Lab: Decision Tree

In [1]:
```
library(ISLR)
```

In [2]:
```
library(tree)
```

In [3]:
```
remove(Carseats)
```

```
Warning message in remove(Carseats):
“object 'Carseats' not found”
```

In [4]:
```
attach(Carseats)
```

In [5]:
```
names(Carseats)
```

'Sales' 'CompPrice' 'Income' 'Advertising' 'Population' 'Price' 'ShelveLoc' 'Age' 'Education' 'Urban' 'US'

In [6]:
```
#ifelse() function to create a variable, called High,
#which takes on a value of Yes if the Sales variable exceeds 8,
#and takes on a value of No otherwise.
#recording sales as binary data from continous data

High=ifelse(Sales<=8,"No","Yes")
Carseats =data.frame(Carseats ,High)
```

In [7]:
```
names(Carseats)
```

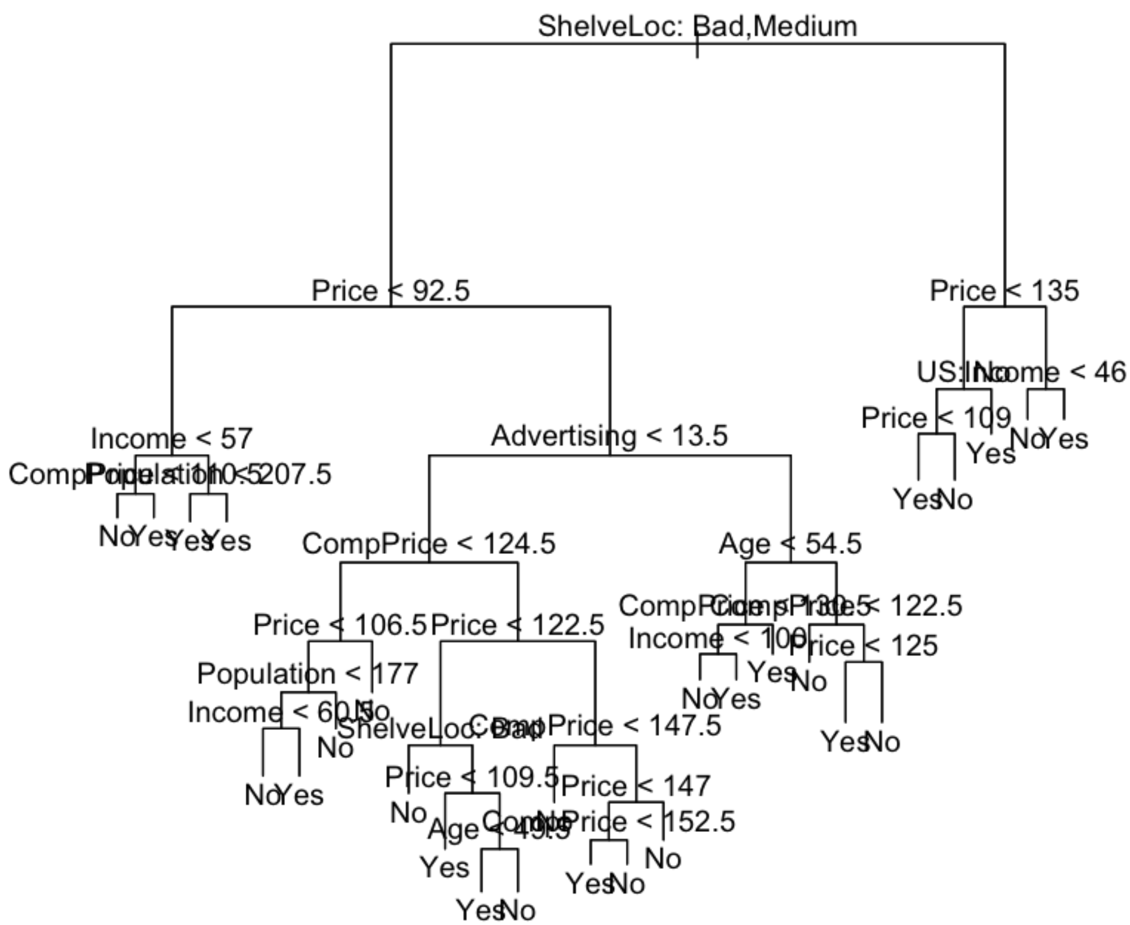
'Sales' 'CompPrice' 'Income' 'Advertising' 'Population' 'Price' 'ShelveLoc' 'Age' 'Education' 'Urban' 'US' 'High'

In [8]:
```
?Carseats
```

In [9]:
```
#We can fit a classification tree using tree() function
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"   "Populati
on"
[6] "Advertising" "Age"         "US"
Number of terminal nodes:  27
Residual mean deviance:  0.4575 = 170.7 / 373
Misclassification error rate: 0.09 = 36 / 400
```

In [10]:
```
#from summary-training error is 9%, residual mean deviance is 373
plot(tree.carseats)
text(tree.carseats,pretty=0)  #pretty=0 include the catg names for
```



In [11]:
```
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 0.8571
4 ) *
                   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 ) *
```

In [12]:

```
set.seed(4)
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)
 (89+52)/200
```

```
         High.test
tree.pred No Yes
      No  89  34
      Yes 25  52
```

0.705

In [13]:

```
summary(tree.carseats)
```

```
Classification tree:
tree(formula = High ~ . - Sales, data = Carseats, subset = train)
Variables actually used in tree construction:
[1] "ShelveLoc"   "Price"       "Advertising" "CompPrice"   "Age"
[6] "Population"  "Income"
Number of terminal nodes:  17
Residual mean deviance:  0.3521 = 64.43 / 183
Misclassification error rate: 0.075 = 15 / 200
```

In [14]:

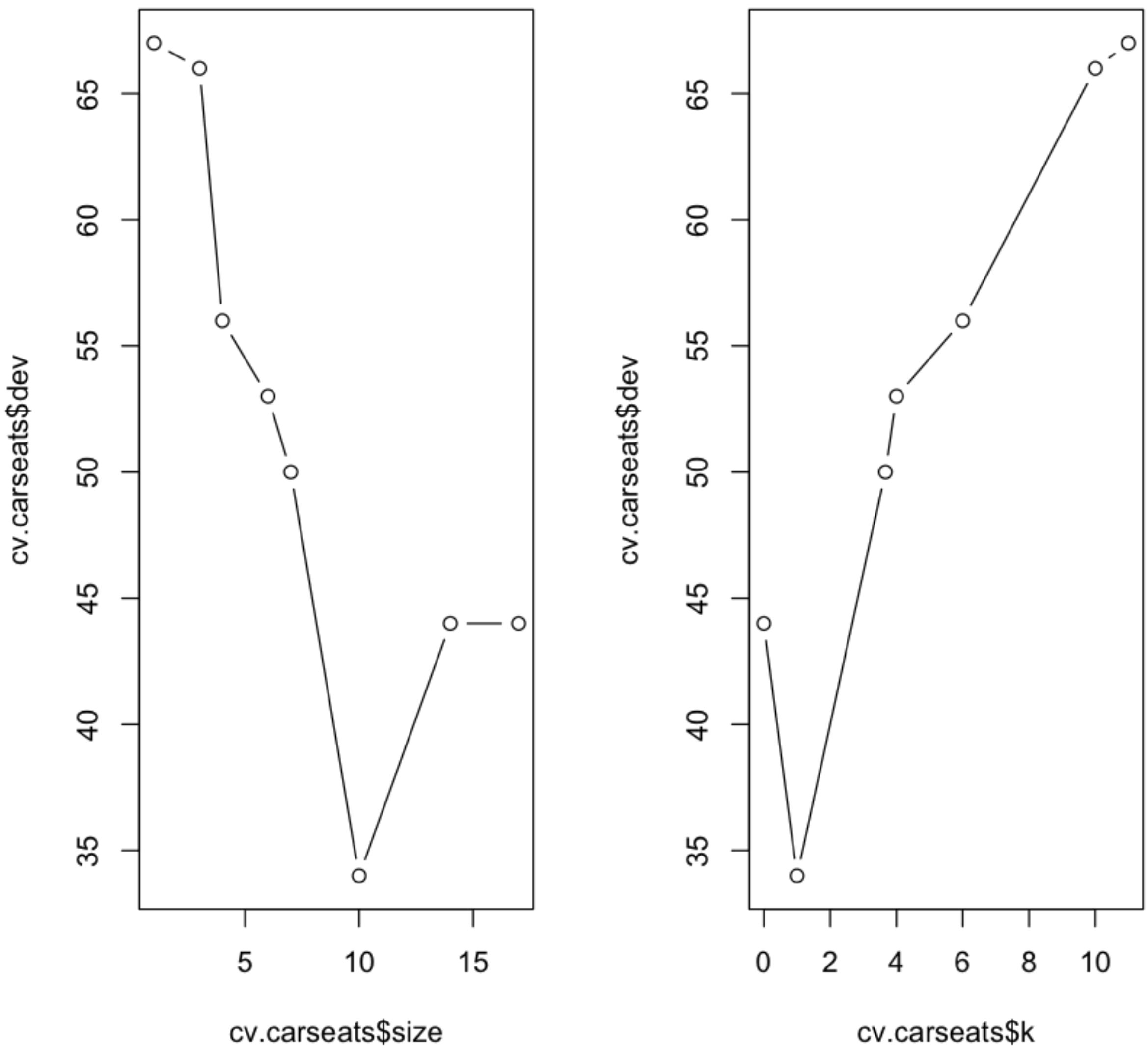
```
set.seed(3)
cv.carseats=cv.tree(tree.carseats,FUN=prune.misclass)
names(cv.carseats)
```

'size' 'dev' 'k' 'method'

In [15]:
```
cv.carseats
```

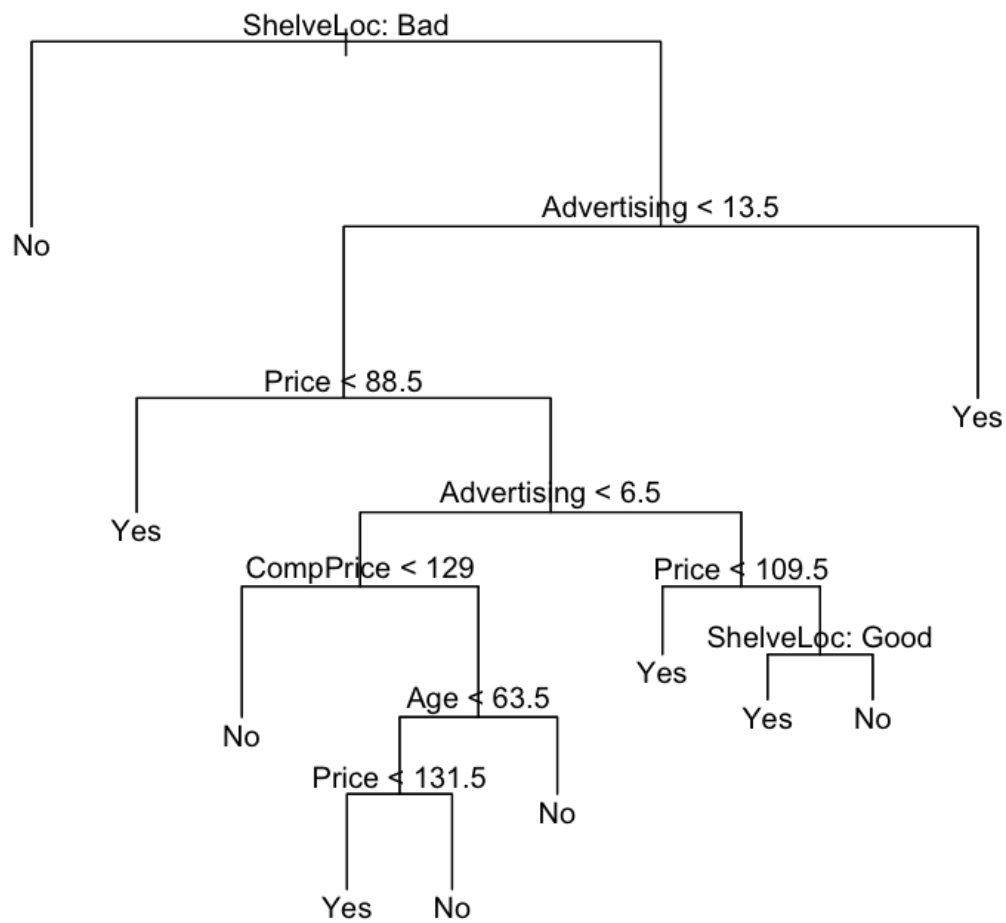
```
$size
[1] 17 14 10  7  6  4  3  1

$dev
[1] 44 44 34 50 53 56 66 67

$k
[1]      -Inf  0.000000  1.000000  3.666667  4.000000  6.000000 10.00
0000
[8] 11.000000

$method
[1] "misclass"

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

In [16]:
```
par(mfrow=c(1,2))
plot(cv.carseats$size, cv.carseats$dev,type="b")
plot(cv.carseats$k, cv.carseats$dev,type="b")
```



In [17]:

```
#prune the tree to obtain nine node tree
prune.carseats=prune.misclass(tree.carseats,best=10)
plot(prune.carseats)
text(prune.carseats,pretty=0)
```



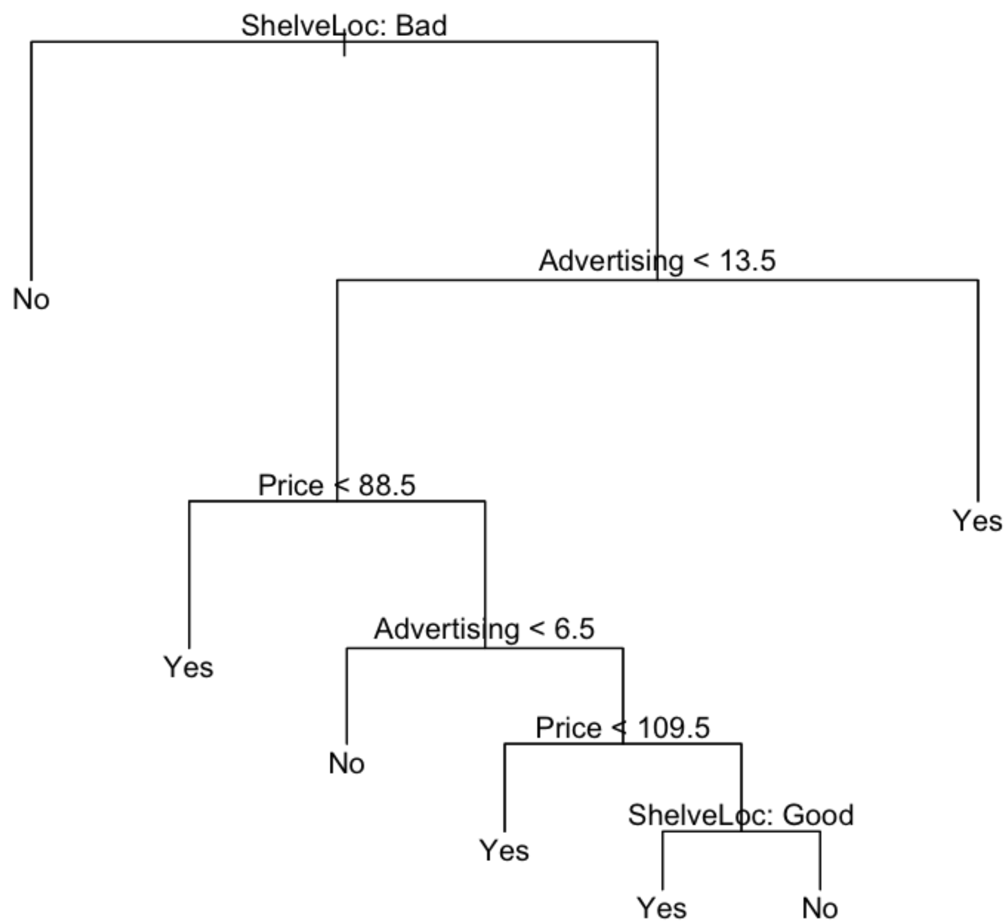
In [18]:

```
#to check how well pruned tree perform on test data, we apply pred
tree.pred=predict(prune.carseats,Carseats.test,type = "class")
table(tree.pred,High.test)
(92+58)/200
```

```
         High.test
tree.pred No Yes
      No  92  28
      Yes 22  58
```

0.75

In [19]:
```
prune.carseats=prune.misclass(tree.carseats,best=7)
plot(prune.carseats)
text(prune.carseats,pretty=0)

```



In [20]:
```
tree.pred=predict(prune.carseats,Carseats.test,type = "class")
table(tree.pred,High.test)
(97+49)/200
```

```
         High.test
tree.pred No  Yes
      No  97  37
      Yes 17  49
```

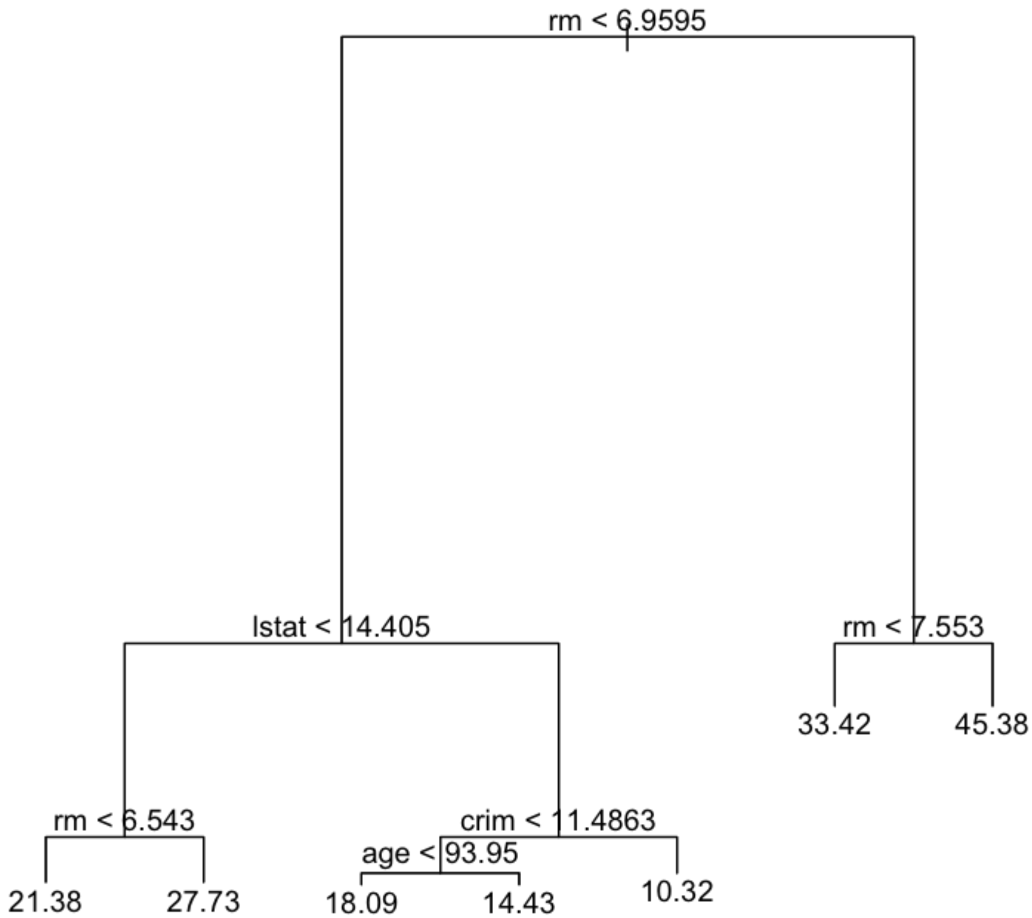
0.73

In [21]:
```
#Fitting regression tree
library(MASS)
```

In [22]:
```
#creating training set and fit tree to the training data
set.seed(1)
train=sample(1:nrow(Boston),nrow(Boston)/2)
tree.boston=tree(medv~.,Boston,subset=train)
summary(tree.boston)
```

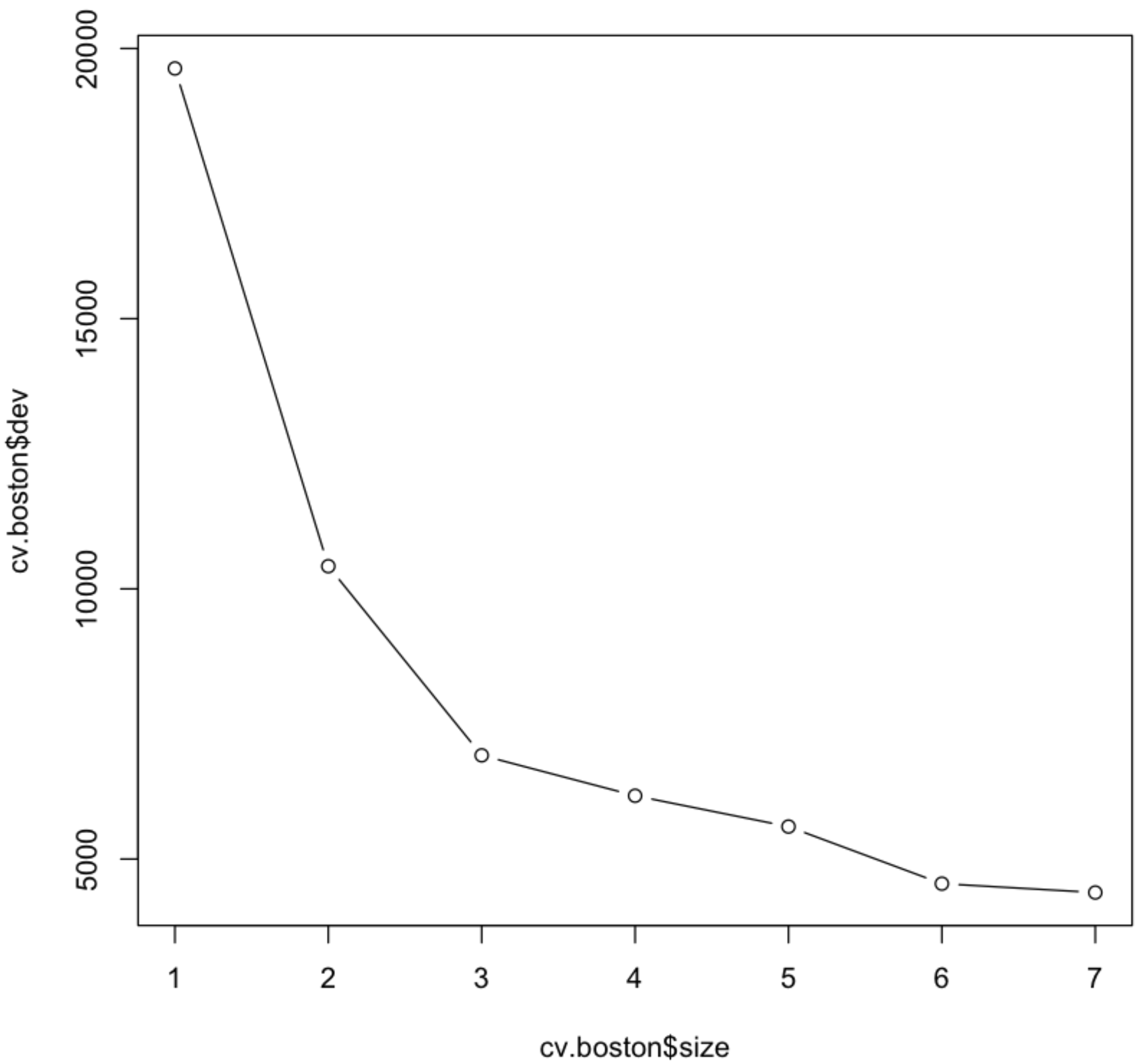
```
Regression tree:
tree(formula = medv ~ ., data = Boston, subset = train)
Variables actually used in tree construction:
[1] "rm"    "lstat" "crim"  "age"
Number of terminal nodes:  7
Residual mean deviance:  10.38 = 2555 / 246
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.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

In [23]:
```
plot(tree.boston)
text(tree.boston,pretty=0)
```



In [24]:

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

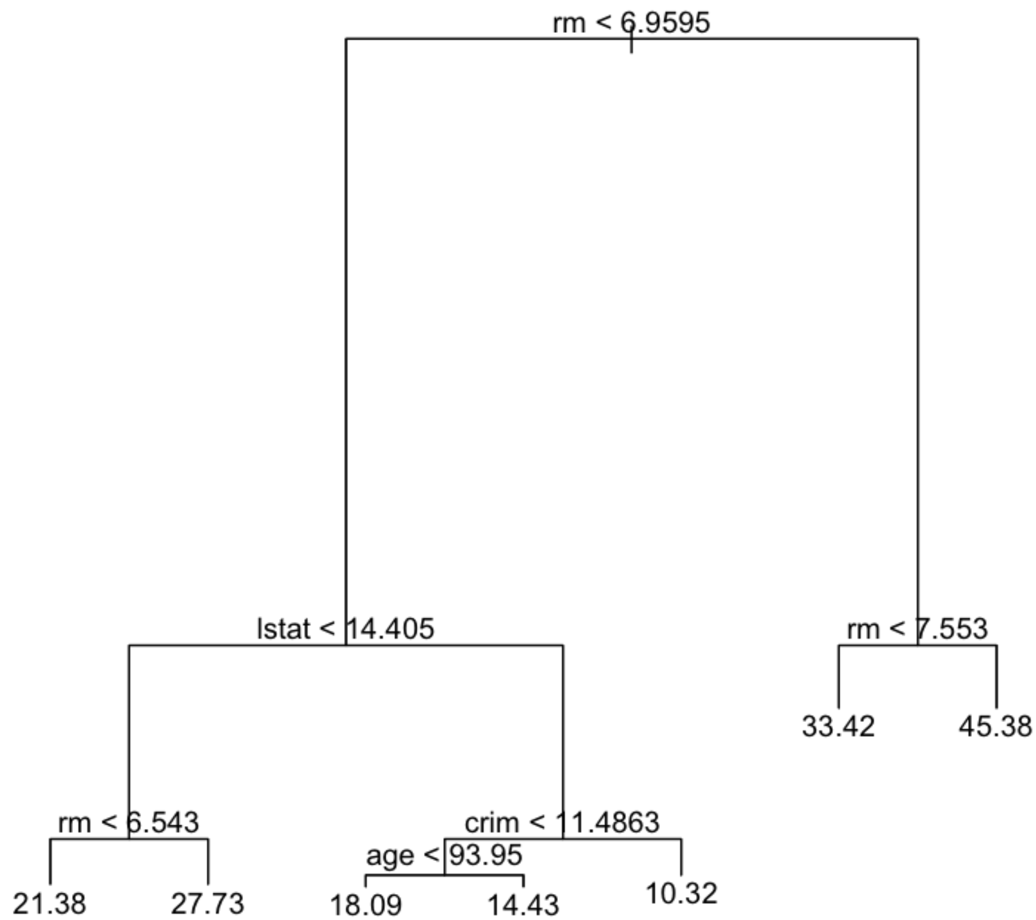


In [25]:
```
cv.boston
```

```
$size
[1] 7 6 5 4 3 2 1

$dev
[1]  4380.849  4544.815  5601.055  6171.917  6919.608 10419.472 19630
.870

$k
[1]       -Inf   203.9641   637.2707   796.1207  1106.4931  3424.7810
10724.5951

$method
[1] "deviance"

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

In [26]:

```
prune.boston=prune.tree(tree.boston,best=7)
plot(prune.boston)
text(prune.boston,pretty=0)
```



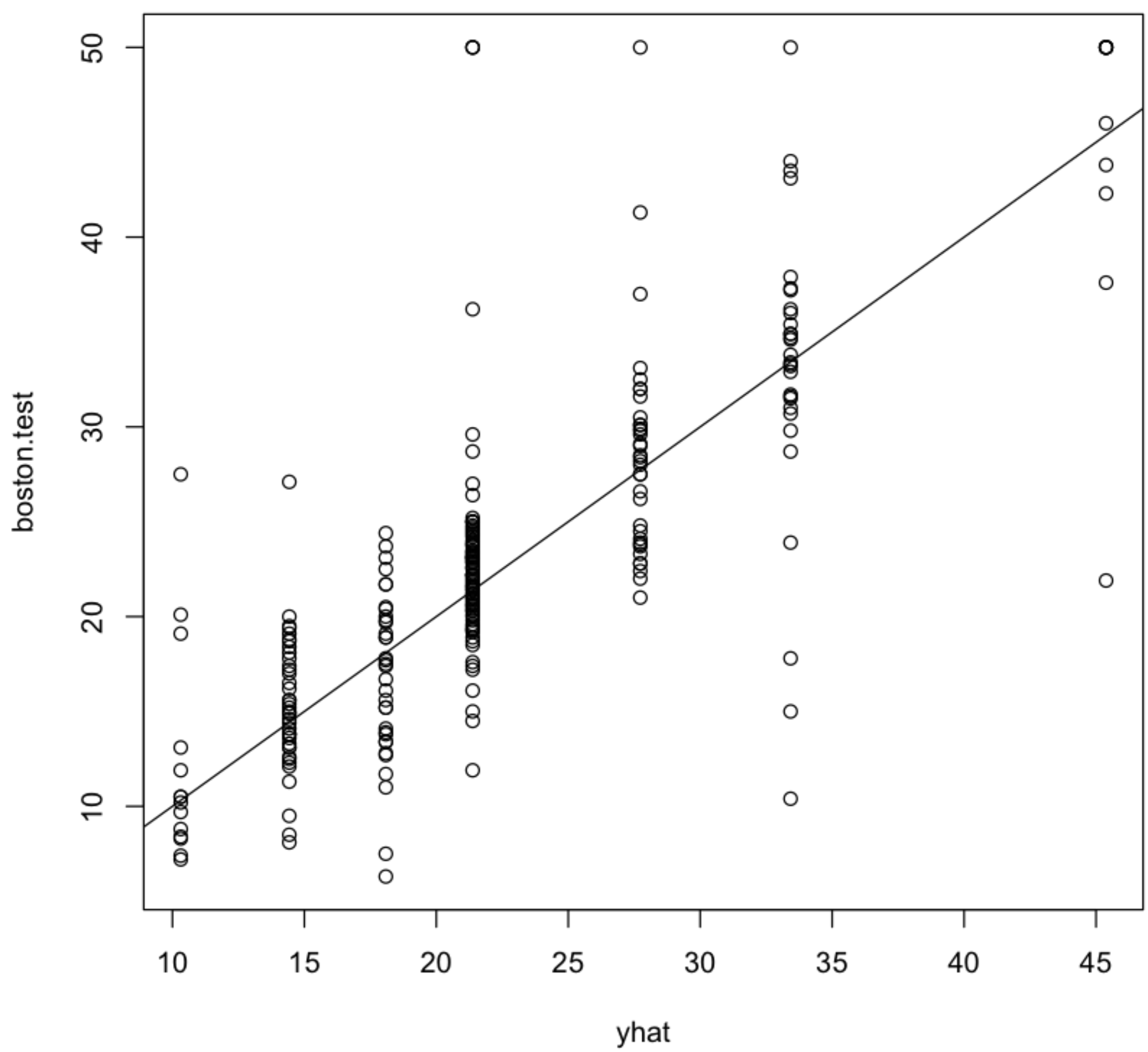
In [27]:
```
summary(prune.boston)
```

```
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 
```

In [28]:

```
yhat=predict(tree.boston,newdata=Boston[-train,])
boston.test=Boston[-train,"medv"]
plot(yhat,boston.test)
abline(0,1)
mean((yhat-boston.test)^2)
```

35.2868818594623



Bagging and Random Forests

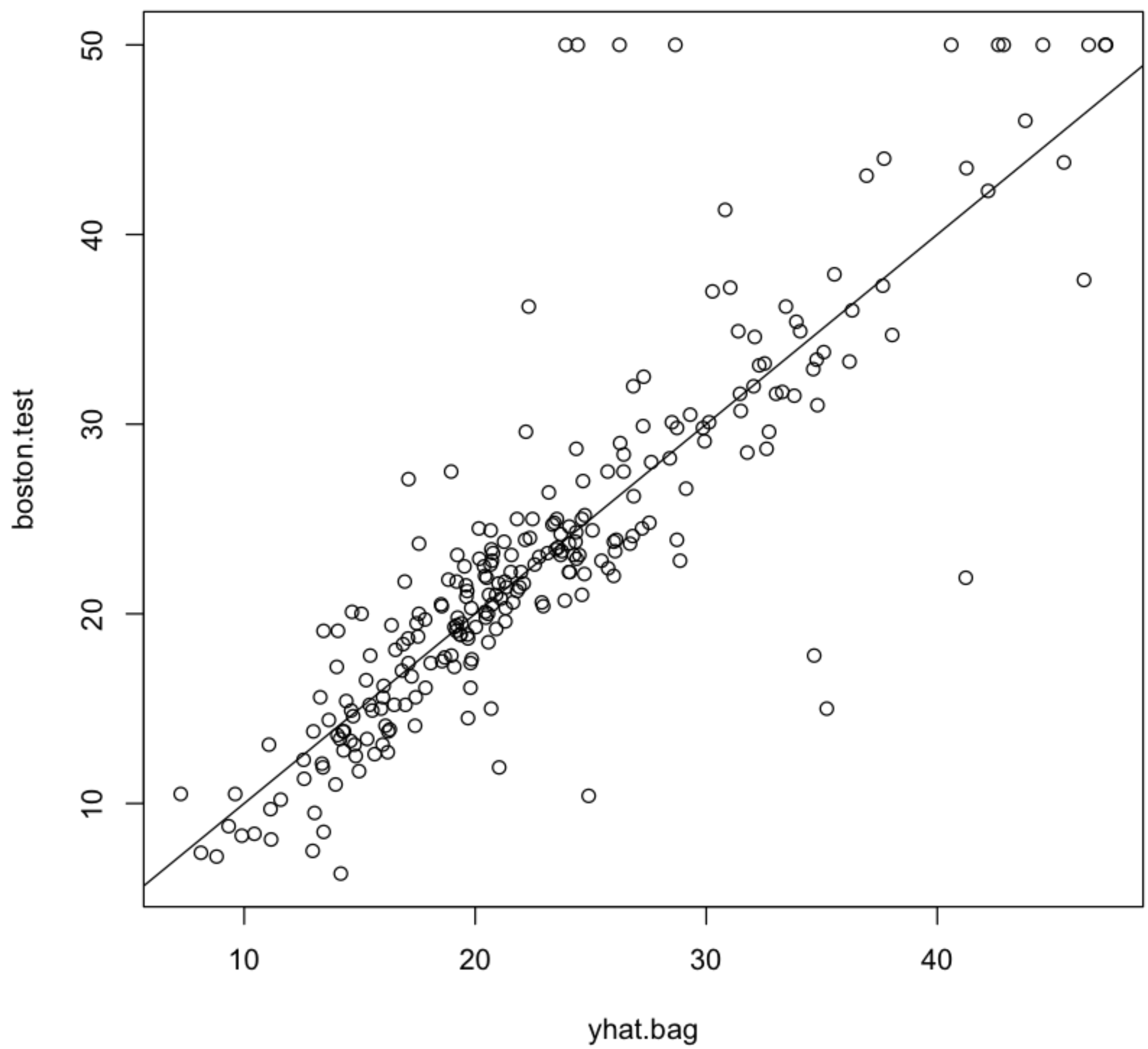
In [29]:

```
library(randomForest)
set.seed(1)
bag.boston=randomForest(medv~.,data=Boston,subset=train,mtry=13,im
bag.boston
```

```
randomForest 4.6-14
Type rfNews() to see new features/changes/bug fixes.


Call:
 randomForest(formula = medv ~ ., data = Boston, mtry = 13, importanc
e = TRUE,      subset = train)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 13

          Mean of squared residuals: 11.39601
                    % Var explained: 85.17
```

In [30]:
```
yhat.bag=predict(bag.boston,newdata=Boston[-train,])
plot(yhat.bag,boston.test)
abline(0,1)
mean((yhat.bag-boston.test)^2)
```

23.5927297079061



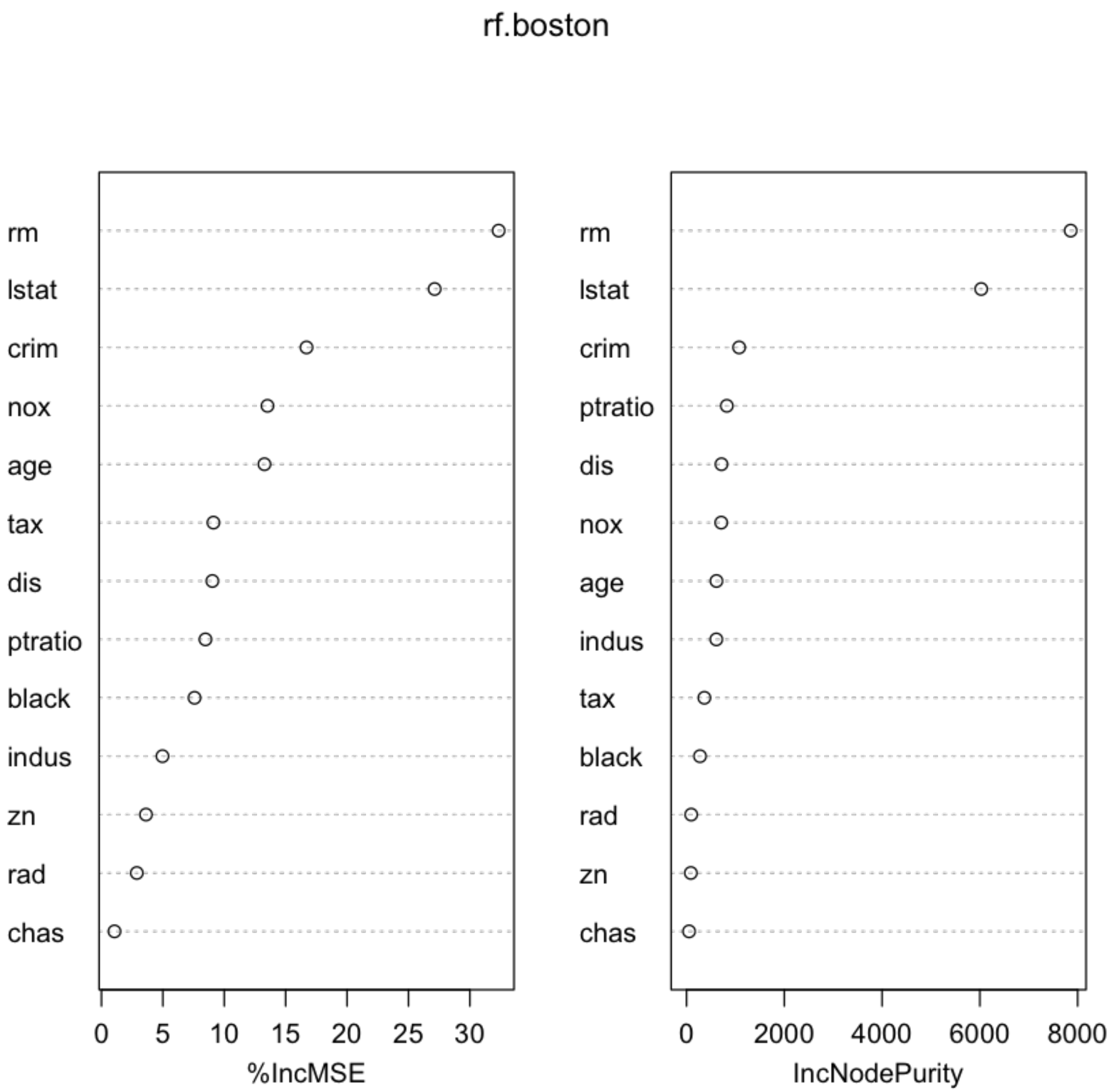
In [31]:
```
set.seed(1)
rf.boston=randomForest(medv~.,data=Boston,subset=train,mtry=6,impo
yhat.rf=predict(rf.boston,newdata=Boston[-train,])
mean((yhat.rf-boston.test)^2)
```

19.6202073910648

In [32]:
```
importance(rf.boston)
```

	%IncMSE	IncNodePurity
crim	16.697017	1076.08786
zn	3.625784	88.35342
indus	4.968621	609.53356
chas	1.061432	52.21793
nox	13.518179	709.87339
rm	32.343305	7857.65451
age	13.272498	612.21424
dis	9.032477	714.94674
rad	2.878434	95.80598
tax	9.118801	364.92479
ptratio	8.467062	823.93341
black	7.579482	275.62272
lstat	27.129817	6027.63740

In [33]:
```
varImpPlot(rf.boston)
```



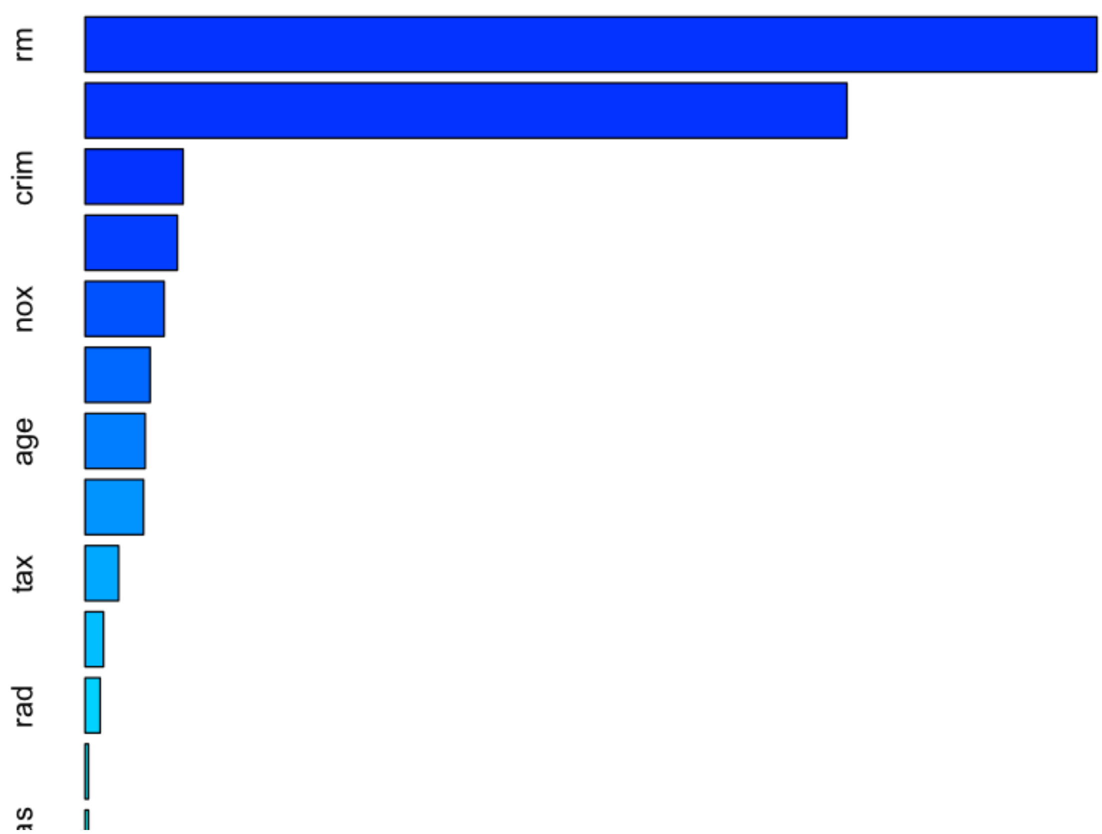
#Boosting

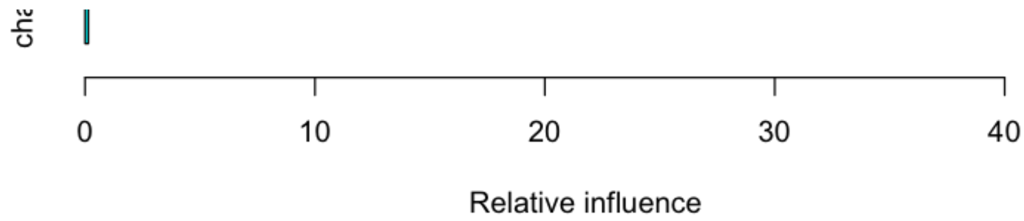
In [34]:
```
library(gbm)

```

Loaded gbm 2.1.5

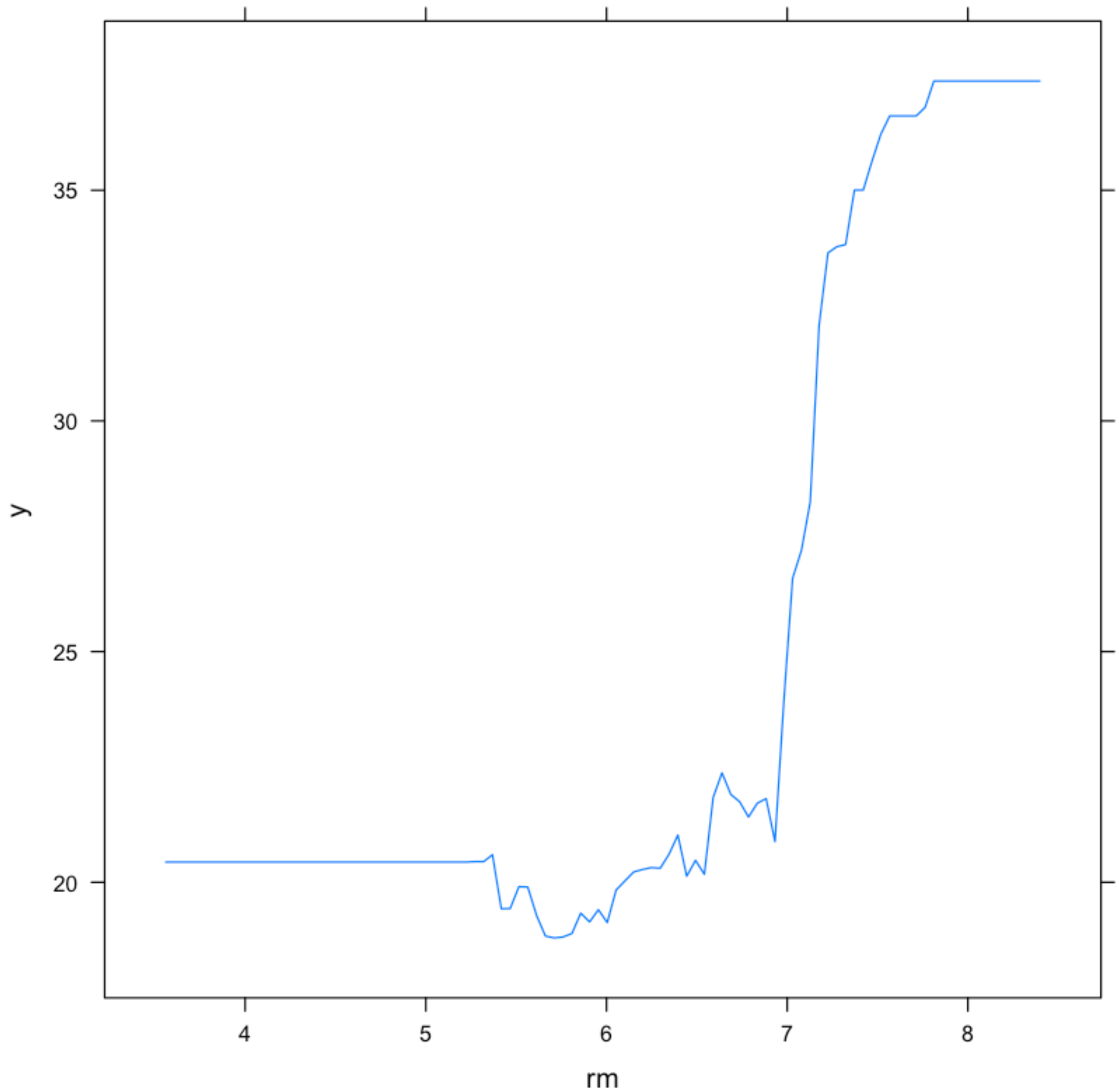
In [35]:
```
set.seed(1)
boost.boston=gbm(medv~.,data=Boston[train,],distribution="gaussian
summary(boost.boston)
```

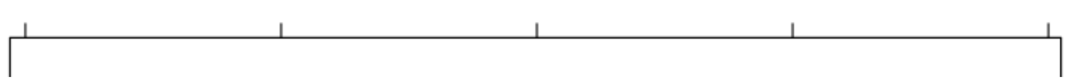
	var	rel.inf
rm	rm	43.9919329
lstat	lstat	33.1216941
crim	crim	4.2604167
dis	dis	4.0111090
nox	nox	3.4353017
black	black	2.8267554
age	age	2.6113938
ptratio	ptratio	2.5403035
tax	tax	1.4565654
indus	indus	0.8008740
rad	rad	0.6546400
zn	zn	0.1446149
chas	chas	0.1443986

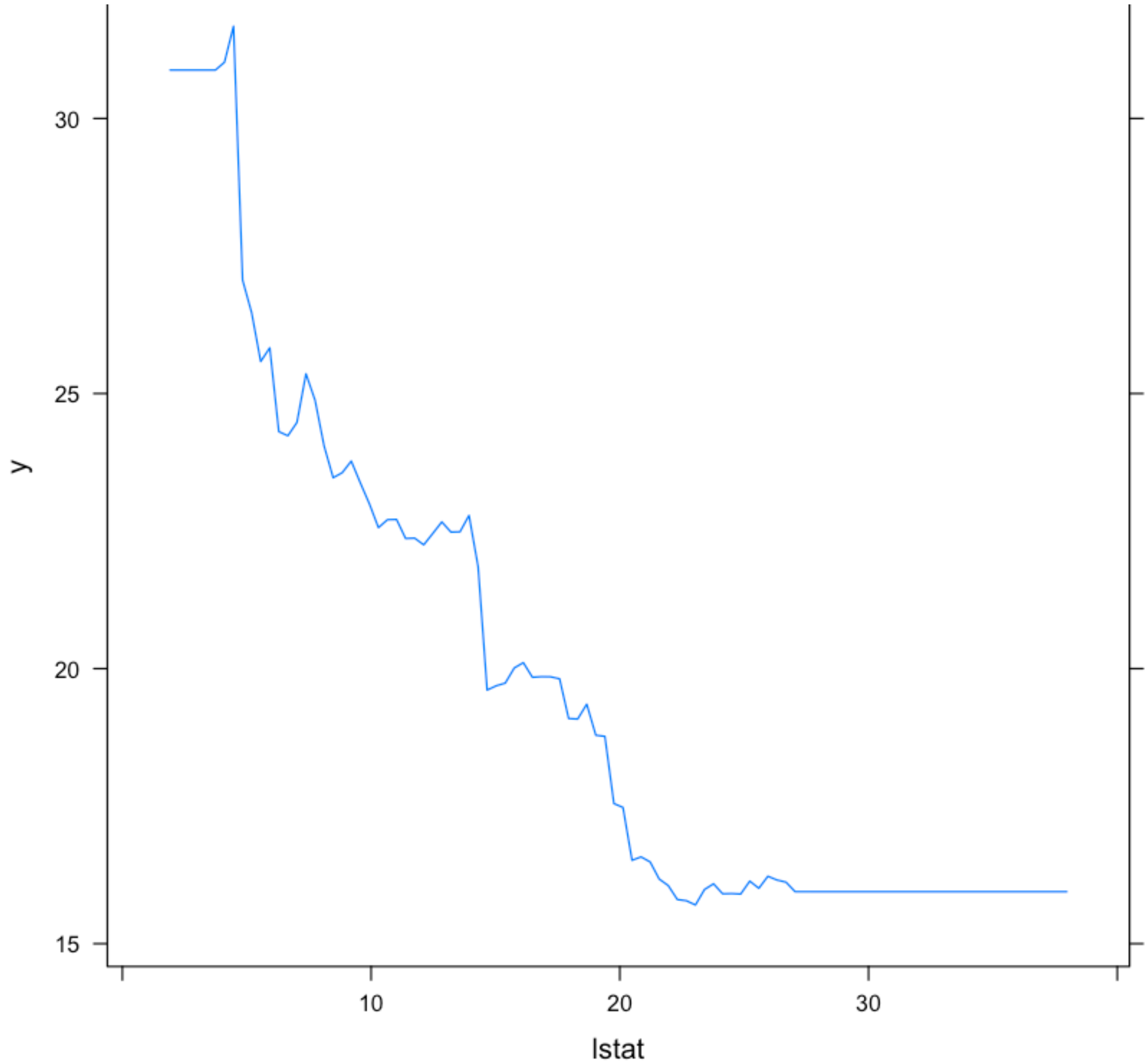




In [36]:
```
par(mfrow=c(1,2))
plot(boost.boston,i="rm")
plot(boost.boston,i="lstat")
```







In [37]:
```
yhat.boost=predict(boost.boston,newdata=Boston[-train,], n.trees=5
mean((yhat.boost-boston.test)^2)
```

18.8470923079959

In [38]:
```
boost.boston=gbm(medv~.,data=Boston[train,],distribution="gaussian
yhat.boost=predict(boost.boston,newdata=Boston[-train,],n.trees=50
mean((yhat.boost-boston.test)^2)
```

18.3345451839923