RandomForest_Boston

December 26, 2020

[44]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import␣
 ↪train_test_split,RandomizedSearchCV,GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor, export_graphviz
from sklearn import tree
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve,␣
 ↪roc_auc_score
from sklearn import datasets
```

[3]:
```
boston=datasets.load_boston()
```

[4]:
```
#Loading all features as a dataframe
features=pd.DataFrame(boston.data,columns=boston.feature_names)
```

[5]:
```
features.head()
```

[5]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	\
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	

	PTRATIO	B	LSTAT
0	15.3	396.90	4.98
1	17.8	396.90	9.14
2	17.8	392.83	4.03
3	18.7	394.63	2.94
4	18.7	396.90	5.33

[6]:
```
#No of rows and features in dataframe
features.shape
```

[6]:
```
(506, 13)
```

[7]:
```
#Loading the feature to be predicted
targets=boston.target
```

[8]:
```
targets[:5]
```

[8]:
```
array([24. , 21.6, 34.7, 33.4, 36.2])
```

[9]:
```
#No of target values
len(targets)
```

[9]:
```
506
```

[10]:
```
features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 13 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
dtypes: float64(13)
memory usage: 51.5 KB
```

[11]:
```
#No Null Values are present in the data
```

[12]:
```
features.describe()
```

[12]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	\
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	

```
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            LSTAT
count  506.000000
mean    12.653063
std      7.141062
min      1.730000
25%      6.950000
50%     11.360000
75%     16.955000
max     37.970000
```

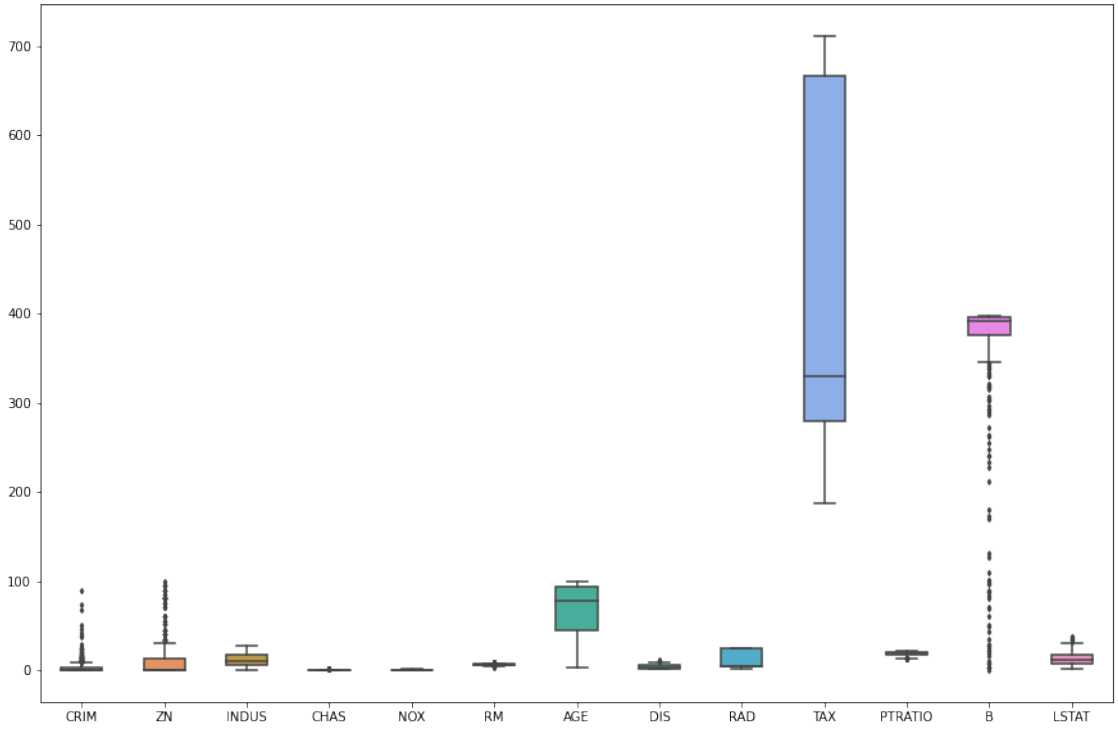
It seems from above table that for features like CRIM,ZN max values deviate highly from mean, hence seems like outliers in such features

[13]:
```
#All numerical values are present

