BOSTON HOUSING ANALYSIS

OCTOBER 30, 2021

KO TSZ NGA (VALERIE)
valerie.ktn@gmail.com

Introduction

The aim of this analysis is to compare different models such as **random forest, boosting** and **other baseline methods' performance** in predicting 'medv'.

Background of the dataset

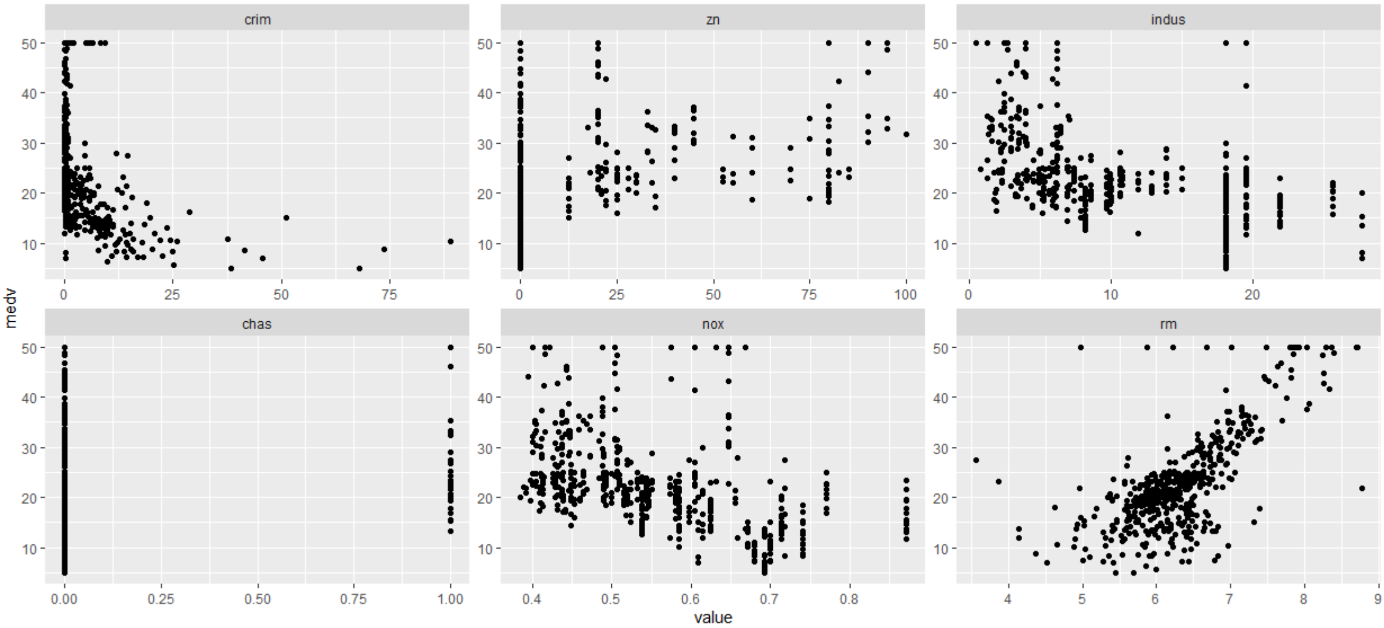
The '**Boston**' dataset from the library MASS will be used in the analysis. It consists of **506 rows** of census data of Boston in 1970. **14 variables** are present, with '**medv**' being the dependent variable. The explanation of the variables is described as follows:

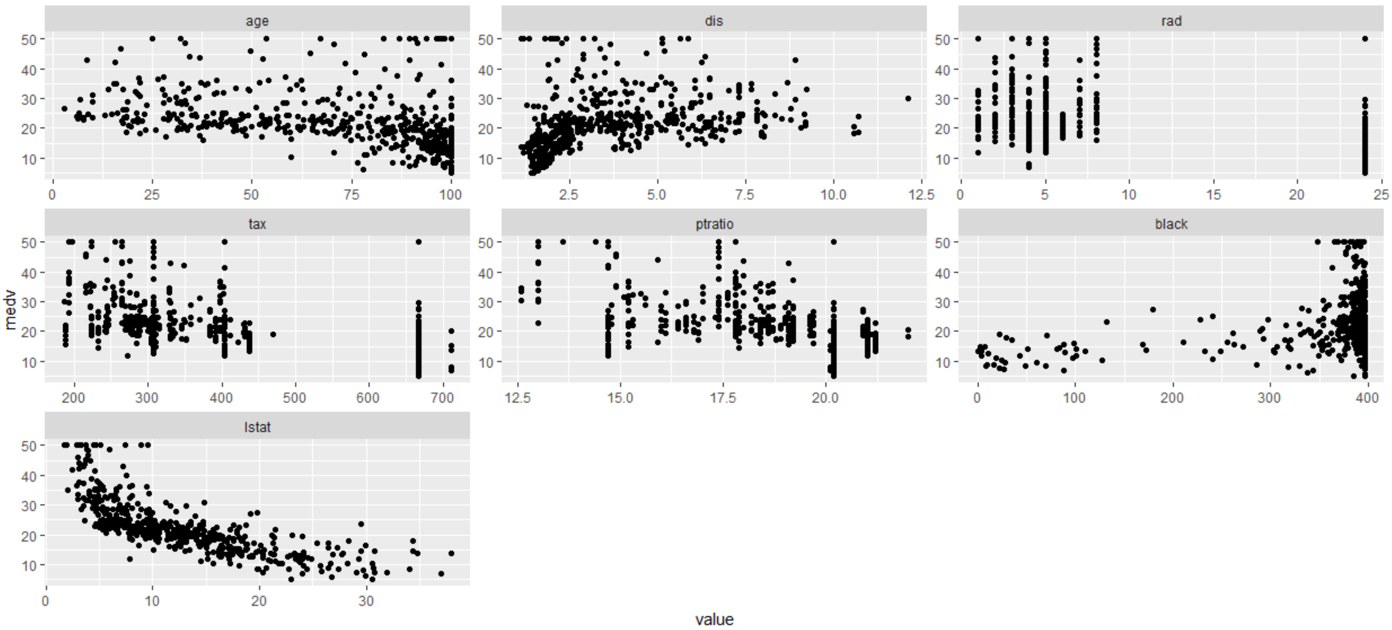
crim	per capita crime rate by town
zn	proportion of residential land zoned for lots over 25,000 sq.ft
indus	proportion of non-retail business acres per town
chas	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
nox	nitric oxides concentration (parts per 10 million)
rm	average number 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 USD 10,000
ptratio	pupil-teacher ratio by town
b	$1000(B - 0.63)^2$ where B is the proportion of blacks by town
lstat	percentage of lower status of the population
medv	median value of owner-occupied homes in USD 1000's

