**Exploring IRIS Data set with EDA**

**DATA MINING**

CAse: boston housing data

**Last Name: Das**

**First Name: Deepa**

# Boston Housing:

## Goal and Background:

Boston Housing dataset is a dataset that contains the details of houses in Boston given its different features. The median price value of the houses in Boston are featured by various factors that may or may not affect the housing price.

The main aim of the report is to build different tree models which can estimate the housing prices given other parameters. The performance of the model is a key factor in finding the best model that can accurately predict the prices. To find the best fitting model, in sample and out of sample performance of the model were compared using MSE and MSPE.

In this report, the primary focus is to compare traditional Linear Regression Model with the advanced Regression models like Regression Tree, Bagging, Random Forest and Boosting.

## Approach:

The entire dataset was split to training and testing data; (75%-25%). The tree models were trained on the training sample and the model performance was evaluated using the testing data. The performance parameter used was to find MSPE (Mean Square Prediction Error)

## Major Findings:

The major findings of comparison of different tree models can be summarized in the below table:

|  |  |  |
| --- | --- | --- |
| **Column1** | **MSE** | **MSPE** |
| Linear Model | 19.5441 | 29.6643 |
| Regression Tree | 13.9776 | 25.9169 |
| RegressionTree(Pruned) | 12.1627 | 23.7464 |
| Bagging | 10.8619 | 20.768 |
| Random Forest | 2.05192 | 12.86456 |
| Boosting | 0.01595 | 11.0053 |

Table1: Model comparison using MSE and MSPE

In the above table we can see that, the prediction accuracy of the model increases with advanced tree models like Bagging, Random forest and Boosting as compared to the traditional linear regression model and regression tree. Regression trees are easy to interpret but their prediction accuracy is not really good compared to the traditional models but by aggregating many decision trees, using methods like bagging, random forest, boosting, the predictive performance of trees can be substantially improved.

## Detailed Study of each model:

### Linear Regression Model:

First of all, a traditional linear regression model was fit using 75% training data and predictions were made using remaining 25% test data.

|  |  |  |
| --- | --- | --- |
| **Column1** | **In-Sample** | **Out-of-Sample** |
| MSE | 19.54406 | 29.66433 |

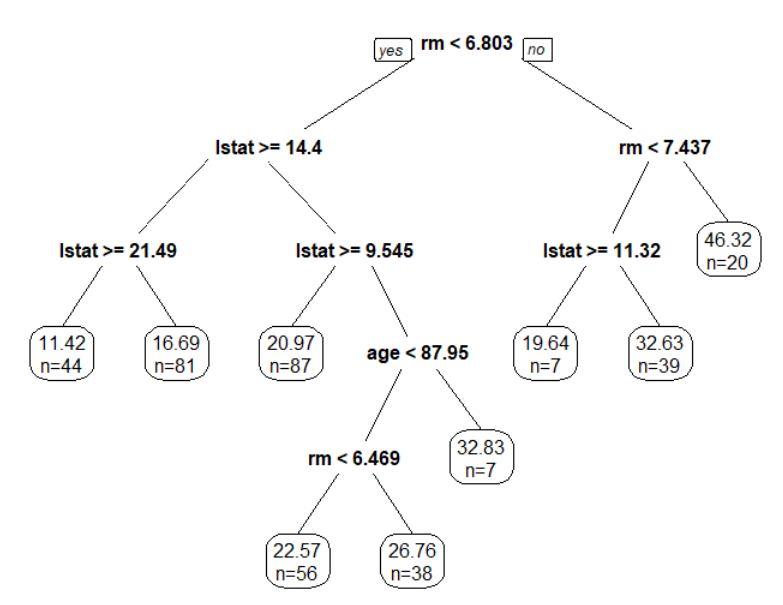
Table2: Prediction error for linear model

10-fold cross validation model was fitted and the out-of-sample MSE was found out to be 23.14775

### Regression Tree:

A regression tree was fitted to the Boston Housing Data which is available in the MASS package which has 14 features and 506 observations. The data is split into 75-25.

