj_r2,rfr_adj_r2,gb_adj_r2,svr_adj_r2)
Score["Test_Residual"]=(lasso_residue,rfr_residue,gb_residue,svr_residue)
```

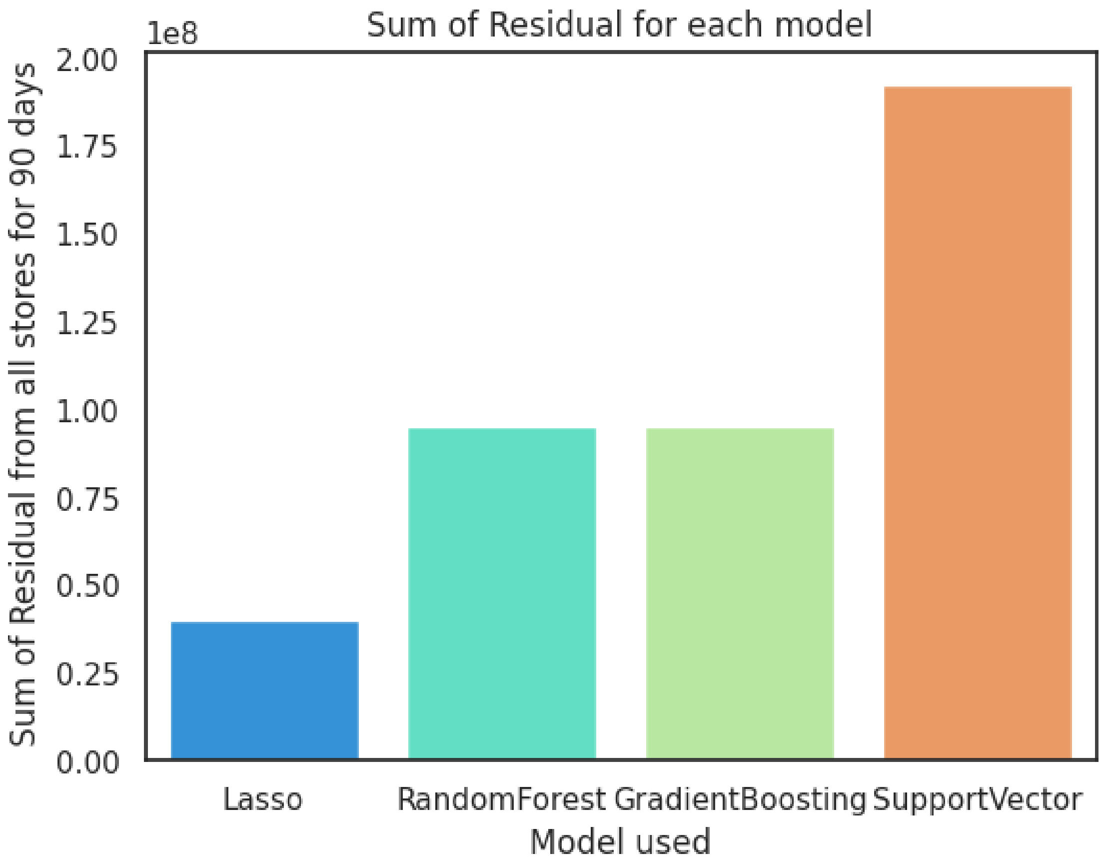
[175]:

```
# Checking how the Type of the store have effect on the sales.
sns.barplot(x="Model_Name", y="Adj_R2", data=Score,orient='v',␣
 ↪palette='rainbow')
plt.xlabel('Model used')
plt.ylabel(' R2 value for test data')
plt.title('R2 for different models')
#plt.savefig('./images/R2_for_different_models.png')
plt.show()
```



Lasso Regressor gives the best prediction and outperforms all other models as indicated by the R2 values.

[176]:
```
sns.barplot(x="Model_Name", y="Test_Residual", data=Score,orient='v',␣
 ↪palette='rainbow')
plt.xlabel('Model used')
plt.ylabel(' Sum of Residual from all stores for 90 days')
plt.title('Sum of Residual for each model')
#plt.savefig('./images/Residual_for_different_models.png')
plt.show()
```


Sum of Residual for each model
1e8
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Sum of Residual from all stores for 90 days
Lasso
RandomForest
GradientBoosting
SupportVector
Model used