Exploratory Data Analysis

(i) Scatter plot of raw data

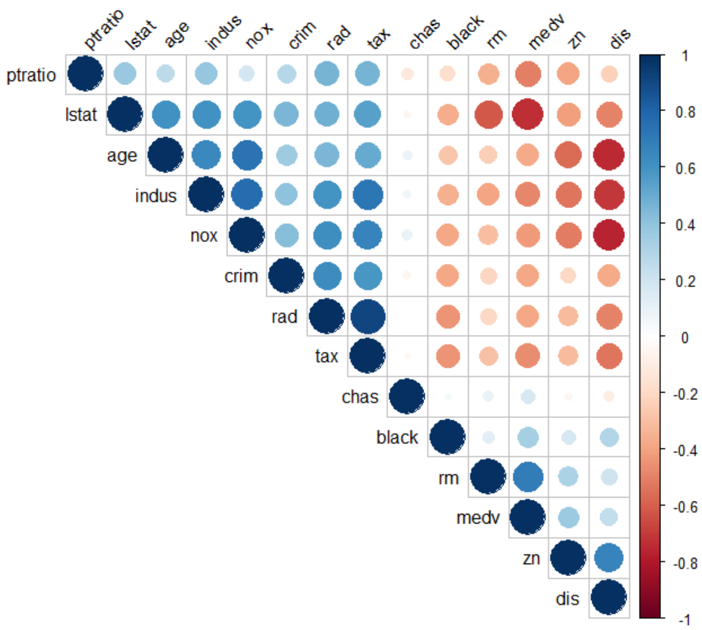
The relationship between each **predicting variables and the response ‘medv’** is displayed below:





Obvious trends can be observed between (1)‘medv’ and ‘lstat’ and (2) ‘medv’ and ‘rm’. For (1), it is shown that ‘medv’ decreases with increasing ‘lstat’, which makes sense because the value of the home decreases when percentage of lower status of the population (ie. ‘lstat’) increases. For (2), ‘medv’ increases with increasing ‘rm’, that implied the average number of rooms per dwelling (ie. ‘rm’) increases with the home value. Other than these two, the other continuous variables don’t show notable relationship with the response, further analysis will be carried out to verify on that.

(ii) Correlation analysis



There is a **positive correlation between 'lstat' and 'medv'** and a **negative correlation between 'rm' and 'medv'** which agrees with what we observe in (i). Another thing to note is that there is **a significant correlation between 'ptratio' and 'medv' too**. The above graph raises a concern for **multilinearity**. For instance, 'dis' is strongly positively related to 'age', 'indus' and 'nox', that would suggest us to pay extra attention in variable selection.

Method

The 'Boston' dataset is split into training and testing dataset. **80%** of the data is randomly selected to be the **training** set and the remaining **20%** data will be the **testing** set.

(1) Random forest, (2) boosting and (3) baseline methods will be performed.

