7

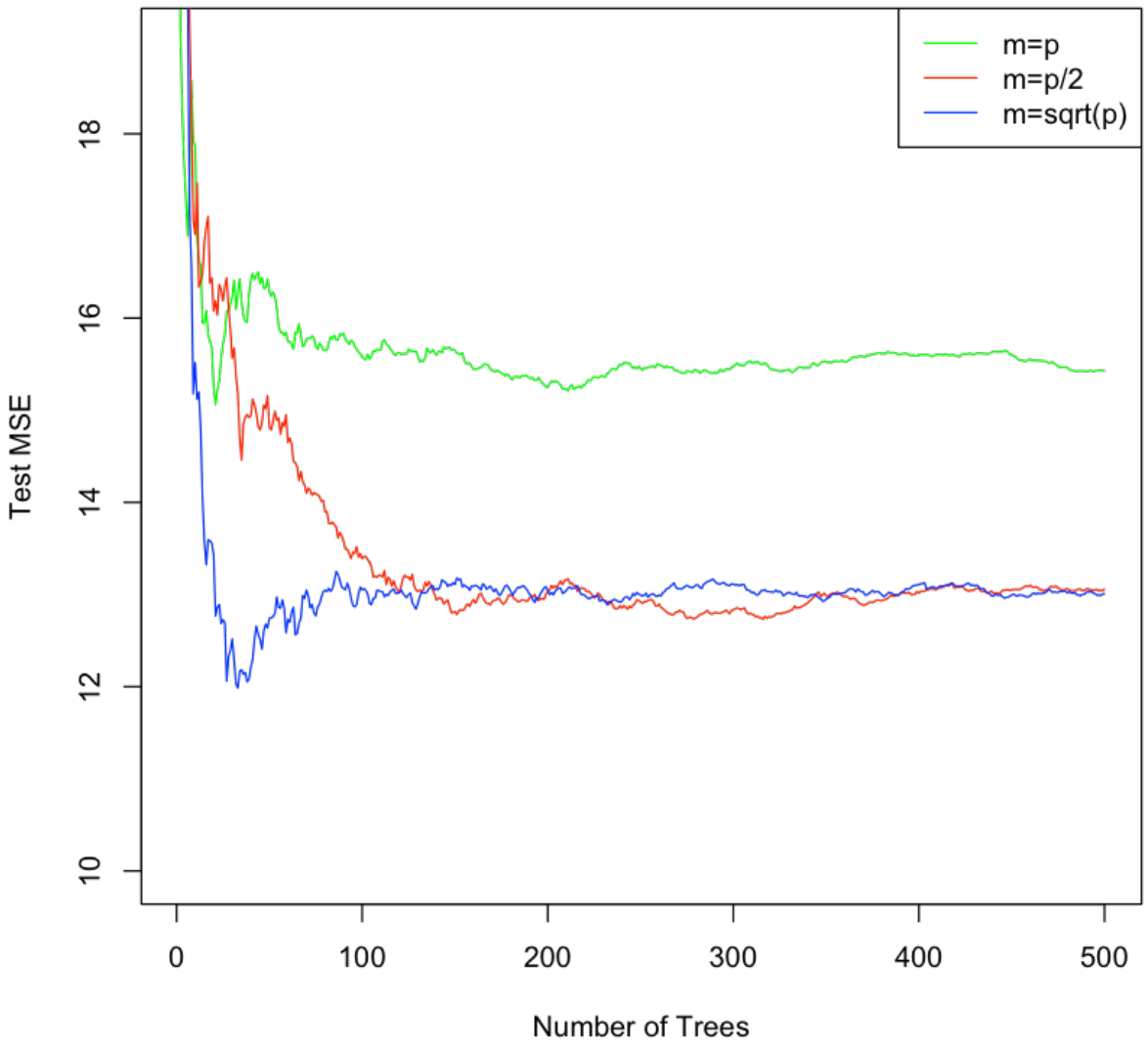
In [11]:
```
library(MASS)
library(randomForest)
```

In [12]:
```
set.seed(1101)
# Construct the train and test matrices
train = sample(dim(Boston)[1], dim(Boston)[1]/2)
X.train = Boston[train, -14]
X.test = Boston[-train, -14]
Y.train = Boston[train, 14]
Y.test = Boston[-train, 14]
```

In [13]:
```
p = dim(Boston)[2] - 1
p.2 = p/2
p.sq = sqrt(p)
```

In [14]:
```
rf.boston.p = randomForest(X.train, Y.train, xtest = X.test,
    ytest = Y.test,  mtry = p, ntree = 500)
rf.boston.p.2 = randomForest(X.train, Y.train,
 xtest = X.test, ytest = Y.test,  mtry = p.2, ntree = 500)
rf.boston.p.sq = randomForest(X.train, Y.train,
xtest = X.test, ytest = Y.test, mtry = p.sq, ntree = 500)
```

In [15]:
```
plot(1:500, rf.boston.p$test$mse, col = "green", type = "l",
     xlab = "Number of Trees",
    ylab = "Test MSE", ylim = c(10, 19))
lines(1:500, rf.boston.p.2$test$mse, col = "red",
      type = "l")
lines(1:500, rf.boston.p.sq$test$mse, col = "blue",
      type = "l")
legend("topright", c("m=p", "m=p/2", "m=sqrt(p)"),
       col = c("green", "red", "blue"),
    cex = 1, lty = 1)
```



In [ ]:
```
#The plot shows that test MSE for single tree is quite high (arour
#It is reduced by adding more trees to the model and stabilizes ar
#a few hundred trees. Test MSE for including all variables at spli
#slightly higher(apprx 11) as compared to both using half or squar
#number of variables (both slightly less than 10).
```