First, a regression tree model is fitted with training data only and the tree is plotted.

  
 Figure1: Regression tree model

Observations:

* The tree grown to full depth has 9 leaves and only 3 variables are used to construct the tree.
* The lstat variable measures the percentage of individuals with lower socioeconomic status. The tree shows that higher values of lstat correspond to lower house values.
* The tree predicts a median house price of $46, 320 for larger homes (rm >= 7.437)

Since the tree has been grown fully, it may be having high variance and low bias leading to potential overfitting of the data. Here, the goal is to reduce out of sample prediction error.

|  |  |  |
| --- | --- | --- |
| **Column1** | **In-Sample** | **Out-of-Sample** |
| MSE | 13.97758 | 25.9169 |

Table3: Prediction error regression tree

The test set MSE associated with regression tree is 25.91. The square root of the MSE is therefore around 5.09, indicating that this model leads to test predictions that are within around $5090 of the true median home value.

Now, we determine the complexity parameter to see whether pruning the tree will improve performance.

First with cp = 0.001, we obtain the large tree as below:

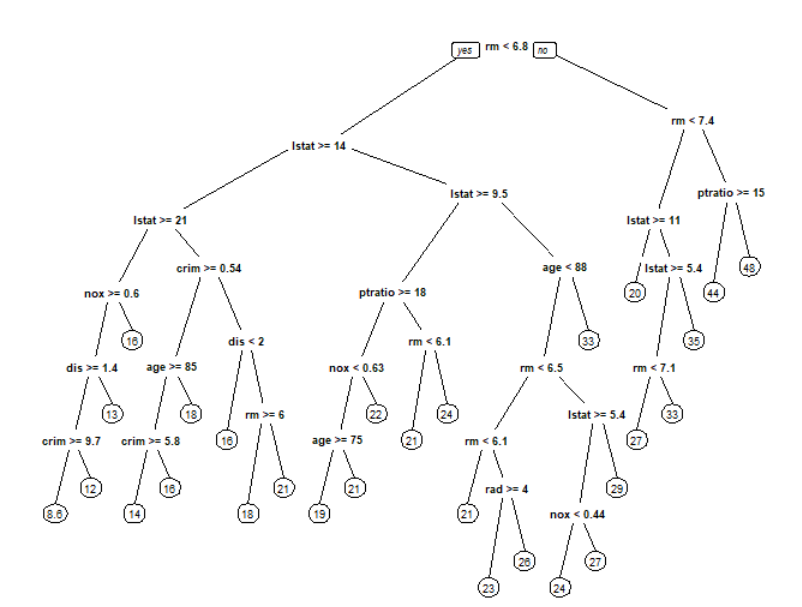


Figure2: Regression tree large size

The plotcp() function gives the relationship between 10-fold cross-validation error in the training set and size of tree.

