Assignment 6 - Tree-based approaches ¶

Conceptual

1. Explain the differences between bagging and boosting and explain how machine learning methods in general use these strategies to build robust models using tree models?

---*Your answer here*---

2. What are the main differences between Random Forests and AdaBoost regarding how the base learners are trained, how samples are weighted, how predictions are made and how robust are they regarding overfitting?

---*Your answer here*---

Practical

Overview of the steps

1. Load the data and get an overview of the data
2. Learn and assess Classification Trees
3. Learn and assess Regression Trees
4. Learn and assess Regression Bagging (Trees) and Random Forests
5. Learn and assess Regression Boosting (Trees)

Steps in detail

Load the data and get an overview of the data

Load the data file `Carseats.rda` or `Carseats.csv` .

In these data, the `Sales` of carseats is a quantitative response variable. Get an overview of the variables here (https://rdrr.io/cran/ISLR/man/Carseats.html).

In [1]:
```
load(file = "../ISLR/data/Carseats.rda")
```

Display the number of predictors (including the response `Sales`) and their names:

In [2]:
```
dim(Carseats)[2]
names(Carseats)
```

11

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

Print a statistic summary of the predictors and the response `medv` :

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

```
     Sales          CompPrice       Income        Advertising    
 Min.   : 0.000   Min.   : 77   Min.   : 21.00   Min.   : 0.000  
 1st Qu.: 5.390   1st Qu.:115   1st Qu.: 42.75   1st Qu.: 0.000  
 Median : 7.490   Median :125   Median : 69.00   Median : 5.000  
 Mean   : 7.496   Mean   :125   Mean   : 68.66   Mean   : 6.635  
 3rd Qu.: 9.320   3rd Qu.:135   3rd Qu.: 91.00   3rd Qu.:12.000  
 Max.   :16.270   Max.   :175   Max.   :120.00   Max.   :29.000  
   Population        Price        ShelveLoc        Age          Educ
ation
 Min.   : 10.0   Min.   : 24.0   Bad   : 96   Min.   :25.00   Min.
:10.0
 1st Qu.:139.0   1st Qu.:100.0   Good  : 85   1st Qu.:39.75   1st Q
u.:12.0
 Median :272.0   Median :117.0   Medium:219   Median :54.50   Median
:14.0
 Mean   :264.8   Mean   :115.8                Mean   :53.32   Mean
:13.9
 3rd Qu.:398.5   3rd Qu.:131.0                3rd Qu.:66.00   3rd Q
u.:16.0
 Max.   :509.0   Max.   :191.0                Max.   :80.00   Max.
:18.0
 Urban       US
 No :118   No :142
 Yes:282   Yes:258
```

Display the number of data points:

In [4]:
```
dim(Carseats)[1]
```

400

Display the data in a table (subset of rows is sufficient):

In [5]:
```
1 Carseats
```

A data.frame: 400 × 11

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<fct>	<dbl>	<dbl>
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
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
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

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<fct>	<dbl>	<dbl>
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

Compute the pairwise correlation of the predictors in the data set.

In R , we need to download and install a library first.

In [6]:
```
install.packages("corrplot")
source("http://www.sthda.com/upload/rquery_cormat.r")
```

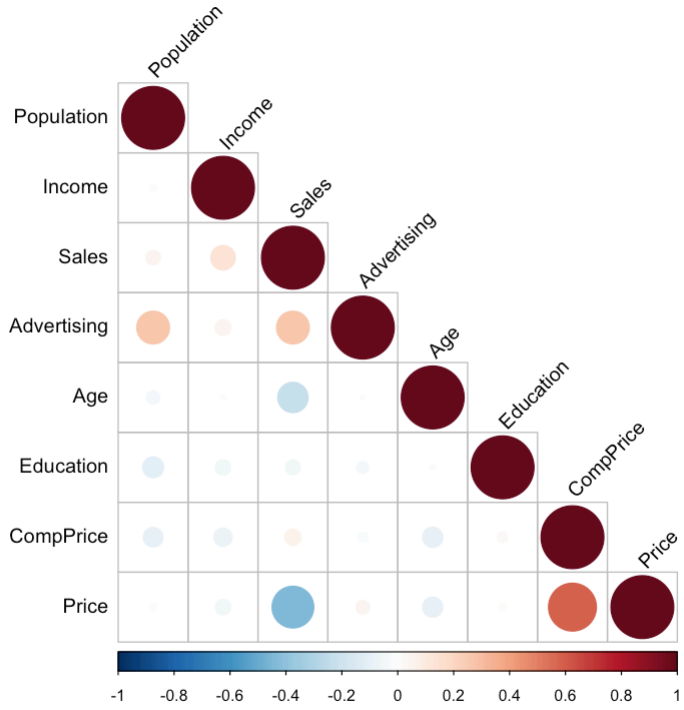
```
The downloaded binary packages are in
        /var/folders/ct/4pcck8t94sdfc73rhymq4t140000gp/T//RtmpCBK2W
j/downloaded_packages
```

In [7]:
```
CarseatsQuantitative = Carseats[,-(10:11)]
CarseatsQuantitative = CarseatsQuantitative[,-7]
names(CarseatsQuantitative)
rquery.cormat(CarseatsQuantitative)
```