#8(a)

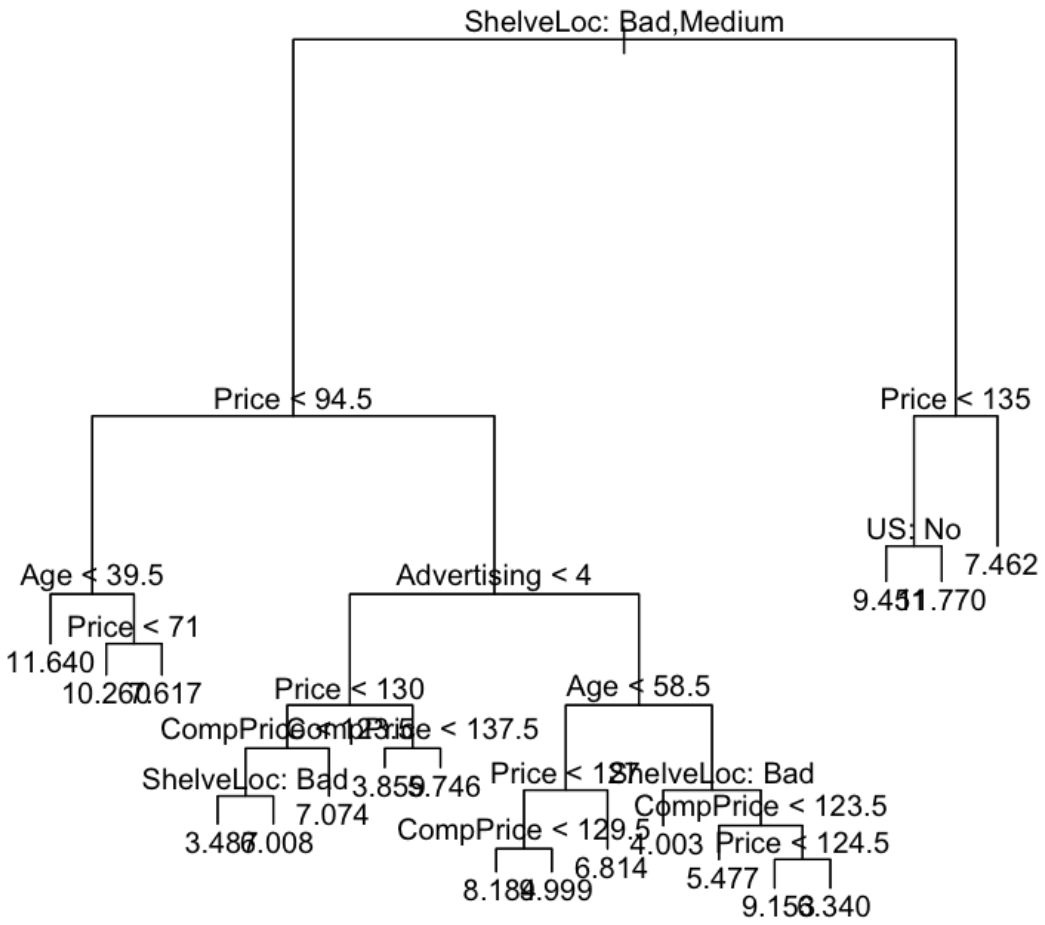
In [16]:
```
library(ISLR)
attach(Carseats)
set.seed(1)

train = sample(dim(Carseats)[1], dim(Carseats)[1]/2)
Carseats.train = Carseats[train, ]
Carseats.test = Carseats[-train, ]
```

#8(b)

In [17]:
```
library(tree)
tree.carseats = tree(Sales ~ ., data = Carseats.train)
summary(tree.carseats)
```

```
Regression tree:
tree(formula = Sales ~ ., data = Carseats.train)
Variables actually used in tree construction:
[1] "ShelveLoc"   "Price"       "Age"         "Advertising" "CompPric
e"
[6] "US"
Number of terminal nodes:  18
Residual mean deviance:  2.167 = 394.3 / 182
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-3.88200 -0.88200 -0.08712  0.00000  0.89590  4.09900
```