#Checking for outliers using box plots

fig, ax = plt.subplots(figsize=(15,10))
sns.boxplot(data=features, width= 0.5,ax=ax, fliersize=3)
```

[13]: <AxesSubplot:>



[14]:
```
#Removing outliers using percentiles
q=features['CRIM'].quantile(0.98)
data_cleaned=features[features['CRIM']<q]

q=data_cleaned['ZN'].quantile(0.98)
data_cleaned=data_cleaned[data_cleaned['ZN']<q]
```

[15]:
```
data_cleaned.describe()
```

[15]:

	CRIM	ZN	INDUS	CHAS	NOX	RM
count	485.000000	485.000000	485.000000	485.000000	485.000000	485.000000
mean	2.688157	9.938144	11.165608	0.070103	0.555158	6.276812
std	4.785798	20.606864	6.803960	0.255584	0.114986	0.687136
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000
25%	0.083700	0.000000	5.190000	0.000000	0.453000	5.888000
50%	0.253870	0.000000	9.690000	0.000000	0.538000	6.208000
75%	3.163600	12.500000	18.100000	0.000000	0.624000	6.595000
max	24.801700	85.000000	27.740000	1.000000	0.871000	8.780000

	AGE	DIS	RAD	TAX	PTRATIO	B
count	485.000000	485.00000	485.000000	485.000000	485.000000	485.000000
mean	68.827216	3.77025	9.348454	405.348454	18.470309	359.160948
std	27.798743	2.04031	8.572987	166.079241	2.154734	87.270817
min	2.900000	1.12960	1.000000	188.000000	12.600000	0.320000

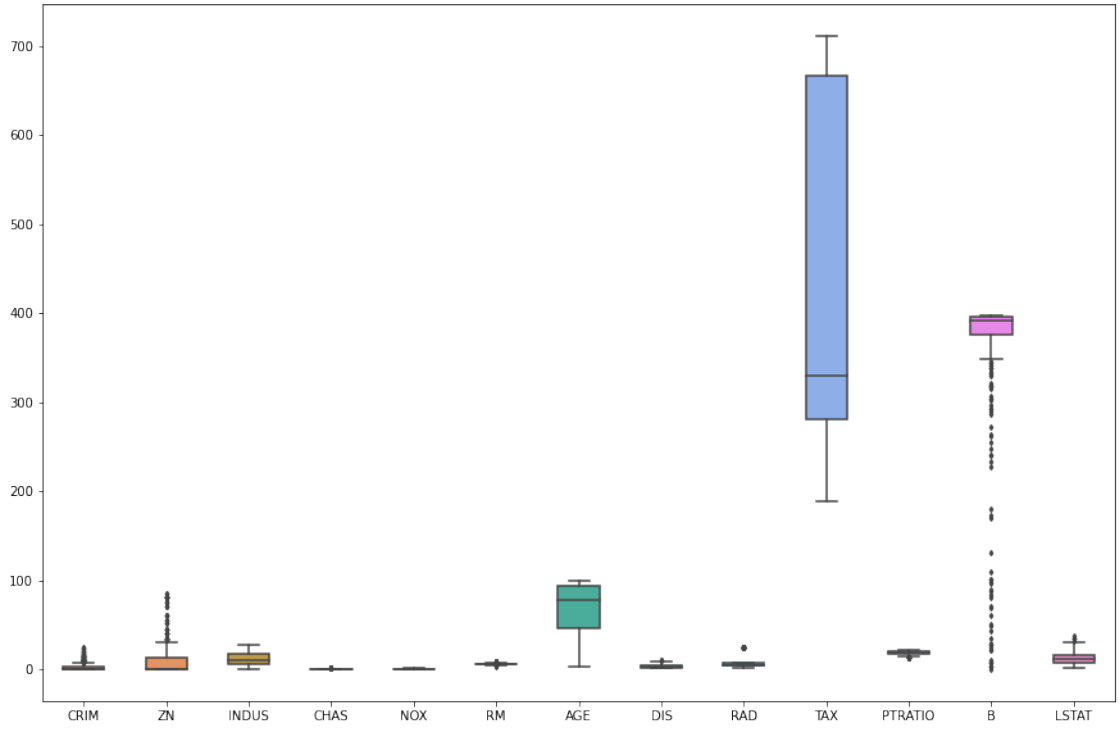
```
25%     45.800000    2.11210    4.000000  280.000000   17.400000  375.870000
50%     77.300000    3.21570    5.000000  330.000000   19.000000  391.500000
75%     93.900000    5.11670    8.000000  666.000000   20.200000  396.240000
max    100.000000   10.71030   24.000000  711.000000   22.000000  396.900000
```

