

# Regression Model on Household Electricity Consumption

Submitted by:- Ambarish Singh



## Problem Statement

- Predict the household power consumption

## Task Performed in this Notebook:-

1. Load Data
2. Exploratory data analysis
3. Check and remove any special character
4. Handle the null Values
5. Graphical Analysis

6. Check and Handle the outliers
7. Train Test Split
8. Model building for :
  - Linear Regression
  - Ridge Regression
  - Lasso Regression
  - ElasticNet Regression
  - Support Vector Regression
  - Decision Tree Regressor
  - Random Forest Regressor
  - Bagging Regressor
9. check for all models:
  - mean\_squared\_error
  - mean\_absolute\_error
  - r2\_score
  - Adjusted r2\_score
10. Hyper-Parameter tuning using RandomSearchCV on:
  - Random Forest Regressor
  - Bagging Regressor
11. Summary
12. Store the best model in pickle file

## Description

- The dataset is collected from UCI website, provided by Senior Researchers from France.
- More than 2 million records
- Data of 47 months ranging from December 2006 to November 2010.
- Dataset has 9 attributes, out of which 3 are meter readings stating how much electricity unit appliances of various type has consumed.

In [8]:

```
1  ## comment
2  ## Observation
```

## Importing required libraries

In [48]:

```
1  # Data Analysing
2  import pandas as pd
3  import numpy as np
4
5  # Graphical analysis
6  import matplotlib.pyplot as plt
7  %matplotlib inline
8  import seaborn as sns
9  import warnings
10 warnings.filterwarnings('ignore')
11
12 # for model building
13 from sklearn.linear_model import LinearRegression,Ridge,Lasso,ElasticNe
14 from sklearn.svm import SVR
15 from sklearn.metrics import accuracy_score,r2_score,mean_squared_error,
16 from sklearn.model_selection import train_test_split
17 from sklearn.preprocessing import StandardScaler
18 from sklearn.ensemble import RandomForestRegressor
19 from sklearn.tree import DecisionTreeRegressor
20 from sklearn.ensemble import BaggingRegressor
21 from sklearn.model_selection import RandomizedSearchCV
22
23 # save the model
24 import pickle
25
```

## Load dataset

In [4]:

```
1 ## Loading Dataset
2 df = pd.read_csv(r"household_power_consumption.txt", sep=';')
3 df.head()
```

Out[4]:

	Date	Time	Global_active_power	Global_reactive_power	Voltage
0	16/12/2006	17:24:00	4.216	0.418	234.840
1	16/12/2006	17:25:00	5.360	0.436	233.630
2	16/12/2006	17:26:00	5.374	0.498	233.290
3	16/12/2006	17:27:00	5.388	0.502	233.740
4	16/12/2006	17:28:00	3.666	0.528	235.680

In [5]:

```
1 ## Checking Shape of Dataset
2 df.shape
```

Out[5]:

(2075259, 9)

## Observations

- Data is very big so we have to take small sample for model building

In [6]:

```
1 ## Creating 60,000 Sample Data from Original dataset
2 df_sample = df.sample(60000)
```

In [7]:

```
1 ## Checking Sample shape of newly created sample dataset.
2 df_sample.shape
```

Out[7]:

(60000, 9)

## Observations

- We have taken 60000 samples out of 2 million to build model

## EDA

In [9]:

```
1 ## Checking all Columns Available in a dataset
2 df_sample.columns
```

Out[9]:

```
Index(['Date', 'Time', 'Global_active_power', 'Global_reactiv
e_power',
      'Voltage', 'Global_intensity', 'Sub_metering_1', 'Sub_
metering_2',
      'Sub_metering_3'],
      dtype='object')
```

## Drop Date and time columns

In [10]:

```
1 ## Dropping Unnecessary columns from dataset
2 df_sample.drop(['Date','Time'],axis = 1, inplace = True)
```

In [11]:

```
1 ## checking Top 5 rows from dataset
2 df_sample.head()
```

Out[11]:

	Global_active_power	Global_reactive_power	Voltage	Global_intens
1773	0.914	0.206	246.160	3.8
1874584	0.346	0.000	240.160	1.4
604094	0.296	0.082	244.350	1.2
1998496	0.474	0.000	240.940	2.0
1636101	2.670	0.118	236.060	11.2

## Check any special character

In [12]:

```
1 ## Checking any special character are present in a dataset or not.
2 special_char = df_sample[df_sample['Voltage'] == "?"]
3 special_char
```

Out[12]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensi
1931833	?	?	?	
1985001	?	?	?	
1713833	?	?	?	
1987508	?	?	?	
192536	?	?	?	
...	...	...	...	...
1930378	?	?	?	
1988688	?	?	?	
1988393	?	?	?	
1619355	?	?	?	
1934148	?	?	?	

