House Price Analysis with ML

Howard Nguyen

2022-11-14

Load the Dataset

The dataset is available in the mlbench package. Let's start o by loading the required packages and loading the dataset.

```
# load packages
library(mlbench)
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
# attach the BostonHousing dataset
data(BostonHousing)
```

Validation Dataset

It is a good idea to use a validation hold out set. This is a sample of the data that we hold back from our analysis and modeling. We use it right at the end of our project to confirm the accuracy of our final model. It is a smoke test that we can use to see if we messed up and to give us confidence on our estimates of accuracy on unseen data.

```
# Split out validation dataset
# create a list of 80% of the rows in the original dataset we can use for training
set.seed(7)
validationIndex <- createDataPartition(BostonHousing$medv, p=0.80, list=FALSE)
# select 20% of the data for validation
validation <- BostonHousing[-validationIndex,]
# use the remaining 80% of data to training and testing the models
dataset <- BostonHousing[validationIndex,]
```

Analyze Data

The objective of this step in the process is to better understand the problem.

Descriptive Statistics

Let's start off by confirming the dimensions of the dataset, e.g. the number of rows and columns.

```
# dimesions of dataset
dim(dataset)
```

```
## [1] 407  14
```

We have 407 instances to work with and can confirm the data has 14 attributes including the class attribute medv.

Let's also look at the data types of each attribute.

```
# lsit types for each attribute
sapply(dataset, class)
```

```
##      crim        zn     indus      chas       nox        rm       age       dis
## "numeric" "numeric" "numeric"  "factor" "numeric" "numeric" "numeric" "numeric"
##       rad       tax   ptratio         b     lstat      medv
## "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
```

We can see that one of the attributes (chas) is a factor while all of the others are numeric.

Let's now take a peak at the first 20 rows of the data.

```
head(dataset, n = 20)
```

```
##       crim   zn indus chas   nox    rm   age    dis rad tax ptratio      b
## 1  0.00632 18.0  2.31    0 0.538 6.575  65.2 4.0900   1 296    15.3 396.90
## 2  0.02731  0.0  7.07    0 0.469 6.421  78.9 4.9671   2 242    17.8 396.90
## 3  0.02729  0.0  7.07    0 0.469 7.185  61.1 4.9671   2 242    17.8 392.83
## 4  0.03237  0.0  2.18    0 0.458 6.998  45.8 6.0622   3 222    18.7 394.63
## 5  0.06905  0.0  2.18    0 0.458 7.147  54.2 6.0622   3 222    18.7 396.90
## 6  0.02985  0.0  2.18    0 0.458 6.430  58.7 6.0622   3 222    18.7 394.12
## 7  0.08829 12.5  7.87    0 0.524 6.012  66.6 5.5605   5 311    15.2 395.60
## 10 0.17004 12.5  7.87    0 0.524 6.004  85.9 6.5921   5 311    15.2 386.71
## 11 0.22489 12.5  7.87    0 0.524 6.377  94.3 6.3467   5 311    15.2 392.52
## 12 0.11747 12.5  7.87    0 0.524 6.009  82.9 6.2267   5 311    15.2 396.90
## 13 0.09378 12.5  7.87    0 0.524 5.889  39.0 5.4509   5 311    15.2 390.50
## 14 0.62976  0.0  8.14    0 0.538 5.949  61.8 4.7075   4 307    21.0 396.90
## 15 0.63796  0.0  8.14    0 0.538 6.096  84.5 4.4619   4 307    21.0 380.02
## 17 1.05393  0.0  8.14    0 0.538 5.935  29.3 4.4986   4 307    21.0 386.85
## 20 0.72580  0.0  8.14    0 0.538 5.727  69.5 3.7965   4 307    21.0 390.95
## 21 1.25179  0.0  8.14    0 0.538 5.570  98.1 3.7979   4 307    21.0 376.57
## 22 0.85204  0.0  8.14    0 0.538 5.965  89.2 4.0123   4 307    21.0 392.53
## 23 1.23247  0.0  8.14    0 0.538 6.142  91.7 3.9769   4 307    21.0 396.90
## 24 0.98843  0.0  8.14    0 0.538 5.813 100.0 4.0952   4 307    21.0 394.54
## 25 0.75026  0.0  8.14    0 0.538 5.924  94.1 4.3996   4 307    21.0 394.33
##    lstat medv
## 1   4.98 24.0
## 2   9.14 21.6
## 3   4.03 34.7
```

```
## 4   2.94 33.4
## 5   5.33 36.2
## 6   5.21 28.7
## 7  12.43 22.9
## 10 17.10 18.9
## 11 20.45 15.0
## 12 13.27 18.9
## 13 15.71 21.7
## 14  8.26 20.4
## 15 10.26 18.2
## 17  6.58 23.1
## 20 11.28 18.2
## 21 21.02 13.6
## 22 13.83 19.6
## 23 18.72 15.2
## 24 19.88 14.5
## 25 16.30 15.6
```

We can confirm that the scales for the attributes are all over the place because of the differing units. We may benefit from some transforms later on.

Let's summarize the distribution of each attribute

```
summary(dataset)
```

```
##       crim                zn             indus       chas         nox
##  Min.   : 0.00632   Min.   :  0.00   Min.   : 0.74   0:378   Min.   :0.3850
##  1st Qu.: 0.07964   1st Qu.:  0.00   1st Qu.: 4.93   1: 29   1st Qu.:0.4480
##  Median : 0.26838   Median :  0.00   Median : 8.56           Median :0.5380
##  Mean   : 3.63725   Mean   : 11.92   Mean   :11.00           Mean   :0.5547
##  3rd Qu.: 3.68567   3rd Qu.: 15.00   3rd Qu.:18.10           3rd Qu.:0.6310
##  Max.   :88.97620   Max.   :100.00   Max.   :27.74           Max.   :0.8710
##        rm             age              dis              rad
##  Min.   :3.561   Min.   :  6.20   Min.   : 1.130   Min.   : 1.000
##  1st Qu.:5.888   1st Qu.: 42.70   1st Qu.: 2.109   1st Qu.: 4.000
##  Median :6.212   Median : 77.30   Median : 3.152   Median : 5.000
##  Mean   :6.290   Mean   : 68.38   Mean   : 3.818   Mean   : 9.595
##  3rd Qu.:6.619   3rd Qu.: 94.20   3rd Qu.: 5.215   3rd Qu.:24.000
##  Max.   :8.780   Max.   :100.00   Max.   :12.127   Max.   :24.000
##       tax           ptratio            b              lstat
##  Min.   :187.0   Min.   :12.60   Min.   :  0.32   Min.   : 1.920
##  1st Qu.:280.5   1st Qu.:17.00   1st Qu.:373.81   1st Qu.: 7.065
##  Median :334.0   Median :19.00   Median :391.27   Median :11.320
##  Mean   :408.7   Mean   :18.42   Mean   :357.19   Mean   :12.556
##  3rd Qu.:666.0   3rd Qu.:20.20   3rd Qu.:396.02   3rd Qu.:16.455
##  Max.   :711.0   Max.   :22.00   Max.   :396.90   Max.   :37.970
##       medv
##  Min.   : 5.00
##  1st Qu.:17.05
##  Median :21.20
##  Mean   :22.52
##  3rd Qu.:25.00
##  Max.   :50.00
```

We can note that ‘chas’ is a pretty unbalanced factor. We could transform this attribute to numeric to make calculating descriptive statistics and plots easier.

Let’s go ahead and convert ‘chas’ to a numberic attribute.

```
dataset[,4] <- as.numeric(as.character(dataset[,4]))
```

Now, let’s now take a look at the correlation between all of the numeric attributes.

```
cor(dataset[,1:13])
```

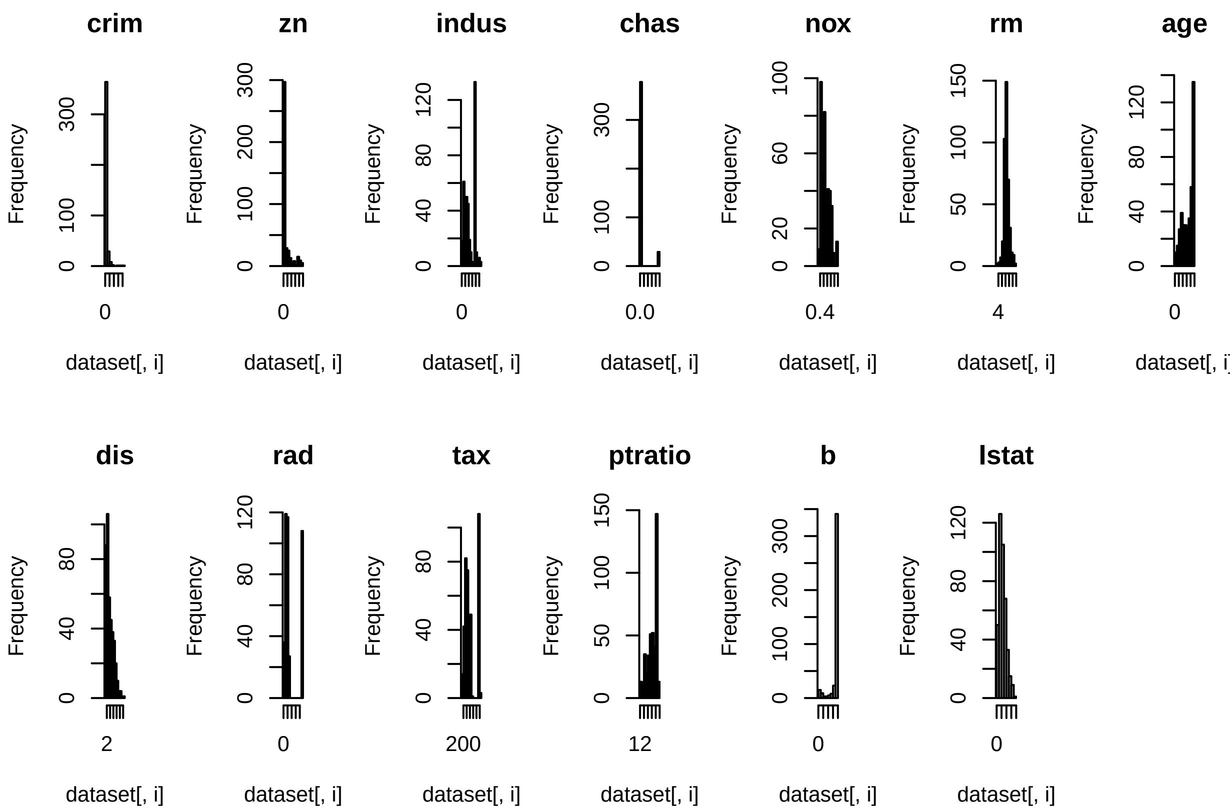
```
##                crim          zn      indus         chas        nox         rm
## crim     1.00000000 -0.19958871  0.4075617 -0.055071619  0.4099499 -0.1939805
## zn      -0.19958871  1.00000000 -0.5314446 -0.029869089 -0.5202171  0.3110684
## indus    0.40756169 -0.53144458  1.0000000  0.065834605  0.7732538 -0.3826220
## chas    -0.05507162 -0.02986909  0.0658346  1.000000000  0.0933997  0.1267673
## nox      0.40994989 -0.52021705  0.7732538  0.093399699  1.0000000 -0.2960506
## rm      -0.19398049  0.31106841 -0.3826220  0.126767331 -0.2960506  1.0000000
## age      0.35240196 -0.58447039  0.6511547  0.073501472  0.7338299 -0.2261571
## dis     -0.37564969  0.67986750 -0.7113431 -0.099046145 -0.7693426  0.2074524
## rad      0.60834345 -0.32273403  0.6199844 -0.002445292  0.6276047 -0.2212616
## tax      0.57112501 -0.31839453  0.7185102 -0.030644325  0.6757625 -0.2952685
## ptratio  0.28970459 -0.38877931  0.3781852 -0.122831776  0.1887657 -0.3648317
## b       -0.34424053  0.17471792 -0.3643714  0.037822587 -0.3683865  0.1259942
## lstat    0.42294190 -0.42188931  0.6135914 -0.084304578  0.5838934 -0.6119520
##                 age         dis          rad         tax    ptratio           b
## crim     0.35240196 -0.37564969  0.608343448  0.57112501  0.2897046 -0.34424053
## zn      -0.58447039  0.67986750 -0.322734034 -0.31839453 -0.3887793  0.17471792
## indus    0.65115470 -0.71134308  0.619984390  0.71851024  0.3781852 -0.36437143
## chas     0.07350147 -0.09904614 -0.002445292 -0.03064433 -0.1228318  0.03782259
## nox      0.73382994 -0.76934256  0.627604694  0.67576249  0.1887657 -0.36838649
## rm      -0.22615708  0.20745239 -0.221261630 -0.29526853 -0.3648317  0.12599418
## age      1.00000000 -0.74924225  0.468963375  0.50581099  0.2709298 -0.27418512
## dis     -0.74924225  1.00000000 -0.503720701 -0.52641543 -0.2279429  0.28427693
## rad      0.46896338 -0.50372070  1.000000000  0.92005320  0.4797148 -0.42313915
## tax      0.50581099 -0.52641543  0.920053202  1.00000000  0.4690561 -0.43026638
## ptratio  0.27092980 -0.22794289  0.479714803  0.46905607  1.0000000 -0.16996117
## b       -0.27418512  0.28427693 -0.423139153 -0.43026638 -0.1699612  1.00000000
## lstat    0.60664241 -0.50129941  0.502511782  0.55382142  0.4092786 -0.35088088
##               lstat
## crim     0.42294190
## zn      -0.42188931
## indus    0.61359140
## chas    -0.08430458
## nox      0.58389344
## rm      -0.61195203
## age      0.60664241
## dis     -0.50129941
## rad      0.50251178
## tax      0.55382142
## ptratio  0.40927864
## b       -0.35088088
## lstat    1.00000000
```

This is interesting. We can see that many of the attributes have a strong correlation (e.g. > 0.70 or < 0.70). For example: - nox and indus with 0.77 - dis with indus with 0.71 - tax and indus with 0.71 - age and nox with 0.73 - dist and nox with 0.77 This is collinearity and we may see better results with regression algorithms if the correlated attributes are removed.

