Assignment 24 Data Science Masters

February 18, 2019

0.1 Session 24 -Assignment - Machine Learning 5

0.1.1 House Pricing from Boston Data Set.

Load Libraries

In [24]:
```
# Core Libraries to load (for data manipulation and analysis)
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Import stats module from scipy for Statistical analysis
import scipy.stats as stats

# Core Libraries to load - Machine Learning
import sklearn

## Import LinearRegression Module - Modelling
from sklearn.linear_model import LinearRegression

## Import RandomForestRegressor Module - Modelling
from sklearn.ensemble import RandomForestRegressor

## Import StandardScaler - Model Data Preprocessing
from sklearn.preprocessing import StandardScaler

## Import train_test_split Module
from sklearn.model_selection import train_test_split, GridSearchCV, ShuffleSplit, cros

## Importing mean_squared_error and r2_score from sklearn.metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

## Import boston dataset from sklearn.datasets
from sklearn.datasets import load_boston
```

```
# Warnings Library - Ignore warnings
import warnings
warnings.filterwarnings('ignore')

### Load Boston dataset into a variable
boston = load_boston()
```

0.1.2 Understand the Dataset and the Data

```
In [3]: # Get keys of boston dataset dictionary
        print(boston.keys())
```

```
dict_keys(['data', 'target', 'feature_names', 'DESCR'])
```

```
In [4]: # Get the number of rows and columns in the dataset
        boston.data.shape
```

```
Out[4]: (506, 13)
```

```
In [5]: # Get the column names in the dataset
        print(boston.feature_names)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

```
In [6]: # Get description of the column names in the dataset
        print(boston.DESCR)
```

```
Boston House Prices dataset
===========================

Notes
------
Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive

    :Median Value (attribute 14) is usually the target

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
```

```
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
http://archive.ics.uci.edu/ml/datasets/Housing


This dataset was taken from the StatLib library which is maintained at Carnegie Mellon Universi

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regressi
problems.

**References**

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources o
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on th
   - many more! (see http://archive.ics.uci.edu/ml/datasets/Housing)
```

In [7]:
```
# Create dataframe from boston.data and target information in variables
features = pd.DataFrame(boston.data, columns=boston.feature_names)
targets = boston.target
```

In [8]:
```
# Updating the dataframe by adding the target column
features["PRICE"] = targets
```

Information of the dataframe created from the data set

In [9]:
```
# Shape of the dataframe
features.shape
```

Out[9]: (506, 14)

In [10]:
```
# Dataframe after updating with target
features.head(5)
```

Out[10]:
```
      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  PRICE
0     15.3  396.90   4.98   24.0
1     17.8  396.90   9.14   21.6
2     17.8  392.83   4.03   34.7
3     18.7  394.63   2.94   33.4
4     18.7  396.90   5.33   36.2
```

In [11]:
```
features.get_dtype_counts()
```

Out[11]:
```
float64    14
dtype: int64
```

The columns' datatypes are all numeric

0.2 Basic Details about Data - For Data Cleaning and Data Wrangling

Check for datatypes and presence of null values using info()

In [12]:
```
features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
CRIM       506 non-null float64
ZN         506 non-null float64
INDUS      506 non-null float64
CHAS       506 non-null float64
NOX        506 non-null float64
RM         506 non-null float64
AGE        506 non-null float64
DIS        506 non-null float64
RAD        506 non-null float64
TAX        506 non-null float64
PTRATIO    506 non-null float64
B          506 non-null float64
LSTAT      506 non-null float64
PRICE      506 non-null float64
```

```
dtypes: float64(14)
memory usage: 55.4 KB
```

No cleaning required as the data is already cleaned and has no null or NaN values

Checking if there are any row values = zero that need our consideration so that we can decide to study those rows

In [15]:
```
# Checking for the number of rows containing all values = 0
features.loc[(features==0).all(axis=1)].shape
```

Out[15]:
```
(0, 14)
```

In [16]:
```
# Checking for the number of rows containing atleast one value = 0
features.loc[(features==0).any(axis=1)].shape
```

Out[16]:
```
(499, 14)
```

The CHAS column is River dummy variable(= 1 if tract bounds river; 0 otherwise) and therefore the zeros in the column are valid values. So we don't need to clean the data in that column

In [19]:
```
features.columns.values
```

Out[19]:
```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT', 'PRICE'], dtype=object)
```

1 Basic Statistical Information

In [20]:
```
# Getting basic statistical information about the columns
features.describe() # Only numerical columns
```

Out[20]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	\
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	
mean	3.593761	11.363636	11.136779	0.069170	0.554695	6.284634	
std	8.596783	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%	3.647423	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       PRICE
count  506.000000  506.000000
mean    12.653063   22.532806
std      7.141062    9.197104
min      1.730000    5.000000
25%      6.950000   17.025000
50%     11.360000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

In [21]:
```python
# Getting correlation between various numerical columns
features.corr()
```

Out[21]:
```
             CRIM        ZN     INDUS      CHAS       NOX        RM       AGE  \
CRIM     1.000000 -0.199458  0.404471 -0.055295  0.417521 -0.219940  0.350784
ZN      -0.199458  1.000000 -0.533828 -0.042697 -0.516604  0.311991 -0.569537
INDUS    0.404471 -0.533828  1.000000  0.062938  0.763651 -0.391676  0.644779
CHAS    -0.055295 -0.042697  0.062938  1.000000  0.091203  0.091251  0.086518
NOX      0.417521 -0.516604  0.763651  0.091203  1.000000 -0.302188  0.731470
RM      -0.219940  0.311991 -0.391676  0.091251 -0.302188  1.000000 -0.240265
AGE      0.350784 -0.569537  0.644779  0.086518  0.731470 -0.240265  1.000000
DIS     -0.377904  0.664408 -0.708027 -0.099176 -0.769230  0.205246 -0.747881
RAD      0.622029 -0.311948  0.595129 -0.007368  0.611441 -0.209847  0.456022
TAX      0.579564 -0.314563  0.720760 -0.035587  0.668023 -0.292048  0.506456
PTRATIO  0.288250 -0.391679  0.383248 -0.121515  0.188933 -0.355501  0.261515
B       -0.377365  0.175520 -0.356977  0.048788 -0.380051  0.128069 -0.273534
LSTAT    0.452220 -0.412995  0.603800 -0.053929  0.590879 -0.613808  0.602339
PRICE   -0.385832  0.360445 -0.483725  0.175260 -0.427321  0.695360 -0.376955

