Lab 22 - Random Forests

Chad Murchison, Guy Twa, Eddie-Williams Owiredu

2023-11-06

Today's Lab

Today's lab will introduce you to decision trees and random forest models for regression. We will learn how to fit decision trees, and the random forest decion tree ensemble method. We'll also learn how to interpret random forest models and assess feature importance.

Loading Packages and Data

For today's lab, we will be using a few new packages called `randomForest`, `randomForestExplainer`, `tree`, and `MASS` which you should go ahead and install prior to loading it in.

```
install.packages("randomForest")
install.packages("randomForestExplainer")
install.packages("tree")
install.packages("MASS")
```

```
# Load libraries
library(randomForest)
library(randomForestExplainer)
library(tree)
library(MASS)

# Set your seed
set.seed(123456)

# Load in the Boston Housing data from the MASS package
data(Boston)
```

Today, we'll be using the boston housing dataset, derived from 1970 U.S. census data. and published by by Harrison, D. and Rubinfeld, D.L. in `Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978. This study looked at how a range of factors impacted boston residents willingness to pay for a marginal increase in air quality, and how best to calculate this willingness. For our lab today, we'll be using this dataset to predict the median value of a home with these 13 variables.

- 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
- pratio = pupil-teacher ratio by town
- black = 1000(Bk - 0.63)^2 where Bk is the proportion of African-American population by town
- lstat = % lower socioeconomic status of the population
- medv = Median value of owner-occupied homes in $1000's

We'll go ahead and partition out our training and testing sets, using 80% of the data for training.

```
# Randomly sample 80% of the observation indices from our data set to determine our trai
ning set
train_vec <- sort(sample(1:nrow(Boston), floor(nrow(Boston) * 0.8)))
# Select the remaining sample indices for our testing set
test_vec <- setdiff(1:nrow(Boston), train_vec)

# Subset the Boston data set using our training and testing data indices
Boston_train <- Boston[train_vec,]
Boston_test <- Boston[test_vec,]
```

Basic Regression Tree

Let's start by building a basic decision tree using the `tree()` method form the `tree` package. This method will automatically select variables for us. It takes a standard formula expression as input.

```
basic_Boston_tree <- tree::tree(medv ~ .,
                                data = Boston_train, #Could also use `data = Boston, sub
set = train_vec`
                                split = "deviance")   #Deviance is the default loss func
tion, gini index is the other option
```

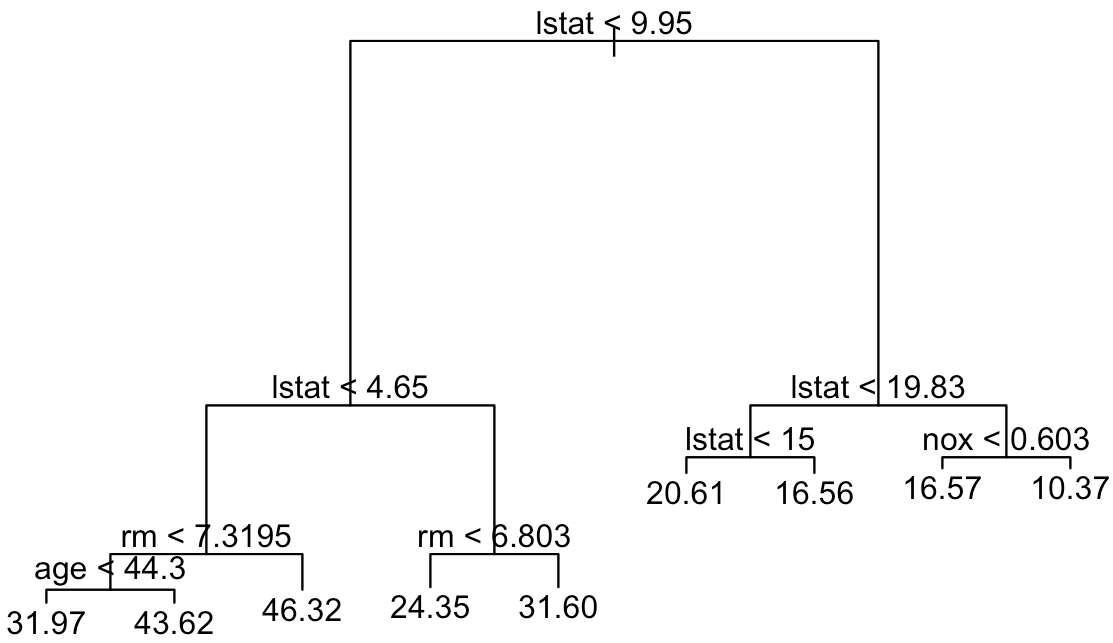
We can take a look at some of the important information about our tree using the `summary()` function. This will tell us about the variables actually used in tree construction, the number of terminal nodes, and the deviance for evaluation.

```
summary(basic_Boston_tree)
```

```
##
## Regression tree:
## tree::tree(formula = medv ~ ., data = Boston_train, split = "deviance")
## Variables actually used in tree construction:
## [1] "lstat" "rm"    "age"   "nox"
## Number of terminal nodes:  9
## Residual mean deviance:  13.85 = 5472 / 395
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -10.220  -2.264  -0.261   0.000   2.190  25.650
```

To view our decision tree, we can use the basic plot function. This gives the criterion for the splits while the values in the terminal node are the predictions, in this case the Median Home Value for each criteria group.

```
plot(basic_Boston_tree); text(basic_Boston_tree)
```