'Sales' 'CompPrice' 'Income' 'Advertising' 'Population' 'Price' 'Age' 'Education'

```
corrplot 0.84 loaded
```

```
$r
            Population  Income  Sales Advertising    Age Education C
ompPrice
Population           1
Income         -0.0079       1
Sales             0.05    0.15      1
Advertising       0.27   0.059   0.27           1
Age             -0.043 -0.0047  -0.23     -0.0046      1
Education        -0.11  -0.057 -0.052      -0.034 0.0065         1
CompPrice       -0.095  -0.081  0.064      -0.024   -0.1     0.025
1
Price           -0.012  -0.057  -0.44       0.045   -0.1     0.012
0.58
            Price
Population
Income
Sales
Advertising
Age
Education
CompPrice
Price           1

$p
            Population Income   Sales Advertising   Age Education Co
mpPrice
Population           0
Income            0.88      0
Sales             0.31 0.0023       0
Advertising    6.9e-08   0.24 4.4e-08           0
Age               0.39   0.93 2.8e-06        0.93     0
Education        0.033   0.26     0.3         0.5   0.9         0
CompPrice        0.058   0.11     0.2        0.63 0.045      0.62
0
Price             0.81   0.26 7.6e-21        0.37 0.041      0.81
4.5e-38
            Price
Population
Income
Sales
Advertising
Age
Education
CompPrice
Price           0

$sym
            Population Income Sales Advertising Age Education CompPr
ice Price
Population  1
Income                1
Sales                       1
Advertising                       1
Age                                           1
Education                                         1
CompPrice                                                     1
Price                         .                               .
1
attr(,"legend")
[1] 0 ‘ ’ 0.3 ‘.’ 0.6 ‘,’ 0.8 ‘+’ 0.9 ‘*’ 0.95 ‘B’ 1
```



Plot the response to its most correlated predictor.

In R , we need to download and install a library first.

In [8]:
```
install.packages("rlang")
library(rlang)
sessionInfo()
```

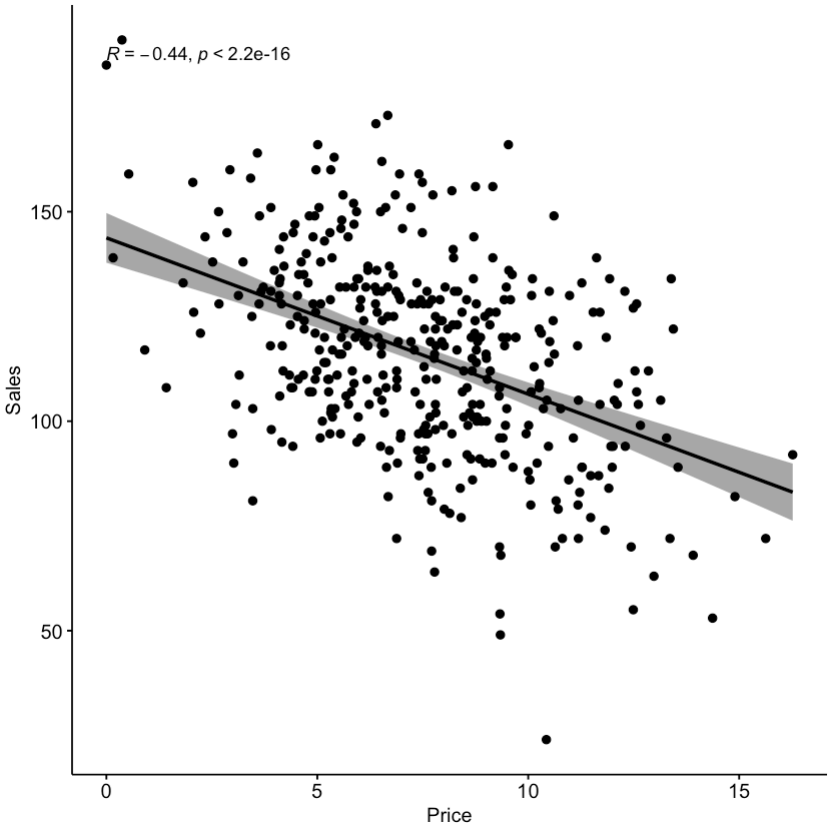
```
The downloaded binary packages are in
        /var/folders/ct/4pcck8t94sdfc73rhymq4t140000gp/T//RtmpCBK2W
j/downloaded_packages

R version 3.6.2 (2019-12-12)
Platform: x86_64-apple-darwin15.6.0 (64-bit)
Running under: macOS  10.16

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/l
ibRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/l
ibRlapack.dylib

locale:
[1] C/UTF-8/C/C/C/C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base
```

In [9]:
```
install.packages("ggpubr")
library("ggpubr")
```

```
The downloaded binary packages are in
        /var/folders/ct/4pcck8t94sdfc73rhymq4t140000gp/T//RtmpCBK2W
j/downloaded_packages
```

```
Loading required package: ggplot2
```

In [10]:
```
ggscatter(Carseats, x = "Sales", y = "Price",
          add = "reg.line", conf.int = TRUE,
          cor.coef = TRUE, cor.method = "pearson",
          xlab = "Price",
          ylab = "Sales")
```



Interprete the results. *Your interpretation of the results goes here!*

Learn and assess Classification Trees

Install the tree library.

In [11]:
```
install.packages("tree")
library(tree)
```

```
The downloaded binary packages are in
        /var/folders/ct/4pcck8t94sdfc73rhymq4t140000gp/T//RtmpCBK2W
j/downloaded_packages
```

Predict that the `Sales` is high using the predictors.

As `Sales` is a quantitative variable, we begin by recoding it as a binary variable.

In [12]:
```
High=ifelse(Carseats$Sales <=8,"No","Yes")
Carseats =data.frame(Carseats, High)
names(Carseats)
```

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

We now use the `tree()` function to fit a classification tree in order to predict `High` using all variables but `Sales` .

In `R` , the syntax of the `tree()` function is quite similar to that of the `lm()` function.