Unimodal Data Visualizations

Let's look at visualizations of individual attributes. It is often useful to look at your data using multiple different visualizations in order to spark ideas. Let's look at histograms of each attribute to get a sense of the data distributions.

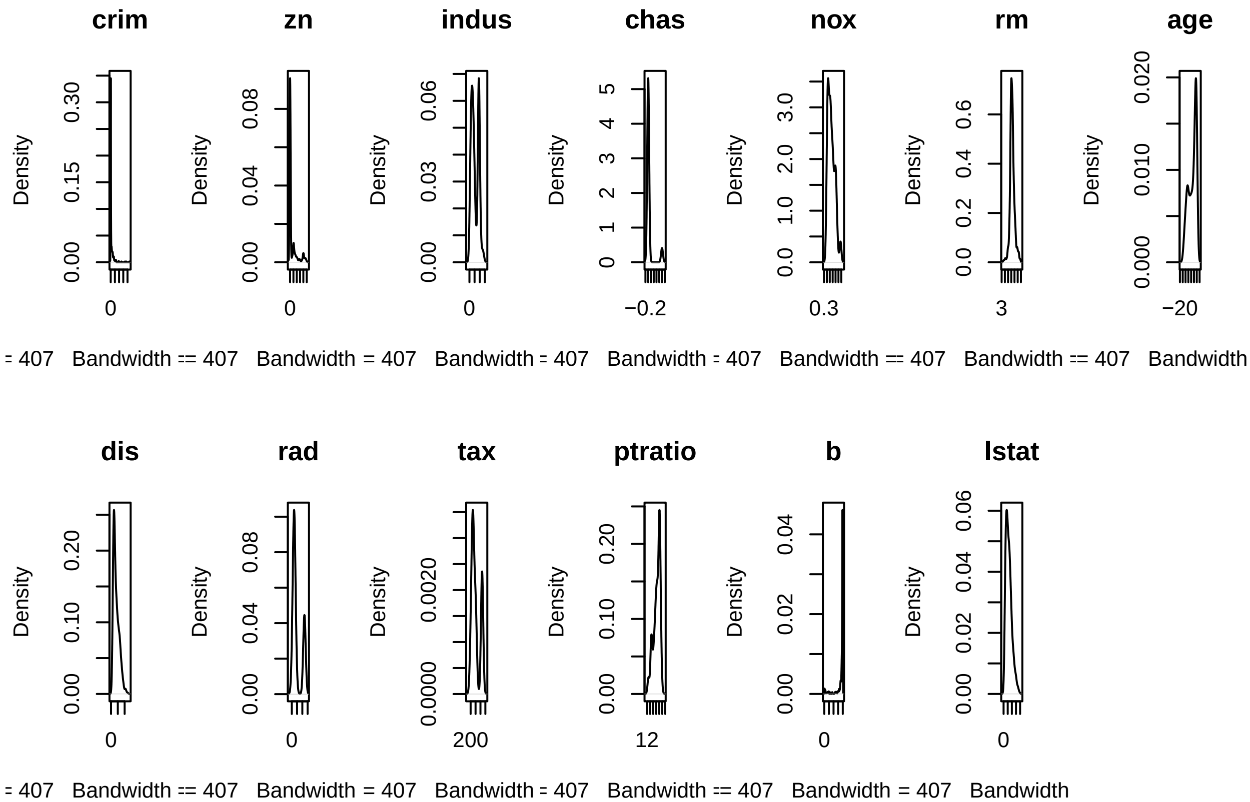
```
# histograms each attribute
par(mfrow=c(2,7))
for(i in 1:13) {
  hist(dataset[,i], main=names(dataset)[i])
}
```



We can see that some attributes may have an exponential distribution, such as crim, zn, age and b. We can see that others may have a bimodal distribution such as rad and tax.

Let's look at the same distributions using density plots that smooth them out a bit.

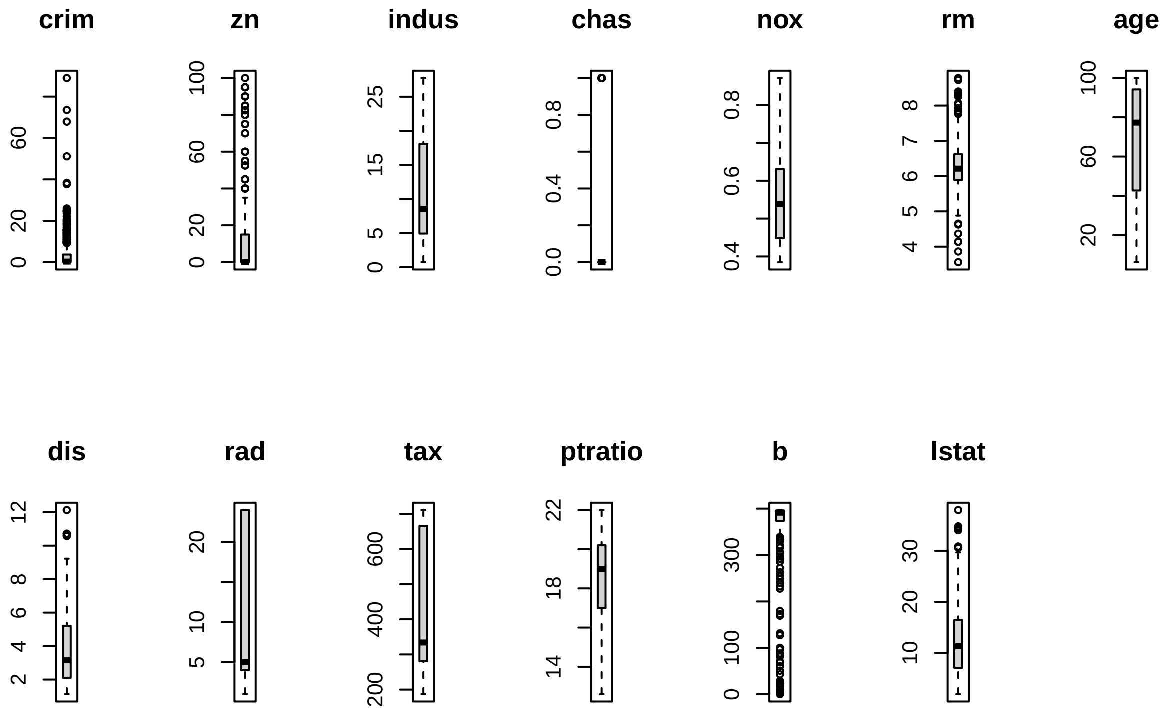
```
# density plot for each attribute
par(mfrow=c(2,7))
for(i in 1:13) {
  plot(density(dataset[,i]), main=names(dataset)[i])
}
```



This perhaps adds more evidence to our suspicion about possible exponential and bimodal distributions. It also looks like nox, rm and lstat may be skewed Gaussian distributions, which might be helpful later with transforms.

Let's look at the data with box and whisker plots of each attribute.

```
# boxplots for each attribute
par(mfrow=c(2,7))
for(i in 1:13) {
  boxplot(dataset[,i], main=names(dataset)[i])
}
```

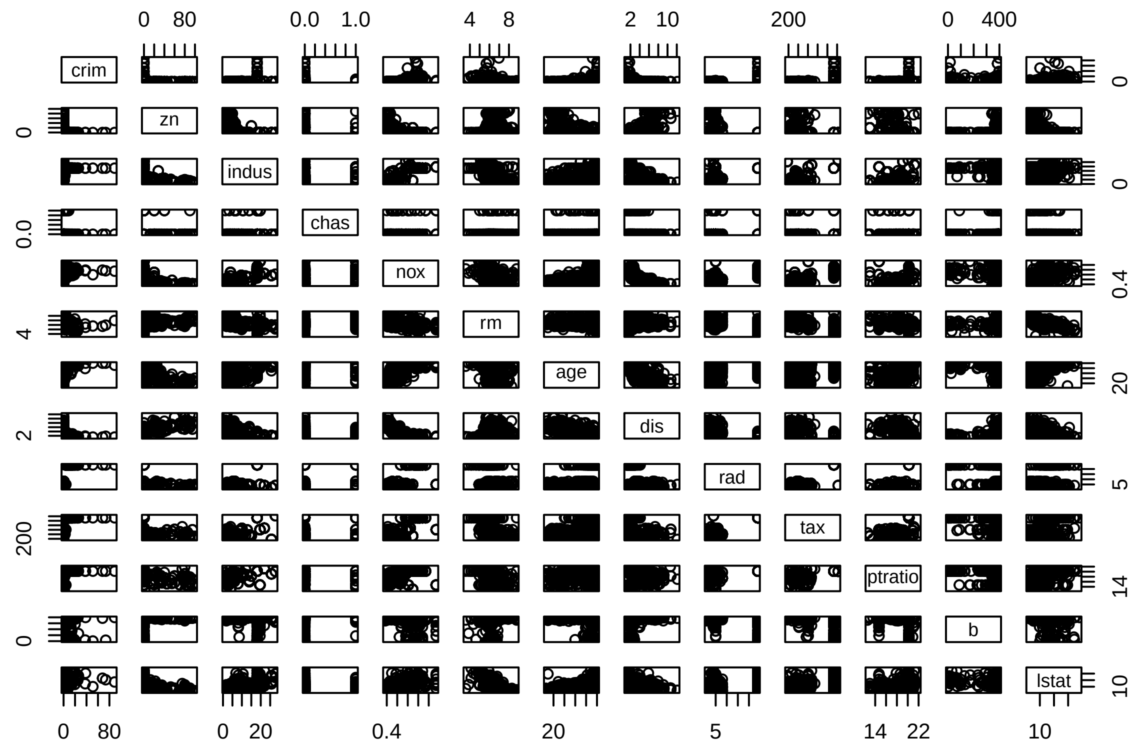


This helps point out the skew in many distributions so much so that data looks like outliers (e.g. beyond the whisker of the plots).

Multi modal Data Visualizations

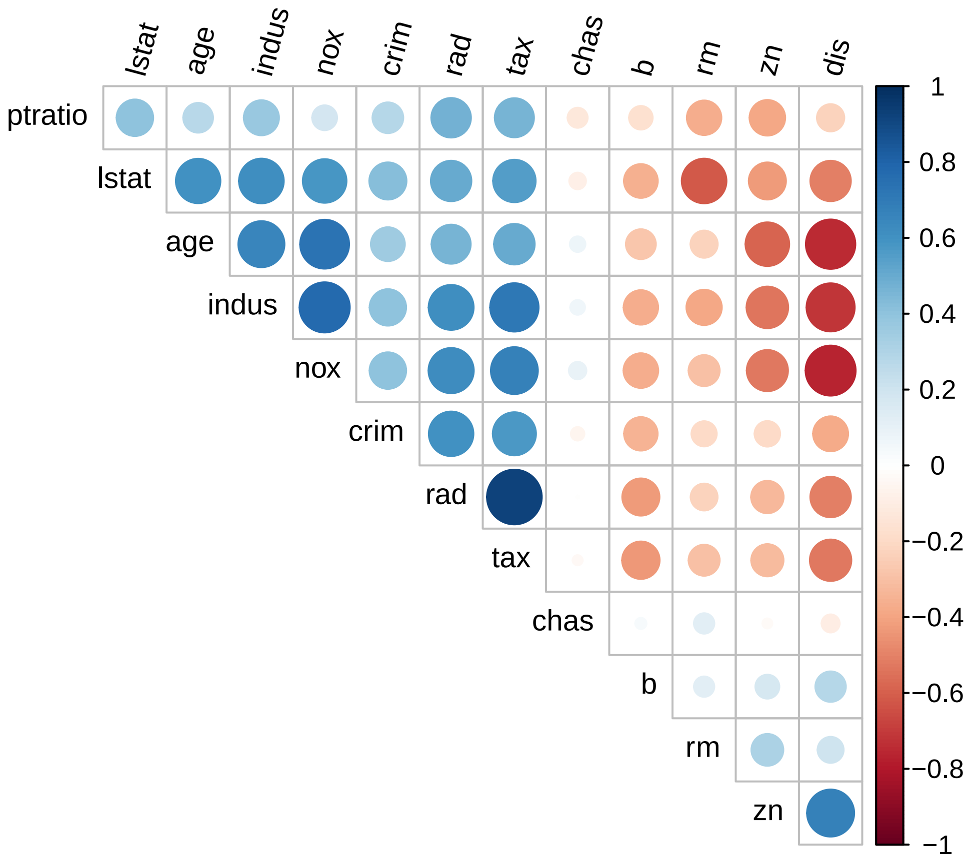
Let's look at some visualizations of the interactions between variables. The best place to start is a scatterplot matrix.

```
# scatterplot matrix
pairs(dataset[,1:13])
```



We can see that some of the higher correlated attributes do show good structure in their relationship. Not linear, but nice predictable curved relationships.

```
# correlation plot
correlations <- cor(dataset[,1:13])
corrplot(correlations, method = "circle",
         order = "hclust", # order for labels, can be "original"
         type = "upper",    # matrix: full, upper, lower
         diag = F,       # remove diagonal
         tl.cex = 0.9,   # font size
         tl.srt = 75, # label angel
         tl.col = "black",
         addrect = 8)
```



The larger darker blue dots confirm the positively correlated attributes we listed early. We can also see some larger darker red dots that suggest some negatively correlated attributes. For example nox and dis. These too may be candidates for removal to better improve accuracy of models later on.