In [18]:
```
plot(tree.carseats)
text(tree.carseats, pretty = 0)
```

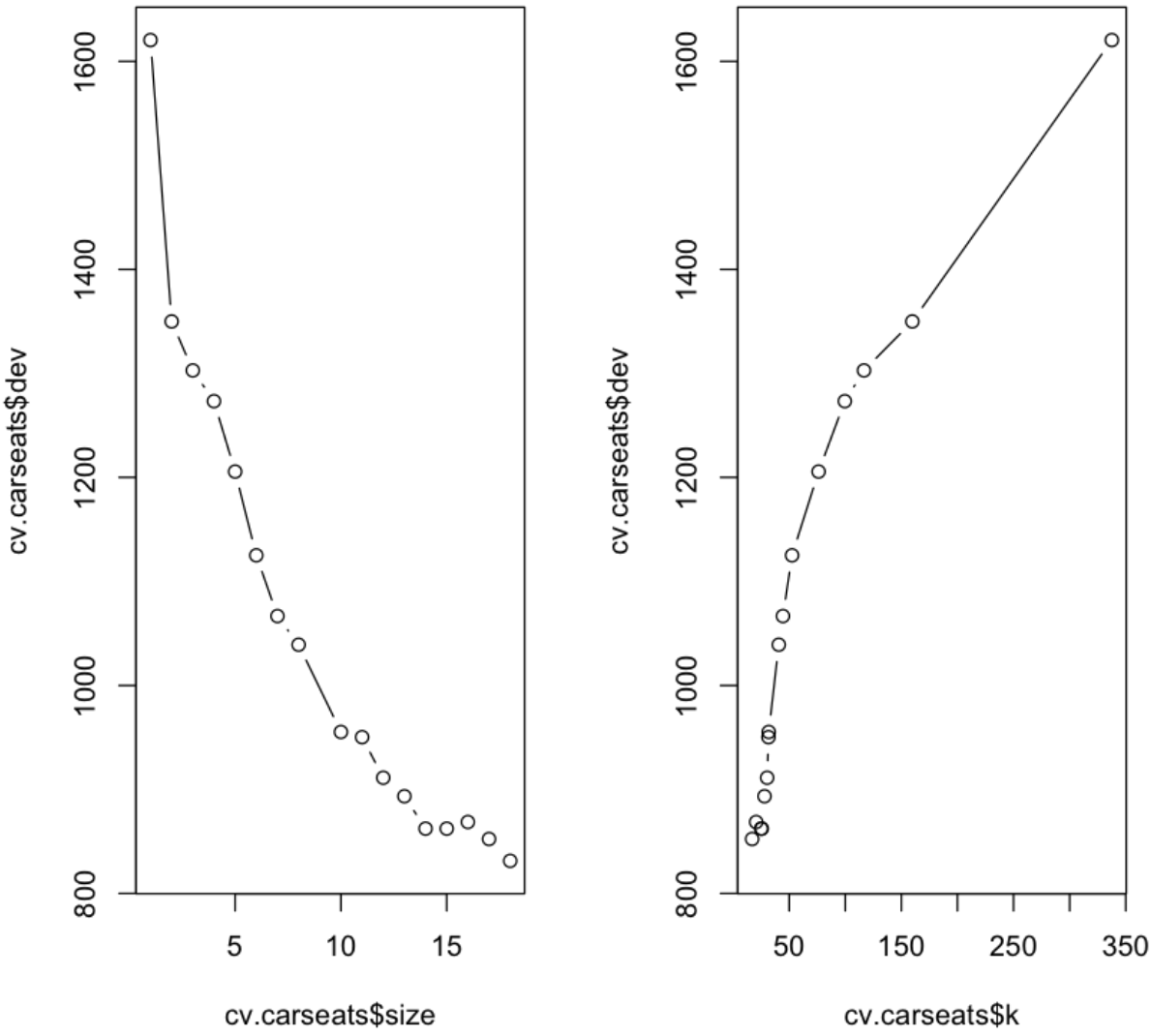


In [19]:
```
pred.carseats = predict(tree.carseats, Carseats.test)
mean((Carseats.test$Sales - pred.carseats)^2)
```

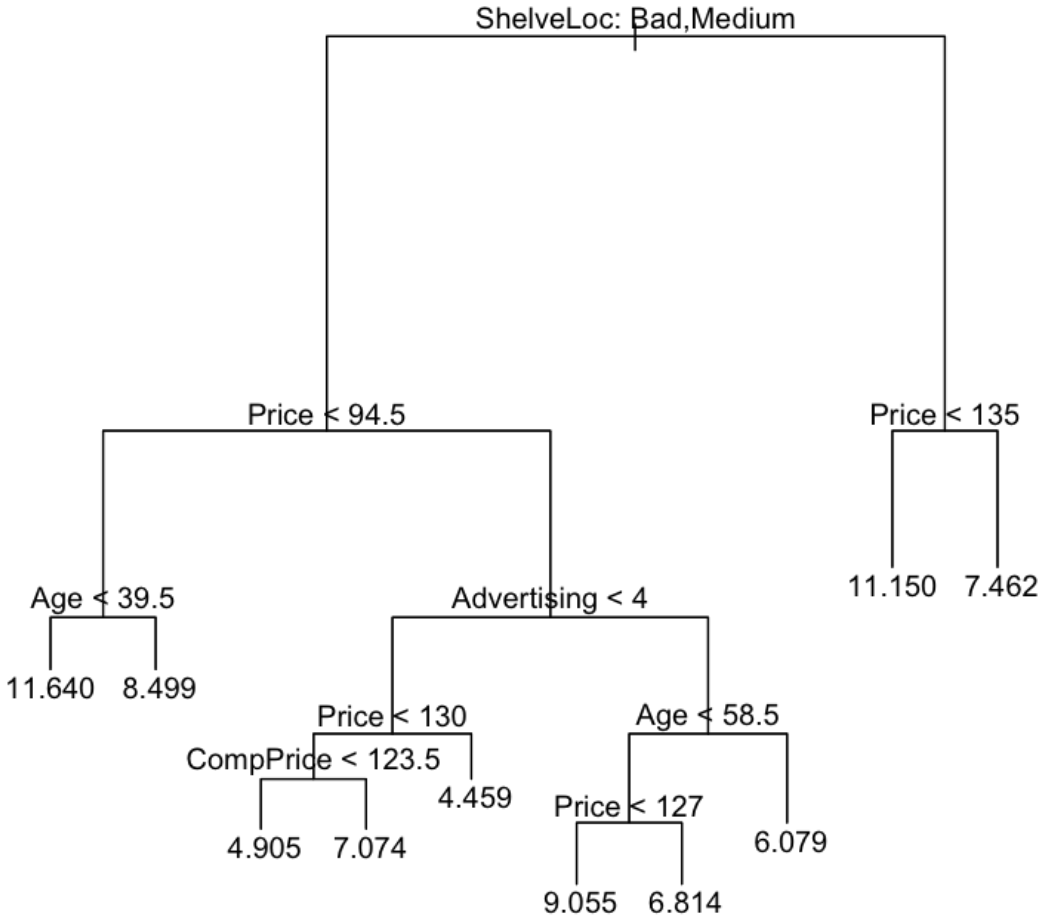
4.92203909112416

#8(c)