In [13]:
```
tree.carseats =tree(High~.-Sales, Carseats)
summary(tree.carseats)
```

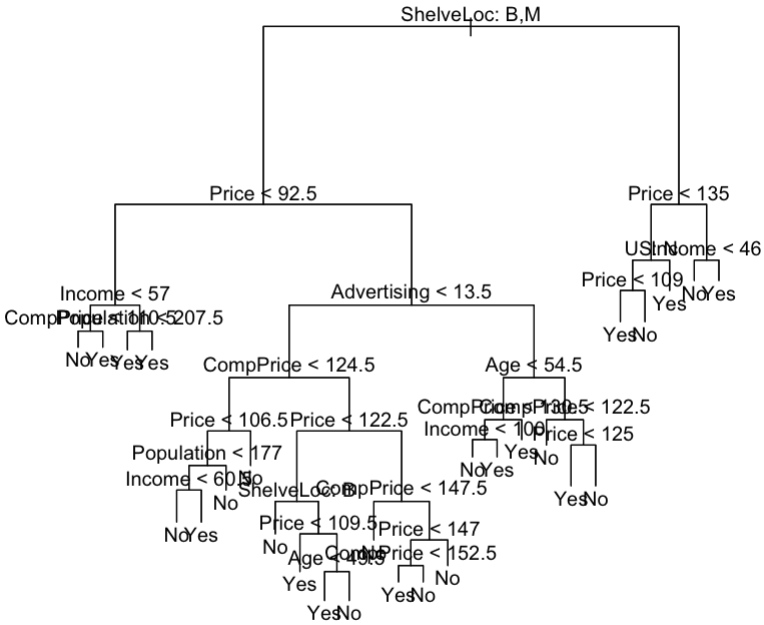
```
Classification tree:
tree(formula = High ~ . - Sales, data = Carseats)
Variables actually used in tree construction:
[1] "ShelveLoc"   "Price"       "Income"      "CompPrice"   "Populat
ion"
[6] "Advertising" "Age"         "US"
Number of terminal nodes:  27
Residual mean deviance:  0.4575 = 170.7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

The `Residual mean deviance` is defined as deviance $D = -2\sum_m \sum_k n_{mk} \log \hat{p}_{mk}$ divided by $n - |T_0|$. Here n_{mk} is the number of observations in the m-th terminal node that belong to the k-th class. A small deviance indicates a tree that provides a good fit to the (training) data.

One of the most attractive properties of trees is that they can be graphically displayed. So display the decision tree.

In `R` , use the `plot()` function to display the tree structure, and the `text()` function to display the node labels. The argument `pretty=0` instructs `text()` to include the category names for any qualitative predictors, rather than simply displaying a letter for each category.

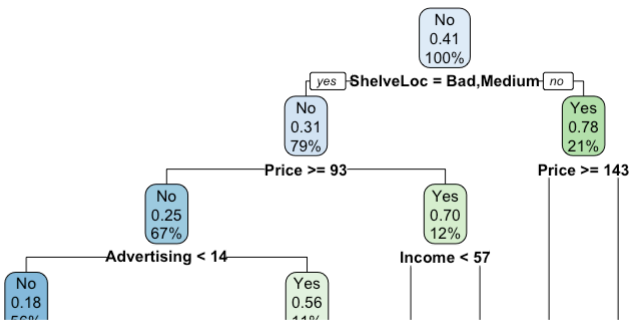
In [14]:
```
plot(tree.carseats)
text(tree.carseats, pretty =1)
```



Other libraries in R (or Python) produce other trees and (maybe nicer) visualizations. Never mind when your tree (visualization) is different form the solutions displayed here.

In [15]:
```
install.packages("rpart")
install.packages("rpart.plot")
library(rpart)
library(rpart.plot)
fit <- rpart(High~.-Sales, data = Carseats, method = 'class')

rpart.plot(fit)
```

```
The downloaded binary packages are in
        /var/folders/ct/4pcck8t94sdfc73rhymq4t140000gp/T//RtmpCBK2W
j/downloaded_packages

The downloaded binary packages are in
        /var/folders/ct/4pcck8t94sdfc73rhymq4t140000gp/T//RtmpCBK2W
j/downloaded_packages
```



Interprete the results. *Your interpretation of the results goes here!*

In order to properly evaluate the performance of a classification tree on these data, we must estimate the test error rather than simply computing the training error. We split the observations into a training set and a test set, build the tree using the training set, and evaluate its performance on the test data.

In `R`, the `predict()` function can be used for this purpose. In the case of a classification tree, the argument `type="class"` instructs `predict()` to return the actual class prediction. This approach leads to correct predictions for around 71.5% of the locations in the test data set.

In [67]:
```
set.seed(1)
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  84  37
      Yes 35  44
```

Interprete the results. *Your interpretation of the results goes here!*

Pruning the tree might lead to improved results.

The function `cv.tree()` performs cross-validation in order to determine the optimal level of tree complexity; cost complexity pruning is used in order to select a sequence of trees for consideration. We use the argument `FUN=prune.misclass` in order to indicate that we want the classification error rate to guide the cross-validation and pruning process, rather than the default for the `cv.tree()` function, which is deviance.

Prune the tree using cross validation.

In [104]:
```
set.seed(0)
cv.carseats = cv.tree(tree.carseats, FUN=prune.misclass)
cv.carseats
```

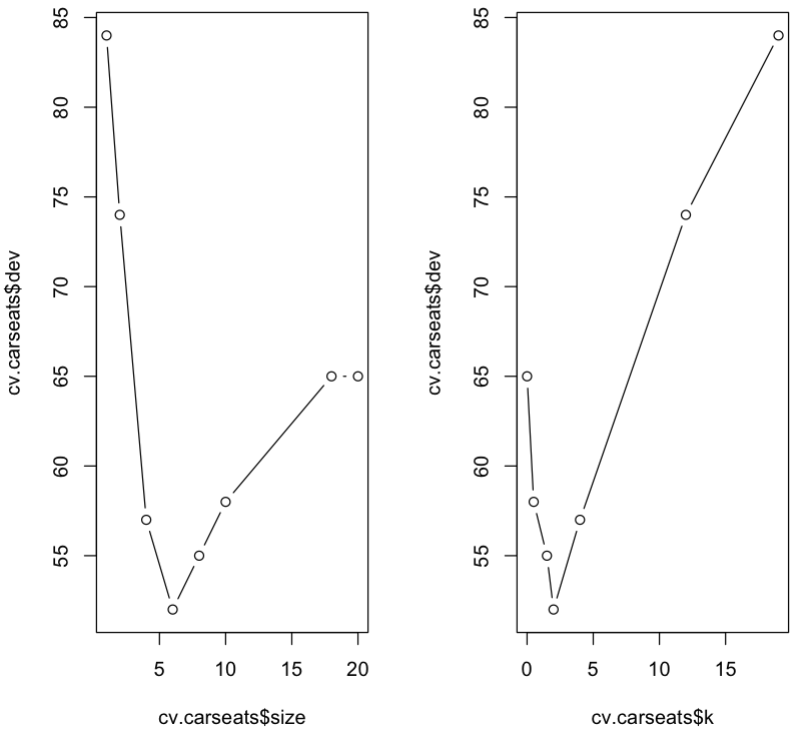
```
$size
[1] 20 18 10  8  6  4  2  1

