Untitled

```
library(tree)
library(ISLR2)
```

##########labs #8.3.1 Fitting Classifcation Trees

```
attach(Carseats)
High <- factor(ifelse(Sales <= 8, "No", "Yes"))
```

#merge High with the rest of the Carseats data.

```
Carseats <- data.frame(Carseats, High)
```

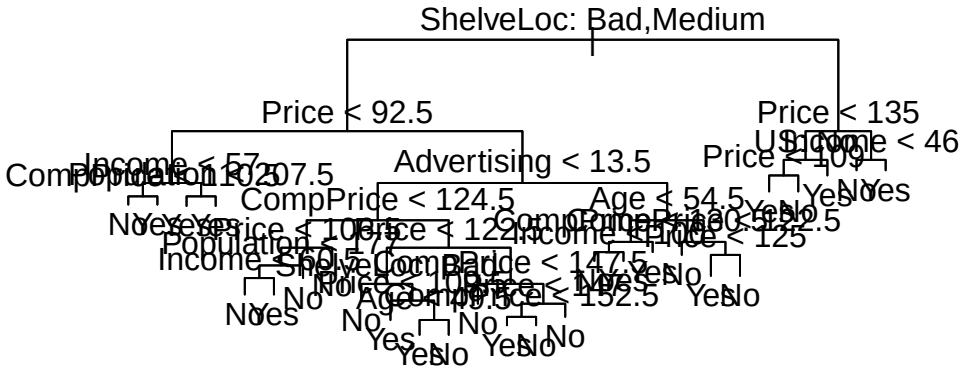
#tree() function to ft a classifcation tree in order to predict High using all variables but Sales

```
tree.carseats <- tree(High ~ . - Sales, Carseats)
summary(tree.carseats)
```

```
Classification tree:
tree(formula = High ~ . - Sales, data = Carseats)
Variables actually used in tree construction:
[1] "ShelveLoc"   "Price"       "Income"      "CompPrice"   "Population"
[6] "Advertising" "Age"         "US"
Number of terminal nodes:  27
Residual mean deviance:  0.4575 = 170.7 / 373
Misclassification error rate: 0.09 = 36 / 400
```

#Misclassification error rate (training error) = 0.09

```
plot(tree.carseats)
text(tree.carseats, pretty = 0)
```




```
tree.carseats
```

```
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

  1) root 400 541.500 No ( 0.59000 0.41000 )
    2) ShelveLoc: Bad,Medium 315 390.600 No ( 0.68889 0.31111 )
      4) Price < 92.5 46  56.530 Yes ( 0.30435 0.69565 )
        8) Income < 57 10  12.220 No ( 0.70000 0.30000 )
         16) CompPrice < 110.5 5   0.000 No ( 1.00000 0.00000 ) *
         17) CompPrice > 110.5 5   6.730 Yes ( 0.40000 0.60000 ) *
        9) Income > 57 36  35.470 Yes ( 0.19444 0.80556 )
         18) Population < 207.5 16  21.170 Yes ( 0.37500 0.62500 ) *
         19) Population > 207.5 20   7.941 Yes ( 0.05000 0.95000 ) *
      5) Price > 92.5 269 299.800 No ( 0.75465 0.24535 )
       10) Advertising < 13.5 224 213.200 No ( 0.81696 0.18304 )
         20) CompPrice < 124.5 96  44.890 No ( 0.93750 0.06250 )
           40) Price < 106.5 38  33.150 No ( 0.84211 0.15789 )
             80) Population < 177 12  16.300 No ( 0.58333 0.41667 )
              160) Income < 60.5 6   0.000 No ( 1.00000 0.00000 ) *
              161) Income > 60.5 6   5.407 Yes ( 0.16667 0.83333 ) *
             81) Population > 177 26   8.477 No ( 0.96154 0.03846 ) *
           41) Price > 106.5 58   0.000 No ( 1.00000 0.00000 ) *
         21) CompPrice > 124.5 128 150.200 No ( 0.72656 0.27344 )
           42) Price < 122.5 51  70.680 Yes ( 0.49020 0.50980 )
             84) ShelveLoc: Bad 11   6.702 No ( 0.90909 0.09091 ) *
             85) ShelveLoc: Medium 40  52.930 Yes ( 0.37500 0.62500 )
```

```
            170) Price < 109.5 16   7.481 Yes ( 0.06250 0.93750 ) *
            171) Price > 109.5 24  32.600 No ( 0.58333 0.41667 )
              342) Age < 49.5 13  16.050 Yes ( 0.30769 0.69231 ) *
              343) Age > 49.5 11   6.702 No ( 0.90909 0.09091 ) *
          43) Price > 122.5 77  55.540 No ( 0.88312 0.11688 )
            86) CompPrice < 147.5 58  17.400 No ( 0.96552 0.03448 ) *
            87) CompPrice > 147.5 19  25.010 No ( 0.63158 0.36842 )
             174) Price < 147 12  16.300 Yes ( 0.41667 0.58333 )
               348) CompPrice < 152.5 7   5.742 Yes ( 0.14286 0.85714 ) *
               349) CompPrice > 152.5 5   5.004 No ( 0.80000 0.20000 ) *
             175) Price > 147 7   0.000 No ( 1.00000 0.00000 ) *
        11) Advertising > 13.5 45  61.830 Yes ( 0.44444 0.55556 )
          22) Age < 54.5 25  25.020 Yes ( 0.20000 0.80000 )
            44) CompPrice < 130.5 14  18.250 Yes ( 0.35714 0.64286 )
              88) Income < 100 9  12.370 No ( 0.55556 0.44444 ) *
              89) Income > 100 5   0.000 Yes ( 0.00000 1.00000 ) *
            45) CompPrice > 130.5 11   0.000 Yes ( 0.00000 1.00000 ) *
          23) Age > 54.5 20  22.490 No ( 0.75000 0.25000 )
            46) CompPrice < 122.5 10   0.000 No ( 1.00000 0.00000 ) *
            47) CompPrice > 122.5 10  13.860 No ( 0.50000 0.50000 )
              94) Price < 125 5   0.000 Yes ( 0.00000 1.00000 ) *
              95) Price > 125 5   0.000 No ( 1.00000 0.00000 ) *
      3) ShelveLoc: Good 85  90.330 Yes ( 0.22353 0.77647 )
        6) Price < 135 68  49.260 Yes ( 0.11765 0.88235 )
         12) US: No 17  22.070 Yes ( 0.35294 0.64706 )
           24) Price < 109 8   0.000 Yes ( 0.00000 1.00000 ) *
           25) Price > 109 9  11.460 No ( 0.66667 0.33333 ) *
         13) US: Yes 51  16.880 Yes ( 0.03922 0.96078 ) *
        7) Price > 135 17  22.070 No ( 0.64706 0.35294 )
         14) Income < 46 6   0.000 No ( 1.00000 0.00000 ) *
         15) Income > 46 11  15.160 Yes ( 0.45455 0.54545 ) *
```

```r
set.seed(2)
train <- sample(1:nrow(Carseats), 200)
Carseats.test <- Carseats[-train, ]
High.test <- High[-train]
tree.carseats <- tree(High ~ . - Sales, Carseats,
                      subset = train)
tree.pred <- predict(tree.carseats, Carseats.test,
                     type = "class")
table(tree.pred, High.test)
```

```
         High.test
tree.pred  No Yes
      No  104  33
      Yes  13  50
```

```
(104 + 50) / 200
```

```
[1] 0.77
```

#cv.tree() performs cross-validation in order to determine the optimal level of tree complexity
#use the argument FUN = prune.misclass in order to indicate that we want the classifcation error rate to guide the cross-validation and pruning process

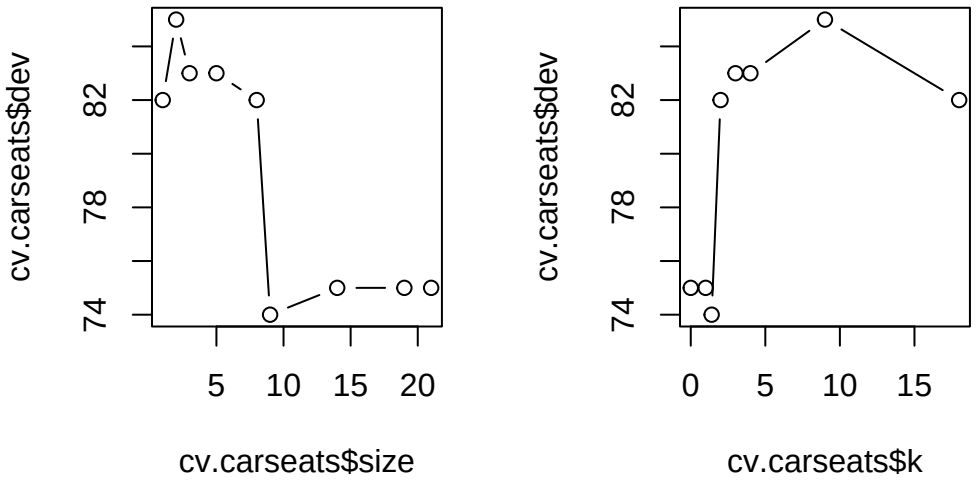
```
set.seed(7)
cv.carseats <- cv.tree(tree.carseats, FUN = prune.misclass)
names(cv.carseats)
```

```
[1] "size"   "dev"    "k"      "method"
```

```
cv.carseats
```

```
$size
[1] 21 19 14  9  8  5  3  2  1

$dev
[1] 75 75 75 74 82 83 83 85 82

$k
[1] -Inf  0.0  1.0  1.4  2.0  3.0  4.0  9.0 18.0

$method
[1] "misclass"

attr(,"class")
[1] "prune"         "tree.sequence"
```

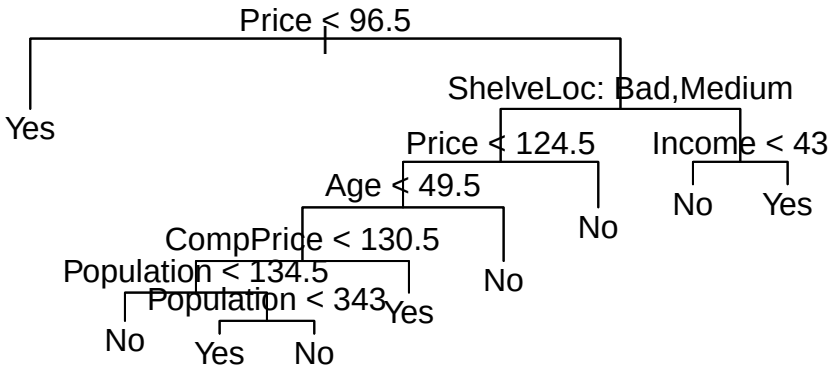
```
par(mfrow = c(1, 2))
plot(cv.carseats$size, cv.carseats$dev, type = "b")
plot(cv.carseats$k, cv.carseats$dev, type = "b")
```



#number of terminal nodes of each tree considered (size) # the value of the cost-complexity parameter used (k, which corresponds to

#now apply the prune.misclass() function in order to prune the tree to obtain the nine-node tree.

```
prune.carseats <- prune.misclass(tree.carseats, best = 9)
plot(prune.carseats)
text(prune.carseats, pretty = 0)
```




```
tree.pred <- predict(prune.carseats, Carseats.test,
                     type = "class")
table(tree.pred, High.test)
```

```
          High.test
tree.pred No Yes
      No  97  25
      Yes 20  58
```

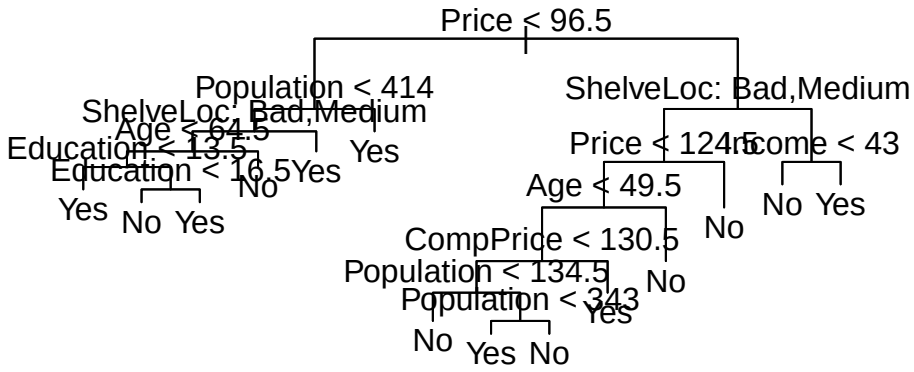
```
(97 + 58) / 200
```

```
[1] 0.775
```

77.5 % of the test observations are correctly classifed

#If we increase the value of best, we obtain a larger pruned tree with lower classifcation accuracy:

```
prune.carseats <- prune.misclass(tree.carseats, best = 14)
plot(prune.carseats)
text(prune.carseats, pretty = 0)
```




```
tree.pred <- predict(prune.carseats, Carseats.test,
                     type = "class")
table(tree.pred, High.test)
```

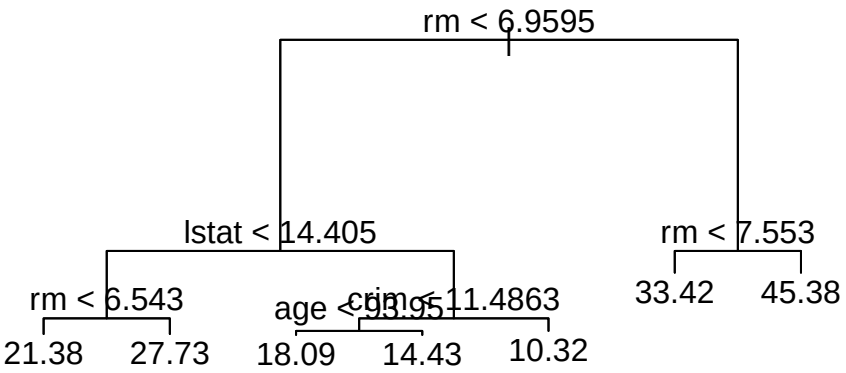
```
         High.test
tree.pred  No Yes
      No  102  31
      Yes  15  52
```

##########8.3.2 Fitting Regression Trees

```
set.seed(1)
train <- sample(1:nrow(Boston), nrow(Boston) / 2)
tree.boston <- tree(medv ~ ., Boston, subset = train)
summary(tree.boston)
```

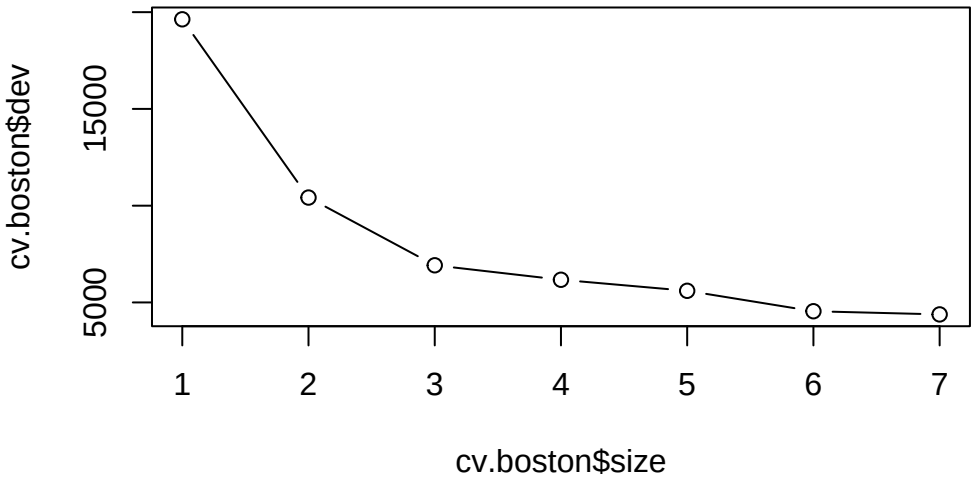
```
Regression tree:
tree(formula = medv ~ ., data = Boston, subset = train)
Variables actually used in tree construction:
[1] "rm"    "lstat" "crim"  "age"
Number of terminal nodes:  7
Residual mean deviance:  10.38 = 2555 / 246
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.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

```
plot(tree.boston)
text(tree.boston, pretty = 0)
```



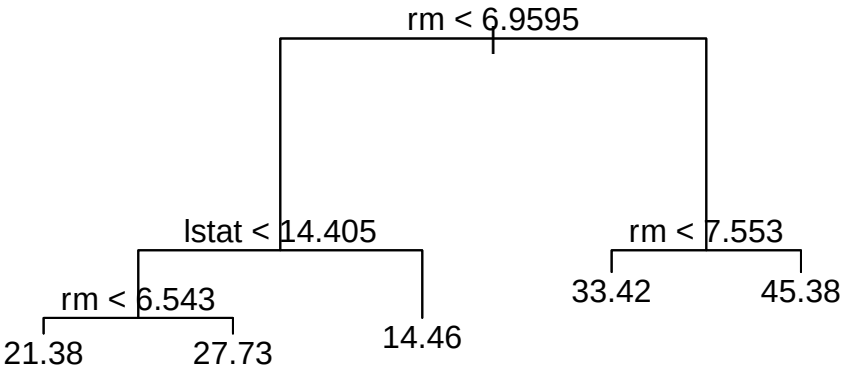
#cv.tree() function to see whether pruning the tree will improve performance.

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



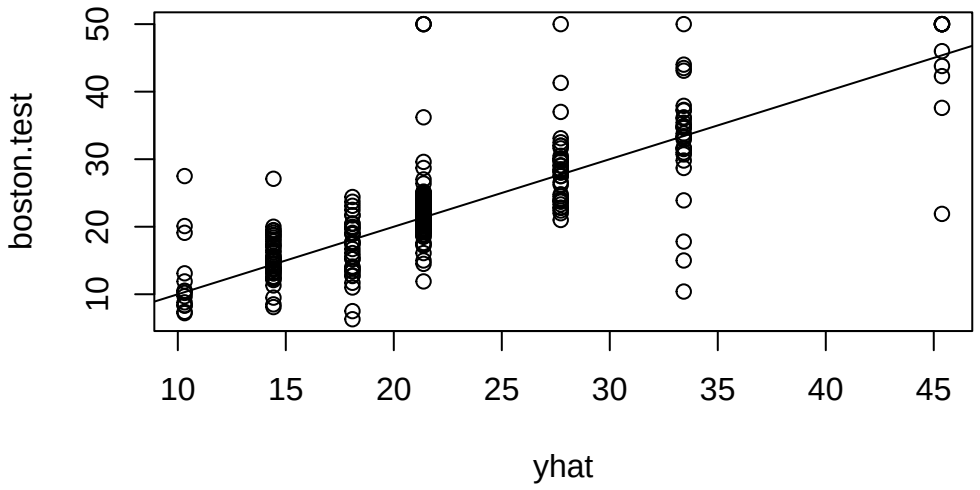
#In this case, the most complex tree under consideration is selected by cross validation. # However, if we wish to prune the tree, we could do so as follows, using the prune.tree() function:

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



#In keeping with the cross-validation results, we use the unpruned tree to make predictions on the test set.

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



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

```
[1] 35.28688
```

#In other words, the test set MSE associated with the regression tree is 35.29.The square root of the MSE is therefore around 5.941, indicating that this model leads to test predictions that are (on average) within approximately $5,941 of the true median home value for the census tract.

#8.3.3 Bagging and Random Forests ########Bagging

```
library(randomForest)
```

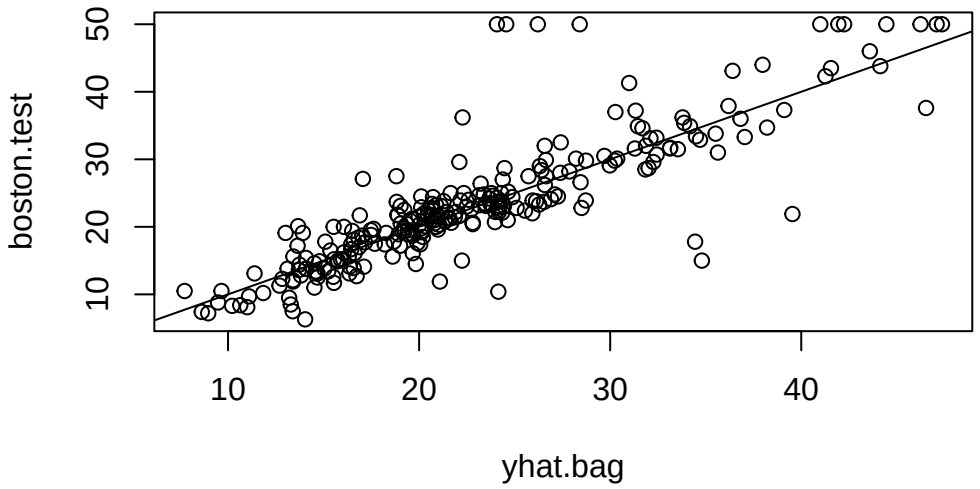
```
randomForest 4.7-1.2
```

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

```
set.seed(1)
bag.boston <- randomForest(medv ~ ., data = Boston,
                           subset = train, mtry = 12, #m = p Bagging
                           importance = TRUE)
bag.boston
```

```
Call:
 randomForest(formula = medv ~ ., data = Boston, mtry = 12, importance = TRUE,      subset =
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 12

          Mean of squared residuals: 11.40162
                    % Var explained: 85.17
```

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




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

```
[1] 23.41916
```

test set MSE associated with the bagged regression tree is 23.42

could change the number of trees grown by randomForest() using the ntree

```
bag.boston <- randomForest(medv ~ ., data = Boston,
                           subset = train, mtry = 12, ntree = 25)
yhat.bag <- predict(bag.boston, newdata = Boston[-train, ])
mean((yhat.bag - boston.test)^2)
```

```
[1] 25.75055
```

##########random forest #randomForest() uses p/3 variables when building a random forest of regression trees, and $\sqrt{p}$ variables when building a random forest of classifcation trees.

```
set.seed(1)
rf.boston <- randomForest(medv ~ ., data = Boston,
                          subset = train, mtry = 6, importance = TRUE)
yhat.rf <- predict(rf.boston, newdata = Boston[-train, ])
mean((yhat.rf - boston.test)^2)
```

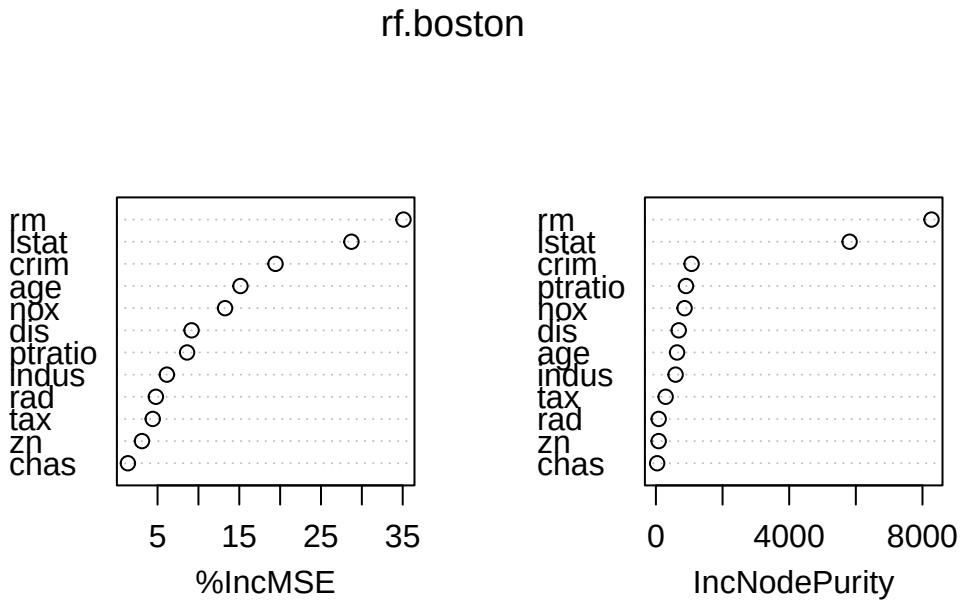
```
[1] 20.06644
```

test set MSE is 20.07; this indicates that random forests yielded an improvement over bagging in this case.

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

```
          %IncMSE IncNodePurity
crim    19.435587    1070.42307
zn       3.091630      82.19257
indus    6.140529     590.09536
chas     1.370310      36.70356
nox     13.263466     859.97091
rm      35.094741    8270.33906
age     15.144821     634.31220
dis      9.163776     684.87953
rad      4.793720      83.18719
tax      4.410714     292.20949
ptratio  8.612780     902.20190
lstat   28.725343    5813.04833
```

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



######8.3.4 Boosting

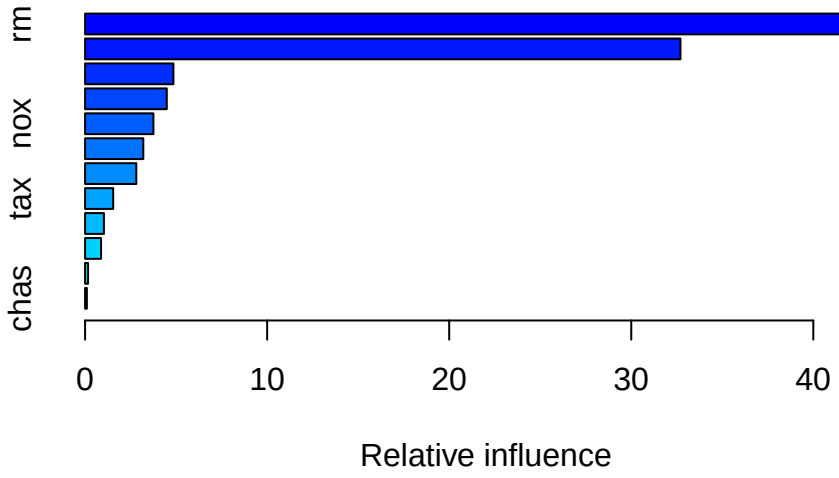
```
library(gbm)
```

```
Loaded gbm 2.2.2
```

```
This version of gbm is no longer under development. Consider transitioning to gbm3, https://g
```

```
set.seed(1)
boost.boston <- gbm(medv ~ ., data = Boston[train, ], # default value is 0.001,lamda
                    distribution = "gaussian", n.trees = 5000, #a binary classifcation proble
                    interaction.depth = 4)
```

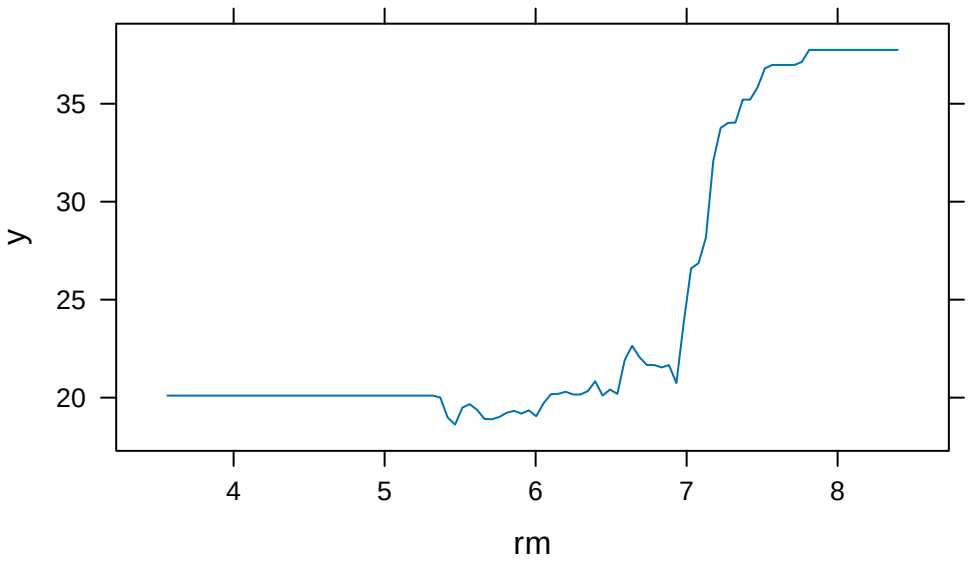
```
summary(boost.boston)
```



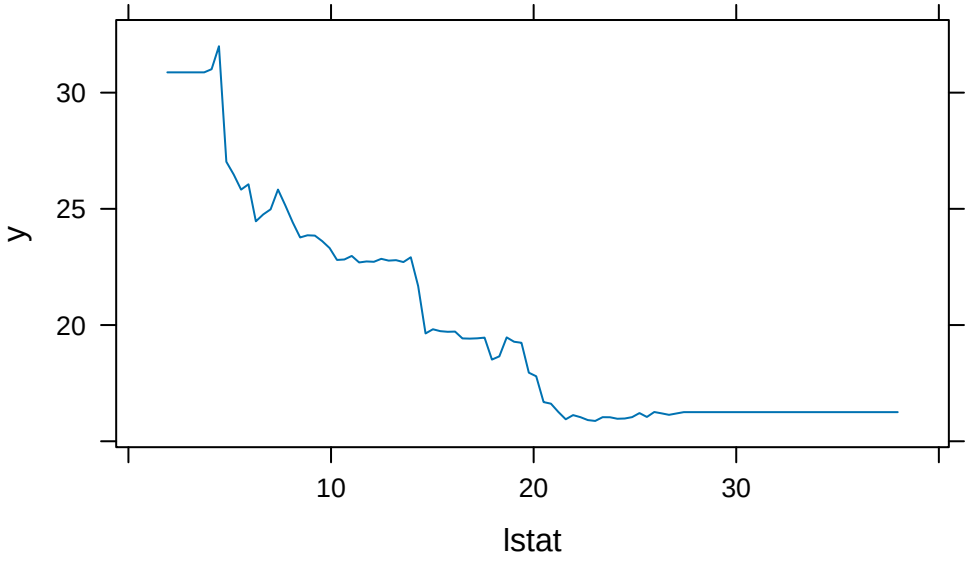

```
            var     rel.inf
rm           rm 44.48249588
lstat     lstat 32.70281223
crim       crim  4.85109954
dis         dis  4.48693083
nox         nox  3.75222394
age         age  3.19769210
ptratio ptratio  2.81354826
tax         tax  1.54417603
indus     indus  1.03384666
rad         rad  0.87625748
zn           zn  0.16220479
chas       chas  0.09671228
```

In this case, as we might expect, median house prices are increasing with rm and decreasing with lstat.

```
plot(boost.boston, i = "rm")
```




```
plot(boost.boston, i = "lstat")
```




```
yhat.boost <- predict(boost.boston,
                      newdata = Boston[-train, ], n.trees = 5000)
mean((yhat.boost - boston.test)^2)
```

```
[1] 18.39057
```

test MSE obtained is 18.39: this is superior to the test MSE of random forests and bagging.

```
boost.boston <- gbm(medv ~ ., data = Boston[train, ],
                    distribution = "gaussian", n.trees = 5000,
                    interaction.depth = 4, shrinkage = 0.2, verbose = F) #take   = 0.2.
yhat.boost <- predict(boost.boston,
                      newdata = Boston[-train, ], n.trees = 5000)
mean((yhat.boost - boston.test)^2)
```

```
[1] 16.54778
```

#8.3.5 Bayesian Additive Regression Trees

```
library(BART)
```

```
Loading required package: nlme
```

```
Loading required package: survival
```

```
x <- Boston[, 1:12]
y <- Boston[, "medv"]
xtrain <- x[train, ]
ytrain <- y[train]
xtest <- x[-train, ]
ytest <- y[-train]
set.seed(1)
bartfit <- gbart(xtrain, ytrain, x.test = xtest)
```

```
*****Calling gbart: type=1
*****Data:
data:n,p,np: 253, 12, 253
y1,yn: 0.213439, -5.486561
x1,x[n*p]: 0.109590, 20.080000
xp1,xp[np*p]: 0.027310, 7.880000
*****Number of Trees: 200
*****Number of Cut Points: 100 ... 100
*****burn,nd,thin: 100,1000,1
```

```
*****Prior:beta,alpha,tau,nu,lambda,offset: 2,0.95,0.795495,3,3.71636,21.7866
*****sigma: 4.367914
*****w (weights): 1.000000 ... 1.000000
*****Dirichlet:sparse,theta,omega,a,b,rho,augment: 0,0,1,0.5,1,12,0
*****printevery: 100

MCMC
done 0 (out of 1100)
done 100 (out of 1100)
done 200 (out of 1100)
done 300 (out of 1100)
done 400 (out of 1100)
done 500 (out of 1100)
done 600 (out of 1100)
done 700 (out of 1100)
done 800 (out of 1100)
done 900 (out of 1100)
done 1000 (out of 1100)
time: 2s
trcnt,tecnt: 1000,1000
```

compute the test error.

```
yhat.bart <- bartfit$yhat.test.mean
mean((ytest - yhat.bart)^2)
```

```
[1] 15.94718
```

#the test error of BART is lower than the test error of random forests and boosting

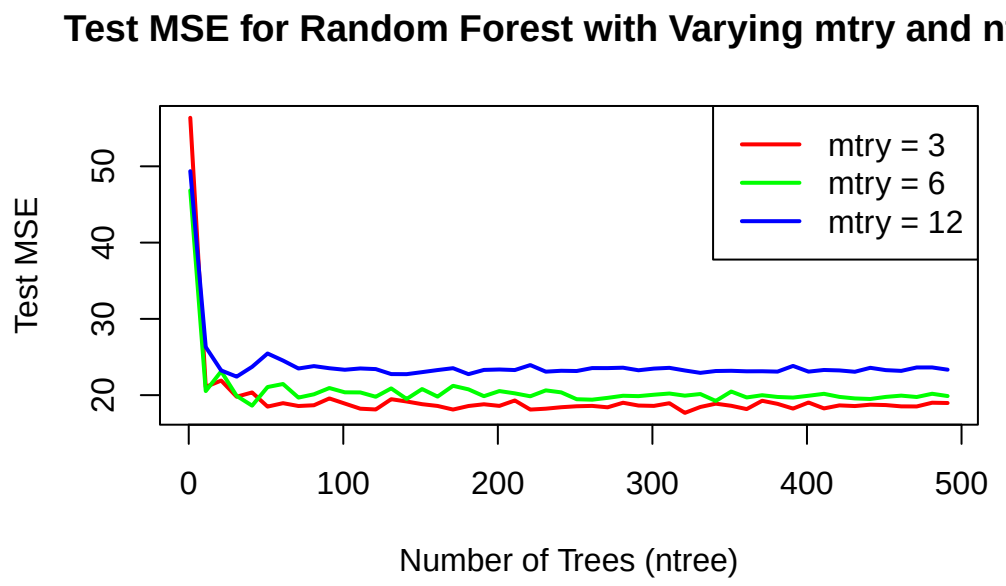
```
ord <- order(bartfit$varcount.mean, decreasing = T)
bartfit$varcount.mean[ord]
```

```
    nox   lstat     tax     rad      rm   indus    chas ptratio     age      zn
 22.952  21.329  21.250  20.781  19.890  19.825  19.051  18.976  18.274  15.952
    dis    crim
 14.457  11.007
```

#########exercises

#7.

```
mtry_values <- c(3, 6, 12)
ntree_values <- seq(1, 500, by = 10)
test_errors <- matrix(NA, nrow = length(ntree_values), ncol = length(mtry_values))

for (m in 1:length(mtry_values)) {
  for (t in 1:length(ntree_values)) {
    rf.boston <- randomForest(medv ~ ., data = Boston,
                              subset = train, mtry = mtry_values[m], ntree = ntree_values[t])
    yhat.rf <-predict(rf.boston, newdata = Boston[-train, ])

    test_errors[t, m] <- mean((yhat.rf - boston.test)^2)
  }
}

colors <- c("red", "green", "blue")
plot(ntree_values, test_errors[, 1], type = "l", col = colors[1], lwd = 2,
     ylim = range(test_errors), xlab = "Number of Trees (ntree)", ylab = "Test MSE",
     main = "Test MSE for Random Forest with Varying mtry and ntree")
lines(ntree_values, test_errors[, 2], col = colors[2], lwd = 2)
lines(ntree_values, test_errors[, 3], col = colors[3], lwd = 2)
legend("topright", legend = paste("mtry =", mtry_values), col = colors, lty = 1, lwd = 2)
```



#8.

```
detach(Carseats)
library(ISLR2)
data(Carseats)
attach(Carseats)
Carseats
```

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education
1	9.50	138	73	11	276	120	Bad	42	17
2	11.22	111	48	16	260	83	Good	65	10
3	10.06	113	35	10	269	80	Medium	59	12
4	7.40	117	100	4	466	97	Medium	55	14
5	4.15	141	64	3	340	128	Bad	38	13
6	10.81	124	113	13	501	72	Bad	78	16
7	6.63	115	105	0	45	108	Medium	71	15
8	11.85	136	81	15	425	120	Good	67	10
9	6.54	132	110	0	108	124	Medium	76	10
10	4.69	132	113	0	131	124	Medium	76	17
11	9.01	121	78	9	150	100	Bad	26	10
12	11.96	117	94	4	503	94	Good	50	13
13	3.98	122	35	2	393	136	Medium	62	18
14	10.96	115	28	11	29	86	Good	53	18
15	11.17	107	117	11	148	118	Good	52	18
16	8.71	149	95	5	400	144	Medium	76	18
17	7.58	118	32	0	284	110	Good	63	13
18	12.29	147	74	13	251	131	Good	52	10
19	13.91	110	110	0	408	68	Good	46	17
20	8.73	129	76	16	58	121	Medium	69	12
21	6.41	125	90	2	367	131	Medium	35	18
22	12.13	134	29	12	239	109	Good	62	18
23	5.08	128	46	6	497	138	Medium	42	13
24	5.87	121	31	0	292	109	Medium	79	10
25	10.14	145	119	16	294	113	Bad	42	12
26	14.90	139	32	0	176	82	Good	54	11
27	8.33	107	115	11	496	131	Good	50	11
28	5.27	98	118	0	19	107	Medium	64	17
29	2.99	103	74	0	359	97	Bad	55	11
30	7.81	104	99	15	226	102	Bad	58	17
31	13.55	125	94	0	447	89	Good	30	12
32	8.25	136	58	16	241	131	Medium	44	18
33	6.20	107	32	12	236	137	Good	64	10
34	8.77	114	38	13	317	128	Good	50	16
35	2.67	115	54	0	406	128	Medium	42	17

36	11.07	131	84	11	29	96	Medium	44	17
37	8.89	122	76	0	270	100	Good	60	18
38	4.95	121	41	5	412	110	Medium	54	10
39	6.59	109	73	0	454	102	Medium	65	15
40	3.24	130	60	0	144	138	Bad	38	10
41	2.07	119	98	0	18	126	Bad	73	17
42	7.96	157	53	0	403	124	Bad	58	16
43	10.43	77	69	0	25	24	Medium	50	18
44	4.12	123	42	11	16	134	Medium	59	13
45	4.16	85	79	6	325	95	Medium	69	13
46	4.56	141	63	0	168	135	Bad	44	12
47	12.44	127	90	14	16	70	Medium	48	15
48	4.38	126	98	0	173	108	Bad	55	16
49	3.91	116	52	0	349	98	Bad	69	18
50	10.61	157	93	0	51	149	Good	32	17
51	1.42	99	32	18	341	108	Bad	80	16
52	4.42	121	90	0	150	108	Bad	75	16
53	7.91	153	40	3	112	129	Bad	39	18
54	6.92	109	64	13	39	119	Medium	61	17
55	4.90	134	103	13	25	144	Medium	76	17
56	6.85	143	81	5	60	154	Medium	61	18
57	11.91	133	82	0	54	84	Medium	50	17
58	0.91	93	91	0	22	117	Bad	75	11
59	5.42	103	93	15	188	103	Bad	74	16
60	5.21	118	71	4	148	114	Medium	80	13
61	8.32	122	102	19	469	123	Bad	29	13
62	7.32	105	32	0	358	107	Medium	26	13
63	1.82	139	45	0	146	133	Bad	77	17
64	8.47	119	88	10	170	101	Medium	61	13
65	7.80	100	67	12	184	104	Medium	32	16
66	4.90	122	26	0	197	128	Medium	55	13
67	8.85	127	92	0	508	91	Medium	56	18
68	9.01	126	61	14	152	115	Medium	47	16
69	13.39	149	69	20	366	134	Good	60	13
70	7.99	127	59	0	339	99	Medium	65	12
71	9.46	89	81	15	237	99	Good	74	12
72	6.50	148	51	16	148	150	Medium	58	17
73	5.52	115	45	0	432	116	Medium	25	15
74	12.61	118	90	10	54	104	Good	31	11
75	6.20	150	68	5	125	136	Medium	64	13
76	8.55	88	111	23	480	92	Bad	36	16
77	10.64	102	87	10	346	70	Medium	64	15
78	7.70	118	71	12	44	89	Medium	67	18

79	4.43	134	48	1	139	145	Medium	65	12
80	9.14	134	67	0	286	90	Bad	41	13
81	8.01	113	100	16	353	79	Bad	68	11
82	7.52	116	72	0	237	128	Good	70	13
83	11.62	151	83	4	325	139	Good	28	17
84	4.42	109	36	7	468	94	Bad	56	11
85	2.23	111	25	0	52	121	Bad	43	18
86	8.47	125	103	0	304	112	Medium	49	13
87	8.70	150	84	9	432	134	Medium	64	15
88	11.70	131	67	7	272	126	Good	54	16
89	6.56	117	42	7	144	111	Medium	62	10
90	7.95	128	66	3	493	119	Medium	45	16
91	5.33	115	22	0	491	103	Medium	64	11
92	4.81	97	46	11	267	107	Medium	80	15
93	4.53	114	113	0	97	125	Medium	29	12
94	8.86	145	30	0	67	104	Medium	55	17
95	8.39	115	97	5	134	84	Bad	55	11
96	5.58	134	25	10	237	148	Medium	59	13
97	9.48	147	42	10	407	132	Good	73	16
98	7.45	161	82	5	287	129	Bad	33	16
99	12.49	122	77	24	382	127	Good	36	16
100	4.88	121	47	3	220	107	Bad	56	16
101	4.11	113	69	11	94	106	Medium	76	12
102	6.20	128	93	0	89	118	Medium	34	18
103	5.30	113	22	0	57	97	Medium	65	16
104	5.07	123	91	0	334	96	Bad	78	17
105	4.62	121	96	0	472	138	Medium	51	12
106	5.55	104	100	8	398	97	Medium	61	11
107	0.16	102	33	0	217	139	Medium	70	18
108	8.55	134	107	0	104	108	Medium	60	12
109	3.47	107	79	2	488	103	Bad	65	16
110	8.98	115	65	0	217	90	Medium	60	17
111	9.00	128	62	7	125	116	Medium	43	14
112	6.62	132	118	12	272	151	Medium	43	14
113	6.67	116	99	5	298	125	Good	62	12
114	6.01	131	29	11	335	127	Bad	33	12
115	9.31	122	87	9	17	106	Medium	65	13
116	8.54	139	35	0	95	129	Medium	42	13
117	5.08	135	75	0	202	128	Medium	80	10
118	8.80	145	53	0	507	119	Medium	41	12
119	7.57	112	88	2	243	99	Medium	62	11
120	7.37	130	94	8	137	128	Medium	64	12
121	6.87	128	105	11	249	131	Medium	63	13

122	11.67	125	89	10	380	87	Bad	28	10
123	6.88	119	100	5	45	108	Medium	75	10
124	8.19	127	103	0	125	155	Good	29	15
125	8.87	131	113	0	181	120	Good	63	14
126	9.34	89	78	0	181	49	Medium	43	15
127	11.27	153	68	2	60	133	Good	59	16
128	6.52	125	48	3	192	116	Medium	51	14
129	4.96	133	100	3	350	126	Bad	55	13
130	4.47	143	120	7	279	147	Bad	40	10
131	8.41	94	84	13	497	77	Medium	51	12
132	6.50	108	69	3	208	94	Medium	77	16
133	9.54	125	87	9	232	136	Good	72	10
134	7.62	132	98	2	265	97	Bad	62	12
135	3.67	132	31	0	327	131	Medium	76	16
136	6.44	96	94	14	384	120	Medium	36	18
137	5.17	131	75	0	10	120	Bad	31	18
138	6.52	128	42	0	436	118	Medium	80	11
139	10.27	125	103	12	371	109	Medium	44	10
140	12.30	146	62	10	310	94	Medium	30	13
141	6.03	133	60	10	277	129	Medium	45	18
142	6.53	140	42	0	331	131	Bad	28	15
143	7.44	124	84	0	300	104	Medium	77	15
144	0.53	122	88	7	36	159	Bad	28	17
145	9.09	132	68	0	264	123	Good	34	11
146	8.77	144	63	11	27	117	Medium	47	17
147	3.90	114	83	0	412	131	Bad	39	14
148	10.51	140	54	9	402	119	Good	41	16
149	7.56	110	119	0	384	97	Medium	72	14
150	11.48	121	120	13	140	87	Medium	56	11
151	10.49	122	84	8	176	114	Good	57	10
152	10.77	111	58	17	407	103	Good	75	17
153	7.64	128	78	0	341	128	Good	45	13
154	5.93	150	36	7	488	150	Medium	25	17
155	6.89	129	69	10	289	110	Medium	50	16
156	7.71	98	72	0	59	69	Medium	65	16
157	7.49	146	34	0	220	157	Good	51	16
158	10.21	121	58	8	249	90	Medium	48	13
159	12.53	142	90	1	189	112	Good	39	10
160	9.32	119	60	0	372	70	Bad	30	18
161	4.67	111	28	0	486	111	Medium	29	12
162	2.93	143	21	5	81	160	Medium	67	12
163	3.63	122	74	0	424	149	Medium	51	13
164	5.68	130	64	0	40	106	Bad	39	17

165	8.22	148	64	0	58	141	Medium	27	13
166	0.37	147	58	7	100	191	Bad	27	15
167	6.71	119	67	17	151	137	Medium	55	11
168	6.71	106	73	0	216	93	Medium	60	13
169	7.30	129	89	0	425	117	Medium	45	10
170	11.48	104	41	15	492	77	Good	73	18
171	8.01	128	39	12	356	118	Medium	71	10
172	12.49	93	106	12	416	55	Medium	75	15
173	9.03	104	102	13	123	110	Good	35	16
174	6.38	135	91	5	207	128	Medium	66	18
175	0.00	139	24	0	358	185	Medium	79	15
176	7.54	115	89	0	38	122	Medium	25	12
177	5.61	138	107	9	480	154	Medium	47	11
178	10.48	138	72	0	148	94	Medium	27	17
179	10.66	104	71	14	89	81	Medium	25	14
180	7.78	144	25	3	70	116	Medium	77	18
181	4.94	137	112	15	434	149	Bad	66	13
182	7.43	121	83	0	79	91	Medium	68	11
183	4.74	137	60	4	230	140	Bad	25	13
184	5.32	118	74	6	426	102	Medium	80	18
185	9.95	132	33	7	35	97	Medium	60	11
186	10.07	130	100	11	449	107	Medium	64	10
187	8.68	120	51	0	93	86	Medium	46	17
188	6.03	117	32	0	142	96	Bad	62	17
189	8.07	116	37	0	426	90	Medium	76	15
190	12.11	118	117	18	509	104	Medium	26	15
191	8.79	130	37	13	297	101	Medium	37	13
192	6.67	156	42	13	170	173	Good	74	14
193	7.56	108	26	0	408	93	Medium	56	14
194	13.28	139	70	7	71	96	Good	61	10
195	7.23	112	98	18	481	128	Medium	45	11
196	4.19	117	93	4	420	112	Bad	66	11
197	4.10	130	28	6	410	133	Bad	72	16
198	2.52	124	61	0	333	138	Medium	76	16
199	3.62	112	80	5	500	128	Medium	69	10
200	6.42	122	88	5	335	126	Medium	64	14
201	5.56	144	92	0	349	146	Medium	62	12
202	5.94	138	83	0	139	134	Medium	54	18
203	4.10	121	78	4	413	130	Bad	46	10
204	2.05	131	82	0	132	157	Bad	25	14
205	8.74	155	80	0	237	124	Medium	37	14
206	5.68	113	22	1	317	132	Medium	28	12
207	4.97	162	67	0	27	160	Medium	77	17

208	8.19	111	105	0	466	97	Bad	61	10
209	7.78	86	54	0	497	64	Bad	33	12
210	3.02	98	21	11	326	90	Bad	76	11
211	4.36	125	41	2	357	123	Bad	47	14
212	9.39	117	118	14	445	120	Medium	32	15
213	12.04	145	69	19	501	105	Medium	45	11
214	8.23	149	84	5	220	139	Medium	33	10
215	4.83	115	115	3	48	107	Medium	73	18
216	2.34	116	83	15	170	144	Bad	71	11
217	5.73	141	33	0	243	144	Medium	34	17
218	4.34	106	44	0	481	111	Medium	70	14
219	9.70	138	61	12	156	120	Medium	25	14
220	10.62	116	79	19	359	116	Good	58	17
221	10.59	131	120	15	262	124	Medium	30	10
222	6.43	124	44	0	125	107	Medium	80	11
223	7.49	136	119	6	178	145	Medium	35	13
224	3.45	110	45	9	276	125	Medium	62	14
225	4.10	134	82	0	464	141	Medium	48	13
226	6.68	107	25	0	412	82	Bad	36	14
227	7.80	119	33	0	245	122	Good	56	14
228	8.69	113	64	10	68	101	Medium	57	16
229	5.40	149	73	13	381	163	Bad	26	11
230	11.19	98	104	0	404	72	Medium	27	18
231	5.16	115	60	0	119	114	Bad	38	14
232	8.09	132	69	0	123	122	Medium	27	11
233	13.14	137	80	10	24	105	Good	61	15
234	8.65	123	76	18	218	120	Medium	29	14
235	9.43	115	62	11	289	129	Good	56	16
236	5.53	126	32	8	95	132	Medium	50	17
237	9.32	141	34	16	361	108	Medium	69	10
238	9.62	151	28	8	499	135	Medium	48	10
239	7.36	121	24	0	200	133	Good	73	13
240	3.89	123	105	0	149	118	Bad	62	16
241	10.31	159	80	0	362	121	Medium	26	18
242	12.01	136	63	0	160	94	Medium	38	12
243	4.68	124	46	0	199	135	Medium	52	14
244	7.82	124	25	13	87	110	Medium	57	10
245	8.78	130	30	0	391	100	Medium	26	18
246	10.00	114	43	0	199	88	Good	57	10
247	6.90	120	56	20	266	90	Bad	78	18
248	5.04	123	114	0	298	151	Bad	34	16
249	5.36	111	52	0	12	101	Medium	61	11
250	5.05	125	67	0	86	117	Bad	65	11

251	9.16	137	105	10	435	156	Good	72	14
252	3.72	139	111	5	310	132	Bad	62	13
253	8.31	133	97	0	70	117	Medium	32	16
254	5.64	124	24	5	288	122	Medium	57	12
255	9.58	108	104	23	353	129	Good	37	17
256	7.71	123	81	8	198	81	Bad	80	15
257	4.20	147	40	0	277	144	Medium	73	10
258	8.67	125	62	14	477	112	Medium	80	13
259	3.47	108	38	0	251	81	Bad	72	14
260	5.12	123	36	10	467	100	Bad	74	11
261	7.67	129	117	8	400	101	Bad	36	10
262	5.71	121	42	4	188	118	Medium	54	15
263	6.37	120	77	15	86	132	Medium	48	18
264	7.77	116	26	6	434	115	Medium	25	17
265	6.95	128	29	5	324	159	Good	31	15
266	5.31	130	35	10	402	129	Bad	39	17
267	9.10	128	93	12	343	112	Good	73	17
268	5.83	134	82	7	473	112	Bad	51	12
269	6.53	123	57	0	66	105	Medium	39	11
270	5.01	159	69	0	438	166	Medium	46	17
271	11.99	119	26	0	284	89	Good	26	10
272	4.55	111	56	0	504	110	Medium	62	16
273	12.98	113	33	0	14	63	Good	38	12
274	10.04	116	106	8	244	86	Medium	58	12
275	7.22	135	93	2	67	119	Medium	34	11
276	6.67	107	119	11	210	132	Medium	53	11
277	6.93	135	69	14	296	130	Medium	73	15
278	7.80	136	48	12	326	125	Medium	36	16
279	7.22	114	113	2	129	151	Good	40	15
280	3.42	141	57	13	376	158	Medium	64	18
281	2.86	121	86	10	496	145	Bad	51	10
282	11.19	122	69	7	303	105	Good	45	16
283	7.74	150	96	0	80	154	Good	61	11
284	5.36	135	110	0	112	117	Medium	80	16
285	6.97	106	46	11	414	96	Bad	79	17
286	7.60	146	26	11	261	131	Medium	39	10
287	7.53	117	118	11	429	113	Medium	67	18
288	6.88	95	44	4	208	72	Bad	44	17
289	6.98	116	40	0	74	97	Medium	76	15
290	8.75	143	77	25	448	156	Medium	43	17
291	9.49	107	111	14	400	103	Medium	41	11
292	6.64	118	70	0	106	89	Bad	39	17
293	11.82	113	66	16	322	74	Good	76	15

294	11.28	123	84	0	74	89	Good	59	10
295	12.66	148	76	3	126	99	Good	60	11
296	4.21	118	35	14	502	137	Medium	79	10
297	8.21	127	44	13	160	123	Good	63	18
298	3.07	118	83	13	276	104	Bad	75	10
299	10.98	148	63	0	312	130	Good	63	15
300	9.40	135	40	17	497	96	Medium	54	17
301	8.57	116	78	1	158	99	Medium	45	11
302	7.41	99	93	0	198	87	Medium	57	16
303	5.28	108	77	13	388	110	Bad	74	14
304	10.01	133	52	16	290	99	Medium	43	11
305	11.93	123	98	12	408	134	Good	29	10
306	8.03	115	29	26	394	132	Medium	33	13
307	4.78	131	32	1	85	133	Medium	48	12
308	5.90	138	92	0	13	120	Bad	61	12
309	9.24	126	80	19	436	126	Medium	52	10
310	11.18	131	111	13	33	80	Bad	68	18
311	9.53	175	65	29	419	166	Medium	53	12
312	6.15	146	68	12	328	132	Bad	51	14
313	6.80	137	117	5	337	135	Bad	38	10
314	9.33	103	81	3	491	54	Medium	66	13
315	7.72	133	33	10	333	129	Good	71	14
316	6.39	131	21	8	220	171	Good	29	14
317	15.63	122	36	5	369	72	Good	35	10
318	6.41	142	30	0	472	136	Good	80	15
319	10.08	116	72	10	456	130	Good	41	14
320	6.97	127	45	19	459	129	Medium	57	11
321	5.86	136	70	12	171	152	Medium	44	18
322	7.52	123	39	5	499	98	Medium	34	15
323	9.16	140	50	10	300	139	Good	60	15
324	10.36	107	105	18	428	103	Medium	34	12
325	2.66	136	65	4	133	150	Bad	53	13
326	11.70	144	69	11	131	104	Medium	47	11
327	4.69	133	30	0	152	122	Medium	53	17
328	6.23	112	38	17	316	104	Medium	80	16
329	3.15	117	66	1	65	111	Bad	55	11
330	11.27	100	54	9	433	89	Good	45	12
331	4.99	122	59	0	501	112	Bad	32	14
332	10.10	135	63	15	213	134	Medium	32	10
333	5.74	106	33	20	354	104	Medium	61	12
334	5.87	136	60	7	303	147	Medium	41	10
335	7.63	93	117	9	489	83	Bad	42	13
336	6.18	120	70	15	464	110	Medium	72	15

337	5.17	138	35	6	60	143	Bad	28	18
338	8.61	130	38	0	283	102	Medium	80	15
339	5.97	112	24	0	164	101	Medium	45	11
340	11.54	134	44	4	219	126	Good	44	15
341	7.50	140	29	0	105	91	Bad	43	16
342	7.38	98	120	0	268	93	Medium	72	10
343	7.81	137	102	13	422	118	Medium	71	10
344	5.99	117	42	10	371	121	Bad	26	14
345	8.43	138	80	0	108	126	Good	70	13
346	4.81	121	68	0	279	149	Good	79	12
347	8.97	132	107	0	144	125	Medium	33	13
348	6.88	96	39	0	161	112	Good	27	14
349	12.57	132	102	20	459	107	Good	49	11
350	9.32	134	27	18	467	96	Medium	49	14
351	8.64	111	101	17	266	91	Medium	63	17
352	10.44	124	115	16	458	105	Medium	62	16
353	13.44	133	103	14	288	122	Good	61	17
354	9.45	107	67	12	430	92	Medium	35	12
355	5.30	133	31	1	80	145	Medium	42	18
356	7.02	130	100	0	306	146	Good	42	11
357	3.58	142	109	0	111	164	Good	72	12
358	13.36	103	73	3	276	72	Medium	34	15
359	4.17	123	96	10	71	118	Bad	69	11
360	3.13	130	62	11	396	130	Bad	66	14
361	8.77	118	86	7	265	114	Good	52	15
362	8.68	131	25	10	183	104	Medium	56	15
363	5.25	131	55	0	26	110	Bad	79	12
364	10.26	111	75	1	377	108	Good	25	12
365	10.50	122	21	16	488	131	Good	30	14
366	6.53	154	30	0	122	162	Medium	57	17
367	5.98	124	56	11	447	134	Medium	53	12
368	14.37	95	106	0	256	53	Good	52	17
369	10.71	109	22	10	348	79	Good	74	14
370	10.26	135	100	22	463	122	Medium	36	14
371	7.68	126	41	22	403	119	Bad	42	12
372	9.08	152	81	0	191	126	Medium	54	16
373	7.80	121	50	0	508	98	Medium	65	11
374	5.58	137	71	0	402	116	Medium	78	17
375	9.44	131	47	7	90	118	Medium	47	12
376	7.90	132	46	4	206	124	Medium	73	11
377	16.27	141	60	19	319	92	Good	44	11
378	6.81	132	61	0	263	125	Medium	41	12
379	6.11	133	88	3	105	119	Medium	79	12

380	5.81	125	111	0	404	107	Bad	54	15
381	9.64	106	64	10	17	89	Medium	68	17
382	3.90	124	65	21	496	151	Bad	77	13
383	4.95	121	28	19	315	121	Medium	66	14
384	9.35	98	117	0	76	68	Medium	63	10
385	12.85	123	37	15	348	112	Good	28	12
386	5.87	131	73	13	455	132	Medium	62	17
387	5.32	152	116	0	170	160	Medium	39	16
388	8.67	142	73	14	238	115	Medium	73	14
389	8.14	135	89	11	245	78	Bad	79	16
390	8.44	128	42	8	328	107	Medium	35	12
391	5.47	108	75	9	61	111	Medium	67	12
392	6.10	153	63	0	49	124	Bad	56	16
393	4.53	129	42	13	315	130	Bad	34	13
394	5.57	109	51	10	26	120	Medium	30	17
395	5.35	130	58	19	366	139	Bad	33	16
396	12.57	138	108	17	203	128	Good	33	14
397	6.14	139	23	3	37	120	Medium	55	11
398	7.41	162	26	12	368	159	Medium	40	18
399	5.94	100	79	7	284	95	Bad	50	12
400	9.71	134	37	0	27	120	Good	49	16

	Urban	US
1	Yes	Yes
2	Yes	Yes
3	Yes	Yes
4	Yes	Yes
5	Yes	No
6	No	Yes
7	Yes	No
8	Yes	Yes
9	No	No
10	No	Yes
11	No	Yes
12	Yes	Yes
13	Yes	No
14	Yes	Yes
15	Yes	Yes
16	No	No
17	Yes	No
18	Yes	Yes
19	No	Yes
20	Yes	Yes
21	Yes	Yes

```
22     No Yes
23    Yes  No
24    Yes  No
25    Yes Yes
26     No  No
27     No Yes
28    Yes  No
29    Yes Yes
30    Yes Yes
31    Yes  No
32    Yes Yes
33     No Yes
34    Yes Yes
35    Yes Yes
36     No Yes
37     No  No
38    Yes Yes
39    Yes  No
40     No  No
41     No  No
42    Yes  No
43    Yes  No
44    Yes Yes
45    Yes Yes
46    Yes Yes
47     No Yes
48    Yes  No
49    Yes  No
50    Yes  No
51    Yes Yes
52    Yes  No
53    Yes Yes
54    Yes Yes
55     No Yes
56    Yes Yes
57    Yes  No
58    Yes  No
59    Yes Yes
60    Yes  No
61    Yes Yes
62     No  No
63    Yes Yes
64    Yes Yes
```

```
65     No Yes
66     No  No
67    Yes  No
68    Yes Yes
69    Yes Yes
70    Yes  No
71    Yes Yes
72     No Yes
73    Yes  No
74     No Yes
75     No Yes
76     No Yes
77    Yes Yes
78     No Yes
79    Yes Yes
80    Yes  No
81    Yes Yes
82    Yes  No
83    Yes Yes
84    Yes Yes
85     No  No
86     No  No
87    Yes  No
88     No Yes
89    Yes Yes
90     No  No
91     No  No
92    Yes Yes
93    Yes  No
94    Yes  No
95    Yes Yes
96    Yes Yes
97     No Yes
98    Yes Yes
99     No Yes
100    No Yes
101    No Yes
102   Yes  No
103    No  No
104   Yes Yes
105   Yes  No
106   Yes Yes
107    No  No
```

```
108   Yes  No
109   Yes  No
110    No  No
111   Yes Yes
112   Yes Yes
113   Yes Yes
114   Yes Yes
115   Yes Yes
116   Yes  No
117    No  No
118   Yes  No
119   Yes Yes
120   Yes Yes
121   Yes Yes
122   Yes Yes
123   Yes Yes
124    No Yes
125   Yes  No
126    No  No
127   Yes Yes
128   Yes Yes
129   Yes Yes
130    No Yes
131   Yes Yes
132   Yes  No
133   Yes Yes
134   Yes Yes
135   Yes  No
136    No Yes
137    No  No
138   Yes  No
139   Yes Yes
140    No Yes
141   Yes Yes
142   Yes  No
143   Yes  No
144   Yes Yes
145    No  No
146   Yes Yes
147   Yes  No
148    No Yes
149    No Yes
150   Yes Yes
```

```
151    No Yes
152    No Yes
153    No  No
154    No Yes
155    No Yes
156   Yes  No
157   Yes  No
158    No Yes
159    No Yes
160    No  No
161    No  No
162    No Yes
163   Yes  No
164    No  No
165    No Yes
166   Yes Yes
167   Yes Yes
168   Yes  No
169   Yes  No
170   Yes Yes
171   Yes Yes
172   Yes Yes
173   Yes Yes
174   Yes Yes
175    No  No
176   Yes  No
177    No Yes
178   Yes Yes
179    No Yes
180   Yes Yes
181   Yes Yes
182   Yes  No
183   Yes  No
184   Yes Yes
185    No Yes
186   Yes Yes
187    No  No
188   Yes  No
189   Yes  No
190    No Yes
191    No Yes
192   Yes Yes
193    No  No
```

```
194   Yes Yes
195   Yes Yes
196   Yes Yes
197   Yes Yes
198   Yes  No
199   Yes Yes
200   Yes Yes
201    No  No
202   Yes  No
203    No Yes
204   Yes  No
205   Yes  No
206   Yes  No
207   Yes Yes
208    No  No
209   Yes  No
210    No Yes
211    No Yes
212   Yes Yes
213   Yes Yes
214   Yes Yes
215   Yes Yes
216   Yes Yes
217   Yes  No
218    No  No
219   Yes Yes
220   Yes Yes
221   Yes Yes
222   Yes  No
223   Yes Yes
224   Yes Yes
225    No  No
226   Yes  No
227   Yes  No
228   Yes Yes
229    No Yes
230    No  No
231    No  No
232    No  No
233   Yes Yes
234    No Yes
235    No Yes
236   Yes Yes
```

```
237   Yes Yes
238   Yes Yes
239   Yes  No
240   Yes Yes
241   Yes  No
242   Yes  No
243    No  No
244   Yes Yes
245   Yes  No
246    No Yes
247   Yes Yes
248   Yes  No
249   Yes Yes
250   Yes  No
251   Yes Yes
252   Yes Yes
253   Yes  No
254    No Yes
255   Yes Yes
256   Yes Yes
257   Yes  No
258   Yes Yes
259    No  No
260    No Yes
261   Yes Yes
262   Yes Yes
263   Yes Yes
264   Yes Yes
265   Yes Yes
266   Yes Yes
267    No Yes
268    No Yes
269   Yes  No
270   Yes  No
271   Yes  No
272   Yes  No
273   Yes  No
274   Yes Yes
275   Yes Yes
276   Yes Yes
277   Yes Yes
278   Yes Yes
279    No Yes
```

```
280   Yes Yes
281   Yes Yes
282    No Yes
283   Yes  No
284    No  No
285    No  No
286   Yes Yes
287    No Yes
288   Yes Yes
289    No  No
290   Yes Yes
291    No Yes
292   Yes  No
293   Yes Yes
294   Yes  No
295   Yes Yes
296    No Yes
297   Yes Yes
298   Yes Yes
299   Yes  No
300    No Yes
301   Yes Yes
302   Yes Yes
303   Yes Yes
304   Yes Yes
305   Yes Yes
306   Yes Yes
307   Yes Yes
308   Yes  No
309   Yes Yes
310   Yes Yes
311   Yes Yes
312   Yes Yes
313   Yes Yes
314   Yes  No
315   Yes Yes
316   Yes Yes
317   Yes Yes
318    No  No
319    No Yes
320    No Yes
321   Yes Yes
322   Yes  No
```

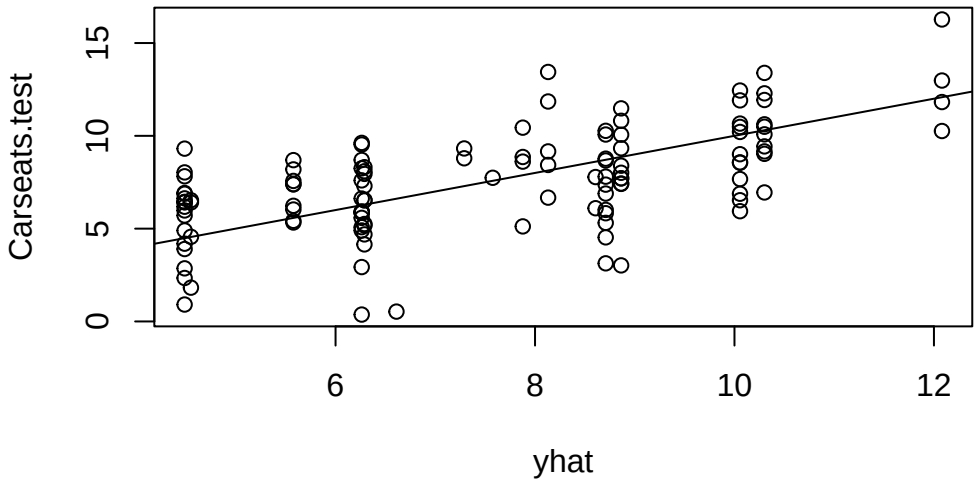
```
323   Yes Yes
324   Yes Yes
325   Yes Yes
326   Yes Yes
327   Yes  No
328   Yes Yes
329   Yes Yes
330   Yes Yes
331    No  No
332   Yes Yes
333   Yes Yes
334   Yes Yes
335   Yes Yes
336   Yes Yes
337   Yes  No
338   Yes  No
339   Yes  No
340   Yes Yes
341   Yes  No
342    No  No
343    No Yes
344   Yes Yes
345    No Yes
346   Yes  No
347    No  No
348    No  No
349   Yes Yes
350    No Yes
351    No Yes
352    No Yes
353   Yes Yes
354    No Yes
355   Yes Yes
356   Yes  No
357   Yes  No
358   Yes Yes
359   Yes Yes
360   Yes Yes
361    No Yes
362    No Yes
363   Yes Yes
364   Yes  No
365   Yes Yes
```

```
366    No  No
367    No Yes
368   Yes  No
369    No Yes
370   Yes Yes
371   Yes Yes
372   Yes  No
373    No  No
374   Yes  No
375   Yes Yes
376   Yes  No
377   Yes Yes
378    No  No
379   Yes Yes
380   Yes  No
381   Yes Yes
382   Yes Yes
383   Yes Yes
384   Yes  No
385   Yes Yes
386   Yes Yes
387   Yes  No
388    No Yes
389   Yes Yes
390   Yes Yes
391   Yes Yes
392   Yes  No
393   Yes Yes
394    No Yes
395   Yes Yes
396   Yes Yes
397    No Yes
398   Yes Yes
399   Yes Yes
400   Yes Yes
```

#a) Split the data set into a training set and a test set. #b) Fit a regression tree to the training set.

```
set.seed(0)
train <- sample(1:nrow(Carseats), 0.7 * nrow(Carseats))
```

```
Carseats.test <- Carseats[-train,"Sales"]
tree.carseats <- tree(Sales ~ . - Sales, Carseats,
                      subset = train)

yhat <- predict(tree.carseats, newdata = Carseats[-train, ])
plot(yhat, Carseats.test)
abline(0, 1)
```



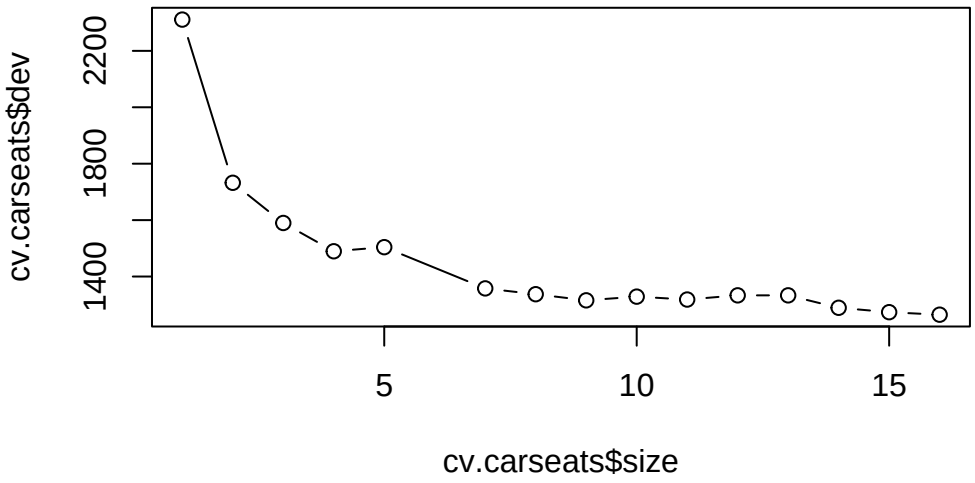
test MSE obtain:4.208383

```
mean((yhat - Carseats.test)^2)
```

```
[1] 5.114749
```

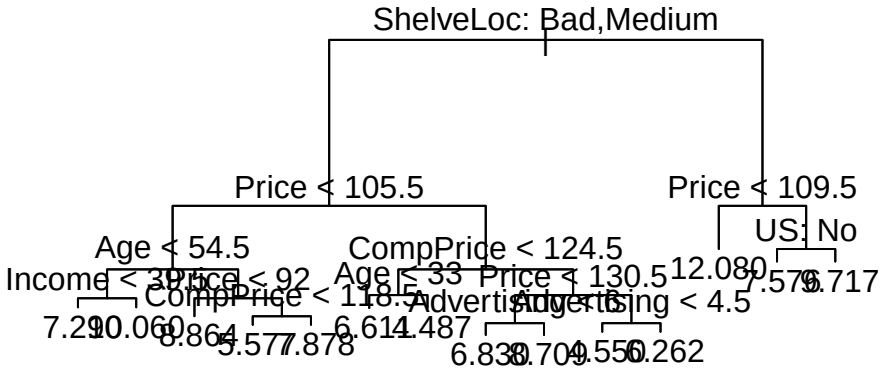
#c)

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



#14 node

```
prune.carseats <- prune.tree(tree.carseats, best = 14)
plot(prune.carseats)
text(prune.carseats, pretty = 0)
```




```
tree.pred <- predict(prune.carseats, newdata = Carseats[-train, ])
mean((tree.pred - Carseats.test)^2)
```

```
[1] 4.985037
```

pruning the tree doesn't improve the test MSE

#d) Use the bagging approach in order to analyze this data

```
set.seed(0)
bag.carseats <- randomForest(Sales ~ . - Sales, data = Carseats,
                             subset = train, mtry = 10, #m = p Bagging
                             importance = TRUE)

tree.pred <- predict(bag.carseats, newdata = Carseats[-train, ])
mean((tree.pred - Carseats.test)^2)
```

```
[1] 2.442813
```

#CompPrice most important

```
importance(bag.carseats)
```

```
               %IncMSE IncNodePurity
CompPrice   34.1090930     237.85419
Income      10.2275700     120.44990
Advertising 22.0440334     162.32865
Population  -0.9935855      62.44170
Price       70.2265476     665.46962
ShelveLoc   65.6422575     638.67282
Age         22.0290148     244.48819
Education    2.7421559      61.00877
Urban       -2.1844625      11.20796
US           3.9897925      14.47018
```

#e) Use random forests to analyze this data. m = p^0.5

```
set.seed(0)
rf.carseats <- randomForest(Sales ~ . - Sales, data = Carseats,
                            subset = train, mtry = 3, #m = p^0.5
                            importance = TRUE)

tree.pred <- predict(rf.carseats, newdata = Carseats[-train, ])
mean((tree.pred - Carseats.test)^2)
```

```
[1] 2.922625
```

#f)Bart

```r
x <- Carseats[, 2:11]
y <- Carseats[, "Sales"]
xtrain <- x[train, ]
ytrain <- y[train]
xtest <- x[-train, ]
ytest <- y[-train]
set.seed(1)
bartfit <- gbart(xtrain, ytrain, x.test = xtest)
```

```
*****Calling gbart: type=1
*****Data:
data:n,p,np: 280, 14, 120
y1,yn: -0.075857, 5.124143
x1,x[n*p]: 162.000000, 1.000000
xp1,xp[np*p]: 113.000000, 1.000000
*****Number of Trees: 200
*****Number of Cut Points: 69 ... 1
*****burn,nd,thin: 100,1000,1
*****Prior:beta,alpha,tau,nu,lambda,offset: 2,0.95,0.276302,3,0.21794,7.48586
*****sigma: 1.057750
*****w (weights): 1.000000 ... 1.000000
*****Dirichlet:sparse,theta,omega,a,b,rho,augment: 0,0,1,0.5,1,14,0
*****printevery: 100

MCMC
done 0 (out of 1100)
done 100 (out of 1100)
done 200 (out of 1100)
done 300 (out of 1100)
done 400 (out of 1100)
done 500 (out of 1100)
done 600 (out of 1100)
done 700 (out of 1100)
done 800 (out of 1100)
done 900 (out of 1100)
done 1000 (out of 1100)
time: 2s
trcnt,tecnt: 1000,1000
```

```r
yhat.bart <- bartfit$yhat.test.mean
mean((ytest - yhat.bart)^2)
```

```
[1] 1.373941
```

##test error of BART is smaller than that of random forest and bagging #9. involves the OJ data set which is part of the ISLR2 package. #Create a training set containing a random sample of 800 observations

```
set.seed(0)
train <- sample(1:nrow(OJ), 800)

OJ.train <- OJ[train, ]
OJ.test <- OJ[-train, ]
Purchase.test <- OJ$Purchase[-train]  # Extract test target variable
```

#b) training error rate? How many terminal nodes does the tree have?

```
tree.OJ <- tree(Purchase ~ . - Purchase, OJ,
                subset = train)
tree.pred <- predict(tree.OJ, OJ.train,
                     type = "class")
table(tree.pred, OJ$Purchase[train])
```

```
tree.pred  CH  MM
       CH 416  52
       MM  75 257
```

#training error =1- (416+257)/800 = 0.15875

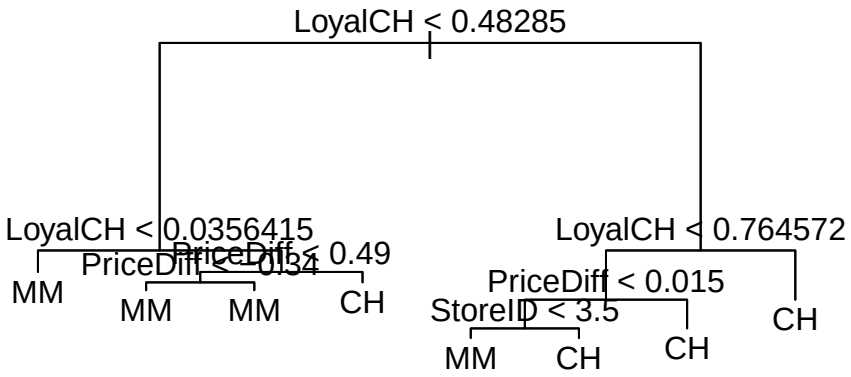
#c) Type in the name of the tree object in order to get a detailed text output, Pick one of the terminal nodes, #d)

```
tree.OJ
```

```
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

 1) root 800 1067.000 CH ( 0.61375 0.38625 )
   2) LoyalCH < 0.48285 297  329.000 MM ( 0.24242 0.75758 )
     4) LoyalCH < 0.0356415 60   10.170 MM ( 0.01667 0.98333 ) *
     5) LoyalCH > 0.0356415 237  289.400 MM ( 0.29958 0.70042 )
      10) PriceDiff < 0.49 228  268.800 MM ( 0.27632 0.72368 )
```

```
        20) PriceDiff < -0.34 16    0.000 MM ( 0.00000 1.00000 ) *
        21) PriceDiff > -0.34 212  258.000 MM ( 0.29717 0.70283 ) *
      11) PriceDiff > 0.49 9    6.279 CH ( 0.88889 0.11111 ) *
    3) LoyalCH > 0.48285 503  453.800 CH ( 0.83300 0.16700 )
      6) LoyalCH < 0.764572 233  288.100 CH ( 0.69099 0.30901 )
       12) PriceDiff < 0.015 83  113.600 MM ( 0.43373 0.56627 )
         24) StoreID < 3.5 44   49.490 MM ( 0.25000 0.75000 ) *
         25) StoreID > 3.5 39   50.920 CH ( 0.64103 0.35897 ) *
       13) PriceDiff > 0.015 150  135.200 CH ( 0.83333 0.16667 ) *
      7) LoyalCH > 0.764572 270   98.180 CH ( 0.95556 0.04444 ) *
```

```
plot(tree.OJ)
text(tree.OJ, pretty = 0)
```



#predicted class at the root node is "CH", meaning the model would predict that customers are more likely to purchase CH (Citrus Hill) if no further splits are made.

#e) Predict the response on the test data

```
tree.pred <- predict(tree.OJ, OJ.test,
                     type = "class")
table(tree.pred, Purchase.test)
```

```
         Purchase.test
tree.pred  CH  MM
       CH 134  24
       MM  28  84
```

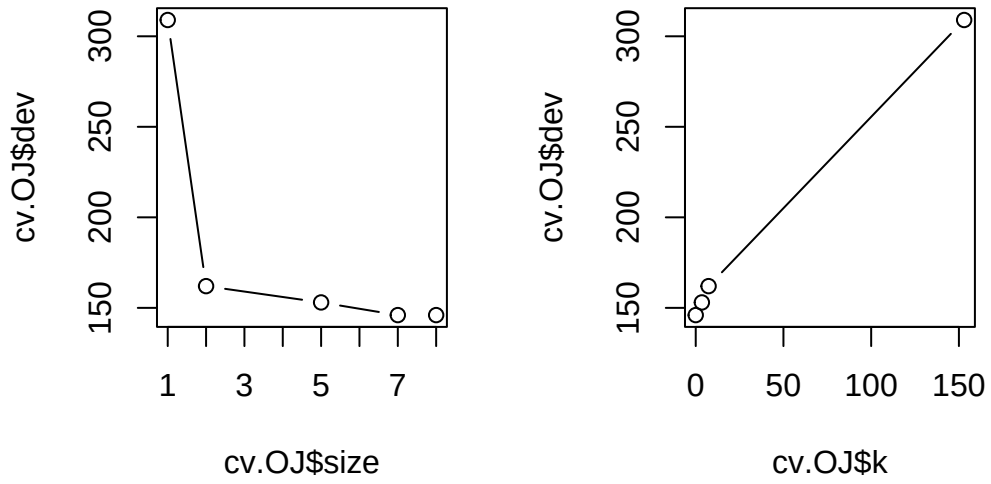
#test error = 1 - (134+84)/270 = 0.1925926

#f)Use cross-validation to determine the optimal tree size.

```
set.seed(0)
cv.OJ <- cv.tree(tree.OJ, FUN = prune.misclass)
names(cv.OJ)
```

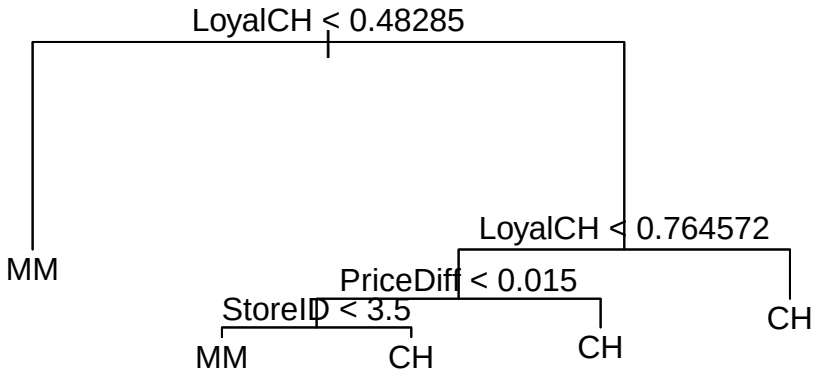
```
[1] "size"   "dev"    "k"      "method"
```

```
par(mfrow = c(1, 2))
plot(cv.OJ$size, cv.OJ$dev, type = "b")
plot(cv.OJ$k, cv.OJ$dev, type = "b")
```



#best = 8 ##In this case, the most complex tree under consideration is selected by cross validation #i) create a pruned tree with five terminal nodes.

```
prune.OJ <- prune.misclass(tree.OJ, best = 5)
plot(prune.OJ)
text(prune.OJ, pretty = 0)
```



#j)

```
tree.pred <- predict(prune.OJ, OJ.train,
                     type = "class")
table(tree.pred, OJ$Purchase[train])
```

```
tree.pred  CH  MM
       CH 408  51
       MM  83 258
```

training error= 1- (408+258)/800 = 0.1675 0.15875

h) Compare the training error rates between the pruned

```
tree.pred <- predict(prune.OJ, OJ.test,
                     type = "class")
table(tree.pred, Purchase.test)
```

```
         Purchase.test
tree.pred  CH  MM
       CH 133  21
       MM  29  87
```

##test error = 1 - (133+87)/270 = 0.1851852 #the pruned node =5 decrease the test error

#10.use boosting to predict Salary in the Hitters data set #a)

```
library(ISLR2)
data(Hitters)
attach(Hitters)
Hitters<- Hitters[!is.na(Hitters$Salary), ]
Hitters$Salary <- log(Hitters$Salary)
```

#b) Create a training set consisting of the frst 200 observations

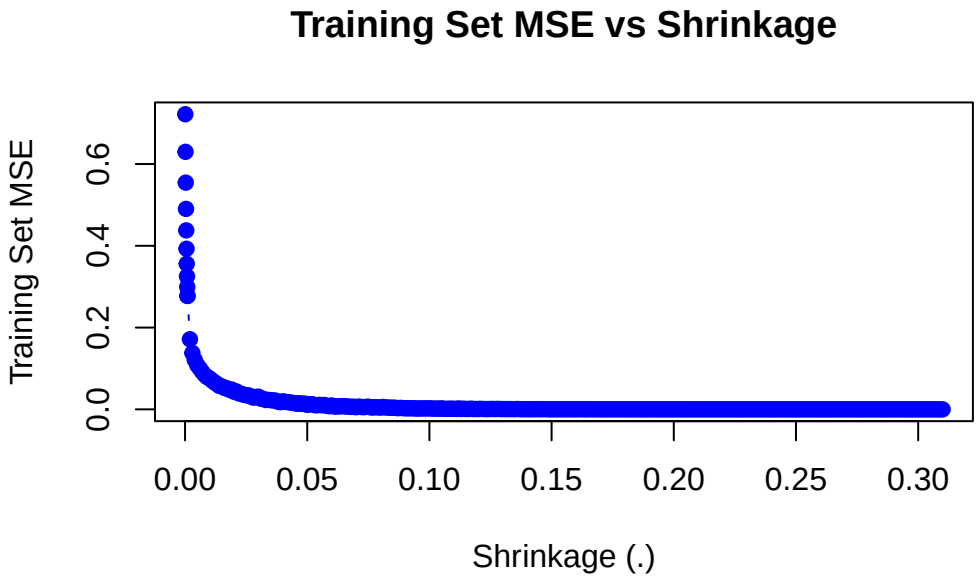
```r
train <- 1:200

Hitters.train <- Hitters[train, ]
Hitters.test <- Hitters[-train, ]
Salary.train <- Hitters$Salary[train]
Salary.test <- Hitters$Salary[-train]
```

#c) Perform boosting on the training set with 1,000 trees

```r
set.seed(0)
learning_rates <- c(seq(0.0001, 0.001, 0.0001), seq(0.001, 0.31, 0.001))

train_mse <- numeric(length(learning_rates))
test_mse <- numeric(length(learning_rates))

# Loop over the learning rates and fit gradient boosting models
for (learning_rate in learning_rates) {
  boost.Hitters <- gbm(Salary ~ .,
                       data = Hitters[train, ],
                       distribution = "gaussian",
                       n.trees = 1000,
                       shrinkage = learning_rate,  # Use learning_rate directly
                       interaction.depth = 3,
                       verbose = F)

  pred_train <- predict(boost.Hitters, newdata = Hitters[train, ], n.trees = 1000)
  pred_test <- predict(boost.Hitters, newdata = Hitters[-train, ], n.trees = 1000)

  train_mse[which(learning_rates == learning_rate)] <- mean((pred_train - Hitters$Salary[tra
  test_mse[which(learning_rates == learning_rate)] <- mean((pred_test - Hitters$Salary[-trai
}
```

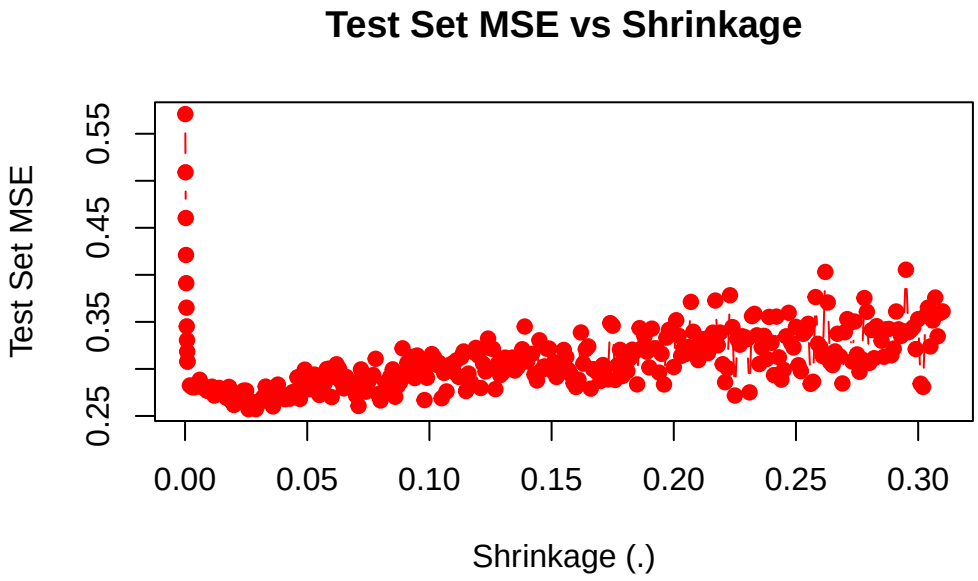
#c)

```r
plot(learning_rates, train_mse, type = "b", col = "blue",
     xlab = "Shrinkage ( )", ylab = "Training Set MSE",
     main = "Training Set MSE vs Shrinkage", pch = 19)
```



#d)

```
plot(learning_rates, test_mse, type = "b", col = "red",
     xlab = "Shrinkage ( )", ylab = "Test Set MSE",
     main = "Test Set MSE vs Shrinkage", pch = 19)
```




```
optimal_index <- which.min(test_mse)
optimal_learning_rate <- learning_rates[optimal_index]
optimal_test_mse <- test_mse[optimal_index]
```

```r
# Output the optimal learning rate and test MSE
cat("Optimal Learning Rate: ", optimal_learning_rate, "\n")
```

```
Optimal Learning Rate:  0.026
```

```r
cat("Test MSE at Optimal Learning Rate: ", optimal_test_mse, "\n")
```

```
Test MSE at Optimal Learning Rate:  0.2571844
```

#e) #OLS

```r
lm <- lm(Salary ~ ., data = Hitters, subset = train)
pred_test <- predict(lm, newdata = Hitters[-train, ])
test_mse <- mean((pred_test - Hitters$Salary[-train])^2)
cat("MSE for linear regression: ", test_mse, "\n")
```

```
MSE for linear regression:  0.4917959
```

Lasso Regression

```r
library(glmnet)
```

```
Loading required package: Matrix
```

```
Loaded glmnet 4.1-8
```

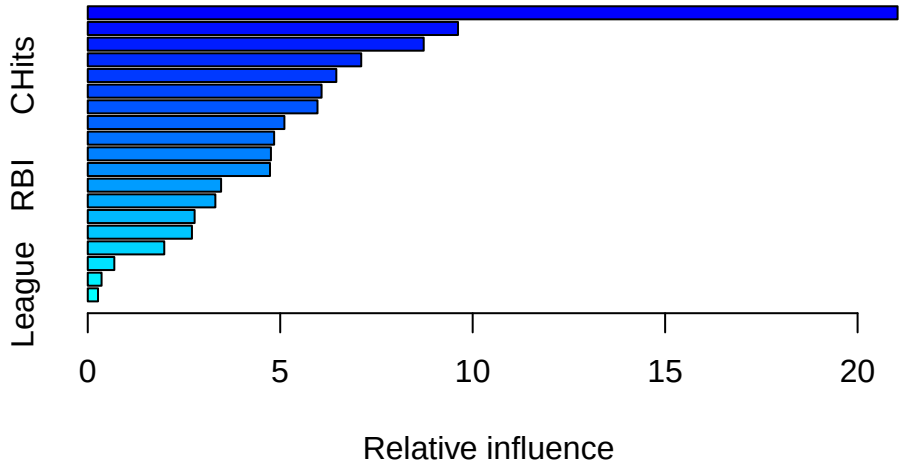
```r
set.seed(0)

x_train <- model.matrix(Salary ~ . - Salary, data = Hitters[train, ])
x_test<- model.matrix(Salary ~ . - Salary, data = Hitters[-train, ])
grid <- 10^seq(10, -2, length = 100)
lasso.mod <- glmnet(x_train, Salary.train, alpha = 1,
                    lambda = grid)
cv.out <- cv.glmnet(x_train, Salary.train, alpha = 1)

bestlam <- cv.out$lambda.min
lasso.pred <- predict(lasso.mod, s = bestlam,
                      newx = x_test)
mean((lasso.pred - Salary.test)^2)
```

```
[1] 0.454857
```

#both are much higher than boosting

#f) Which variables appear to be the most important predictors in the boosted model

```
boost.Hitters <- gbm(Salary ~ .,
                        data = Hitters[train, ],
                        distribution = "gaussian",
                        n.trees = 1000,
                        shrinkage = 0.047,  # Use learning_rate directly
                        interaction.depth = 3,
                        verbose = F)

summary(boost.Hitters)
```




```
               var    rel.inf
CAtBat       CAtBat 21.0389883
CWalks       CWalks  9.6188129
CRuns         CRuns  8.7276990
CRBI           CRBI  7.1043984
CHits         CHits  6.4573076
Walks         Walks  6.0744552
PutOuts     PutOuts  5.9685078
CHmRun       CHmRun  5.1090045
Years         Years  4.8422479
AtBat         AtBat  4.7588705
Assists     Assists  4.7343484
```

```
RBI             RBI  3.4640195
HmRun         HmRun  3.3152274
Hits           Hits  2.7759473
Errors       Errors  2.7068833
Runs           Runs  1.9878125
NewLeague NewLeague  0.6899792
Division   Division  0.3586048
League       League  0.2668856
```

#CAtBat #g)bagging

```
set.seed(0)
bag.Hitters <- randomForest(Salary ~ ., data = Hitters,
                            subset = train, mtry = 19, #m = p Bagging
                            importance = TRUE)
yhat.bag <- predict(bag.Hitters, newdata = Hitters[-train, ])
mean((yhat.bag - Salary.test)^2)
```

```
[1] 0.231094
```

#11. This question uses the Caravan data set.

```
train <- 1:1000

Caravan.train <- Caravan[train, ]
Caravan.test <- Caravan[-train, ]
Purchase.train <- Caravan$Purchase[train]
Purchase.test <- Caravan$Purchase[-train]
```

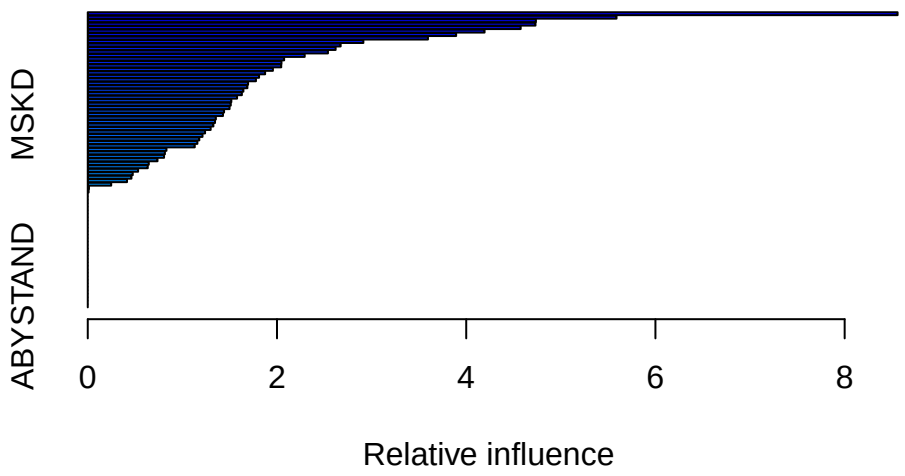
#Fit a boosting model #PPERSAUT #c)

```
set.seed(0)
Caravan.train$Purchase_binary <- ifelse(Caravan.train$Purchase == 'Yes', 1, 0)
Caravan.test$Purchase_binary <- ifelse(Caravan.test$Purchase == 'Yes', 1, 0)


boost.Caravan <- gbm(Purchase_binary ~ . - Purchase,
                     data = Caravan.train,
                     distribution = "bernoulli",  # Binary classification
                     n.trees = 1000,
                     shrinkage = 0.01,
                     interaction.depth = 3,
                     verbose = FALSE)
```

```
Warning in gbm.fit(x = x, y = y, offset = offset, distribution = distribution,
: variable 50: PVRAAUT has no variation.
```

```
Warning in gbm.fit(x = x, y = y, offset = offset, distribution = distribution,
: variable 71: AVRAAUT has no variation.
```

```
summary(boost.Caravan)
```




```
              var     rel.inf
PPERSAUT PPERSAUT 8.559366480
MOPLHOOG MOPLHOOG 5.588605772
MGODGE     MGODGE 4.736510405
MKOOPKLA MKOOPKLA 4.731925295
MOSTYPE   MOSTYPE 4.575630754
PBRAND     PBRAND 4.194034578
MBERMIDD MBERMIDD 3.893408110
MGODPR     MGODPR 3.595477164
MINK3045 MINK3045 2.912898077
MAUT2       MAUT2 2.672839143
MSKB1       MSKB1 2.619791842
MBERARBG MBERARBG 2.538503714
MSKC         MSKC 2.292874167
PWAPART   PWAPART 2.075451988
MINK7512 MINK7512 2.047251632
MRELOV     MRELOV 2.044953821
MSKA         MSKA 1.956965604
MRELGE     MRELGE 1.874625026
```

```
MFWEKIND MFWEKIND 1.810524710
MGODOV     MGODOV 1.777569258
MAUT1       MAUT1 1.694528233
MZFONDS   MZFONDS 1.687162589
MBERARBO MBERARBO 1.644868596
ABRAND     ABRAND 1.628784299
MBERHOOG MBERHOOG 1.578188653
MFGEKIND MFGEKIND 1.517399257
MINKM30   MINKM30 1.511430953
MAUT0       MAUT0 1.498898314
MSKB2       MSKB2 1.440769221
MSKD         MSKD 1.428892089
MFALLEEN MFALLEEN 1.353939726
MINKGEM   MINKGEM 1.345102775
MGODRK     MGODRK 1.327682613
MHHUUR     MHHUUR 1.298948819
MINK4575 MINK4575 1.239900631
MOPLMIDD MOPLMIDD 1.212782021
MZPART     MZPART 1.179056520
MHKOOP     MHKOOP 1.158664140
MGEMLEEF MGEMLEEF 1.130830881
APERSAUT APERSAUT 0.831703208
MBERZELF MBERZELF 0.815477038
MGEMOMV   MGEMOMV 0.806510187
MRELSA     MRELSA 0.736946636
PMOTSCO   PMOTSCO 0.644761595
MOSHOOFD MOSHOOFD 0.634029920
PBYSTAND PBYSTAND 0.532000245
PLEVEN     PLEVEN 0.475464645
MBERBOER MBERBOER 0.461721936
MOPLLAAG MOPLLAAG 0.414838476
MINK123M MINK123M 0.247837973
MAANTHUI MAANTHUI 0.012749400
ALEVEN     ALEVEN 0.008920872
PWABEDR   PWABEDR 0.000000000
PWALAND   PWALAND 0.000000000
PBESAUT   PBESAUT 0.000000000
PVRAAUT   PVRAAUT 0.000000000
PAANHANG PAANHANG 0.000000000
PTRACTOR PTRACTOR 0.000000000
PWERKT     PWERKT 0.000000000
PBROM       PBROM 0.000000000
PPERSONG PPERSONG 0.000000000
```

```
PGEZONG   PGEZONG 0.000000000
PWAOREG   PWAOREG 0.000000000
PZEILPL   PZEILPL 0.000000000
PPLEZIER PPLEZIER 0.000000000
PFIETS     PFIETS 0.000000000
PINBOED   PINBOED 0.000000000
AWAPART   AWAPART 0.000000000
AWABEDR   AWABEDR 0.000000000
AWALAND   AWALAND 0.000000000
ABESAUT   ABESAUT 0.000000000
AMOTSCO   AMOTSCO 0.000000000
AVRAAUT   AVRAAUT 0.000000000
AAANHANG AAANHANG 0.000000000
ATRACTOR ATRACTOR 0.000000000
AWERKT     AWERKT 0.000000000
ABROM       ABROM 0.000000000
APERSONG APERSONG 0.000000000
AGEZONG   AGEZONG 0.000000000
AWAOREG   AWAOREG 0.000000000
AZEILPL   AZEILPL 0.000000000
APLEZIER APLEZIER 0.000000000
AFIETS     AFIETS 0.000000000
AINBOED   AINBOED 0.000000000
ABYSTAND ABYSTAND 0.000000000
```

```r
yhat.boost_prob <- predict(boost.Caravan,
                           newdata = Caravan.test,
                           n.trees = 1000,
                           type = "response")  # 'response' for probabilities


yhat.boost <- ifelse(yhat.boost_prob > 0.2, 'Yes', 'No')


conf_matrix <- table(yhat.boost, Caravan.test$Purchase)
print(conf_matrix)
```

```
yhat.boost   No  Yes
       No  4351  252
       Yes  182   37
```

#What fraction of the people predicted to make a purchase do in fact make one? 37/(182+37) =0.169 >0.145.(KNN in lab4, but the training and testing set are different)