In [20]:
```
cv.carseats = cv.tree(tree.carseats, FUN = prune.tree)
par(mfrow = c(1, 2))
plot(cv.carseats$size, cv.carseats$dev, type = "b")
plot(cv.carseats$k, cv.carseats$dev, type = "b")
```



In [21]:
```
# Best size = 9
pruned.carseats = prune.tree(tree.carseats, best = 9)
par(mfrow = c(1, 1))
plot(pruned.carseats)
text(pruned.carseats, pretty = 0)
```



In [22]:
```
pred.pruned = predict(pruned.carseats, Carseats.test)
mean((Carseats.test$Sales - pred.pruned)^2)
```

4.9181343772529

#8(d)

In [23]:
```
library(randomForest)
```

In [24]:
```
bag.carseats = randomForest(Sales ~ ., data = Carseats.train,
mtry = 10, ntree = 500,  importance = T)
bag.pred = predict(bag.carseats, Carseats.test)
mean((Carseats.test$Sales - bag.pred)^2)
```

2.65729621830957

In [25]:
```
importance(bag.carseats)
```

	%IncMSE	IncNodePurity
CompPrice	23.07909904	171.185734
Income	2.82081527	94.079825
Advertising	11.43295625	99.098941
Population	-3.92119532	59.818905
Price	54.24314632	505.887016
ShelveLoc	46.26912996	361.962753
Age	14.24992212	159.740422
Education	-0.07662320	46.738585
Urban	0.08530119	8.453749
US	4.34349223	15.157608

8(e)

In [26]:
```
rf.carseats = randomForest(Sales ~ ., data = Carseats.train,
mtry = 5, ntree = 500, importance = T)
rf.pred = predict(rf.carseats, Carseats.test)
mean((Carseats.test$Sales - rf.pred)^2)
```

2.70166474672903

In [27]:
```
importance(rf.carseats)
```

	%IncMSE	IncNodePurity
CompPrice	19.8160444	162.73603
Income	2.8940268	106.96093
Advertising	11.6799573	106.30923
Population	-1.6998805	79.04937
Price	46.3454015	448.33554
ShelveLoc	40.4412189	334.33610
Age	12.5440659	169.06125
Education	1.0762096	55.87510
Urban	0.5703583	13.21963
US	5.8799999	25.59797

#9(a)

In [28]:
```
library(ISLR)
attach(OJ)
set.seed(1013)

train = sample(dim(OJ)[1], 800)
OJ.train = OJ[train, ]
OJ.test = OJ[-train, ]
```

#9(b)

In [29]:
```
library(tree)
oj.tree = tree(Purchase ~ ., data = OJ.train)
summary(oj.tree)
```

```
Classification tree:
tree(formula = Purchase ~ ., data = OJ.train)
Variables actually used in tree construction:
[1] "LoyalCH"       "PriceDiff"     "ListPriceDiff" "SalePriceMM"
Number of terminal nodes:  7
Residual mean deviance:  0.7564 = 599.8 / 793
Misclassification error rate: 0.1612 = 129 / 800
```

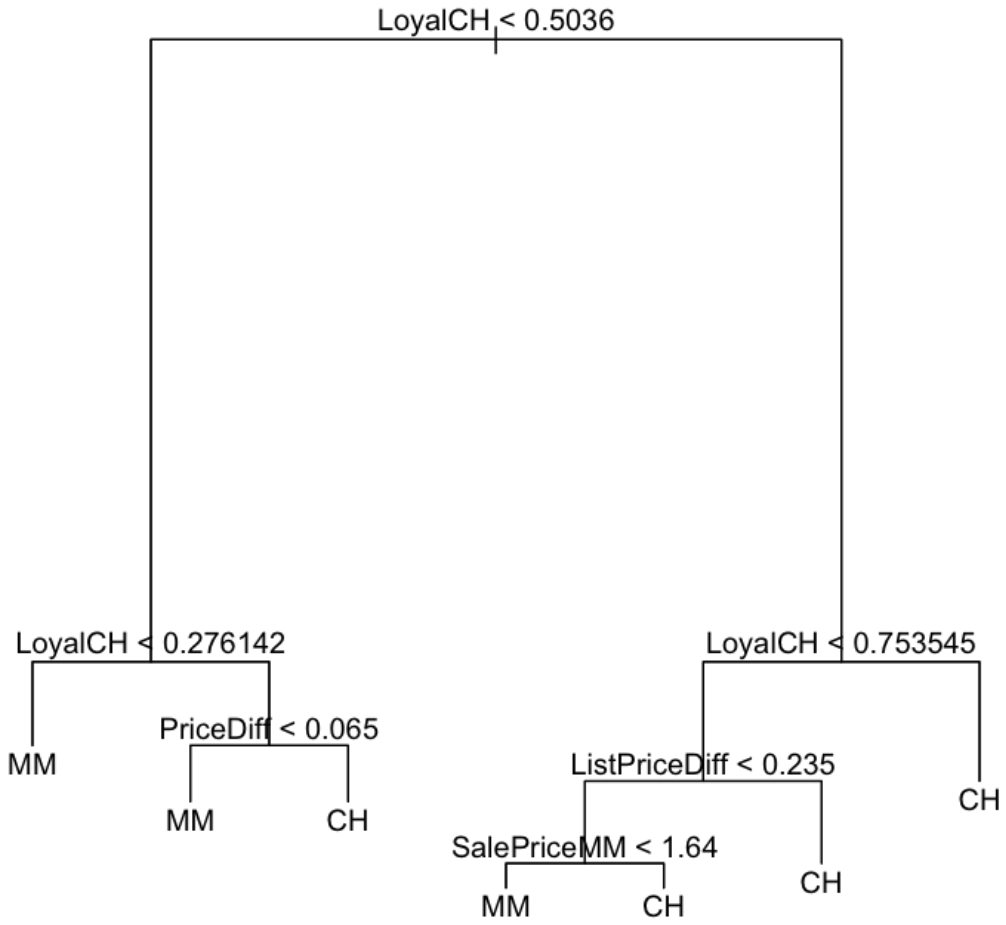
#9(c)