Summary of Ideas

There is a lot of structure in this dataset. We need to think about transforms that we could use later to better expose the structure which in turn may improve modeling accuracy. So far it would be worth trying:

- Feature selection and removing the most correlated attributes.
- Normalizing the dataset to reduce the effect of differing scales.
- Standardizing the dataset to reduce the effects of differing distributions.
- Box-Cox transform to see if flattening out some of the distributions improves accuracy.

With lots of additional time I would also explore the possibility of binning (discretization) of the data. This can often improve accuracy for decision tree algorithms.

Evaluate Algorithms: Baseline

We have no idea what algorithms will do well on this problem. Gut feel suggests regression algorithms like GLM and GLMNET may do well. It is also possible that decision trees and even SVM may do well. I have no idea. Let's design our test harness. We will use 10-fold cross validation (each fold will be about 360 instances for training and 40 for test) with 3 repeats.

The dataset is not too small and this is a good standard test harness configuration. We will evaluate algorithms using the RMSE and R2 metrics. RMSE will give a gross idea of how wrong all predictions are (0 is perfect) and R2 will give an idea of how well the model has fit the data (1 is perfect, 0 is worst).

```
# Run algorithms using 10-fold cross validation - prepare the test harness for evaluating algorithms
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"
```

Let's create a baseline of performance on this problem and spot-check a number of different algorithms. We will select a suite of different algorithms capable of working on this regression problem. The 6 algorithms selected include:

- Linear Algorithms: Linear Regression (LR), Generalized Linear Regression (GLM) and Penalized Linear Regression (GLMNET)
- Non-Linear Algorithms: Classification and Regression Trees (CART), Support Vector Machines (SVM) with a radial basis function and k-Nearest Neighbors (KNN)

We know the data has differing units of measure so we will standardize the data for this baseline comparison. This will those algorithms that prefer data in the same scale (e.g. instance based methods and some regression algorithms) a chance to do well.

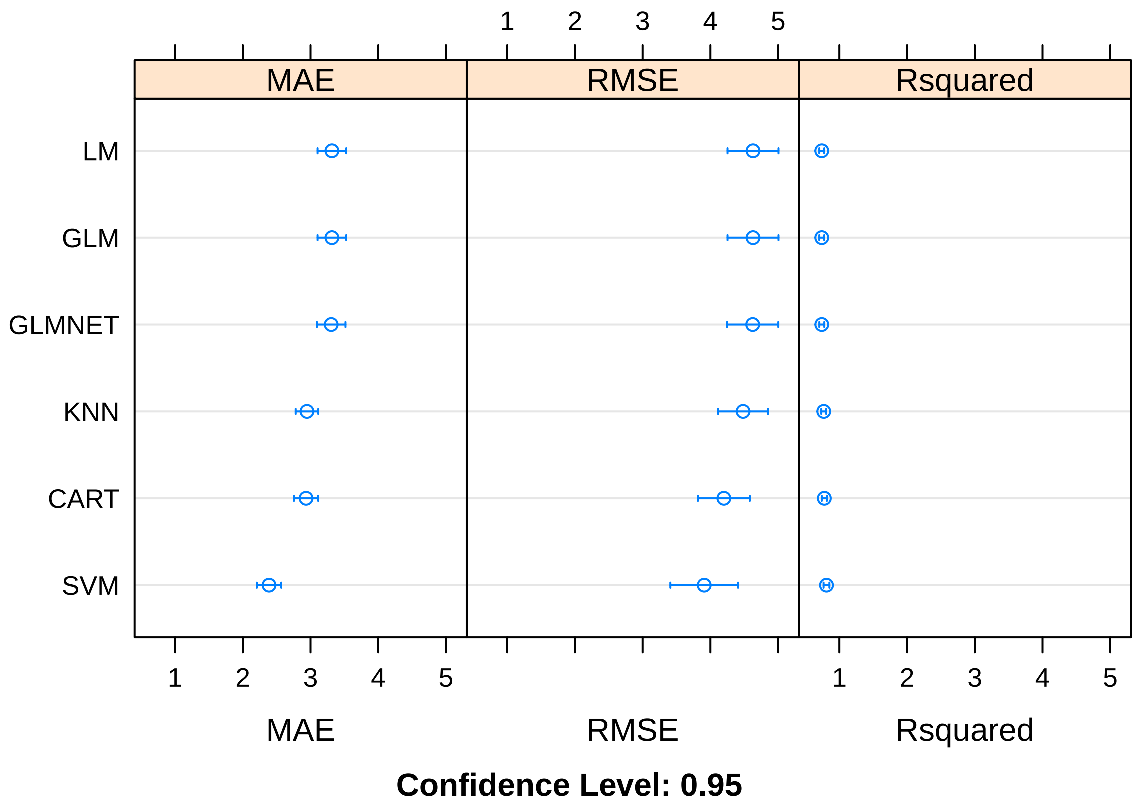
```
# Estimate accuracy of machine learning algorithms.
# LM
set.seed(7)
fit.lm <- train(medv~., data=dataset, method="lm", metric=metric, preProc=c("center", "scale"), trContro
# GLM
set.seed(7)
fit.glm <- train(medv~., data=dataset, method="glm", metric=metric, preProc=c("center", "scale"), trCon
# GLMNET
set.seed(7)
fit.glmnet <- train(medv~., data=dataset, method="glmnet", metric=metric, preProc=c("center", "scale"),
# SVM
set.seed(7)
fit.svm <- train(medv~., data=dataset, method="svmRadial", metric=metric, preProc=c("center", "scale"),
# CART
set.seed(7)
grid <- expand.grid(.cp=c(0, 0.05, 0.1))
fit.cart <- train(medv~., data=dataset, method="rpart", metric=metric, tuneGrid=grid, preProc=c("center
# KNN
set.seed(7)
fit.knn <- train(medv~., data=dataset, method="knn", metric=metric, preProc=c("center", "scale"), trCon
```

The algorithms all use default tuning parameters, except CART which is fussy on this dataset and has 3 default parameters specified. Let's compare the algorithms. We will use a simple table of results to get a quick idea of what is going on. We will also use a dot plot to show the 95% confidence level for the estimated metrics.

```
# Compare algorithms
results <- resamples(list(LM=fit.lm, GLM=fit.glm, GLMNET=fit.glmnet, SVM=fit.svm,
CART=fit.cart, KNN=fit.knn))
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: LM, GLM, GLMNET, SVM, CART, KNN
## Number of resamples: 30
##
## MAE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## LM     2.296870 2.897637 3.368197 3.315490 3.701331 4.644634    0
## GLM    2.296870 2.897637 3.368197 3.315490 3.701331 4.644634    0
## GLMNET 2.300353 2.883646 3.336121 3.304888 3.697932 4.625185    0
## SVM    1.422355 1.992094 2.516175 2.387478 2.654770 3.345278    0
## CART   2.216672 2.620729 2.883975 2.933934 3.081861 4.161930    0
## KNN    1.978049 2.685764 2.866806 2.947500 3.237153 3.998333    0
##
## RMSE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## LM     2.991624 3.869651 4.632032 4.629323 5.317398 6.694651    0
## GLM    2.991624 3.869651 4.632032 4.629323 5.317398 6.694651    0
## GLMNET 2.990832 3.878814 4.615843 4.624746 5.316638 6.692580    0
## SVM    2.049509 2.949716 3.814475 3.908781 4.455574 6.979067    0
## CART   2.766558 3.379400 3.997915 4.199124 4.600681 7.087656    0
## KNN    2.653686 3.741499 4.415526 4.482558 5.064124 6.976913    0
##
## Rsquared
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## LM     0.5050763 0.6744319 0.7474419 0.7404282 0.8127596 0.8999946    0
## GLM    0.5050763 0.6744319 0.7474419 0.7404282 0.8127596 0.8999946    0
## GLMNET 0.5033006 0.6730052 0.7474746 0.7410698 0.8155090 0.9039292    0
## SVM    0.5193410 0.7623629 0.8447175 0.8103317 0.8958487 0.9700899    0
## CART   0.5144771 0.7367598 0.8156225 0.7782237 0.8421169 0.8989506    0
## KNN    0.5187652 0.7478699 0.8043069 0.7699680 0.8293089 0.9307667    0
```

```
dotplot(results)
```



It looks like SVM has the lowest RMSE, followed closely by the other non-linear algorithms CART and KNN. The linear regression algorithms all appear to be in the same ball park and slightly worse error. We can also see that SVM and the other non-linear algorithms have the best fit for the data in their R2 measures.

Did centering and scaling make a difference to the algorithms other than KNN? I doubt it. But I prefer to hold the data constant at this stage. Perhaps the worse performance of the linear regression algorithms has something to do with the highly correlated attributes. Let's look at that in the next section.

Evaluate Algorithms: Feature Selection

We have a theory that the correlated attributes are reducing the accuracy of the linear algorithms tried in the base line spot-check in the last step. In this step we will remove the highly correlated attributes and see what effect that has on the evaluation metrics. We can find and remove the highly correlated attributes using the findCorrelation() function from the caret package as follows:

```
# remove correlated attributes
# find attributes that are highly corrected
set.seed(7)
cutoff <- 0.70
correlations <- cor(dataset[,1:13])
highlyCorrelated <- findCorrelation(correlations, cutoff=cutoff)
for (value in highlyCorrelated) {
  print(names(dataset)[value])
}
```

```
## [1] "indus"
```

```
## [1] "nox"
## [1] "tax"
## [1] "dis"
```

```
# create a new dataset without highly corrected features
datasetFeatures <- dataset[,-highlyCorrelated]
dim(datasetFeatures)
```

```
## [1] 407  10
```

We can see that we have dropped 4 attributes: indus, box, tax and dis.

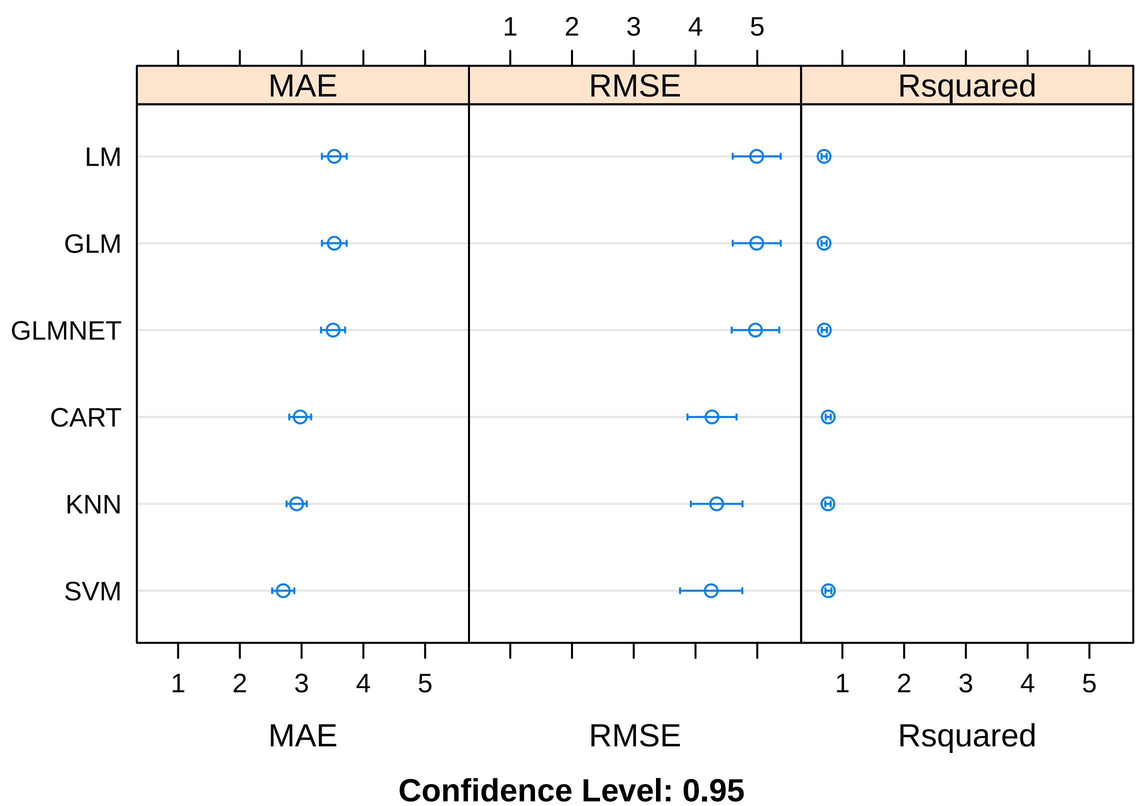
Now let's try the same 6 algorithms from our base line experiment.

```
# Run algorithms using 10-fold cross validation
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"
# lm
set.seed(7)
fit.lm <- train(medv~., data=datasetFeatures, method="lm", metric=metric, preProc=c("center", "scale"),
# GLM
set.seed(7)
fit.glm <- train(medv~., data=datasetFeatures, method="glm", metric=metric, preProc=c("center", "scale"
# GLMNET
set.seed(7)
fit.glmnet <- train(medv~., data=datasetFeatures, method="glmnet", metric=metric, preProc=c("center", "
# SVM
set.seed(7)
fit.svm <- train(medv~., data=datasetFeatures, method="svmRadial", metric=metric, preProc=c("center", "
# CART
set.seed(7)
grid <- expand.grid(.cp=c(0, 0.05, 0.1))
fit.cart <- train(medv~., data=datasetFeatures, method="rpart", metric=metric, tuneGrid=grid, preProc=c
# KNN
set.seed(7)
fit.knn <- train(medv~., data=datasetFeatures, method="knn", metric=metric, preProc=c("center", "scale"
# Compare algorithms
feature_results <- resamples(list(LM=fit.lm, GLM=fit.glm, GLMNET=fit.glmnet, SVM=fit.svm,
CART=fit.cart, KNN=fit.knn))
summary(feature_results)
```

```
##
## Call:
## summary.resamples(object = feature_results)
##
## Models: LM, GLM, GLMNET, SVM, CART, KNN
## Number of resamples: 30
##
## MAE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## LM     2.597582 3.198186 3.533505 3.529329 3.810776 4.776058    0
## GLM    2.597582 3.198186 3.533505 3.529329 3.810776 4.776058    0
## GLMNET 2.628837 3.117948 3.542265 3.509583 3.868589 4.690752    0
```

```
## SVM    1.869960 2.362733 2.734075 2.702728 2.963509 3.597276    0
## CART   2.081461 2.680047 2.875717 2.977352 3.128356 4.103067    0
## KNN    2.169500 2.603482 2.873902 2.919836 3.205521 3.720976    0
##
## RMSE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## LM     3.385745 4.172629 4.913293 4.991283 5.767162 7.146177    0
## GLM    3.385745 4.172629 4.913293 4.991283 5.767162 7.146177    0
## GLMNET 3.382741 4.117021 4.962157 4.971174 5.726988 7.173045    0
## SVM    2.506230 3.229388 4.119766 4.253523 4.983229 7.412917    0
## CART   2.734757 3.571064 3.893022 4.266904 4.762564 7.431916    0
## KNN    2.668598 3.530793 4.046128 4.343244 5.062585 7.029593    0
##
## Rsquared
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## LM     0.4215797 0.6389221 0.6803902 0.7039736 0.8000201 0.8996477    0
## GLM    0.4215797 0.6389221 0.6803902 0.7039736 0.8000201 0.8996477    0
## GLMNET 0.4167207 0.6546938 0.6911644 0.7077446 0.8071089 0.9051059    0
## SVM    0.4694850 0.7139120 0.8125167 0.7732194 0.8608522 0.9460838    0
## CART   0.4852551 0.7379702 0.7967859 0.7711166 0.8474429 0.9176368    0
## KNN    0.4346147 0.6931495 0.7991336 0.7664082 0.8450951 0.9301857    0
```

```
dotplot(feature_results)
```