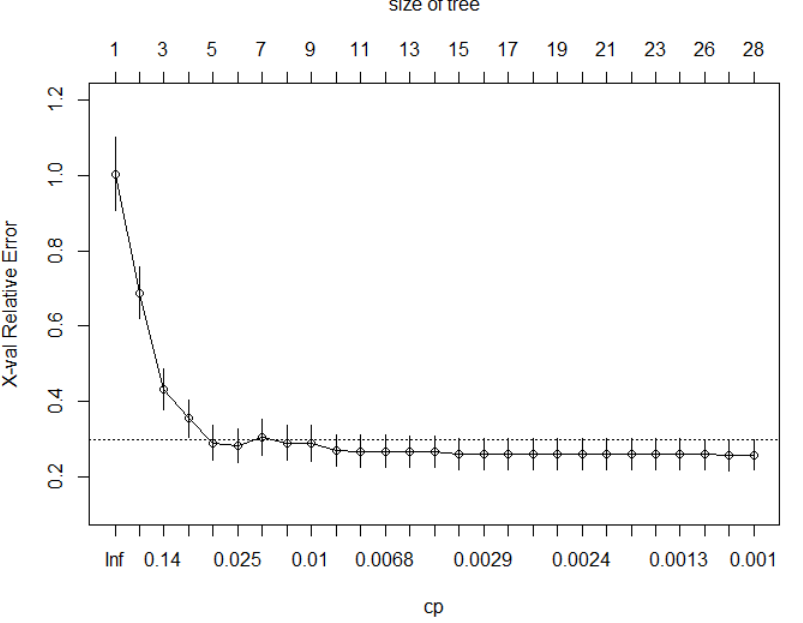


Figure3: cp plot to find size of tree with minimum cv error

In the above plot, as the complexity of the tree increases the x-val i.e cross validation error decreases but after a point doesn’t decrease much. This means that the cv error will not necessarily decrease when we increase the complexity of the model. So, we need to choose an optimal value of cp which is usually the leftmost value for which the mean lies below the horizontal line.

We can obtaing error value vs size of tree using printcp() which shows that after 10 splits the cv error doesn’t decrease much ie cp = 0.007

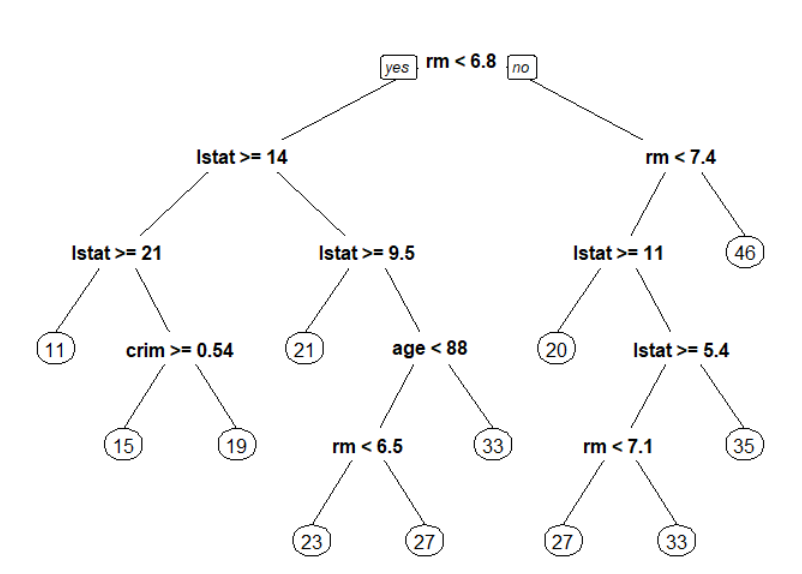
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | CP | nsplit | Relerror | xerror | xstd |
| 1 | 0.436704 | 0 | 1 | 1.00292 | 0.097242 |
| 2 | 0.166645 | 1 | 0.5633 | 0.68791 | 0.068337 |
| 3 | 0.112211 | 2 | 0.39665 | 0.4335 | 0.053706 |
| 4 | 0.032838 | 3 | 0.28444 | 0.35631 | 0.049399 |
| 5 | 0.025992 | 4 | 0.2516 | 0.29106 | 0.044693 |
| 6 | 0.023181 | 5 | 0.22561 | 0.28243 | 0.044645 |
| 7 | 0.015676 | 6 | 0.20243 | 0.30617 | 0.04823 |
| 8 | 0.012987 | 7 | 0.18675 | 0.29096 | 0.046732 |
| 9 | 0.008455 | 8 | 0.17377 | 0.28935 | 0.046828 |
| 10 | 0.007079 | 9 | 0.16531 | 0.27013 | 0.04239 |
| 11 | 0.007028 | 10 | 0.15823 | 0.26855 | 0.042379 |
| 12 | 0.006666 | 11 | 0.1512 | 0.26802 | 0.042354 |
| 13 | 0.004618 | 12 | 0.14454 | 0.26735 | 0.042419 |
| 14 | 0.003154 | 13 | 0.13992 | 0.26753 | 0.042213 |
| 15 | 0.002968 | 14 | 0.13676 | 0.26056 | 0.041768 |
| 16 | 0.002811 | 15 | 0.1338 | 0.26165 | 0.041767 |
| 17 | 0.00264 | 16 | 0.13099 | 0.26138 | 0.04124 |
| 18 | 0.002453 | 17 | 0.12835 | 0.26059 | 0.041239 |
| 19 | 0.002436 | 18 | 0.12589 | 0.26097 | 0.041236 |
| 20 | 0.00241 | 19 | 0.12346 | 0.26149 | 0.041235 |
| 21 | 0.001608 | 20 | 0.12105 | 0.26117 | 0.041257 |
| 22 | 0.00147 | 21 | 0.11944 | 0.26187 | 0.041261 |
| 23 | 0.001389 | 22 | 0.11797 | 0.26095 | 0.041255 |
| 24 | 0.001288 | 23 | 0.11658 | 0.2616 | 0.041243 |
| 25 | 0.001246 | 25 | 0.114 | 0.25957 | 0.041044 |
| 26 | 0.001003 | 26 | 0.11276 | 0.25775 | 0.039879 |
| 27 | 0.001 | 27 | 0.11176 | 0.25898 | 0.039895 |