```
            LSTAT
count  485.000000
mean    12.588639
std      6.931560
min      1.730000
25%      7.180000
50%     11.380000
75%     16.650000
max     37.970000
```

[16]:
```
#Checking for outliers using box plots

fig, ax = plt.subplots(figsize=(15,10))
sns.boxplot(data=data_cleaned, width= 0.5,ax=ax, fliersize=3)
```

[16]: <AxesSubplot:>



[17]:
```
#Scaling the features
scaler =StandardScaler()
X_scaled = scaler.fit_transform(features)
```

[21]:
```
x_train,x_test,y_train,y_test = train_test_split(X_scaled,targets,test_size = 0.
↪30, random_state= 355)
```

[22]:
```
x_train.shape
```

[22]:
```
(354, 13)
```

[23]:
```
x_test.shape
```

[23]:
```
(152, 13)
```

[24]:
```
y_train.shape
```

[24]:
```
(354,)
```

[25]:
```
y_test.shape
```

[25]:
```
(152,)
```

[31]:
```
#Lets check the model accuracy using decision trees
clf = DecisionTreeRegressor(min_samples_split= 2)
clf.fit(x_train,y_train)
```

[31]:
```
DecisionTreeRegressor()
```

[32]:
```
clf.score(x_test,y_test)
```

[32]:
```
0.7120613040467936
```

[35]:
```
rand_clf=RandomForestRegressor(random_state=365)
```

[36]:
```
rand_clf.fit(x_train,y_train)
```

[36]:
```
RandomForestRegressor(random_state=365)
```

[40]:
```
rand_clf.estimators_
```

[40]:
```
[DecisionTreeRegressor(max_features='auto', random_state=1896037970),
 DecisionTreeRegressor(max_features='auto', random_state=1724779188),
 DecisionTreeRegressor(max_features='auto', random_state=608178977),
 DecisionTreeRegressor(max_features='auto', random_state=1574187867),
 DecisionTreeRegressor(max_features='auto', random_state=792895916),
 DecisionTreeRegressor(max_features='auto', random_state=46676229),
```