Comparing the results, we can see that this has made the RMSE worse for the linear and the non-linear algorithms. The correlated attributes we removed are contributing to the accuracy of the models.

Evaluate Algorithms: Box-Cox Transform

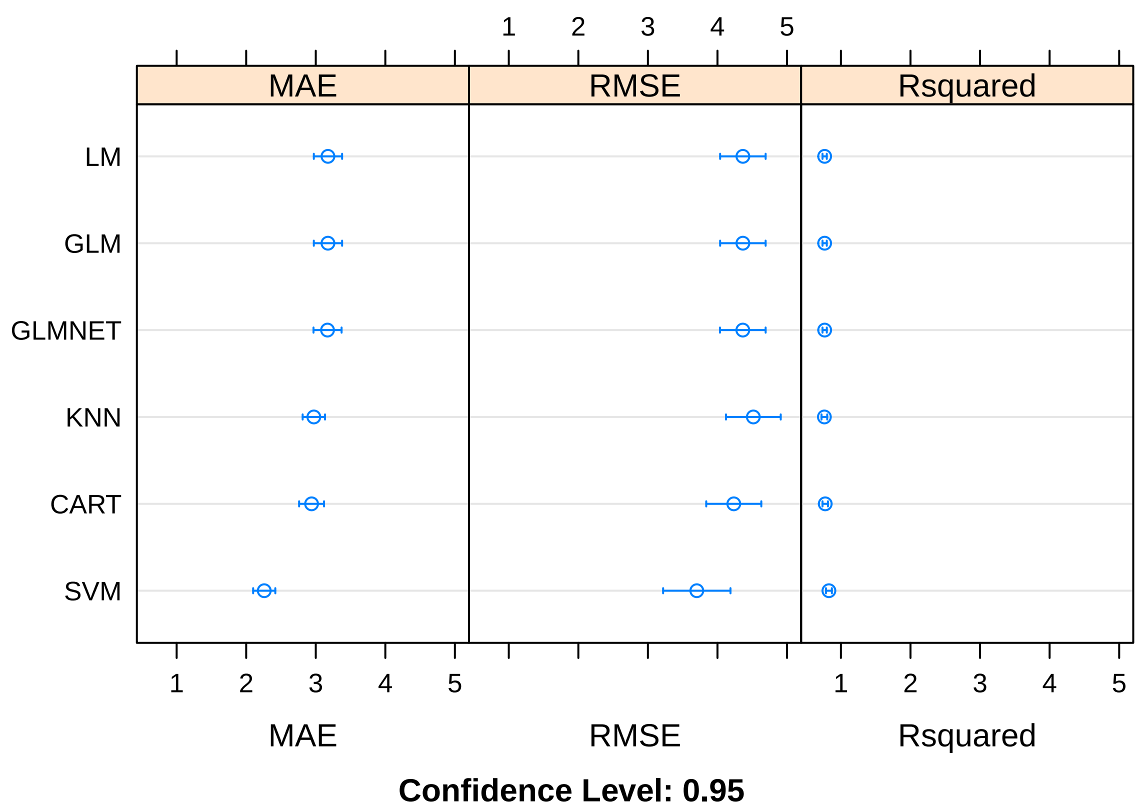
We know that some of the attributes have a skew and others perhaps have an exponential distribution. One option would be to explore squaring and log transforms respectively (you could try this!). Another approach would be to use a power transform and let it figure out the amount to correct each attribute. One example is the Box-Cox power transform. Let's try using this transform to rescale the original data and evaluate the effect on the same 6 algorithms. We will also leave in the centering and scaling for the benefit of the instance based method.

```r
# Run algorithms using 10-fold cross validation
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"
# lm
set.seed(7)
fit.lm <- train(medv~., data=dataset, method="lm", metric=metric, preProc=c("center",
"scale", "BoxCox"), trControl=trainControl)
# GLM
set.seed(7)
fit.glm <- train(medv~., data=dataset, method="glm", metric=metric, preProc=c("center",
"scale", "BoxCox"), trControl=trainControl)
# GLMNET
set.seed(7)
fit.glmnet <- train(medv~., data=dataset, method="glmnet", metric=metric,
preProc=c("center", "scale", "BoxCox"), trControl=trainControl)
# SVM
set.seed(7)
fit.svm <- train(medv~., data=dataset, method="svmRadial", metric=metric,
preProc=c("center", "scale", "BoxCox"), trControl=trainControl)
# CART
set.seed(7)
grid <- expand.grid(.cp=c(0, 0.05, 0.1))
fit.cart <- train(medv~., data=dataset, method="rpart", metric=metric, tuneGrid=grid,
preProc=c("center", "scale", "BoxCox"), trControl=trainControl)
# KNN
set.seed(7)
fit.knn <- train(medv~., data=dataset, method="knn", metric=metric, preProc=c("center",
"scale", "BoxCox"), trControl=trainControl)
# Compare algorithms
transformResults <- resamples(list(LM=fit.lm, GLM=fit.glm, GLMNET=fit.glmnet, SVM=fit.svm,
CART=fit.cart, KNN=fit.knn))
summary(transformResults)
```

```
##
## Call:
## summary.resamples(object = transformResults)
##
## Models: LM, GLM, GLMNET, SVM, CART, KNN
## Number of resamples: 30
##
## MAE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## LM     2.104233 2.797637 3.207088 3.175320 3.454257 4.435940    0
## GLM    2.104233 2.797637 3.207088 3.175320 3.454257 4.435940    0
## GLMNET 2.114563 2.798935 3.209521 3.169094 3.448723 4.432529    0
```

```
## SVM    1.295913 1.953108 2.248283 2.259257 2.477442 3.189012    0
## CART   2.216672 2.620729 2.891569 2.939787 3.106475 4.161930    0
## KNN    2.328184 2.664636 2.818222 2.972252 3.271885 3.955278    0
##
## RMSE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## LM     2.819687 3.811635 4.431132 4.365406 4.995457 6.163792    0
## GLM    2.819687 3.811635 4.431132 4.365406 4.995457 6.163792    0
## GLMNET 2.831297 3.791174 4.421466 4.364087 5.000829 6.184862    0
## SVM    1.804804 2.726898 3.414005 3.702891 4.235294 6.732293    0
## CART   2.766558 3.379400 3.997915 4.234486 4.834258 7.087656    0
## KNN    3.010540 3.725322 4.371531 4.515830 5.022143 7.297372    0
##
## Rsquared
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## LM     0.5598749 0.7120390 0.7753579 0.7661885 0.8254844 0.9096007    0
## GLM    0.5598749 0.7120390 0.7753579 0.7661885 0.8254844 0.9096007    0
## GLMNET 0.5564608 0.7134025 0.7748693 0.7663531 0.8264084 0.9097210    0
## SVM    0.5239104 0.7783632 0.8540991 0.8274173 0.9074699 0.9786169    0
## CART   0.5144771 0.7273216 0.8156225 0.7743938 0.8430400 0.8989506    0
## KNN    0.4922397 0.7231826 0.7919972 0.7622184 0.8419261 0.9365145    0
```

```
dotplot(transformResults)
```



We can see that this indeed decrease the RMSE and increased the R2 on all except the CART algorithms. The RMSE of SVM dropped to an average of 3.761.

Improve Results With Tuning

We can improve the accuracy of the well performing algorithms by tuning their parameters. In this section we will look at tuning the parameters of SVM with a Radial Basis Function (RBF). with more time it might be worth exploring tuning of the parameters for CART and KNN. It might also be worth exploring other kernels for SVM besides the RBF. Let's look at the default parameters already adopted.

```
print(fit.svm)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 407 samples
##  13 predictor
##
## Pre-processing: centered (13), scaled (13), Box-Cox transformation (11)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 365, 366, 366, 367, 366, 366, ...
## Resampling results across tuning parameters:
##
##   C     RMSE      Rsquared   MAE
##   0.25  4.540050  0.7717378  2.731568
##   0.50  4.073587  0.8016477  2.459456
##   1.00  3.702891  0.8274173  2.259257
##
## Tuning parameter 'sigma' was held constant at a value of 0.1160954
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.1160954 and C = 1.
```

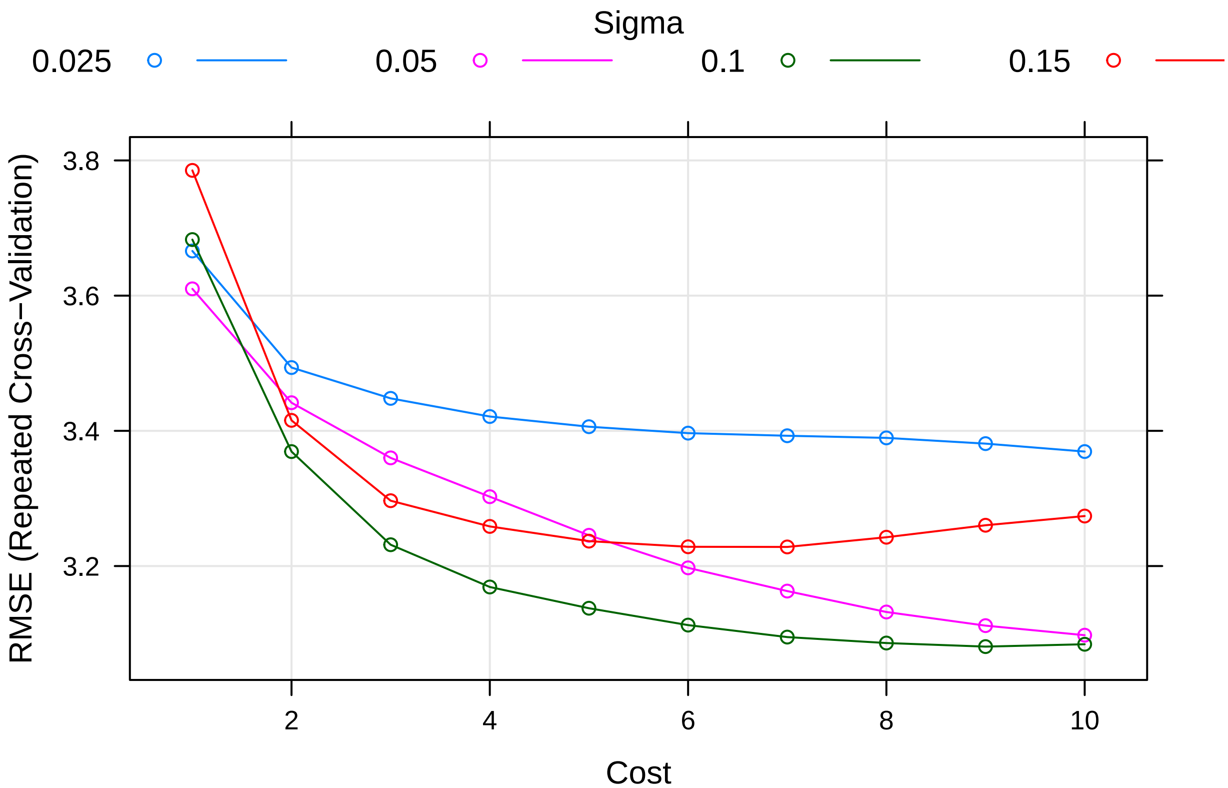
Let's design a grid search around a C value of 1. We might see a small trend of decreasing RMSE with increasing C, so lets try all integer C values between 1 and 10. Another parameter that caret lets us tune is the sigma parameter. This is a smoothing parameter. Good sigma values are often start around 0.1, so we will try numbers before and after.

```
# tune SVM sigma and C parametres
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"
set.seed(7)
grid <- expand.grid(.sigma=c(0.025, 0.05, 0.1, 0.15), .C=seq(1, 10, by=1))
fit.svm <- train(medv~., data=dataset, method="svmRadial", metric=metric, tuneGrid=grid,
preProc=c("BoxCox"), trControl=trainControl)
print(fit.svm)
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 407 samples
##  13 predictor
##
## Pre-processing: Box-Cox transformation (11)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 365, 366, 366, 367, 366, 366, ...
## Resampling results across tuning parameters:
##
##   sigma  C   RMSE      Rsquared   MAE
```

```
##   0.025   1  3.666043  0.8304258  2.341474
##   0.025   2  3.493599  0.8402446  2.210585
##   0.025   3  3.448014  0.8424544  2.166453
##   0.025   4  3.421181  0.8441227  2.144864
##   0.025   5  3.406115  0.8452055  2.126418
##   0.025   6  3.396604  0.8459398  2.118049
##   0.025   7  3.392738  0.8458445  2.112822
##   0.025   8  3.389565  0.8456293  2.109829
##   0.025   9  3.381083  0.8459197  2.107825
##   0.025  10  3.369348  0.8466370  2.103474
##   0.050   1  3.610049  0.8333400  2.253343
##   0.050   2  3.441596  0.8430992  2.172363
##   0.050   3  3.359987  0.8478358  2.111457
##   0.050   4  3.302593  0.8519590  2.081731
##   0.050   5  3.245491  0.8563049  2.054489
##   0.050   6  3.197339  0.8597979  2.032993
##   0.050   7  3.162954  0.8623870  2.024577
##   0.050   8  3.132037  0.8645406  2.016218
##   0.050   9  3.111871  0.8658531  2.009842
##   0.050  10  3.097720  0.8667507  2.006447
##   0.100   1  3.682927  0.8286381  2.256095
##   0.100   2  3.369310  0.8482288  2.120248
##   0.100   3  3.231542  0.8577532  2.059344
##   0.100   4  3.169000  0.8623355  2.043249
##   0.100   5  3.137603  0.8645132  2.038883
##   0.100   6  3.112648  0.8663063  2.037535
##   0.100   7  3.094928  0.8675313  2.038308
##   0.100   8  3.086138  0.8678926  2.039503
##   0.100   9  3.080821  0.8682460  2.040369
##   0.100  10  3.084304  0.8680754  2.046062
##   0.150   1  3.785264  0.8219252  2.301285
##   0.150   2  3.415495  0.8464293  2.141587
##   0.150   3  3.296656  0.8543875  2.090351
##   0.150   4  3.258605  0.8567077  2.088199
##   0.150   5  3.236878  0.8575705  2.092306
##   0.150   6  3.228490  0.8576464  2.102085
##   0.150   7  3.228245  0.8573952  2.115114
##   0.150   8  3.242641  0.8563199  2.133632
##   0.150   9  3.260325  0.8550399  2.153825
##   0.150  10  3.273954  0.8540617  2.170274
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.1 and C = 9.
```

```
plot(fit.svm)
```



