Gurjus Singh

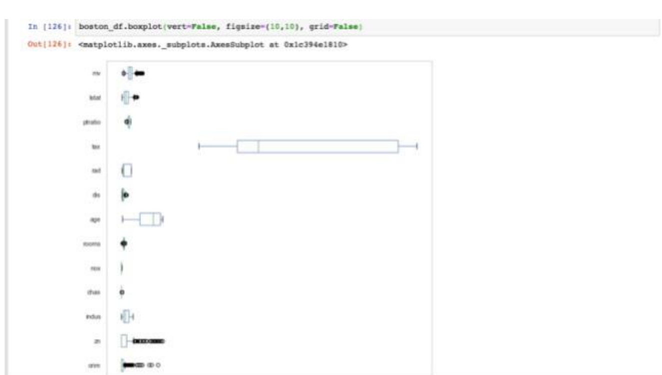
MSDS 422 – Practical Machine Learning

October 11th, 2020

Assignment #4 Random Forests and Gradient Boosting PART A

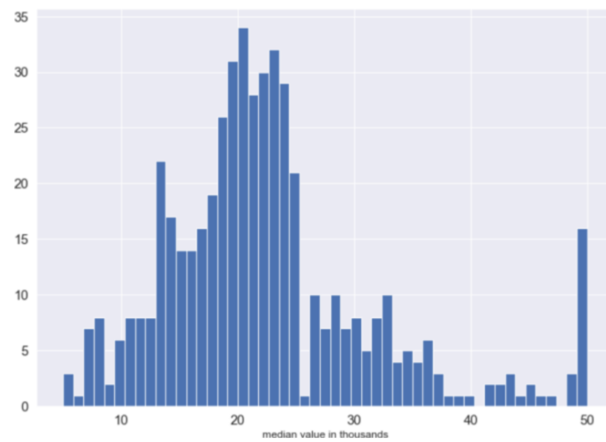
Data preparation, exploration, visualization

In the Boston dataset our goal was to see the effect of nox on the median value of houses, I also wanted to see which features in general had a great effect on the median value. I first started out by loading the dataset “Boston.csv”. I first wanted to get an initial look and feel for the data set. So, I looked at the size of the data set which was (512, 14). This means there are 512 rows and 14 columns mainly features and one variable we want to predict. After getting a feel for the size of the dataset I wanted to check out the head of data frame and types for every column. I saw non-numeric row, which we do not want to use in linear regression, so I dropped this feature column. Most of the columns were mainly floats, except for two columns which contained integer values. I used **.describe()** value and .**isnull.sum()** to find out whether the initial data had any typos, or missing value. From looking at min-max category in describe table I did not see anything obviously wrong, so I decided to continue. I also did not see any columns with NA values which is what **isnull.sum()** does. This counts up all the values which evaluate to True in a column.



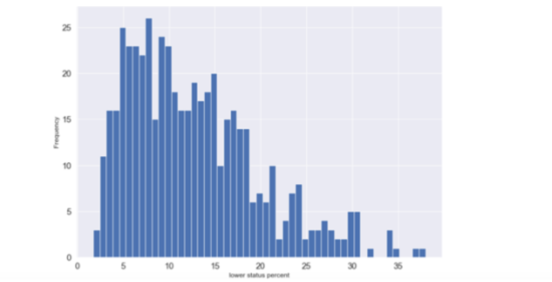
Boxplot of Columns (Plot 1-1)

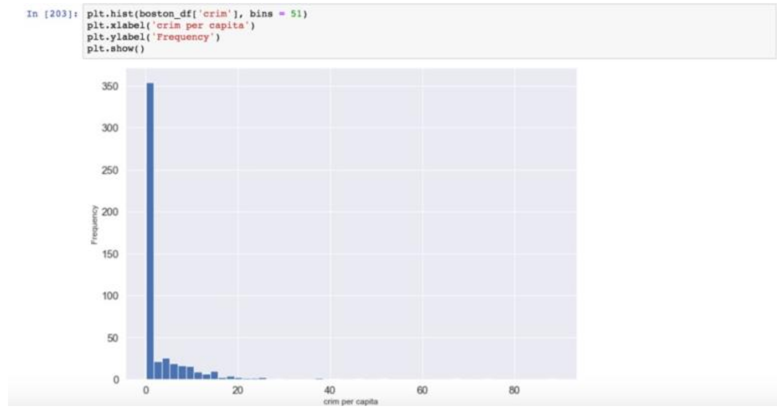
I then decided to go to exploring through data visualizations of the data set. I first tried to boxplot as depicted in Plot 1-1. In the plot I could see many of our columns had an abundance of outliers such as our target variable **MV, lstat, dis, rooms, chas, zn and crim**. I wanted to play around more and look closely at some of the variables, so I decided to see distributions through histograms.



Histogram Plot 1-2

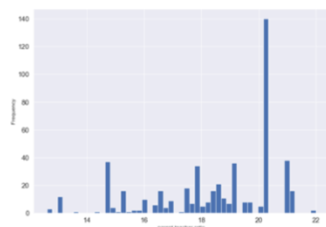
In plot 1-2, it is the median value of the houses, which we will be predicting on this using the independent variables. This looks like a less skewed distribution in comparison to doing the boxplot. I also wanted to look at features I thought were more important to predicting this dataset such as **crim**, which is the per capita crime rate per town, **ptratio** which is pupil to teacher ratio by town and **lstat** which is percent of lower status in an area.





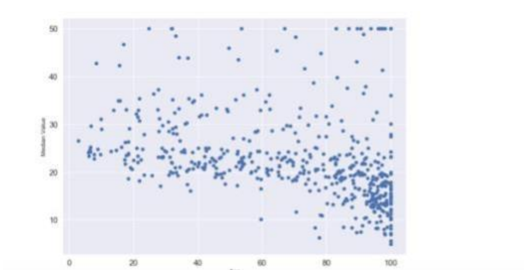
Histogram Plot 1-3

Histogram 1-4



Histogram 1-5

The **lstat** variable in 1-3 seems to be skewed to the right where 10% in a given neighborhood or region are mostly of lower status. In 1-4 the histogram is right skewed as well with most of the crime happening near 0 in the areas in Boston where the data is collected. In Histogram 1-5, we see for **ptratio**, is near 20 which mean the difference in teacher to student ratio is high, this data is more skewed to the left. Lastly, I wanted to make some plots to see any initial correlation between MV, and these variables. I also wanted to see if age was in any way correlated.



Dot plot of Age variable compared to MV Plot 1-6

Figure 1-7 of highly linear 'rooms' variable

In plot 1-6, I do not see any linear shape compared to age, which suggests there is no relationship in MV, and the **age** variable. I also looked at crim variable which did not seem to have a linear shape as well. During my initial findings I also saw that rooms and **lstat** variables seemed to have a linear shape which means they were highly positive or negatively correlated as seen in figure 1-7.

After examining linear shaped data and nonlinear data, I then wanted to scale the features between 0 and 1 as it helps speed up the learning process. In order to make the data more linear I had to use the lambda function which helps to map the function to all the data in one line of code. I did not want any zeros in my data, so I added .01 to every data point. I then used "boxcox" which is great for making sure my data is more normalized and more linear to use linear models on it. I had to hand pick the columns, so I chose the features shown below because I felt they were nonlinear by looking at their scatterplots.

```
['age', 'zn', 'crim', 'ptratio', 'chas', 'indus', 'nox', 'dis', 'rad', 'tax', 'ptratio', 'lstat']]
```

Before modeling using the linear regression method and tree methods, I first wanted to create a correlation heatmap to get an overall picture of which features in the data set might be more useful to predict median value price of a neighborhood of houses. This heatmap showed exactly what might initial findings confirmed in that lstat was negatively correlated, and rooms was highly positively correlated as shown in figure 1-8 below. It also shows which columns we would like to drop that have the multicollinear property.

Figure 1-8 Heatmap

Review research design and modeling methods

After we choose, the features I am going to use 3 modeling methods which are mainly tree learning algorithms but have some variations and one regression algorithm. The three types are Regular Linear Regression, Random Forest, and Extra Randomized Trees. The differences between the three is that Linear Regression does not add regularization term which is important in most cases to prevent overfitting [1]. Linear Regression assumes linearity so that is why it is important to transform your features. Linear Regression is also important because to use when doing predictions on numerical response values [1]. Random Forest is another special algorithm which involves decision trees [1]. Decision trees are trees that use feature cutoffs to decide on the final value of the prediction [1]. With Random Forests it involves an ensembled method which involves different trees which all aggregate together to form a predicted result [1]. Each tree uses a method to collect training data which is known as “Bagging”. “Bagging” involves sampling which collects n data points from the training set to train on. Thirdly another Tree learning algorithm used is Extra Randomized Trees [1]. This is similar to Random Forests in a sense because they both pick a random number of features to train on, but this also involves random thresholds to use as a cutoff for each feature chosen [1]. I think these algorithms are important because Linear Regression as mentioned before is mostly used to predict numerical values. Trees are kind of different in that one can see how the algorithm works and it is not a “black-box” like Linear Regression is. Trees do not assume linearity like Regression does and does not need transformation and scaling [1].

Before we start training the algorithm, I had to save the target variable in its own data frame to separate it from the features. In order to use the regression methods, it is important to split up the data into train data, and test data. I did the 80-20 split on the data. Our goal was to make sure that the data trained and predicted well on the test set.