              DIS       RAD       TAX   PTRATIO         B     LSTAT     PRICE
CRIM    -0.377904  0.622029  0.579564  0.288250 -0.377365  0.452220 -0.385832
ZN       0.664408 -0.311948 -0.314563 -0.391679  0.175520 -0.412995  0.360445
INDUS   -0.708027  0.595129  0.720760  0.383248 -0.356977  0.603800 -0.483725
CHAS    -0.099176 -0.007368 -0.035587 -0.121515  0.048788 -0.053929  0.175260
NOX     -0.769230  0.611441  0.668023  0.188933 -0.380051  0.590879 -0.427321
RM       0.205246 -0.209847 -0.292048 -0.355501  0.128069 -0.613808  0.695360
AGE     -0.747881  0.456022  0.506456  0.261515 -0.273534  0.602339 -0.376955
DIS      1.000000 -0.494588 -0.534432 -0.232471  0.291512 -0.496996  0.249929
RAD     -0.494588  1.000000  0.910228  0.464741 -0.444413  0.488676 -0.381626
TAX     -0.534432  0.910228  1.000000  0.460853 -0.441808  0.543993 -0.468536
PTRATIO -0.232471  0.464741  0.460853  1.000000 -0.177383  0.374044 -0.507787
B        0.291512 -0.444413 -0.441808 -0.177383  1.000000 -0.366087  0.333461
LSTAT   -0.496996  0.488676  0.543993  0.374044 -0.366087  1.000000 -0.737663
PRICE    0.249929 -0.381626 -0.468536 -0.507787  0.333461 -0.737663  1.000000
