In [1]:
```
#Importing the basic librarires

import os
import math
import numpy as np
import pandas as pd
import seaborn as sns
from IPython.display import display

#from brokenaxes import brokenaxes
from statsmodels.formula import api
from sklearn.feature_selection import RFE
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.decomposition import PCA
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error

import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [10,6]

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
#Importing the dataset

df = pd.read_csv('Walmart.csv')

#df.drop(['car name'], axis=1, inplace=True)
display(df.head())

original_df = df.copy(deep=True)

print('\n\033[1mInference:\033[0m The Datset consists of {} features & {} samp
```

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

Inference: The Datset consists of 8 features & 6435 samples.

In [3]:
```
# Reframing the columns

df.Date=pd.to_datetime(df.Date)

df['weekday'] = df.Date.dt.weekday
df['month'] = df.Date.dt.month
df['year'] = df.Date.dt.year

# df['Monthly_Quarter'] = df.month.map({1:'Q1',2:'Q1',3:'Q1',4:'Q2',5:'Q2',6:'
#                                        8:'Q3',9:'Q3',10:'Q4',11:'Q4',12:'Q4'}

df.drop(['Date'], axis=1, inplace=True)#,'month'

target = 'Weekly_Sales'
features = [i for i in df.columns if i not in [target]]
original_df = df.copy(deep=True)

df.head()
```

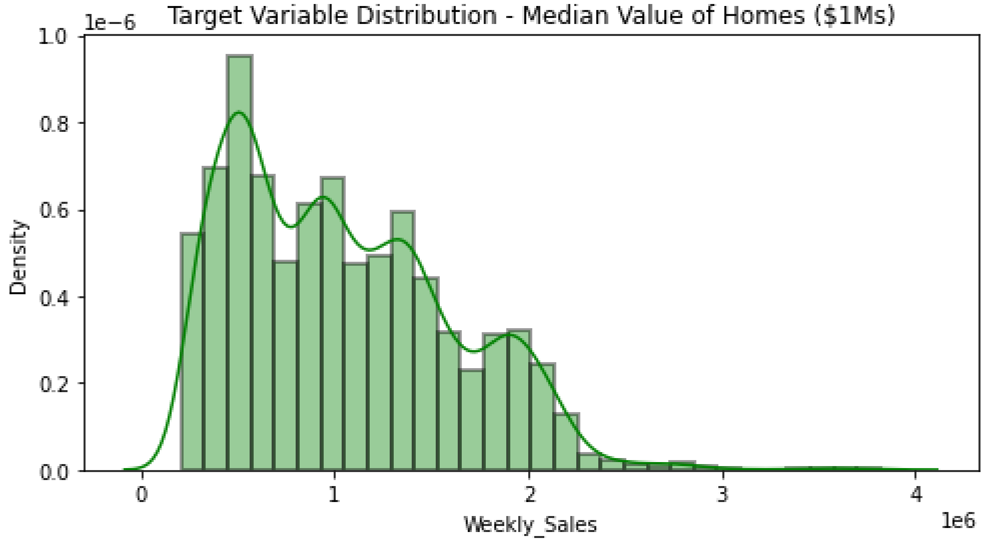
Out[3]:

	Store	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	weel
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	
4	1	1554806.68	0	46.50	2.625	211.350143	8.106	

In [4]:
```
#Checking number of unique rows in each feature

df.nunique().sort_values()
```

Out[4]:
```
Holiday_Flag       2
year               3
weekday            7
month             12
Store             45
Unemployment     349
Fuel_Price       892
CPI             2145
Temperature     3528
Weekly_Sales    6435
dtype: int64
```

In [5]:
```
#Let us first analyze the distribution of the target variable

plt.figure(figsize=[8,4])
sns.distplot(df[target], color='g',hist_kws=dict(edgecolor="black", linewidth=
plt.title('Target Variable Distribution - Median Value of Homes ($1Ms)')
plt.show()
```