Results

(1) Random forest

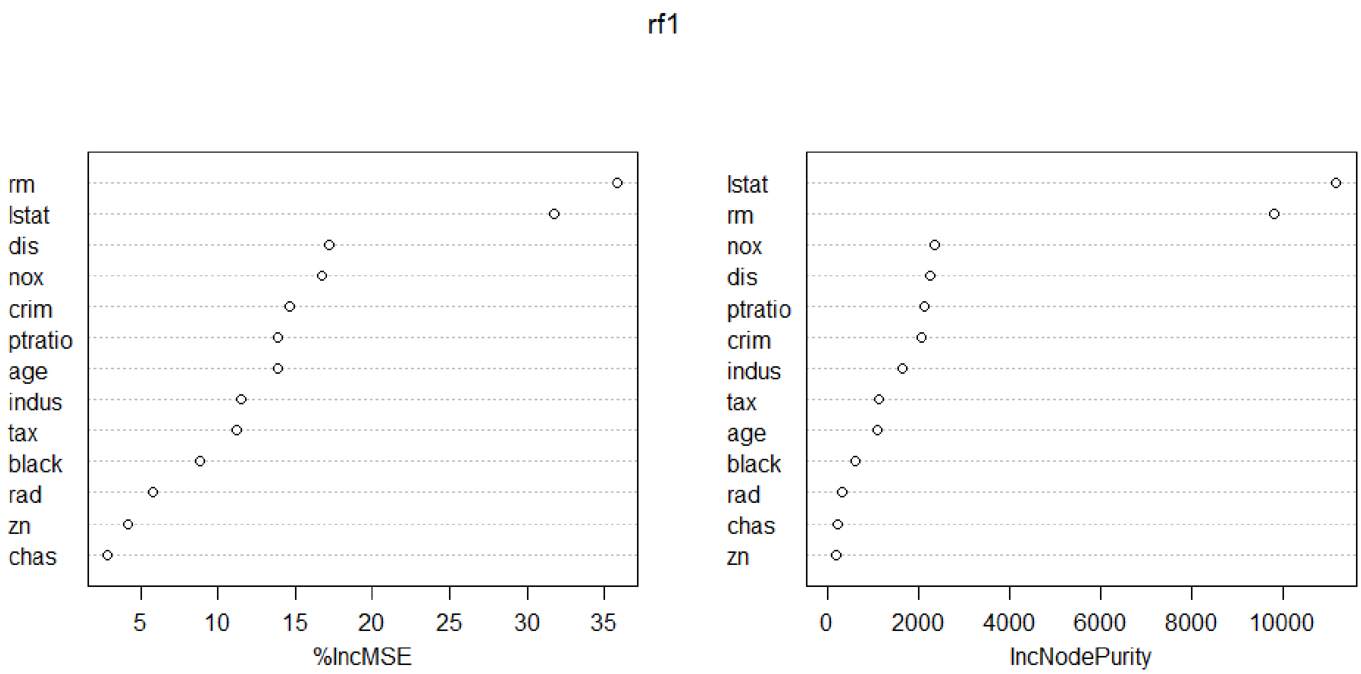
(1.1) rf1

First, I created a random forest model with the **default parameters** and that gives me the below result.

```
Call:
 randomForest(formula = medv ~ ., data = btrain, importance = TRUE)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 4

          Mean of squared residuals: 11.21297
                    % Var explained: 87.25
```

There are **4 variables** in each split and the **mean squared residuals is 11.21297**. **87.25% variance** is explained.

Variable importance is displayed as:



From the left graph is about **%IncMSE** , which is the % that prediction accuracy (measured by MSE) decreases when predicting on Out-Of-Bag samples with the given variable removed. As we can see **'rm' and 'lstat' are the 2 variables that correspond to the greatest drop, they are the important variable in this case.**

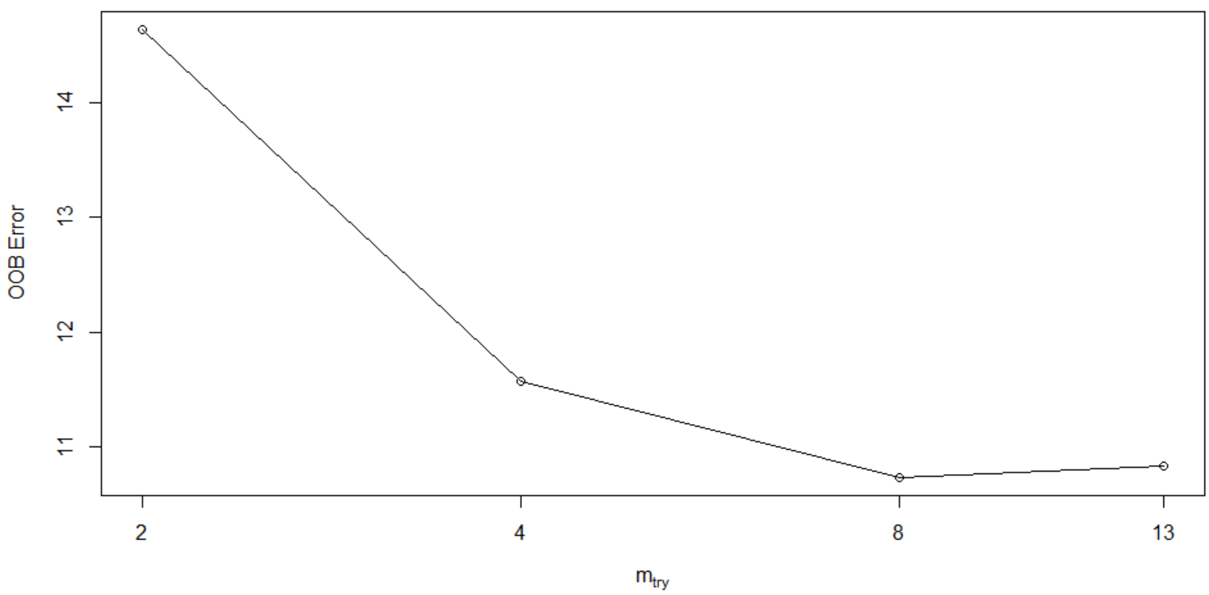
The right graph shows **IncNodePurity**, which is a measure of how much node purity decreases in splits on that variable, averaged over all grown trees. Again, in this case, a higher number indicates a variable is more important. Hence, again, **‘rm’ and ‘lstat’ are the 2 most important variables in this case.**

I then fit the model with testing set and that gives a **MSE value of 6.812437**.

(1.2) Tuning parameter

To inspect how the performance of the model change with tuning parameter, I rebuild the random forest again with different mtry.

I use the tuneRF function to find which mtry should be used. Below is the result.



As shown, **mtry=8** gives the lowest OOB error. Hence, I will try to use that to train another mode — rf2.