We can see that the sigma values flatten out with larger C cost constraints. It looks like we might do well with a sigma of 0.1 and a C of 9. This gives us a respectable RMSE of 3.08. If we wanted to take this further, we could try even more fine tuning with more grid searches. We could also explore trying to tune other parameters of the underlying ksvm() function. Finally and as already mentioned, we could perform some grid searches on the other non-linear regression methods.

Ensemble Methods

We can try some ensemble methods on the problem and see if we can get a further decrease in our RMSE. In this section we will look at some boosting and bagging techniques for decision trees. Additional approaches you could look into would be blending the predictions of multiple well performing models together, called stacking. Let's take a look at the following ensemble methods:

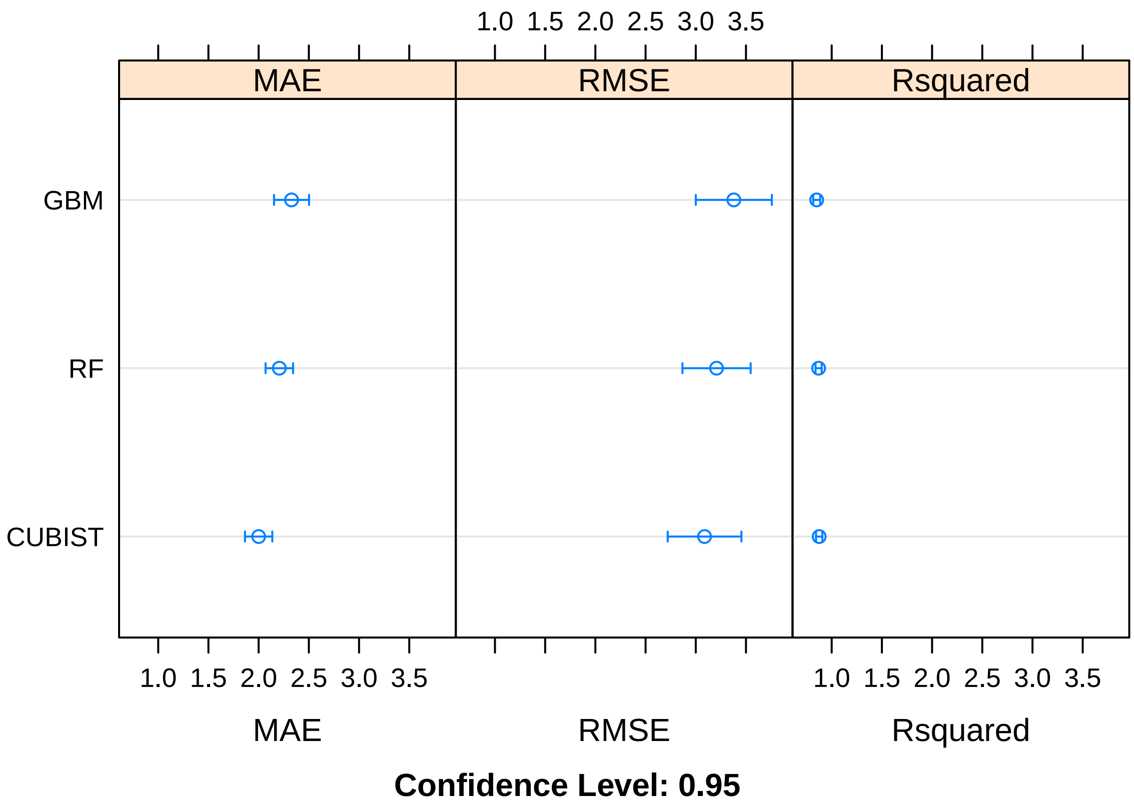
- Random Forest, bagging (RF).
- Gradient Boosting Machines boosting (GBM).
- Cubist, boosting (CUBIST).

```
# try ensembles
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"
# Random Forest
set.seed(7)
fit.rf <- train(medv~., data=dataset, method="rf", metric=metric, preProc=c("BoxCox"),
trControl=trainControl)
# Stochastic Gradient Boosting
```

```r
set.seed(7)
fit.gbm <- train(medv~., data=dataset, method="gbm", metric=metric, preProc=c("BoxCox"),
trControl=trainControl, verbose=FALSE)
# Cubist
set.seed(7)
fit.cubist <- train(medv~., data=dataset, method="cubist", metric=metric,
preProc=c("BoxCox"), trControl=trainControl)
# Compare algorithms
ensembleResults <- resamples(list(RF=fit.rf, GBM=fit.gbm, CUBIST=fit.cubist))
summary(ensembleResults)
```

```
##
## Call:
## summary.resamples(object = ensembleResults)
##
## Models: RF, GBM, CUBIST
## Number of resamples: 30
##
## MAE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## RF     1.636635 1.990929 2.197644 2.206004 2.339188 3.220658    0
## GBM    1.646824 1.979192 2.265152 2.327776 2.545692 3.752348    0
## CUBIST 1.311433 1.754070 1.952186 2.000247 2.173831 2.891565    0
##
## RMSE
##            Min.  1st Qu.   Median     Mean  3rd Qu.     Max. NA's
## RF     2.112991 2.606145 3.089201 3.207066 3.627072 6.521859    0
## GBM    1.917678 2.544770 3.310917 3.379109 3.665515 6.851109    0
## CUBIST 1.786686 2.379441 2.738151 3.087677 3.779192 5.788691    0
##
## Rsquared
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## RF     0.5972355 0.8496110 0.8993259 0.8698528 0.9182063 0.9722639    0
## GBM    0.5581867 0.8154935 0.8886156 0.8508565 0.9120289 0.9643755    0
## CUBIST 0.6805940 0.8176997 0.9092059 0.8754041 0.9318374 0.9699312    0
```

```r
dotplot(ensembleResults)
```



We can see that Cubist was the most accurate with an RMSE that was lower than that achieved by tuning SVM.

Let's dive deeper into Cubist and see if we can tune it further and get more skill out of it. Cubist has two parameters that are tunable with caret: committees which is the number of boosting operations and neighbors which is used during prediction and is the number of instances used to correct the rule based prediction (although the documentation is perhaps a little ambiguous on this). For more information about Cubist see the help on the function ?cubist. Let's first look at the default tuning parameter used by caret that resulted in our accurate model.

```
# look at parameters used for Cubist
print(fit.cubist)
```

```
## Cubist
##
## 407 samples
##  13 predictor
##
## Pre-processing: Box-Cox transformation (11)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 365, 366, 366, 367, 366, 366, ...
## Resampling results across tuning parameters:
##
##   committees  neighbors  RMSE      Rsquared   MAE
##    1          0          3.935283  0.8050980  2.501518
##    1          5          3.663278  0.8278677  2.239888
##    1          9          3.685088  0.8254836  2.257340
```

```
##   10          0          3.449718  0.8480935  2.288926
##   10          5          3.191849  0.8675510  2.044831
##   10          9          3.229558  0.8643281  2.074724
##   20          0          3.339729  0.8576020  2.247981
##   20          5          3.087677  0.8754041  2.000247
##   20          9          3.122622  0.8723800  2.031336
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were committees = 20 and neighbors = 5.
```

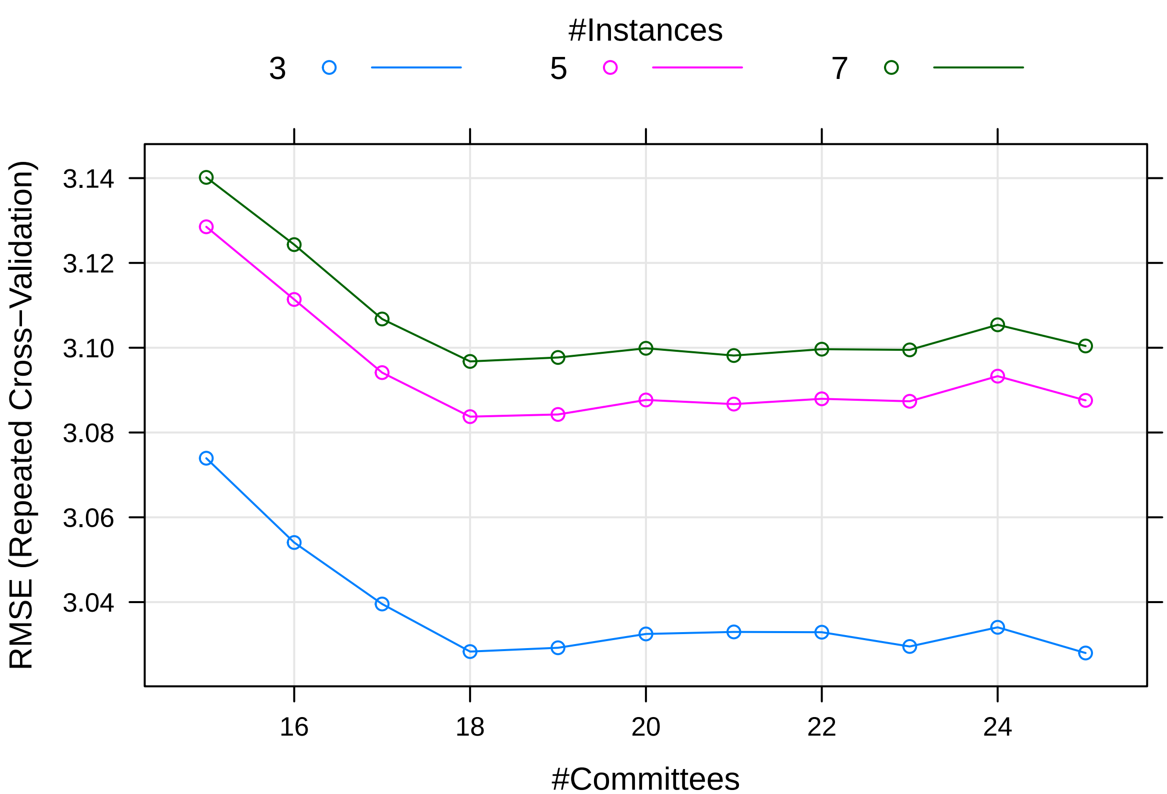
Let's use a grid search to tune around those values. We'll try all committees between 15 and 25 and spot-check a neighbors value above and below 5.

```
# Tune the Cubist algorithm
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3)
metric <- "RMSE"
set.seed(7)
grid <- expand.grid(.committees=seq(15, 25, by=1), .neighbors=c(3, 5, 7))
tune.cubist <- train(medv~., data=dataset, method="cubist", metric=metric,
preProc=c("BoxCox"), tuneGrid=grid, trControl=trainControl)
print(tune.cubist)
```

```
## Cubist
##
## 407 samples
##  13 predictor
##
## Pre-processing: Box-Cox transformation (11)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 365, 366, 366, 367, 366, 366, ...
## Resampling results across tuning parameters:
##
##   committees  neighbors  RMSE      Rsquared   MAE
##   15          3          3.073937  0.8767747  1.995981
##   15          5          3.128523  0.8727199  2.016761
##   15          7          3.140179  0.8711152  2.029532
##   16          3          3.054057  0.8778157  1.988389
##   16          5          3.111381  0.8736537  2.010829
##   16          7          3.124320  0.8719441  2.024607
##   17          3          3.039552  0.8788573  1.979358
##   17          5          3.094127  0.8748690  2.000618
##   17          7          3.106775  0.8731719  2.013405
##   18          3          3.028333  0.8797190  1.973590
##   18          5          3.083740  0.8757237  1.997690
##   18          7          3.096770  0.8740027  2.012345
##   19          3          3.029223  0.8797359  1.971444
##   19          5          3.084257  0.8757493  1.994837
##   19          7          3.097699  0.8740371  2.008627
##   20          3          3.032489  0.8793502  1.977003
##   20          5          3.087677  0.8754041  2.000247
##   20          7          3.099866  0.8737809  2.014252
##   21          3          3.032965  0.8794883  1.978889
##   21          5          3.086705  0.8755892  2.002846
```

```
##   21          7          3.098157  0.8740286  2.016069
##   22          3          3.032896  0.8793041  1.980806
##   22          5          3.087947  0.8754550  2.003035
##   22          7          3.099654  0.8739021  2.016758
##   23          3          3.029535  0.8797109  1.980914
##   23          5          3.087364  0.8756108  2.006294
##   23          7          3.099492  0.8740565  2.020778
##   24          3          3.034052  0.8792030  1.982144
##   24          5          3.093291  0.8750003  2.006240
##   24          7          3.105403  0.8734220  2.021364
##   25          3          3.027991  0.8800070  1.982355
##   25          5          3.087580  0.8757971  2.006807
##   25          7          3.100416  0.8741655  2.021189
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were committees = 25 and neighbors = 3.
```

```
plot(tune.cubist)
```



We can see that we have achieved a more accurate model again with an RMSE of 3.0875 using committees = 25 and neighbors = 3.