```
DecisionTreeRegressor(max_features='auto', random_state=380371170),
DecisionTreeRegressor(max_features='auto', random_state=1121678066),
DecisionTreeRegressor(max_features='auto', random_state=188494153),
DecisionTreeRegressor(max_features='auto', random_state=1409479322),
DecisionTreeRegressor(max_features='auto', random_state=1580852536),
DecisionTreeRegressor(max_features='auto', random_state=764334372),
DecisionTreeRegressor(max_features='auto', random_state=1145148976),
DecisionTreeRegressor(max_features='auto', random_state=325727772),
DecisionTreeRegressor(max_features='auto', random_state=1310018403),
DecisionTreeRegressor(max_features='auto', random_state=40830815),
DecisionTreeRegressor(max_features='auto', random_state=1898744798),
DecisionTreeRegressor(max_features='auto', random_state=1054841347),
DecisionTreeRegressor(max_features='auto', random_state=474350503),
DecisionTreeRegressor(max_features='auto', random_state=1143524991),
DecisionTreeRegressor(max_features='auto', random_state=1882801063),
DecisionTreeRegressor(max_features='auto', random_state=1980187262),
DecisionTreeRegressor(max_features='auto', random_state=1084976586),
DecisionTreeRegressor(max_features='auto', random_state=570517931),
DecisionTreeRegressor(max_features='auto', random_state=1252525196),
DecisionTreeRegressor(max_features='auto', random_state=703610615),
DecisionTreeRegressor(max_features='auto', random_state=1656666573),
DecisionTreeRegressor(max_features='auto', random_state=153552746),
DecisionTreeRegressor(max_features='auto', random_state=2133272715),
DecisionTreeRegressor(max_features='auto', random_state=223070107),
DecisionTreeRegressor(max_features='auto', random_state=2089795264),
DecisionTreeRegressor(max_features='auto', random_state=673318104),
DecisionTreeRegressor(max_features='auto', random_state=1500705372),
DecisionTreeRegressor(max_features='auto', random_state=1632846692),
DecisionTreeRegressor(max_features='auto', random_state=650312894),
DecisionTreeRegressor(max_features='auto', random_state=909065971),
DecisionTreeRegressor(max_features='auto', random_state=1683402162),
DecisionTreeRegressor(max_features='auto', random_state=1643930175),
DecisionTreeRegressor(max_features='auto', random_state=1174856262),
DecisionTreeRegressor(max_features='auto', random_state=1657369362),
DecisionTreeRegressor(max_features='auto', random_state=1203509615),
DecisionTreeRegressor(max_features='auto', random_state=1301765435),
DecisionTreeRegressor(max_features='auto', random_state=2100240486),
DecisionTreeRegressor(max_features='auto', random_state=1180514799),
DecisionTreeRegressor(max_features='auto', random_state=141144828),
DecisionTreeRegressor(max_features='auto', random_state=1732008653),
DecisionTreeRegressor(max_features='auto', random_state=815421416),
DecisionTreeRegressor(max_features='auto', random_state=1908934553),
DecisionTreeRegressor(max_features='auto', random_state=1969465270),
DecisionTreeRegressor(max_features='auto', random_state=1436986589),
DecisionTreeRegressor(max_features='auto', random_state=1643524654),
DecisionTreeRegressor(max_features='auto', random_state=1983775377),
DecisionTreeRegressor(max_features='auto', random_state=371513279),
```