739 rows × 7 columns



## Drop these records having special character

In [13]:

```
1 print("Data before special characters", df_sample.shape)
2 df_sample.drop(special_char.index,axis= 0,inplace=True)
3 print("Data before removal of special characters", df_sample.shape)
```

Data before special characters (60000, 7)

Data before removal of special characters (59261, 7)

## Check duplicated

In [14]:

```
1 ## Checking Total Duplicate rows present in a dataset.
2 df_sample.duplicated().sum()
```

Out[14]:

276

In [15]:

```
1 ## Dropping Duplicate Rows from dataset and also checking shape of datas
2 print("Data before duplicate records", df_sample.shape)
3 df_sample.drop_duplicates(inplace=True)
4 print("Data after removal of duplicate records", df_sample.shape)
```

Data before duplicate records (59261, 7)

Data after removal of duplicate records (58985, 7)

## Check the null values

In [17]:

```
1 ## Checking Total Null Value present in a dataset.
2 df_sample.isna().sum()
```

Out[17]:

```
Global_active_power      0
Global_reactive_power    0
Voltage                  0
Global_intensity         0
Sub_metering_1           0
Sub_metering_2           0
Sub_metering_3           0
dtype: int64
```

## Observations

- No Null Value

## Convert all dtypes to float

In [18]:

```
1 ## Checking dtypes for all Columns present in a dataset
2 df_sample.dtypes
```

Out[18]:

```
Global_active_power    object
Global_reactive_power  object
Voltage                object
Global_intensity       object
Sub_metering_1         object
Sub_metering_2         object
Sub_metering_3         float64
dtype: object
```

In [19]:

```
1 ## Converting all Columns dtype to Float type
2 df_sample = df_sample.astype(float)
```

In [20]:

```
1 ## Checking dtypes after converting all columns dtype to float
2 df_sample.dtypes
```

Out[20]:

```
Global_active_power    float64
Global_reactive_power  float64
Voltage                float64
Global_intensity       float64
Sub_metering_1         float64
Sub_metering_2         float64
Sub_metering_3         float64
dtype: object
```

**Combine reading of sub\_metering\_1,sub\_metering\_2 and sub\_metering\_3**



In [21]:

```
1 ## Checking all Columns name Present in a dataset
2 df_sample.columns
```

Out[21]:

```
Index(['Global_active_power', 'Global_reactive_power', 'Voltage',
      'Global_intensity', 'Sub_metering_1', 'Sub_metering_2',
      'Sub_metering_3'],
      dtype='object')
```

In [22]:

```
1 ## Combining 3 submeter into one main meter.
2 df_sample['meter'] =df_sample['Sub_metering_1'] + df_sample['Sub_meteri
```

## Drop 3 columns

In [23]:

```
1 ## Dropping 3 Sub-meter columns from dataset.
2 df_sample.drop(['Sub_metering_1','Sub_metering_2','Sub_metering_3'],axi
```

In [24]:

```
1 ## Checking top 5 rows from dataset
2 df_sample.head()
```

Out[24]:

	Global_active_power	Global_reactive_power	Voltage	Global_intensi
1773	0.914	0.206	246.16	3
1874584	0.346	0.000	240.16	1
604094	0.296	0.082	244.35	1
1998496	0.474	0.000	240.94	2
1636101	2.670	0.118	236.06	11



In [25]:

```
1 ## Checking basic Statistics method with the help of Describe()  
2 df_sample.describe().T
```

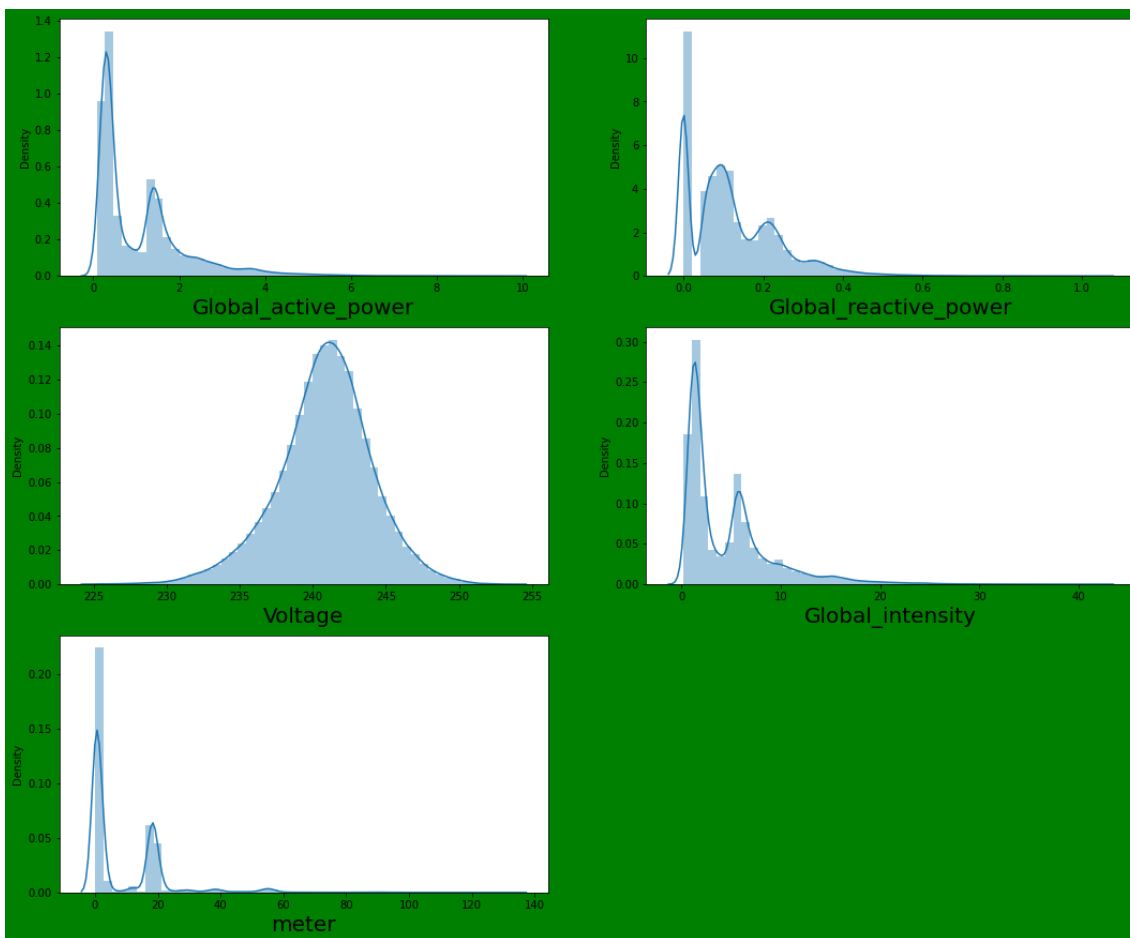
Out[25]:

	count	mean	std	min	25%	75%
<b>Global_active_power</b>	58985.0	1.104116	1.068752	0.078	0.310	0.996
<b>Global_reactive_power</b>	58985.0	0.123839	0.112869	0.000	0.048	0.289
<b>Voltage</b>	58985.0	240.829027	3.239738	225.250	239.000	241.000
<b>Global_intensity</b>	58985.0	4.680766	4.495246	0.200	1.400	10.000
<b>meter</b>	58985.0	8.983013	13.037285	0.000	0.000	10.000

## Graphical Representation

In [28]:

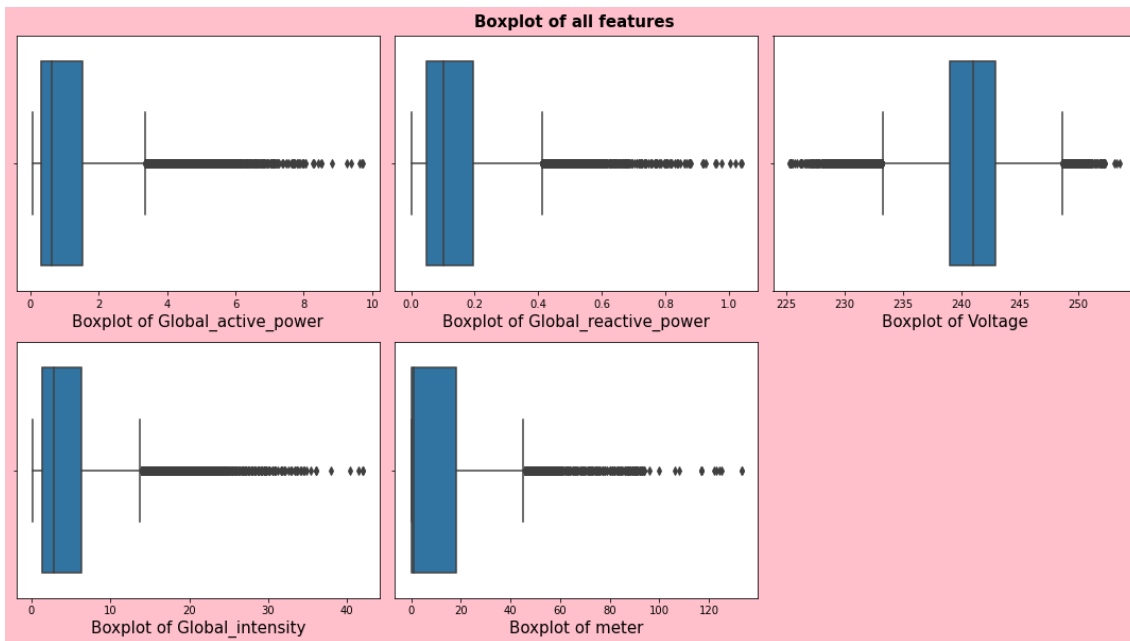
```
1  ## Let's see data distribution in each column
2
3  plt.figure(figsize=(18,15), facecolor='green')
4  plotnumber = 1
5
6  for column in df_sample.columns[:]:
7      if plotnumber<=5 :
8          ax = plt.subplot(3,2,plotnumber)
9          sns.distplot(df_sample[column])
10         plt.xlabel(column,fontsize=20)
11         plotnumber+=1
12 plt.show()
```



**Check the outliers**

In [33]:

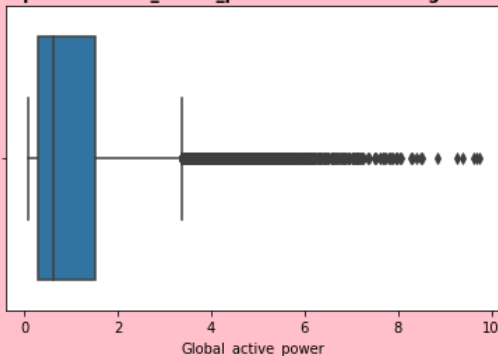
```
1  ## Checking Outlier in Dataset
2  plt.figure(figsize=(15,20),facecolor='pink')
3  plt.suptitle("Boxplot of all features", fontweight = 'bold', fontsize =
4  for i in range(0,len(df_sample.columns)):
5      plt.subplot(5,3,i+1)
6      sns.boxplot(x = df_sample.columns[i], data = df_sample)
7      plt.xlabel("Boxplot of {}".format(df_sample.columns[i]),fontsize =
8      plt.tight_layout()
```