Table4: Results of cp plot with showing different split and errors

The root node error for this tree is 80.439

The MSE for in-sample data for this large tree was calculated using the predict() function as 8.989568

In the above table we see that the in-sample error(rel error) deceases with increasing model complexity but cv error(xerror) doesn’t necessarily. So, we decide to prune the tree such that the complexity is less and cv error is also minimum. This was achieved using prune() function.

  
Figure4: Regression tree with cp= 0.07

With cp 0.007 we obtain a comparatively less complex model and reduced cv error as shown in above figure.

Now we can evaluate the performance of this pruned tree using the same predict function. The MSE for training and testing data for this pruned tree was found out as below:

|  |  |  |
| --- | --- | --- |
| **Column1** | **In-Sample** | **Out-of-Sample** |
| MSE | 12.16268 | 23.74635 |

Table5: Prediction error for pruned regression tree

We can see that the out-of-sample MSE decreases from 25.91 to 23.74 for the pruned tree.

### Bagging:

Bagging (Bootstrap + Aggregation) is a method for reducing the variance by building multiple tree models on bootstrapped data, and aggregating the results of all trees.

We now fit the model using bagging technique to improve the prediction accuracy. It fits a tree for each bootstrap sample, and then aggregate the predicted values from all these different trees. We used nbagg = 100 in our case.

In bagging, all 13 variables are used for each split of the tree. Now, we will find out how well does the model perform with testing data by calculating test MSE which was found to be 20.76787. This is significantly small as compared to the test MSE for the previous tree model. Also, for a single tree the test MSE was found as 25.91 so this model has higher prediction accuracy compared to single tree. Hence, we can say that bagging improves the prediction accuracy. It is worth noting here that each of these trees are grown fully and not pruned so each tree has high variance and low bias, bagging averages these trees thus reducing the variance which improves the accuracy. The number of trees is not a critical parameter here and increasing the number of tree will not lead to overfitting.

|  |  |  |
| --- | --- | --- |
| **Column1** | **In-Sample** | **Out-of-Sample** |
| MSE | 10.86187 | 20.76787 |

Table6: Prediction error for bagging

In order to find out the number of trees that will give the best prediction we plot a graph between number of trees and MSE test for those number of trees.