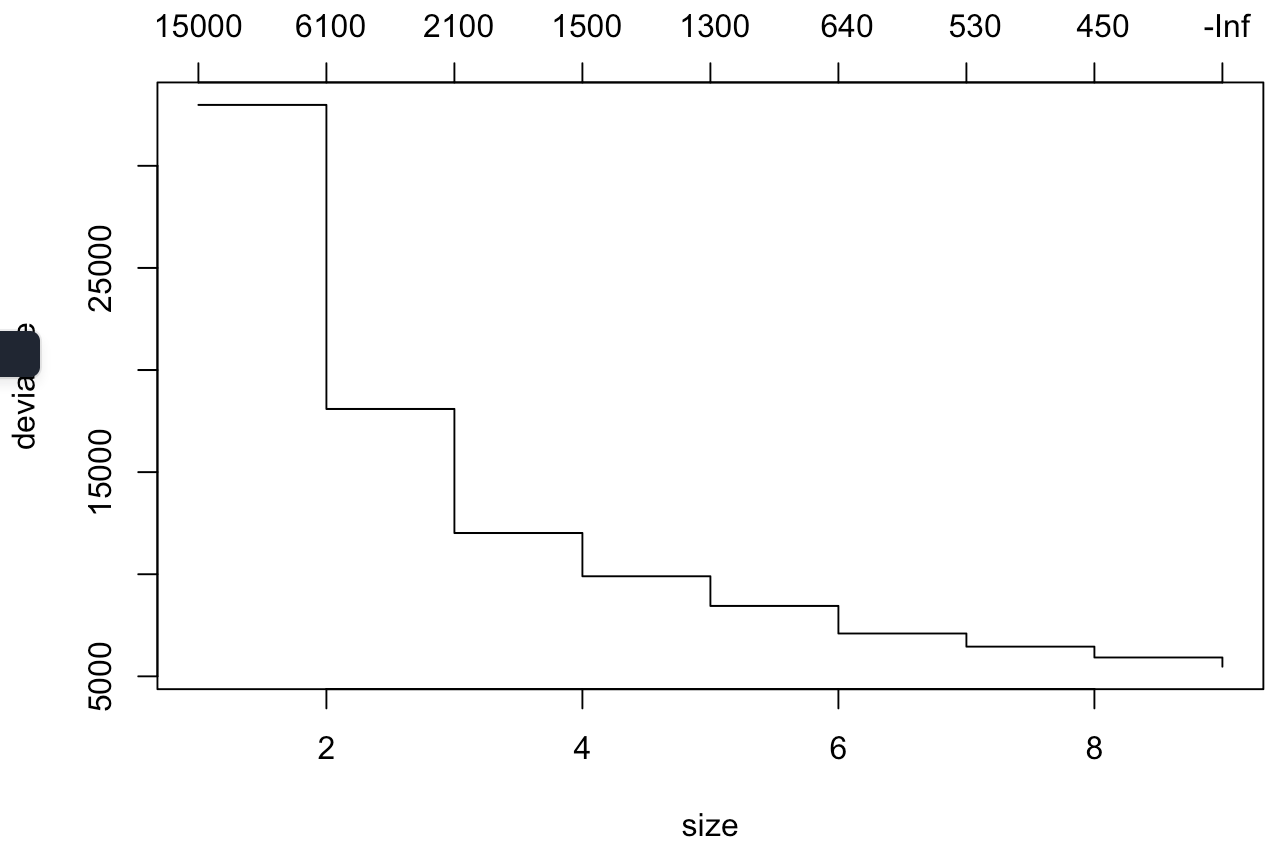


Tree pruning

We can also prune the tree, by removing less important splits from the bottom of the tree. To do this, we use the `prune.tree()` function. This lets us select a particular number of nodes we want or we can select it based on some sweep.

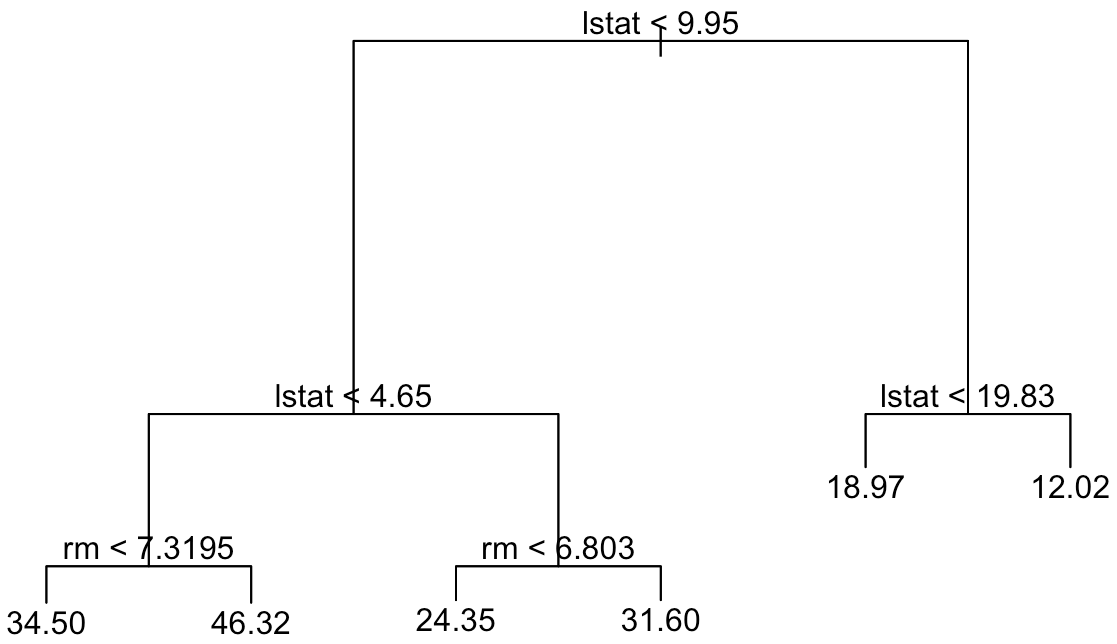
To determine the number of nodes we want, we can run `prune.tree()` without any parameters and plot the result.

```
prune_Boston_tree <- prune.tree(basic_Boston_tree)
plot(prune_Boston_tree)
```