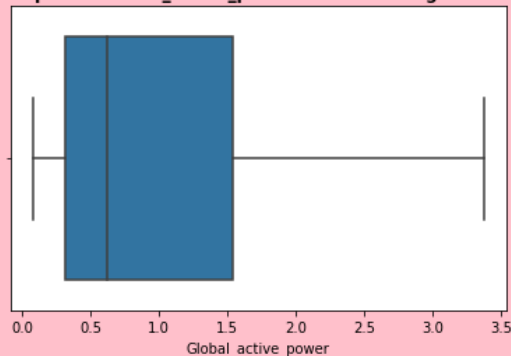
In [68]:

```
1  ## Handling the outliers
2  df1 = df_sample.copy()
3  feature_to_use = df1.columns
4
5  for i in range(len(feature_to_use)):
6      IQR = df1[feature_to_use[i]].quantile(0.75) - df1[feature_to_use[i]]
7      Lower_Limit = df1[feature_to_use[i]].quantile(0.25) - (1.5*IQR)
8      UPPER_LIMIT = df1[feature_to_use[i]].quantile(0.75) + (1.5*IQR)
9      df1[feature_to_use[i]] = np.where(df1[feature_to_use[i]] > UPPER_LIMIT,
10                                       np.where(df1[feature_to_use[i]] < Lower_Limit,
11
12
13  for fea in feature_to_use:
14      plt.figure(figsize = (14,4),facecolor='pink')
15      plt.subplot(121)
16      sns.boxplot(x = fea, data = df_sample)
17      plt.title("Boxplot of {} before handling outliers".format(fea),font
18
19      plt.subplot(122)
20      sns.boxplot(x = fea, data = df1)
21      plt.title("Boxplot of {} After handling outliers".format(fea),fontw
22      plt.show()
```

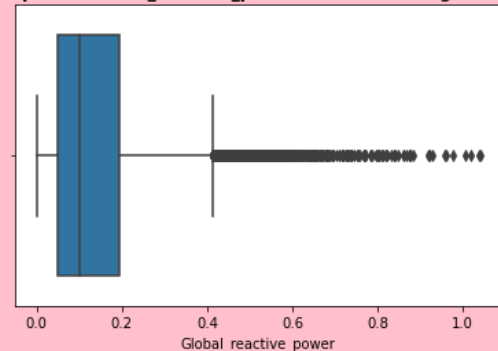
Boxplot of Global\_active\_power before handling outliers



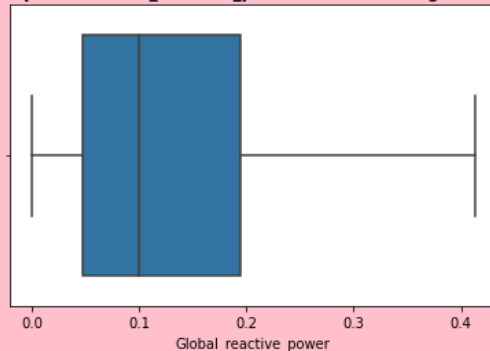
Boxplot of Global\_active\_power After handling outliers