```
DecisionTreeRegressor(max_features='auto', random_state=2029662525),
DecisionTreeRegressor(max_features='auto', random_state=1046295062),
DecisionTreeRegressor(max_features='auto', random_state=1065024732),
DecisionTreeRegressor(max_features='auto', random_state=1201414421),
DecisionTreeRegressor(max_features='auto', random_state=2044133863),
DecisionTreeRegressor(max_features='auto', random_state=926261975),
DecisionTreeRegressor(max_features='auto', random_state=754455846),
DecisionTreeRegressor(max_features='auto', random_state=8333147),
DecisionTreeRegressor(max_features='auto', random_state=262737036),
DecisionTreeRegressor(max_features='auto', random_state=848289493),
DecisionTreeRegressor(max_features='auto', random_state=2093139929),
DecisionTreeRegressor(max_features='auto', random_state=400607875),
DecisionTreeRegressor(max_features='auto', random_state=1671343631),
DecisionTreeRegressor(max_features='auto', random_state=1107755368),
DecisionTreeRegressor(max_features='auto', random_state=1967709626),
DecisionTreeRegressor(max_features='auto', random_state=1487541098),
DecisionTreeRegressor(max_features='auto', random_state=346582503),
DecisionTreeRegressor(max_features='auto', random_state=1214838233),
DecisionTreeRegressor(max_features='auto', random_state=130752329),
DecisionTreeRegressor(max_features='auto', random_state=42168976),
DecisionTreeRegressor(max_features='auto', random_state=143829505),
DecisionTreeRegressor(max_features='auto', random_state=741159663),
DecisionTreeRegressor(max_features='auto', random_state=497373012),
DecisionTreeRegressor(max_features='auto', random_state=1024975056),
DecisionTreeRegressor(max_features='auto', random_state=2111545663),
DecisionTreeRegressor(max_features='auto', random_state=1228719626),
DecisionTreeRegressor(max_features='auto', random_state=1299339897),
DecisionTreeRegressor(max_features='auto', random_state=1314932038),
DecisionTreeRegressor(max_features='auto', random_state=133966629),
DecisionTreeRegressor(max_features='auto', random_state=965148982),
DecisionTreeRegressor(max_features='auto', random_state=985839273),
DecisionTreeRegressor(max_features='auto', random_state=636744453),
DecisionTreeRegressor(max_features='auto', random_state=1663687859),
DecisionTreeRegressor(max_features='auto', random_state=517305378),
DecisionTreeRegressor(max_features='auto', random_state=610654365),
DecisionTreeRegressor(max_features='auto', random_state=1381747885),
DecisionTreeRegressor(max_features='auto', random_state=1889745431),
DecisionTreeRegressor(max_features='auto', random_state=467705161),
DecisionTreeRegressor(max_features='auto', random_state=1165602546),
DecisionTreeRegressor(max_features='auto', random_state=277189798),
DecisionTreeRegressor(max_features='auto', random_state=1078312182),
DecisionTreeRegressor(max_features='auto', random_state=1789967955),
DecisionTreeRegressor(max_features='auto', random_state=1022176734),
DecisionTreeRegressor(max_features='auto', random_state=403736999),
DecisionTreeRegressor(max_features='auto', random_state=1987419159),
DecisionTreeRegressor(max_features='auto', random_state=1390446015),
DecisionTreeRegressor(max_features='auto', random_state=787765825)]
```

[41]:
```
rand_clf.feature_importances_
```

[41]:
```
array([0.0335399 , 0.00185688, 0.01078536, 0.00126716, 0.02083828,
       0.41272916, 0.01461241, 0.07144171, 0.00486142, 0.01634288,
       0.01291486, 0.01091772, 0.38789228])
```

[42]:
```
rand_clf.score(x_test,y_test)
```

[42]:
```
0.8831304179605852
```

[43]:
```
rand_clf.score(x_train,y_train)
```

[43]:
```
0.9772169063907366
```

[45]:
```
#The above value is obtained using default values
```

Now finding the best hyperparameters values using RandomSearchCV

[62]:
```
grid_param = {
    "n_estimators" : [90,100,115,130],
    'criterion': ['mse'],
    'max_depth' : range(2,20,1),
    'min_samples_leaf' : range(1,10,1),
    'min_samples_split': range(2,10,1),
    'max_features' : ['auto','log2']
}
```

[63]:
```
rand_search = GridSearchCV(estimator=rand_clf,param_grid=grid_param,cv=5,n_jobs␣
↪=-1,verbose = 3)
```

[64]:
```
rand_search.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 10368 candidates, totalling 51840 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:    9.5s
[Parallel(n_jobs=-1)]: Done 112 tasks      | elapsed:   17.3s
[Parallel(n_jobs=-1)]: Done 272 tasks      | elapsed:   28.5s
[Parallel(n_jobs=-1)]: Done 496 tasks      | elapsed:   46.1s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done 1136 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done 1552 tasks      | elapsed:  2.3min
[Parallel(n_jobs=-1)]: Done 2032 tasks      | elapsed:  2.9min
[Parallel(n_jobs=-1)]: Done 2576 tasks      | elapsed:  3.6min
[Parallel(n_jobs=-1)]: Done 3184 tasks      | elapsed:  4.4min
[Parallel(n_jobs=-1)]: Done 3856 tasks      | elapsed:  5.4min
[Parallel(n_jobs=-1)]: Done 4592 tasks      | elapsed:  6.2min
[Parallel(n_jobs=-1)]: Done 5392 tasks      | elapsed:  7.0min
```