We can see that after six nodes, we see a fairly minimal step size in the deviance. This means the first six nodes are the most important for redicing model deviance. Let's prune our tree, and use this as our pruning criteria as the `best` argument.

```
prune_Boston_tree <- prune.tree(basic_Boston_tree, best = 6)
plot(prune_Boston_tree)
text(prune_Boston_tree, pretty = 0)
```



When we compare the deviance of our pruned and unpruned tree…

```
summary(basic_Boston_tree)
```

```
##
## Regression tree:
## tree::tree(formula = medv ~ ., data = Boston_train, split = "deviance")
## Variables actually used in tree construction:
## [1] "lstat" "rm"    "age"   "nox"
## Number of terminal nodes:  9
## Residual mean deviance:  13.85 = 5472 / 395
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -10.220  -2.264  -0.261   0.000   2.190  25.650
```

```
summary(prune_Boston_tree)
```

```
##
## Regression tree:
## snip.tree(tree = basic_Boston_tree, nodes = c(7L, 8L, 6L))
## Variables actually used in tree construction:
## [1] "lstat" "rm"
## Number of terminal nodes:  6
## Residual mean deviance:  17.83 = 7098 / 398
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -10.67000  -2.67800   0.04667   0.00000   2.42700  25.65000
```

… we see that we don't reduce the deviance as much, but we do get a simpler tree with only two variables.

Note: We won't look at it here, but you can also do cross-validation using `cv.tree()`

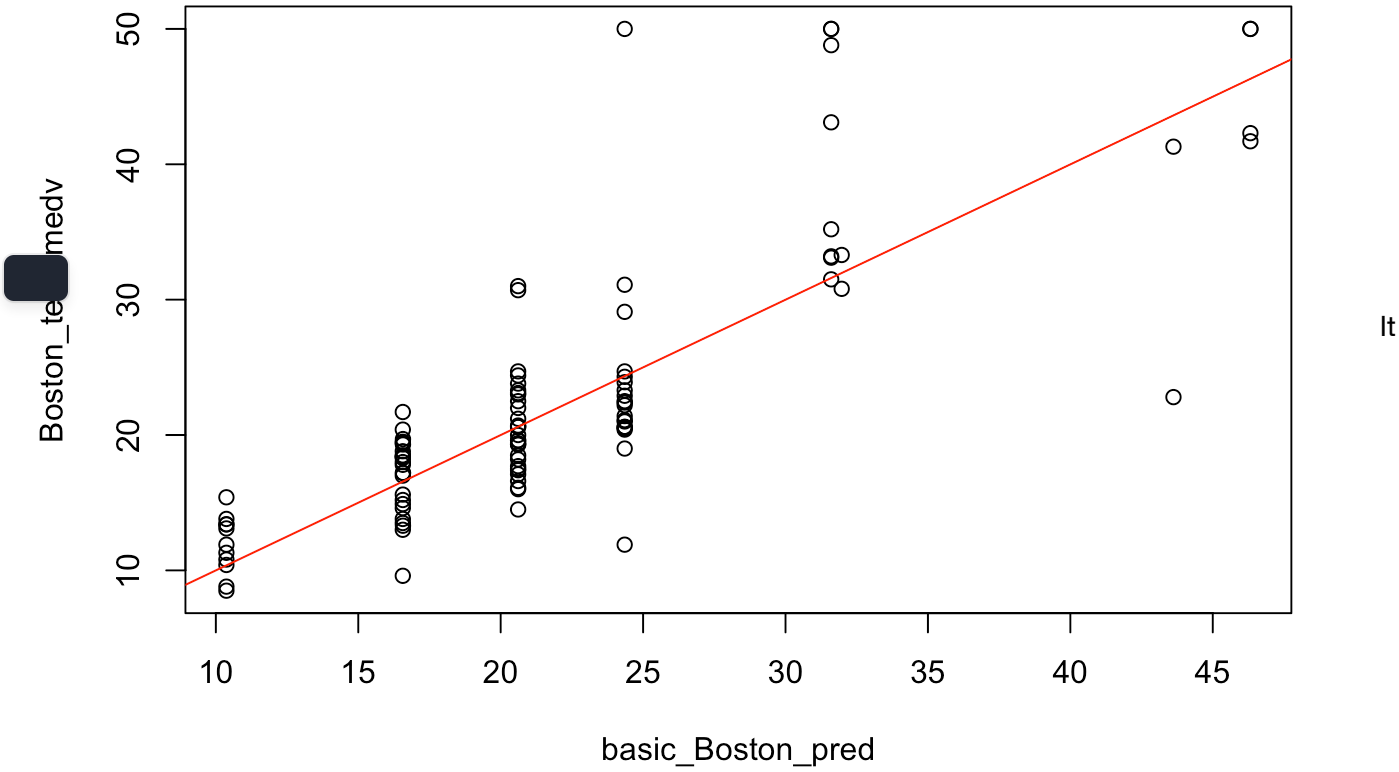
Prediction

Back to our original tree, we can evaluate the tree's performance on median home value in our test data. To do this, we use the `predict()` function.

```
basic_Boston_pred <- predict(basic_Boston_tree, newdata = Boston_test)
```

We can visualize the relationship between our predicted values and our real values using the `plot()` function. Let's add a red line with a slope of 1 for reference. If our predictions match the true values, then points should follow this line closely.

```
plot(basic_Boston_pred, Boston_test$medv)
abline(0,1, col="red")
```



looks like we're not too far off for lower median home values, but there's a lot to be desired on the upper end of the scale.

Finally, let's calculate the MSE on our test set.

```
basic_MSE <- mean((basic_Boston_pred - Boston_test$medv)^2)
sqrt(basic_MSE)
```

```
## [1] 5.723498
```

Our estimates are within ~$5,723 of the true median value. Not to bad, but we can do better!

Random Forest

Random forest models are built an an ensemble, or collection, of many small decision trees built with different sets of features whose results are aggregated for the final decision. They have the same hyperparameters as decision trees (e.g. tree depth, number of branches, and sampling proportions) but they have the additional hyperparameter for the number of trees.