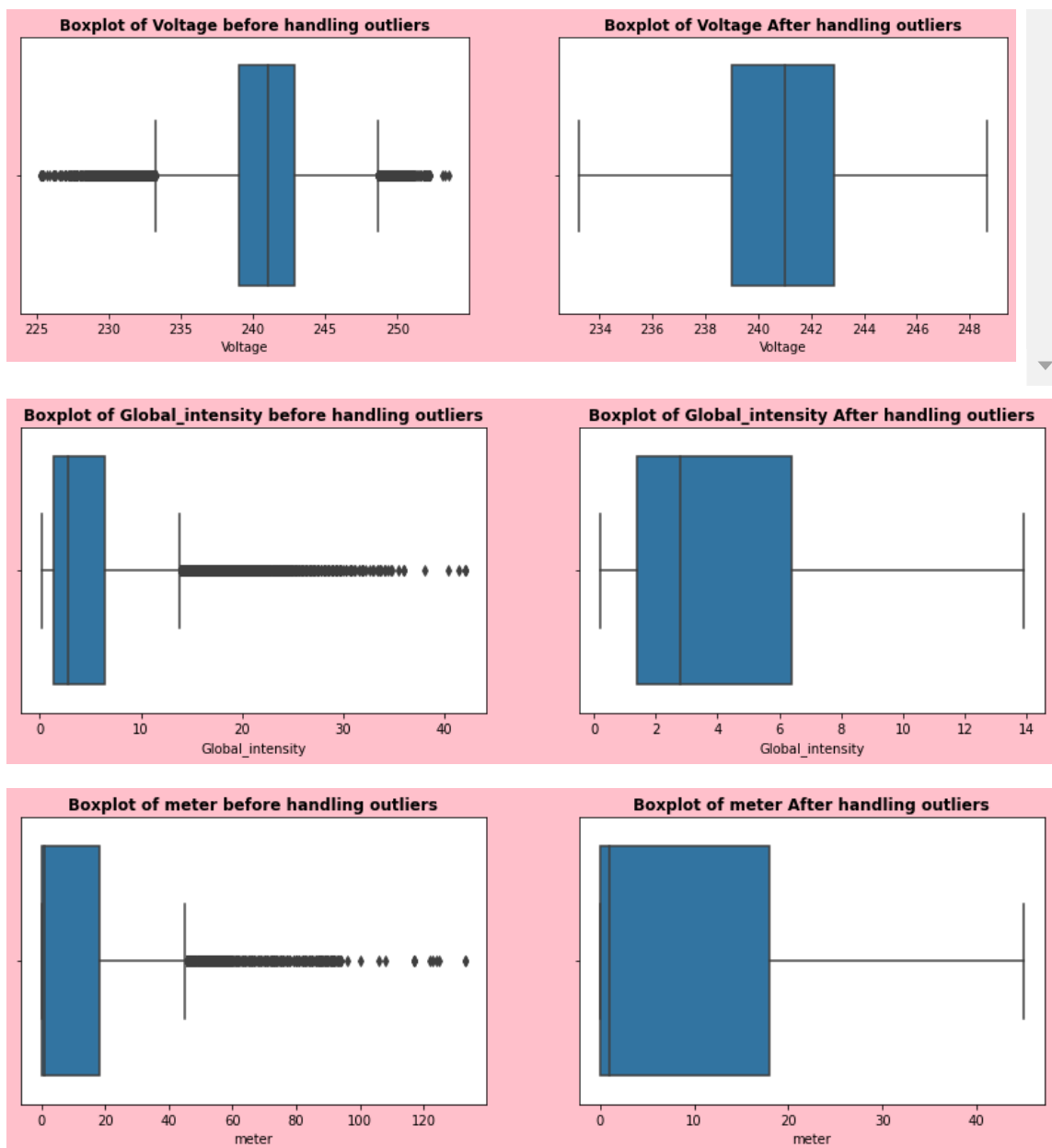


Boxplot of Global\_reactive\_power before handling outliers



Boxplot of Global\_reactive\_power After handling outliers





## Seperate Independent and Dependent Features

In [37]:

```
1 x = df1.drop('meter', axis = 1)
2 y = df1['meter']
```

In [40]:

```
1 ## Cheking Shape of x and y
2 x.shape, y.shape
```

Out[40]:

```
((58985, 4), (58985,))
```

## Train Test Split

In [41]:

```
1 x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.33,r
```

In [44]:

```
1 ## Ckecking the shape of Training Data  
2 x_train.shape,y_train.shape
```

Out[44]:

```
((39519, 4), (39519,))
```

In [45]:

```
1 ## Ckecking the shape of Testing Data  
2 x_test.shape,y_test.shape
```

Out[45]:

```
((19466, 4), (19466,))
```

## Standardization

In [46]:

```
1 scaler = StandardScaler()  
2 x_train = scaler.fit_transform(x_train)  
3 x_test = scaler.fit_transform(x_test)
```

## Model Building

In [47]:

```
1 report = []
```

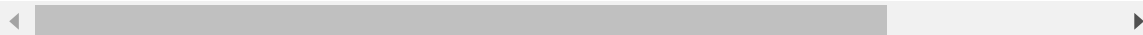
In [49]:

```
1 models = {
2     "Linear Regression" : LinearRegression(),
3     "Ridge Regression" : Ridge(),
4     "Lasso Regression" : Lasso(),
5     "ElasticNet Regression" : ElasticNet(),
6     "Support Vector Regression" : SVR(),
7     "Decision Tree Regressor" : DecisionTreeRegressor(),
8     "Random Forest Regressor" : RandomForestRegressor()
9 }
10
11 for i in range(len(list(models))):
12     model = list(models.values())[i]
13     model.fit(x_train,y_train)    # Training Model
14
15     # Prediction
16     y_train_pred = model.predict(x_train)
17     y_test_pred = model.predict(x_test)
18
19     # Training Data perfomance Matrix
20     model_train_mse = mean_squared_error(y_train,y_train_pred)    # Calc
21     model_train_mae = mean_absolute_error(y_train,y_train_pred)    # Calc
22     model_train_r2 = r2_score(y_train,y_train_pred)    # Calc
23     model_train_ad_r2 = 1 - (1-model_train_r2)*(len(y_train)-1) / (len(y_t
24
25
26     # Test Data perfomance Matrix
27     model_test_mse = mean_squared_error(y_test,y_test_pred)    # Calcula
28     model_test_mae = mean_absolute_error(y_test,y_test_pred)    # Calcula
29     model_test_r2 = r2_score(y_test,y_test_pred)    # Calcula
30     model_test_ad_r2 = 1 - (1-model_test_r2)*(len(y_test)-1) / (len(y_t
31
32
33     report.append({
34         "model" : (list(models.keys()))[i],
35         'Train Mean Squared Error ' : model_train_mse,
36         'Test Mean Squared Error' : model_test_mse,
37         'Train Mean Absolute Error' : model_train_mae,
38         'Test Mean Absolute Error' : model_test_mae,
39         'Train R Sqaure' : model_train_r2,
40         'Test R Sqaure' : model_test_r2,
41         'Train Adj R Sqaure' : model_train_ad_r2,
42         'Test Adj R Sqaure' : model_test_ad_r2
43     })
44
45 all_model = pd.DataFrame(report)
46 all_model
```

Out[49]:



	model	Train Mean Squared Error	Test Mean Squared Error	Train Mean Absolute Error	Test Mean Absolute Error	Train R Sqaure	Test R Sqaure
0	Linear Regression	40.317066	38.626226	4.227816	4.118774	0.692251	0.697878
1	Ridge Regression	40.317870	38.619047	4.226963	4.117645	0.692245	0.697934
2	Lasso Regression	42.259282	40.205873	4.489305	4.388920	0.677426	0.685522
3	ElasticNet Regression	46.791558	44.627403	5.048629	4.955296	0.642830	0.650939
4	Support Vector Regression	39.213283	39.200943	3.114148	3.059588	0.700677	0.693383
5	Decision Tree Regressor	0.583525	61.139598	0.045608	3.666033	0.995546	0.521786
6	Random Forest Regressor	5.270498	34.186604	1.164584	3.039165	0.959769	0.732603



## Hyper-Parameter Tunning on RandomSearchCV

In [50]:

```

1 Ran_param = {
2     "max_depth" : [5,8,15,None,10],
3     'max_features' : [3,'auto'],
4     'min_samples_split' : [2,8,15,20],
5     'n_estimators' : [50,100,200,500]
6 }
```