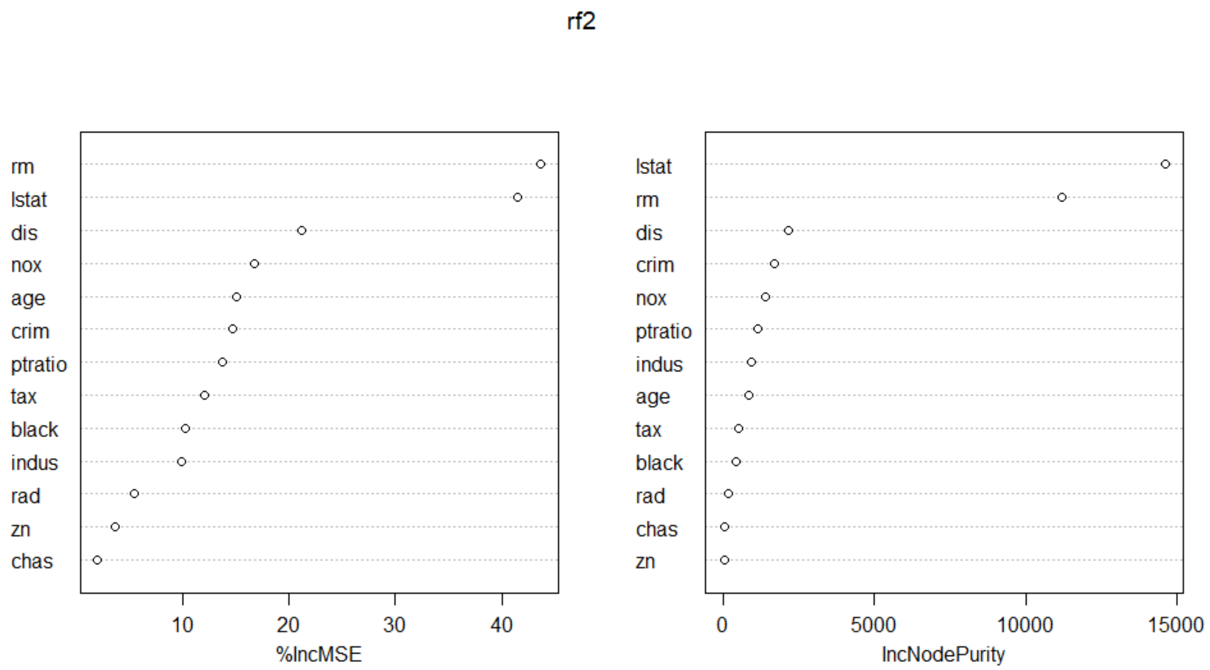
(1.3) rf2

rf2 is built with all the parameter unchanged except mtry = 8 instead the default.

```
Call:
 randomForest(formula = medv ~ ., data = btrain, mtry = 8, importance = TRUE)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 8

          Mean of squared residuals: 10.43316
                    % Var explained: 88.14
```

The **mean squared residuals is 10.43316**, which is lower than the random forest we did before. **88.14% variance** is explained, higher than the value in rf1.

Variable importance is displayed as:

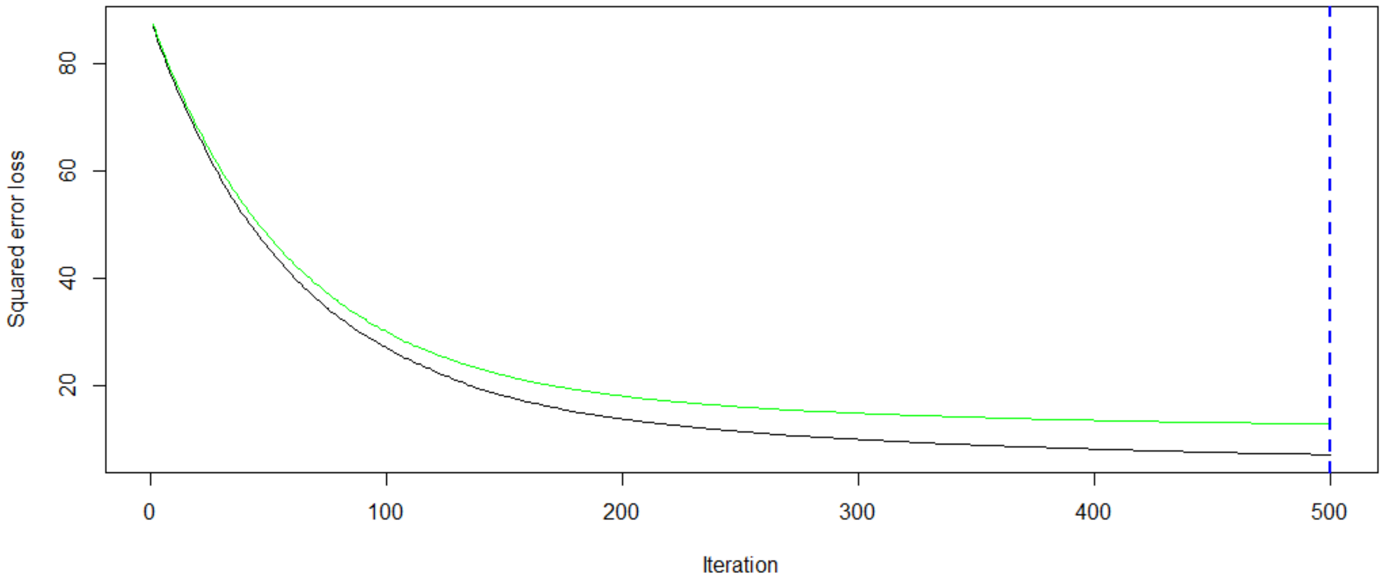


The top 2 most important parameters reflected are the same as rf1 model. In this case, the **testing error (MSE) is 6.767315** which is lower than rf1.