We can build a random forest with the `randomForest()` function from the `randomForest` package. The function takes the same formula notation as input to define the outcome variable (`medv` in this case) and predictor variables. Additionally, it has the parametrs `ntree` to set the number of tree and `mtry` to set the number of predictors to sample.

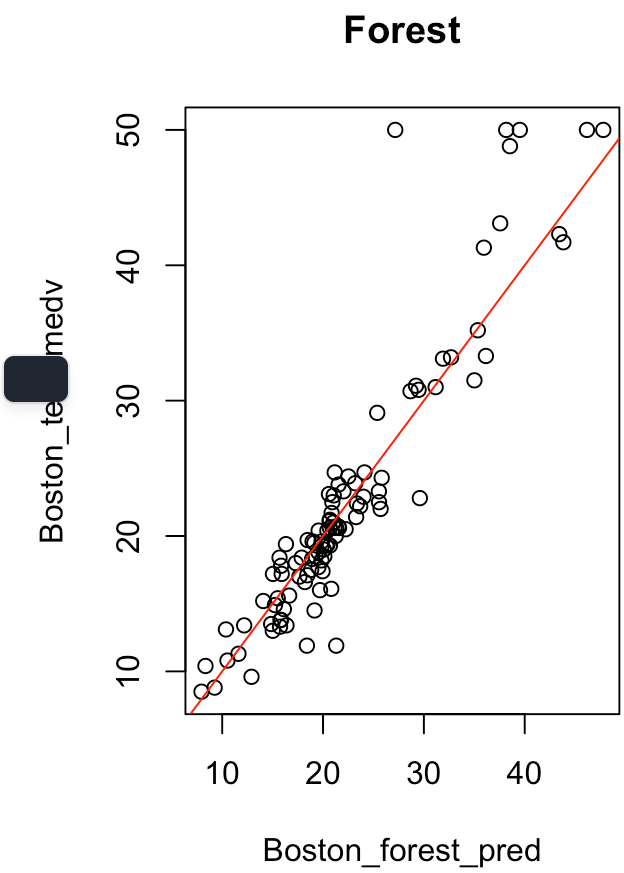
```
Boston_forest <- randomForest(medv ~ ., data = Boston_train,
                              ntree = 500,  #The number of trees to grow in your forest, default is 500
                              #mtry = ...,  #The number of predictors to sample, determined heuristically by default as 1/3*(p) (reg) or sqrt(p) (classification)
                              importance = TRUE,  #We can also determine importance of variables
                              )
Boston_forest
```

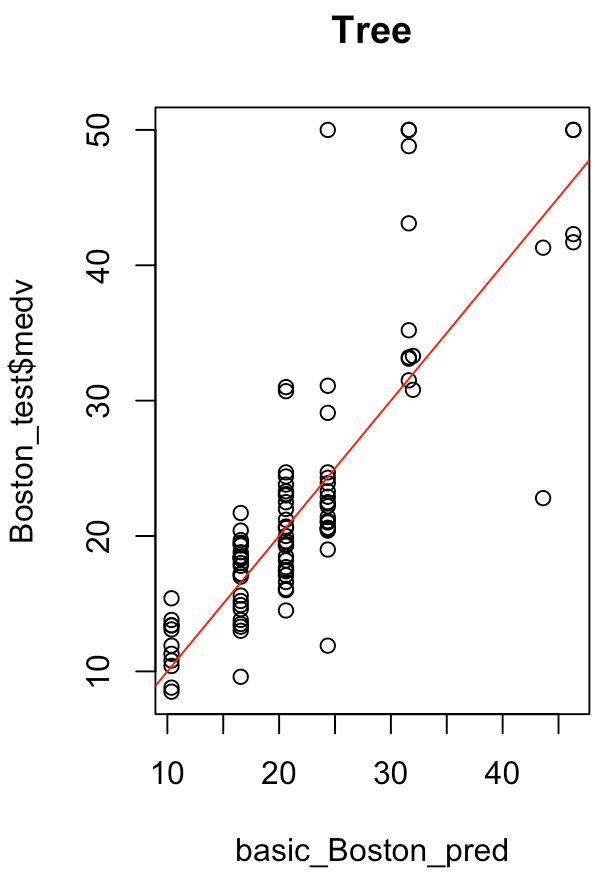
```
#
## Call:
##  randomForest(formula = medv ~ ., data = Boston_train, ntree = 500,      importance = TRUE, )
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 9.878208
##                     % Var explained: 87.9
```

Note: We can perform bagging (wherein we build FULL trees with all covariates) by setting `mtry` to the number of predictors, but we won't bother with that here.

As before, we can predict on the test set to see how the random forest compares to the single tree.

```
Boston_forest_pred <- predict(Boston_forest, newdata = Boston_test)
par(mfrow = c(1,2))
plot(Boston_forest_pred, Boston_test$medv, main = "Forest"); abline(0,1, col="red")
plot(basic_Boston_pred, Boston_test$medv, main = "Tree"); abline(0,1, col="red")
```






```
par(mfrow = c(1,1))
```

And we can look at how our MSE has improved.

```
forest_MSE <- mean((Boston_forest_pred - Boston_test$medv)^2)
sqrt(forest_MSE)
```

```
## [1] 3.760417
```

Now we're within ~$3,661 of the median home value

variable importance

We can also assess the importance of variables from the forest.