Review Results, and Evaluate Model

After implementing the model, I took at the results at both the Training Set and Test Set. I also wanted to test the models by dropping features with insignificant P-values as shown in 1-9. I first looked at the results from when no columns were dropped. It looks like the data was overfitting by looking at the Training Data Results from 1-9. What I noticed for was Extra Trees was the highest with an R^2 of 0.97. I then looked at the Results of the Test Set in 1-10 and I saw Extra Trees was still in the lead with an R^2 of 0.8848. I then dropping features with high P-values mainly age, zn, and crim first seen in 1-12. What I noticed is that Extra Trees was still in the lead with an R^2 performing slightly lower, but not that lower compared to Linear Regression with a R^2 of 0.8846. I then dropped other columns to only leave P-Values of 0 shown in 1-13. What I noticed is Extra Trees slightly went down, but still came out on top with an R^2 of 0.878.

```
Linear Regression R_squared =  0.7799189130578416
Linear Regression RMSE =  4.904241528656867

Random Forest Regressor R_squared =  0.948699033441263
Random Forest Regressor RMSE =  2.0776807299041447

Extra-Trees Regressor R_squared =  0.9757791138600801
Extra-Trees Regressor RMSE =  1.4276148341287662
```

Results without columns dropped on Training Set 1-9

```
Linear Regression R_squared =  0.748352165708644
Linear Regression RMSE =  4.35223867453816

Random Forest Regressor R_squared =  0.8653145231674304
Random Forest Regressor RMSE =  3.3664829714165996

Extra-Trees Regressor R_squared =  0.8848625751947294
Extra-Trees Regressor RMSE =  3.1126070284945997
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                     mv   R-squared (uncentered):          0.947
Model:                            OLS   Adj. R-squared (uncentered):     0.946
Method:                 Least Squares   F-statistic:                     737.3
Date:                Fri, 09 Oct 2020   Prob (F-statistic):         2.85e-306
Time:                        18:43:57   Log-Likelihood:                -1589.1
No. Observations:                 506   AIC:                             3202.
Df Residuals:                     494   BIC:                             3253.
Df Model:                          12
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
crim           1.2428      2.645      0.470      0.639      -3.954       6.440
zn             1.6288      0.825      1.975      0.049       0.009       3.249
indus          7.3184      2.025      3.614      0.000       3.339      11.298
chas           2.7376      1.030      2.658      0.008       0.714       4.761
nox            3.1374      2.468      1.271      0.204      -1.711       7.986
rooms         38.4414      2.101     18.295      0.000      34.313      42.570
age            3.6980      1.521      2.431      0.015       0.710       6.686
dis            9.3141      1.799      5.176      0.000       5.779      12.850
rad            4.2205      1.906      2.214      0.027       0.476       7.965
tax           -6.7608      1.656     -4.082      0.000     -10.015      -3.506
ptratio       -4.0290      1.363     -2.956      0.003      -6.707      -1.351
lstat        -16.2658      2.093     -7.771      0.000     -20.378     -12.153
==============================================================================
Omnibus:                      172.493   Durbin-Watson:                   0.896
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1090.183
Skew:                           1.336   Prob(JB):                    1.86e-237
Kurtosis:                       9.676   Cond. No.                         23.0
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Test Data Results W/O Columns Dropped 1-10

P Value Table 1-11

```
X = X.drop(['crim', 'zn', 'age'], 1)
```

```
X = X.drop(['chas', 'rad', 'ptratio'], 1)
```

Dropped Features with High P-Values 1-12

Left P-Values of 0 in 1-13.

After looking at the results I wanted to see the feature importance of the Extra Trees and Linear Regression for when they performed the best, which was when no features were dropped. I looked particularly at the Train Data/Test Data feature coefficients. For Linear Regression I got the following Train /Test Data formula:

45.6 + 0.49*crim + 0.30*zn -2.93*indus + 2.06*chas – 8.38*nox + 12.42*rooms + 2.42*age - 14.72*dis + 3.59*rad -5.96*tax – 4.64*ptratio – 29.74*lstat

Train Data Formula 1-14

```
[  0.49836874    0.30405808  -2.93067293    2.06843942  -8.38263356
  12.42796078    2.42287747 -14.71996785    3.59660469  -5.95831525
  -4.63513041 -29.74021908]
45.63986505610792
```

```
array([0.03434324, 0.00747711, 0.03959489, 0.00766035, 0.03439174,
       0.27874734, 0.02657861, 0.04776678, 0.01110838, 0.03148018,
       0.04489207, 0.43595932])
```

Train Data Coefficients 1-15

Train Data Feature Importance 1-16

38.8 + 11.21*crim + 1.33*zn -1.89*indus + 3.71*chas – 11.04*nox + 23.94*rooms - 2.34*age - 18.44*dis + 3.41*rad -9.04*tax – 7.68*ptratio - 21.9*lstat

Test Data Formula 1-17

```
[ 11.21073066    1.32822039  -1.89162496    3.71932116 -11.04283759
  23.94308498   -2.34720544 -18.44677227    3.41590743  -9.04467456
  -7.68466112 -21.91713556]
38.80670993301582
```

```
array([0.03434324, 0.00747711, 0.03959489, 0.00766035, 0.03439174,
       0.27874734, 0.02657861, 0.04776678, 0.01110838, 0.03148018,
       0.04489207, 0.43595932])
```

Coefficients and Intercept for Linear Regression Test Data 1-18 *Feature importance Test Data for Extra Trees 1-19*

The Linear Regression formula for Training data tells us that **lstat** has the biggest negative influence on median house values and tells us the target value falls by -29.74 with every increase in lstat. The other big negative weight I see is with **dis** which shows that with every increase in this there is a -14.72 decrease in the target variable. I also see rooms which seems to have the biggest positive weight. What this says is that with every increase in a room the median value increases by 12.42.

The Linear Regression formula for test data tells us that **lstat** has the biggest negative influence on median house values and tells us the target value falls by -21.9 with every increase in lstat. The other big negative weight I see is with **dis** which shows that with every increase in this there is a -18.44 decrease in the target variable. I also see rooms which seems to have the biggest positive weight. What this says is that with every increase in a room the median value increases by 23.94.

For interpreting the extra trees matrix, you have to see the largest number to find the most important feature. It seems from the matrix that the most important features are **rooms** and **lstats**. Rooms has an importance of 28 percent while lstat has one of 43.6 percent. The least important ones are **zn** with importance of 0.7 percent and **chas** with 0.7 percent.

Implementation and Programming

Before implementation it is important to import packages such as sklearn, pandas, matplotlib and stats. After initial importing we start our data prep phase, which will involve using **.describe() method** from pandas package. This is important to get basic statistics of each column of Pandas Dataframe. It is also important to look at the **.dtypes** to look at variable types for each column to get a sense of the data. To see the data without doing anything on it one can use **.head()** method on the dataframe to get initial rows and columns of the data. Using **isnull.sum()** on the dataset allows one to find null values for imputation purposes. After seeing the initial data, we can visualize each feature by using **matplotlib.** This is useful to use different visualizations such as **boxplot()** method to get a sense of how skewed the data is. One can also use **matplotlib.plot()** function to get scatterplots and line plots. **.Hist() function** is another good visual to plot a histogram. When one wants to see relationships of features to the response variable used to predict on they can use a for loop, and make subplots using the **subplot() and scatter() functions.** It is also important to use **boxcox()** on select features that do not have linear relationships with the target variable. This is why the stats package is used. It is also important to scale features to compare features. In order to scale features the following code can be used: **boston_df=boston_df.transform(lambda x: (x - x.min()) / (x.max() - x.min())).** **Drop** function is also used to drop unnecessary features before training such as variables that a multicollinear when seeing the heatmap depicted in 1-8 . When its time to start modeling it is important to save your response variable using this format: **y = "response column".** Using **train_test.split()** one can split data into training set and test set. From here one can instantiate models such as **LinearRegression(), RandomForestRegressor(), and ExtraTreeRegressor().** Once instantiated they can call attributes such as **.coef** for coefficients for Linear Regression or functions such as **.fit()** or **.score()** for R^2.

```
#imported seaborn; matplotlib
import matplotlib
import numpy as np
import pandas as pd
import os
import itertools
from math import sqrt
from scipy import stats as st
#import cvxopt

import sklearn
from sklearn.preprocessing import StandardScaler # used for variable scaling data
from sklearn.preprocessing import MinMaxScaler as Scaler # used for variable scaling data
from sklearn.model_selection import train_test_split

import sklearn.linear_model
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.ensemble import RandomForestRegressor # Random Forest package
from sklearn.ensemble import ExtraTreesRegressor # Extra Trees package
from sklearn.ensemble import GradientBoostingRegressor # Gradient Boosting package

from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import make_scorer, accuracy_score

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold

import statsmodels.api as sm

import matplotlib.pyplot as plt

from collections import OrderedDict
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier

from matplotlib import pyplot as plt
from matplotlib import rc
import seaborn as sns
sns.set_style("whitegrid")
sns.set(style="whitegrid", color_codes=True)
plt.rc('font', size=14)
```

```
features = boston_df1.drop('mv', 1).columns
target = boston_Target
plt.figure(figsize=(20,20))
for index, feature_name in enumerate(features):
    plt.subplot(4,len(features)/2, index+1)
    plt.scatter(boston_df1[feature_name], target)
    plt.title(feature_name, fontsize=15)
    plt.xlabel(feature_name, fontsize=8)
    plt.ylabel('mv', fontsize=15)
```

Packages to Import 1-20 *Scatterplot matrix for finding relationships. 1-21*

Exposition, problem description and management recommendations

After examining the Boston dataset, it is clear that **Nox** does not have a significant influence on median value of a house. Two ways one can see this is by looking at the p value for Nox which is at 0.20 so it is nonsignificant since it above 0.05. Another way is to look at Extra Trees feature importance which marks Nox as 3.4% important. Therefore, Nox is not a good indicator of median value. For this Boston Dataset, the best model was Extra Trees algorithm. It was the best do to it's R^2 score which stayed relatively the same after dropping columns from the dataset. I also tried to use OOB score to see if Random Forest Regressor performed well, but it actually was overfitting in the training dataset as the OOB Score got significantly worse on the test set. For example, with no columns dropped the OOB score was around 0.8511 for fitting the training data, but for the test data the score was only 0.769. Therefore, it showed how much worse it was doing using the test data. In conclusion, **I recommend Extra Trees to management** as it did well on the R^2 metric in all three trials which involved dropped features while the linear regression did significantly worst, and Random Trees did not perform well on OOB score.

References

[1] Géron, A. *Hands-On Machine Learning with Scikit-Learn & TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* 2d Edition. Sebastopol, Calif.: O'Reilly. [ISBN 9781492032649], 2019.

Appendix

Import Packages

I imported seaborn, matplotlib, numpy and pandas

In [1]:

```
#imported seaborn; matplotlib
import matplotlib
import numpy as np
import pandas as pd
import os
import itertools
from math import sqrt
from scipy import stats as st
#import cvxopt

import sklearn
from sklearn.preprocessing import StandardScaler # used for
from sklearn.preprocessing import MinMaxScaler as Scaler #
from sklearn.model_selection import train_test_split

import sklearn.linear_model
from sklearn.linear_model import LinearRegression, Ridge, L
from sklearn.ensemble import RandomForestRegressor # Random
from sklearn.ensemble import ExtraTreesRegressor # Extra Tr
from sklearn.ensemble import GradientBoostingRegressor # Gr

from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import make_scorer, accuracy_score

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold

import statsmodels.api as sm

import matplotlib.pyplot as plt

from collections import OrderedDict
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier

from matplotlib import pyplot as plt
from matplotlib import rc
import seaborn as sns
sns.set_style("whitegrid")
sns.set(style="whitegrid", color_codes=True)
plt.rc("font", size=14)
```

In [2]:

```
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

Read in data from CSV file

In [3]:

```
#import dataset
boston_df=pd.read_csv('https://raw.githubusercontent.com/dj
```

Get shape of data

In [4]:

```
#calculate shape
boston_df.shape
```

Out[4]:

```
(506, 14)
```

See the initial data

In [5]:

```
boston_df.head()
```

Out[5]:

	neighborhood	crim	zn	indus	chas	nox	rooms	ag
0	Nahant	0.00632	18.0	2.31	0	0.538	6.575	65
1	Swampscott	0.02731	0.0	7.07	0	0.469	6.421	78
2	Swanpscott	0.02729	0.0	7.07	0	0.469	7.185	61
3	Marblehead	0.03237	0.0	2.18	0	0.458	6.998	45
4	Marblehead	0.06905	0.0	2.18	0	0.458	7.147	54

See the data types of columns

In [6]:

```
boston_df.dtypes
```

Out[6]:

```
neighborhood     object
crim            float64
zn              float64
indus           float64
chas              int64
nox             float64
rooms           float64
age             float64
dis             float64
rad               int64
tax               int64
ptratio         float64
lstat           float64
mv              float64
dtype: object
```

Drop the not quantitative data

In [7]:

```
#Drop non numeric columns
boston_df = boston_df.drop('neighborhood', 1)
```

See the data again with column dropped

In [8]:

```
#see if column has been dropped
boston_df.head()
```

Out[8]:

	crim	zn	indus	chas	nox	rooms	age	dis	rad
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3

Run the traditional statistics to summarize data

In [9]:

```
#check to see if there are typos and look at typical stats
boston_df.describe()
```

Out[9]:

	crim	zn	indus	chas	I
count	506.000000	506.000000	506.000000	506.000000	506.000
mean	3.613524	11.363636	11.136779	0.069170	0.554
std	8.601545	23.322453	6.860353	0.253994	0.115
min	0.006320	0.000000	0.460000	0.000000	0.385
25%	0.082045	0.000000	5.190000	0.000000	0.449
50%	0.256510	0.000000	9.690000	0.000000	0.538
75%	3.677082	12.500000	18.100000	0.000000	0.624
max	88.976200	100.000000	27.740000	1.000000	0.871

See if there is any missing data

In [10]:

```
#look at data and check if there is NA values
boston_df.isnull().sum()
```

Out[10]:

```
crim       0
zn         0
indus      0
chas       0
nox        0
rooms      0
age        0
dis        0
rad        0
tax        0
ptratio    0
lstat      0
mv         0
dtype: int64
```

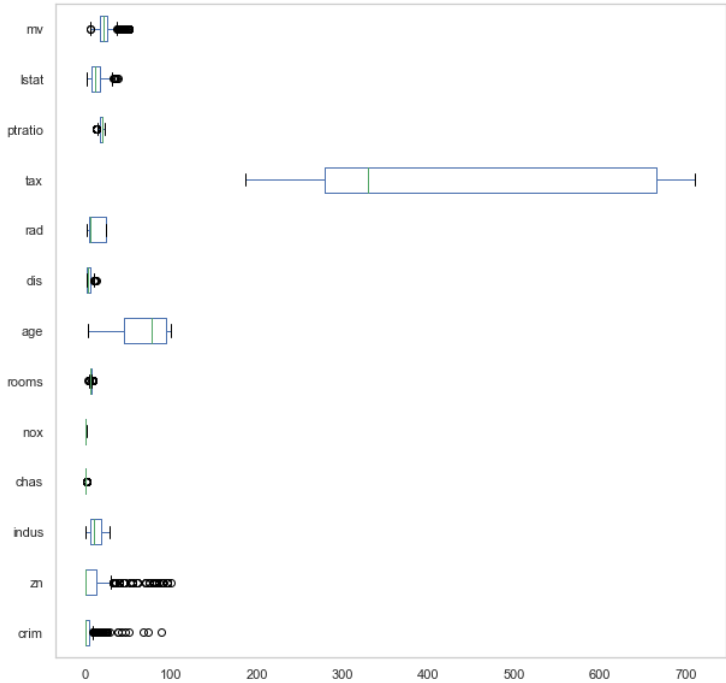
See if there is any outliers in the data

In [11]:

```
boston_df.boxplot(vert=False, figsize=(10,10), grid=False)
```

Out[11]:

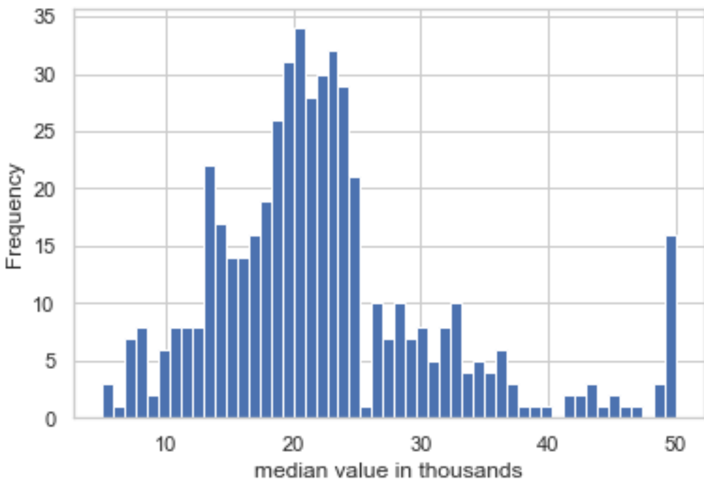
```
<AxesSubplot:>
```



Make a histograms and see median values of house prices in each neigborhood

In [12]:

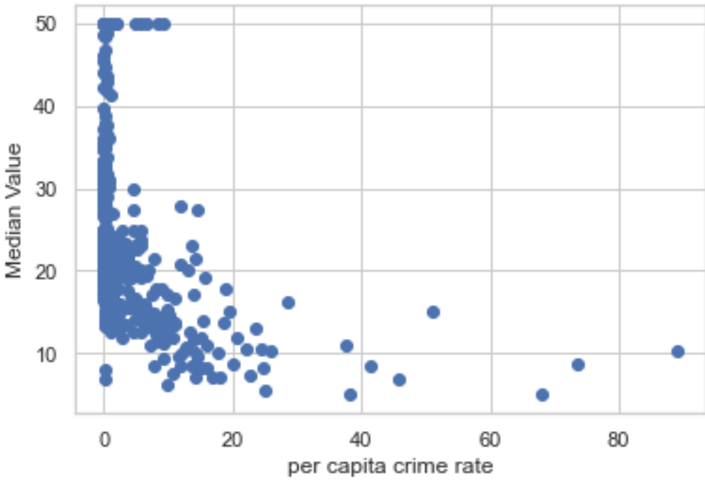
```
plt.hist(boston_df['mv'], bins = 51)
plt.xlabel('median value in thousands')
plt.ylabel('Frequency')
plt.show()
```



Wanted to see which data had linear shape

In [13]:

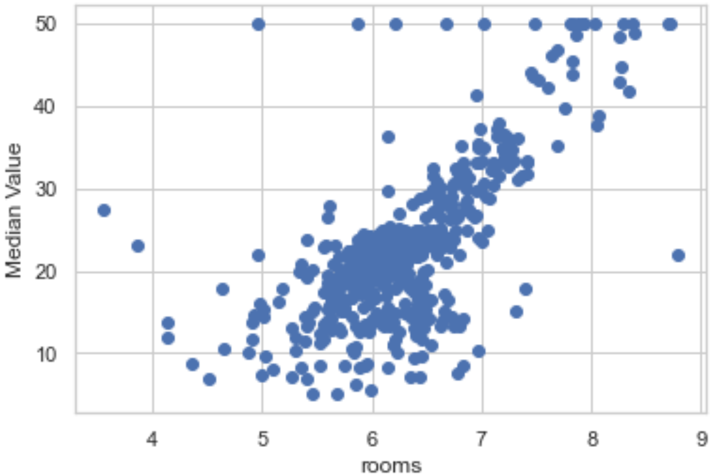
```
plt.plot(boston_df['crim'], boston_df['mv'], 'bo')
plt.xlabel('per capita crime rate')
plt.ylabel('Median Value')
plt.show()
```



Saw rooms column and saw a liner shape which means positive correlation

In [14]:

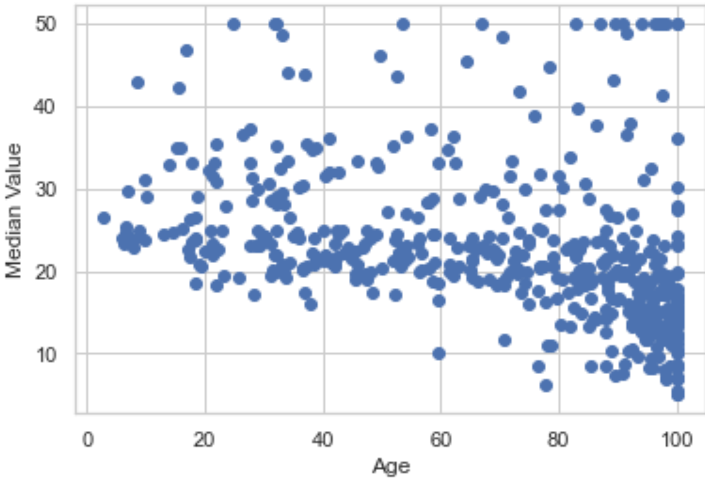
```
plt.plot(boston_df['rooms'], boston_df['mv'], 'bo')
plt.xlabel('rooms')
plt.ylabel('Median Value')
plt.show()
```



Do not see any strong correlation

In [15]:

```
plt.plot(boston_df['age'], boston_df['mv'], 'bo')
plt.xlabel('Age')
plt.ylabel('Median Value')
plt.show()
```



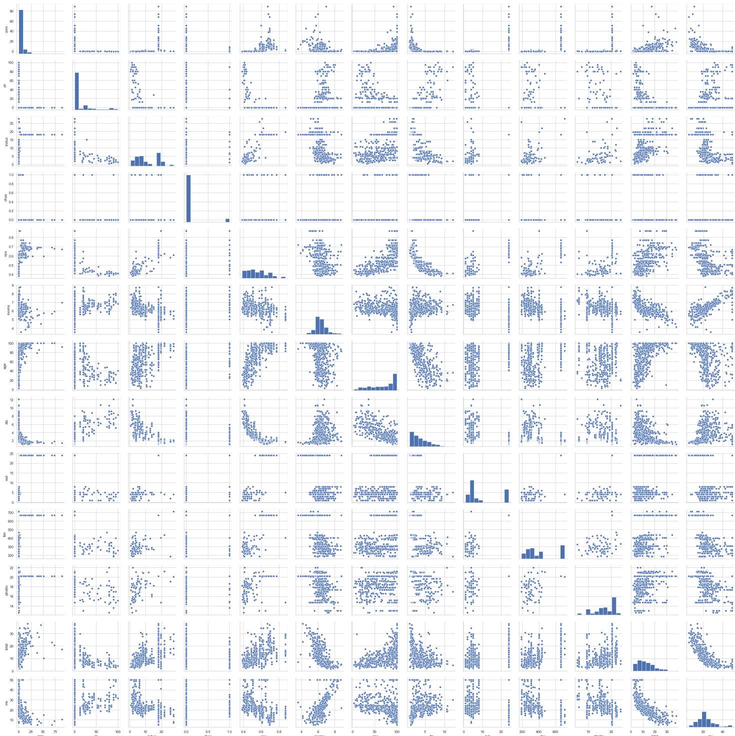
I see initial data in graphs in this pairplot

In [16]:

```
sns.pairplot(boston_df, diag_kind='hist')
```

Out[16]:

<seaborn.axisgrid.PairGrid at 0x1c27018510>



Make copy of data

In [17]:

```
boston_df1=boston_df.copy()
```

Drop all columns except for target column

In [18]:

```
#saves the column we want to predict
columns = ['crim','zn','indus','chas','nox','rooms','age','
boston_Target = boston_df1.drop(columns=columns)
```

In [19]:

```
boston_Target
```

Out[19]:

	mv
0	24.0
1	21.6
2	34.7
3	33.4
4	36.2
...	...
501	22.4
502	20.6
503	23.9
504	22.0
505	19.0

506 rows × 1 columns

See shape again should only have 1 column, 506 rows

In [20]:

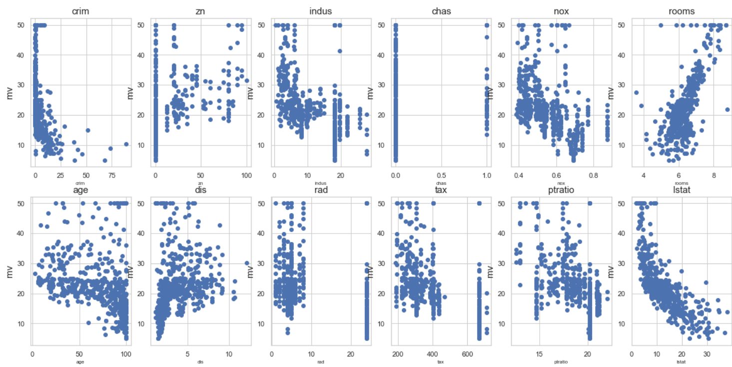
```
#makes sure shape is right
boston_Target.shape
```

Out[20]:

```
(506, 1)
```

In [21]:

```
features = boston_df1.drop('mv', 1).columns
target = boston_Target
plt.figure(figsize=(20,20))
for index, feature_name in enumerate(features):
    plt.subplot(4,len(features)/2, index+1)
    plt.scatter(boston_df1[feature_name], target)
    plt.title(feature_name, fontsize=15)
    plt.xlabel(feature_name, fontsize=8)
    plt.ylabel('mv', fontsize=15)
```



In [22]:

```
boston_df1
```

Out[22]:

	crim	zn	indus	chas	nox	rooms	age	dis	r
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	
...	...	...	...	...	...	...	...	...	
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	

506 rows × 13 columns

Make lambda function to add .01 to all columns with zeros; then use boxcox to make more normal distribution to columns which are non-linear and make more linear for linear regression

In [23]:

```
#need to transform data to find linear relationship
boston_df=boston_df.apply(lambda x: x+.01)
boston_df[['age', 'zn', 'crim', 'ptratio', 'chas', 'indus',
```

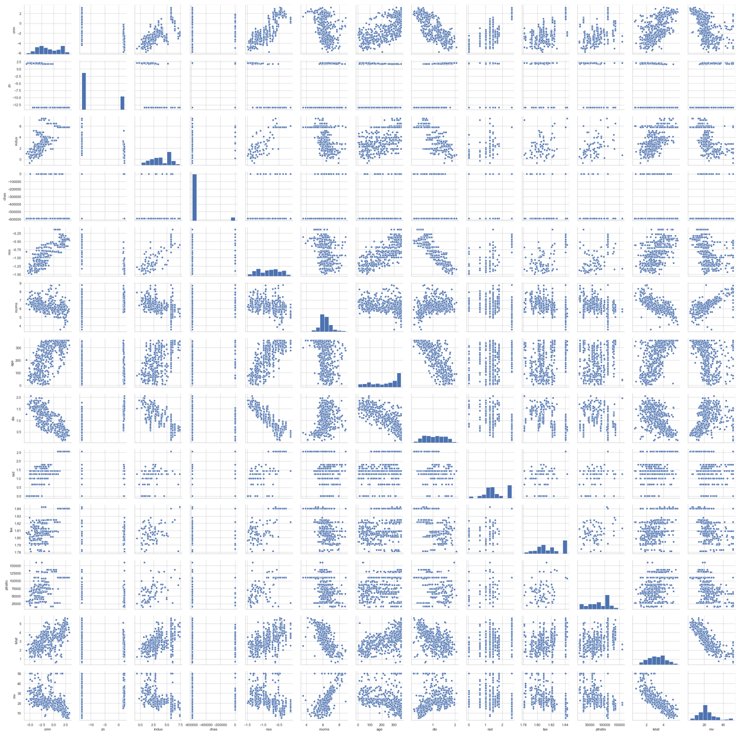
See pairplot again

In [24]:

```
sns.pairplot(boston_df, diag_kind='hist')
```

Out[24]:

<seaborn.axisgrid.PairGrid at 0x1c2aef8d90>



Use scaler to make everythiing with 0 and 1 using min max scaler

In [25]:

```
#scale data by min max scaler
boston_df=boston_df.transform(lambda x: (x - x.min()) / (x.
```

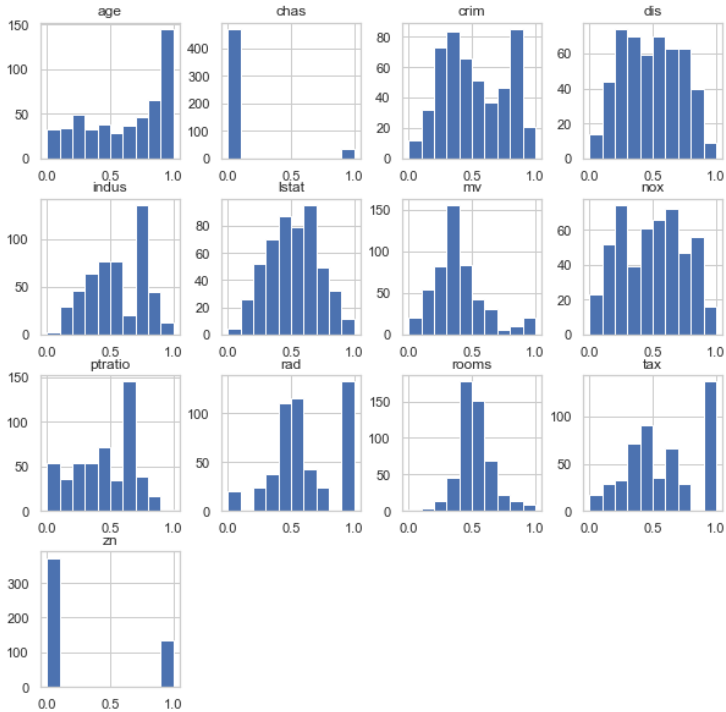
Make histogram matrix and as you can see everything is within 0 and 1.