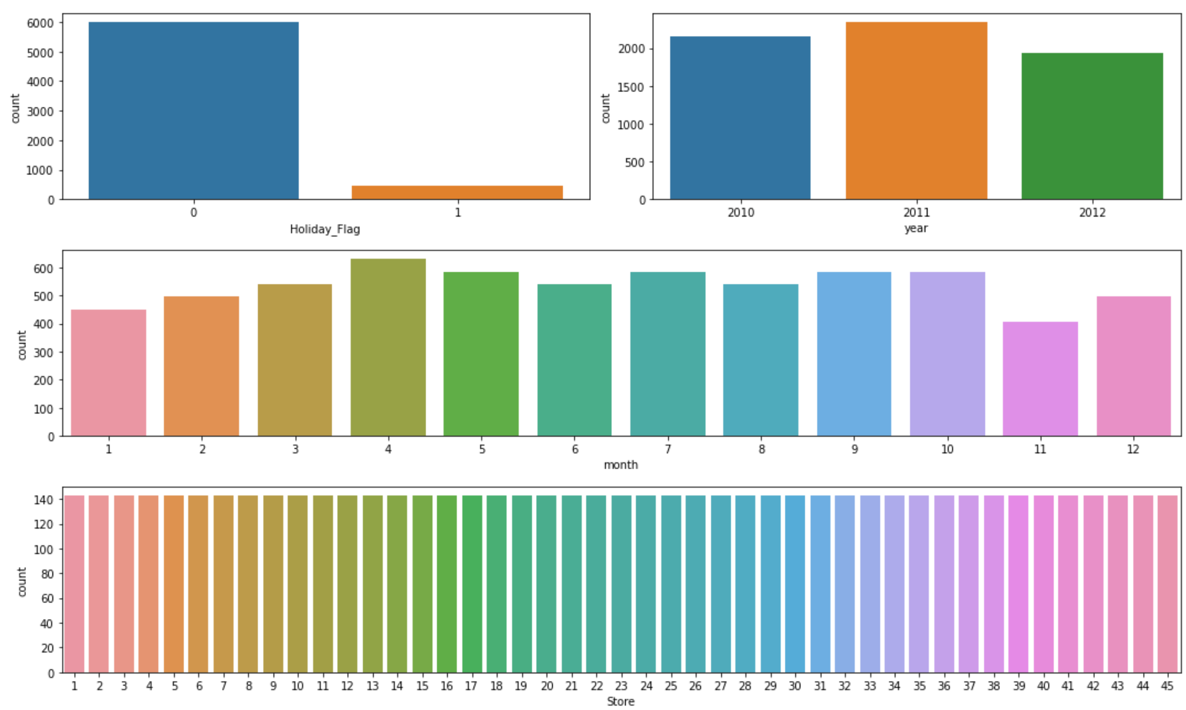


In [9]:
```
#Checking number of unique rows in each feature

nu = df[features].nunique().sort_values()
nf = []; cf = []; nnf = 0; ncf = 0; #numerical & categorical features

for i in range(df[features].shape[1]):
    if nu.values[i]<=45:cf.append(nu.index[i])
    else: nf.append(nu.index[i])

print('\n\033[1mInference:\033[0m The Datset has {} numerical & {} categorical
```

Inference: The Datset has 4 numerical & 5 categorical features.

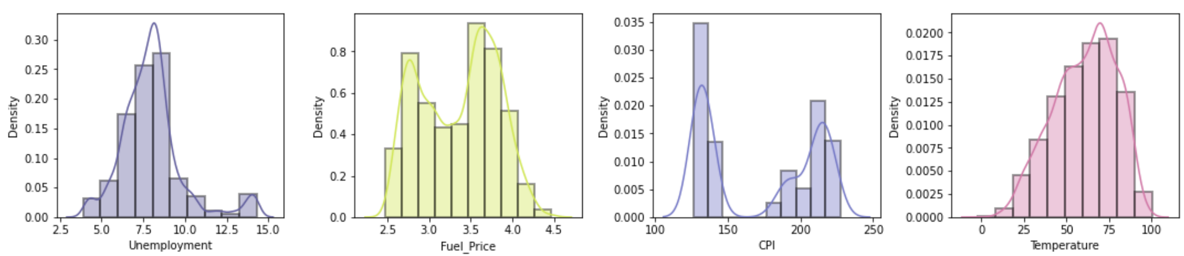
In [10]:
```
#Visualising the categorical features

print('\033[1mVisualising Categorical Features:'.center(100))

n=2
plt.figure(figsize=[15,3*math.ceil(len(cf)/n)])

for i in range(len(cf)):
    if df[cf[i]].nunique()<=8:
        plt.subplot(math.ceil(len(cf)/n),n,i+1)
        sns.countplot(df[cf[i]])
    else:
        plt.subplot(3,1,i-1)
        sns.countplot(df[cf[i]])

plt.tight_layout()
plt.show()
```