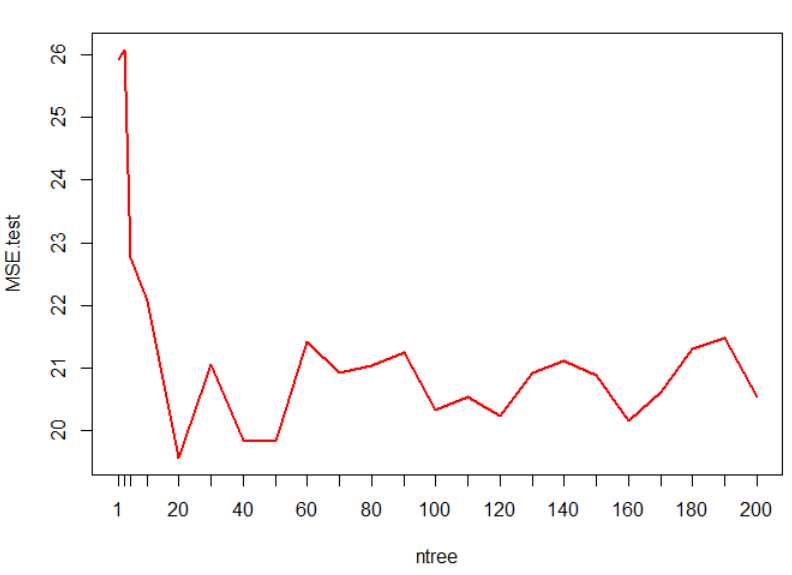


Figure5: cp plot to see the prediction accuracy for different ntree value

Out-of-Bag Prediction:

 In every bootstrap, the unused sample serves as testing sample, and testing error is calculated. OOB prediction of root mean squared error by default, is obtained which is 3.9714

.

### Random Forest:

Random forest provides an improvement over the bagged trees by a way that decorrelates the trees. The way it does this is by random sampling m predictors as split candidates from the full set of p predictors.

We fit a random forest model on the Boston Housing Data using randomForest() function. We usually take m= p/3 for regression tree and m= root p for classification tree. Using the default value(here mtry= 4 and ntree=500) we get MSE as 10.81. Next, we use different value of mtry(mtry=6) to find the MSE. Here, MSE was found to be 10.03 which is slightly less. This indicates that random forests yielded an improvement over bagging in this case.

Using the importance() function, we can view the importance of each variable.

The MSR is MSE of out-of-bag prediction. We make a plot we see how the OOB error changes with different ntree.

|  |  |  |
| --- | --- | --- |
|  | %IncMSE | IncNodePurity |
| crim | 5.918602 | 1504.6788 |
| zn | 0.2165649 | 103.9238 |
| indus | 4.2135965 | 1177.8502 |
| chas | 0.2132766 | 131.6888 |
| nox | 8.4816348 | 1691.8784 |
| rm | 40.3402572 | 11025.9477 |
| age | 2.8968836 | 754.3813 |
| dis | 7.0019996 | 2098.7756 |
| rad | 2.1903751 | 232.9921 |
| tax | 5.8523022 | 982.1319 |
| ptratio | 5.1951071 | 1453.3578 |
| black | 1.0820818 | 417.966 |
| lstat | 66.3968828 | 10102.574 |