$dev
[1] 65 65 58 55 52 57 74 84

$k
[1] -Inf  0.0  0.5  1.5  2.0  4.0 12.0 19.0

$method
[1] "misclass"

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

Note that, despite the name, `dev` corresponds to the cross-validation error rate in this instance and `k` corresponds to α.

Interprete the results. *Your interpretation of the results goes here!*

Consider plotting the error rate as a function of tree size and α (here called `k`). What is the tree with teh right size?

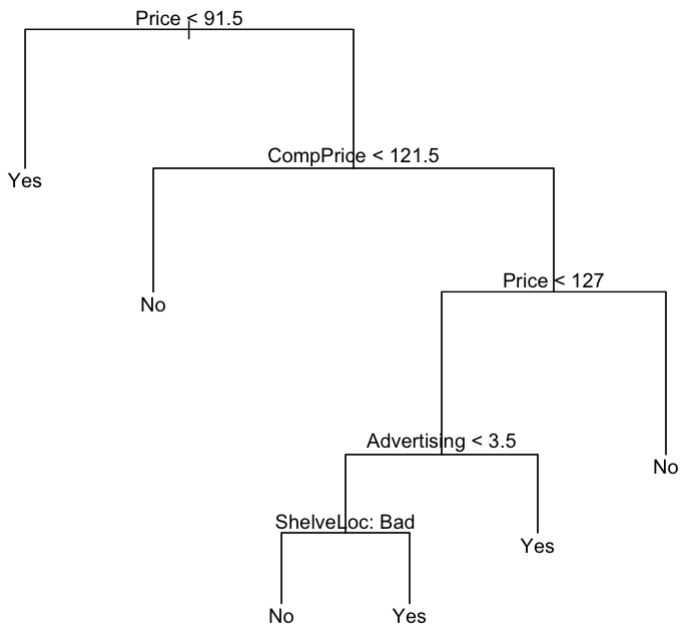
In [105]:
```
par(mfrow=c(1,2))
plot(cv.carseats$size ,cv.carseats$dev ,type="b")
plot(cv.carseats$k ,cv.carseats$dev ,type="b")
```



Interprete the results. *Your interpretation of the results goes here!*

Now we are ready to prune the tree according to our findings and plot the pruned tree.

In [106]:
```
prune.carseats=prune.misclass(tree.carseats,best=6)
plot(prune.carseats)
text(prune.carseats,pretty=0)
```



How well does this pruned tree perform on the test data set?

In R , we apply the `predict()` function once again.

In [107]:
```
tree.pred=predict(prune.carseats,Carseats.test,type="class")
table(tree.pred ,High.test)
```

```
         High.test
tree.pred No Yes
      No  86  32
      Yes 33  49
```

In [108]:
```
(86+49)/200
```

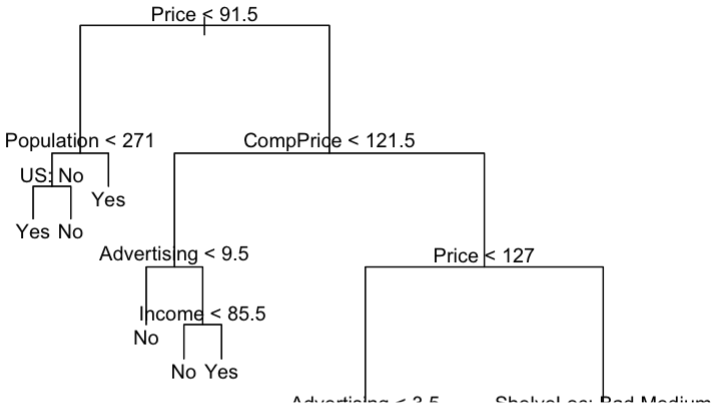
0.675

Now 67.5% of the test observations are correctly classified, so not only has the pruning process produced a more interpretable tree, but it has also improved the classification accuracy.

If we increase the value of best, we obtain a larger pruned tree with lower classification accuracy.

In [109]:
```
prune.carseats=prune.misclass(tree.carseats,best=18)
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  84  37
      Yes 35  44
```



In [110]:
```
(84+44)/(102+53+15+30)
```

0.64

Interprete the results. *Your interpretation of the results goes here!*

Learn and assess Regression Trees

Recall the `Boston` data set from Assignments 2 and 3. Check the details of this data set and load it.

In [111]:
```
load(file = "../ISLR/data/Boston.rda")
```

Create a training set, and fit the tree to the training data.