In [30]:
```
1 oj.tree
```

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

 1) root 800 1069.00 CH ( 0.61125 0.38875 )
   2) LoyalCH < 0.5036 344  407.30 MM ( 0.27907 0.72093 )
     4) LoyalCH < 0.276142 163  121.40 MM ( 0.12270 0.87730 ) *
     5) LoyalCH > 0.276142 181  246.30 MM ( 0.41989 0.58011 )
      10) PriceDiff < 0.065 75   75.06 MM ( 0.20000 0.80000 ) *
      11) PriceDiff > 0.065 106  144.50 CH ( 0.57547 0.42453 ) *
   3) LoyalCH > 0.5036 456  366.30 CH ( 0.86184 0.13816 )
     6) LoyalCH < 0.753545 189  224.30 CH ( 0.71958 0.28042 )
      12) ListPriceDiff < 0.235 79  109.40 MM ( 0.48101 0.51899 )
        24) SalePriceMM < 1.64 22   20.86 MM ( 0.18182 0.81818 ) *
        25) SalePriceMM > 1.64 57   76.88 CH ( 0.59649 0.40351 ) *
      13) ListPriceDiff > 0.235 110   75.81 CH ( 0.89091 0.10909 ) *
     7) LoyalCH > 0.753545 267   85.31 CH ( 0.96255 0.03745 ) *
```

#9(d)

In [31]:
```
plot(oj.tree)
text(oj.tree, pretty = 0)
```



#9(e)

In [32]:
```
oj.pred = predict(oj.tree, OJ.test, type = "class")
table(OJ.test$Purchase, oj.pred)
```

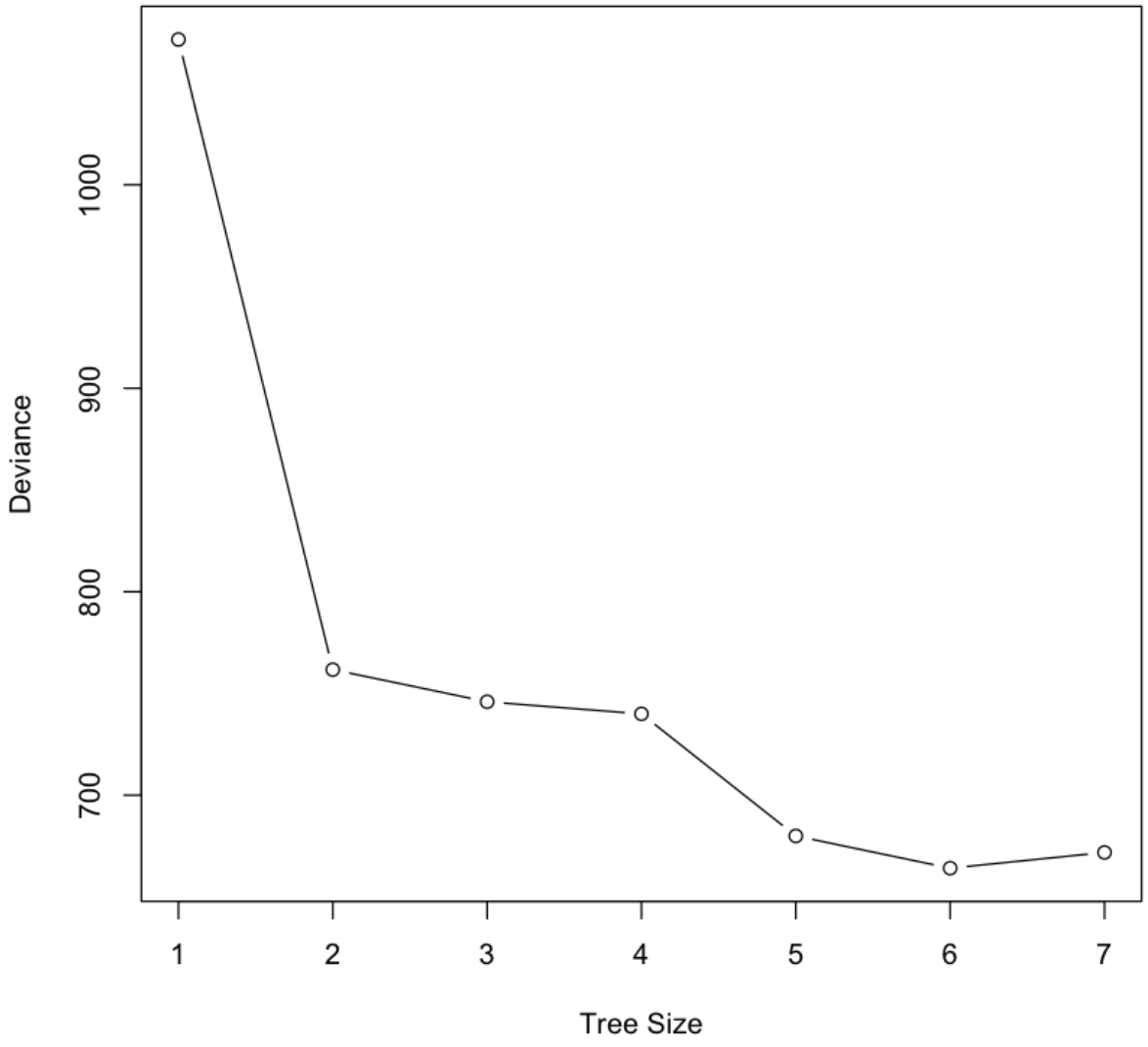
```
   oj.pred
     CH  MM
CH 149  15
MM  30  76
```

#9(f)

In [35]:
```
cv.oj = cv.tree(oj.tree, FUN = prune.tree)
```

#9(g)

In [36]:
```
plot(cv.oj$size, cv.oj$dev, type = "b", xlab = "Tree Size", ylab =
```



#9(h)

In [ ]:
```
#Size of 6 gives lowest cross-validation error.
```

#9(i)

In [37]:
```
oj.pruned = prune.tree(oj.tree, best = 6)
```

#9(j)

In [38]:
```
summary(oj.pruned)
```