Finalize Model

It looks like that cubist results in our most accurate model. Let's finalize it by creating a new standalone Cubist model with the parameters above trained using the whole dataset. We must also use the Box-Cox power transform.

```
library(Cubist)
# prepare the data transform using training data
set.seed(7)
x <- dataset[,1:13]
y <- dataset[,14]
preprocessParams <- preProcess(x, method=c("BoxCox"))
transX <- predict(preprocessParams, x)
# train the final model
finalModel <- cubist(x=transX, y=y, committees=18)
summary(finalModel)
```

```
##
## Call:
## cubist.default(x = transX, y = y, committees = 18)
##
##
## Cubist [Release 2.07 GPL Edition]  Sun Nov 13 22:38:26 2022
## ---------------------------------
##
##     Target attribute `outcome'
##
## Read 407 cases (14 attributes) from undefined.data
##
## Model 1:
##
##   Rule 1/1: [84 cases, mean 14.29, range 5 to 27.5, est err 1.97]
##
##     if
##  nox > -0.4864544
##     then
##  outcome = 35.08 - 2.45 crim - 4.31 lstat + 2.1e-05 b
##
##   Rule 1/2: [163 cases, mean 19.37, range 7 to 31, est err 2.10]
##
##     if
##  nox <= -0.4864544
##  lstat > 2.848535
##     then
##  outcome = 186.8 - 2.34 lstat - 3.3 dis - 88 tax + 2 rad + 4.4 rm
##            - 0.033 ptratio - 0.0116 age + 3.3e-05 b
##
##   Rule 1/3: [24 cases, mean 21.65, range 18.2 to 25.3, est err 1.19]
##
##     if
##  rm <= 3.326479
##  dis > 1.345056
##  lstat <= 2.848535
##     then
##  outcome = 43.83 + 14.5 rm - 2.29 lstat - 3.8 dis - 30 tax
##            - 0.014 ptratio - 1.4 nox + 0.017 zn + 0.4 rad + 0.15 crim
##            - 0.0025 age + 8e-06 b
##
```

```
##   Rule 1/4: [7 cases, mean 27.66, range 20.7 to 50, est err 7.89]
##
##     if
##  rm > 3.326479
##  ptratio > 193.545
##  lstat <= 2.848535
##     then
##  outcome = 19.64 + 7.8 rm - 3.4 dis - 1.62 lstat + 0.27 crim - 0.006 age
##            + 0.023 zn - 7 tax - 0.003 ptratio
##
##   Rule 1/5: [141 cases, mean 30.60, range 15 to 50, est err 2.09]
##
##     if
##  rm > 3.326479
##  ptratio <= 193.545
##     then
##  outcome = 137.95 + 21.7 rm - 3.43 lstat - 4.9 dis - 87 tax - 0.0162 age
##            - 0.039 ptratio + 0.06 crim + 0.005 zn
##
##   Rule 1/6: [8 cases, mean 32.16, range 22.1 to 50, est err 8.67]
##
##     if
##  rm <= 3.326479
##  dis <= 1.345056
##  lstat <= 2.848535
##     then
##  outcome = -19.71 + 18.58 lstat - 15.9 dis + 5.6 rm
##
## Model 2:
##
##   Rule 2/1: [23 cases, mean 10.57, range 5 to 15, est err 3.06]
##
##     if
##  crim > 2.086391
##  dis <= 0.6604174
##  b > 67032.41
##     then
##  outcome = 37.22 - 4.83 crim - 7 dis - 1.9 lstat - 1.9e-05 b - 0.7 rm
##
##   Rule 2/2: [70 cases, mean 14.82, range 5 to 50, est err 3.90]
##
##     if
##  rm <= 3.620525
##  dis <= 0.6604174
##     then
##  outcome = 74.6 - 21 dis - 5.09 lstat - 15 tax - 0.0017 age + 6e-06 b
##
##   Rule 2/3: [18 cases, mean 18.03, range 7.5 to 50, est err 6.81]
##
##     if
##  crim > 2.086391
##  dis <= 0.6604174
##  b <= 67032.41
##     then
```

```
##  outcome = 94.95 - 40.1 dis - 8.15 crim - 7.14 lstat - 3.5e-05 b - 1.3 rm
##
##   Rule 2/4: [258 cases, mean 20.74, range 9.5 to 36.2, est err 1.92]
##
##     if
##  rm <= 3.620525
##  dis > 0.6604174
##  lstat > 1.805082
##     then
##  outcome = 61.89 - 2.56 lstat + 5.5 rm - 2.8 dis + 7.3e-05 b - 0.0132 age
##            - 26 tax - 0.11 indus - 0.004 ptratio + 0.05 crim
##
##   Rule 2/5: [37 cases, mean 31.66, range 10.4 to 50, est err 3.70]
##
##     if
##  rm > 3.620525
##  lstat > 1.805082
##     then
##  outcome = 370.03 - 180 tax - 2.19 lstat - 1.7 dis + 2.6 rm
##            - 0.016 ptratio - 0.25 indus + 0.12 crim - 0.0021 age
##            + 9e-06 b - 0.5 nox
##
##   Rule 2/6: [42 cases, mean 38.23, range 22.8 to 50, est err 3.70]
##
##     if
##  lstat <= 1.805082
##     then
##  outcome = -73.87 + 32.4 rm - 9.4e-05 b - 1.8 dis + 0.028 zn
##            - 0.013 ptratio
##
##   Rule 2/7: [4 cases, mean 40.20, range 37.6 to 42.8, est err 7.33]
##
##     if
##  rm > 4.151791
##  dis > 1.114486
##     then
##  outcome = 35.8
##
##   Rule 2/8: [8 cases, mean 47.45, range 41.3 to 50, est err 10.01]
##
##     if
##  dis <= 1.114486
##  lstat <= 1.805082
##     then
##  outcome = 48.96 + 7.53 crim - 4.1e-05 b - 0.8 dis + 1.2 rm + 0.008 zn
##
## Model 3:
##
##   Rule 3/1: [81 cases, mean 13.93, range 5 to 23.2, est err 2.24]
##
##     if
##  nox > -0.4864544
##  lstat > 2.848535
##     then
```

```
##  outcome = 55.03 - 0.0631 age - 2.11 crim + 12 nox - 4.16 lstat
##            + 3.2e-05 b
##
##   Rule 3/2: [163 cases, mean 19.37, range 7 to 31, est err 2.29]
##
##     if
##  nox <= -0.4864544
##  lstat > 2.848535
##     then
##  outcome = 77.73 - 0.059 ptratio + 5.8 rm - 3.2 dis - 0.0139 age
##            - 1.15 lstat - 30 tax - 1.1 nox + 0.4 rad
##
##   Rule 3/3: [62 cases, mean 24.01, range 18.2 to 50, est err 3.56]
##
##     if
##  rm <= 3.448196
##  lstat <= 2.848535
##     then
##  outcome = 94.86 + 18.2 rm + 0.63 crim - 68 tax - 2.3 dis - 3 nox
##            - 0.0098 age - 0.41 indus - 0.011 ptratio
##
##   Rule 3/4: [143 cases, mean 28.76, range 16.5 to 50, est err 2.53]
##
##     if
##  dis > 0.9547035
##  lstat <= 2.848535
##     then
##  outcome = 269.46 + 17.9 rm - 6.1 dis - 153 tax + 0.96 crim - 0.0217 age
##            - 5.5 nox - 0.62 indus - 0.028 ptratio - 0.89 lstat + 0.4 rad
##            + 0.004 zn
##
##   Rule 3/5: [10 cases, mean 35.13, range 21.9 to 50, est err 9.31]
##
##     if
##  dis <= 0.6492998
##  lstat <= 2.848535
##     then
##  outcome = 58.69 - 56.8 dis - 8.4 nox
##
##   Rule 3/6: [10 cases, mean 41.67, range 22 to 50, est err 9.89]
##
##     if
##  dis > 0.6492998
##  dis <= 0.9547035
##  lstat <= 2.848535
##     then
##  outcome = 47.93
##
## Model 4:
##
##   Rule 4/1: [69 cases, mean 12.69, range 5 to 27.5, est err 2.55]
##
##     if
##  dis <= 0.719156
```

```
## 	lstat > 3.508535
##     then
## 	outcome = 180.13 - 7.2 dis + 0.039 age - 3.78 lstat - 83 tax
##
##   Rule 4/2: [164 cases, mean 19.42, range 12 to 31, est err 1.96]
##
##     if
## 	dis > 0.719156
## 	lstat > 2.848535
##     then
## 	outcome = 52.75 + 7.1 rm - 2.05 lstat - 3.6 dis + 8.2e-05 b - 0.0152 age
## 	          - 25 tax + 0.5 rad - 1.2 nox - 0.008 ptratio
##
##   Rule 4/3: [11 cases, mean 20.39, range 15 to 27.9, est err 3.51]
##
##     if
## 	dis <= 0.719156
## 	lstat > 2.848535
## 	lstat <= 3.508535
##     then
## 	outcome = 21.69
##
##   Rule 4/4: [63 cases, mean 23.22, range 16.5 to 31.5, est err 1.67]
##
##     if
## 	rm <= 3.483629
## 	dis > 0.9731624
## 	lstat <= 2.848535
##     then
## 	outcome = 59.35 - 3.96 lstat - 3.1 dis + 1 rm - 14 tax + 0.3 rad
## 	          - 0.7 nox - 0.005 ptratio + 6e-06 b
##
##   Rule 4/5: [8 cases, mean 33.08, range 22 to 50, est err 23.91]
##
##     if
## 	rm > 3.369183
## 	dis <= 0.9731624
## 	lstat > 2.254579
## 	lstat <= 2.848535
##     then
## 	outcome = -322.28 + 64.9 lstat + 56.8 rm - 30.2 dis
##
##   Rule 4/6: [7 cases, mean 33.87, range 22.1 to 50, est err 13.21]
##
##     if
## 	rm <= 3.369183
## 	dis <= 0.9731624
## 	lstat <= 2.848535
##     then
## 	outcome = -52.11 + 43.45 lstat - 30.8 dis
##
##   Rule 4/7: [91 cases, mean 34.43, range 21.9 to 50, est err 3.32]
##
##     if
```

```
##  rm > 3.483629
##  lstat <= 2.848535
##     then
##  outcome = -33.09 + 22 rm - 5.02 lstat - 0.038 ptratio - 0.9 dis
##            + 0.005 zn
##
##   Rule 4/8: [22 cases, mean 36.99, range 21.9 to 50, est err 13.21]
##
##     if
##  dis <= 0.9731624
##  lstat <= 2.848535
##     then
##  outcome = 80.3 - 17.43 lstat - 0.134 ptratio + 2.5 rm - 1.2 dis
##            + 0.008 zn
##
## Model 5:
##
##   Rule 5/1: [84 cases, mean 14.29, range 5 to 27.5, est err 2.81]
##
##     if
##  nox > -0.4864544
##     then
##  outcome = 56.48 + 28.5 nox - 0.0875 age - 3.58 crim - 5.9 dis
##            - 2.96 lstat + 0.073 ptratio + 1.7e-05 b
##
##   Rule 5/2: [163 cases, mean 19.37, range 7 to 31, est err 2.38]
##
##     if
##  nox <= -0.4864544
##  lstat > 2.848535
##     then
##  outcome = 61.59 - 0.064 ptratio + 5.9 rm - 3.1 dis - 0.0142 age
##            - 0.77 lstat - 21 tax
##
##   Rule 5/3: [163 cases, mean 29.94, range 16.5 to 50, est err 3.65]
##
##     if
##  lstat <= 2.848535
##     then
##  outcome = 264.17 + 21.9 rm - 8 dis - 155 tax - 0.0317 age
##            - 0.032 ptratio + 0.29 crim - 1.6 nox - 0.25 indus
##
##   Rule 5/4: [10 cases, mean 35.13, range 21.9 to 50, est err 11.79]
##
##     if
##  dis <= 0.6492998
##  lstat <= 2.848535
##     then
##  outcome = 68.19 - 73.4 dis + 1.1 rm + 0.11 crim - 0.6 nox - 0.1 indus
##            - 0.0017 age - 0.12 lstat
##
## Model 6:
##
##   Rule 6/1: [71 cases, mean 15.57, range 5 to 50, est err 4.42]
```

```