In [26]:

```
boston_df.hist(figsize=(10,10))
```

Out[26]:

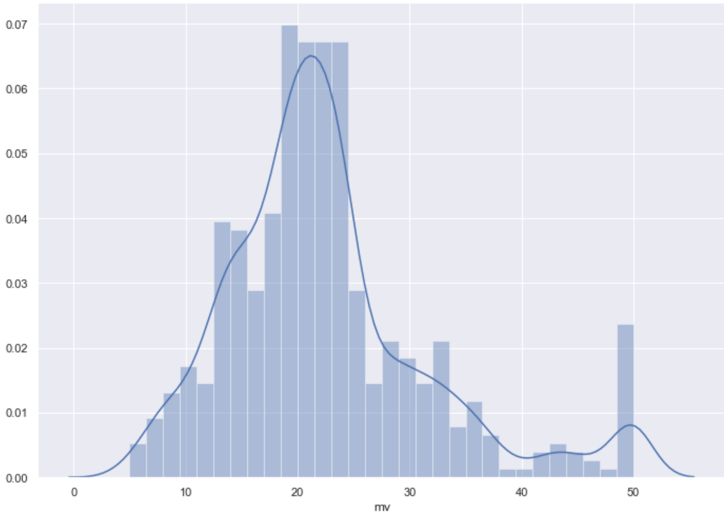
```
array([[<AxesSubplot:title={'center':'age'}>,
        <AxesSubplot:title={'center':'chas'}>,
        <AxesSubplot:title={'center':'crim'}>,
        <AxesSubplot:title={'center':'dis'}>],
       [<AxesSubplot:title={'center':'indus'}
>,
        <AxesSubplot:title={'center':'lstat'}
>,
        <AxesSubplot:title={'center':'mv'}>,
        <AxesSubplot:title={'center':'nox'}>],
       [<AxesSubplot:title={'center':'ptrati
o'}>,
        <AxesSubplot:title={'center':'rad'}>,
        <AxesSubplot:title={'center':'rooms'}
>,
        <AxesSubplot:title={'center':'tax'}>],
       [<AxesSubplot:title={'center':'zn'}>, <
AxesSubplot:>,
        <AxesSubplot:>, <AxesSubplot:>]], dtyp
e=object)
```



Did not have to scale median value of house variable

In [27]:

```
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.distplot(boston_Target['mv'], bins=30)
plt.show()
```



See data scaled and transformed and perform tradition summary statistics; move target variable to front

In [28]:

```
cols = boston_df.columns.tolist()
cols = cols[-1:] + cols[:-1]
boston_df4=boston_df[cols]
boston_df4.describe(include="all")
```

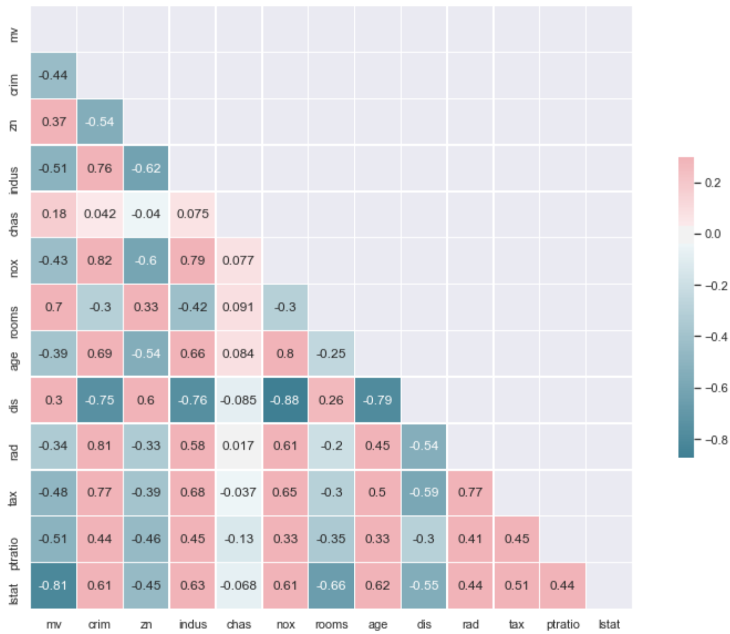
Out[28]:

	mv	crim	zn	indus	cl
count	506.000000	506.000000	506.000000	506.000000	506.000
mean	0.389530	0.518606	0.261061	0.562812	0.069
std	0.204048	0.247164	0.435430	0.232825	0.253
min	0.000000	0.000000	0.000000	0.000000	0.000
25%	0.267222	0.319026	0.000000	0.378568	0.000
50%	0.360000	0.476717	0.000000	0.559586	0.000
75%	0.444444	0.771394	0.967068	0.796857	0.000
max	1.000000	1.000000	1.000000	1.000000	1.000

Make correlation heat map

In [29]:

```
#check correlations
plt.figure(figsize=(15,10))
corr=boston_df4.corr(method='pearson')
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
sns.heatmap(corr, mask=mask, cmap=sns.diverging_palette(220
              square=True, linewidths=.5, cbar_kws={"shrink":
plt.show()
```



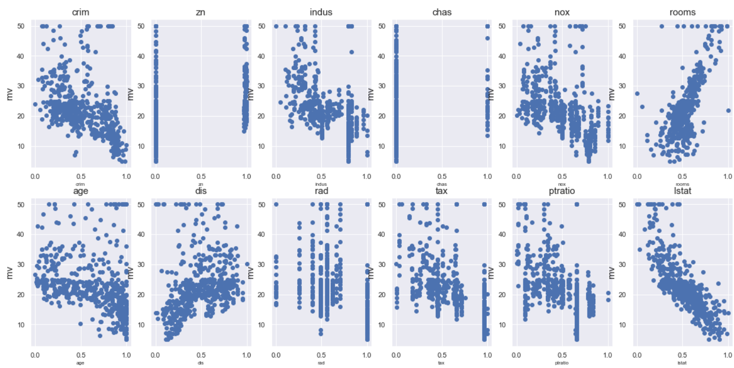
In [30]:

```
boston_df6=boston_df4.copy()
boston_df5 = boston_df4
```

See features scaled and see their dot graph

In [31]:

```
features = boston_df5.drop('mv', 1).columns
target = boston_Target
plt.figure(figsize=(20,20))
for index, feature_name in enumerate(features):
    plt.subplot(4,len(features)/2, index+1)
    plt.scatter(boston_df5[feature_name], target)
    plt.title(feature_name, fontsize=15)
    plt.xlabel(feature_name, fontsize=8)
    plt.ylabel('mv', fontsize=15)
```



Drop taget variable since we already have it save in it's own df

In [32]:

```
featuresdf = boston_df5.drop('mv', 1)
featuresdf
```

Out[32]:

	crim	zn	indus	chas	nox	rooms
0	0.000000	0.974993	0.206111	0.0	0.500909	0.577505
1	0.163141	0.000000	0.462097	0.0	0.313594	0.547998
2	0.163042	0.000000	0.462097	0.0	0.313594	0.694386
3	0.186434	0.000000	0.195938	0.0	0.278772	0.658555
4	0.294254	0.000000	0.195938	0.0	0.278772	0.687105
...	...	...	...	...	...	...
501	0.280213	0.000000	0.631565	0.0	0.579388	0.580954
502	0.233660	0.000000	0.631565	0.0	0.579388	0.490324
503	0.275852	0.000000	0.631565	0.0	0.579388	0.654340
504	0.360297	0.000000	0.631565	0.0	0.579388	0.619467
505	0.240252	0.000000	0.631565	0.0	0.579388	0.473079

506 rows × 12 columns

In [182]:

```
#remove target and keep all features independent variables
X = featuresdf
```

In [183]:

```
# Save target in Y
Y = boston_Target['mv']
```

Run initial model with Ordinary Least Squares Linear Regression befor splitting.

In [184]:

```
model=sm.OLS(Y, X)
```

In [185]:

```
#save learned algorithm
results=model.fit()
```

Saw R^2 is pretty high means it overfits to existing data

In [186]:

```
print(results.summary())
```

```
                                 OLS Regressio
n Results
==============================================
=========================================
Dep. Variable:                     mv   R-squa
red (uncentered):                    0.947
Model:                            OLS   Adj. R
-squared (uncentered):               0.946
Method:                 Least Squares   F-stat
istic:                               737.3
Date:                Fri, 09 Oct 2020   Prob
(F-statistic):                    2.85e-306
Time:                        20:45:43   Log-Li
kelihood:                          -1589.1
No. Observations:                 506   AIC:
3202.
Df Residuals:                     494   BIC:
3253.
Df Model:                          12
Covariance Type:            nonrobust
==============================================
================================
                 coef    std err          t
P>|t|      [0.025      0.975]
----------------------------------------------
--------------------------------
crim           1.2428      2.645      0.470
0.639      -3.954       6.440
zn             1.6288      0.825      1.975
0.049       0.009       3.249
indus          7.3184      2.025      3.614
0.000       3.339      11.298
chas           2.7376      1.030      2.658
0.008       0.714       4.761
nox            3.1374      2.468      1.271
0.204      -1.711       7.986
rooms         38.4414      2.101     18.295
0.000      34.313      42.570
age            3.6980      1.521      2.431
0.015       0.710       6.686
```

```
dis               9.3141       1.799       5.176
0.000        5.779       12.850
rad               4.2205       1.906       2.214
0.027        0.476        7.965
tax              -6.7608       1.656      -4.082
0.000      -10.015       -3.506
ptratio          -4.0290       1.363      -2.956
0.003       -6.707       -1.351
lstat           -16.2658       2.093      -7.771
0.000      -20.378      -12.153
==============================================
================================
Omnibus:                          172.493   Durbin
-Watson:                       0.896
Prob(Omnibus):                      0.000   Jarque
-Bera (JB):                 1090.183
Skew:                               1.336   Prob(J
B):                        1.86e-237
Kurtosis:                           9.676   Cond.
No.                              23.0
==============================================
================================

Warnings:
[1] Standard Errors assume that the covariance
matrix of the errors is correctly specified.
```

Train with all features involved

In [211]:

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, t
```

In [212]:

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(404, 12)
(102, 12)
(404,)
(102,)
```

In [213]:

```
lrm = LinearRegression()

# Fit data on to the model
lrm.fit(X_train, y_train)

# Predict
y_predicted_lrm = lrm.predict(X_test)
```