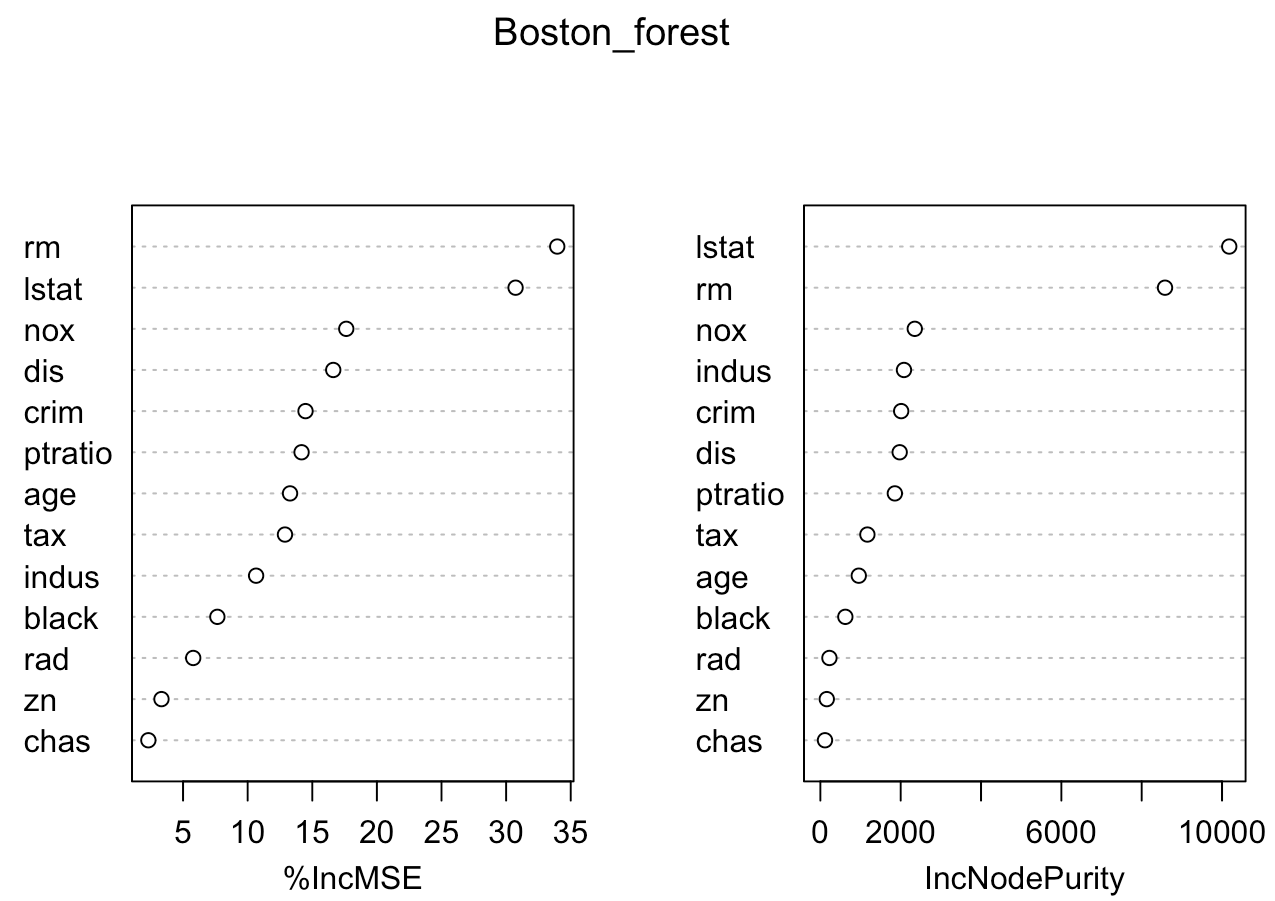
```
Boston_forest_imp <- importance(Boston_forest)
Boston_forest_imp[order(Boston_forest_imp[,1]),]
```

```
##           %IncMSE IncNodePurity
## chas     2.314870      115.2676
## zn       3.330210      159.8099
## rad      5.785157      225.6056
## black    7.654373      621.8057
## indus   10.652673     2081.6633
## tax     12.886911     1170.8417
## age     13.278339      956.3218
## ptratio 14.171450     1854.4443
## crim    14.476028     2012.1819
## dis     16.622300     1975.0658
## nox     17.628881     2350.5625
## lstat   30.732164    10179.4515
## rm      33.956200     8580.4766
```

We see lstat and rm are still the two most important although our forest gives greater PERCENT MSE improvement for rm unlike the single tree. We can also visualize based on improvements in MSE or purity ()

Additionally, we can plot this using the `varImpPlot()` function for a graphical representation.

```
varImpPlot(Boston_forest)
```