##
##     if
##  dis <= 0.6443245
##  lstat > 1.793385
##     then
##  outcome = 45.7 - 20.6 dis - 5.38 lstat
##
##   Rule 6/2: [159 cases, mean 19.53, range 8.3 to 36.2, est err 2.08]
##
##     if
##  rm <= 3.329365
##  dis > 0.6443245
##     then
##  outcome = 24.33 + 8.8 rm + 0.000118 b - 0.0146 age - 2.5 dis
##            - 0.95 lstat + 0.37 crim - 0.32 indus + 0.02 zn - 16 tax
##            + 0.2 rad - 0.5 nox - 0.004 ptratio
##
##   Rule 6/3: [175 cases, mean 27.80, range 9.5 to 50, est err 2.95]
##
##     if
##  rm > 3.329365
##  dis > 0.6443245
##     then
##  outcome = 0.11 + 18.7 rm - 3.11 lstat + 8.1e-05 b - 1.1 dis + 0.19 crim
##            - 20 tax - 0.19 indus + 0.3 rad - 0.7 nox - 0.005 ptratio
##            + 0.006 zn
##
##   Rule 6/4: [8 cases, mean 32.50, range 21.9 to 50, est err 10.34]
##
##     if
##  dis <= 0.6443245
##  lstat > 1.793385
##  lstat <= 2.894121
##     then
##  outcome = 69.38 - 71.2 dis - 0.14 lstat
##
##   Rule 6/5: [34 cases, mean 37.55, range 22.8 to 50, est err 3.55]
##
##     if
##  rm <= 4.151791
##  lstat <= 1.793385
##     then
##  outcome = -125.14 + 41.7 rm + 4.3 rad + 1.48 indus - 0.014 ptratio
##
##   Rule 6/6: [7 cases, mean 43.66, range 37.6 to 50, est err 3.12]
##
##     if
##  rm > 4.151791
##  lstat <= 1.793385
##     then
##  outcome = -137.67 + 44.6 rm - 0.064 ptratio
##
## Model 7:
##
```

```
##   Rule 7/1: [84 cases, mean 14.29, range 5 to 27.5, est err 2.91]
##
##     if
##  nox > -0.4864544
##     then
##  outcome = 46.85 - 3.45 crim - 0.0621 age + 14.2 nox + 4.4 dis
##           - 2.01 lstat + 2.5e-05 b
##
##   Rule 7/2: [323 cases, mean 24.66, range 7 to 50, est err 3.68]
##
##     if
##  nox <= -0.4864544
##     then
##  outcome = 57.59 - 0.065 ptratio - 4.4 dis + 6.8 rm - 0.0143 age
##           - 1.36 lstat - 19 tax - 0.8 nox - 0.12 crim + 0.09 indus
##
##   Rule 7/3: [132 cases, mean 28.24, range 16.5 to 50, est err 2.55]
##
##     if
##  dis > 1.063503
##  lstat <= 2.848535
##     then
##  outcome = 270.92 + 24.5 rm - 0.0418 age - 165 tax - 5.7 dis
##           - 0.028 ptratio + 0.26 crim + 0.017 zn
##
##   Rule 7/4: [7 cases, mean 36.01, range 23.3 to 50, est err 3.87]
##
##     if
##  dis <= 0.6002641
##  lstat <= 2.848535
##     then
##  outcome = 57.18 - 69.5 dis - 6.5 nox + 1.9 rm - 0.015 ptratio
##
##   Rule 7/5: [24 cases, mean 37.55, range 21.9 to 50, est err 8.66]
##
##     if
##  dis > 0.6002641
##  dis <= 1.063503
##  lstat <= 2.848535
##     then
##  outcome = -3.76 - 14.8 dis - 2.93 crim - 0.16 ptratio + 17.5 rm - 15 nox
##
## Model 8:
##
##   Rule 8/1: [80 cases, mean 13.75, range 5 to 27.9, est err 3.51]
##
##     if
##  dis <= 0.719156
##  lstat > 2.848535
##     then
##  outcome = 123.46 - 11.3 dis - 5.06 lstat - 45 tax + 0.9 rad + 1.7e-05 b
##
##   Rule 8/2: [164 cases, mean 19.42, range 12 to 31, est err 2.05]
##
```

```
##     if
##  dis > 0.719156
##  lstat > 2.848535
##     then
##  outcome = 227.11 - 120 tax + 6.4 rm + 9.3e-05 b - 3.3 dis + 2 rad
##            - 0.0183 age - 0.93 lstat + 0.05 crim - 0.3 nox
##
##   Rule 8/3: [163 cases, mean 29.94, range 16.5 to 50, est err 3.54]
##
##     if
##  lstat <= 2.848535
##     then
##  outcome = 158.14 - 5.73 lstat + 10.8 rm - 4 dis - 83 tax - 4.1 nox
##            + 0.61 crim - 0.54 indus + 1 rad + 3.6e-05 b
##
##   Rule 8/4: [7 cases, mean 36.01, range 23.3 to 50, est err 11.44]
##
##     if
##  dis <= 0.6002641
##  lstat <= 2.848535
##     then
##  outcome = 72.89 - 87.2 dis + 0.6 rm - 0.13 lstat
##
##   Rule 8/5: [47 cases, mean 38.44, range 15 to 50, est err 5.71]
##
##     if
##  rm > 3.726352
##     then
##  outcome = 602.95 - 10.4 lstat + 21 rm - 326 tax - 0.093 ptratio
##
## Model 9:
##
##   Rule 9/1: [81 cases, mean 13.93, range 5 to 23.2, est err 2.91]
##
##     if
##  nox > -0.4864544
##  lstat > 2.848535
##     then
##  outcome = 41.11 - 3.98 crim - 4.42 lstat + 6.7 nox
##
##   Rule 9/2: [163 cases, mean 19.37, range 7 to 31, est err 2.49]
##
##     if
##  nox <= -0.4864544
##  lstat > 2.848535
##     then
##  outcome = 44.98 - 0.068 ptratio - 4.4 dis + 6.6 rm - 1.25 lstat
##            - 0.0118 age - 0.9 nox - 12 tax - 0.08 crim + 0.06 indus
##
##   Rule 9/3: [132 cases, mean 28.24, range 16.5 to 50, est err 2.35]
##
##     if
##  dis > 1.063503
##  lstat <= 2.848535
```

```
##     then
##  outcome = 157.67 + 22.2 rm - 0.0383 age - 104 tax - 0.033 ptratio
##            - 2.2 dis
##
##   Rule 9/4: [7 cases, mean 30.76, range 21.9 to 50, est err 6.77]
##
##     if
##  dis <= 1.063503
##  b <= 66469.73
##  lstat <= 2.848535
##     then
##  outcome = 48.52 - 56.1 dis - 12.9 nox - 0.032 ptratio + 2.7 rm
##
##   Rule 9/5: [24 cases, mean 39.09, range 22 to 50, est err 6.20]
##
##     if
##  dis <= 1.063503
##  b > 66469.73
##  lstat <= 2.848535
##     then
##  outcome = -5.49 - 34.8 dis - 20.7 nox + 18.2 rm - 0.051 ptratio
##
## Model 10:
##
##   Rule 10/1: [327 cases, mean 19.45, range 5 to 50, est err 2.77]
##
##     if
##  rm <= 3.617282
##  lstat > 1.805082
##     then
##  outcome = 270.78 - 4.09 lstat - 131 tax + 2.9 rad + 5.3e-05 b - 0.6 dis
##            - 0.16 indus + 0.7 rm - 0.3 nox
##
##   Rule 10/2: [38 cases, mean 31.57, range 10.4 to 50, est err 4.71]
##
##     if
##  rm > 3.617282
##  lstat > 1.805082
##     then
##  outcome = 308.44 - 150 tax - 2.63 lstat + 1.6 rad - 1.9 dis - 0.49 indus
##            + 2.5 rm + 3e-05 b - 1.2 nox + 0.14 crim - 0.005 ptratio
##
##   Rule 10/3: [35 cases, mean 37.15, range 22.8 to 50, est err 2.76]
##
##     if
##  rm <= 4.151791
##  lstat <= 1.805082
##     then
##  outcome = -71.65 + 33.4 rm - 0.017 ptratio - 0.34 lstat + 0.2 rad
##            - 0.3 dis - 7 tax - 0.4 nox
##
##   Rule 10/4: [10 cases, mean 42.63, range 21.9 to 50, est err 7.11]
##
##     if
```

```
##  rm > 4.151791
##     then
##  outcome = -92.51 + 32.8 rm - 0.03 ptratio
##
## Model 11:
##
##   Rule 11/1: [84 cases, mean 14.29, range 5 to 27.5, est err 4.13]
##
##     if
##  nox > -0.4864544
##     then
##  outcome = 42.75 - 4.12 crim + 18.1 nox - 0.045 age + 6.8 dis
##            - 1.86 lstat
##
##   Rule 11/2: [244 cases, mean 17.56, range 5 to 31, est err 4.29]
##
##     if
##  lstat > 2.848535
##     then
##  outcome = 34.83 - 5.2 dis - 0.058 ptratio - 0.0228 age + 5.8 rm
##            - 0.56 lstat - 0.07 crim - 0.4 nox - 5 tax
##
##   Rule 11/3: [163 cases, mean 29.94, range 16.5 to 50, est err 3.49]
##
##     if
##  lstat <= 2.848535
##     then
##  outcome = 151.5 + 23.3 rm - 5.5 dis + 1.01 crim - 0.0211 age
##            - 0.052 ptratio - 98 tax + 0.031 zn
##
##   Rule 11/4: [10 cases, mean 35.13, range 21.9 to 50, est err 25.19]
##
##     if
##  dis <= 0.6492998
##  lstat <= 2.848535
##     then
##  outcome = 130.87 - 157.1 dis - 15.76 crim
##
## Model 12:
##
##   Rule 12/1: [80 cases, mean 13.75, range 5 to 27.9, est err 4.76]
##
##     if
##  dis <= 0.719156
##  lstat > 2.894121
##     then
##  outcome = 182.68 - 6.03 lstat - 7.6 dis - 76 tax + 1.3 rad - 0.52 indus
##            + 2.6e-05 b
##
##   Rule 12/2: [300 cases, mean 19.10, range 5 to 50, est err 2.76]
##
##     if
##  rm <= 3.50716
##  lstat > 1.793385
```

```
##     then
##  outcome = 83.61 - 3 lstat + 9.6e-05 b - 0.0072 age - 33 tax + 0.7 rad
##            + 0.32 indus
##
##   Rule 12/3: [10 cases, mean 24.25, range 15.7 to 36.2, est err 13.88]
##
##     if
##  rm <= 3.50716
##  tax <= 1.865769
##     then
##  outcome = 35.46
##
##   Rule 12/4: [10 cases, mean 32.66, range 21.9 to 50, est err 6.28]
##
##     if
##  dis <= 0.719156
##  lstat > 1.793385
##  lstat <= 2.894121
##     then
##  outcome = 82.78 - 69.5 dis - 3.66 indus
##
##   Rule 12/5: [89 cases, mean 32.75, range 13.4 to 50, est err 3.39]
##
##     if
##  rm > 3.50716
##  dis > 0.719156
##     then
##  outcome = 313.22 + 13.7 rm - 174 tax - 3.06 lstat + 4.8e-05 b - 1.5 dis
##            - 0.41 indus + 0.7 rad - 0.0055 age + 0.22 crim
##
##   Rule 12/6: [34 cases, mean 37.55, range 22.8 to 50, est err 3.25]
##
##     if
##  rm <= 4.151791
##  lstat <= 1.793385
##     then
##  outcome = -86.8 + 36 rm - 0.3 lstat - 5 tax
##
##   Rule 12/7: [7 cases, mean 43.66, range 37.6 to 50, est err 5.79]
##
##     if
##  rm > 4.151791
##  lstat <= 1.793385
##     then
##  outcome = -158.68 + 47.4 rm - 0.02 ptratio
##
## Model 13:
##
##   Rule 13/1: [84 cases, mean 14.29, range 5 to 27.5, est err 2.87]
##
##     if
##  nox > -0.4864544
##     then
##  outcome = 54.69 - 3.79 crim - 0.0644 age + 11.4 nox - 2.53 lstat
```

```
##
##   Rule 13/2: [8 cases, mean 17.76, range 7 to 27.9, est err 13.69]
##
##     if
##   nox <= -0.4864544
##   age > 296.3423
##   b <= 60875.57
##     then
##   outcome = -899.55 + 3.0551 age
##
##   Rule 13/3: [31 cases, mean 17.94, range 7 to 27.9, est err 5.15]
##
##     if
##   nox <= -0.4864544
##   b <= 60875.57
##   lstat > 2.848535
##     then
##   outcome = 44.43 - 3.51 lstat - 0.054 ptratio - 1.4 dis - 0.26 crim
##             - 0.0042 age - 0.21 indus + 0.9 rm
##
##   Rule 13/4: [163 cases, mean 19.37, range 7 to 31, est err 3.37]
##
##     if
##   nox <= -0.4864544
##   lstat > 2.848535
##     then
##   outcome = -5.76 + 0.000242 b + 8.9 rm - 5.2 dis - 0.0209 age
##             - 0.042 ptratio - 0.63 indus
##
##   Rule 13/5: [163 cases, mean 29.94, range 16.5 to 50, est err 3.45]
##
##     if
##   lstat <= 2.848535
##     then
##   outcome = 178.84 + 23.8 rm - 0.0343 age - 4.5 dis - 114 tax + 0.88 crim
##             - 0.048 ptratio + 0.026 zn
##
##   Rule 13/6: [7 cases, mean 36.01, range 23.3 to 50, est err 14.09]
##
##     if
##   dis <= 0.6002641
##   lstat <= 2.848535
##     then
##   outcome = 45.82 - 70.3 dis - 9.9 nox + 5.1 rm + 1.5 rad
##
##   Rule 13/7: [31 cases, mean 37.21, range 21.9 to 50, est err 7.73]
##
##     if
##   dis <= 1.063503
##   lstat <= 2.848535
##     then
##   outcome = 95.05 - 4.52 lstat - 7.5 dis + 8.8 rm - 0.064 ptratio
##             - 6.2 nox - 36 tax
##
```

```