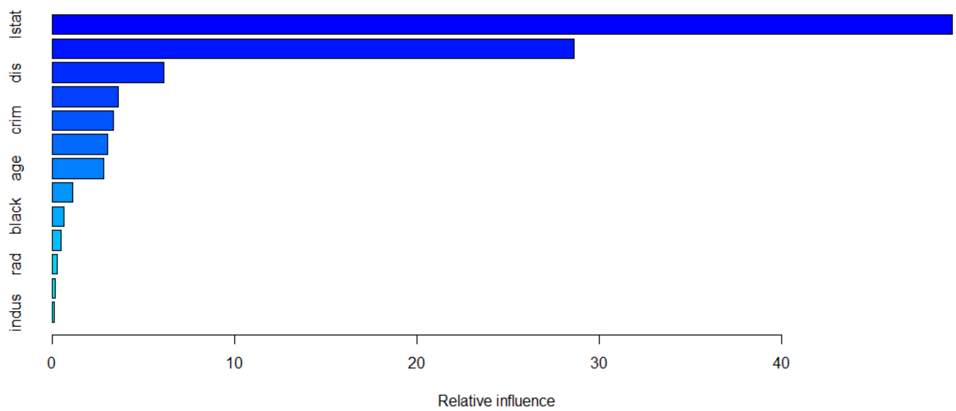
(2) Boosting

(2.1) gbm.bos1

To allow a fair comparison with the rf1 model, the parameters are set to be the same. **n.trees is set to be 500, interaction depth is set to be 4, shrinkage is 0.01 with cv.folds =10.**



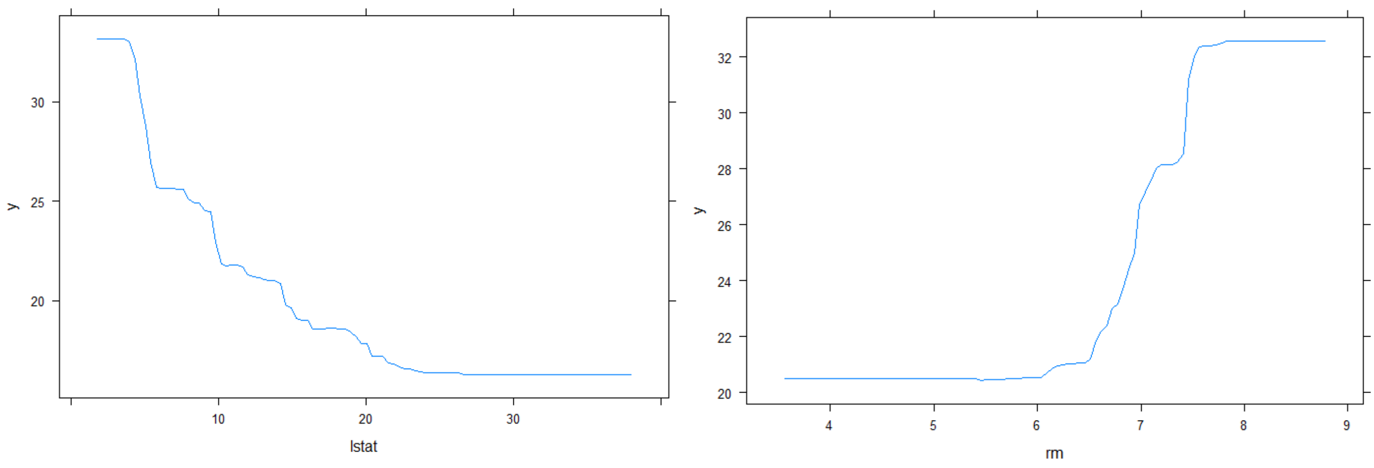
The estimated optimal number for iterations by cross-validation is found to be 500. The black and green lines are the deviance of training and testing dataset respectively.



	var	rel.inf
lstat	lstat	49.3896468
rm	rm	28.6448832
dis	dis	6.1271416
nox	nox	3.6268330
crim	crim	3.3962836
ptratio	ptratio	3.0554201
age	age	2.8459008
tax	tax	1.1280732
black	black	0.6545913
chas	chas	0.5269385
rad	rad	0.3091180
zn	zn	0.1816595
indus	indus	0.1135105

‘lstat’ is found to be the most important, ‘rm’ follows.

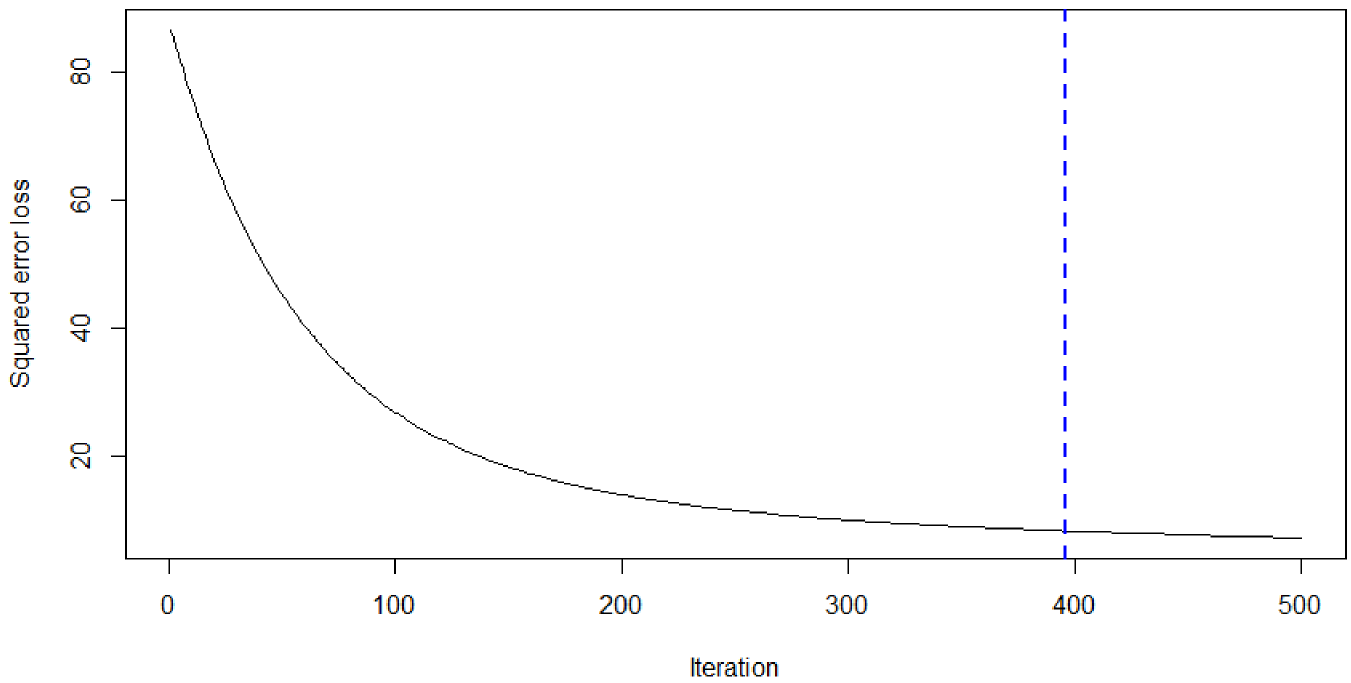
The performance of 'lstat' and 'rm' is displayed as:



That again agrees on the corelation we found in exploratory data analysis.

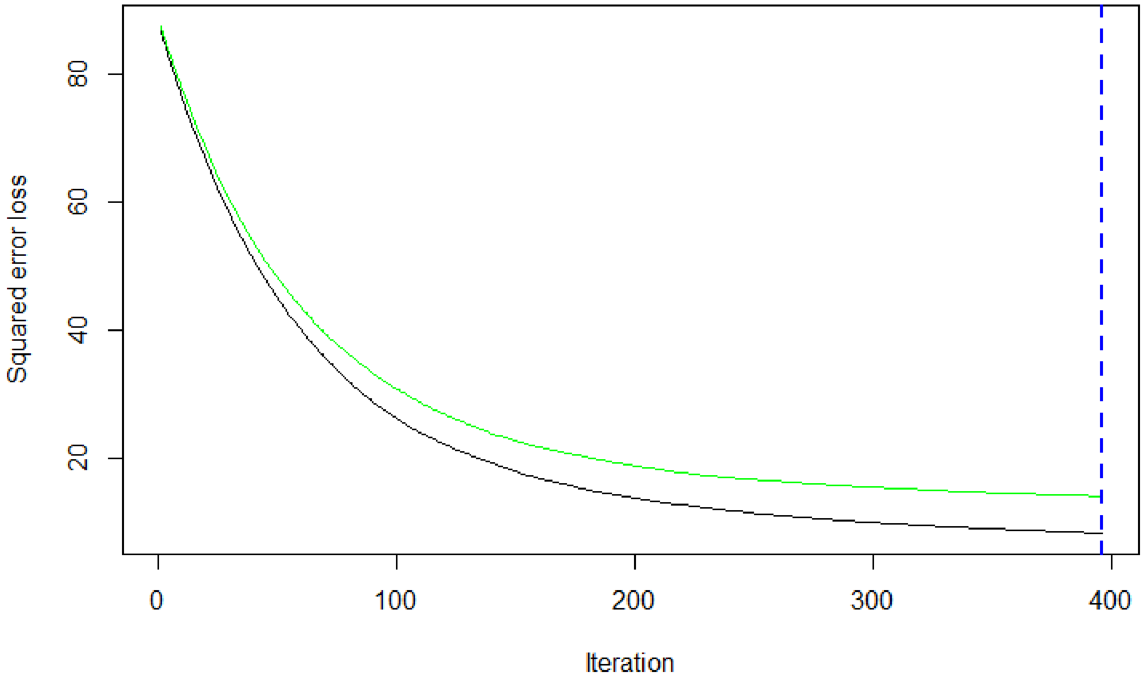
The **training error** of the model is calculated to be **7.219746** and **testing error is 8.692032**.

(2.2) Tuning parameters



The estimated optimal number for iterations by OOB is found to be **396**. Therefore, another boost model with ntree = 396 will be used.

(2.3) gbm.bos2



The estimated optimal number for iterations by cross-validation is found to be 396.

Noted that it gives the same result in terms of important variables, so I skipped the output of that.

The **training error** is calculated to be **8.379418** and the **testing error is 9.130149**.

(3) Baseline methods

(3.1). Linear regression with stepwise variable selection using AIC

Below is the final output.

```
Call:
lm(formula = medv ~ crim + zn + chas + nox + rm + dis + rad + 
    tax + ptratio + black + lstat, data = btrain)

Coefficients:
(Intercept)         crim           zn         chas          nox  
  37.852432    -0.118176     0.032773     3.174489   -17.298344  
         rm          dis          rad          tax      ptratio  
   3.715369    -1.388145     0.306700    -0.011164    -0.999520  
      black        lstat  
   0.008554    -0.542784  
```

The **number of predicting parameters** recommended is reduced to **11**.