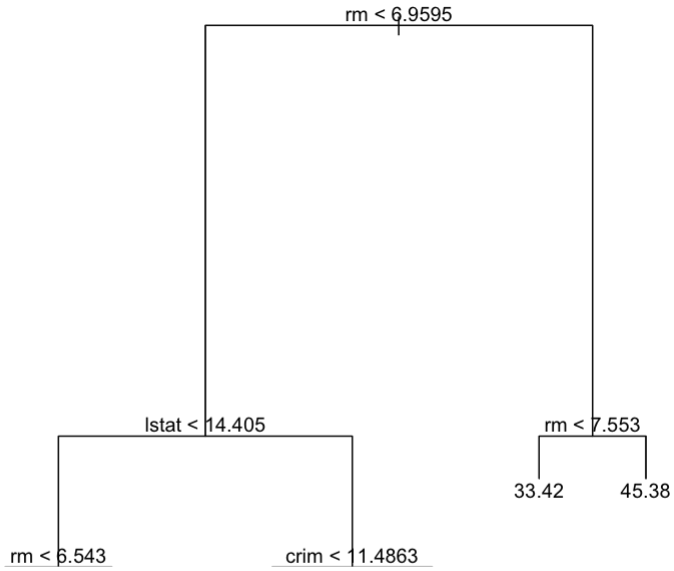
In [112]:
```
set.seed(1)
train = sample(1:nrow(Boston), nrow(Boston)/2)
tree.boston=tree(medv~.,Boston ,subset=train)
summary(tree.boston)
```

```
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
```

The output of `summary()` indicates that only three of the variables have been used in constructing the tree. In the context of a regression tree, the deviance `dev` is simply the sum of squared errors for the tree.

Plot the tree.

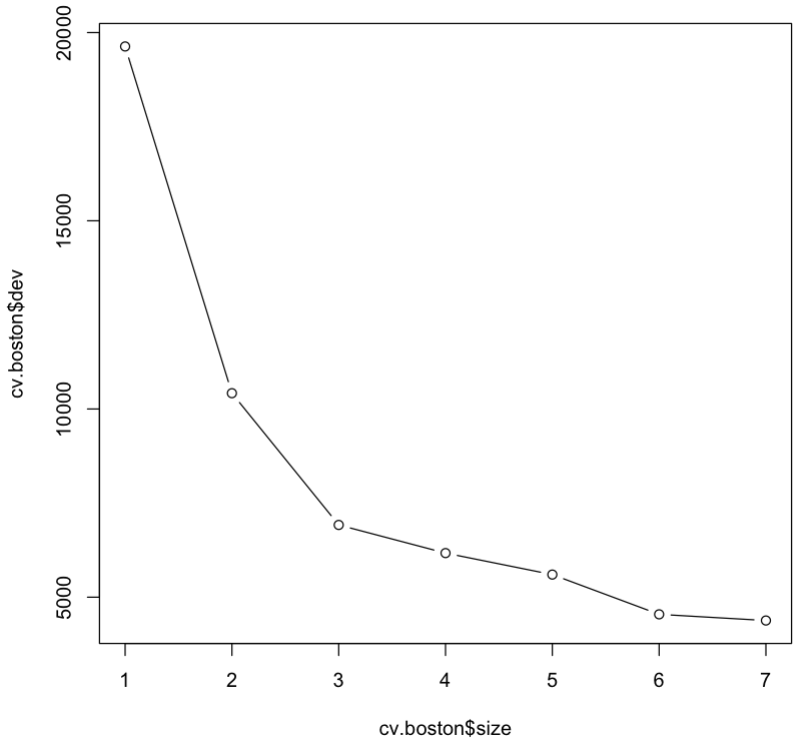
In [113]:
```
plot(tree.boston)
text(tree.boston ,pretty=0)
```



Interprete the results. *Your interpretation of the results goes here!*

Now use the cross-validation and pruning to see whether it will improve performance. Plot `size` (or α) against the error, e.g., `dev` .

In [114]:
```
cv.boston=cv.tree(tree.boston)
plot(cv.boston$size, cv.boston$dev, type='b')
```

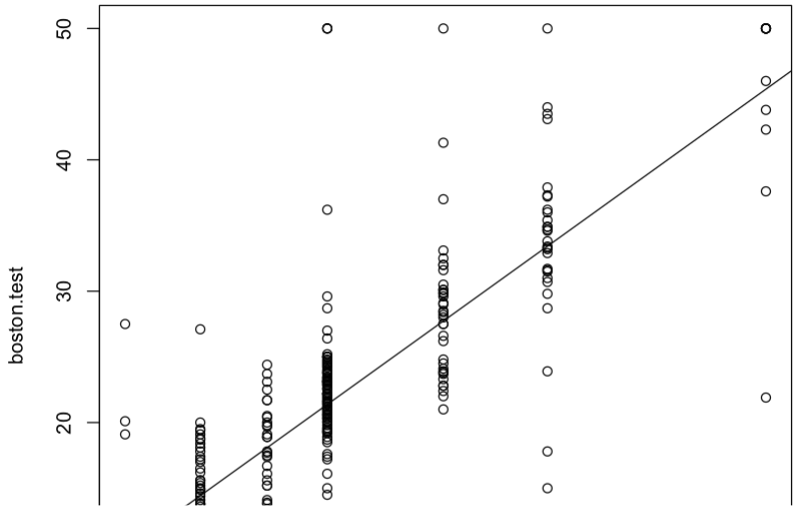


Interprete the results. *Your interpretation of the results goes here!*

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

In [115]:
```
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



Interprete the results. *Your interpretation of the results goes here!*

Learn and assess Regression Bagging (Trees) and Random Forests

Use the `Boston` data again.

In R we need to load the library `randomForest` .

In [91]:
```
install.packages("randomForest")
library(randomForest)
```