```

The column RM is highly correlated with PRICE. (Correlation = 0.695)

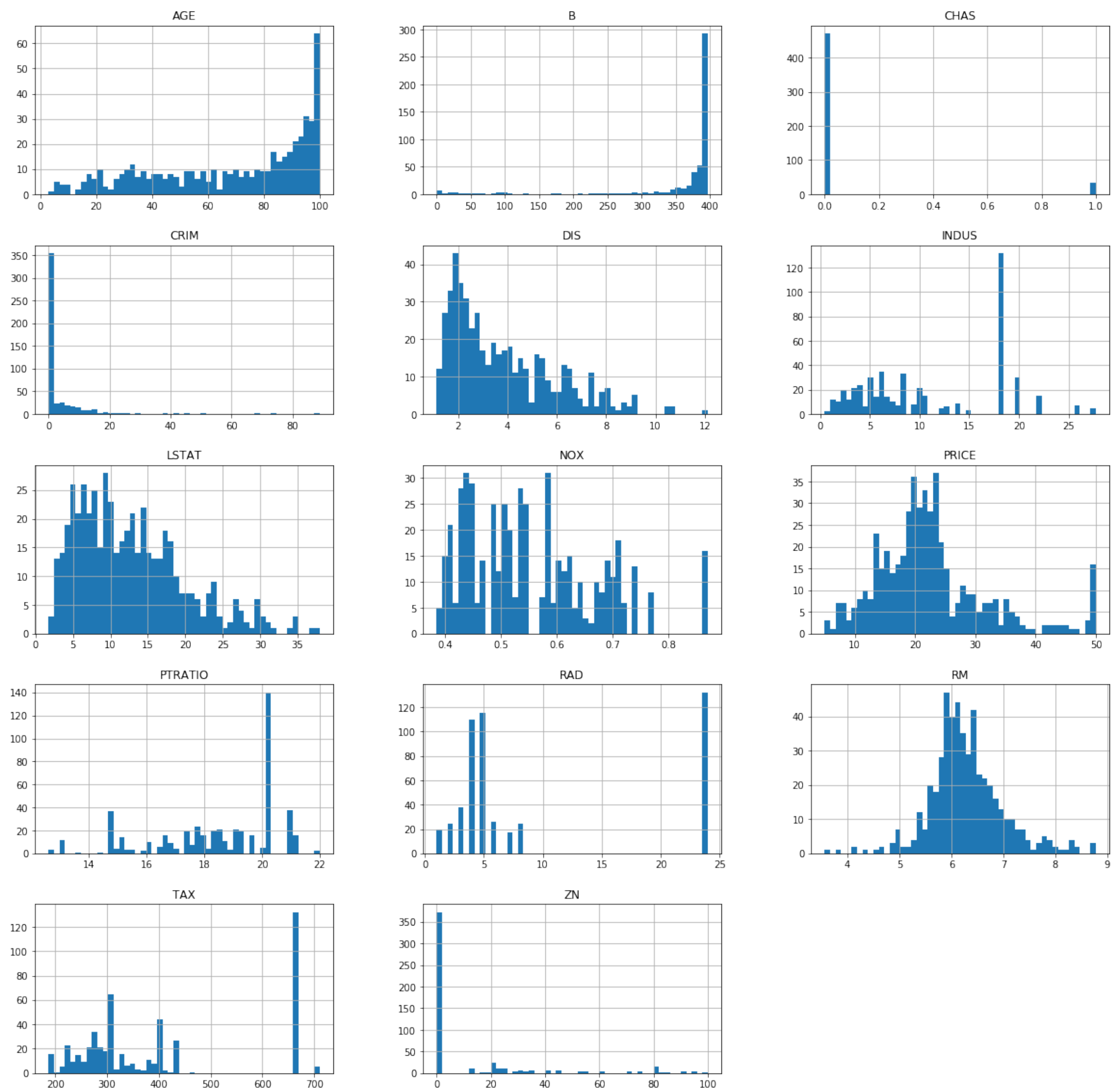
The column LSTAT is highly negatively correlated with PRICE. (Correlation = -0.737)

The columns INDUS, NOX, TAX, PTRATIO are moderately negatively correlated with PRICE(-0.7 < Correlations < -0.4)

2 Data Exploration - Visual Analysis OR Exploratory Analysis

2.1 Uni-variate

In [22]:
```
# Plotting the histograms of numerical columns to understand their distribution
features.hist(bins=50, figsize=(20,20), layout=(5,3))
plt.show()
```

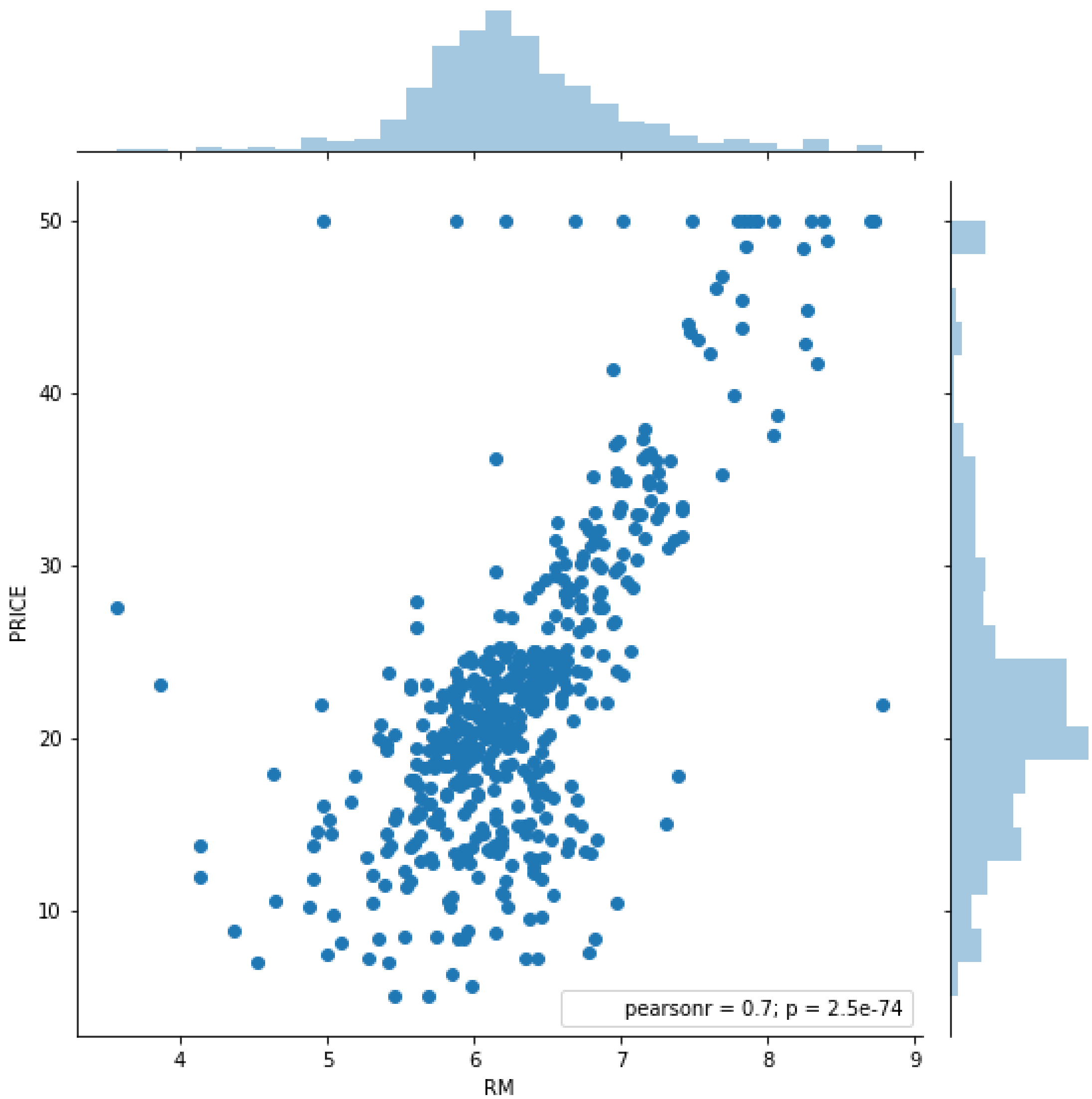


2.2 Bi-Variate

Plotting: PRICE vs RM, LSTAT, INDUS, NOX, TAX and PTRATIO

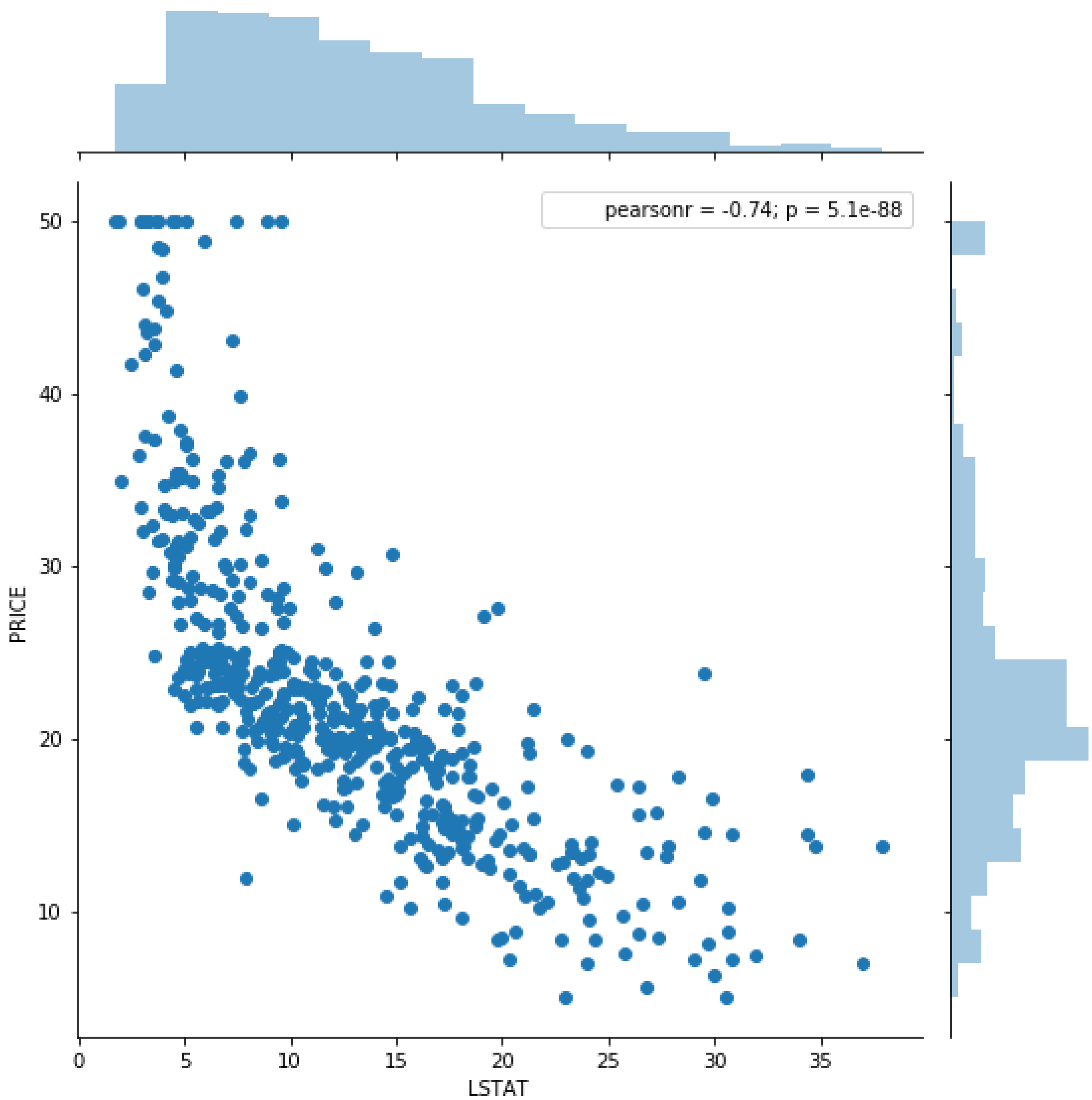
In [25]:
```
sns.jointplot(x=features["RM"], y=features["PRICE"], kind='scatter',size = 8)
```

Out[25]: <seaborn.axisgrid.JointGrid at 0x29ba6f56b00>



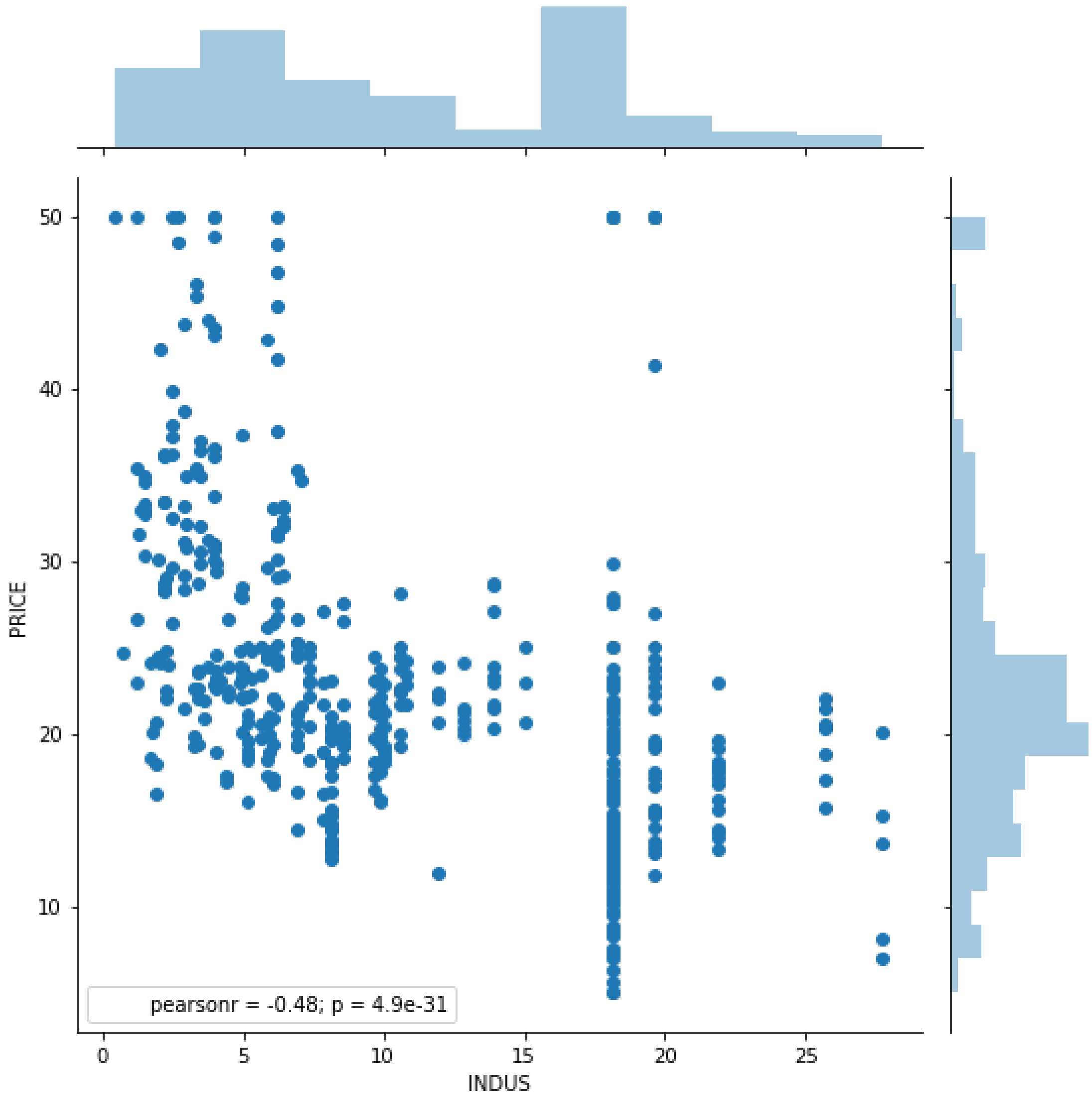
In [26]:
```
sns.jointplot(x=features["LSTAT"], y=features["PRICE"], kind='scatter',size = 8)
```

Out[26]: <seaborn.axisgrid.JointGrid at 0x29ba73b4470>



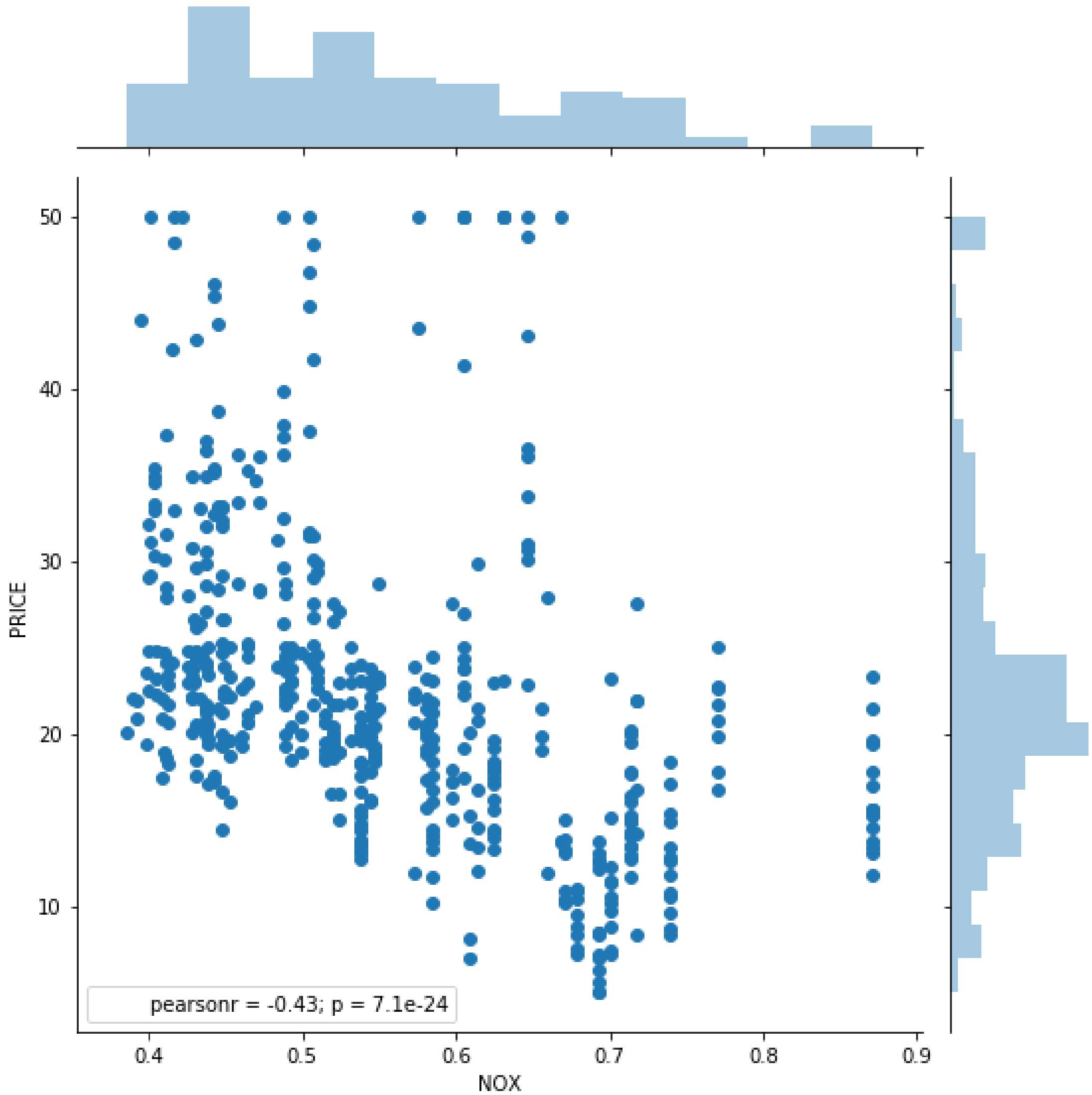