```
Classification tree:
snip.tree(tree = oj.tree, nodes = 12L)
Variables actually used in tree construction:
[1] "LoyalCH"       "PriceDiff"     "ListPriceDiff"
Number of terminal nodes:  6
Residual mean deviance:  0.7701 = 611.5 / 794
Misclassification error rate: 0.175 = 140 / 800
```

#9(k)

In [39]:
```
pred.unpruned = predict(oj.tree, OJ.test, type = "class")
misclass.unpruned = sum(OJ.test$Purchase != pred.unpruned)
misclass.unpruned/length(pred.unpruned)
```

0.166666666666667

In [40]:
```
pred.pruned = predict(oj.pruned, OJ.test, type = "class")
misclass.pruned = sum(OJ.test$Purchase != pred.pruned)
misclass.pruned/length(pred.pruned)
```

0.2

#10(a)

In [1]:
```
library(ISLR)
sum(is.na(Hitters$Salary))
```

59

In [2]:
```
Hitters = Hitters[-which(is.na(Hitters$Salary)), ]
sum(is.na(Hitters$Salary))
```

```
0
```

In [3]:
```
Hitters$Salary = log(Hitters$Salary)
```

#10(b)

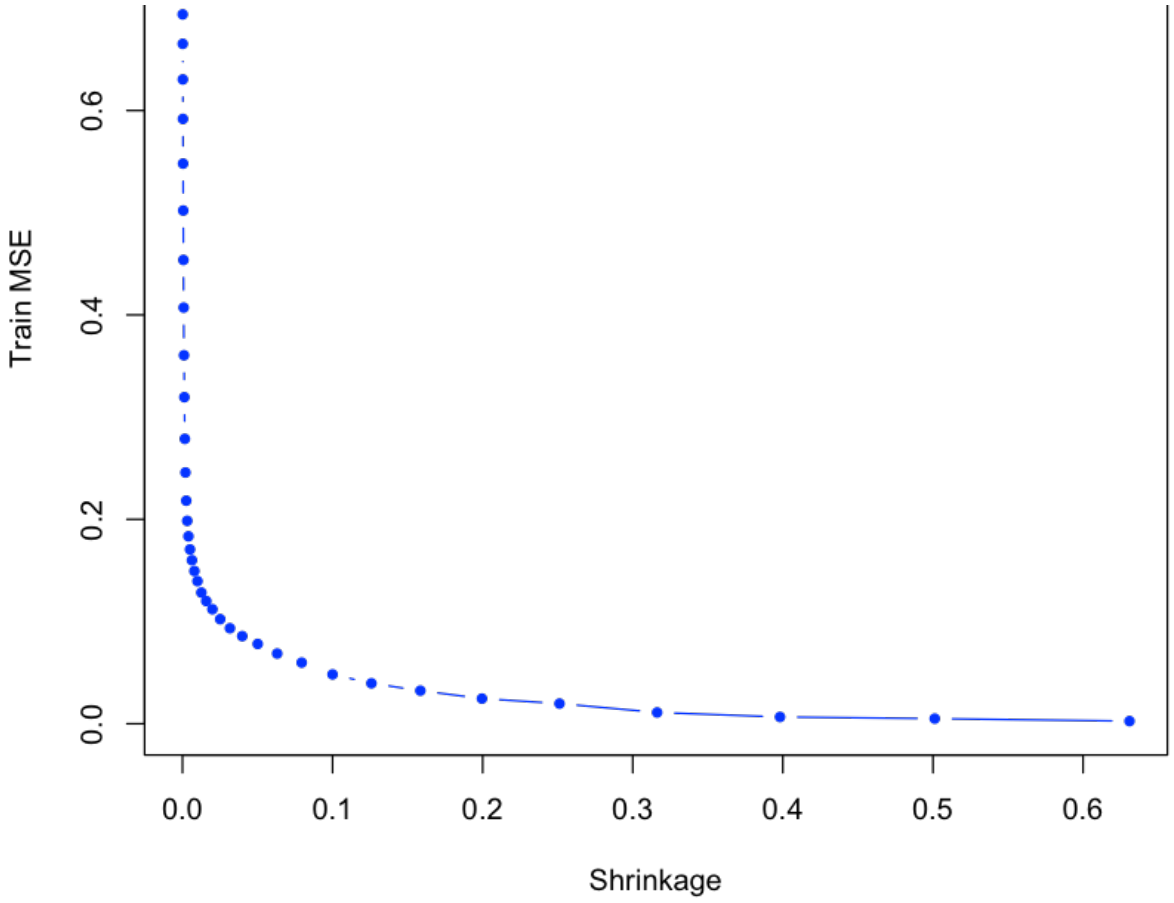
In [4]:
```
train = 1:200
Hitters.train = Hitters[train, ]
Hitters.test = Hitters[-train, ]
```

#10(c)

In [5]:
```
library(gbm)
```

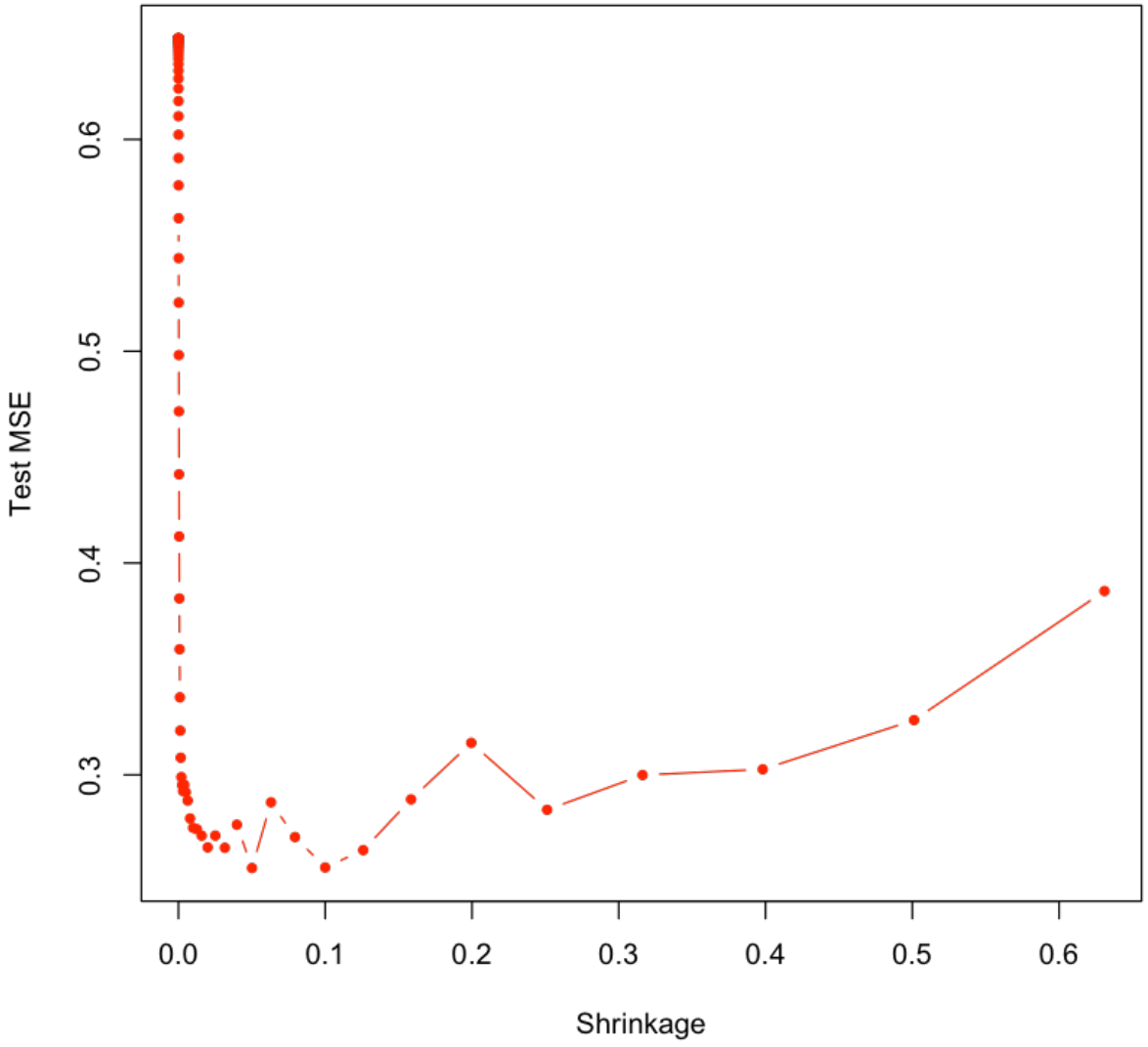
```
Loaded gbm 2.1.5
```