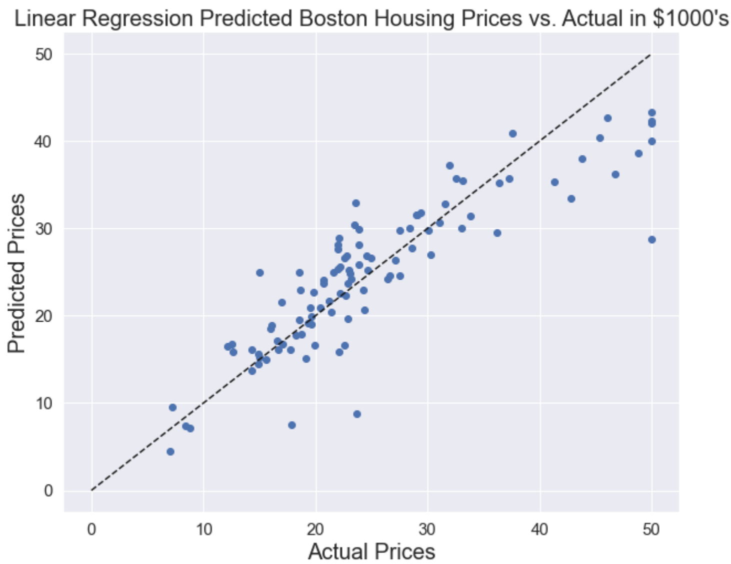
In [214]:

```
print(lrm.coef_)
print(lrm.intercept_)
```

```
[  0.49836874   0.30405808  -2.93067293   2.06
843942  -8.38263356
  12.42796078   2.42287747 -14.71996785   3.59
660469  -5.95831525
  -4.63513041 -29.74021908]
45.63986505610792
```

In [191]:

```
plt.figure(figsize=(10,8))
plt.scatter(y_test, y_predicted_lrm)
plt.plot([0, 50], [0, 50], '--k')
plt.axis('tight')
plt.ylabel('Predicted Prices', fontsize=20);
plt.xlabel('Actual Prices', fontsize=20);
plt.title("Linear Regression Predicted Boston Housing Price

plt.rc('xtick', labelsize=15)
plt.rc('ytick', labelsize=15)

plt.show()
```



In [192]:

```
print("Linear Regression R_squared = ",lrm.score(X,Y))
pred= lrm.predict(X_test)
rmse = sqrt(mean_squared_error(pred, y_test))
print('Linear Regression RMSE = ', rmse)
```

```
Linear Regression R_squared =  0.7799189130578
416
Linear Regression RMSE =  4.904241528656867
```

In [134]:

```
# Seed value for random number generators to obtain reprodu
RANDOM_SEED = 1

# The model input data outside of the modeling method calls
names = ['Linear_Regression']

# Specify the set of regression models being evaluated (we
regressors = [LinearRegression(fit_intercept = True, normal
              Ridge(alpha = 75, solver = 'cholesky', fit_in
              Lasso(alpha = 0.01, max_iter=10000, tol=0.01,
              ElasticNet(alpha = 0.01, l1_ratio = 0.5, max_
              ]
```

Now we want to same thing with some columns dropped with values higher than 0.05 and variables that involve colinearlity but including nox

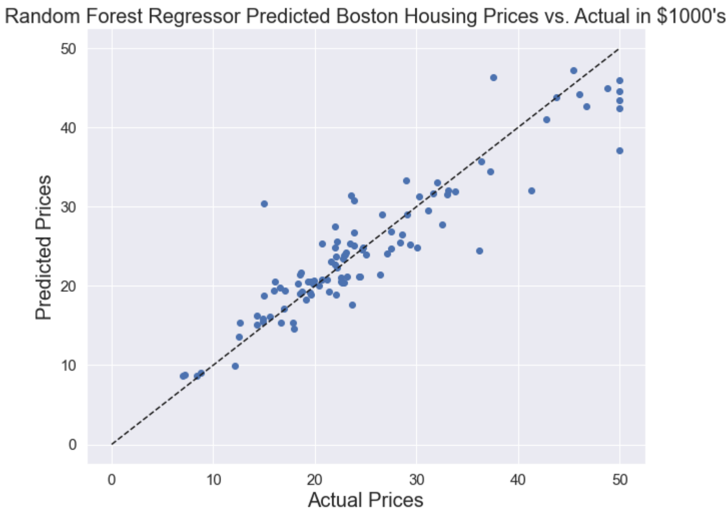
In [198]:

```
Randreg = RandomForestRegressor(oob_score = True)

# Fit data on to the model
Randreg.fit(X_train, y_train)
print(Randreg.oob_score_)
# Predict
y_predicted_Randreg = Randreg.predict(X_test)
```

```
0.8469227952496666
```

In [136]:

```
plt.figure(figsize=(10,8))
plt.scatter(y_test, y_predicted_Randreg)
plt.plot([0, 50], [0, 50], '--k')
plt.axis('tight')
plt.ylabel('Predicted Prices', fontsize=20);
plt.xlabel('Actual Prices', fontsize=20);
plt.title("Random Forest Regressor Predicted Boston Housing

plt.rc('xtick', labelsize=15)
plt.rc('ytick', labelsize=15)

plt.show()
```



In [137]:

```
print("Random Forest Regressor R_squared = ",Randreg.score(
pred= Randreg.predict(X)
rmse = sqrt(mean_squared_error(pred, Y))
print('Random Forest Regressor RMSE = ', rmse)
```

```
Random Forest Regressor R_squared =  0.9486990
33441263
Random Forest Regressor RMSE =  2.077680729904
1447
```

In [138]:

```
print(Randreg.decision_path(X))
```

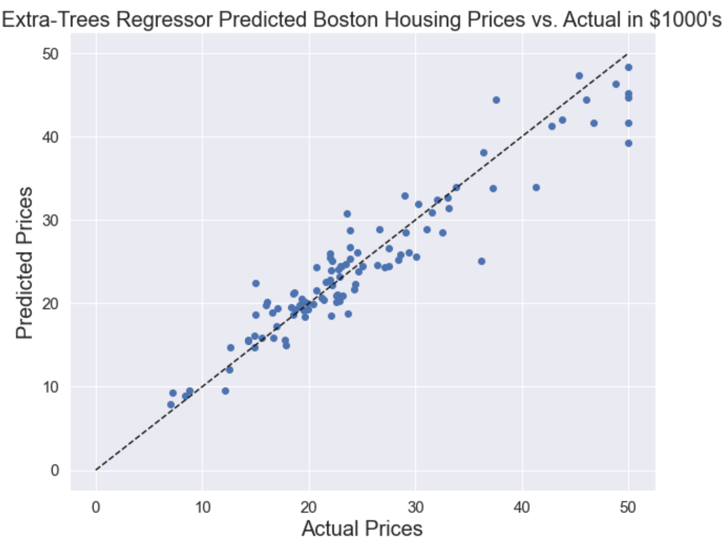
```
(<506x48614 sparse matrix of type '<class 'num
py.int64'>'
        with 562830 stored elements in Compres
sed Sparse Row format>, array([     0,    487,
964,   1427,   1926,   2411,   2892,   3359,   3840,
        4313,   4792,   5289,   5752,   6253,   672
6,   7217,   7700,   8173,
        8650,   9143,   9646,  10133,  10598,  1108
5,  11586,  12083,  12558,
       13055,  13538,  14019,  14512,  14977,  1547
6,  15951,  16456,  16921,
       17396,  17873,  18362,  18845,  19348,  1984
1,  20318,  20815,  21296,
       21795,  22300,  22787,  23298,  23781,  2427
8,  24769,  25262,  25745,
       26234,  26745,  27208,  27699,  28162,  2865
1,  29130,  29615,  30100,
       30579,  31068,  31553,  32046,  32523,  3301
6,  33499,  33992,  34459,
       34942,  35411,  35884,  36353,  36850,  3734
5,  37850,  38343,  38824,
       39325,  39800,  40289,  40772,  41261,  4176
6,  42249,  42726,  43241,
       43696,  44211,  44702,  45203,  45694,  4619
7,  46672,  47167,  47644,
       48129,  48614]))
```

In [139]:

```
ETreg = ExtraTreesRegressor()

# Fit data on to the model
ETreg.fit(X_train, y_train)

# Predict
y_predicted_ETreg = ETreg.predict(X_test)
```

In [140]:

```
plt.figure(figsize=(10,8))
plt.scatter(y_test,y_predicted_ETreg)
plt.plot([0, 50], [0, 50], '--k')
plt.axis('tight')
plt.ylabel('Predicted Prices', fontsize=20);
plt.xlabel('Actual Prices', fontsize=20);
plt.title("Extra-Trees Regressor Predicted Boston Housing P

plt.rc('xtick', labelsize=15)
plt.rc('ytick', labelsize=15)

plt.show()
```



In [141]:

```
print("Extra-Trees Regressor R_squared = ",ETreg.score(X,Y))
pred= ETreg.predict(X)
rmse = sqrt(mean_squared_error(pred, Y))
print('Extra-Trees Regressor RMSE = ', rmse)
```

```
Extra-Trees Regressor R_squared =  0.975779113
8600801
Extra-Trees Regressor RMSE =  1.42761483412876
62
```

In [142]:

```
ETreg.feature_importances_
```

Out[142]:

```
array([0.03434324, 0.00747711, 0.03959489, 0.0
0766035, 0.03439174,
       0.27874734, 0.02657861, 0.04776678, 0.0
1110838, 0.03148018,
       0.04489207, 0.43595932])
```

In [209]:

```
lrm = LinearRegression()

# Fit data on to the model
lrm.fit(X_test, y_test)
```

Out[209]:

```
LinearRegression()
```

In [210]:

```
print("Linear Regression R_squared = ",lrm.score(X,Y))
pred= lrm.predict(X_test)
rmse = sqrt(mean_squared_error(pred, y_test))
print('Linear Regression RMSE = ', rmse)

print(lrm.coef_)
print(lrm.intercept_)
```

```
Linear Regression R_squared =  0.7483521657086
44
Linear Regression RMSE =  4.35223867453816
[ 11.21073066   1.32822039  -1.89162496   3.71
932116 -11.04283759
  23.94308498  -2.34720544 -18.44677227   3.41
590743  -9.04467456
  -7.68466112 -21.91713556]
38.80670993301582
```