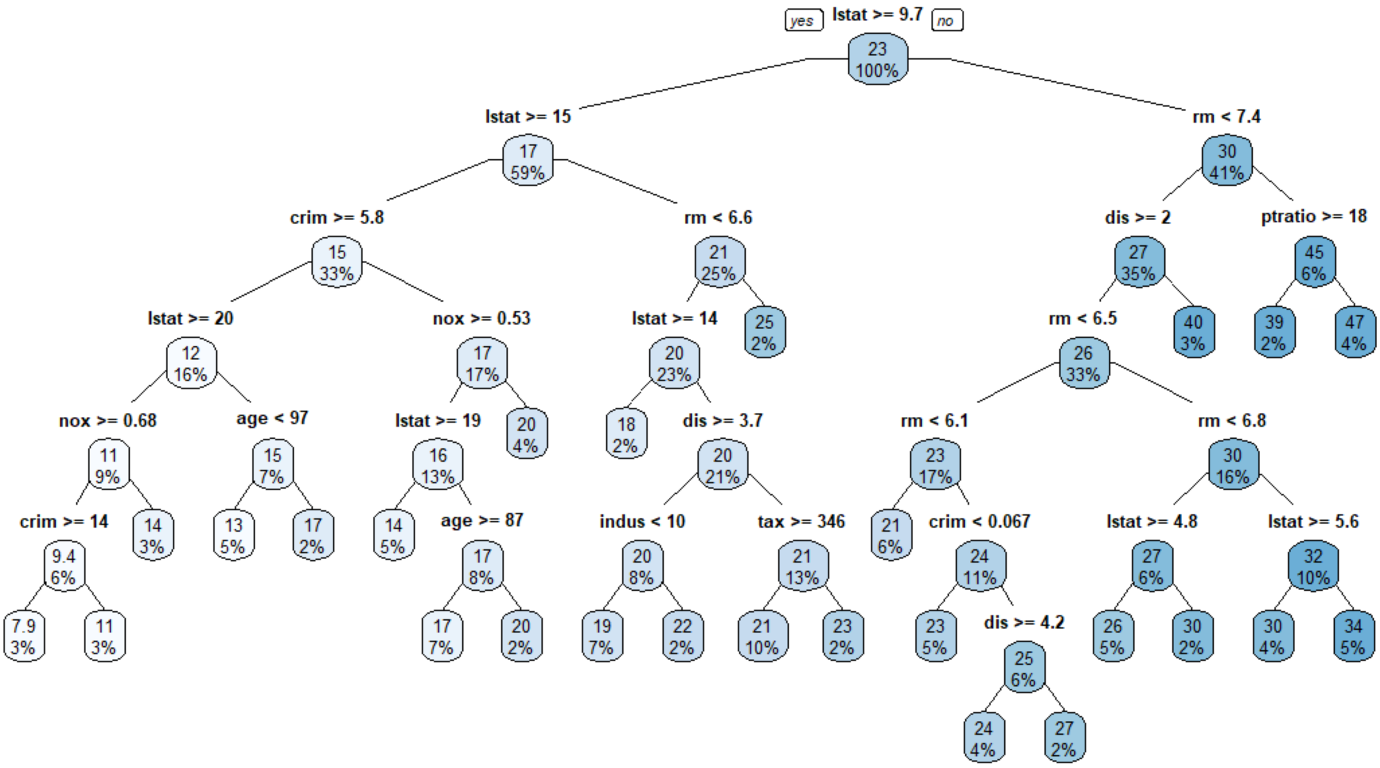
Linear regression model is built with these 11 parameters and that gives the below result.

```
Call:
lm(formula = medv ~ crim + zn + chas + nox + rm + dis + rad + 
    tax + ptratio + black + lstat, data = btrain)

Residuals:
     Min       1Q   Median       3Q      Max 
-16.1701  -2.8473  -0.6796   1.8125  25.8477 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)  37.852432   5.855411   6.465 3.02e-10 ***
crim         -0.118176   0.038602  -3.061 0.002354 ** 
zn            0.032773   0.016252   2.017 0.044427 *  
chas          3.174489   0.942623   3.368 0.000833 ***
nox         -17.298344   4.086575  -4.233 2.87e-05 ***
rm            3.715369   0.471892   7.873 3.40e-14 ***
dis          -1.388145   0.218204  -6.362 5.56e-10 ***
rad           0.306700   0.072360   4.239 2.81e-05 ***
tax          -0.011164   0.003861  -2.892 0.004045 ** 
ptratio      -0.999520   0.152246  -6.565 1.65e-10 ***
black         0.008554   0.003231   2.648 0.008430 ** 
lstat        -0.542784   0.055135  -9.845  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.962 on 393 degrees of freedom
Multiple R-squared:  0.7283,	Adjusted R-squared:  0.7207 
F-statistic: 95.79 on 11 and 393 DF,  p-value: < 2.2e-16
```