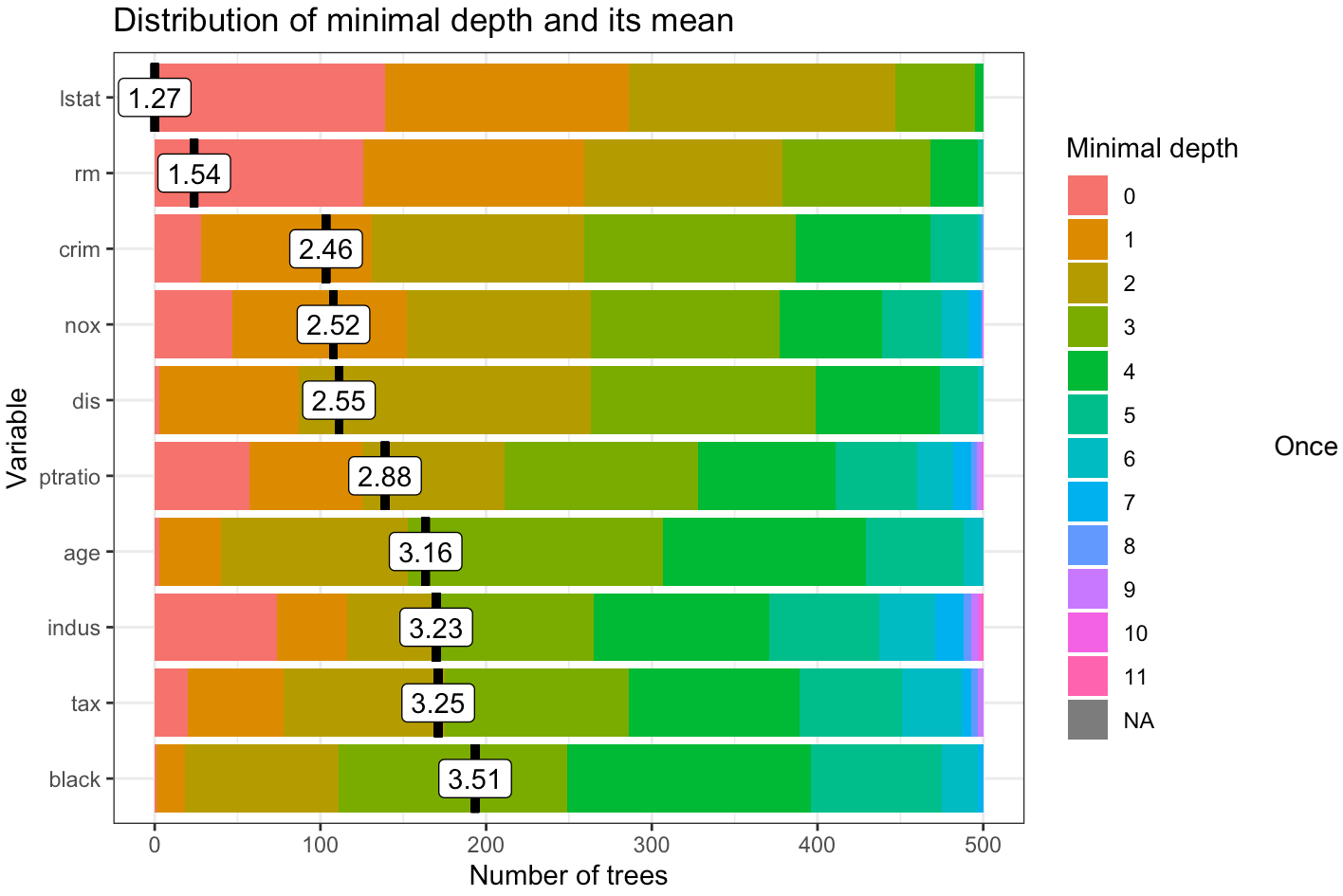
Extension with randomForestExplainer

`randomForestExplainer` is a set of tools to help pick apart our random forest model and explain which variables are most important.

Mininal depth

We look at the minimal depth of our features. This is a measure of how early a variable appears in decision trees in the fores on average. By plotting this, we can get an idea of the distribution across the forest.

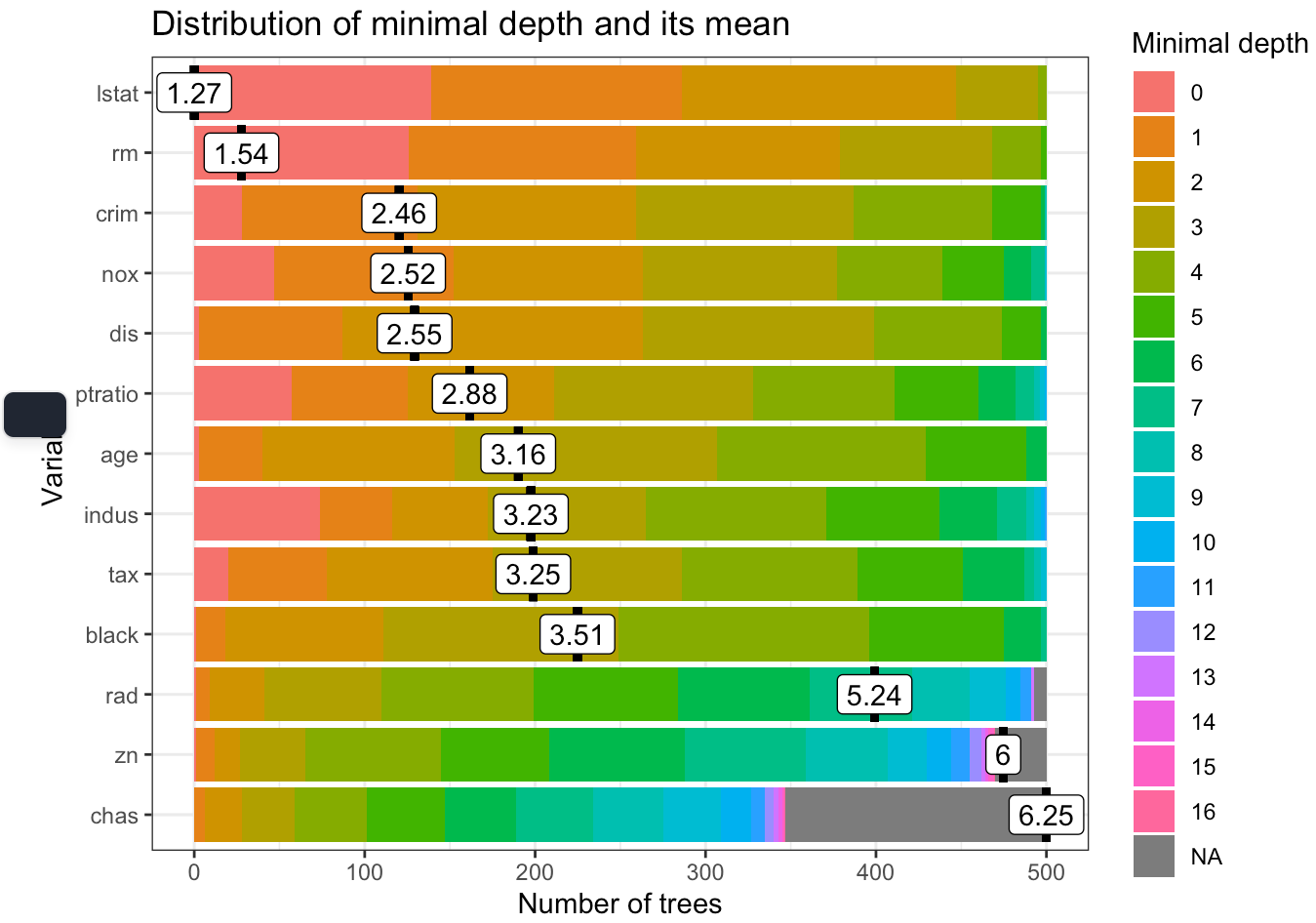
```
Boston_min_depth <- min_depth_distribution(Boston_forest)
plot_min_depth_distribution(Boston_min_depth,
                            k = 10,  #k is the number variables to be shown
                            mean_sample = "top_trees"  #This is the default, it mainly d
etermines how trees with missing variables are used
                            )
```



again, lstat and rm tend to appear earlier.