In [51]:

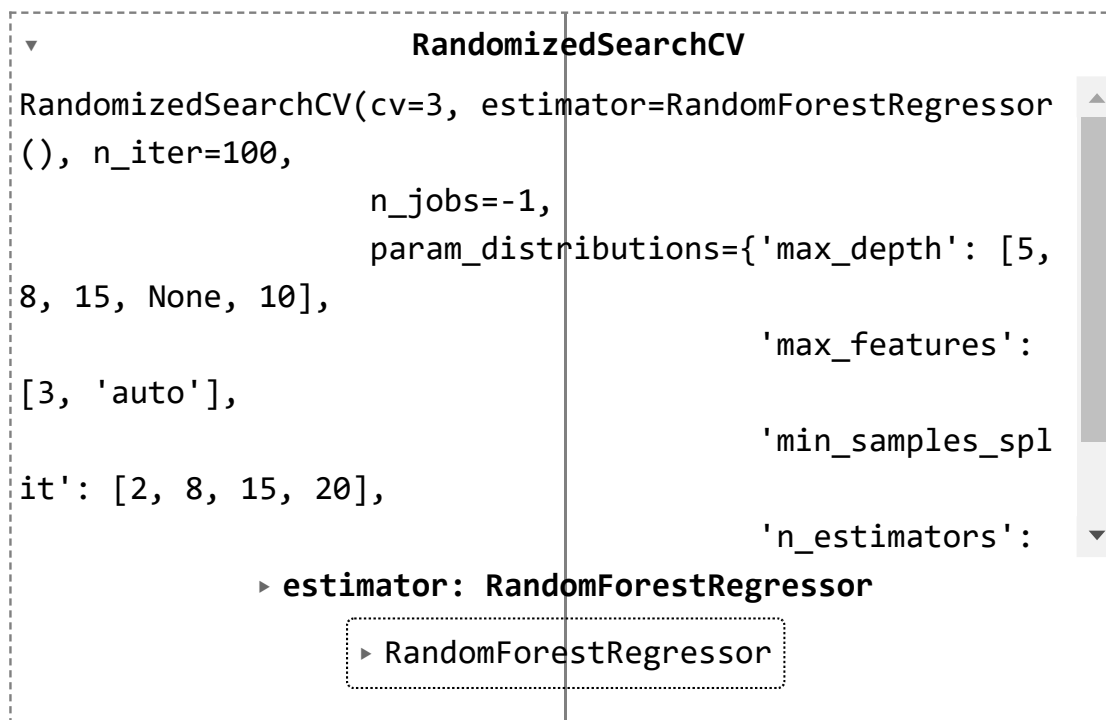
```
1 random = RandomizedSearchCV(estimator = RandomForestRegressor(),
2                             param_distributions = Ran_param,
3                             n_iter= 100,
4                             cv = 3,
5                             verbose = 2,
6                             n_jobs=-1)
```

In [52]:

```
1 random.fit(x_train,y_train)
```

Fitting 3 folds for each of 100 candidates, totalling 300 fits

Out[52]:



```
RandomizedSearchCV
RandomizedSearchCV(cv=3, estimator=RandomForestRegressor(), n_iter=100,
                  n_jobs=-1,
                  param_distributions={'max_depth': [5, 8, 15, None, 10],
                                     'max_features': [3, 'auto'],
                                     'min_samples_split': [2, 8, 15, 20],
                                     'n_estimators': ...},
                  estimator=RandomForestRegressor())
  estimator: RandomForestRegressor
    RandomForestRegressor
```

In [53]:

```
1 random.best_params_,random.best_estimator_
```

Out[53]:

```
({'n_estimators': 500,
  'min_samples_split': 20,
  'max_features': 3,
  'max_depth': 10},
 RandomForestRegressor(max_depth=10, max_features=3, min_samples_split=20,
                       n_estimators=500))
```

In [54]:

```
1 rf_best_para = RandomForestRegressor(max_depth=10, max_features=3, min_
2                                     n_estimators=500)
3
4 rf_best_para.fit(x_train,y_train)
5
6 # make predictions
7 rf_pred_train = rf_best_para.predict(x_train)
8 rf_pred_test = rf_best_para.predict(x_test)
9
10 # Training dataset performance matrix
11 rf_train_mse = mean_squared_error(y_train,rf_pred_train)    # Calculate
12 rf_train_mae = mean_absolute_error(y_train,rf_pred_train)  # Calculate
13 rf_train_r2 = r2_score(y_train,rf_pred_train)              # Calculate
14 rf_train_ad_r2 = 1 - (1-rf_train_r2)*(len(y_train)-1) / (len(y_train)-
15
16
17 # Test Data perfomance Matrix
18 rf_test_mse = mean_squared_error(y_test,rf_pred_test)    # Calculate MSE
19 rf_test_mae = mean_absolute_error(y_test,rf_pred_test)  # Calculate MAE
20 rf_test_r2 = r2_score(y_test,rf_pred_test)              # Calculate 2 s
21 rf_test_ad_r2 = 1 - (1-rf_test_r2)*(len(y_test)-1) / (len(y_test)- x_te
22
23 print("\n")
24 print("Hyperparameter tuning on random forest")
25
26 print("Model Performance For Training Data")
27 print("-Mean Squared Error : {:.4f}".format(rf_train_mse))
28 print("-Mean Absolute Error : {:.4f}".format(rf_train_mae))
29 print("-R Sqaure : {:.4f}".format(rf_train_r2))
30 print("-Adj R Sqaure : {:.4f}".format(rf_train_ad_r2))
31
32 print("-----")
33
34 print("Model Performance For Test Data")
35 print("-Mean Squared Error : {:.4f}".format(rf_test_mse))
36 print("-Mean Absolute Error : {:.4f}".format(rf_test_mae))
37 print("-R Sqaure : {:.4f}".format(rf_test_r2))
38 print("-Adj R Sqaure : {:.4f}".format(rf_test_ad_r2))
```