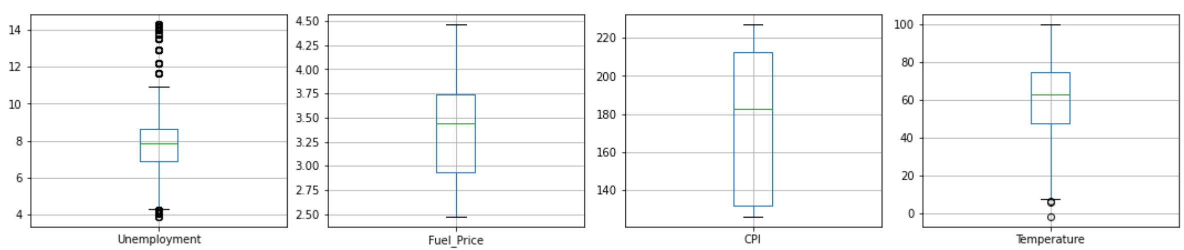
Visualising Categorical Features:



In [11]:
```python
#Visualising the numeric features

print('\033[1mNumeric Features Distribution'.center(130))

n=4

clr=['r','g','b','g','b','r']

plt.figure(figsize=[15,6*math.ceil(len(nf)/n)])
for i in range(len(nf)):
    plt.subplot(math.ceil(len(nf)/3),n,i+1)
    sns.distplot(df[nf[i]],hist_kws=dict(edgecolor="black", linewidth=2), bins
plt.tight_layout()
plt.show()

plt.figure(figsize=[15,6*math.ceil(len(nf)/n)])
for i in range(len(nf)):
    plt.subplot(math.ceil(len(nf)/3),n,i+1)
    df.boxplot(nf[i])
plt.tight_layout()
plt.show()
```

Numeric Features Distribution





In [12]:
```python
#Removal of any Duplicate rows (if any)

counter = 0
rs,cs = original_df.shape

df.drop_duplicates(inplace=True)

if df.shape==(rs,cs):
    print('\n\033[1mInference:\033[0m The dataset doesn\'t have any duplicates
else:
    print(f'\n\033[1mInference:\033[0m Number of duplicates dropped/fixed --->
```

Inference: The dataset doesn't have any duplicates

In [13]:
```
#Check for empty elements

nvc = pd.DataFrame(df.isnull().sum().sort_values(), columns=['Total Null Value
nvc['Percentage'] = round(nvc['Total Null Values']/df.shape[0],3)*100
print(nvc)
```

```
              Total Null Values  Percentage
Store                         0         0.0
Weekly_Sales                  0         0.0
Holiday_Flag                  0         0.0
Temperature                   0         0.0
Fuel_Price                    0         0.0
CPI                           0         0.0
Unemployment                  0         0.0
weekday                       0         0.0
month                         0         0.0
year                          0         0.0
```

In [14]:
```
#Converting categorical Columns to Numeric

df3 = df.copy()

ecc = nvc[nvc['Percentage']!=0].index.values
fcc = [i for i in cf if i not in ecc]
#One-Hot Binay Encoding
oh=True
dm=True
for i in fcc:
    #print(i)
    if df3[i].nunique()==2:
        if oh==True: print("\033[1mOne-Hot Encoding on features:\033[0m")
        print(i);oh=False
        df3[i]=pd.get_dummies(df3[i], drop_first=True, prefix=str(i))
    if (df3[i].nunique()>2):
        if dm==True: print("\n\033[1mDummy Encoding on features:\033[0m")
        print(i);dm=False
        df3 = pd.concat([df3.drop([i], axis=1), pd.DataFrame(pd.get_dummies(df

df3.shape
```

```
One-Hot Encoding on features:
Holiday_Flag

Dummy Encoding on features:
year
weekday
month
Store
```

Out[14]: (6435, 69)