```
The downloaded binary packages are in
        /var/folders/ct/4pcck8t94sdfc73rhymq4t140000gp/T//RtmpCBK2W
j/downloaded_packages
```

Fit boosted regression trees to the `Boston` data set.

In [92]:
```
set.seed(1)
bag.boston=randomForest(medv~.,data=Boston,subset=train,mtry=13,im
bag.boston
```

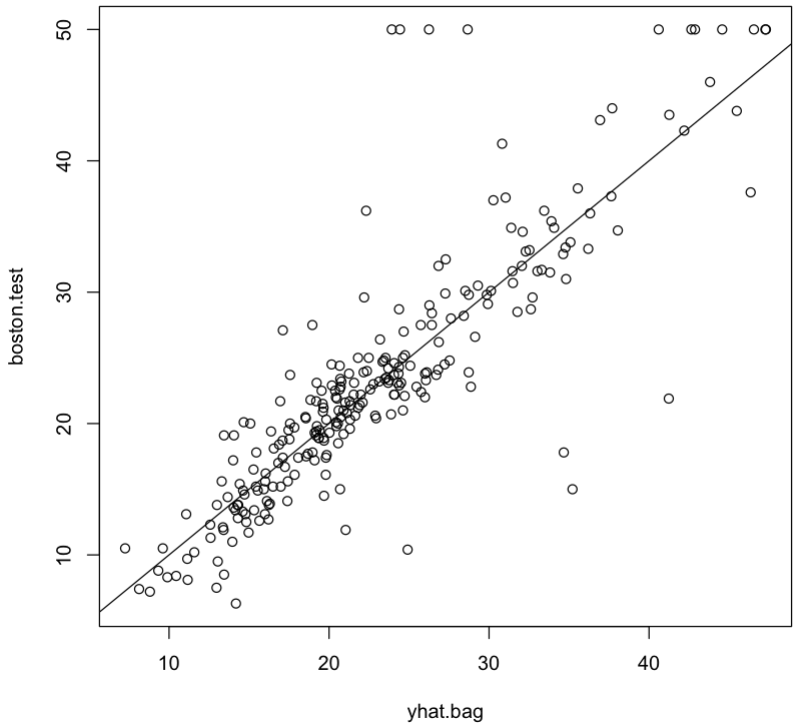
```
Call:
 randomForest(formula = medv ~ ., data = Boston, mtry = 13, importan
ce = TRUE,      subset = train)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 13

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

The argument `mtry=13` indicates that all $m = p = 13$ predictors should be considered for each split of the tree—in other words, that bagging should be done.

How well does this bagged model perform on the test set?

In [95]:
```
yhat.bag = predict(bag.boston,newdata=Boston[-train ,])
plot(yhat.bag, boston.test)
abline(0,1)
mean((yhat.bag-boston.test)^2)
```

23.5927297079061



Change the number of trees grown in the random forest.

In R , we use the `ntree` argument of `randomForest()` .

In [96]:
```
bag.boston=randomForest(medv~.,data=Boston,subset=train, mtry=13,r
bag.boston
yhat.bag = predict(bag.boston ,newdata=Boston[-train ,])
mean((yhat.bag-boston.test)^2)
```

```
Call:
 randomForest(formula = medv ~ ., data = Boston, mtry = 13, ntree =
25,      subset = train)
               Type of random forest: regression
                     Number of trees: 25
No. of variables tried at each split: 13

          Mean of squared residuals: 14.38913
                    % Var explained: 81.28
```

23.6671575152042

Interprete the results. *Your interpretation of the results goes here!*

Growing a random forest proceeds in exactly the same way, except that we use a smaller value of the m.

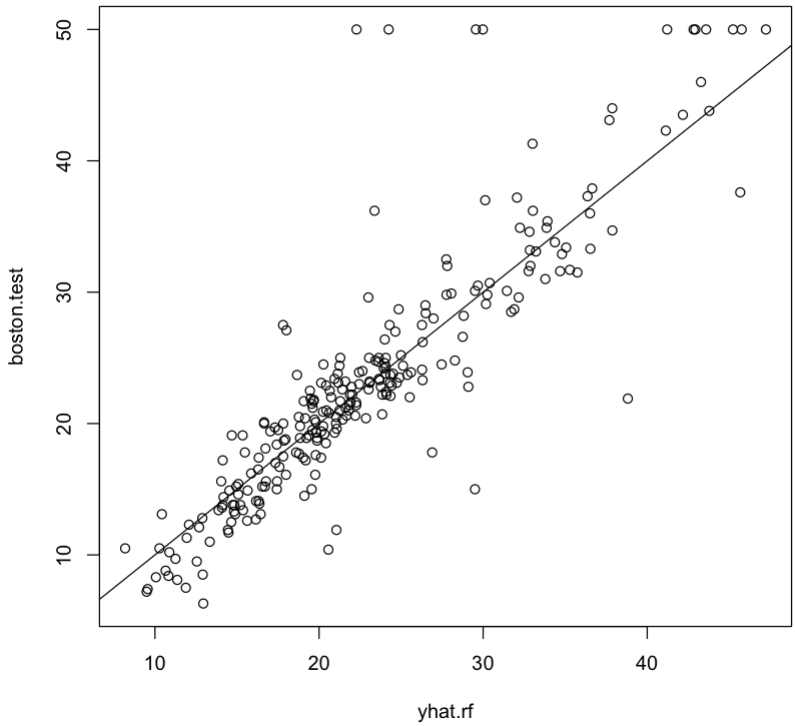
In `R`, the `randomForest()` uses $p/3$ variables when building a random forest of regression trees, and $\sqrt{p}$ variables when building a random forest of classification trees.

Here we use `mtry = 6`.