```
In [27]: sns.jointplot(x=features["INDUS"], y=features["PRICE"], kind='scatter',size = 8)
```

```
Out[27]: <seaborn.axisgrid.JointGrid at 0x29ba7503780>
```



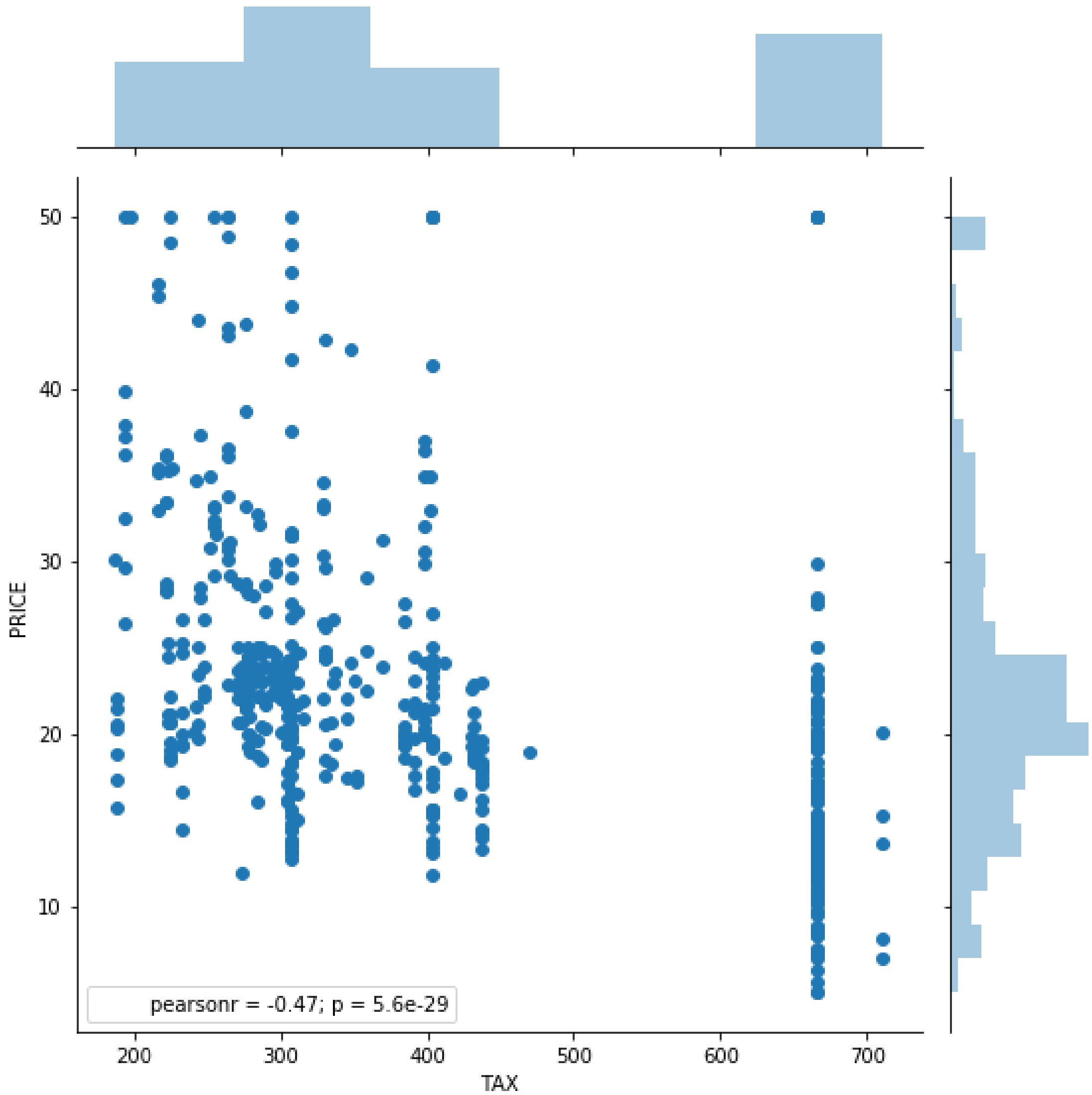

```
In [28]: sns.jointplot(x=features["NOX"], y=features["PRICE"], kind='scatter',size = 8)
```