This can be slightly adjusted based on what trees are returned rather than fill in values we can use only relevant trees. This won't have a huge effect on simple trees like this but may not always be the case. If nothing, it won't impact the order of depth.

```
plot_min_depth_distribution(Boston_min_depth, mean_sample = "relevant_trees", k = ncol(B
oston)-1)
```



Feature importance

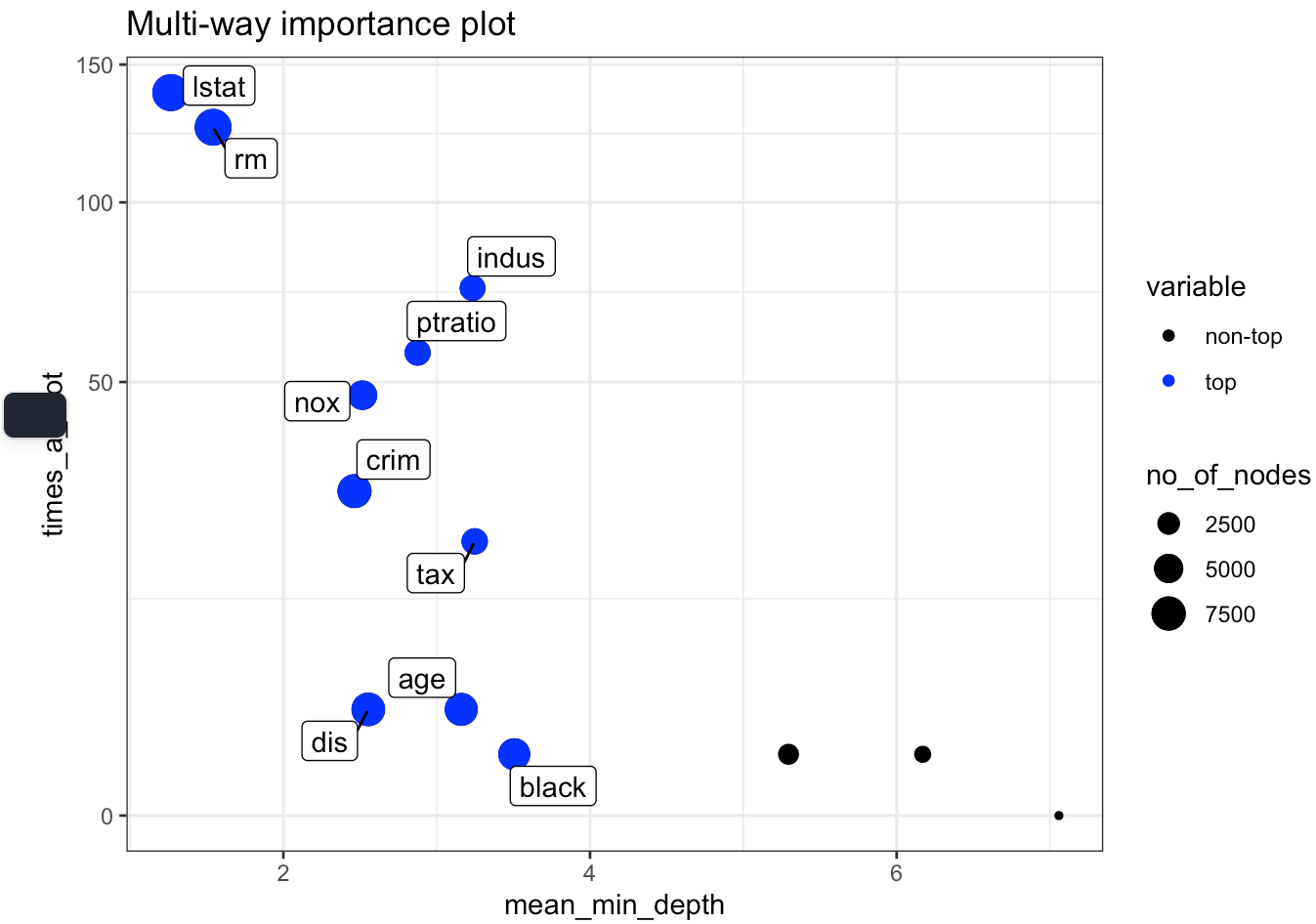
We can also get a more elaborate explanation of variable importance from the forest using the `measure_importance()` function. Some of the important outcomes are the numebr of times a variable is a root node (the first decision), the number of trees in which a split on that variable occurs, the total number of nodes using that variable as a split, a p-value on whether the probability of having a split on that variable is greater than chance.

```
Boston_forest_imp <- measure_importance(Boston_forest, mean_sample = "top_trees")
Boston_forest_imp
```

```
##    variable mean_min_depth no_of_nodes mse_increase node_purity_increase
## 1       age       3.160000        6797    4.3102726             956.3218
## 2     black       3.506000        6288    1.0788438             621.8057
## 3      chas       7.058176         559    0.1007223             115.2676
## 4      crim       2.464000        7351    7.4523501            2012.1819
## 5       dis       2.554000        7147    6.4674432            1975.0658
## 6     indus       3.234000        3597    5.6242422            2081.6633
## 7     lstat       1.266000        9113   61.2527338           10179.4515
## 8       nox       2.516000        5120   11.3536589            2350.5625
## 9   ptratio       2.876000        3713    6.0045586            1854.4443
## 10      rad       5.294544        2114    0.8713163             225.6056
## 11       rm       1.542000        8987   29.2381635            8580.4766
## 12      tax       3.248000        3787    4.0535960            1170.8417
## 13       zn       6.169760        1268    0.6303892             159.8099
##    no_of_trees times_a_root       p_value
## 1          500            3 2.290109e-129
## 2          500            1  2.978415e-67
## 3          347            0  1.000000e+00
## 4          500           28 6.893385e-218
## 5          500            3 7.583795e-183
## 6          500           74  1.000000e+00
## 7          500          139  0.000000e+00
## 8          500           47  2.111847e-01
## 9          500           57  1.000000e+00
## 10         493            1  1.000000e+00
## 11         500          126  0.000000e+00
## 12         500           20  1.000000e+00
## 13         470            1  1.000000e+00
```