In [97]:
```
set.seed(1)
rf.boston=randomForest(medv~.,data=Boston,subset=train,mtry=6,impo
rf.boston
yhat.rf = predict(rf.boston ,newdata=Boston[-train ,])
plot(yhat.rf, boston.test)
abline(0,1)
mean((yhat.rf-boston.test)^2)
```

```
Call:
 randomForest(formula = medv ~ ., data = Boston, mtry = 6, importanc
e = TRUE,      subset = train)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 6

          Mean of squared residuals: 9.779194
                    % Var explained: 87.28
```

19.6202073910648



Interprete the results. *Your interpretation of the results goes here!*

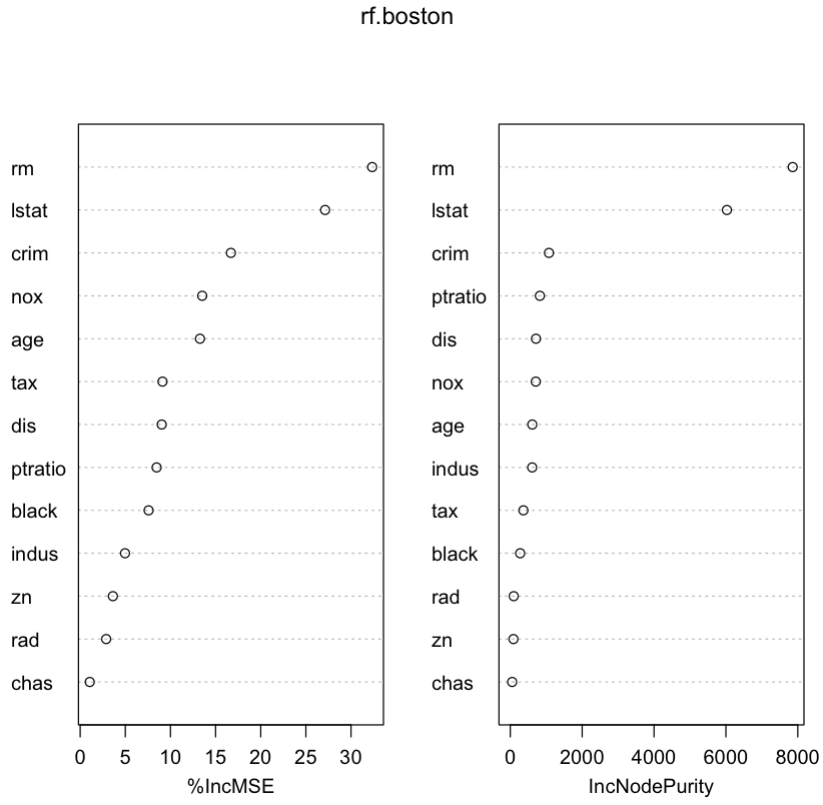
Check the importance of each variable.

In R , we can use the `importance()` and the `varImpPlot()` functions. Two measures of variable importance are reported. The former is based upon the mean decrease of accuracy in predictions on the out of bag samples when a given variable is excluded from the model. The latter is a measure of the total decrease in node impurity (measured by the training RSS) that results from splits over that variable, averaged over all trees.

In [98]:
```
importance(rf.boston)
varImpPlot(rf.boston)
```

A matrix: 13 × 2 of type dbl

	%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



Interprete the results. *Your interpretation of the results goes here!*

Learn and assess Regression Boosting (Trees)

In R we need to load the library `gbm` .

In [99]:
```
install.packages("gbm")
library(gbm)
```

```
The downloaded binary packages are in
        /var/folders/ct/4pcck8t94sdfc73rhymq4t140000gp/T//RtmpCBK2W
j/downloaded_packages
```

```
Loaded gbm 2.1.8
```

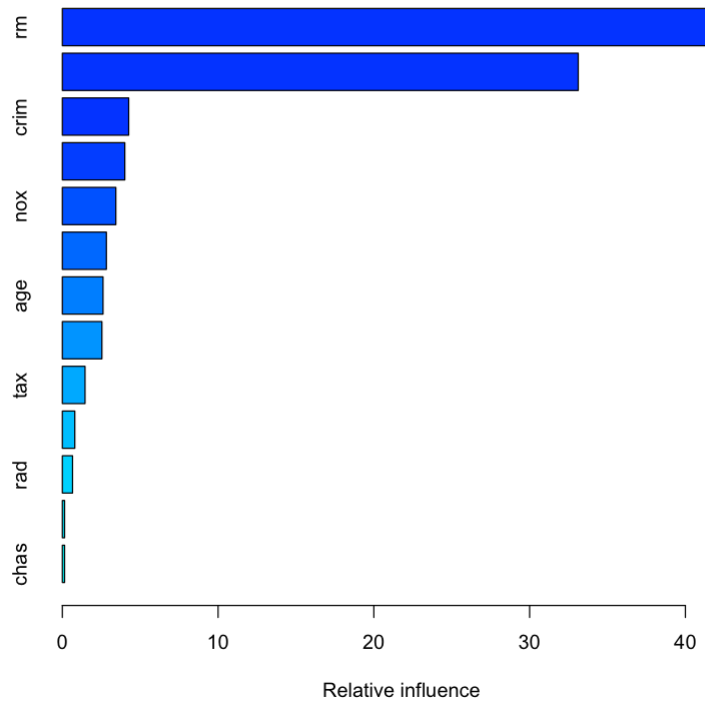
Fit fit boosted regression trees to the `Boston` data set.