```
Out[28]: <seaborn.axisgrid.JointGrid at 0x29ba6122be0>
```



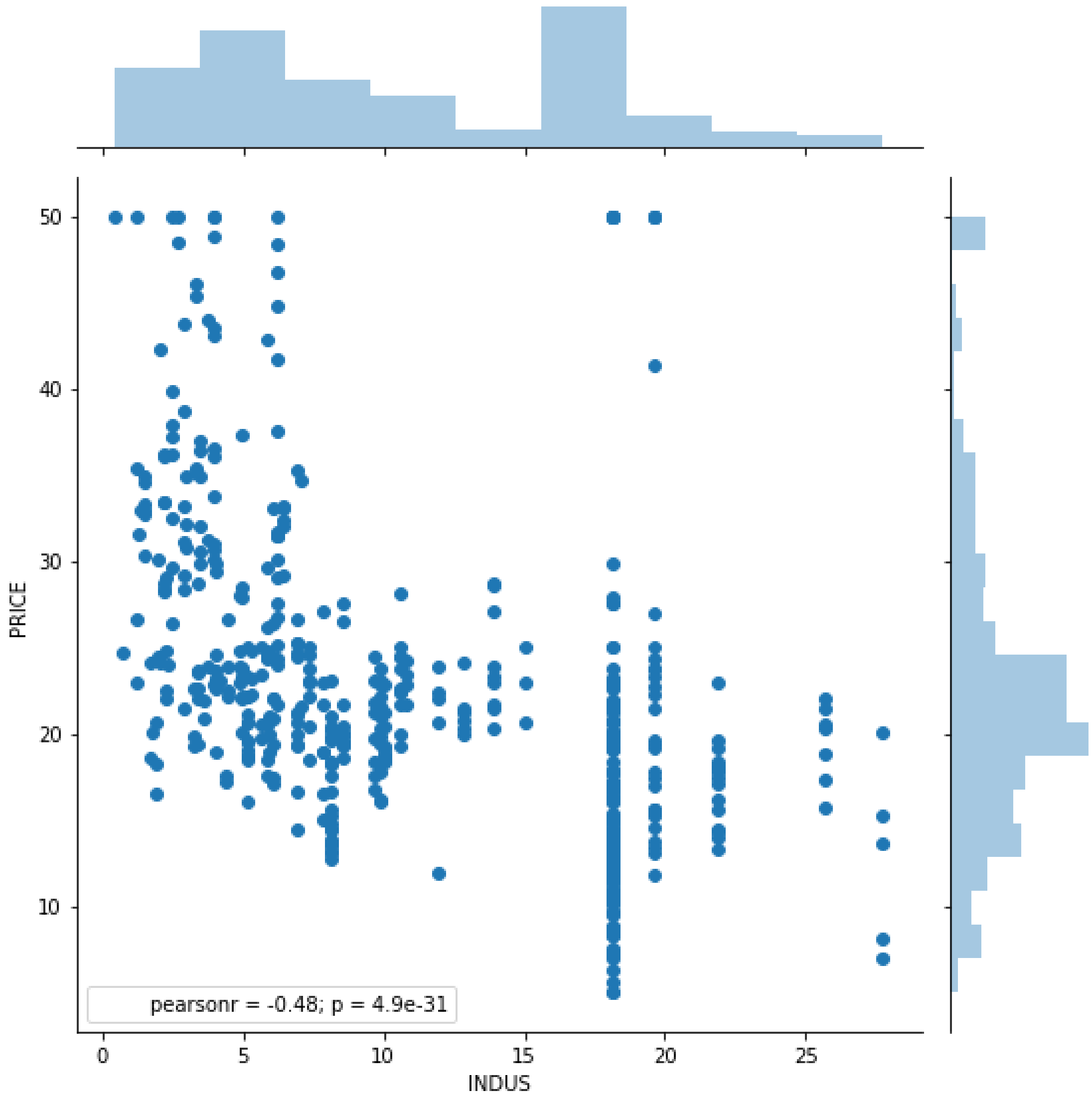

```
In [29]: sns.jointplot(x=features["TAX"], y=features["PRICE"], kind='scatter',size = 8)

Out[29]: <seaborn.axisgrid.JointGrid at 0x29ba67762b0>
```



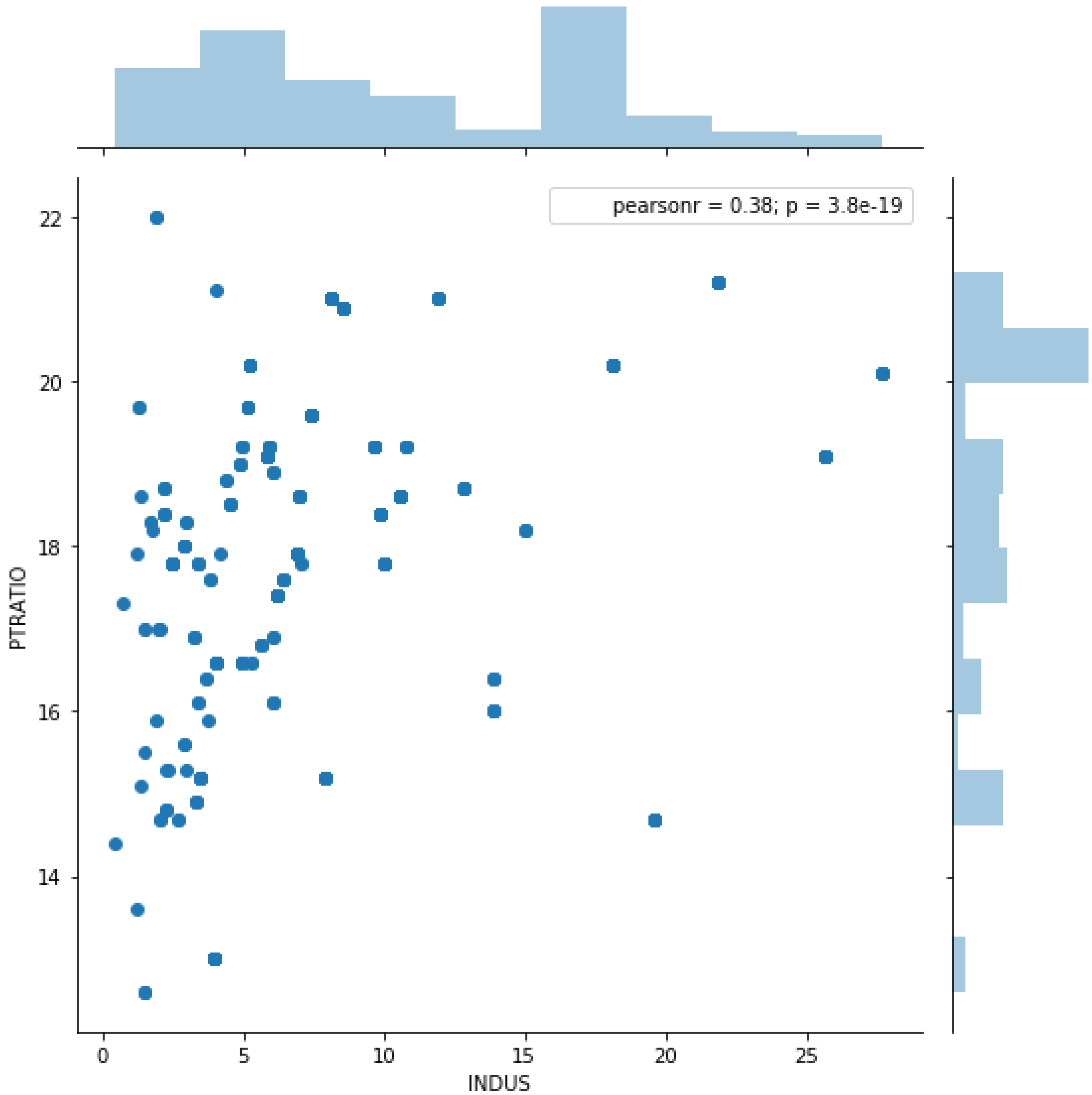

```
In [30]:  sns.jointplot(x=features["INDUS"], y=features["PRICE"], kind='scatter',size = 8)
```