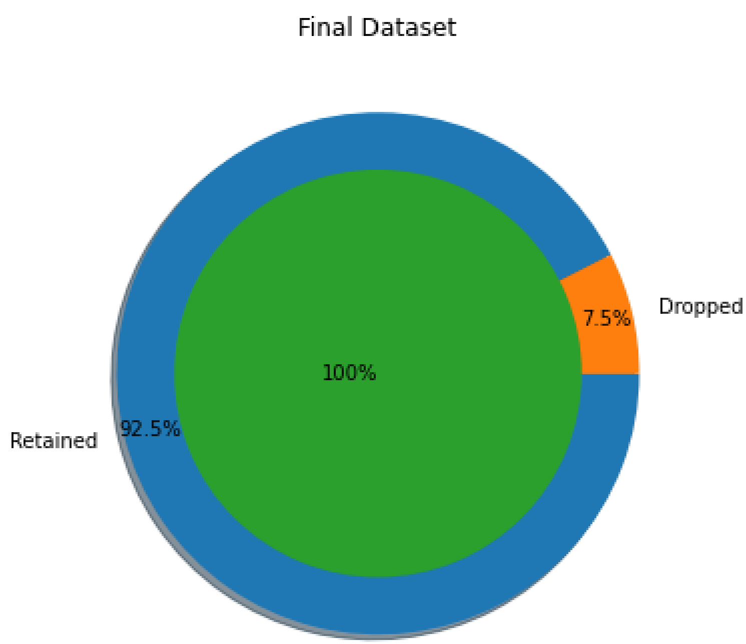
In [15]:
```
#Removal of outlier:

df1 = df3.copy()

#features1 = [i for i in features if i not in ['CHAS','RAD']]
features1 = nf

for i in features1:
    Q1 = df1[i].quantile(0.25)
    Q3 = df1[i].quantile(0.75)
    IQR = Q3 - Q1
    df1 = df1[df1[i] <= (Q3+(1.5*IQR))]
    df1 = df1[df1[i] >= (Q1-(1.5*IQR))]
    df1 = df1.reset_index(drop=True)
display(df1.head())
print('\n\033[1mInference:\033[0m\nBefore removal of outliers, The dataset had
print('After removal of outliers, The dataset now has {} samples.'.format(df1.
```

	Weekly_Sales	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	year_2011
0	1643690.90	0	42.31	2.572	211.096358	8.106	0
1	1641957.44	1	38.51	2.548	211.242170	8.106	0
2	1611968.17	0	39.93	2.514	211.289143	8.106	0
3	1409727.59	0	46.63	2.561	211.319643	8.106	0
4	1554806.68	0	46.50	2.625	211.350143	8.106	0

5 rows × 69 columns

```
Inference:
Before removal of outliers, The dataset had 6435 samples.
After removal of outliers, The dataset now has 5953 samples.
```

In [16]:
```
#Final Dataset size after performing Preprocessing

df = df1.copy()
df.columns=[i.replace('-','_') for i in df.columns]

plt.title('Final Dataset')
plt.pie([df.shape[0], original_df.shape[0]-df.shape[0]], radius = 1, labels=['
        autopct='%1.1f%%', pctdistance=0.9, explode=[0,0], shadow=True)
plt.pie([df.shape[0]], labels=['100%'], labeldistance=-0, radius=0.78)
plt.show()

print(f'\n\033[1mInference:\033[0m After the cleanup process, {original_df.sha
while retaining {round(100 - (df.shape[0]*100/(original_df.shape[0])),2)}% of
```



Inference: After the cleanup process, 482 samples were dropped, while retaini
ng 7.49% of the data.

In [20]:
```
#Splitting the data intro training & testing sets

m=[]
for i in df.columns.values:
    m.append(i.replace(' ','_'))

df.columns = m
X = df.drop([target],axis=1)
Y = df[target]
Train_X, Test_X, Train_Y, Test_Y = train_test_split(X, Y, train_size=0.8, test
Train_X.reset_index(drop=True,inplace=True)

print('Original set  ---> ',X.shape,Y.shape,'\nTraining set  ---> ',Train_X.sh
```

```
Original set  ---> (5953, 68) (5953,)
Training set  ---> (4762, 68) (4762,)
Testing set   ---> (1191, 68)  (1191,)
```

In [21]:
```
#Feature Scaling (Standardization)

std = StandardScaler()

print('\033[1mStandardardization on Training set'.center(120))
Train_X_std = std.fit_transform(Train_X)
Train_X_std = pd.DataFrame(Train_X_std, columns=X.columns)
display(Train_X_std.describe())

print('\n','\033[1mStandardardization on Testing set'.center(120))
Test_X_std = std.transform(Test_X)
Test_X_std = pd.DataFrame(Test_X_std, columns=X.columns)
display(Test_X_std.describe())
```