In [100]:
```
set.seed(1)
boost.boston=gbm(medv~.,data=Boston[train,],distribution="gaussian
boost.boston
summary(boost.boston)
```

```
gbm(formula = medv ~ ., distribution = "gaussian", data = Boston[tra
in,
    ], n.trees = 5000, interaction.depth = 4)
A gradient boosted model with gaussian loss function.
5000 iterations were performed.
There were 13 predictors of which 13 had non-zero influence.
```

A data.frame: 13 × 2

	var	rel.inf
	<fct>	<dbl>
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 R , the argument `n.trees=5000` indicates that we want 5000 trees, and the option `interaction.depth=4` limits the depth of each tree. By default $\lambda = 0.001$. The `summary()` function produces a relative influence plot and also outputs the relative influence statistics.

Interprete the results. *Your interpretation of the results goes here!*

We now use the boosted model to predict `medv` on the test set.

In [101]:
```
yhat.boost=predict(boost.boston,newdata=Boston[-train,], n.trees=5
mean((yhat.boost -boston.test)^2)
```

18.8470923079959

Use other tuning parameters, e.g., set the shrinkage parameter $\lambda = 0.2$.

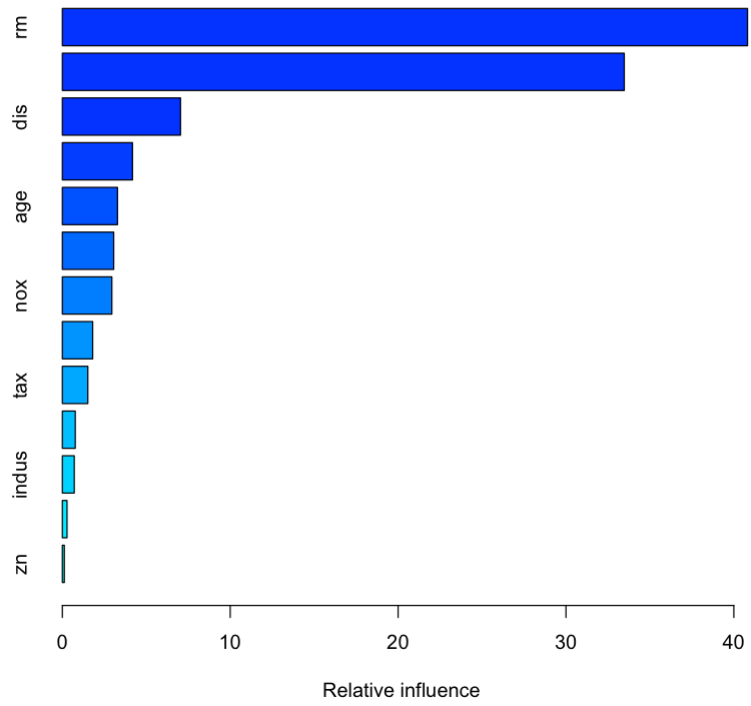
In [102]:

```
boost.boston=gbm(medv~.,data=Boston[train,],distribution="gaussian
boost.boston
summary(boost.boston)
yhat.boost=predict(boost.boston,newdata=Boston[-train,],n.trees=50
plot(yhat.boost, boston.test)
abline(0,1)
mean((yhat.boost -boston.test)^2)
```

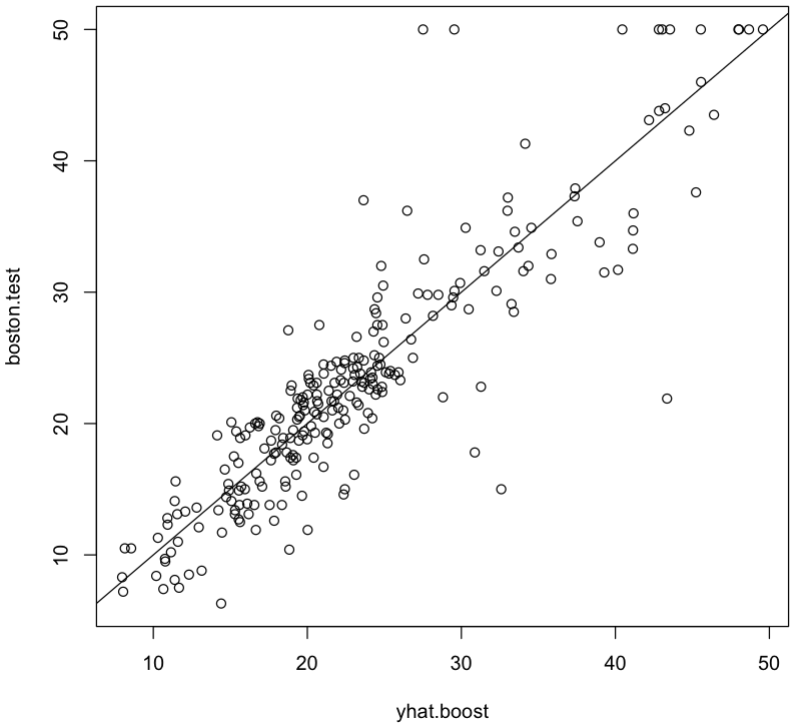
```
gbm(formula = medv ~ ., distribution = "gaussian", data = Boston[tra
in,
    ], n.trees = 5000, interaction.depth = 4, shrinkage = 0.2)
A gradient boosted model with gaussian loss function.
5000 iterations were performed.
There were 13 predictors of which 13 had non-zero influence.
```

A data.frame: 13 × 2

	var	rel.inf
	<fct>	<dbl>
rm	rm	40.8162321
lstat	lstat	33.4732855
dis	dis	7.0384032
crim	crim	4.1787493
age	age	3.2855228
black	black	3.0597513
nox	nox	2.9495066
ptratio	ptratio	1.8071990
tax	tax	1.5182039
rad	rad	0.7693462
indus	indus	0.7087921
chas	chas	0.2781884
zn	zn	0.1168195



18.3345451839923



Interprete the results. *Your interpretation of the results goes here!*