```
Out[30]: <seaborn.axisgrid.JointGrid at 0x29ba69225c0>
```




```
In [31]: sns.jointplot(x=features["INDUS"], y=features["PTRATIO"], kind='scatter',size = 8)
```

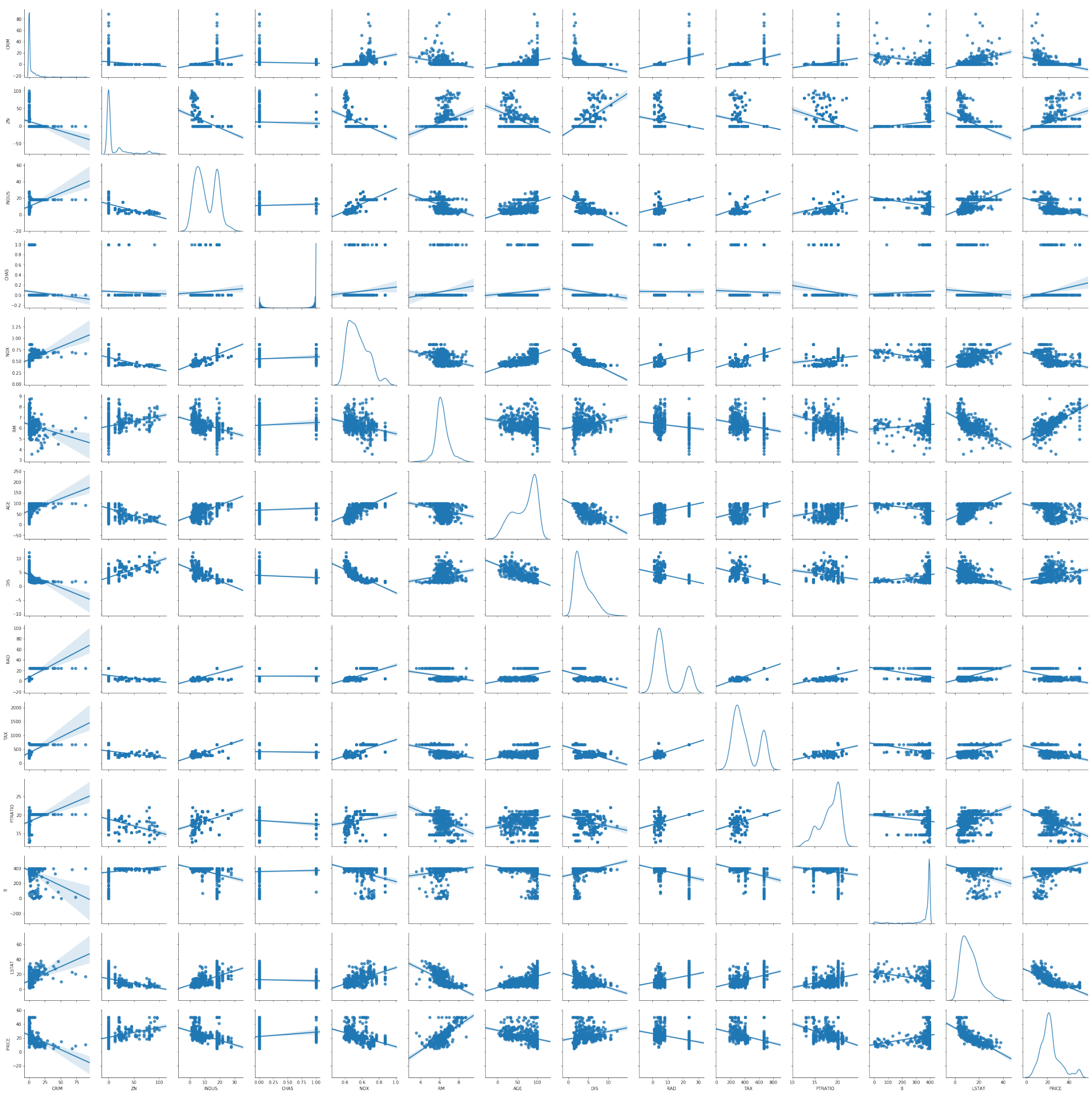
```
Out[31]: <seaborn.axisgrid.JointGrid at 0x29ba6d0ce10>
```



2.3 Multi-Variate

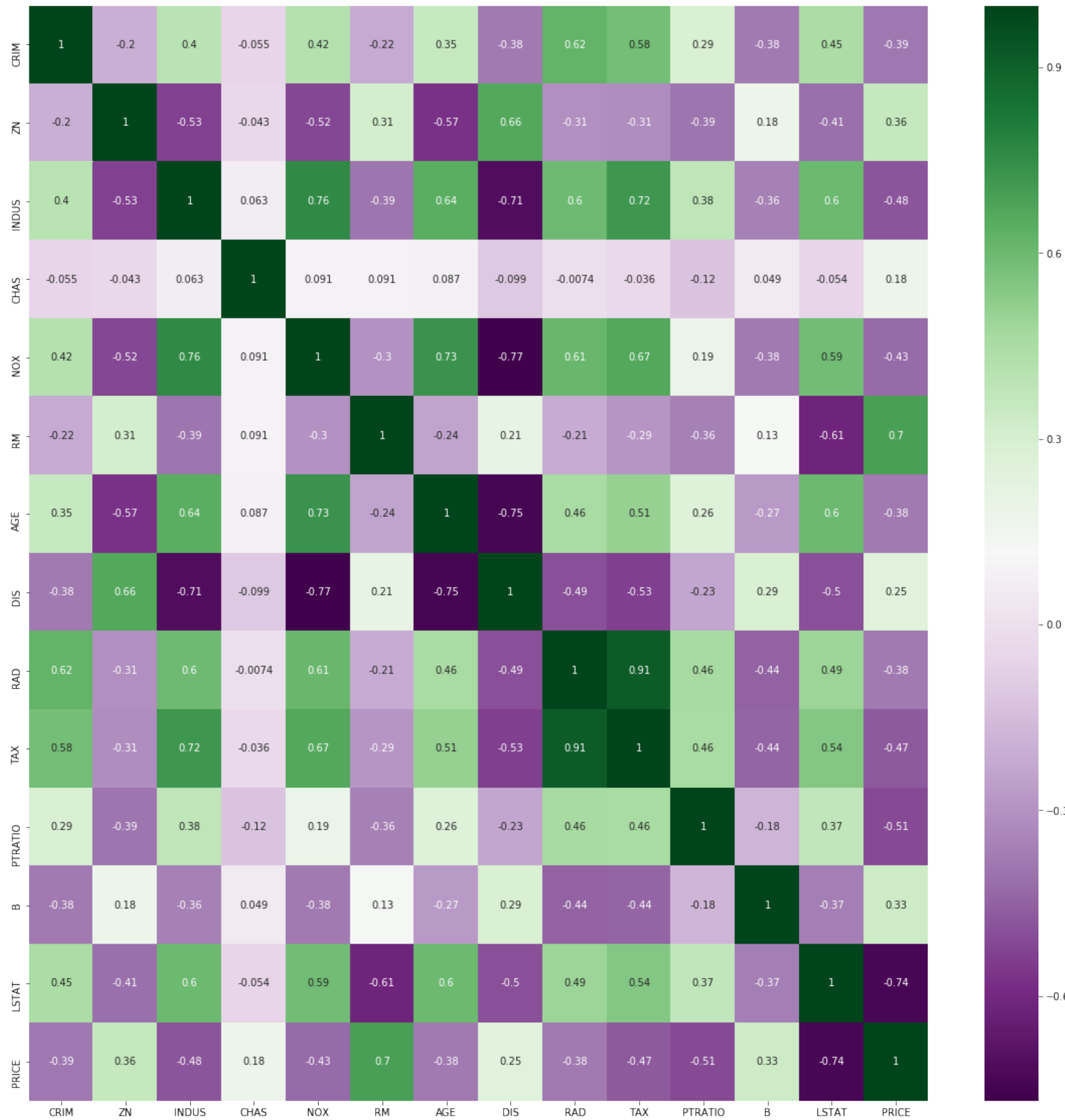
```
In [32]: sns.pairplot(features, kind='reg', diag_kind = 'kde')
```

```
Out[32]: <seaborn.axisgrid.PairGrid at 0x29ba6b41ac8>
```



```
In [33]: # Checking for correlations using HEATMAP
         plt.figure(figsize=(20,20))
         sns.heatmap(features.corr(), cmap="PRGn", annot= True)

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x29baf130c50>
```



3 Feature Engineering

There is no need to engineer features from this dataset. Also, there are no categorical variables to engineer

4 Train - Test Split

```
In [34]: X = features.drop("PRICE", axis=1)
         Y = features["PRICE"]
```

```
# We will be using 80:20 split for train and test datasets
x_train, x_test, y_train, y_test = train_test_split(X,Y,test_size=0.20, random_state =
```

```
In [35]: print(x_train.shape, x_test.shape)
```

```
(404, 13) (102, 13)
```

```
In [36]: print(y_train.shape, y_test.shape)
```

```
(404,) (102,)
```

5 Fitting Models

5.1 Linear Regression

```
In [38]: lm = LinearRegression()
         model = lm.fit(x_train, y_train) # Sklearn already considers the intercepts for linea
         print("Estimated Beta Coefficients: \n", model.coef_)

         y_test_pred = model.predict(x_test)

         print("\nLinear Regression - Base", "\n\t R2-Score:", model.score(x_test, y_test),
                                 "\n\t RMSE:", math.sqrt(mean_squared_error(y_test_pred, y_tes
```

```
Estimated Beta Coefficients:
 [-8.01644009e-02  4.79926054e-02 -5.07131765e-03  3.06486600e+00
 -1.61596810e+01  3.66858142e+00 -8.46805789e-03 -1.51719956e+00
  2.86612524e-01 -1.21155515e-02 -9.24761912e-01  9.62688265e-03
 -4.86676845e-01]

Linear Regression - Base
         R2-Score: 0.7554467329645207
         RMSE: 4.860294126345348
```

5.2 RandomForestRegressor