## Model 14:
##
##   Rule 14/1: [49 cases, mean 16.06, range 8.4 to 22.7, est err 3.17]
##
##     if
##  nox > -0.4205732
##  lstat > 2.848535
##     then
##  outcome = 12.83 + 42.3 nox - 4.77 lstat + 9.7 rm + 7.8e-05 b
##
##   Rule 14/2: [78 cases, mean 16.36, range 5 to 50, est err 5.17]
##
##     if
##  dis <= 0.6604174
##     then
##  outcome = 110.6 - 10.4 dis - 4.85 lstat + 0.0446 age - 46 tax + 0.8 rad
##
##   Rule 14/3: [57 cases, mean 18.40, range 9.5 to 31, est err 2.43]
##
##     if
##  nox > -0.9365134
##  nox <= -0.4205732
##  age > 245.2507
##  dis > 0.6604174
##  lstat > 2.848535
##     then
##  outcome = 206.69 - 0.1012 age - 7.05 lstat + 12.2 nox - 67 tax + 0.3 rad
##            + 0.5 rm - 0.3 dis
##
##   Rule 14/4: [230 cases, mean 20.19, range 9.5 to 36.2, est err 2.09]
##
##     if
##  rm <= 3.483629
##  dis > 0.6492998
##     then
##  outcome = 119.15 - 2.61 lstat + 5.2 rm - 57 tax - 1.8 dis - 2.4 nox
##            + 0.7 rad + 0.24 crim + 0.003 age - 0.007 ptratio + 9e-06 b
##
##   Rule 14/5: [48 cases, mean 20.28, range 10.2 to 24.5, est err 2.13]
##
##     if
##  nox > -0.9365134
##  nox <= -0.4205732
##  age <= 245.2507
##  dis > 0.6604174
##  lstat > 2.848535
##     then
##  outcome = 19.4 - 1.91 lstat + 1.02 indus - 0.013 age + 2.7 rm + 2.6 nox
##            - 0.009 ptratio
##
##   Rule 14/6: [44 cases, mean 20.69, range 14.4 to 29.6, est err 2.26]
##
##     if
##  nox <= -0.9365134
```

```
##  lstat > 2.848535
##     then
##  outcome = 87.55 - 0.000315 b - 6.5 dis + 2.6 rad - 0.59 lstat - 18 tax
##
##   Rule 14/7: [102 cases, mean 32.44, range 13.4 to 50, est err 3.35]
##
##     if
##  rm > 3.483629
##  dis > 0.6492998
##     then
##  outcome = 126.92 + 22.7 rm - 4.68 lstat - 85 tax - 0.036 ptratio
##            - 1.1 dis + 0.007 zn
##
##   Rule 14/8: [84 cases, mean 33.40, range 21 to 50, est err 2.44]
##
##     if
##  rm > 3.483629
##  tax <= 1.896025
##     then
##  outcome = 347.12 + 25.2 rm - 213 tax - 3.5 lstat - 0.013 ptratio
##
##   Rule 14/9: [10 cases, mean 35.13, range 21.9 to 50, est err 12.13]
##
##     if
##  dis <= 0.6492998
##  lstat <= 2.848535
##     then
##  outcome = 72.65 - 77.8 dis
##
## Model 15:
##
##   Rule 15/1: [28 cases, mean 12.35, range 5 to 27.9, est err 4.09]
##
##     if
##  crim > 2.405809
##  b > 16084.5
##     then
##  outcome = 53.45 - 7.8 crim - 3.5 lstat - 0.0189 age
##
##   Rule 15/2: [11 cases, mean 13.56, range 8.3 to 27.5, est err 5.99]
##
##     if
##  crim > 2.405809
##  b <= 16084.5
##     then
##  outcome = 8.73 + 0.001756 b
##
##   Rule 15/3: [244 cases, mean 17.56, range 5 to 31, est err 2.73]
##
##     if
##  lstat > 2.848535
##     then
##  outcome = 103.02 - 0.0251 age - 2.37 lstat - 3.5 dis + 6.8e-05 b + 4 rm
##            - 0.035 ptratio - 41 tax - 0.25 crim
```

```
##
##   Rule 15/4: [131 cases, mean 28.22, range 16.5 to 50, est err 2.59]
##
##     if
##  dis > 1.086337
##  lstat <= 2.848535
##     then
##  outcome = 267.07 + 17.7 rm - 0.0421 age - 150 tax - 5.5 dis + 0.88 crim
##            - 0.035 ptratio + 0.031 zn - 0.12 lstat - 0.3 nox
##
##   Rule 15/5: [13 cases, mean 33.08, range 22 to 50, est err 4.44]
##
##     if
##  nox <= -0.7229691
##  dis <= 1.086337
##  lstat <= 2.848535
##     then
##  outcome = 148.52 - 0.002365 b - 85.9 nox - 1 dis + 0.16 crim + 0.8 rm
##            + 0.007 zn - 0.0016 age - 7 tax - 0.003 ptratio
##
##   Rule 15/6: [7 cases, mean 36.01, range 23.3 to 50, est err 7.00]
##
##     if
##  dis <= 0.6002641
##  lstat <= 2.848535
##     then
##  outcome = 50.55 - 68.1 dis - 11.4 nox + 0.00012 b + 1 rm - 0.008 ptratio
##
##   Rule 15/7: [12 cases, mean 41.77, range 21.9 to 50, est err 9.73]
##
##     if
##  nox > -0.7229691
##  dis > 0.6002641
##  lstat <= 2.848535
##     then
##  outcome = 13.74 - 92 nox - 40.5 dis - 0.023 ptratio + 2.6 rm
##
## Model 16:
##
##   Rule 16/1: [60 cases, mean 15.95, range 7.2 to 27.5, est err 3.16]
##
##     if
##  nox > -0.4344906
##     then
##  outcome = 46.98 - 6.53 lstat - 6.9 dis - 1.1 rm
##
##   Rule 16/2: [45 cases, mean 16.89, range 5 to 50, est err 5.45]
##
##     if
##  nox <= -0.4344906
##  dis <= 0.6557049
##     then
##  outcome = 35.33 - 37 dis - 51.7 nox - 7.38 lstat - 0.4 rm
##
```

```
##   Rule 16/3: [128 cases, mean 19.97, range 9.5 to 36.2, est err 2.52]
##
##     if
##  rm <= 3.626081
##  dis > 0.6557049
##  dis <= 1.298828
##  lstat > 2.133251
##     then
##  outcome = 61.65 - 3.35 lstat + 4.9 dis + 1.6 rm - 1.3 nox - 22 tax
##            + 0.5 rad + 1.8e-05 b + 0.09 crim - 0.004 ptratio
##
##   Rule 16/4: [140 cases, mean 21.93, range 12.7 to 35.1, est err 2.19]
##
##     if
##  rm <= 3.626081
##  dis > 1.298828
##     then
##  outcome = 54.16 - 3.58 lstat + 2.2 rad - 1.6 dis - 1.9 nox + 1.8 rm
##            - 17 tax + 1.3e-05 b + 0.06 crim - 0.003 ptratio
##
##   Rule 16/5: [30 cases, mean 21.97, range 14.4 to 29.1, est err 2.41]
##
##     if
##  rm <= 3.626081
##  dis > 1.298828
##  tax <= 1.879832
##  lstat > 2.133251
##     then
##  outcome = -1065.35 + 566 tax + 8.7 rm - 0.13 lstat - 0.2 dis - 0.3 nox
##
##   Rule 16/6: [22 cases, mean 30.88, range 10.4 to 50, est err 4.51]
##
##     if
##  rm > 3.626081
##  lstat > 2.133251
##     then
##  outcome = 42.24 + 18.7 rm - 1.5 indus - 1.84 lstat - 2.5 nox - 1.6 dis
##            - 39 tax + 0.7 rad - 0.012 ptratio + 0.0035 age + 1.2e-05 b
##            + 0.11 crim
##
##   Rule 16/7: [73 cases, mean 34.52, range 20.6 to 50, est err 3.36]
##
##     if
##  lstat <= 2.133251
##     then
##  outcome = 50.6 + 19.6 rm - 2.77 lstat - 3.2 nox - 1.7 dis - 45 tax
##            + 1 rad + 0.007 age - 0.014 ptratio
##
## Model 17:
##
##   Rule 17/1: [116 cases, mean 15.37, range 5 to 27.9, est err 2.55]
##
##     if
##  crim > 0.4779842
```

```
##  lstat > 2.944963
##     then
##  outcome = 35.96 - 3.68 crim - 3.41 lstat + 0.3 nox
##
##   Rule 17/2: [112 cases, mean 19.13, range 7 to 31, est err 2.14]
##
##     if
##  crim <= 0.4779842
##  lstat > 2.944963
##     then
##  outcome = 184.65 - 0.0365 age + 9 rm - 4.1 dis - 97 tax + 8.4e-05 b
##           - 0.024 ptratio
##
##   Rule 17/3: [9 cases, mean 28.37, range 15 to 50, est err 11.17]
##
##     if
##  dis <= 0.9547035
##  b <= 66469.73
##  lstat <= 2.944963
##     then
##  outcome = -1.12 + 0.000454 b
##
##   Rule 17/4: [179 cases, mean 29.28, range 15 to 50, est err 3.35]
##
##     if
##  lstat <= 2.944963
##     then
##  outcome = 278.16 + 20 rm - 7.4 dis - 0.0356 age - 161 tax + 0.051 zn
##           - 0.61 lstat + 0.17 crim - 0.008 ptratio
##
##   Rule 17/5: [23 cases, mean 36.10, range 15 to 50, est err 10.83]
##
##     if
##  dis <= 0.9547035
##  lstat <= 2.944963
##     then
##  outcome = 233.74 - 8.5 dis + 12.1 rm + 1.15 crim - 2.42 lstat - 113 tax
##           - 0.0221 age + 0.068 zn - 0.031 ptratio
##
## Model 18:
##
##   Rule 18/1: [84 cases, mean 14.29, range 5 to 27.5, est err 2.44]
##
##     if
##  nox > -0.4864544
##     then
##  outcome = 41.55 - 6.2 lstat + 14.6 nox + 3.8e-05 b
##
##   Rule 18/2: [163 cases, mean 19.37, range 7 to 31, est err 2.44]
##
##     if
##  nox <= -0.4864544
##  lstat > 2.848535
##     then
```

```
##  outcome = 172.79 - 3.67 lstat + 3.1 rad - 3.5 dis - 72 tax - 0.72 indus
##            - 0.033 ptratio - 1.2 nox + 0.0027 age + 0.6 rm + 0.05 crim
##            + 5e-06 b
##
##   Rule 18/3: [106 cases, mean 25.41, range 16.5 to 50, est err 2.76]
##
##     if
##  rm <= 3.626081
##  lstat <= 2.848535
##     then
##  outcome = 10.71 - 4.6 dis - 2.21 lstat + 2.3 rad + 5.5 rm - 5.3 nox
##            - 0.83 indus - 0.003 ptratio
##
##   Rule 18/4: [4 cases, mean 33.47, range 30.1 to 36.2, est err 5.61]
##
##     if
##  rm <= 3.626081
##  tax <= 1.863917
##  lstat <= 2.848535
##     then
##  outcome = 36.84
##
##   Rule 18/5: [10 cases, mean 35.13, range 21.9 to 50, est err 17.40]
##
##     if
##  dis <= 0.6492998
##  lstat <= 2.848535
##     then
##  outcome = 84.58 - 94.7 dis - 0.15 lstat
##
##   Rule 18/6: [57 cases, mean 38.38, range 21.9 to 50, est err 3.97]
##
##     if
##  rm > 3.626081
##  lstat <= 2.848535
##     then
##  outcome = 100.34 + 22.3 rm - 5.79 lstat - 0.062 ptratio - 69 tax
##            + 0.3 rad - 0.5 nox - 0.3 dis + 0.0011 age
##
##
## Evaluation on training data (407 cases):
##
##     Average  |error|               1.72
##     Relative |error|               0.26
##     Correlation coefficient        0.96
##
##
##  Attribute usage:
##    Conds  Model
##
##     72%    84%    lstat
##     38%    85%    dis
##     35%    80%    rm
##     27%    55%    nox
```

```
##      4%    58%    crim
##      2%    49%    b
##      2%    68%    ptratio
##      1%    78%    tax
##      1%    67%    age
##            41%    rad
##            36%    indus
##            20%    zn
##
##
## Time: 0.1 secs
```

We can now use this model to evaluate our held out validation dataset. Again, we must prepare the input data using the same Box-Cox transform.

```
# transform the validation dataset
set.seed(7)
valX <- validation[,1:13]
trans_valX <- predict(preprocessParams, valX)
valY <- validation[,14]
# use final model to make predictions on the validation dataset
predictions <- predict(finalModel, newdata=trans_valX, neighbors=3)
# calculate RMSE
rmse <- RMSE(predictions, valY)
r2 <- R2(predictions, valY)
print(rmse)
```

```
## [1] 3.237403
```

We can see that the estimated RMSE on this unseen data is 3.237, lower but not too dissimilar from our expected RMSE of 3.23.