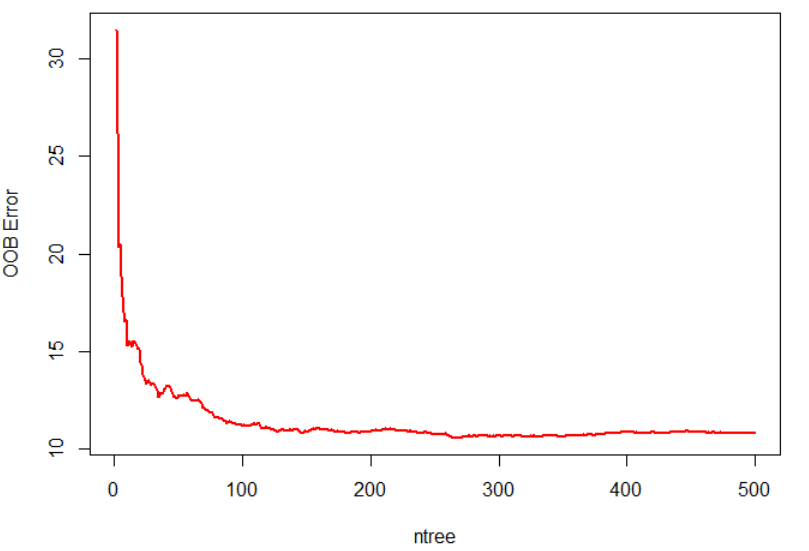
Table7: Significant of each predictor  


Figure6: Plot to see OOB error for different ntree values

We can check the MSE error using predict() which is 12.8006.

With ntree= 100, the model was built and the MSE was found which was 12.721 that is not too different fom ntree=500.

We can plot MSE and mtry value to see how OOB error and test error changes with the mtry value. This can be achieved using matplot() function.

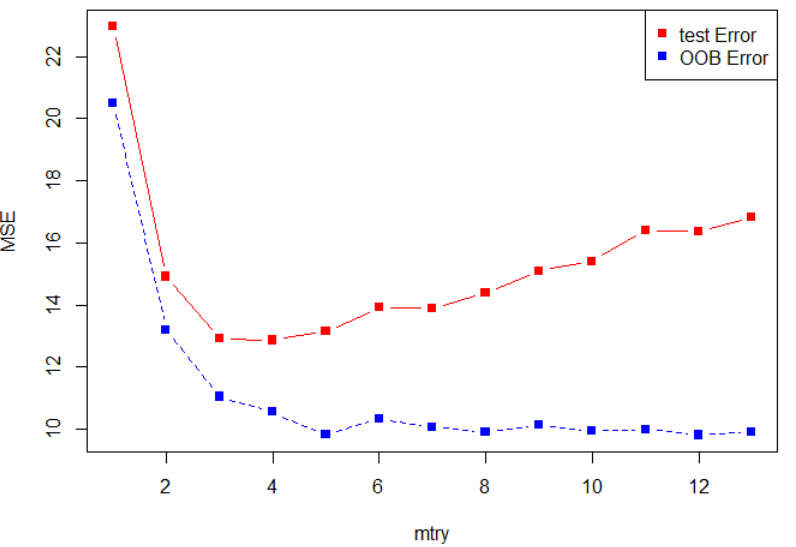


Figure7: Plot to see OOB and test error for different mtry values

From the above curve, we can see that blue line is the OOB error and red line is test error. Both curves are quite smooth and seems somewhat correlated. Both the errors tend to minimize at mtry= 4. The right most points indicate mtry=13 which is bagging.

|  |  |  |
| --- | --- | --- |
| **Mtry** | **oob.err** | **test.err** |
| 1 | 20.49276 | 22.96428 |
| 2 | 13.185608 | 14.90973 |
| 3 | 11.048005 | 12.92239 |
| 4 | 10.551702 | 12.86456 |
| 5 | 9.814673 | 13.1511 |
| 6 | 10.319375 | 13.92431 |
| 7 | 10.057806 | 13.87943 |
| 8 | 9.910603 | 14.38058 |
| 9 | 10.127106 | 15.10288 |
| 10 | 9.941375 | 15.40602 |
| 11 | 9.982359 | 16.40804 |
| 12 | 9.809509 | 16.37559 |
| 13 | 9.895213 | 16.82611 |

Table8: Prediction error for random forest

We can observe that the predictive performance of the model has increased by using random forest where only a split of m predictors are using for building the tree each time. Decorrelating the trees helps in making the average of resulting trees less variable and hence more reliable.