```
[Parallel(n_jobs=-1)]: Done 6256 tasks      | elapsed:  8.1min
[Parallel(n_jobs=-1)]: Done 7184 tasks      | elapsed:  9.5min
[Parallel(n_jobs=-1)]: Done 8176 tasks      | elapsed: 10.6min
[Parallel(n_jobs=-1)]: Done 9232 tasks      | elapsed: 12.3min
[Parallel(n_jobs=-1)]: Done 10352 tasks      | elapsed: 14.0min
[Parallel(n_jobs=-1)]: Done 11536 tasks      | elapsed: 15.6min
[Parallel(n_jobs=-1)]: Done 12784 tasks      | elapsed: 17.7min
[Parallel(n_jobs=-1)]: Done 14096 tasks      | elapsed: 19.6min
[Parallel(n_jobs=-1)]: Done 15472 tasks      | elapsed: 21.8min
[Parallel(n_jobs=-1)]: Done 16912 tasks      | elapsed: 24.0min
[Parallel(n_jobs=-1)]: Done 18416 tasks      | elapsed: 26.6min
[Parallel(n_jobs=-1)]: Done 19984 tasks      | elapsed: 28.8min
[Parallel(n_jobs=-1)]: Done 21616 tasks      | elapsed: 31.7min
[Parallel(n_jobs=-1)]: Done 23312 tasks      | elapsed: 34.1min
[Parallel(n_jobs=-1)]: Done 25072 tasks      | elapsed: 36.2min
[Parallel(n_jobs=-1)]: Done 26896 tasks      | elapsed: 38.7min
[Parallel(n_jobs=-1)]: Done 28784 tasks      | elapsed: 41.8min
[Parallel(n_jobs=-1)]: Done 30736 tasks      | elapsed: 46.1min
[Parallel(n_jobs=-1)]: Done 32752 tasks      | elapsed: 49.5min
[Parallel(n_jobs=-1)]: Done 34832 tasks      | elapsed: 52.9min
[Parallel(n_jobs=-1)]: Done 36976 tasks      | elapsed: 56.8min
[Parallel(n_jobs=-1)]: Done 39184 tasks      | elapsed: 60.7min
[Parallel(n_jobs=-1)]: Done 41456 tasks      | elapsed: 64.7min
[Parallel(n_jobs=-1)]: Done 43792 tasks      | elapsed: 68.2min
[Parallel(n_jobs=-1)]: Done 46192 tasks      | elapsed: 71.8min
[Parallel(n_jobs=-1)]: Done 48656 tasks      | elapsed: 75.6min
[Parallel(n_jobs=-1)]: Done 51184 tasks      | elapsed: 79.7min
[Parallel(n_jobs=-1)]: Done 51840 out of 51840 | elapsed: 80.7min finished
```

[64]:
```
GridSearchCV(cv=5,
             estimator=RandomForestRegressor(max_depth=11, n_estimators=90,
                                             random_state=365),
             n_jobs=-1,
             param_grid={'criterion': ['mse'], 'max_depth': range(2, 20),
                         'max_features': ['auto', 'log2'],
                         'min_samples_leaf': range(1, 10),
                         'min_samples_split': range(2, 10),
                         'n_estimators': [90, 100, 115, 130]},
             verbose=3)
```

[65]:
```
rand_search.best_params_
```

[65]:
```
{'criterion': 'mse',
 'max_depth': 13,
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'min_samples_split': 2,
```

```
 'n_estimators': 90}
```

[66]:
```
rand_clf=RandomForestRegressor(max_depth=13,
criterion='mse',
max_features = 'auto',
 min_samples_leaf = 1,
 min_samples_split= 2,
 n_estimators = 90,random_state=365)
```

[67]:
```
rand_clf.fit(x_train,y_train)
```

[67]:
```
RandomForestRegressor(max_depth=13, n_estimators=90, random_state=365)
```

[68]:
```
rand_clf.score(x_test,y_test)
```

[68]:
```
0.8839001823438692
```