In [6]:
```
set.seed(103)
pows = seq(-10, -0.2, by = 0.1)
lambdas = 10^pows
length.lambdas = length(lambdas)
train.errors = rep(NA, length.lambdas)
test.errors = rep(NA, length.lambdas)
for (i in 1:length.lambdas) {
    boost.hitters = gbm(Salary~., data = Hitters.train, distributi
        n.trees = 1000, shrinkage = lambdas[i])
    train.pred = predict(boost.hitters, Hitters.train, n.trees = 1
    test.pred = predict(boost.hitters, Hitters.test, n.trees = 100
    train.errors[i] = mean((Hitters.train$Salary - train.pred)^2)
    test.errors[i] = mean((Hitters.test$Salary - test.pred)^2)
}

plot(lambdas, train.errors, type = "b", xlab = "Shrinkage", ylab =
    col = "blue", pch = 20)
```



#10(d)

In [7]:
```
plot(lambdas,test.errors, type = "b", xlab = "Shrinkage", ylab = "
    col = "red", pch = 20)
```



In [8]:
```
min(test.errors)
```

0.256050707266756

In [9]:
```
lambdas[which.min(test.errors)]
```

0.0501187233627274

#10(e)

In [10]:
```
lm.fit = lm(Salary~., data = Hitters.train)
lm.pred = predict(lm.fit, Hitters.test)
mean((Hitters.test$Salary - lm.pred)^2)
```

0.491795937545494

In [11]:
```
library(glmnet)
```