### Boosting:

|  |  |  |
| --- | --- | --- |
| **Column1** | **var** | **rel.inf** |
| rm | rm | 36.74534 |
| lstat | lstat | 31.16255 |
| dis | dis | 9.837353 |
| crim | crim | 4.727512 |
| nox | nox | 4.24953 |
| age | age | 4.177415 |
| black | black | 2.886539 |
| ptratio | ptratio | 2.468622 |
| tax | tax | 1.457777 |
| indus | indus | 1.092741 |
| rad | rad | 0.5377 |
| chas | chas | 0.526689 |
| zn | zn | 0.130234 |

A boosting model was fit on the training data using gbm() function with gaussian distribution, 10000 trees, shrinkage 0.01 and interaction depth as 8.

Table9: Relative influence for each predictor using boosting

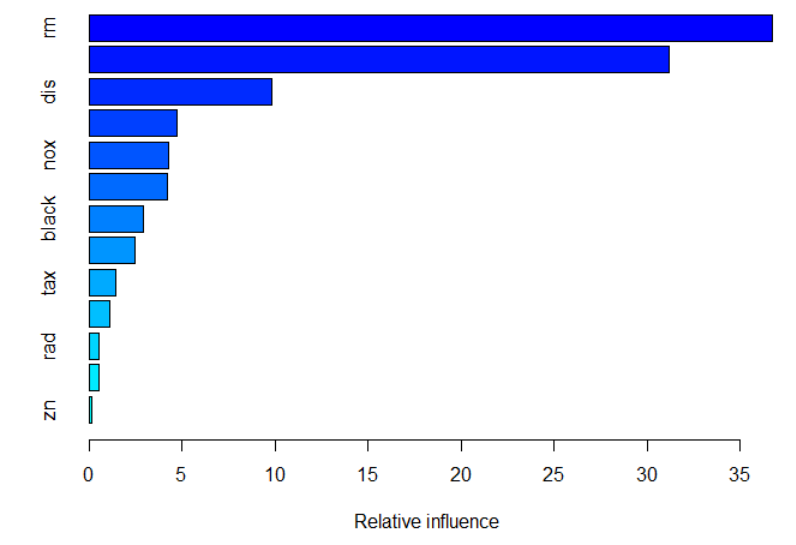


Figure8: Summary of boosting model

Here, we can see that rm and lstat are the most influential variables.

The out-of-sample MSE was found to be 11.0053.

|  |  |  |
| --- | --- | --- |
| **Column1** | **In-Sample** | **Out-of-Sample** |
| MSE | 0.0159493 | 11.0053 |

Table10: Prediction error for Boosting

The fitted boosted tree also gives the relation between response and each predictor.