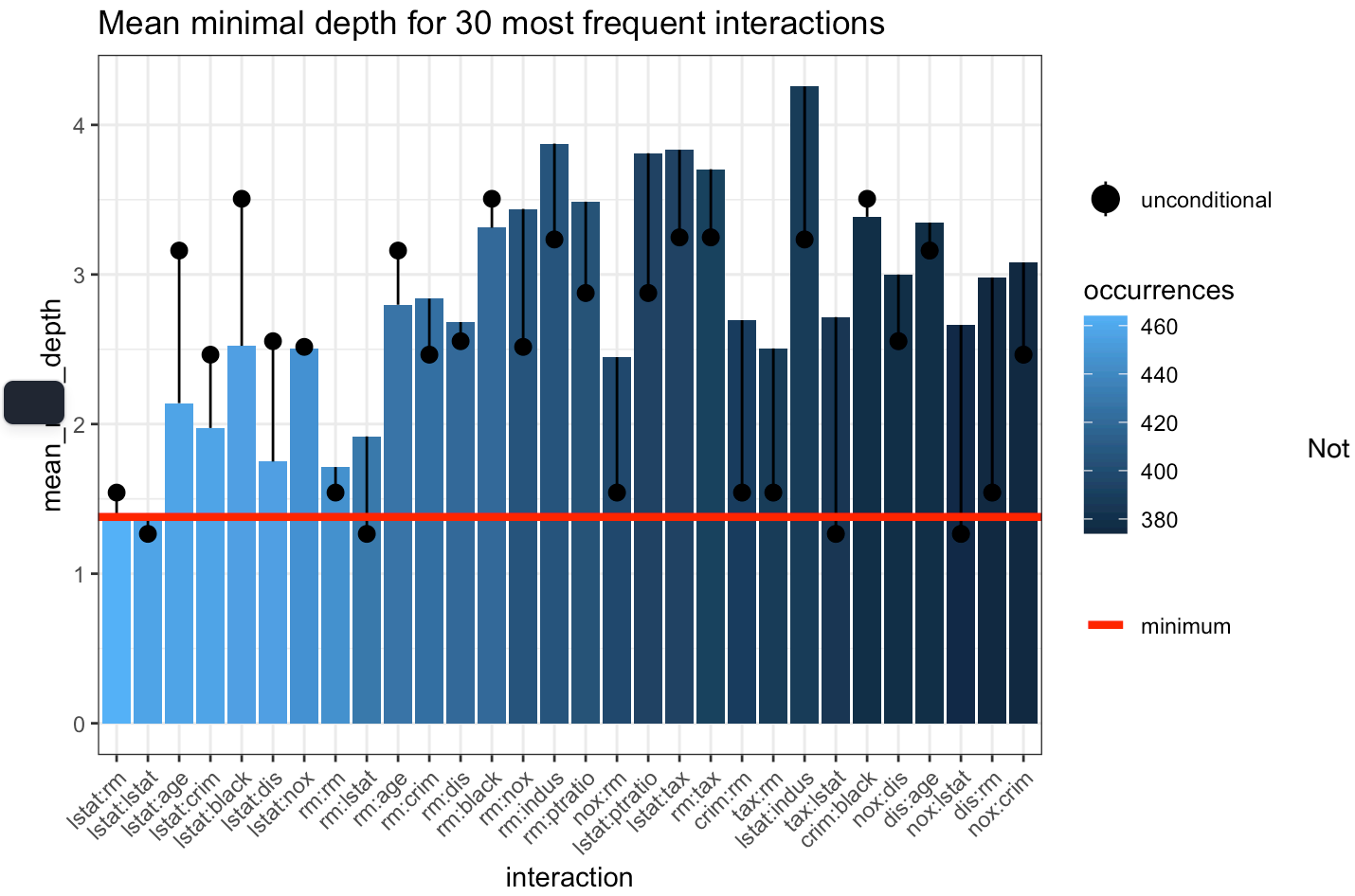
We can also plot importance using `plot_multi_way_importance()` , it can take any of the columns in the importance dataframe as x,y,size. This isn't elaborated upon here but you can compare importance measure pairwise using the plot_importance_ggpairs() and plot_importance_rankings() functions

```
plot_multi_way_importance(Boston_forest_imp,
                          x = "mean_min_depth", y = "times_a_root",
                          size_measure = "no_of_nodes"
                          )
```



Finally, we can look at pairwise interactions of the depth of our variables to see what variables are used as downstream splits in subtrees. So to get an idea how if you split on one variable, how quickly does the next variables show up

```
Boston_forest_imp_var <- important_variables(Boston_forest_imp, measures = c("mean_min_depth", "no_of_trees"))
Boston_forest_imp_var_itx <- min_depth_interactions(Boston_forest, Boston_forest_imp_var)
plot_min_depth_interactions(Boston_forest_imp_var_itx)
```



surprisingly, the lstat and rm and jointly tied since they're so powerfully predictive