```
In [39]: rf_reg = RandomForestRegressor(n_estimators=100)
         rf_model= rf_reg.fit(x_train, y_train)
         y_test_pred = rf_model.predict(x_test)

         print("## RandomForest Regressor - Unscaled Data", "\n\t R2-Score:", rf_model.score(x
                                 "\n\t RMSE:", math.sqrt(mean_squared_error(y_test_pred, y_tes
```

```
## RandomForest Regressor - Unscaled Data
         R2-Score: 0.8912837096319373
```

```
RMSE: 3.2405828542554054
```

5.3 RandomForestRegressor with StandardScaled data

```
In [40]: scaler = StandardScaler()
         scaled_features = scaler.fit_transform(features)
         scaled_features = pd.DataFrame(scaled_features, columns = features.columns.values)
         scaled_features.head()
```

```
Out[40]:        CRIM        ZN     INDUS      CHAS       NOX        RM       AGE  \
         0 -0.417713  0.284830 -1.287909 -0.272599 -0.144217  0.413672 -0.120013
         1 -0.415269 -0.487722 -0.593381 -0.272599 -0.740262  0.194274  0.367166
         2 -0.415272 -0.487722 -0.593381 -0.272599 -0.740262  1.282714 -0.265812
         3 -0.414680 -0.487722 -1.306878 -0.272599 -0.835284  1.016303 -0.809889
         4 -0.410409 -0.487722 -1.306878 -0.272599 -0.835284  1.228577 -0.511180

                 DIS       RAD       TAX   PTRATIO         B     LSTAT     PRICE
         0  0.140214 -0.982843 -0.666608 -1.459000  0.441052 -1.075562  0.159686
         1  0.557160 -0.867883 -0.987329 -0.303094  0.441052 -0.492439 -0.101524
         2  0.557160 -0.867883 -0.987329 -0.303094  0.396427 -1.208727  1.324247
         3  1.077737 -0.752922 -1.106115  0.113032  0.416163 -1.361517  1.182758
         4  1.077737 -0.752922 -1.106115  0.113032  0.441052 -1.026501  1.487503
```

```
In [41]: X_scaled = scaled_features.drop("PRICE", axis=1)
         Y_scaled = scaled_features["PRICE"]

         # We will be using 80:20 split for train and test datasets
         x_scaled_train, x_scaled_test, y_scaled_train, y_scaled_test = train_test_split(X_scal
```

```
In [43]: rf_reg = RandomForestRegressor(n_estimators=100)
         rf_model_scaled = rf_reg.fit(x_scaled_train, y_scaled_train)
         y_scaled_test_pred = rf_model_scaled.predict(x_scaled_test)

         print("## RandomForestRegressor - ScaledData", "\n\t R2-Score:", rf_model_scaled.score
                                "\n\t RMSE:", math.sqrt(mean_squared_error(y_scaled_test_pred
```

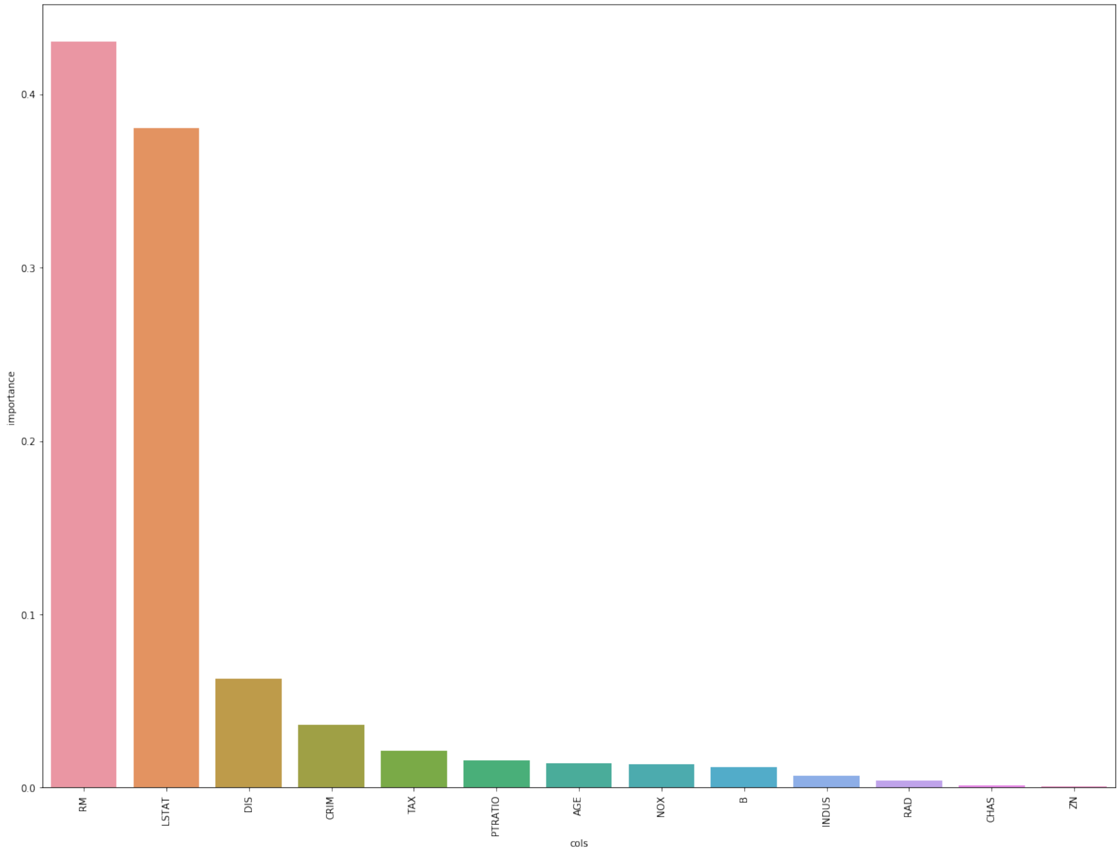
```
## RandomForestRegressor - ScaledData
         R2-Score: 0.8843132183938184
         RMSE: 0.36382802988641866
```

6 Feature Selection

```
In [44]: importance = pd.DataFrame.from_dict({'cols':x_train.columns, 'importance': rf_reg.fea
         importance = importance.sort_values(by='importance', ascending=False)
```

```
plt.figure(figsize=(20,15))
sns.barplot(importance.cols, importance.importance)
plt.xticks(rotation=90)
```

Out[44]: (array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]),
 <a list of 13 Text xticklabel objects>)



In [46]:
```
imp_cols = importance[importance.importance >= 0.01].cols.values
imp_cols
```

Out[46]: array(['RM', 'LSTAT', 'DIS', 'CRIM', 'TAX', 'PTRATIO', 'AGE', 'NOX', 'B'],
      dtype=object)

In [47]:
```
# Fitting models with columns where feature importance>=0.01

x_train, x_test, y_train, y_test = train_test_split(X[imp_cols],Y,test_size=0.20, ranc

rf_reg = RandomForestRegressor(n_estimators=100)
rf_model= rf_reg.fit(x_train, y_train)
y_test_pred = rf_model.predict(x_test)
```

```
print("## RandomForest Regressor - Unscaled Data", "\n\t R2-Score:", rf_model.score(x
                          "\n\t RMSE:", math.sqrt(mean_squared_error(y_test_pred, y_tes



x_scaled_train, x_scaled_test, y_scaled_train, y_scaled_test = train_test_split(X_scal

rf_reg = RandomForestRegressor(n_estimators=100)
rf_model_scaled = rf_reg.fit(x_scaled_train, y_scaled_train)
y_scaled_test_pred = rf_model_scaled.predict(x_scaled_test)

print("## RandomForestRegressor - Scaled Data", "\n\t R2-Score:", rf_model_scaled.scor
                          "\n\t RMSE:", math.sqrt(mean_squared_error(y_scaled_test_pred
```

```
## RandomForest Regressor - Unscaled Data
         R2-Score: 0.8883031363395156
         RMSE: 3.2847045096456196