Standardardization on Training set

	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	year_2011
count	4.762000e+03	4.762000e+03	4.762000e+03	4.762000e+03	4.762000e+03	4.762000e+03
mean	-1.741339e-16	-1.494674e-16	-3.367039e-16	-2.799804e-16	-4.039888e-16	3.964583e-16
std	1.000105e+00	1.000105e+00	1.000105e+00	1.000105e+00	1.000105e+00	1.000105e+00
min	-2.742012e-01	-2.961575e+00	-1.871814e+00	-1.248731e+00	-2.762670e+00	-7.526270e-01
25%	-2.742012e-01	-7.314248e-01	-9.886990e-01	-1.076949e+00	-6.783836e-01	-7.526270e-01
50%	-2.742012e-01	1.062547e-01	1.663112e-01	3.842133e-01	9.596435e-02	-7.526270e-01
75%	-2.742012e-01	7.731979e-01	8.427860e-01	9.933828e-01	6.138095e-01	1.328679e+00
max	3.646958e+00	2.170008e+00	2.469806e+00	1.340791e+00	2.575491e+00	1.328679e+00

8 rows × 68 columns

Standardardization on Testing set

	Holiday_Flag	Temperature	Fuel_Price	CPI	Unemployment	year_2011	year
count	1191.000000	1191.000000	1191.000000	1191.000000	1191.000000	1191.000000	1191.0
mean	0.005646	0.044406	0.075113	0.021041	-0.050953	0.052984	0.0
std	1.009885	1.000220	0.971917	1.004644	1.010206	1.014188	1.0
min	-0.274201	-2.857425	-1.780457	-1.248731	-2.762670	-0.752627	-0.6
25%	-0.274201	-0.657516	-0.852751	-1.077025	-0.699355	-0.752627	-0.6
50%	-0.274201	0.187351	0.298996	0.393492	0.058860	-0.752627	-0.6
75%	-0.274201	0.818764	0.844961	1.019967	0.611390	1.328679	1.5
max	3.646958	2.035481	2.469806	1.345814	2.575491	1.328679	1.5

8 rows × 68 columns

In [22]:

```
#Testing a Linear Regression model with statsmodels

Train_xy = pd.concat([Train_X_std,Train_Y.reset_index(drop=True)],axis=1)
a = Train_xy.columns.values

API = api.ols(formula='{} ~ {}'.format(target,' + '.join(i for i in Train_X.cc
#print(API.conf_int())
#print(API.pvalues)
API.summary()
```

Out[22]:

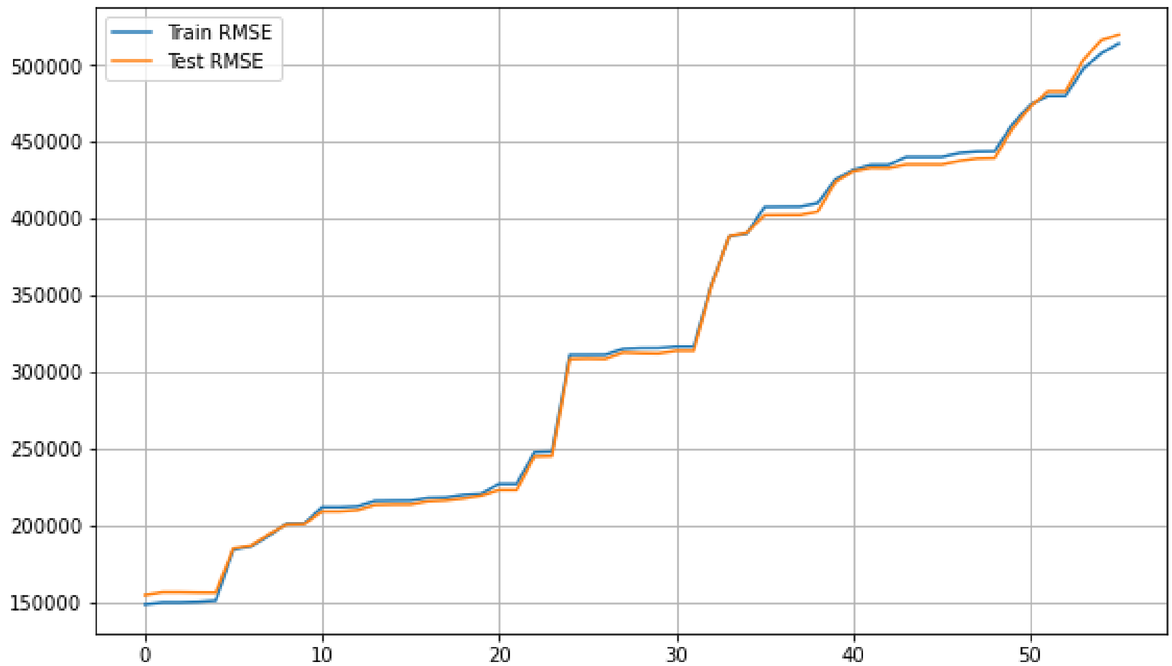
OLS Regression Results

Dep. Variable:	Weekly_Sales	**R-squared:**	0.933
Model:	OLS	**Adj. R-squared:**	0.932
Method:	Least Squares	**F-statistic:**	958.4
Date:	Mon, 15 May 2023	**Prob (F-statistic):**	0.00
Time:	19:37:22	**Log-Likelihood:**	-63430.
No. Observations:	4762	**AIC:**	1.270e+05
Df Residuals:	4693	**BIC:**	1.274e+05
Df Model:	68		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1.048e+06	2152.234	486.752	0.000	1.04e+06	1.05e+06

In [25]:
```
from sklearn.preprocessing import PolynomialFeatures
Trr=[]; Tss=[]; n=3
order=['ord-'+str(i) for i in range(2,n)]
#Trd = pd.DataFrame(np.zeros((10,n-2)), columns=order)
#Tsd = pd.DataFrame(np.zeros((10,n-2)), columns=order)

DROP=[];b=[]

for i in range(len(Train_X_std.columns)):
    vif = pd.DataFrame()
    X = Train_X_std.drop(DROP,axis=1)
    vif['Features'] = X.columns
    vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shap
    vif['VIF'] = round(vif['VIF'], 2)
    vif = vif.sort_values(by = "VIF", ascending = False)
    vif.reset_index(drop=True, inplace=True)
    if vif.loc[0][1]>1:
        DROP.append(vif.loc[0][0])
        LR = LinearRegression()
        LR.fit(Train_X_std.drop(DROP,axis=1), Train_Y)

        pred1 = LR.predict(Train_X_std.drop(DROP,axis=1))
        pred2 = LR.predict(Test_X_std.drop(DROP,axis=1))

        Trr.append(np.sqrt(mean_squared_error(Train_Y, pred1)))
        Tss.append(np.sqrt(mean_squared_error(Test_Y, pred2)))

        #Trd.loc[i,'ord-'+str(k)] = round(np.sqrt(mean_squared_error(Train_Y,
        #Tsd.loc[i,'ord-'+str(k)] = round(np.sqrt(mean_squared_error(Test_Y, p

print('Dropped Features --> ',DROP)


plt.plot(Trr, label='Train RMSE')
plt.plot(Tss, label='Test RMSE')
#plt.ylim([19.75,20.75])
plt.legend()
plt.grid()
plt.show()
```

```
Dropped Features -->  ['CPI', 'Unemployment', 'Fuel_Price', 'weekday_4', 'mon
th_7', 'Store_7', 'Temperature', 'month_12', 'Store_43', 'year_2012', 'Store_
30', 'month_2', 'month_11', 'Store_16', 'month_5', 'Store_25', 'Store_29', 'm
onth_10', 'Store_17', 'Holiday_Flag', 'Store_18', 'year_2011', 'Store_19', 'm
onth_9', 'Store_20', 'Store_8', 'Store_34', 'Store_15', 'Store_22', 'month_
6', 'Store_21', 'Store_35', 'Store_14', 'Store_13', 'Store_45', 'Store_27',
'month_3', 'weekday_1', 'Store_23', 'Store_44', 'Store_42', 'Store_11', 'week
day_5', 'Store_39', 'weekday_2', 'weekday_3', 'Store_24', 'Store_41', 'Store_
40', 'Store_10', 'Store_36', 'Store_9', 'month_4', 'Store_2', 'Store_3', 'Sto
re_6']
```


Train RMSE
Test RMSE
500000
450000
400000
350000
300000
250000
200000
150000
0
10
20
30
40
50

In [ ]: