Math 457 Project

Dylan Wingfield, Owen Brown, Haoyu Fang

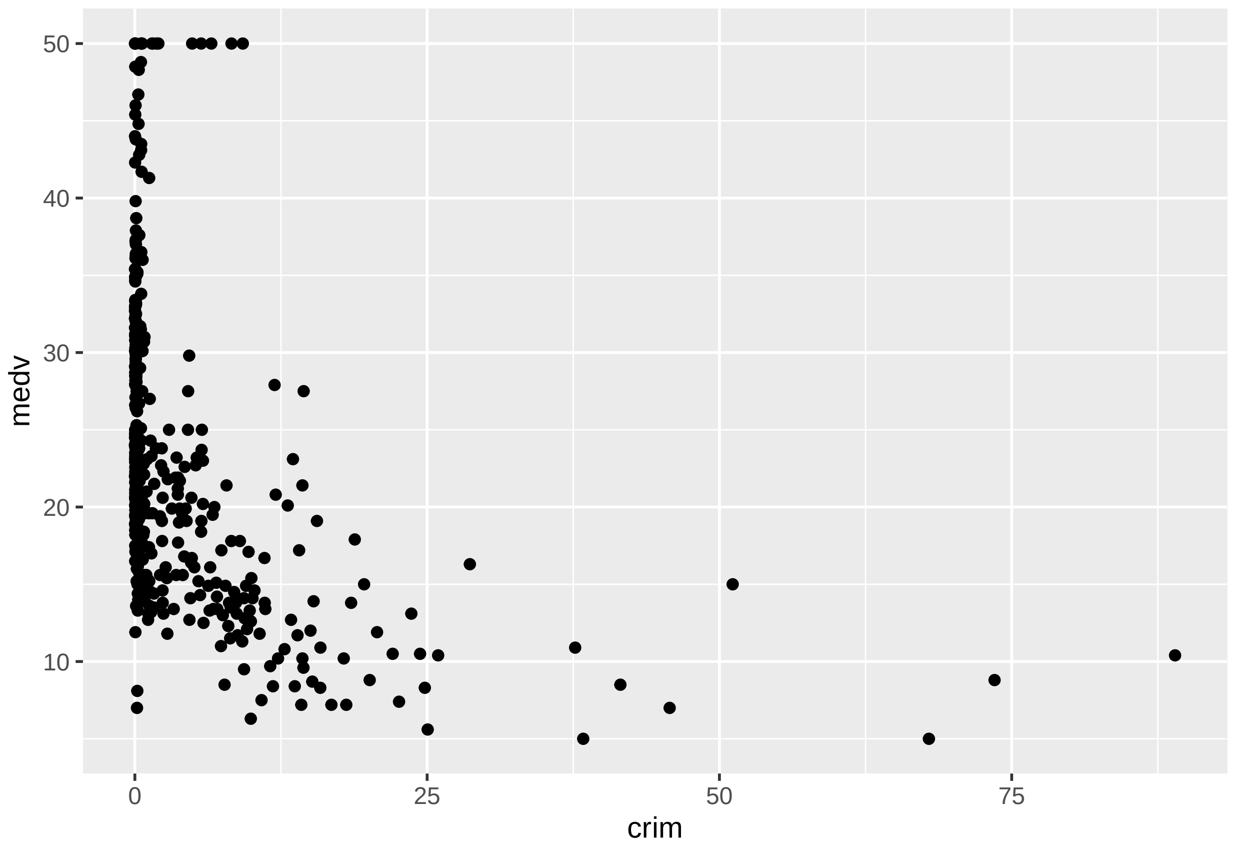
2022-12-08

Introduction

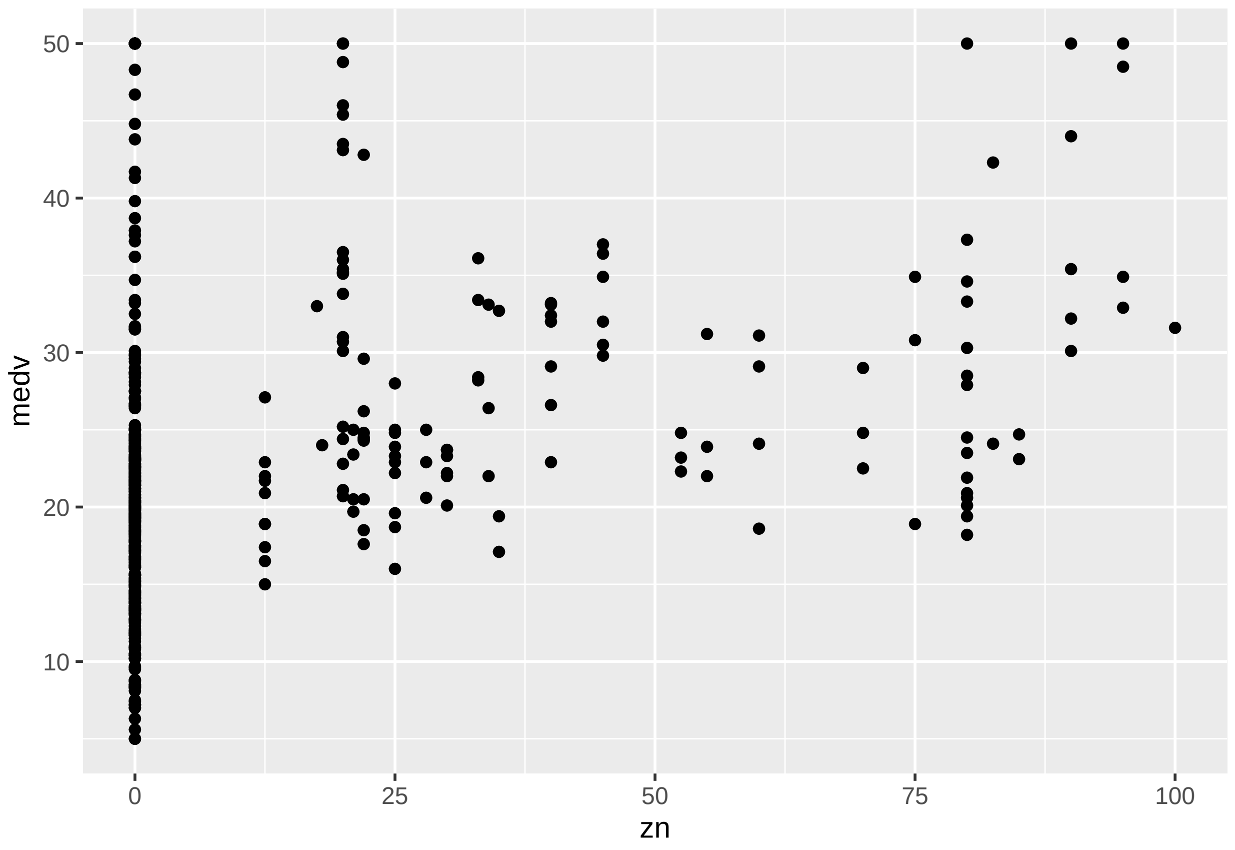
The dataset we have chosen to work with is BostonHousing in the mlbench package. BostonHousing consists of 506 observations of housing data acquired from the 1970 census. The response variable we are analyzing is medv, the median value of owner-occupied homes in USD 1000's. The 13 predictors include per capita crime rate by town (crim), nitric oxides concentration (nox), average number of rooms per dwelling (rm), and percentage of lower status of the population (lstat), among others. We have one categorical predictor, chas, a Charles River dummy variable, and the rest of our predictors are continuous. Our analysis of BostonHousing falls under supervised learning because we want to predict a response measurement, medv; and since our response is continuous, we will begin by performing linear regression with best subset variable selection, then explore penalized regression with ridge, lasso, and elastic net. Afterwards, we will segment our predictor space using tree-based methods, including regression trees, bagging, random forest and xgboosting. Finally, we will closely examine how the accuracy and complexity of each model compares to the rest. We will begin by plotting medv against the predictors to explore the data.

```r
library(mlbench)
data("BostonHousing")
```

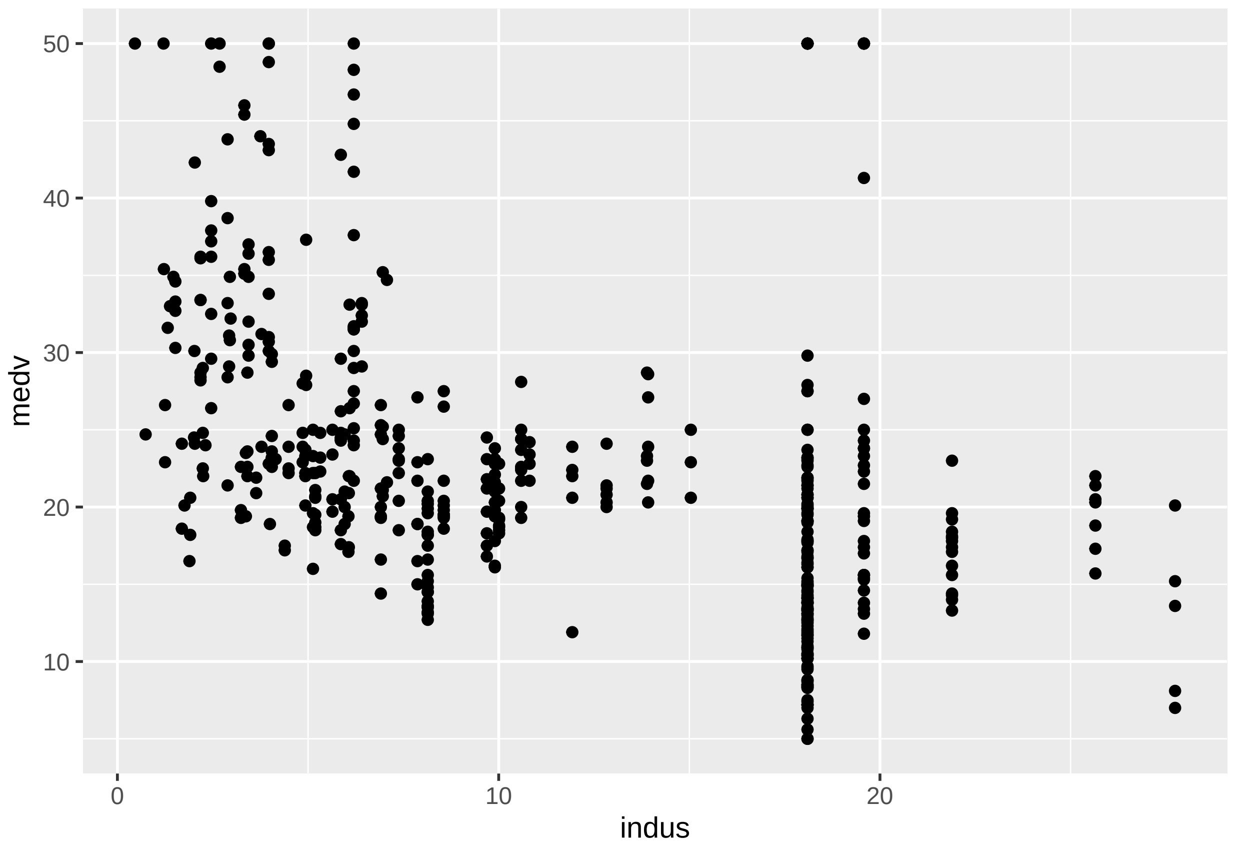
```r
library(ggplot2)
ggplot(BostonHousing, aes(x = crim,y= medv)) + geom_point()
```



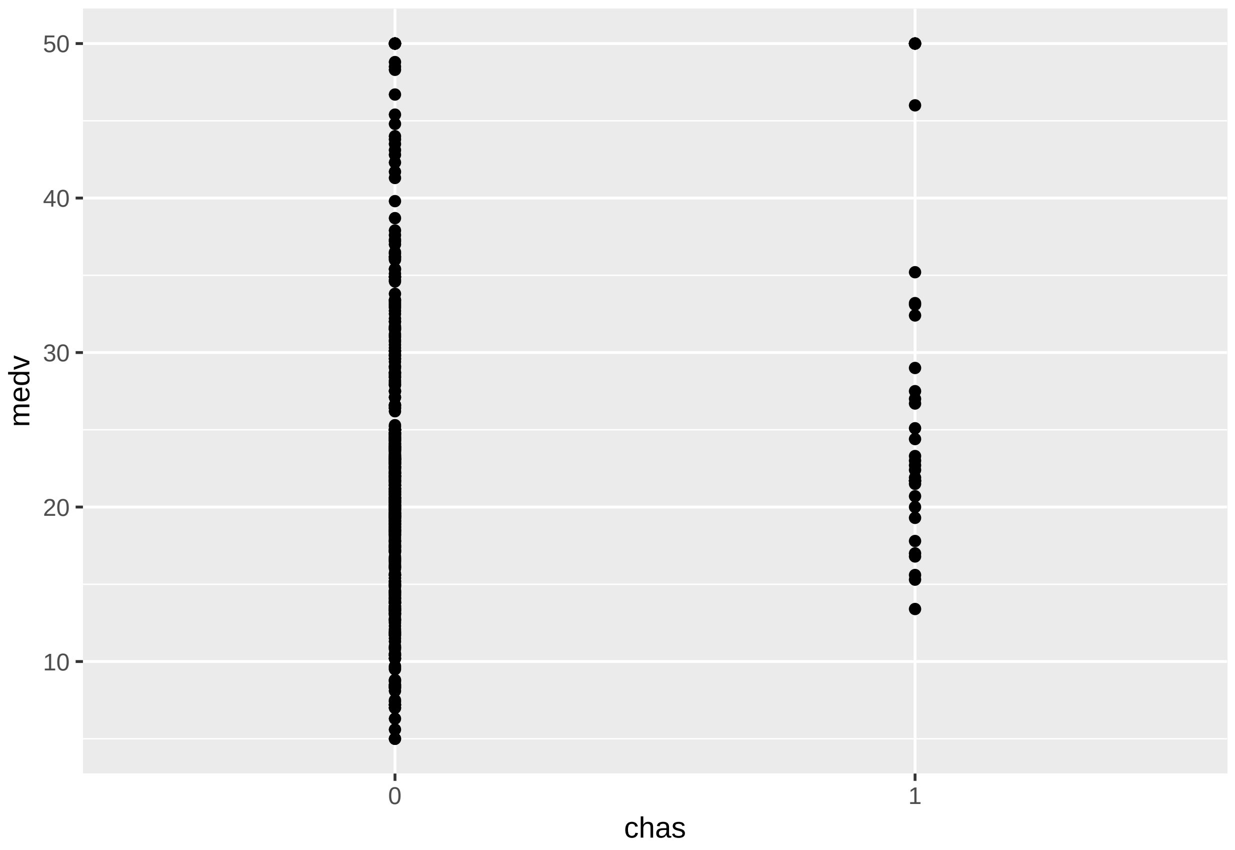
```
ggplot(BostonHousing, aes(x = zn,y= medv)) + geom_point()
```



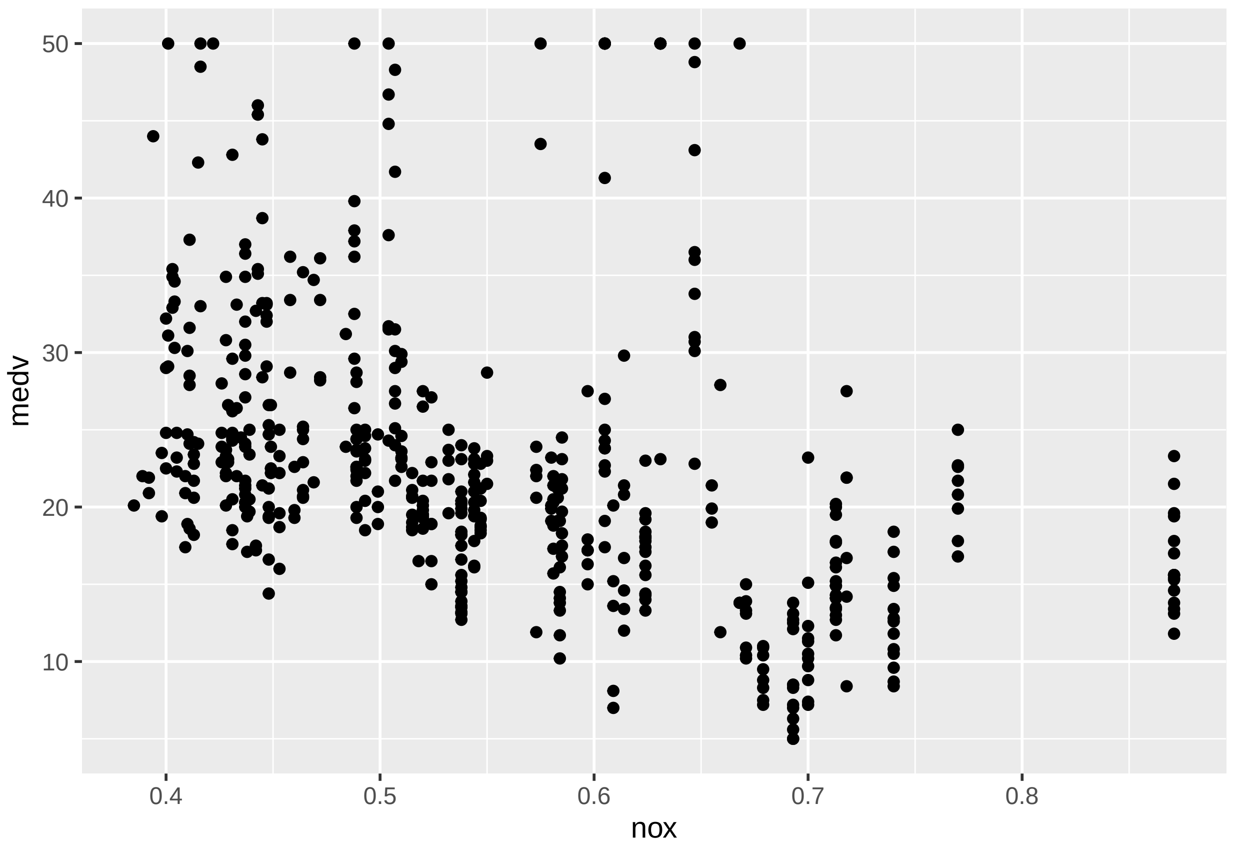
```
ggplot(BostonHousing, aes(x = indus,y= medv)) + geom_point()
```



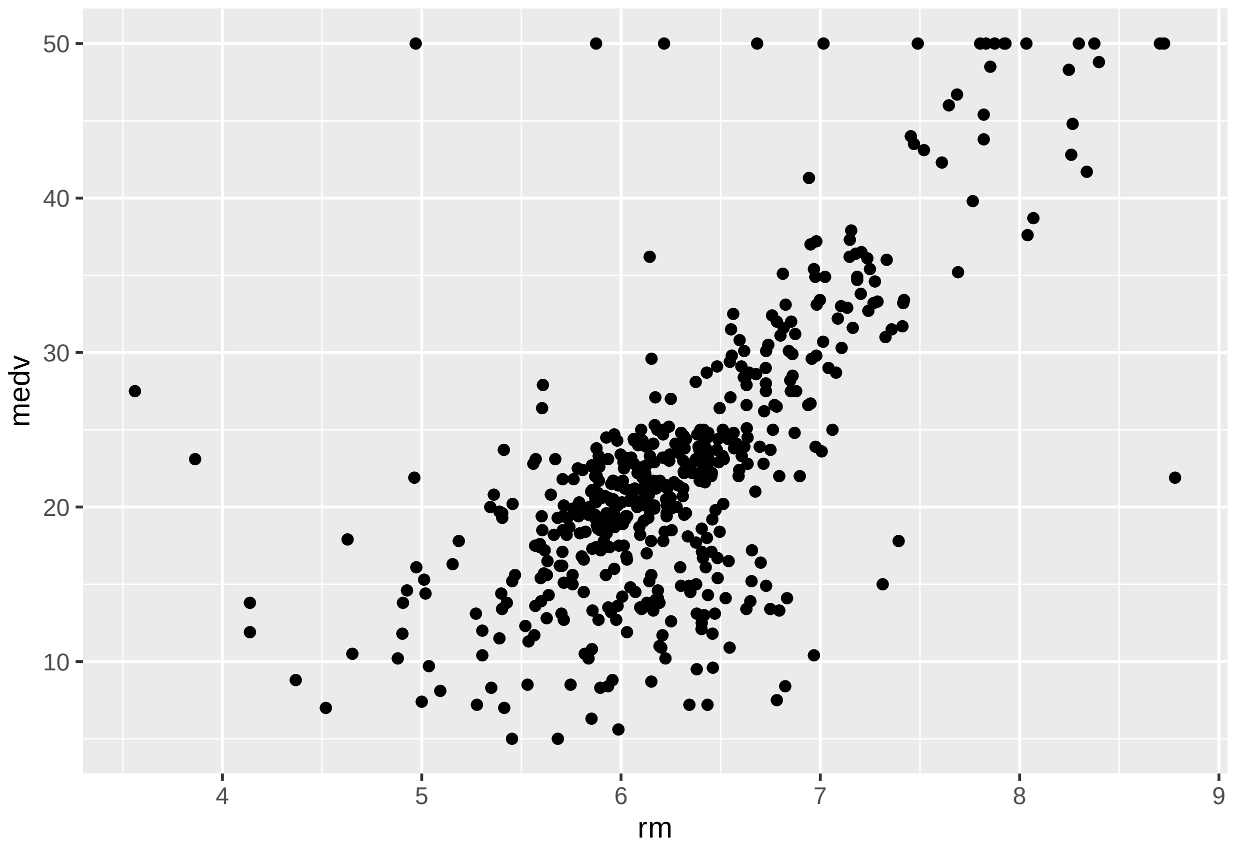
```
ggplot(BostonHousing, aes(x = chas,y= medv)) + geom_point()
```



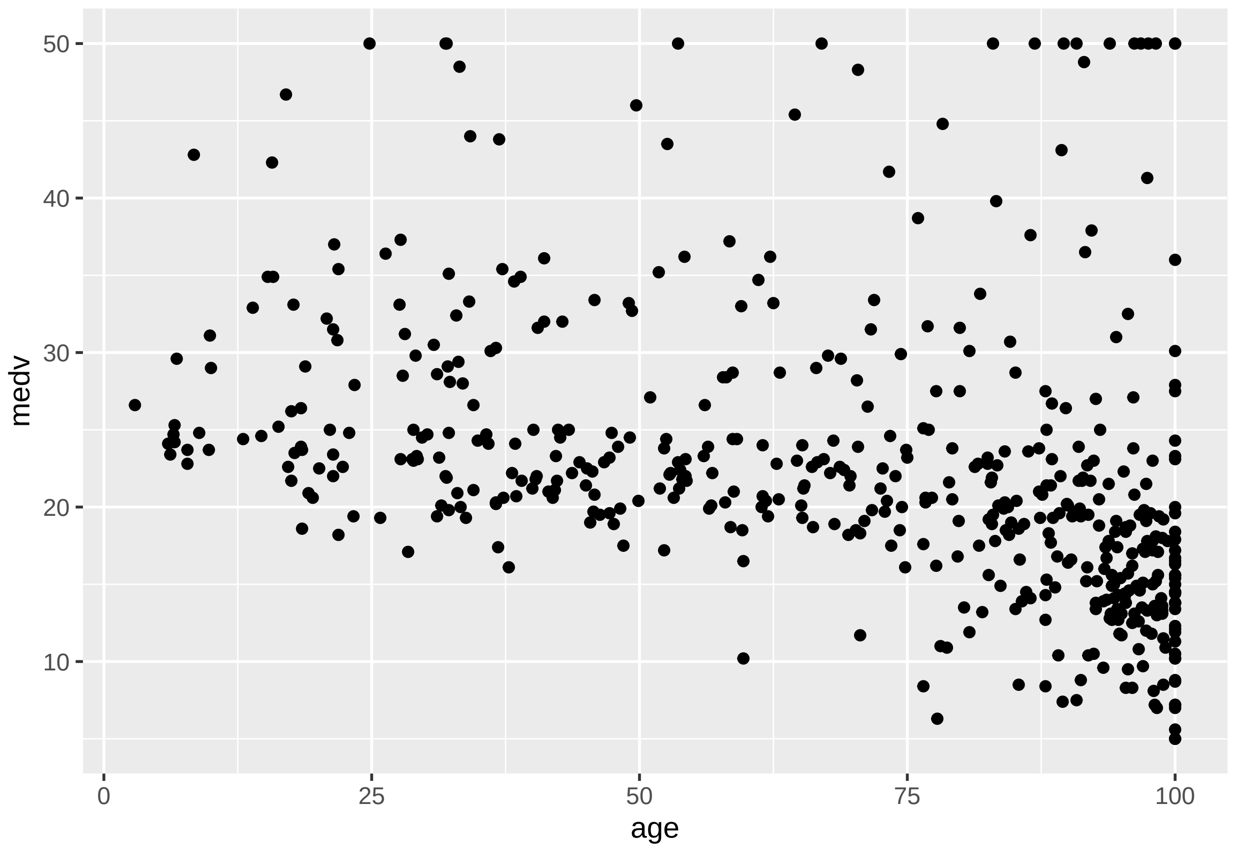
```
ggplot(BostonHousing, aes(x = nox,y= medv)) + geom_point()
```



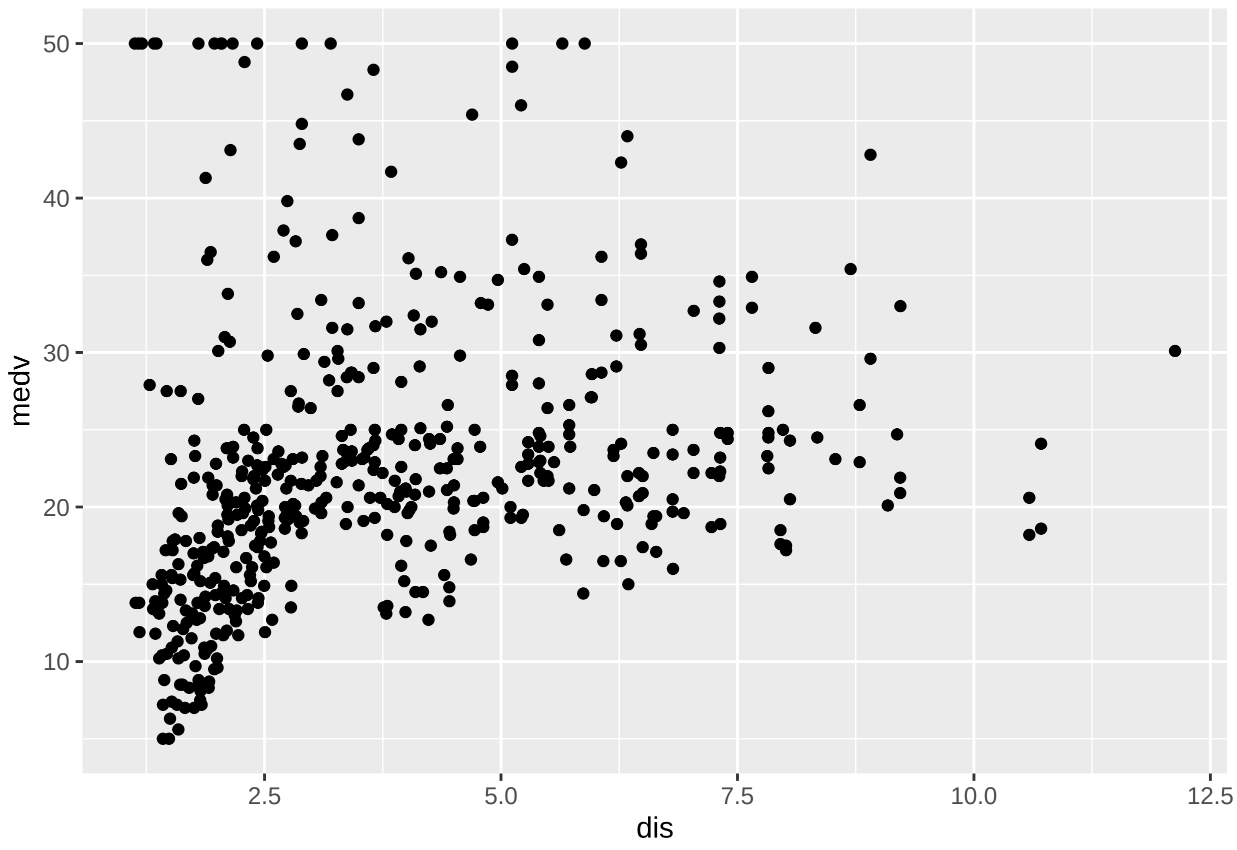
```
ggplot(BostonHousing, aes(x = rm,y= medv)) + geom_point()
```



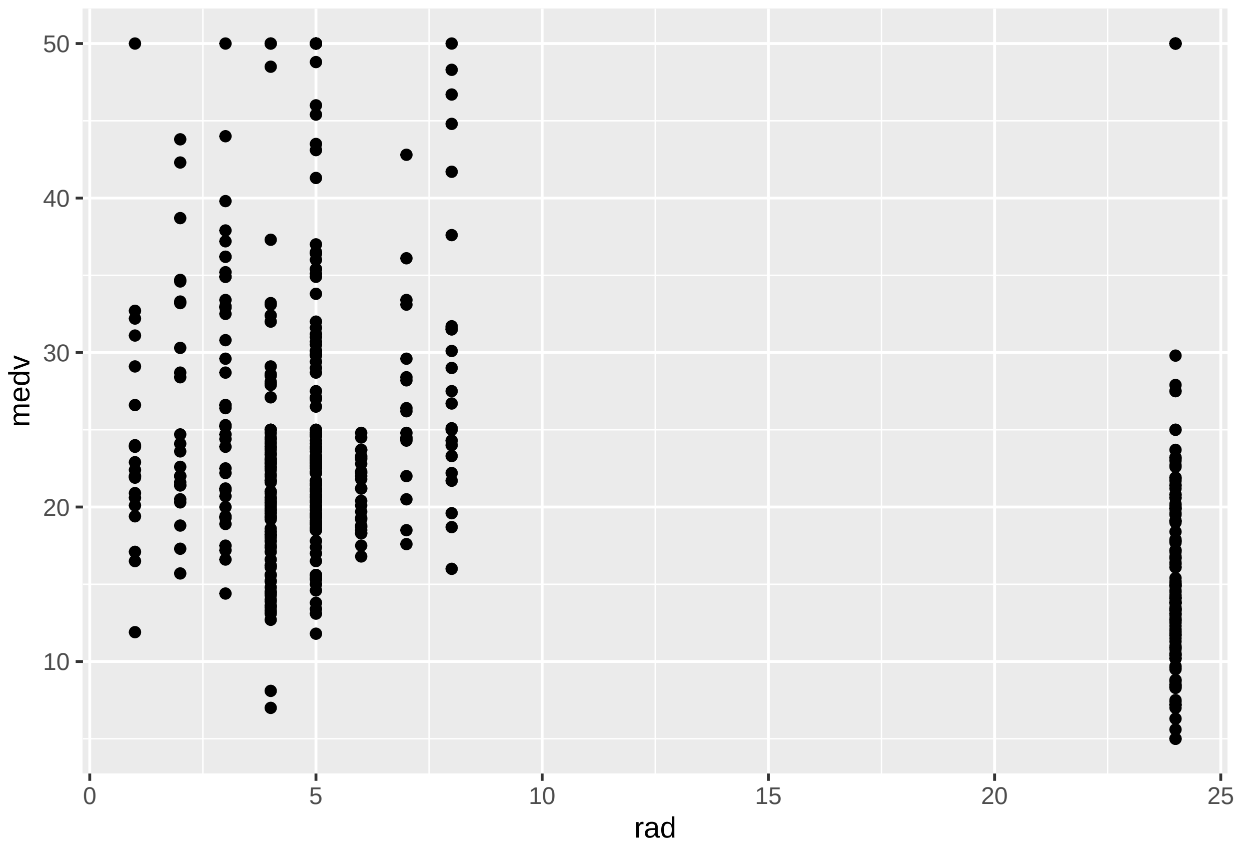
```
ggplot(BostonHousing, aes(x = age,y= medv)) + geom_point()
```



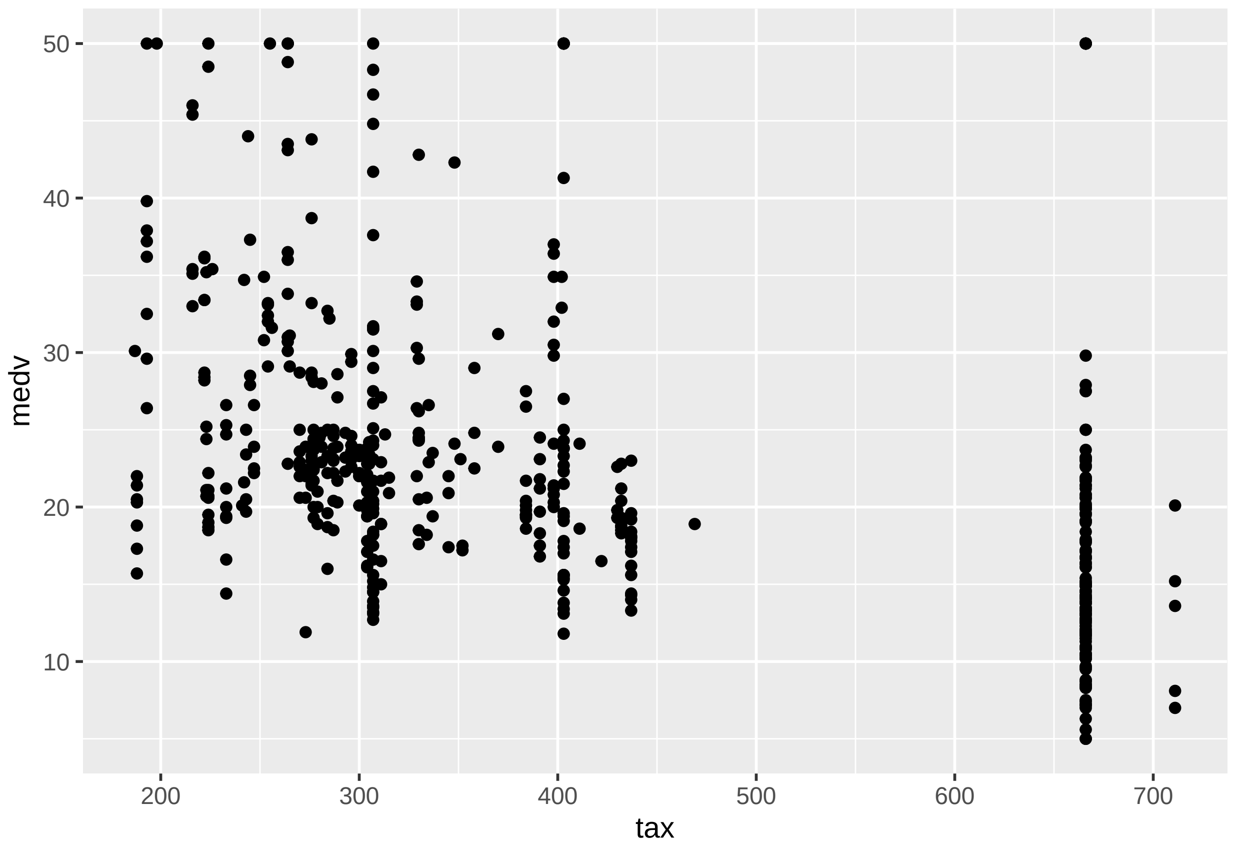
```
ggplot(BostonHousing, aes(x = dis,y= medv)) + geom_point()
```



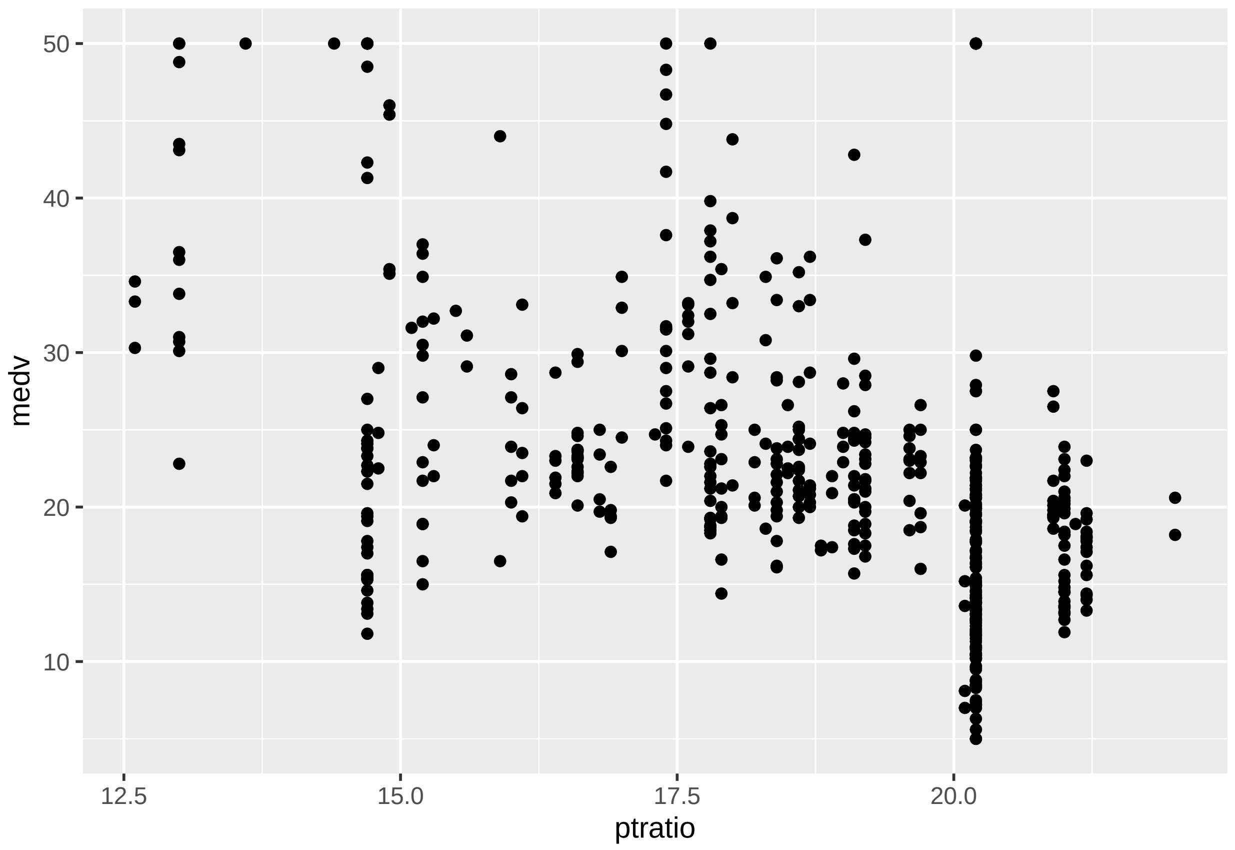
```
ggplot(BostonHousing, aes(x = rad,y= medv)) + geom_point()
```



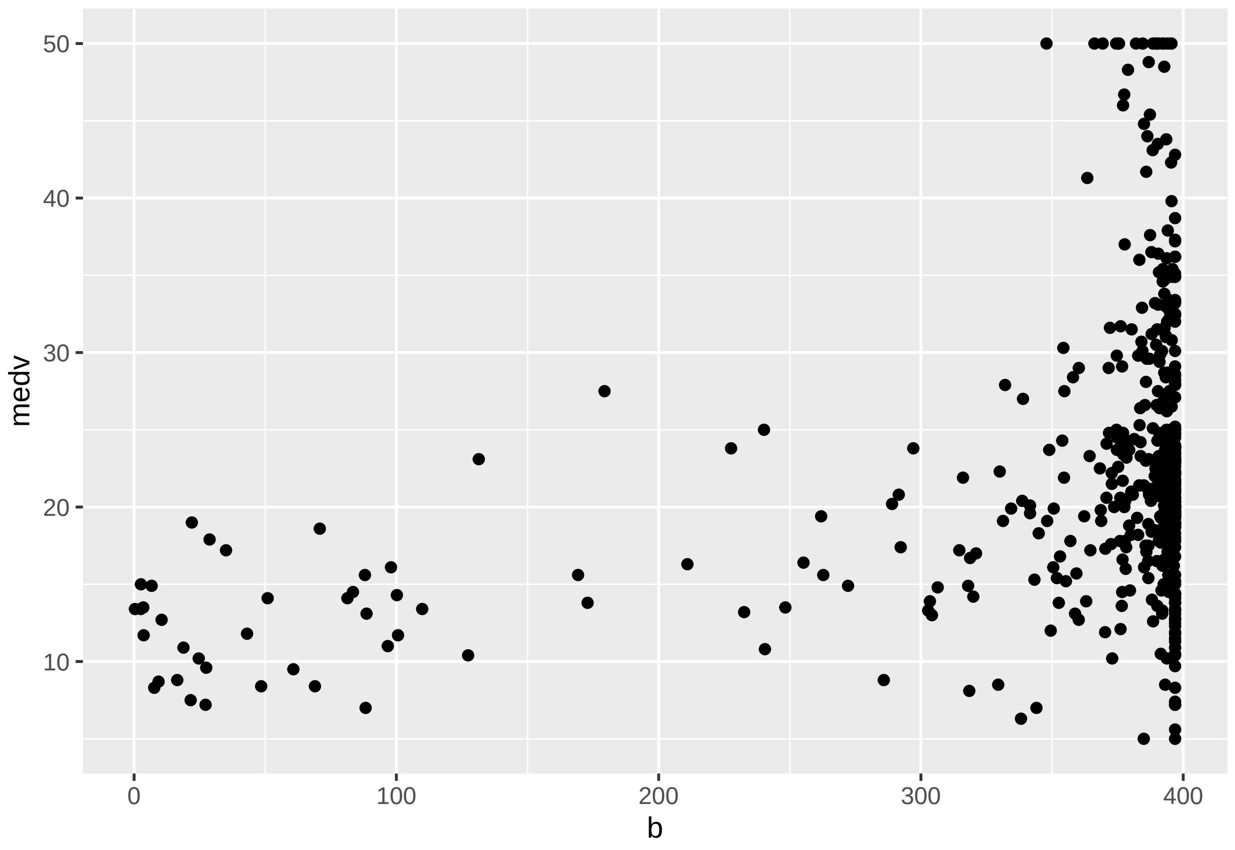
```
ggplot(BostonHousing, aes(x = tax,y= medv)) + geom_point()
```



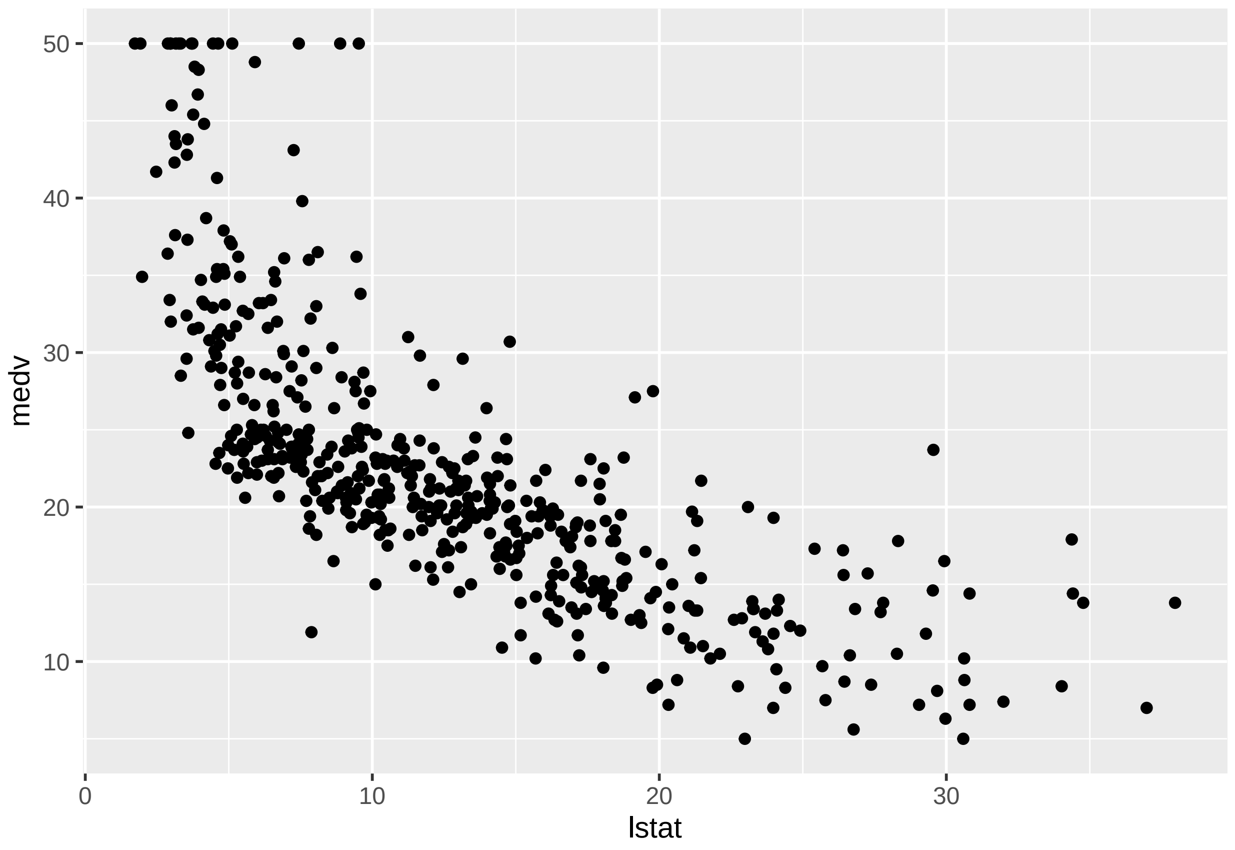
```r
ggplot(BostonHousing, aes(x = ptratio,y= medv)) + geom_point()
```



```
ggplot(BostonHousing, aes(x = b,y= medv)) + geom_point()
```



```
ggplot(BostonHousing, aes(x = lstat,y= medv)) + geom_point()
```

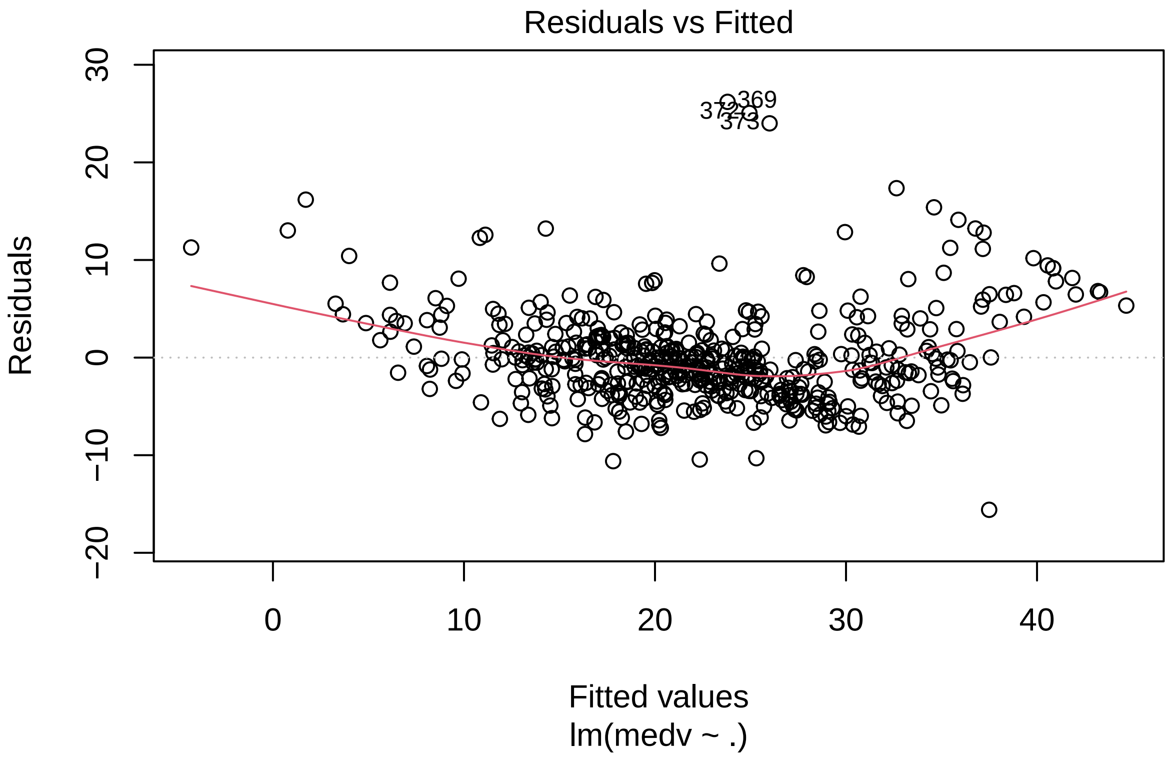

50
40
30
20
10
medv
0
10
20
30
lstat

Linear Regression

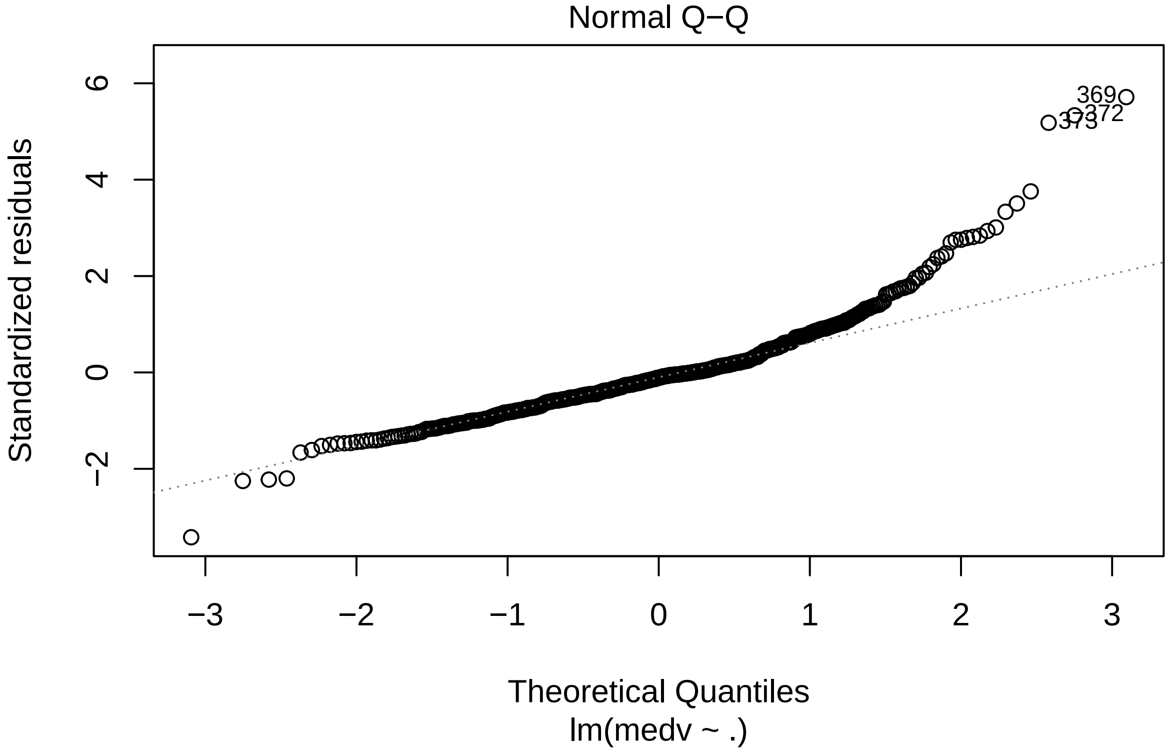
```
fit1 = lm(medv~.,data = BostonHousing)
summary(fit1)
```

```
##
## Call:
## lm(formula = medv ~ ., data = BostonHousing)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.595  -2.730  -0.518   1.777  26.199
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.646e+01  5.103e+00   7.144 3.28e-12 ***
## crim        -1.080e-01  3.286e-02  -3.287 0.001087 **
## zn           4.642e-02  1.373e-02   3.382 0.000778 ***
## indus        2.056e-02  6.150e-02   0.334 0.738288
## chas1        2.687e+00  8.616e-01   3.118 0.001925 **
## nox         -1.777e+01  3.820e+00  -4.651 4.25e-06 ***
## rm           3.810e+00  4.179e-01   9.116  < 2e-16 ***
## age          6.922e-04  1.321e-02   0.052 0.958229
## dis         -1.476e+00  1.995e-01  -7.398 6.01e-13 ***
## rad          3.060e-01  6.635e-02   4.613 5.07e-06 ***
## tax         -1.233e-02  3.760e-03  -3.280 0.001112 **
## ptratio     -9.527e-01  1.308e-01  -7.283 1.31e-12 ***
## b            9.312e-03  2.686e-03   3.467 0.000573 ***
## lstat       -5.248e-01  5.072e-02 -10.347  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.745 on 492 degrees of freedom
## Multiple R-squared:  0.7406, Adjusted R-squared:  0.7338
## F-statistic: 108.1 on 13 and 492 DF,  p-value: < 2.2e-16
```

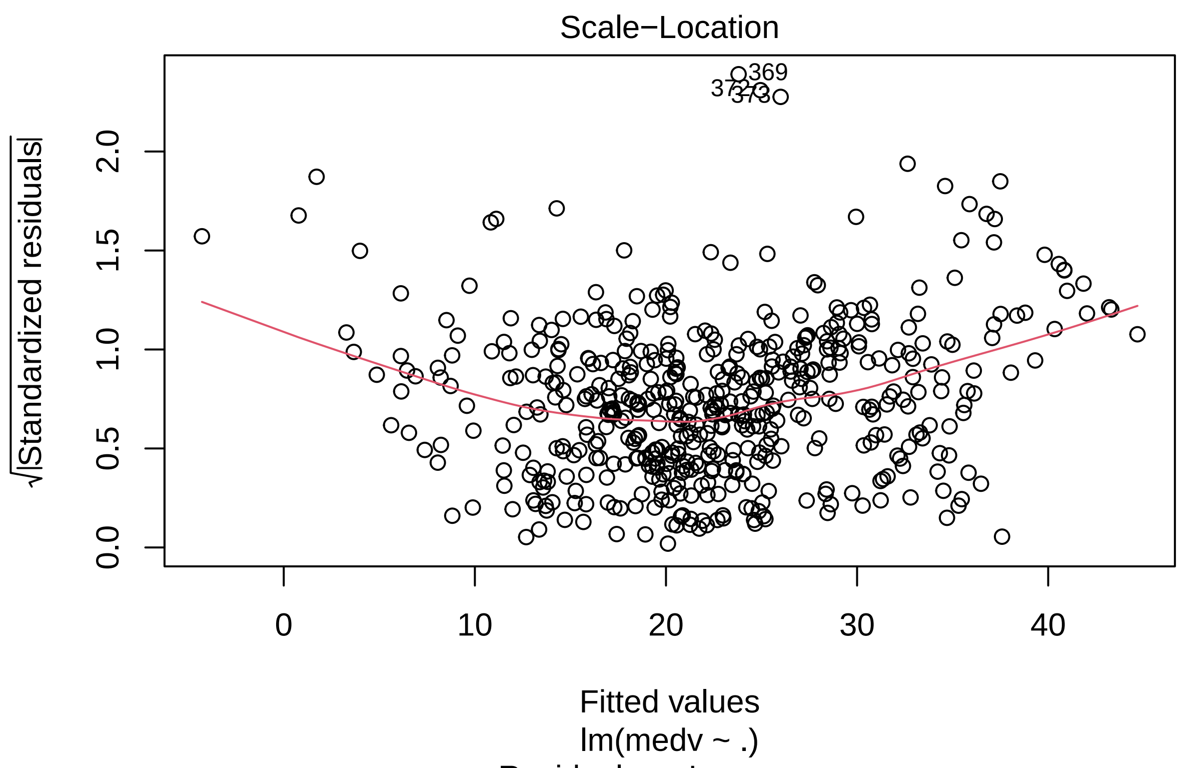
```
plot(fit1)
```

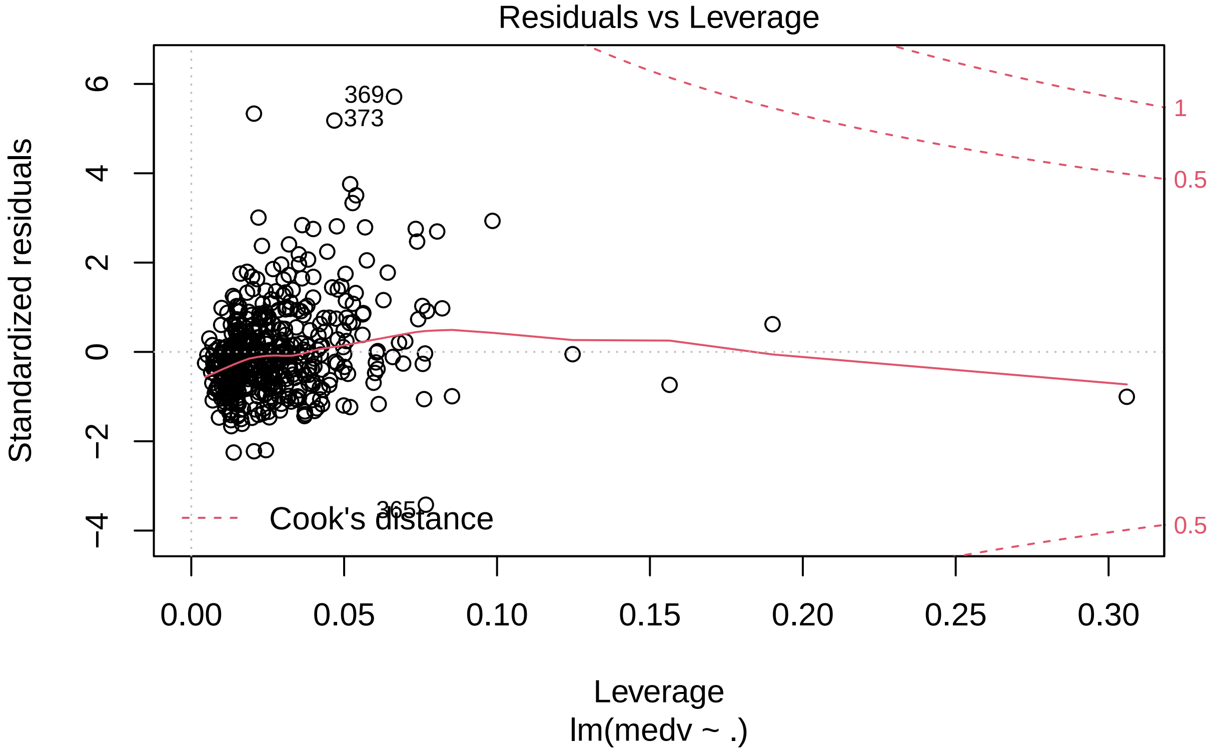


Residuals vs Fitted
369
372
373
Residuals
30
20
10
0
−10
−20
0
10
20
30
40
Fitted values
lm(medv ~ .)



Normal Q−Q
369
372
373
Standardized residuals
6
4
2
0
−2
−3
−2
−1
0
1
2
3
Theoretical Quantiles
lm(medv ~ .)





```
set.seed(123)
random_order = sample(c(rep(TRUE,405),rep(FALSE,101)))
train = BostonHousing[random_order,]
test = BostonHousing[!random_order,]
fit2 = lm(medv~.,data = train)
summary(fit2)
```

```
##
## Call:
## lm(formula = medv ~ ., data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.1126  -2.5759  -0.5468   1.5479  25.5823
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  35.401949   5.552720   6.376 5.14e-10 ***
## crim         -0.123094   0.036659  -3.358 0.000863 ***
## zn            0.046391   0.015079   3.077 0.002242 **
## indus        -0.027014   0.065920  -0.410 0.682177
## chas1         2.259623   0.945376   2.390 0.017313 *
## nox         -16.418753   4.204794  -3.905 0.000111 ***
## rm            3.645301   0.441596   8.255 2.36e-15 ***
## age           0.007557   0.015046   0.502 0.615781
## dis          -1.485241   0.229194  -6.480 2.76e-10 ***
## rad           0.309384   0.072267   4.281 2.34e-05 ***
## tax          -0.012414   0.004028  -3.082 0.002202 **
## ptratio      -0.863209   0.145302  -5.941 6.28e-09 ***
## b             0.009715   0.002964   3.277 0.001142 **
## lstat        -0.550417   0.055366  -9.941  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.637 on 391 degrees of freedom
## Multiple R-squared:  0.7557, Adjusted R-squared:  0.7476
## F-statistic: 93.05 on 13 and 391 DF,  p-value: < 2.2e-16
```

```
MSE1 = sqrt(mean((predict.lm(fit2, test) - test$medv)^2))
MSE1
```

```
## [1] 5.195852
```

```
#choosing variables for OLS bestsubset

library(leaps)
best = regsubsets(medv~., data = BostonHousing, method = c("exhaustive"))
summary(best)$which
```

```
##   (Intercept)  crim    zn indus chas1   nox    rm   age   dis   rad   tax
## 1        TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 2        TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## 3        TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## 4        TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE
## 5        TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE
## 6        TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE
## 7        TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE
## 8        TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE
##   ptratio     b lstat
## 1   FALSE FALSE  TRUE
## 2   FALSE FALSE  TRUE
## 3    TRUE FALSE  TRUE
## 4    TRUE FALSE  TRUE
## 5    TRUE FALSE  TRUE
## 6    TRUE FALSE  TRUE
## 7    TRUE  TRUE  TRUE
## 8    TRUE  TRUE  TRUE
```

```
(size_ind = which.min(summary(best)$bic))
```

```
## [1] 8
```

```
(var_ind = colnames(BostonHousing)[summary(best)$which[size_ind,][-1]])
```

```
## [1] "zn"      "chas"    "nox"     "rm"      "dis"     "ptratio" "b"
## [8] "lstat"
```

```
forward = regsubsets(medv~., data = BostonHousing, method = c("forward"))
summary(forward)$which
```

```
##   (Intercept)  crim    zn indus chas1   nox    rm   age   dis   rad   tax
## 1        TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 2        TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## 3        TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## 4        TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE
## 5        TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE
## 6        TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE
## 7        TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE
## 8        TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE
##   ptratio     b lstat
## 1   FALSE FALSE  TRUE
## 2   FALSE FALSE  TRUE
## 3    TRUE FALSE  TRUE
## 4    TRUE FALSE  TRUE
## 5    TRUE FALSE  TRUE
## 6    TRUE FALSE  TRUE
## 7    TRUE  TRUE  TRUE
## 8    TRUE  TRUE  TRUE
```

```
(size_ind = which.min(summary(forward)$bic))
```

```
## [1] 8
```

```
(var_ind = colnames(BostonHousing)[summary(forward)$which[size_ind,][-1]])
```

```
## [1] "zn"      "chas"    "nox"     "rm"      "dis"     "ptratio" "b"
## [8] "lstat"
```

```
backward = regsubsets(medv~., data = BostonHousing, method = c("backward"))
summary(backward)$which
```

```
##   (Intercept)  crim    zn indus chas1   nox    rm   age   dis   rad   tax
## 1        TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## 2        TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## 3        TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## 4        TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE
## 5        TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE
## 6        TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE
## 7        TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE
## 8        TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE
##   ptratio     b lstat
## 1   FALSE FALSE  TRUE
## 2   FALSE FALSE  TRUE
## 3    TRUE FALSE  TRUE
## 4    TRUE FALSE  TRUE
## 5    TRUE FALSE  TRUE
## 6    TRUE  TRUE  TRUE
## 7    TRUE  TRUE  TRUE
## 8    TRUE  TRUE  TRUE
```

```
(size_ind = which.min(summary(backward)$bic))
```

```
## [1] 8
```

```
(var_ind = colnames(BostonHousing)[summary(backward)$which[size_ind,][-1]])
```

```
## [1] "crim"    "nox"     "rm"      "dis"     "rad"     "ptratio" "b"
## [8] "lstat"   "medv"
```

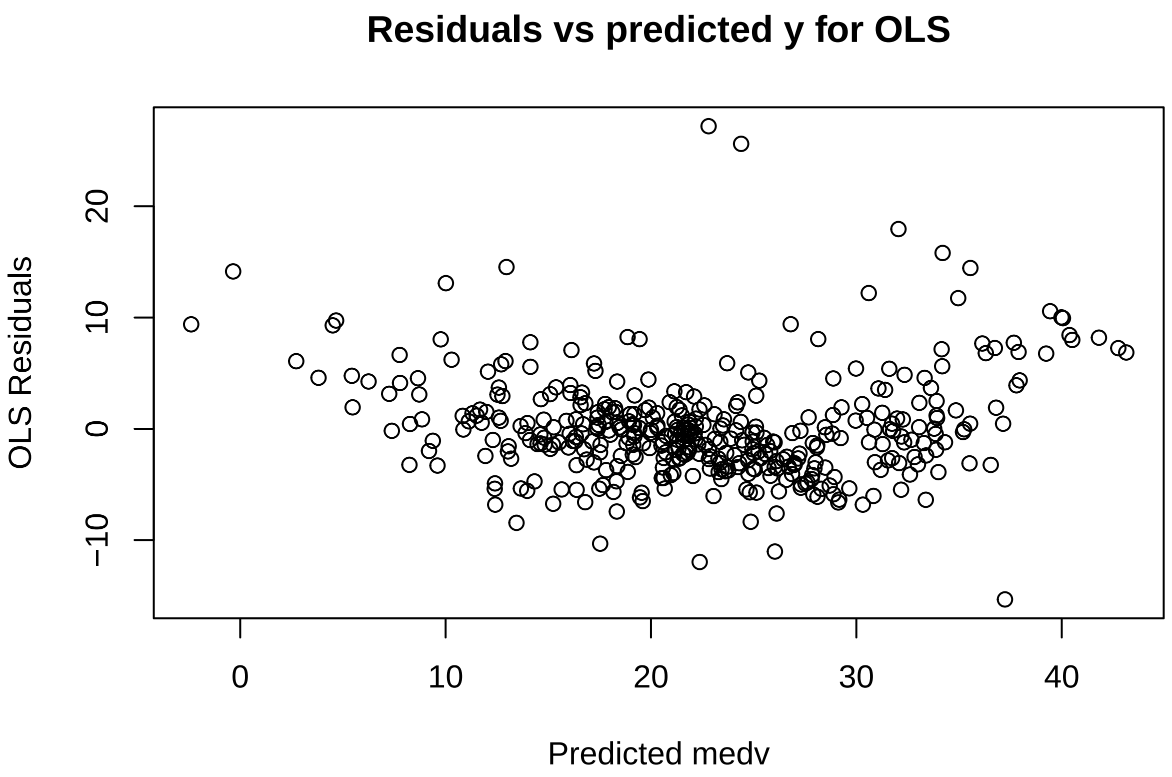
```
fit3 = lm(medv~zn+chas+nox+rm+dis+ptratio+b+lstat,data = train)
summary(fit3)
```

```
##
## Call:
## lm(formula = medv ~ zn + chas + nox + rm + dis + ptratio + b +
##     lstat, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.335  -2.767  -0.461   1.723  27.194
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  29.965757   5.314854   5.638 3.27e-08 ***
## zn            0.035593   0.014727   2.417 0.016107 *
## chas1         2.661487   0.955381   2.786 0.005596 **
## nox         -16.101732   3.542522  -4.545 7.30e-06 ***
## rm            4.018548   0.433395   9.272  < 2e-16 ***
## dis          -1.372609   0.212042  -6.473 2.84e-10 ***
## ptratio      -0.839917   0.128118  -6.556 1.73e-10 ***
## b             0.009779   0.002937   3.329 0.000952 ***
## lstat        -0.567217   0.051260 -11.066  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.76 on 396 degrees of freedom
## Multiple R-squared:  0.7393, Adjusted R-squared:  0.734
## F-statistic: 140.4 on 8 and 396 DF,  p-value: < 2.2e-16
```

```
MSE6 = sqrt(mean((predict.lm(fit3, test) - test$medv)^2))
MSE6
```

```
## [1] 5.195209
```

```
boston.res.lm <- resid(fit3)
yhat.lm <- predict(fit3)
plot(yhat.lm, boston.res.lm, ylab = "OLS Residuals", xlab = "Predicted medv", main = "Residuals vs predicted y for
```



Our first model we trained is the simple linear regression model, as well as using best subset selection in order to remove variables from the model. The best subset selection selected zn, chas1, nox, rm, dis, ptratio, b, and lstat as the varibles to use in the model.Our final model had a RMSE of 5.195209.

Ridge Regression

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.1.2
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-6
```

```
set.seed(1)

x = model.matrix(medv~., BostonHousing )[,-1]
y = BostonHousing$medv

ridge.mod <- glmnet(x, y, alpha = 0, nlambda = 100)
ridge.mod$lambda[100]
```

```
## [1] 0.6777654
```

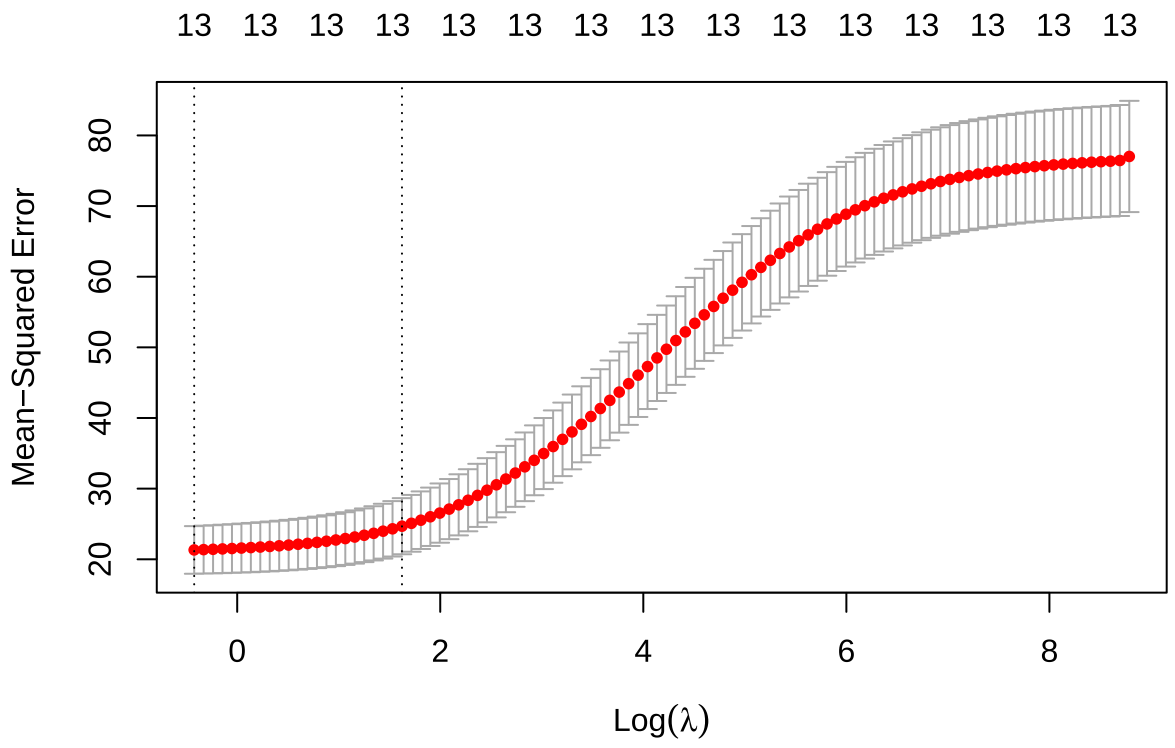
```
coef(ridge.mod)[, 100]
```

```
##   (Intercept)          crim            zn         indus         chas1
##  28.001475824  -0.087572712   0.032681030  -0.038003639   2.899781645
##           nox            rm           age           dis           rad
## -11.913360479   4.011308385  -0.003731470  -1.118874607   0.153730052
##           tax       ptratio             b         lstat
##  -0.005751054  -0.854984614   0.009073740  -0.472423800
```

```
sqrt(sum(coef(ridge.mod)[-1, 100]^2))
```

```
## [1] 12.9872
```

```
set.seed(1)
train <- sample(1:nrow(x), nrow(x) / 2)
test <- (-train)
y.test <- y[test]

cv.out <- cv.glmnet(x[train, ], y[train], alpha = 0)
plot(cv.out)
```



```r
bestlam <- cv.out$lambda.min
bestlam
```

```
## [1] 0.6546266
```

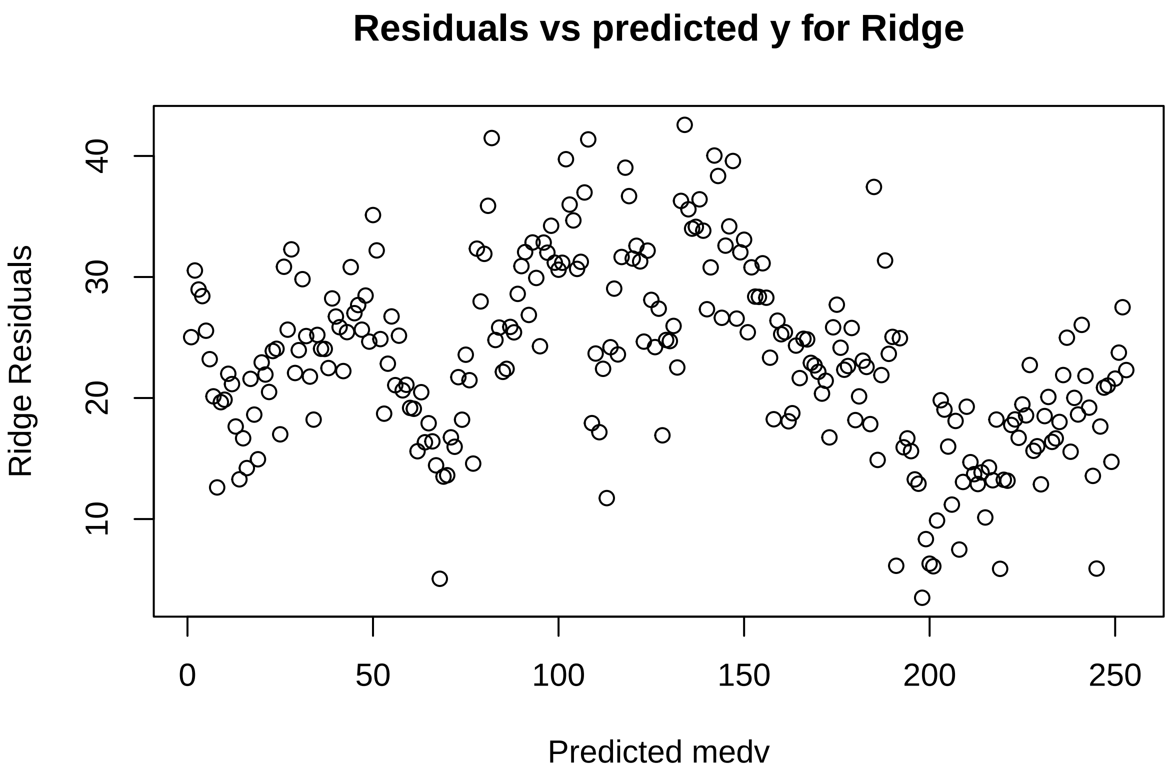
```r
ridge.pred <- predict(ridge.mod, s = bestlam, newx = x[test, ])

sqrt(mean((ridge.pred - y.test)^2))
```

```
## [1] 5.116954
```

```r
med <- glmnet(x, y, alpha = 0)
predict(med, type = "coefficients", s = bestlam)[1:13, ]
```

```
##   (Intercept)          crim            zn         indus         chas1 
##  28.001475824  -0.087572712   0.032681030  -0.038003639   2.899781645 
##           nox            rm           age           dis           rad 
## -11.913360479   4.011308385  -0.003731470  -1.118874607   0.153730052 
##           tax       ptratio             b 
##  -0.005751054  -0.854984614   0.009073740
```

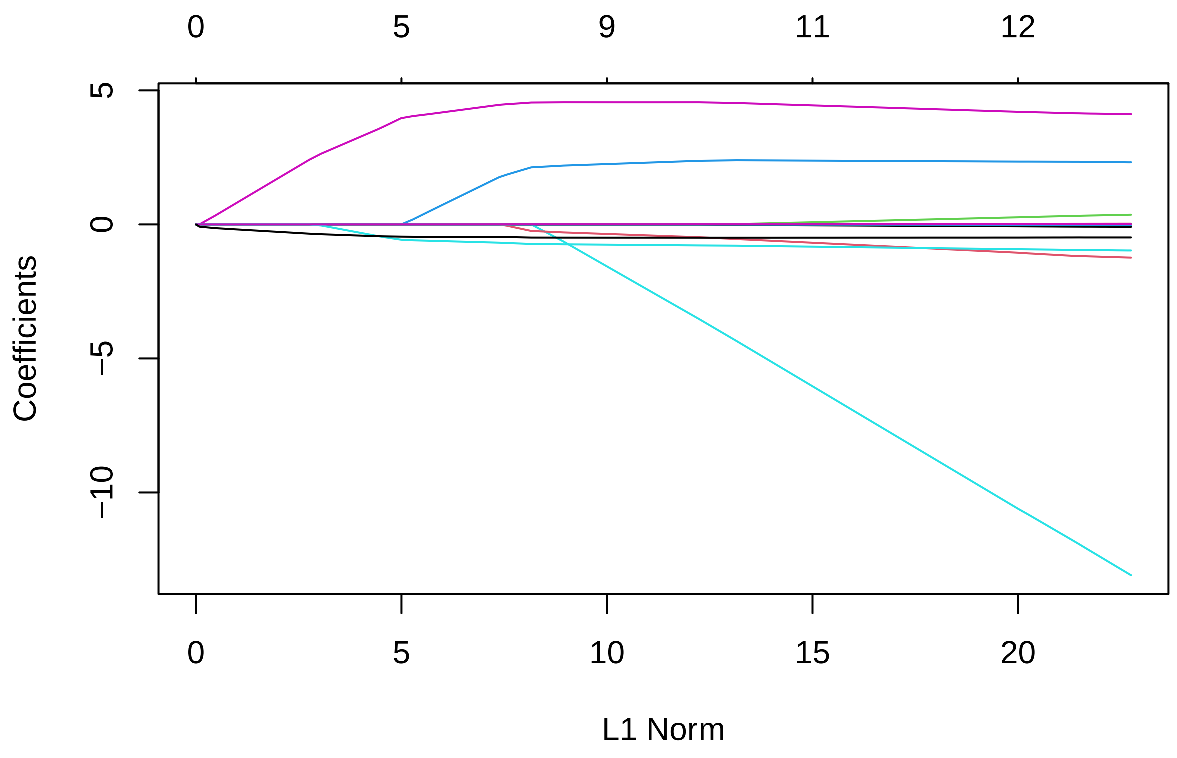
```r
boston.res.ridge <- resid(med)
plot(ridge.pred, boston.res.ridge, ylab = "Ridge Residuals", xlab = "Predicted medv", main = "Residuals vs predicte
```



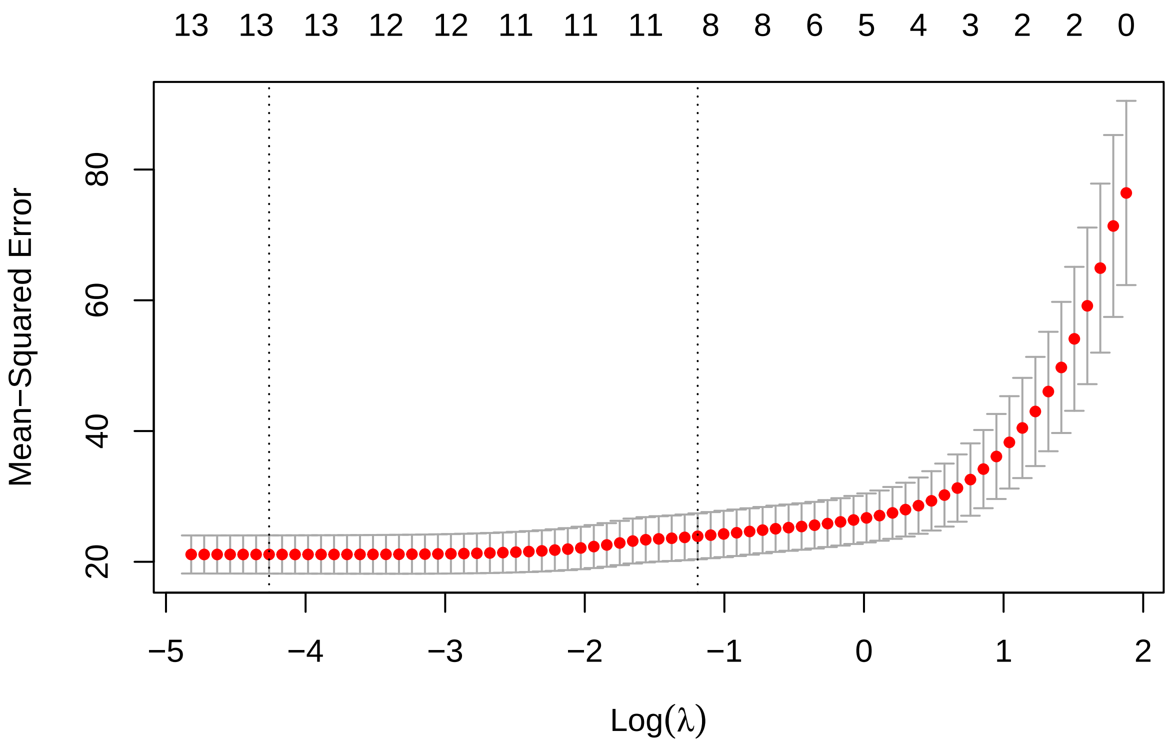
Here we used a ridge regression model. We used cross-validation in order to find the best lambda. The analysis found that the best lambda was 0.6546266. We used this lambda to predict. The RMSE/test error was 5.116954.

Lasso Regression

```
lasso.mod <- glmnet(x[train, ], y[train], alpha = 1, nlambda = 100)
plot(lasso.mod)
```



```
set.seed(1)
cv.out <- cv.glmnet(x[train, ], y[train], alpha = 1)
plot(cv.out)
```



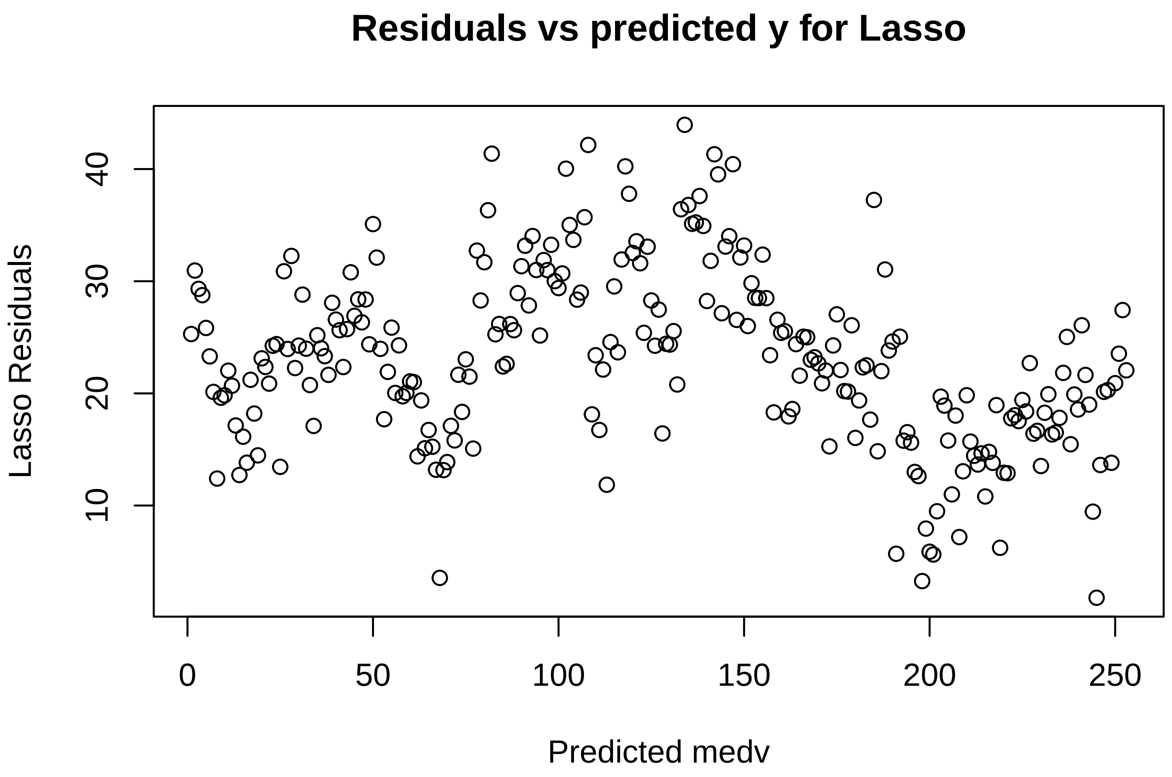
```
bestlamlasso <- cv.out$lambda.min
bestlamlasso
```

```
## [1] 0.0141035
```

```
lasso.pred <- predict(lasso.mod, s = bestlamlasso,newx = x[test, ])
sqrt(mean((lasso.pred - y.test)^2))
```

```
## [1] 5.182662
```

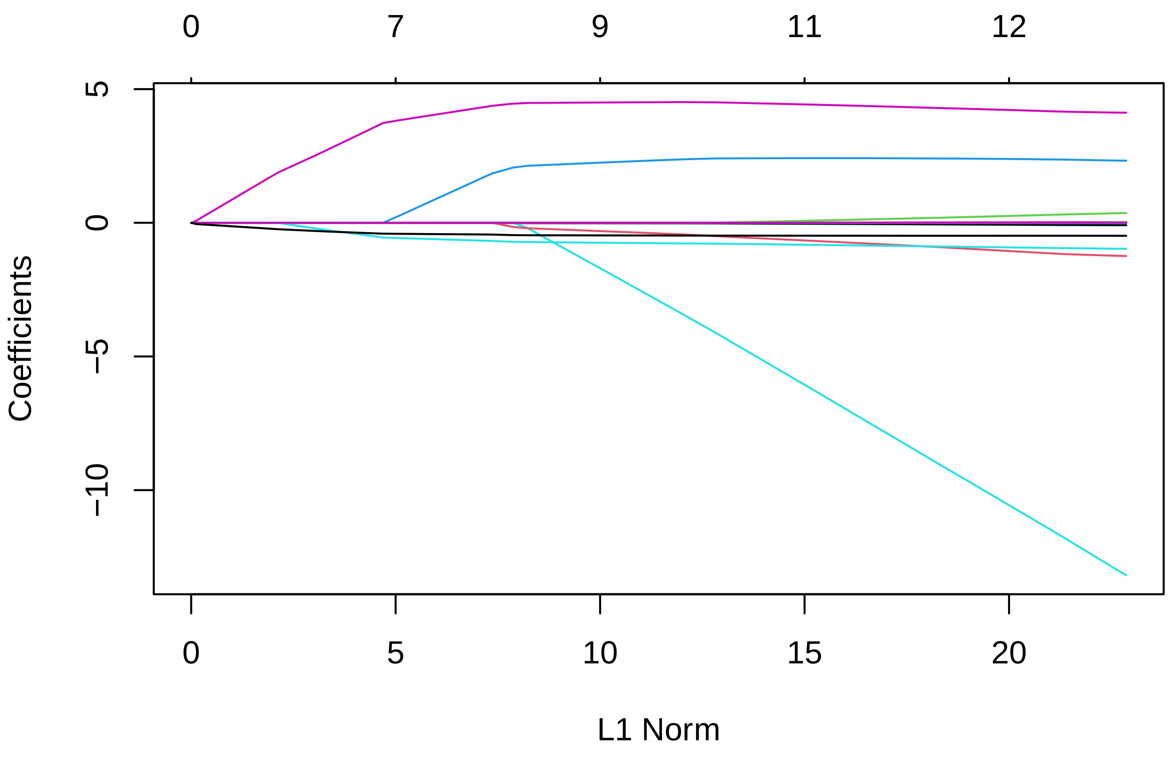
```
boston.res.lasso <- resid(lasso.mod)
plot(lasso.pred, boston.res.lasso, ylab = "Lasso Residuals", xlab = "Predicted medv", main = "Residuals vs predicte
```



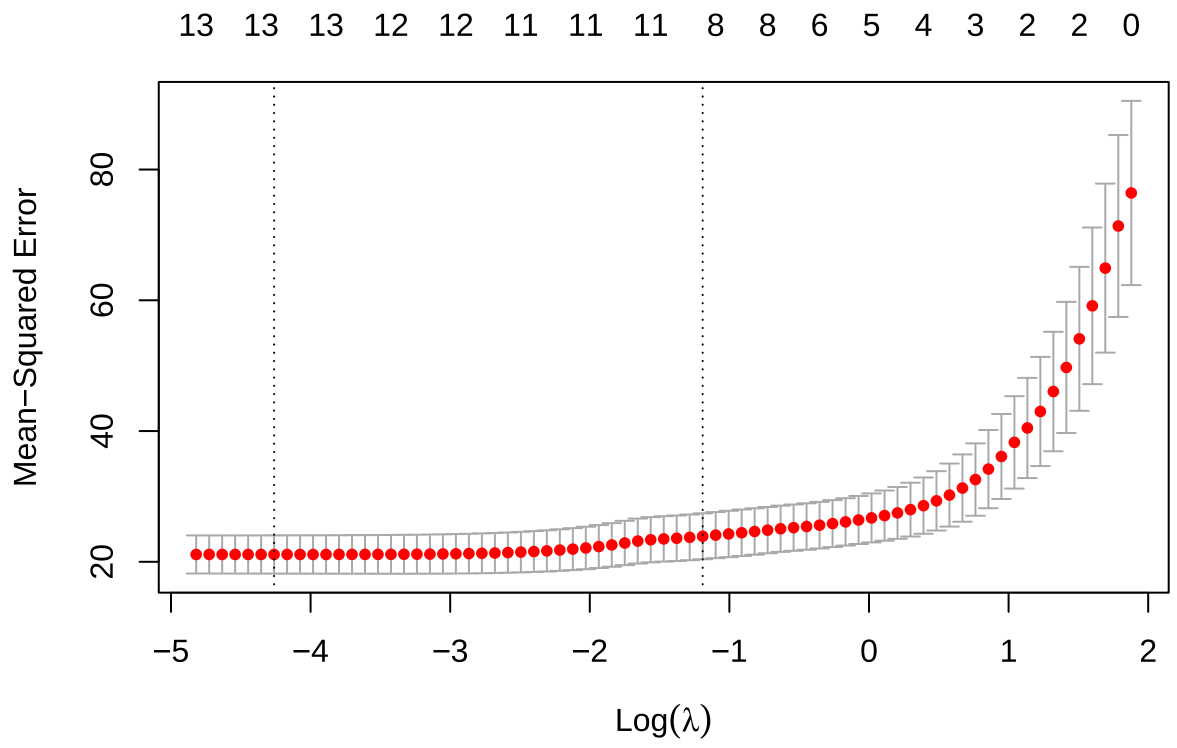
The lasso model held similar results to the ridge regression. After Cross-Validation, the analysis found that the optimal lambda was 0.0141035. Here, the RMSE/test error is 5.182662.

Elastic Net

```
net.mod <- glmnet(x[train, ], y[train], alpha = 0.5, nlambda = 100)
plot(net.mod)
```



```
set.seed(1)
cv.out.net <- cv.glmnet(x[train, ], y[train], alpha = 0.5)
plot(cv.out)
```



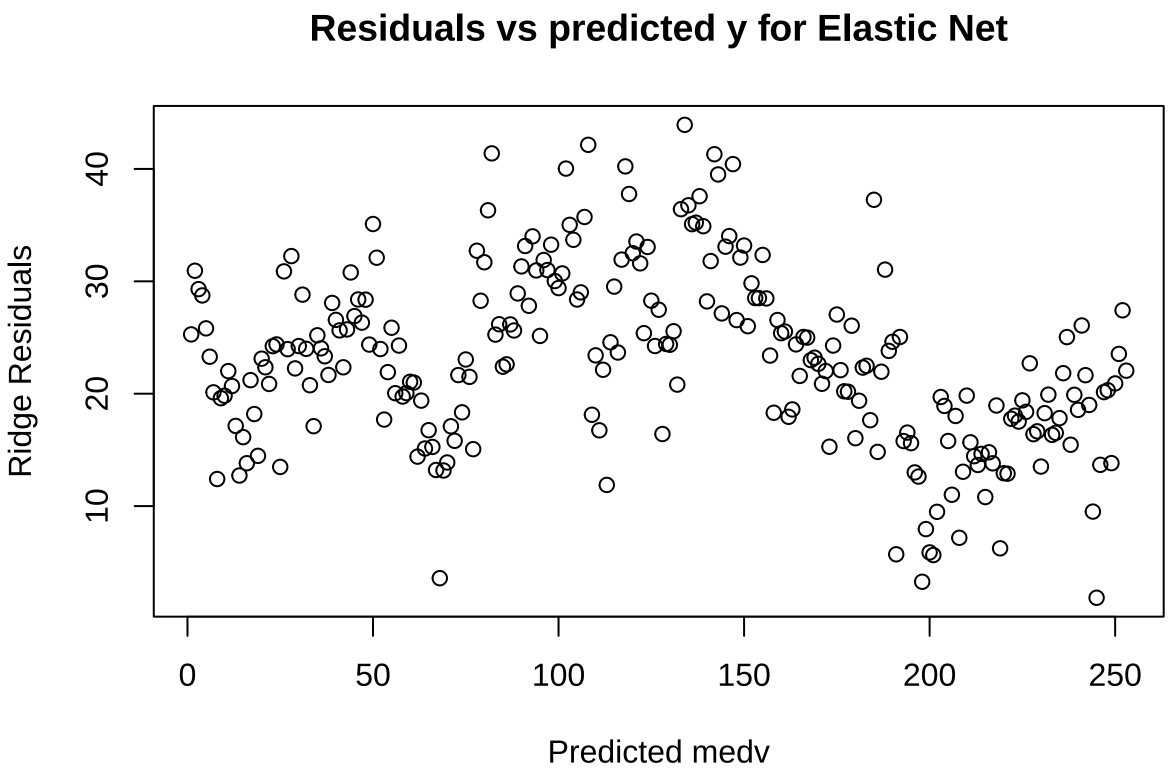
```
bestlamnet <- cv.out.net$lambda.min
bestlamnet
```

```
## [1] 0.02341795
```

```
net.pred <- predict(net.mod, s = bestlamnet,newx = x[test, ])
sqrt(mean((net.pred - y.test)^2))
```

```
## [1] 5.181197
```

```
boston.res.net <- resid(net.mod)
plot(net.pred, boston.res.net, ylab = "Ridge Residuals", xlab = "Predicted medv", main = "Residuals vs predicted y
```



The Elastic Net held similar results to the lasso. The optimal lambda, after Cross-Validation, 0.02341795. After the prediction, the RMSE/test error was 5.181197.

Regression Trees

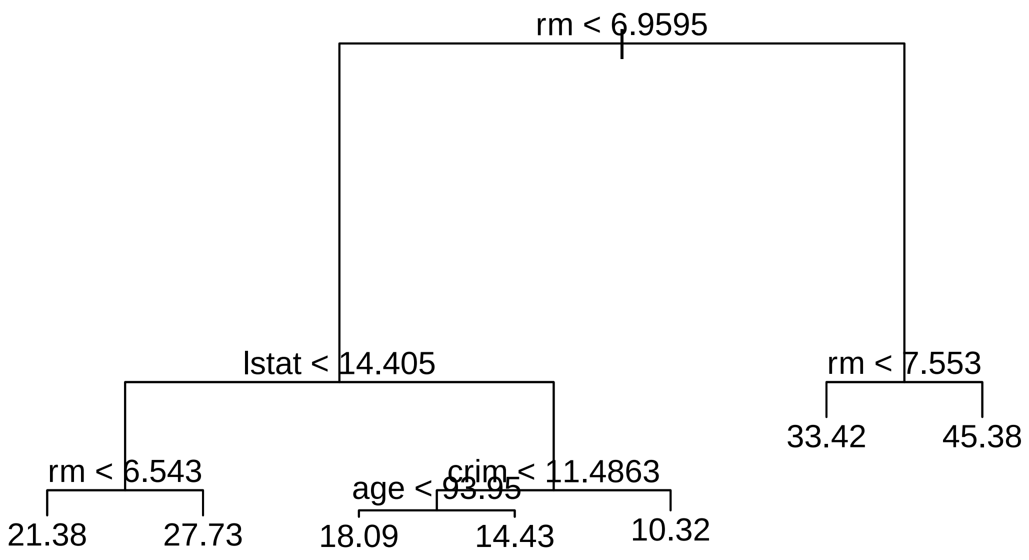
```
library(tree)
```

```
## Warning: package 'tree' was built under R version 4.1.2
```

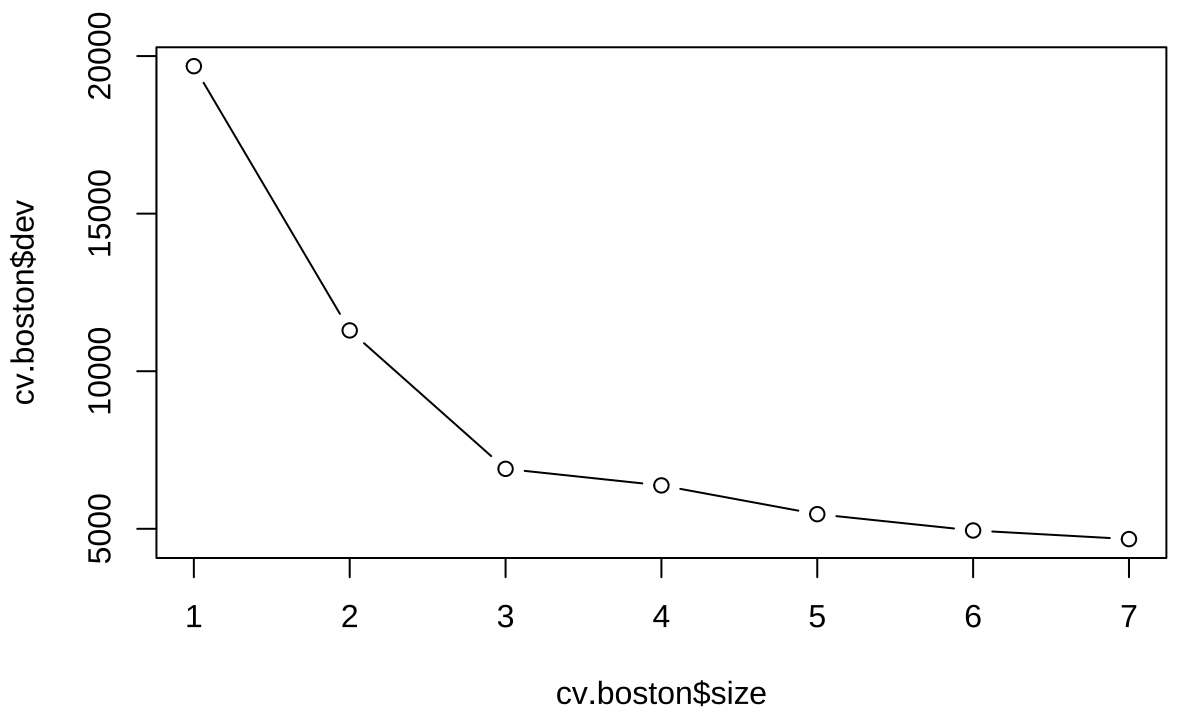
```
set.seed(2)
tree.medv <- tree(medv~., BostonHousing, subset = train)
summary(tree.medv)
```

```
##
## Regression tree:
## tree(formula = medv ~ ., data = BostonHousing, subset = train)
## Variables actually used in tree construction:
## [1] "rm"    "lstat" "crim"  "age"
## Number of terminal nodes:  7
## Residual mean deviance:  10.38 = 2555 / 246
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -10.1800  -1.7770  -0.1775   0.0000   1.9230  16.5800
```

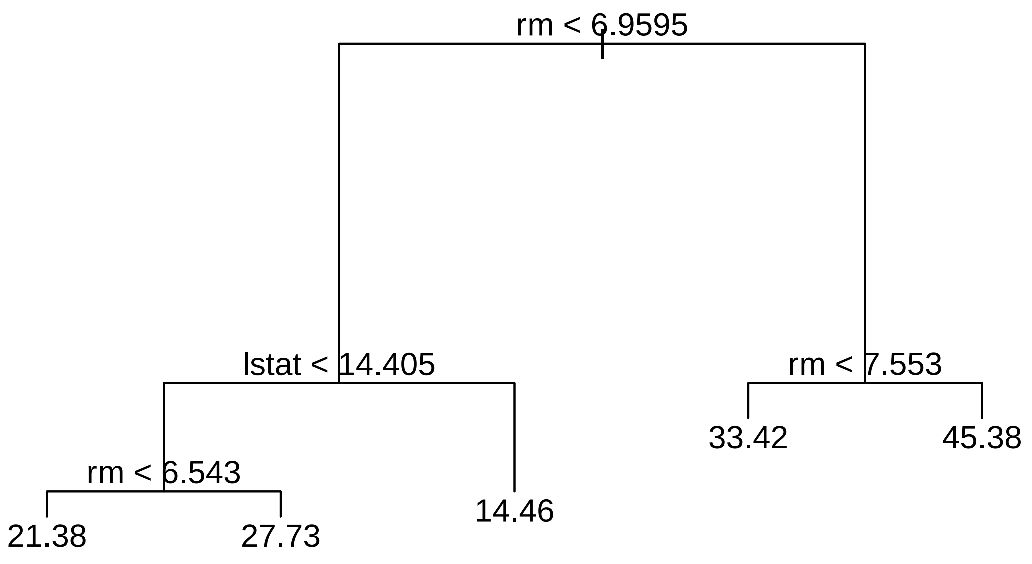
```
library(tree)
plot(tree.medv)
text(tree.medv, pretty = 0)
```



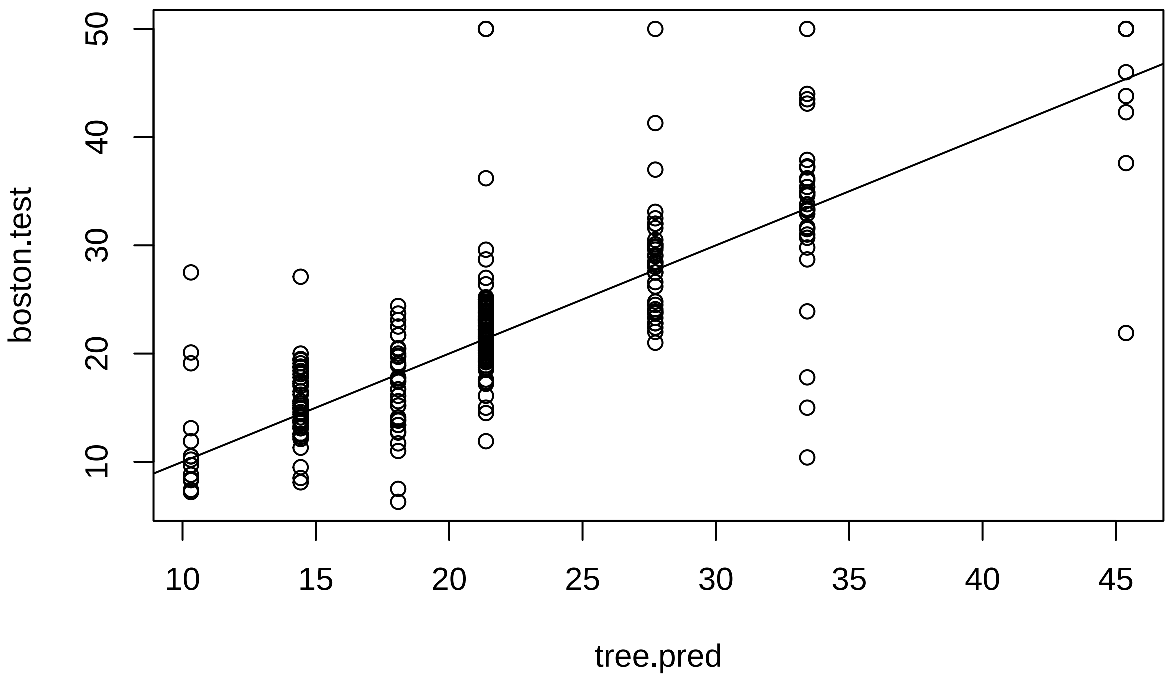
```
library(tree)
cv.boston <- cv.tree(tree.medv)
plot(cv.boston$size, cv.boston$dev, type = "b")
```



```
library(tree)
prune.boston <- prune.tree(tree.medv, best = 5)
plot(prune.boston)
text(prune.boston, pretty = 0)
```



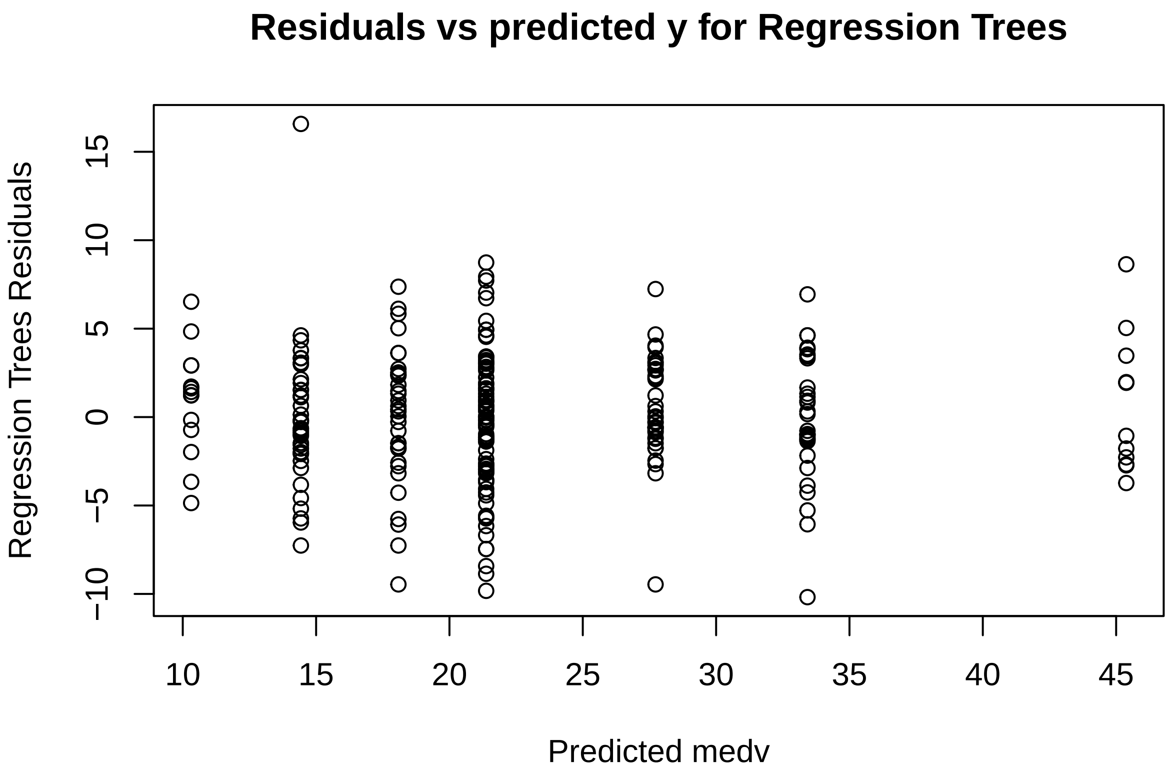
```
library(tree)
tree.pred <- predict(tree.medv, newdata = BostonHousing[-train, ])
boston.test <- BostonHousing[-train, "medv"]
plot(tree.pred, boston.test)
abline(0, 1)
```



```
sqrt(mean((tree.pred - boston.test)^2))
```

```
## [1] 5.940276
```

```
library(tree)
boston.res.tree <- resid(prune.boston)
plot(tree.pred, boston.res.tree, ylab = "Regression Trees Residuals", xlab = "Predicted medv", main = "Residuals vs
```



We can see from the regression tree, that the number of rooms has the most significance as it is the first branch. The number of rooms also appear in the 3rd branch on the LHS and on the 2nd branch on the RHS. This emphasizes the importance of number of rooms. After the pruning of the tree, the RMSE/test error is 5.940276.

Bagging

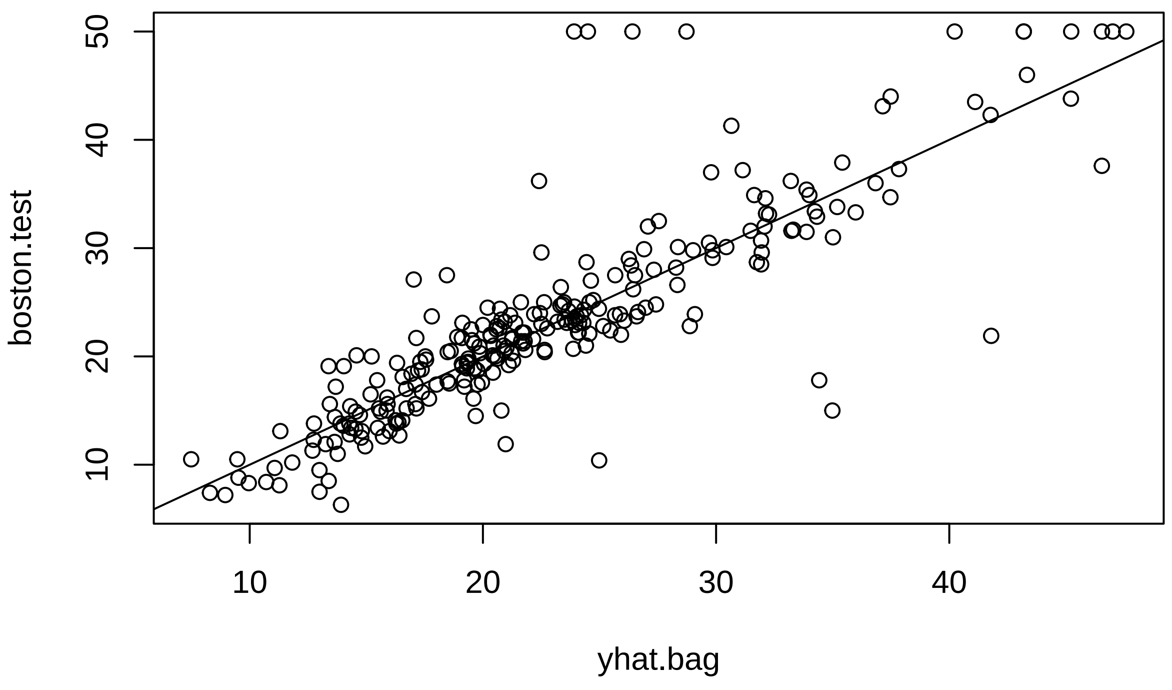
```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.1.2

## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
set.seed(129)
bag.boston <- randomForest(medv~., data = BostonHousing,subset = train, mtry = 13, importance = TRUE)
bag.boston
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = BostonHousing, mtry = 13,      importance = TRUE, subset = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 13
##
##           Mean of squared residuals: 11.17811
##                     % Var explained: 85.46
```

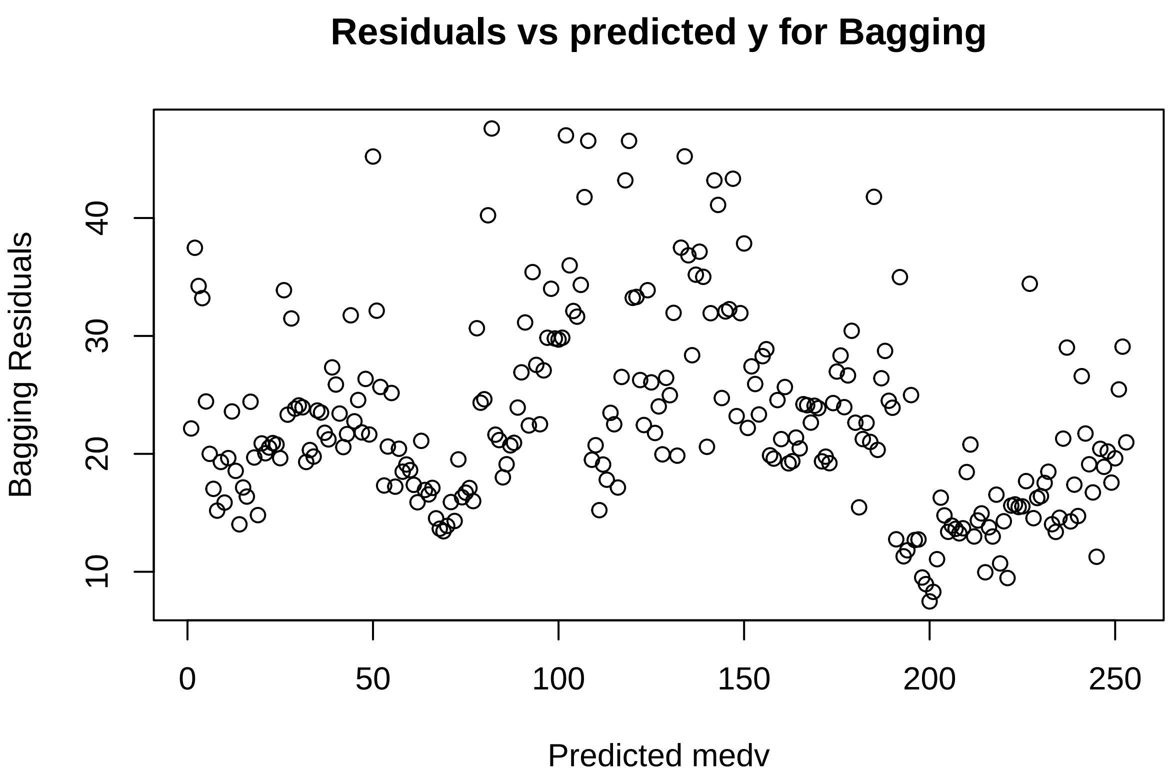
```
yhat.bag <- predict(bag.boston, newdata = BostonHousing[-train, ])
plot(yhat.bag, boston.test)
abline(0, 1)
```



```
sqrt(mean((yhat.bag - boston.test)^2))
```

```
## [1] 4.843261
```

```
boston.res.bag <- resid(bag.boston)
plot(yhat.bag, boston.res.bag, ylab = "Bagging Residuals", xlab = "Predicted medv", main = "Residuals vs predicted
```



The number of trees we chose to use for this bagging model was 500 trees. We can also see from the graph that the test data is postiviely correlated with the predicted values using the bagging model. The RMSE/test error here is 4.843261.

Random Forest

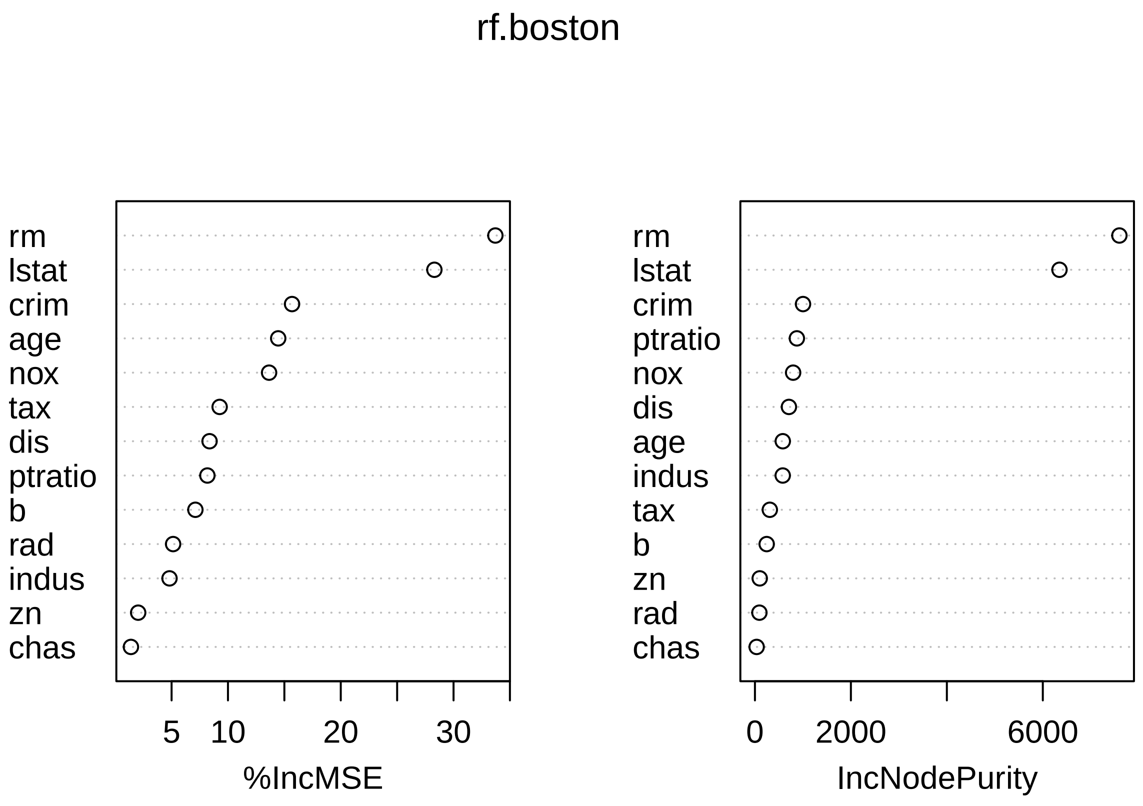
```r
#Random Forest

set.seed(129)
rf.boston <- randomForest(medv~., data = BostonHousing, subset = train, mtry = 6, importance = TRUE)
yhat.rf <- predict(rf.boston, newdata = BostonHousing[-train, ])
sqrt(mean((yhat.rf - boston.test)^2))
```

```
## [1] 4.431934
```

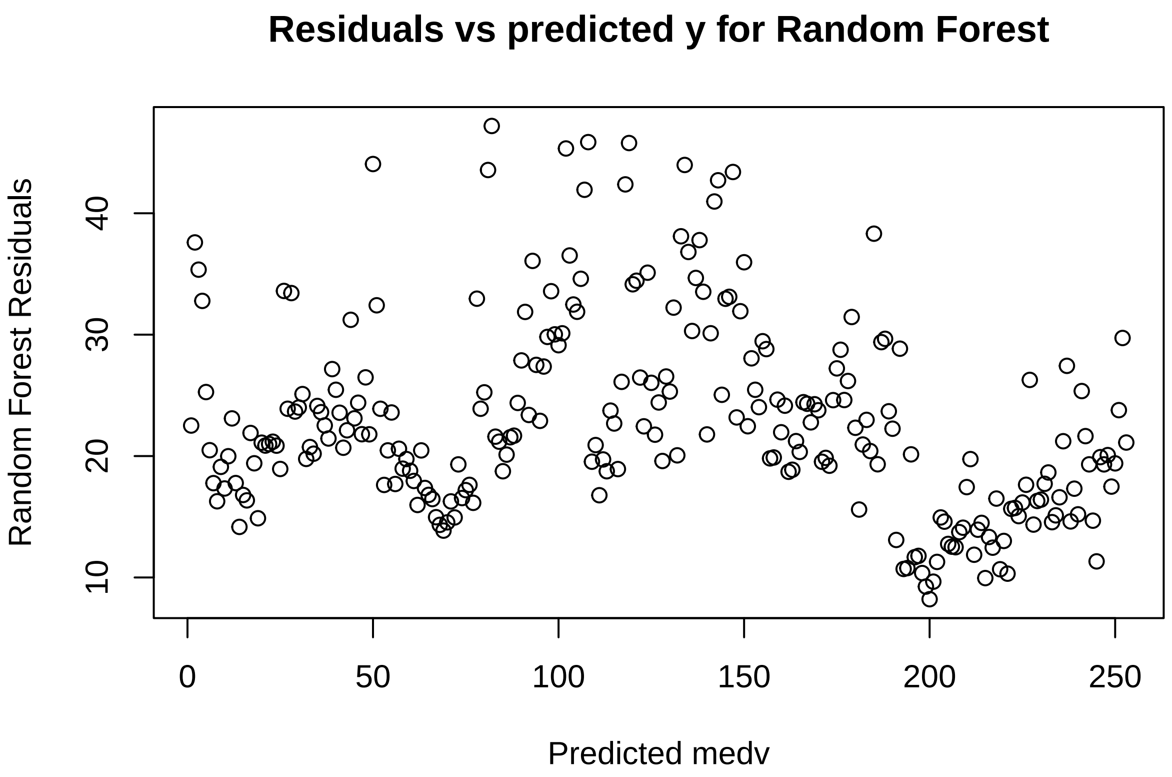
```r
importance(rf.boston)
```

```
##            %IncMSE IncNodePurity
## crim    15.680151    1002.18615
## zn       2.038581     100.14756
## indus    4.816118     579.41485
## chas     1.393545      35.65192
## nox     13.644160     796.17221
## rm      33.708820    7595.74968
## age     14.452588     580.59996
## dis      8.367186     708.38057
## rad      5.135045      93.05800
## tax      9.261585     310.11639
## ptratio  8.174518     874.14120
## b        7.108551     245.33588
## lstat   28.300810    6347.17412
```

```r
varImpPlot(rf.boston)
```



```r
boston.res.rf <- resid(rf.boston)
plot(yhat.rf, boston.res.rf, ylab = "Random Forest Residuals", xlab = "Predicted medv", main = "Residuals vs predic
```



Using the random forest model, the analysis showed that the number of rooms has the most importance in the model; indicated by the table. The RMSE/test error is 4.431934 for this model.

Gradient Boosting

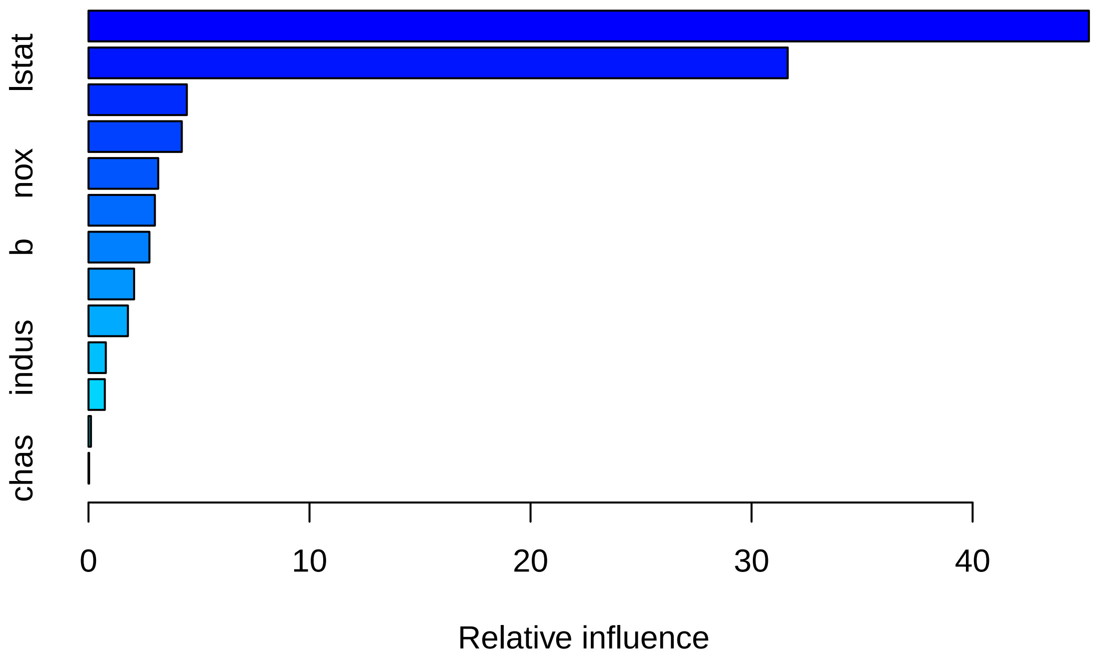
```
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 4.1.2

## Loaded gbm 2.1.8.1
```

```
set.seed(234)
Boston.boost=gbm(medv ~ . ,data = BostonHousing[train,],distribution = "gaussian",n.trees = 10000, shrinkage = 0.01
Boston.boost
```

```
## gbm(formula = medv ~ ., distribution = "gaussian", data = BostonHousing[train,
##     ], n.trees = 10000, interaction.depth = 4, shrinkage = 0.01)
## A gradient boosted model with gaussian loss function.
## 10000 iterations were performed.
## There were 13 predictors of which 13 had non-zero influence.
```

```
summary(Boston.boost) #Summary gives a table of Variable Importance and a plot of Variable Importance
```

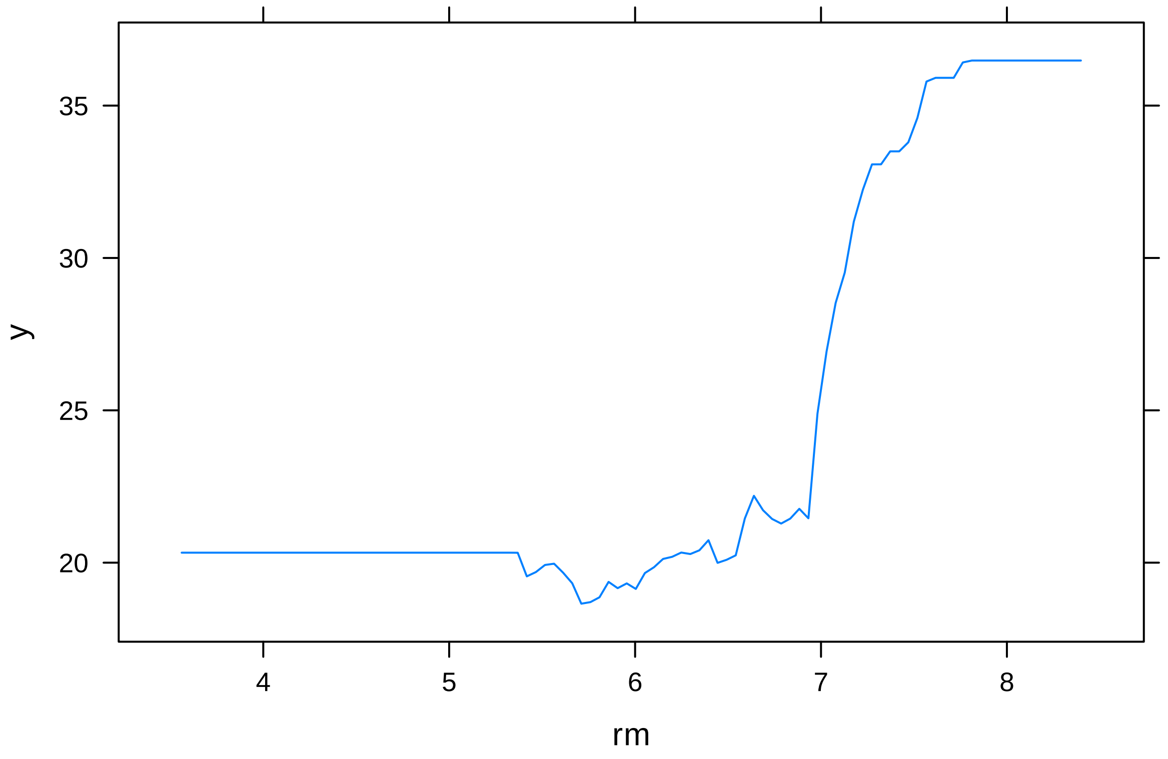


```
##              var     rel.inf
## rm            rm 45.26201716
## lstat      lstat 31.63474335
## crim        crim  4.45045942
## dis          dis  4.22272916
## nox          nox  3.15577939
## age          age  3.00430584
## b              b  2.75674821
## ptratio ptratio  2.06457715
## tax          tax  1.78362035
## indus      indus  0.78437776
## rad          rad  0.73985783
## zn            zn  0.11203421
## chas        chas  0.02875016
```

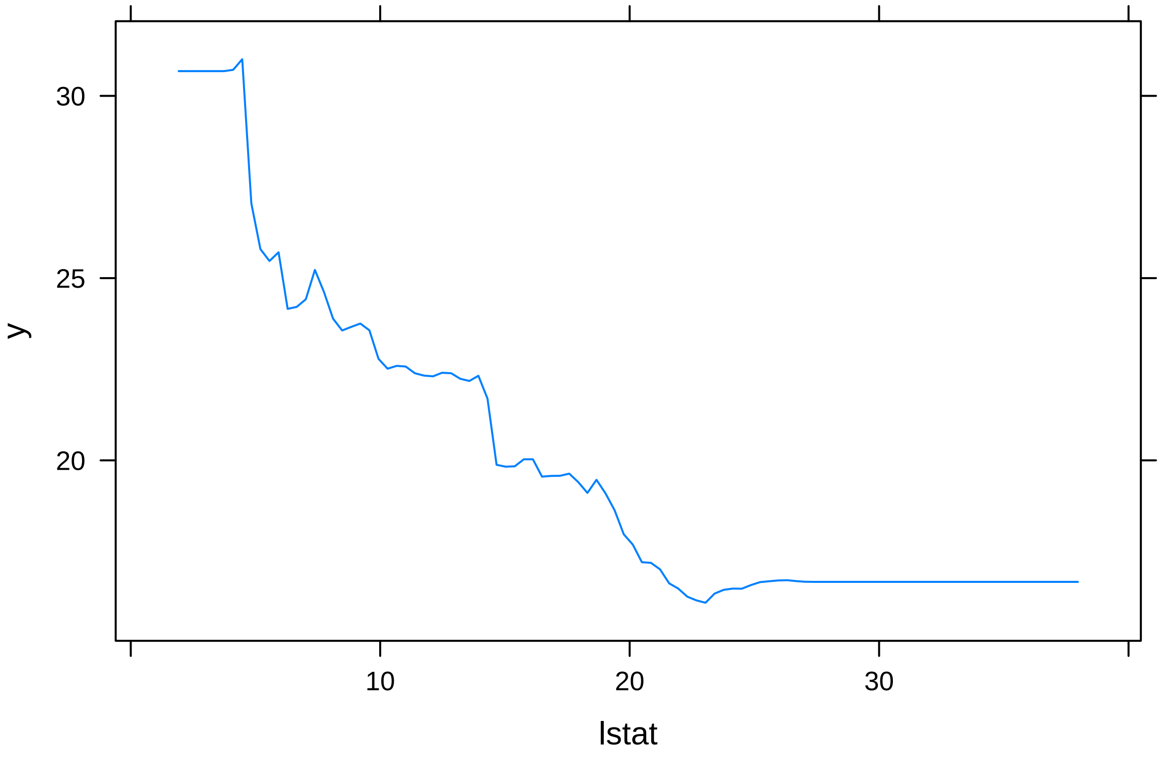
```r
gbm(formula = medv~., distribution = "gaussian", data = BostonHousing[-train, ], n.trees = 10000, interaction.depth
```

```
## gbm(formula = medv ~ ., distribution = "gaussian", data = BostonHousing[-train,
##     ], n.trees = 10000, interaction.depth = 4, shrinkage = 0.01)
## A gradient boosted model with gaussian loss function.
## 10000 iterations were performed.
## There were 13 predictors of which 13 had non-zero influence.
```

```r
plot(Boston.boost, i = "rm")
```



```r
plot(Boston.boost, i = "lstat")
```



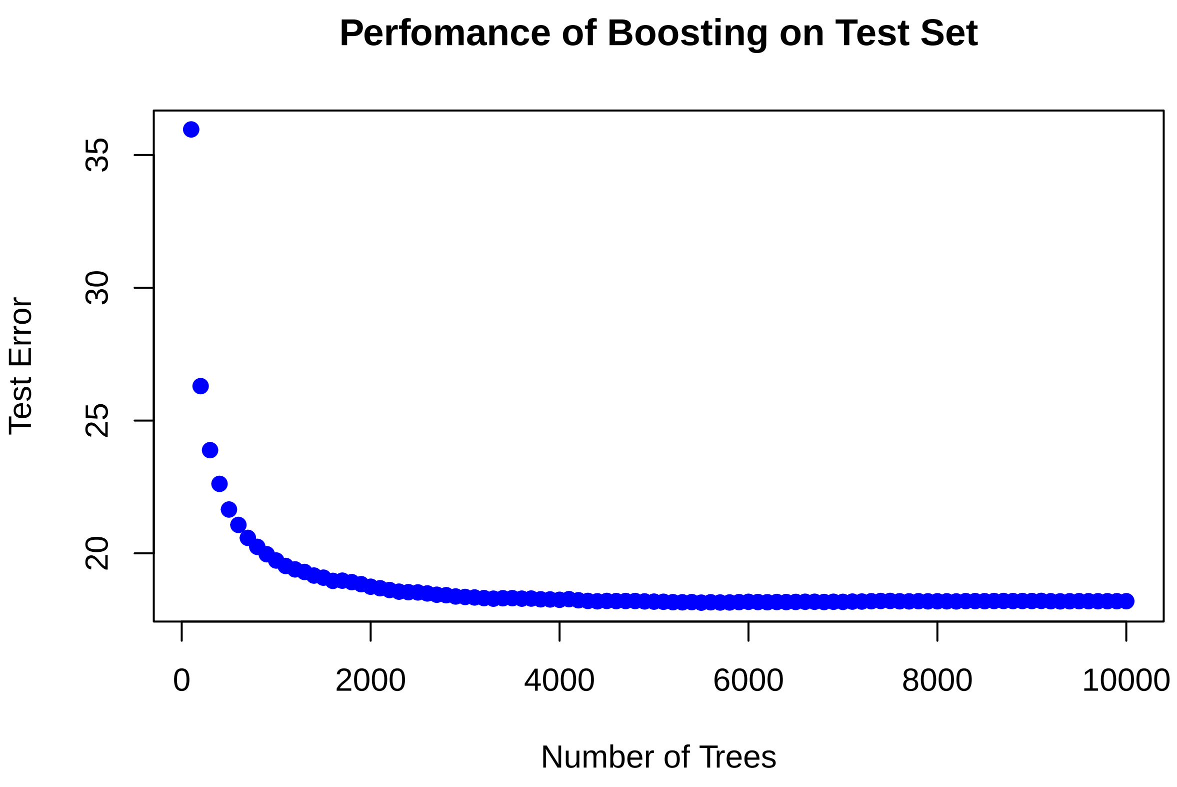
```
n.trees = seq(from=100 ,to=10000, by=100) #no of trees-a vector of 100 values
#Generating a Prediction matrix for each Tree
predmatrix<-predict(Boston.boost,BostonHousing[-train,],n.trees = n.trees)
dim(predmatrix) #dimentions of the Prediction Matrix
```

```
## [1] 253 100
```

```
#Calculating The Mean squared Test Error
test.error<-with(BostonHousing[-train,],apply( (predmatrix-medv)^2,2,mean))
head(test.error) #contains the Mean squared test error for each of the 100 trees averaged
```

```
##      100      200      300      400      500      600
## 35.96346 26.29789 23.88685 22.61696 21.65165 21.07112
```

```
#Plotting the test error vs number of trees
plot(n.trees , test.error , pch=19,col="blue",xlab="Number of Trees",ylab="Test Error", main = "Perfomance of Boost
```



```
dim(predmatrix)
```

```
## [1] 253 100
```

```
head(test.error)
```

```
##      100      200      300      400      500      600
## 35.96346 26.29789 23.88685 22.61696 21.65165 21.07112
```

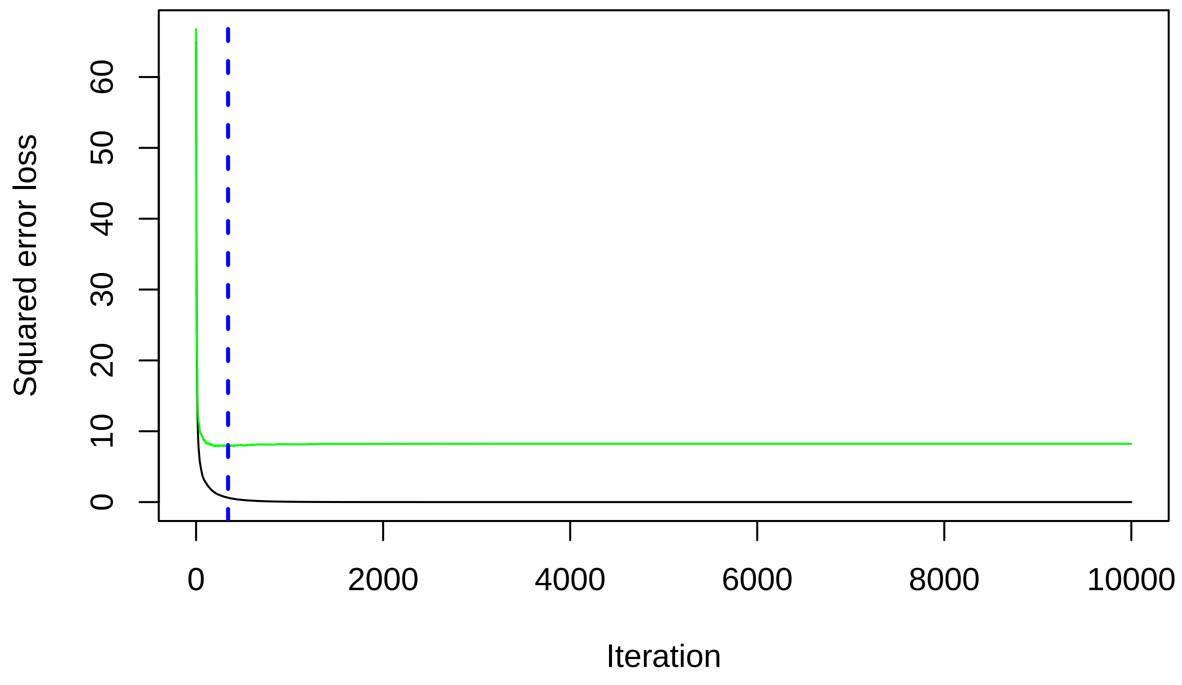
```
which.min(test.error)
```

```
## 5500
##   55
```

```
sqrt(test.error[which.min(test.error)])
```

```
##     5500
## 4.259665
```

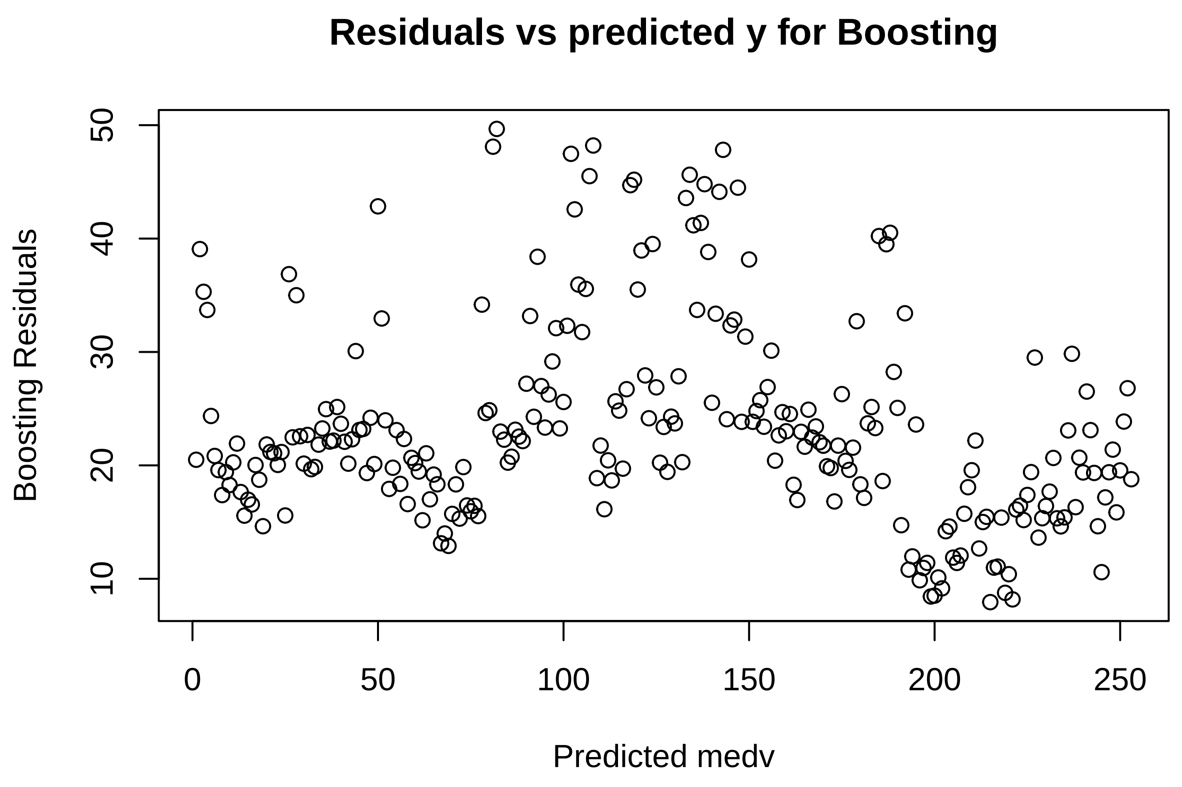
```
boston.boost.cv <- gbm(medv~., data = BostonHousing[train,], distribution = "gaussian", n.trees=10000, interaction.

#find the best prediction
bestTreeForPrediction <- gbm.perf(boston.boost.cv)
```



```
yhat.boost = predict(boston.boost.cv, newdata = BostonHousing[-train,],n.trees = bestTreeForPrediction)
sqrt((mean((yhat.boost-boston.test)^2)))
```

```
## [1] 4.335797
```

```
boston.res.boost <- resid(Boston.boost)
plot(yhat.boost, boston.res.boost, ylab = "Boosting Residuals", xlab = "Predicted medv", main = "Residuals vs predi
```



The boosting model showed us once again that the number of rooms has the most importance in the model. The graph showed us that the error after around 3000 trees levels out.The analysis found that the optimal number of trees was 5500, with an RMSE/test error of 4.259665. The best prediction, however had an RMSE/test error of 4.335797.

XGBoost

```
#XGboost
library(xgboost)
```

```
## Warning: package 'xgboost' was built under R version 4.1.2
```

```
train.boston <- BostonHousing[train,-4]
test.boston <- BostonHousing[-train,-4]

dtrain <- xgb.DMatrix(data = as.matrix(train.boston[!names(train.boston) %in% c("medv")]), label = train.boston$med

boston.xgb = xgboost(data=dtrain, max_depth=3, eta = 0.2, nthread=3, nrounds=40, lambda=0
, objective="reg:linear")
```

```
## [19:04:02] WARNING: amalgamation/../src/objective/regression_obj.cu:203: reg:linear is now deprecated in favor c
## [1]  train-rmse:18.506101
## [2]  train-rmse:14.911403
## [3]  train-rmse:12.052529
## [4]  train-rmse:9.780349
## [5]  train-rmse:7.978850
## [6]  train-rmse:6.551236
## [7]  train-rmse:5.418784
## [8]  train-rmse:4.546455
## [9]  train-rmse:3.877830
## [10] train-rmse:3.359612
## [11] train-rmse:2.963477
## [12] train-rmse:2.659171
## [13] train-rmse:2.424763
## [14] train-rmse:2.245043
## [15] train-rmse:2.107634
## [16] train-rmse:1.981864
## [17] train-rmse:1.904819
## [18] train-rmse:1.813042
## [19] train-rmse:1.741383
## [20] train-rmse:1.689975
## [21] train-rmse:1.658761
## [22] train-rmse:1.588507
## [23] train-rmse:1.540747
## [24] train-rmse:1.491813
## [25] train-rmse:1.453644
## [26] train-rmse:1.418362
## [27] train-rmse:1.377274
## [28] train-rmse:1.340356
## [29] train-rmse:1.330193
## [30] train-rmse:1.292565
## [31] train-rmse:1.263948
## [32] train-rmse:1.241936
## [33] train-rmse:1.214943
## [34] train-rmse:1.175159
## [35] train-rmse:1.168366
## [36] train-rmse:1.133257
## [37] train-rmse:1.116006
## [38] train-rmse:1.096613
## [39] train-rmse:1.073980
## [40] train-rmse:1.052533
```

```
dtest <- as.matrix(test.boston[!names(train.boston) %in% c("medv")])
yhat.xgb <- predict(boston.xgb,dtest)
sqrt(mean((yhat.xgb - boston.test)^2))
```

```
## [1] 4.58367
```

```
set.seed(42)
param <- list("max_depth" = 3, "eta" = 0.2, "objective" = "reg:linear", "lambda" = 0)
cv.nround <- 500
cv.nfold <- 3
boston.xgb.cv <- xgb.cv(param=param, data = dtrain, nfold = cv.nfold, nrounds=cv.nround,
                        early_stopping_rounds = 200, verbose=0)
```

```
## [19:04:02] WARNING: amalgamation/../src/objective/regression_obj.cu:203: reg:linear is now deprecated in favor o
## [19:04:02] WARNING: amalgamation/../src/objective/regression_obj.cu:203: reg:linear is now deprecated in favor o
## [19:04:02] WARNING: amalgamation/../src/objective/regression_obj.cu:203: reg:linear is now deprecated in favor o
```

```
dtrain <- xgb.DMatrix(data = as.matrix(train.boston[!names(train.boston) %in% c("medv")]), label = train.boston$med
boston.xgb = xgboost(param=param, data=dtrain, nthread=3, nrounds=boston.xgb.cv$best_iteration, verbose=0)
```

```
## [19:04:02] WARNING: amalgamation/../src/objective/regression_obj.cu:203: reg:linear is now deprecated in favor o
```

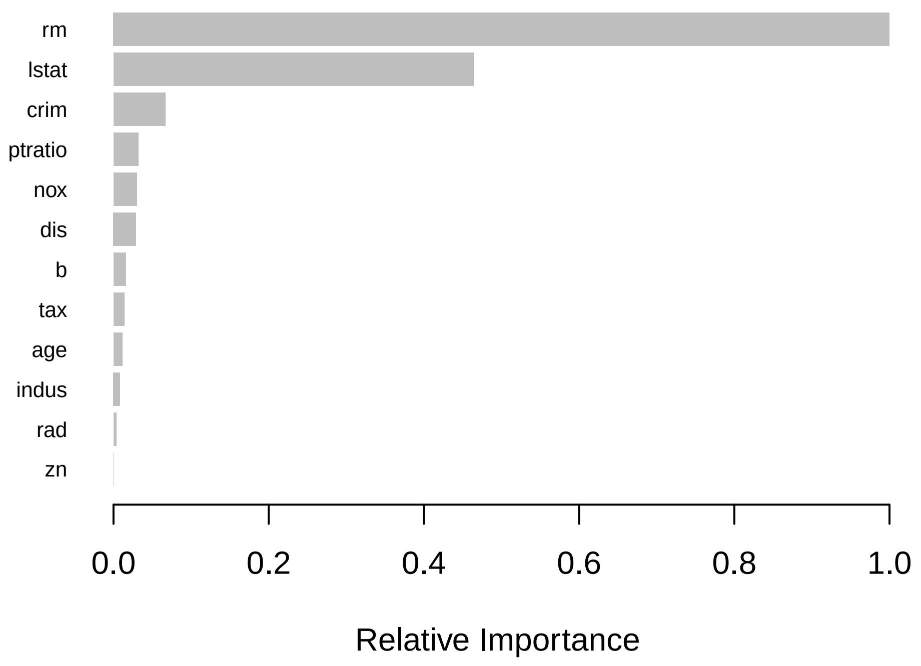
```
dtest <- as.matrix(test.boston[!names(train.boston) %in% c("medv")])
yhat.xgb <- predict(boston.xgb,dtest)
sqrt(mean((yhat.xgb - boston.test)^2))
```

```
## [1] 4.558034
```

```
importance <- xgb.importance(colnames(train.boston[!names(train.boston) %in% c("medv")]),model=boston.xgb)
importance
```

```
##     Feature         Gain        Cover   Frequency
##  1:      rm 0.5967002222 0.1916673830 0.176470588
##  2:   lstat 0.2767173652 0.1112066248 0.104575163
##  3:    crim 0.0399197791 0.1046734863 0.153594771
##  4: ptratio 0.0191010918 0.0559328348 0.049019608
##  5:     nox 0.0180010976 0.0484541104 0.062091503
##  6:     dis 0.0173213194 0.1365655176 0.124183007
##  7:       b 0.0096052809 0.0857617697 0.088235294
##  8:     tax 0.0082896593 0.0783976618 0.065359477
##  9:     age 0.0068412229 0.0999742113 0.101307190
## 10:   indus 0.0050994161 0.0623800109 0.052287582
## 11:     rad 0.0022817009 0.0248144646 0.019607843
## 12:      zn 0.0001218446 0.0001719247 0.003267974
```

```
xgb.plot.importance(importance, rel_to_first=TRUE, xlab="Relative Importance")
```



```
library(caret)
```

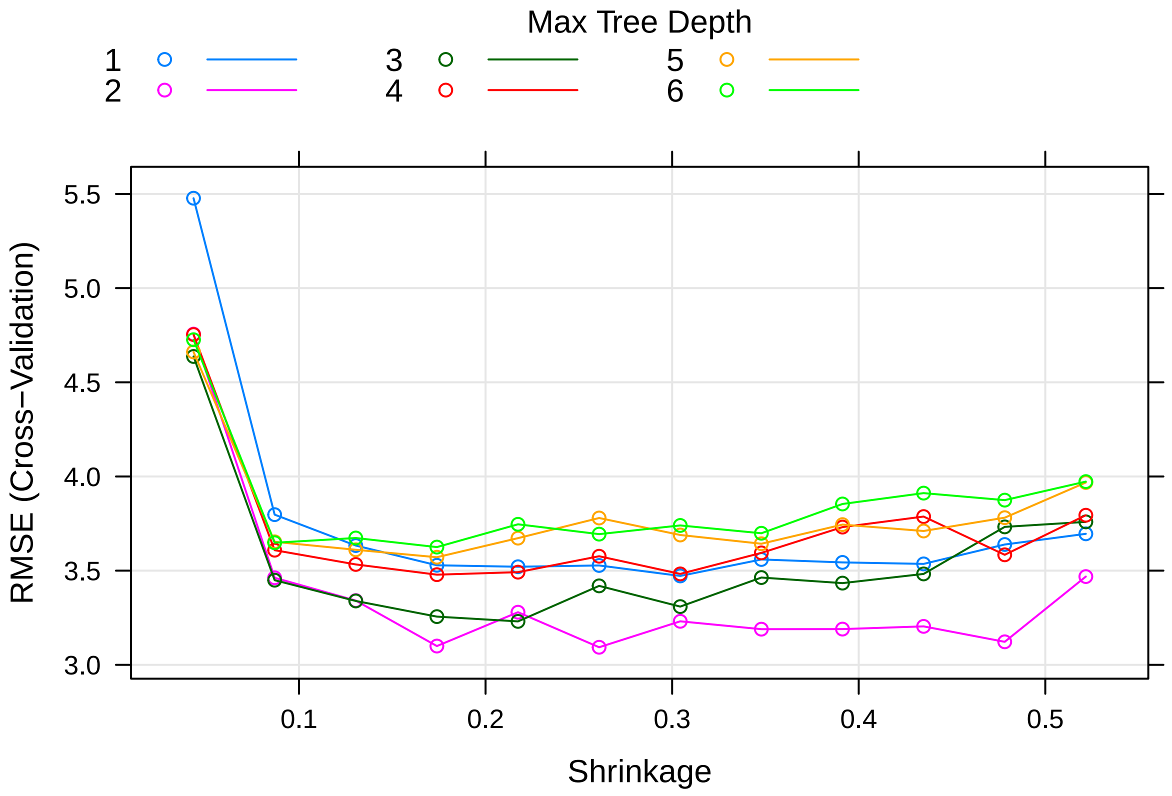
```
## Warning: package 'caret' was built under R version 4.1.2
```

```
## Loading required package: lattice
```

```
ntrees <- boston.xgb.cv$best_iteration
param_grid <- expand.grid(
  nrounds = ntrees,
  eta = seq(2,24,2)/ntrees,
  subsample = 1.0,
  colsample_bytree = 1.0,
  max_depth = c(1,2,3,4,5,6),
  gamma = 1,
  min_child_weight = 1
)

xgb_control <- trainControl(
  method="cv",
  number = 5
)
set.seed(42)
boston.xgb.tuned <- train(medv~., data=train.boston, trControl=xgb_control, tuneGrid=param_grid,lambda=0, method="x

boston.xgb.tuned$bestTune
```

```
##    nrounds max_depth       eta gamma colsample_bytree min_child_weight
## 32      46         2 0.2608696     1                1                1
##    subsample
## 32         1
```

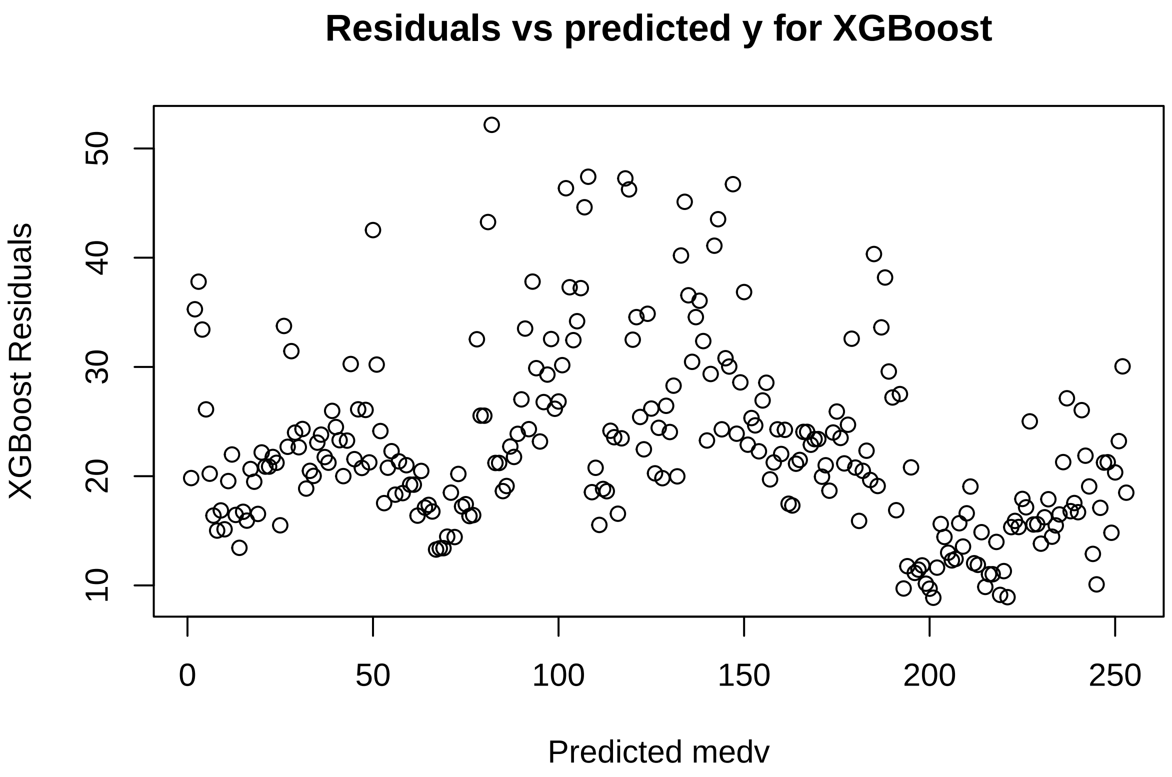
```
plot(boston.xgb.tuned)
```



```
yhat.xgb.tuned <- predict(boston.xgb.tuned$finalModel,newdata=dtest)
sqrt(mean((yhat.xgb.tuned - boston.test)^2))
```

```
## [1] 4.057732
```

```
boston.res.xgb <- resid(boston.xgb)
plot(yhat.xgb.tuned, boston.res.xgb , ylab = "XGBoost Residuals", xlab = "Predicted medv", main = "Residuals vs pre
```



Using Cross-Validation on the XGBoost we found the the optimal max tree depth was 2.This, once again, showed us that the number of rooms is the most important variable in the model. Using this parameter as well as the other optimal parameters, the model obtained an RMSE/test error of 4.057732. The other tree depths obtained had a worse RMSE, however these models, with a higher tree depth can be too complex, and so 2 is the best depth.

Conclusion From thorough analysis of the models we tested, we have determined that the XGBoost model is the most accurate for predicting the median house value in Boston. We also concluded that the number of rooms per household(rm) was the most influential and important predictor in our models and the proportion of residential land zoned for lots over 25000 square ft(zn) was the least imporatant predictor. This was a common theme throughout all our models. We can see that in all the residual vs predicated y(medv) graphs that the points are random, which indicates that the residuals are random. Therefore, our models do not have a patterns.