## RandomForestRegressor - Scaled Data
         R2-Score: 0.8933982831642273
         RMSE: 0.34924997580717
```

In [48]:
```
imp_cols = importance[importance.importance >= 0.005].cols.values
imp_cols
```

Out[48]:
```
array(['RM', 'LSTAT', 'DIS', 'CRIM', 'TAX', 'PTRATIO', 'AGE', 'NOX', 'B',
       'INDUS'], dtype=object)
```

In [49]:
```
# Fitting models with columns where feature importance>=0.005

x_train, x_test, y_train, y_test = train_test_split(X[imp_cols],Y,test_size=0.20, ranc

rf_reg = RandomForestRegressor(n_estimators=100)
rf_model= rf_reg.fit(x_train, y_train)
y_test_pred = rf_model.predict(x_test)

print("## RandomForest Regressor - Unscaled Data", "\n\t R2-Score:", rf_model.score(x
                          "\n\t RMSE:", math.sqrt(mean_squared_error(y_test_pred, y_tes



x_scaled_train, x_scaled_test, y_scaled_train, y_scaled_test = train_test_split(X_scal

rf_reg = RandomForestRegressor(n_estimators=100)
rf_model_scaled = rf_reg.fit(x_scaled_train, y_scaled_train)
y_scaled_test_pred = rf_model_scaled.predict(x_scaled_test)
```

```
print("## RandomForestRegressor - Scaled Data", "\n\t R2-Score:", rf_model_scaled.sco
                             "\n\t RMSE:", math.sqrt(mean_squared_error(y_scaled_test_pred
```

```
## RandomForest Regressor - Unscaled Data
         R2-Score: 0.888963184316594
         RMSE: 3.274985011191563

## RandomForestRegressor - Scaled Data
         R2-Score: 0.8840231988338807
         RMSE: 0.36428379153493723
```

In [50]:
```
imp_cols = importance[importance.importance >= 0.002].cols.values
imp_cols
```

Out[50]:
```
array(['RM', 'LSTAT', 'DIS', 'CRIM', 'TAX', 'PTRATIO', 'AGE', 'NOX', 'B',
       'INDUS', 'RAD'], dtype=object)
```

In [51]:
```
# Fitting models with columns where feature importance>=0.002

x_train, x_test, y_train, y_test = train_test_split(X[imp_cols],Y,test_size=0.20, ranc

rf_reg = RandomForestRegressor(n_estimators=100)
rf_model= rf_reg.fit(x_train, y_train)
y_test_pred = rf_model.predict(x_test)

print("## RandomForest Regressor - Unscaled Data", "\n\t R2-Score:", rf_model.score(x_
                             "\n\t RMSE:", math.sqrt(mean_squared_error(y_test_pred, y_tes



x_scaled_train, x_scaled_test, y_scaled_train, y_scaled_test = train_test_split(X_scal

rf_reg = RandomForestRegressor(n_estimators=100)
rf_model_scaled = rf_reg.fit(x_scaled_train, y_scaled_train)
y_scaled_test_pred = rf_model_scaled.predict(x_scaled_test)

print("## RandomForestRegressor - Scaled Data", "\n\t R2-Score:", rf_model_scaled.scor
                             "\n\t RMSE:", math.sqrt(mean_squared_error(y_scaled_test_pred
```

```
## RandomForest Regressor - Unscaled Data
         R2-Score: 0.8811702177745246
         RMSE: 3.3879615118655564

## RandomForestRegressor - Scaled Data
         R2-Score: 0.8888968021878442
         RMSE: 0.35654763879104023
```

7 Validation

In [52]:
```
# Cross validating the model created with columns whose feature importances >= 0.002,
scoring = 'neg_mean_squared_error'
kfold = KFold(n_splits=10, random_state=100)

cv_results = cross_val_score(model, x_train,y_train, cv=kfold, scoring=scoring)
print("## Linear Regression","\n\t CV-Mean:", cv_results.mean(),
                                                "\n\t CV-Std. Dev:",  cv_results.std

cv_results = cross_val_score(rf_model, x_train,y_train, cv=kfold, scoring=scoring)
print("## RandomForestRegressor - Unscaled data","\n\t CV-Mean:", cv_results.mean(),
                                                "\n\t CV-Std. Dev:",  cv_results.std

cv_results = cross_val_score(rf_model_scaled, x_scaled_train,y_scaled_train, cv=kfold
print("## RandomForestRegressor - Scaled data","\n\t CV-Mean:", cv_results.mean(),
                                                "\n\t CV-Std. Dev:",  cv_results.std
```

```
## Linear Regression
         CV-Mean: -24.501361520600785
         CV-Std. Dev: 7.478252464775003

## RandomForestRegressor - Unscaled data
         CV-Mean: -10.768933929329275
         CV-Std. Dev: 4.507874541914067

## RandomForestRegressor - Scaled data
         CV-Mean: -0.13622902633058565
         CV-Std. Dev: 0.060291200294718905
```

8 Optimization - Model Tuning

In [53]:
```
RF_Regressor =  RandomForestRegressor(n_estimators=100, n_jobs = -1, random_state = 1(

CV = ShuffleSplit(test_size=0.20, random_state=100)

param_grid = {"max_depth": [5, None],
              "n_estimators": [50, 100, 150, 200],
              "min_samples_split": [2, 4, 5],
              "min_samples_leaf": [2, 4, 6]
             }
```

8.1 Best Estimator - Unscaled Data

In [54]:
```
rscv_grid1 = GridSearchCV(RF_Regressor, param_grid=param_grid, verbose=1)
```

In [55]: rscv_grid1.fit(x_train, y_train)

```
Fitting 3 folds for each of 72 candidates, totalling 216 fits


[Parallel(n_jobs=1)]: Done 216 out of 216 | elapsed:  1.6min finished
```

Out[55]:
```
GridSearchCV(cv=None, error_score='raise',
       estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=-1,
           oob_score=False, random_state=100, verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'max_depth': [5, None], 'n_estimators': [50, 100, 150, 200], 'min_s
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=1)
```

In [56]: rscv_grid1.best_params_

Out[56]:
```
{'max_depth': None,
 'min_samples_leaf': 2,
 'min_samples_split': 2,
 'n_estimators': 50}
```

In [57]:
```
# Best Estimator - Unscaled
rf_model = rscv_grid1.best_estimator_
rf_model.fit(x_train, y_train)
```

Out[57]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=2, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=50, n_jobs=-1,
           oob_score=False, random_state=100, verbose=0, warm_start=False)
```

In [58]: rf_model.score(x_test, y_test)

Out[58]: 0.8781370746758193

8.2 Best Estimator - Scaled Data

In [59]: rscv_grid2 = GridSearchCV(RF_Regressor, param_grid=param_grid, verbose=1)

In [60]: rscv_grid2.fit(x_scaled_train, y_scaled_train)

```
Fitting 3 folds for each of 72 candidates, totalling 216 fits
```

```
[Parallel(n_jobs=1)]: Done 216 out of 216 | elapsed:  1.6min finished
```

```
Out[60]: GridSearchCV(cv=None, error_score='raise',
                estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None
                    max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=-1,
                    oob_score=False, random_state=100, verbose=0, warm_start=False),
                fit_params=None, iid=True, n_jobs=1,
                param_grid={'max_depth': [5, None], 'n_estimators': [50, 100, 150, 200], 'min_s
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring=None, verbose=1)
```

```
In [61]: rscv_grid2.best_params_
```

```
Out[61]: {'max_depth': None,
          'min_samples_leaf': 2,
          'min_samples_split': 4,
          'n_estimators': 150}
```

```
In [62]: # Best Estimator - Scaled
         rf_model_scaled = rscv_grid2.best_estimator_
         rf_model_scaled.fit(x_scaled_train, y_scaled_train)
```

```
Out[62]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                    max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=2, min_samples_split=4,
                    min_weight_fraction_leaf=0.0, n_estimators=150, n_jobs=-1,
                    oob_score=False, random_state=100, verbose=0, warm_start=False)
```

```
In [63]: rf_model_scaled.score(x_scaled_test, y_scaled_test)
```

```
Out[63]: 0.8771363772523068
```

9 Comparing Performance Metrics

```
In [64]: print("RandomForestRegressor - Unscaled Data\n\t R2-Score:", rf_model.score(x_test, y
                       "\n\t RMSE:", math.sqrt(mean_squared_error(rf_model.predict(x_test),

         print("RandomForestRegressor - Scaled Data\n\t R2-Score:", rf_model_scaled.score(x_sca
                       "\n\t RMSE:", math.sqrt(mean_squared_error(rf_model_scaled.predict(x_
```

```
RandomForestRegressor - Unscaled Data
         R2-Score: 0.8781370746758193
         RMSE: 3.430928100601086
```

```
RandomForestRegressor - Scaled Data
         R2-Score: 0.8771363772523068
         RMSE: 0.37494359878281436
```

10 Choosing the model

We can see that Random Forest Regressor trained on scaled data gives better RMSE value (= 0.37447). So, Random Forest Regressor trained on scaled data should be used as the regression model for this dataset.