In [201]:

```
Randreg = RandomForestRegressor(oob_score = True)

# Fit data on to the model
Randreg.fit(X_test, y_test)
print(Randreg.oob_score_)
```

```
0.7561275016777594
```

In [146]:

```
print("Random Forest Regressor R_squared = ",Randreg.score(
pred= Randreg.predict(X)
rmse = sqrt(mean_squared_error(pred, Y))
print('Random Forest Regressor RMSE = ', rmse)

print(Randreg.decision_path(X))
```

```
Random Forest Regressor R_squared =  0.8653145
231674304
Random Forest Regressor RMSE =  3.366482971416
5996
(<506x12362 sparse matrix of type '<class 'num
py.int64'>'
        with 420508 stored elements in Compres
sed Sparse Row format>, array([    0,   127,
254,   375,   500,   615,   744,   879,   996,
        1113,  1218,  1345,  1476,  1607,  173
0,  1865,  1994,  2125,
        2234,  2351,  2482,  2615,  2732,  285
7,  2986,  3119,  3244,
        3375,  3494,  3607,  3730,  3855,  398
0,  4097,  4230,  4359,
        4480,  4603,  4736,  4867,  4990,  510
7,  5238,  5367,  5486,
        5609,  5730,  5857,  5968,  6093,  621
0,  6331,  6452,  6583,
        6716,  6827,  6948,  7083,  7198,  732
7,  7448,  7579,  7702,
        7825,  7940,  8061,  8176,  8295,  842
6,  8543,  8674,  8799,
        8920,  9049,  9170,  9281,  9408,  953
5,  9662,  9789,  9910,
       10025, 10148, 10271, 10396, 10533, 1065
4, 10765, 10892, 11011,
       11122, 11249, 11378, 11497, 11630, 1175
5, 11880, 11999, 12118,
       12243, 12362]))
```

In [147]:

```
ETreg = ExtraTreesRegressor()

# Fit data on to the model
ETreg.fit(X_test, y_test)
```

Out[147]:

```
ExtraTreesRegressor()
```

In [148]:

```
print("Extra-Trees Regressor R_squared = ",ETreg.score(X,Y))
pred= ETreg.predict(X)
rmse = sqrt(mean_squared_error(pred, Y))
print('Extra-Trees Regressor RMSE = ', rmse)

ETreg.feature_importances_
```

```
Extra-Trees Regressor R_squared =  0.884862575
1947294
Extra-Trees Regressor RMSE =  3.11260702849459
97
```

Out[148]:

```
array([0.02891771, 0.00740353, 0.02269477, 0.0
3461415, 0.04545708,
       0.36245977, 0.01947177, 0.03958159, 0.0
2482691, 0.03431309,
       0.04720921, 0.33305042])
```

In [149]:

```
model_data=boston_df6.values
```

In [150]:

```
# Seed value for random number generators to obtain reprodu
RANDOM_SEED = 1

# The model input data outside of the modeling method calls
names = ['Linear Regression', 'Random Forest Regressor', 'E

# Specify the set of regression models being evaluated (we
regressors = [LinearRegression(fit_intercept = True, normal
              RandomForestRegressor(n_estimators = 100, cri
              ExtraTreesRegressor(n_estimators = 100, crite
             ]
```

In [151]:

```
# Establish number of cross folds employed for cross-valida
N_FOLDS = 10

# Setup numpy array for storing results
cv_results = np.zeros((N_FOLDS, len(names)))

# Initiate splitting process
kf = KFold(n_splits = N_FOLDS, shuffle=False, random_state 

# Check the splitting process by looking at fold observatio
index_for_fold = 0  # Fold count initialized
for train_index, test_index in kf.split(model_data):
    print('\nFold index:', index_for_fold, '---------------

# The structure of modeling data for this study has the res
# so 1:model_data.shape[1] slices for explanatory variables
    X_train = model_data[train_index, 1:model_data.shape[1]
    X_test = model_data[test_index, 1:model_data.shape[1]]
    y_train = model_data[train_index, 0]
    y_test = model_data[test_index, 0]

    index_for_method = 0  # Method count initialized
    for name, reg_model in zip(names, regressors):
        reg_model.fit(X_train, y_train)  # Fit on the train

        # Evaluate on the test set for this fold
        y_test_predict = reg_model.predict(X_test)
        fold_method_result = sqrt(mean_squared_error(y_test
        cv_results[index_for_fold, index_for_method] = fold
        index_for_method += 1

    index_for_fold += 1

cv_results_df = pd.DataFrame(cv_results)
cv_results_df.columns = names

print('\n----------------------------------------------------
print('Average results from ', N_FOLDS, '-fold cross-valida
      'in standardized units (mean 0, standard deviation 1)
      '\nMethod                  Root mean-squared error', sep
print(cv_results_df.mean())
```

```
Fold index: 0 --------------------------------
----------------------------------------------
---------

Fold index: 1 --------------------------------
----------------------------------------------
---------

Fold index: 2 --------------------------------
----------------------------------------------
---------

Fold index: 3 --------------------------------
----------------------------------------------
---------

Fold index: 4 --------------------------------
----------------------------------------------
---------

Fold index: 5 --------------------------------
----------------------------------------------
---------

Fold index: 6 --------------------------------
----------------------------------------------
---------

Fold index: 7 --------------------------------
----------------------------------------------
---------

Fold index: 8 --------------------------------
----------------------------------------------
---------

Fold index: 9 --------------------------------
----------------------------------------------
---------

----------------------------------------------
-----------------------------------------
Average results from 10-fold cross-validation
in standardized units (mean 0, standard deviat
```

```
ion 1)

Method                    Root mean-squared error
Linear Regression               0.106519
Random Forest Regressor         0.089357
Extra Trees Regressor           0.085445
dtype: float64
```

In [152]:

```
cv_results_df.head(10)
```

Out[152]:

	Linear Regression	Random Forest Regressor	Extra Trees Regressor
0	0.075081	0.068434	0.070781
1	0.078732	0.050347	0.050986
2	0.066255	0.045301	0.041568
3	0.110983	0.103409	0.109652
4	0.106297	0.073054	0.066398
5	0.108108	0.098385	0.090506
6	0.069981	0.062374	0.063247
7	0.235195	0.206456	0.194840
8	0.109354	0.103207	0.092707
9	0.105199	0.082600	0.073768

In [153]:

```
param_grid = {
            "n_estimators"      : [100,125,150],
            "max_features"      : ["auto", "sqrt", "log2"],
            "min_samples_split" : [2,4,8],
            "bootstrap": [True, False],
            }
```

In [154]:

```
estimator = RandomForestRegressor()
```

In [155]:

```
grid = GridSearchCV(estimator, param_grid, cv=10)
```

In [156]:

```
grid.fit(X_train, y_train)
```

Out[156]:

```
GridSearchCV(cv=10, estimator=RandomForestRegr
essor(),
             param_grid={'bootstrap': [True, F
alse],
                         'max_features': ['aut
o', 'sqrt', 'log2'],
                         'min_samples_split':
[2, 4, 8],
                         'n_estimators': [100,
125, 150]})
```

In [157]:

```
X = X.drop(['crim', 'zn', 'age'], 1)
```

In [158]:

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, t
```

In [159]:

```
lrm = LinearRegression()

# Fit data on to the model
lrm.fit(X_train, y_train)

# Predict
y_predicted_lrm = lrm.predict(X_test)
```

In [160]:

```
print("Linear Regression R_squared = ",lrm.score(X,Y))
pred= lrm.predict(X_test)
rmse = sqrt(mean_squared_error(pred, y_test))
print('Linear Regression RMSE = ', rmse)
```

```
Linear Regression R_squared =  0.7788884828495
811
Linear Regression RMSE =  4.8832481772422955
```

In [161]:

```
lrm = LinearRegression()

# Fit data on to the model
lrm.fit(X_test, y_test)

print("Linear Regression R_squared = ",lrm.score(X,Y))
pred= lrm.predict(X_test)
rmse = sqrt(mean_squared_error(pred, y_test))
print('Linear Regression RMSE = ', rmse)
```

```
Linear Regression R_squared =  0.7575187568027
08
Linear Regression RMSE =  4.5063094117325075
```

In [203]:

```
Randreg = RandomForestRegressor(oob_score = True)

# Fit data on to the model
Randreg.fit(X_train, y_train)
print(Randreg.oob_score_)
# Predict
y_predicted_Randreg = Randreg.predict(X_test)
```

```
0.8511319157105197
```

In [163]:

```
print("Random Forest Regressor R_squared = ",Randreg.score(
pred= Randreg.predict(X)
rmse = sqrt(mean_squared_error(pred, Y))
print('Random Forest Regressor RMSE = ', rmse)

print(Randreg.decision_path(X))
```

```
Random Forest Regressor R_squared =  0.9522091
7119023
Random Forest Regressor RMSE =  2.005341396520
999
(<506x49104 sparse matrix of type '<class 'num
py.int64'>'
        with 566466 stored elements in Compres
sed Sparse Row format>, array([    0,   473,
984,  1481,  1994,  2499,  3006,  3507,  4004,
        4481,  5008,  5479,  5942,  6423,  692
4,  7427,  7916,  8405,
        8910,  9419,  9882, 10371, 10862, 1134
3, 11844, 12353, 12828,
       13329, 13808, 14275, 14782, 15283, 1578
2, 16257, 16732, 17223,
       17716, 18207, 18700, 19203, 19688, 2017
5, 20668, 21161, 21654,
       22123, 22608, 23091, 23582, 24077, 2455
6, 25057, 25536, 26039,
       26522, 27007, 27508, 27981, 28470, 2897
5, 29478, 29965, 30454,
       30939, 31424, 31937, 32404, 32901, 3339
0, 33877, 34358, 34841,
       35336, 35825, 36292, 36785, 37272, 3775
9, 38264, 38737, 39244,
       39753, 40230, 40723, 41220, 41721, 4219
8, 42693, 43200, 43691,
       44164, 44651, 45140, 45639, 46144, 4664
3, 47140, 47621, 48104,
       48601, 49104]))
```

In [204]:

```
Randreg = RandomForestRegressor(oob_score=True)

# Fit data on to the model
Randreg.fit(X_test, y_test)
print(Randreg.oob_score_)
print("Random Forest Regressor R_squared = ",Randreg.score(
pred= Randreg.predict(X)
rmse = sqrt(mean_squared_error(pred, Y))
print('Random Forest Regressor RMSE = ', rmse)

print(Randreg.decision_path(X))
```

```
0.7690693844384108
Random Forest Regressor R_squared =  0.8657236
418646057
Random Forest Regressor RMSE =  3.361366092220
37
(<506x12416 sparse matrix of type '<class 'num
py.int64'>'
        with 417861 stored elements in Compres
sed Sparse Row format>, array([    0,   125,
244,   357,   470,   585,   708,   839,   970,
        1087,  1210,  1333,  1456,  1579,  170
6,  1825,  1962,  2097,
        2222,  2359,  2492,  2617,  2746,  286
9,  2992,  3111,  3242,
        3369,  3490,  3611,  3744,  3861,  398
8,  4107,  4220,  4341,
        4470,  4595,  4716,  4835,  4962,  508
9,  5214,  5353,  5476,
        5605,  5728,  5859,  5978,  6097,  622
2,  6351,  6472,  6585,
        6714,  6823,  6954,  7079,  7208,  733
1,  7450,  7579,  7692,
        7811,  7946,  8077,  8200,  8319,  844
8,  8569,  8684,  8811,
        8940,  9073,  9190,  9311,  9418,  955
3,  9676,  9805,  9936,
       10063, 10194, 10315, 10444, 10557, 1068
6, 10817, 10938, 11059,
       11190, 11307, 11428, 11555, 11678, 1179
```

```
9, 11928, 12049, 12168,
       12301, 12416]))
```

In [165]:

```
ETreg = ExtraTreesRegressor()

# Fit data on to the model
ETreg.fit(X_train, y_train)

# Predict
y_predicted_ETreg = ETreg.predict(X_test)
```

In [166]:

```
print("Extra-Trees Regressor R_squared = ",ETreg.score(X,Y))
pred= ETreg.predict(X)
rmse = sqrt(mean_squared_error(pred, Y))
print('Extra-Trees Regressor RMSE = ', rmse)

ETreg.feature_importances_
```

```
Extra-Trees Regressor R_squared =  0.971433330
6997563
Extra-Trees Regressor RMSE =  1.55040743210731
42
```

Out[166]:

```
array([0.05020814, 0.00768331, 0.04984039, 0.3
0247079, 0.05747416,
       0.01533284, 0.03958037, 0.05673681, 0.4
2067319])
```

In [167]:

```
ETreg = ExtraTreesRegressor()

# Fit data on to the model
ETreg.fit(X_test, y_test)

print("Extra-Trees Regressor R_squared = ",ETreg.score(X,Y))
pred= ETreg.predict(X)
rmse = sqrt(mean_squared_error(pred, Y))
print('Extra-Trees Regressor RMSE = ', rmse)

ETreg.feature_importances_
```

```
Extra-Trees Regressor R_squared =  0.884679829
74532
Extra-Trees Regressor RMSE =  3.11507620498212
57
```

Out[167]:

```
array([0.03512377, 0.03896161, 0.03888613, 0.3
9904249, 0.04258862,
       0.02458222, 0.0444144 , 0.05965775, 0.3
1674301])
```

In [168]:

```
X = X.drop(['chas', 'rad', 'ptratio'], 1)
```

In [169]:

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, t
```

In [170]:

```
lrm = LinearRegression()

# Fit data on to the model
lrm.fit(X_train, y_train)

# Predict
y_predicted_lrm = lrm.predict(X_test)
```

In [171]:

```
print("Linear Regression R_squared = ",lrm.score(X,Y))
pred= lrm.predict(X_test)
rmse = sqrt(mean_squared_error(pred, y_test))
print('Linear Regression RMSE = ', rmse)
```

```
Linear Regression R_squared =  0.7538953990267
538
Linear Regression RMSE =  5.370813842614971
```

In [206]:

```
Randreg = RandomForestRegressor(oob_score = True)

# Fit data on to the model
Randreg.fit(X_train, y_train)
print(Randreg.oob_score_)
# Predict
y_predicted_Randreg = Randreg.predict(X_test)
```

```
0.8460076065336399
```

In [173]:

```
print("Random Forest Regressor R_squared = ",Randreg.score(
pred= Randreg.predict(X)
rmse = sqrt(mean_squared_error(pred, Y))
print('Random Forest Regressor RMSE = ', rmse)

print(Randreg.decision_path(X))
```

```
Random Forest Regressor R_squared =  0.9460225
309217469
Random Forest Regressor RMSE =  2.131190622716
7325
(<506x49198 sparse matrix of type '<class 'num
py.int64'>'
        with 563039 stored elements in Compres
sed Sparse Row format>, array([    0,    495,
968,  1445,  1928,  2441,  2942,  3439,  3952,
        4423,  4904,  5411,  5914,  6411,  691
8,  7411,  7902,  8367,
        8860,  9365,  9854, 10341, 10850, 1134
5, 11846, 12343, 12830,
       13311, 13796, 14285, 14794, 15285, 1577
0, 16261, 16782, 17299,
       17780, 18251, 18730, 19221, 19698, 2017
9, 20678, 21157, 21662,
       22141, 22624, 23123, 23608, 24121, 2462
4, 25121, 25618, 26105,
       26600, 27093, 27572, 28073, 28574, 2908
9, 29576, 30073, 30552,
       31023, 31510, 32003, 32516, 33011, 3350
0, 33971, 34482, 34949,
       35436, 35951, 36428, 36915, 37408, 3791
5, 38402, 38875, 39368,
       39871, 40340, 40823, 41310, 41803, 4229
6, 42785, 43272, 43759,
       44258, 44747, 45258, 45761, 46260, 4674
3, 47220, 47721, 48218,
       48725, 49198]))
```

In [174]:

```
ETreg = ExtraTreesRegressor()

# Fit data on to the model
ETreg.fit(X_train, y_train)

# Predict
y_predicted_ETreg = ETreg.predict(X_test)
```

In [175]:

```
print("Extra-Trees Regressor R_squared = ",ETreg.score(X,Y))
pred= ETreg.predict(X)
rmse = sqrt(mean_squared_error(pred, Y))
print('Extra-Trees Regressor RMSE = ', rmse)

ETreg.feature_importances_
```

```
Extra-Trees Regressor R_squared =  0.972437203
6218448
Extra-Trees Regressor RMSE =  1.52292205544358
74
```

Out[175]:

```
array([0.07066006, 0.05756401, 0.28634819, 0.0
6159156, 0.06808642,
       0.45574976])
```

In [176]:

```
lrm = LinearRegression()

# Fit data on to the model
lrm.fit(X_test, y_test)
```

Out[176]:

```
LinearRegression()
```

In [177]:

```
print("Linear Regression R_squared = ",lrm.score(X,Y))
pred= lrm.predict(X)
rmse = sqrt(mean_squared_error(pred, Y))
print('Linear Regression RMSE = ', rmse)
```

```
Linear Regression R_squared =  0.7330092454955
464
Linear Regression RMSE =  4.739845103037422
```

In [207]:

```
Randreg = RandomForestRegressor(oob_score = True)

# Fit data on to the model
Randreg.fit(X_test, y_test)
print(Randreg.oob_score_)
```

```
0.7786598216920804
```

In [179]:

```
print("Random Forest Regressor R_squared = ",Randreg.score(
pred= Randreg.predict(X)
rmse = sqrt(mean_squared_error(pred, Y))
print('Random Forest Regressor RMSE = ', rmse)

print(Randreg.decision_path(X))
```

```
Random Forest Regressor R_squared =  0.8576266
582003425
Random Forest Regressor RMSE =  3.461229347621
728
(<506x12280 sparse matrix of type '<class 'num
py.int64'>'
        with 421832 stored elements in Compres
sed Sparse Row format>, array([    0,   125,
250,   365,   486,   607,   732,   863,   990,
        1127,  1252,  1377,  1510,  1631,  174
4,  1859,  1984,  2103,
        2220,  2333,  2448,  2573,  2704,  283
5,  2952,  3065,  3192,
        3321,  3464,  3587,  3714,  3843,  396
2,  4089,  4206,  4337,
        4466,  4591,  4714,  4849,  4972,  510
1,  5228,  5345,  5470,
        5593,  5708,  5839,  5968,  6083,  620
2,  6319,  6440,  6555,
        6670,  6797,  6914,  7029,  7154,  727
5,  7382,  7503,  7626,
        7739,  7870,  7999,  8124,  8227,  834
4,  8467,  8590,  8721,
        8848,  8975,  9100,  9227,  9348,  947
1,  9604,  9721,  9850,
        9967, 10094, 10215, 10344, 10481, 1059
8, 10715, 10834, 10959,
       11078, 11201, 11324, 11449, 11576, 1170
3, 11824, 11947, 12060,
       12175, 12280]))
```

In [180]:

```
ETreg = ExtraTreesRegressor()

# Fit data on to the model
ETreg.fit(X_test, y_test)
```

Out[180]:

```
ExtraTreesRegressor()
```

In [181]:

```
print("Extra-Trees Regressor R_squared = ",ETreg.score(X,Y))
pred= ETreg.predict(X)
rmse = sqrt(mean_squared_error(pred, Y))
print('Extra-Trees Regressor RMSE = ', rmse)

ETreg.feature_importances_
```

```
Extra-Trees Regressor R_squared =  0.878160074
5576015
Extra-Trees Regressor RMSE =  3.20192275269551
7
```

Out[181]:

```
array([0.04743791, 0.05638703, 0.39701832, 0.0
5766252, 0.05678102,
       0.38471321])
```