```
Loading required package: Matrix
Loading required package: foreach
Loaded glmnet 2.0-16
```

In [12]:
```
set.seed(134)
x = model.matrix(Salary~., data = Hitters.train)
y = Hitters.train$Salary
x.test = model.matrix(Salary~., data = Hitters.test)
lasso.fit = glmnet(x, y, alpha = 1)
lasso.pred = predict(lasso.fit, s = 0.01, newx = x.test)
mean((Hitters.test$Salary - lasso.pred)^2)
```

0.470053732710274

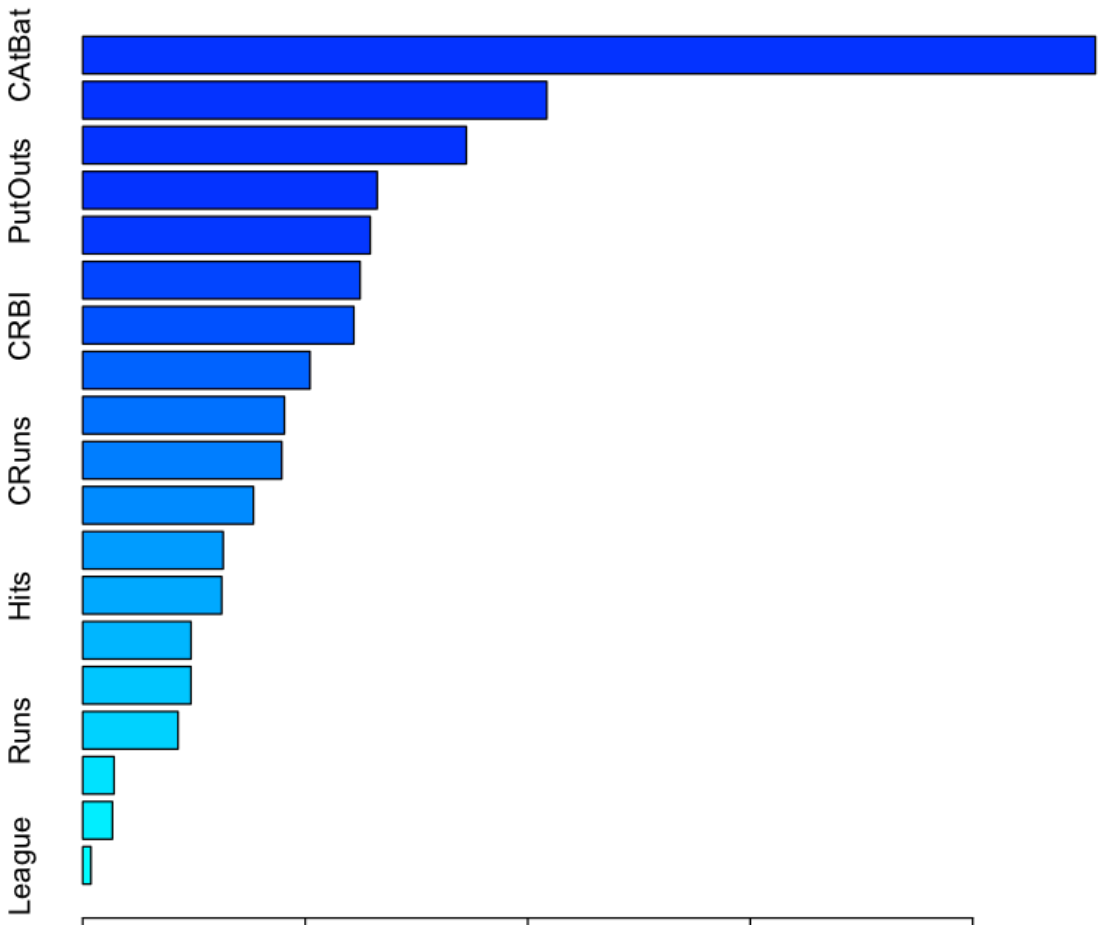
In [ ]:
```
#Both the models linear and regularization (Lasso) have
#higher test MSE than boosting.
```

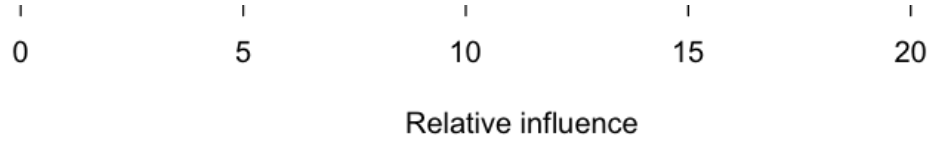
#10(f)

In [13]:
```
boost.best = gbm(Salary~., data = Hitters.train, distribution = "g
    n.trees = 1000, shrinkage = lambdas[which.min(test.errors)])
summary(boost.best)
```

	var	rel.inf
CAtBat	CAtBat	22.7562681
CWalks	CWalks	10.4279674
CHits	CHits	8.6198109
PutOuts	PutOuts	6.6159325
Years	Years	6.4611683
Walks	Walks	6.2331148

CRBI	CRBI	6.0926744
CHmRun	CHmRun	5.1076104
RBI	RBI	4.5321678
CRuns	CRuns	4.4728132
Assists	Assists	3.8366575
HmRun	HmRun	3.1554038
Hits	Hits	3.1229284
AtBat	AtBat	2.4338530
Errors	Errors	2.4324185
Runs	Runs	2.1425481
Division	Division	0.7041949
NewLeague	NewLeague	0.6675446
League	League	0.1849234





In [ ]:
```
#CAtBat CRBI Cwalks are imp variables

```

In [14]:
```
library(randomForest)
```

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

#10(g)

In [15]:
```
set.seed(21)
rf.hitters = randomForest(Salary~., data = Hitters.train, ntree =
rf.pred = predict(rf.hitters, Hitters.test)
mean((Hitters.test$Salary - rf.pred)^2)
```

```
0.230391897514157
```

In [ ]:
```
#MSE for bagging is 0.23 which less than the boosting
```