Hyperparameter tuning on random forest

Model Performance For Training Data

-Mean Squared Error : 27.816961

-Mean Absolute Error : 2.8878

-R Sqaure : 0.7877

-Adj R Sqaure : 0.7876

-----

-----

Model Performance For Test Data  
-Mean Squared Error : 31.002924  
-Mean Absolute Error : 2.9927  
-R Sqaure : 0.7575  
-Adj R Sqaure : 0.7575

In [55]:

```
1 rf_record = []
2 rf_record.append({
3     "model" : "Hyper-Parameter Tunning on random forest
4     "Train Mean Squared Error " : rf_train_mse,
5     "Test Mean Squared Error" : rf_test_mse,
6     "Train Mean Absolute Error" : rf_train_mae,
7     "Test Mean Absolute Error" : rf_test_mae,
8     "Train R Sqaure" : rf_train_r2,
9     "Test R Sqaure" : rf_test_r2,
10    "Train Adj R Sqaure" : rf_train_ad_r2,
11    "Test Adj R Sqaure" : rf_test_ad_r2
12 })
13
14 Hypertuned_rf = pd.DataFrame(rf_record)
15 Hypertuned_rf
```

Out[55]:

	model	Train Mean Squared Error	Test Mean Squared Error	Train Mean Absolute Error	Test Mean Absolute Error	Train R Sqaure	Test R Sqaure
0	Hyper-Parameter Tunning on random forest	27.816961	31.002924	2.887815	2.992724	0.787667	0.757505

## Bagging Regressor

In [70]:

```

1 report2 = []
2 # Bagging using DecisionTreeRegressor
3 dt_bag = BaggingRegressor(n_estimators=100)
4 #If None, then the base estimator is a DecisionTreeRegressor.
5 dt_bag.fit(x_train,y_train)
6
7 # Make predictions
8 train_pred_bag =dt_bag.predict(x_train)
9 test_pred_bag = dt_bag.predict(x_test)
10
11 # Training dataset performance matrix
12 bag_train_mse = mean_squared_error(y_train,train_pred_bag) # Calculat
13 bag_train_mae = mean_absolute_error(y_train,train_pred_bag) # Calculat
14 bag_train_r2 = r2_score(y_train,train_pred_bag) # Calculat
15 bag_train_ad_r2 = 1 - (1-bag_train_r2)*(len(y_train)-1) / (len(y_train)
16
17
18 # Test Data perfomance Matrix
19 bag_test_mse = mean_squared_error(y_test,test_pred_bag) # Calculate M
20 bag_test_mae = mean_absolute_error(y_test,test_pred_bag) # Calculate M
21 bag_test_r2 = r2_score(y_test,test_pred_bag) # Calculate 2
22 bag_test_ad_r2 = 1 - (1-bag_test_r2)*(len(y_test)-1) / (len(y_test)- x_
23
24
25
26 report2.append({
27     "model" : 'Bagging Regressor',
28     'Train Mean Squared Error ' : bag_train_mse,
29     'Test Mean Squared Error' : bag_test_mse,
30     'Train Mean Absolute Error' : bag_train_mae,
31     'Test Mean Absolute Error' : bag_test_mae,
32     'Train R Sqaure' : bag_train_r2,
33     'Test R Sqaure' : bag_test_r2,
34     'Train Adj R Sqaure' : bag_train_ad_r2,
35     'Test Adj R Sqaure' : bag_test_ad_r2
36 })
37
38 Bagging_report = pd.DataFrame(report2)
39 Bagging_report

```

Out[70]:

model	Train Mean Squared Error	Test Mean Squared Error	Train Mean Absolute Error	Test Mean Absolute Error	Train R Square	Test R Square
-------	--------------------------	-------------------------	---------------------------	--------------------------	----------------	---------------

## Hyper-parameter tuning of bagging regressor

In [58]:

```
1 bag_param = {
2     'n_estimators' : [50,100,200,500],
3     'max_samples' : range(2,5,1),
4     'max_features' : [2,3]
5 }
```

In [59]:

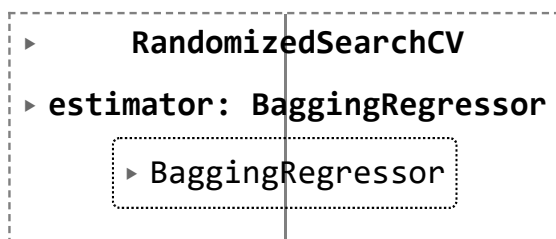
```
1 bag_ran_search = RandomizedSearchCV(estimator=dt_bag,
2                                     param_distributions= bag_param,
3                                     n_iter=100,
4                                     n_jobs=-1,
5                                     cv = 3,
6                                     verbose= 2
7                                     )
```

In [60]:

```
1 bag_ran_search.fit(x_train,y_train)
```

Fitting 3 folds for each of 24 candidates, totalling 72 fits

Out[60]:



In [61]:

```
1 bag_ran_search.best_params_,bag_ran_search.best_estimator_
```

Out[61]:

```
({'n_estimators': 50, 'max_samples': 4, 'max_features': 3},
 BaggingRegressor(max_features=3, max_samples=4, n_estimators
=50))
```