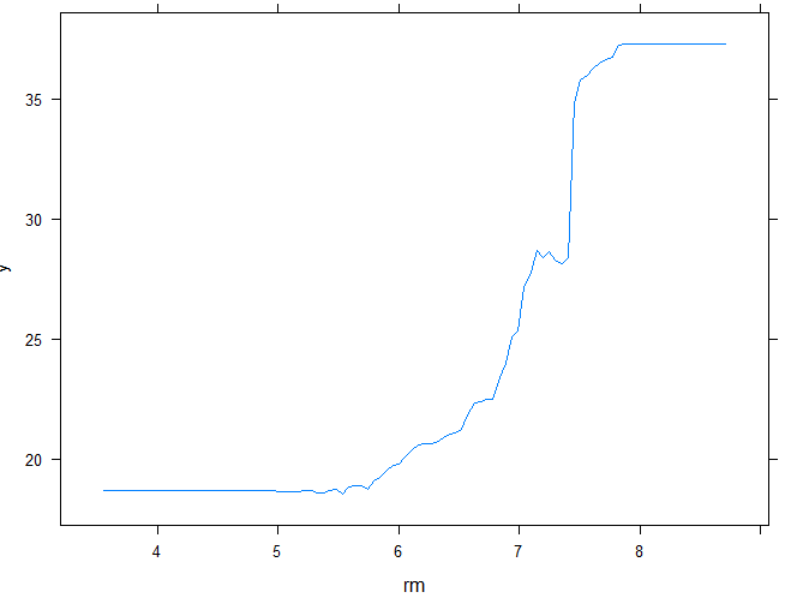
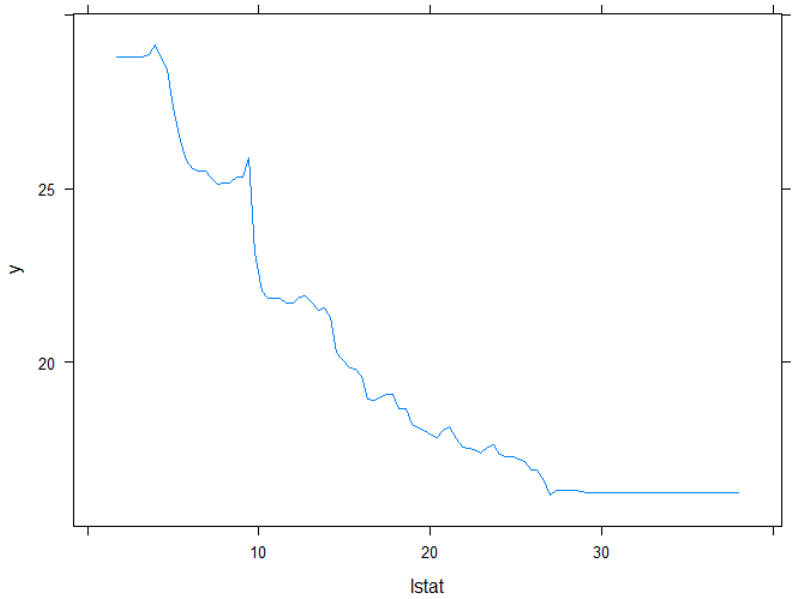


Figure9: Relation between each predictor and response

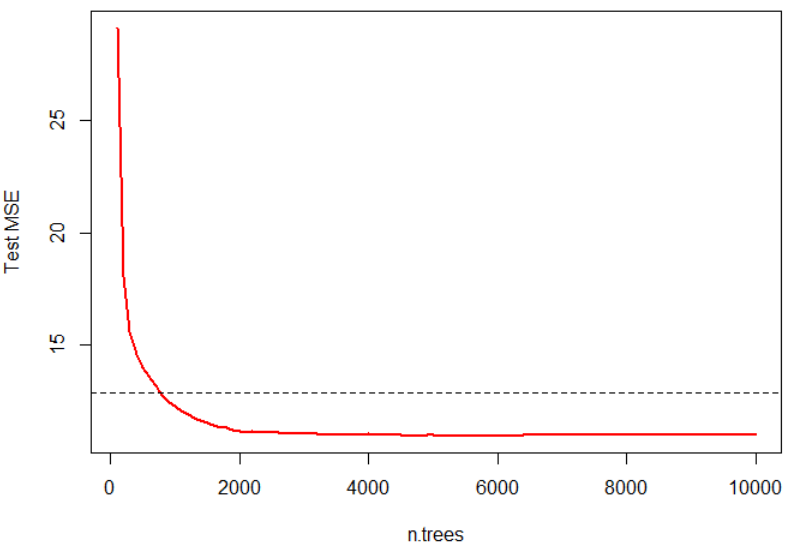
We can plot a graph between number of trees and MSE test to find out how the testing error changes with different number of trees.   


Figure10: Plot showing test error for different ntree value

The horizontal line shows the best prediction error.

Decision Trees themselves are bad in Prediction on test set, but when used with Ensembling Techniques like Bagging , Random Forests etc their Predictive perfomance are improved a lot.