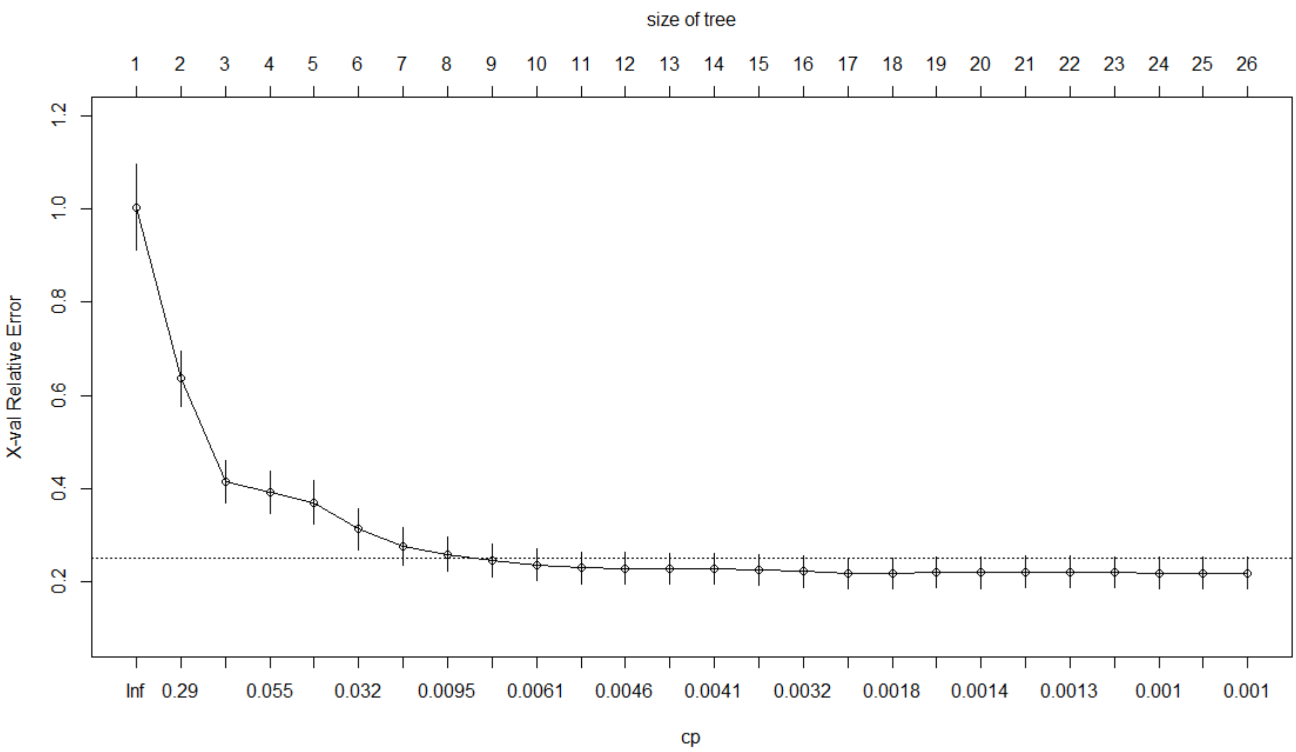
The **testing error** is calculated to be **14.33994.**

(3.2) Regression tree

I arbitrarily set **cp = 0.001** in the regression tree model and that gives us the below result

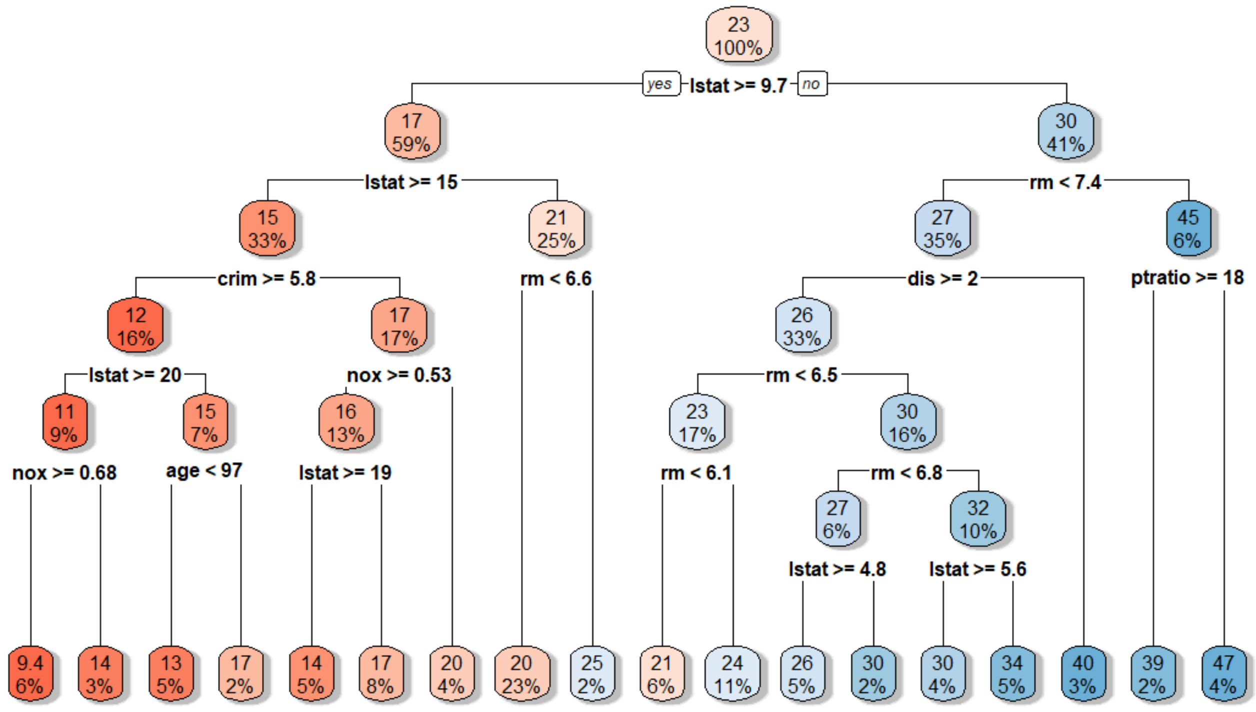


The optimal tree size is found with the below graph.



The **optimal tree size** should be **18** with **cp= 0.001592299**.

The tree is pruned according to the newly found cp.



Its **testing error** is found to be **12.39004.**

Findings

The testing error of all the models performed is described as:

	Random Forest		Boosting		Baseline method	
Models	rf1	rf2	gbm.bos1	gbm.bos2	Linear regression	Regression tree
Testing error (MSE)	6.812437	6.767315	8.692032	9.130149	14.33994	12.39004

To conclude, **the random forest and boosting models perform better than baseline methods.** The performance of **random forest after turning improved** but not the case for boosting. In boosting, the number of trees is reduced from 500 to 396 after tuning. That makes sense because **boosting error should drop down as the number of trees increases**, which is evidence showing that **boosting is reluctant to overfit**. Overall, **random forest with tuned parameters performs the best.**