In [62]:

```
1 bag_model_hyp = BaggingRegressor(max_features=4, max_samples=10, n_est
2 bag_model_hyp.fit(x_train,y_train)
3
4 # Make predictions
5 train_pred_baghyp =bag_model_hyp.predict(x_train)
6 test_pred_baghyp = bag_model_hyp.predict(x_test)
7
8 # Training dataset performance matrix
9 baghy_train_mse = mean_squared_error(y_train,train_pred_baghyp)    # Cal
10 baghy_train_mae = mean_absolute_error(y_train,train_pred_baghyp)  # Cal
11 baghy_train_r2 = r2_score(y_train,train_pred_baghyp)              # Cal
12 baghy_train_ad_r2 = 1 - (1-baghy_train_r2)*(len(y_train)-1) / (len(y_tr
13
14 # Test Data perfomance Matrix
15 baghy_test_mse = mean_squared_error(y_test,test_pred_baghyp)    # Calcul
16 baghy_test_mae = mean_absolute_error(y_test,test_pred_baghyp)  # Calcul
17 baghy_test_r2 = r2_score(y_test,test_pred_baghyp)              # Calcul
18 baghy_test_ad_r2 = 1 - (1-baghy_test_r2)*(len(y_test)-1) / (len(y_test)
19
20 print("Hyperparameter tunning of Bagging Regressor")
21
22 print("Model Performance For Training Data")
23 print("-Mean Squared Error : {:.4f}".format(baghy_train_mse))
24 print("-Mean Absolute Error : {:.4f}".format(baghy_train_mae))
25 print("-R Sqaure : {:.4f}".format(baghy_train_r2))
26 print("-Adj R Sqaure : {:.4f}".format(baghy_train_ad_r2))
27
28 print("-----")
29
30 print("Model Performance For Test Data")
31 print("-Mean Squared Error : {:.4f}".format(baghy_test_mse))
32 print("-Mean Absolute Error : {:.4f}".format(baghy_test_mae))
33 print("-R Sqaure : {:.4f}".format(baghy_test_r2))
34 print("-Adj R Sqaure : {:.4f}".format(baghy_test_ad_r2))
35
```

Hyperparameter tunning of Bagging Regressor

Model Performance For Training Data

-Mean Squared Error : 40.121305

-Mean Absolute Error : 3.7882

-R Sqaure : 0.6937

-Adj R Sqaure : 0.6937

-----

Model Performance For Test Data

-Mean Squared Error : 38.382493

-Mean Absolute Error : 3.6898

-R Sqaure : 0.6998  
-Adj R Sqaure : 0.6997

In [63]:

```
1 bag_record = []
2 bag_record.append({
3     "model" : "Hyper-Parameter Tunning on Bagging Regre
4     "Train Mean Squared Error " : baghy_train_mse,
5     "Test Mean Squared Error" : baghy_test_mse,
6     "Train Mean Absolute Error" : baghy_train_mae,
7     "Test Mean Absolute Error" : baghy_test_mae,
8     "Train R Sqaure" : baghy_train_r2,
9     "Test R Sqaure" : baghy_test_r2,
10    "Train Adj R Sqaure" : baghy_train_ad_r2,
11    "Test Adj R Sqaure" : baghy_test_ad_r2
12    })
13
14 Hypertuned_bag = pd.DataFrame(bag_record)
15 Hypertuned_bag
```

Out[63]:

	model	Train Mean Squared Error	Test Mean Squared Error	Train Mean Absolute Error	Test Mean Absolute Error	Train R Sqaure	Test R Sqaure
0	Hyper- Parameter Tunning on Bagging Regressor	40.121305	38.382493	3.788187	3.689813	0.693746	0.699784

## Summary

- Accuracy report of all columns



In [65]:

```
1 frames3 = [all_model,Bagging_report,Hypertuned_rf,Hypertuned_bag]
2 all_records = pd.concat(frames3)
3 all_records.reset_index(inplace=True)
4 all_records.drop('index',axis = 1,inplace = True)
5 all_records.sort_values(by = 'Test R Sqaure',ascending=False)
```

Out[65]:

	model	Train Mean Squared Error	Test Mean Squared Error	Train Mean Absolute Error	Test Mean Absolute Error	Train R Sqaure	Test R Sqaure
8	Hyper- Parameter Tunning on random forest	27.816961	31.002924	2.887815	2.992724	0.787667	0.757505
7	Bagging Regressor	5.296305	34.156389	1.164451	3.039126	0.959572	0.732840
6	Random Forest Regressor	5.270498	34.186604	1.164584	3.039165	0.959769	0.732603
9	Hyper- Parameter Tunning on Bagging Regressor	40.121305	38.382493	3.788187	3.689813	0.693746	0.699784
1	Ridge Regression	40.317870	38.619047	4.226963	4.117645	0.692245	0.697934
0	Linear Regression	40.317066	38.626226	4.227816	4.118774	0.692251	0.697878
4	Support Vector Regression	39.213283	39.200943	3.114148	3.059588	0.700677	0.693383
2	Lasso Regression	42.259282	40.205873	4.489305	4.388920	0.677426	0.685522
3	ElasticNet Regression	46.791558	44.627403	5.048629	4.955296	0.642830	0.650939
5	Decision Tree Regressor	0.583525	61.139598	0.045608	3.666033	0.995546	0.521786

Store the best model in pickle file

In [66]:

```
1 import pickle
2 pickle.dump(rf_best_para, open('random_forest_hypertuned.sav', 'wb'))
```

**Thank you**