ISLR

Wenjie Tu

Linear Regression

```r
library(MASS)
library(ISLR)
```

Simplie Linear Regression

```r
attach(Boston)
data(Boston)
names(Boston)
```

```
##  [1] "crim"    "zn"      "indus"   "chas"    "nox"     "rm"      "age"
##  [8] "dis"     "rad"     "tax"     "ptratio" "black"   "lstat"   "medv"
```

```r
lm.fit <- lm(medv~lstat, data=Boston)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = medv ~ lstat, data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.168  -3.990  -1.318   2.034  24.500
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 34.55384    0.56263   61.41   <2e-16 ***
## lstat       -0.95005    0.03873  -24.53   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.216 on 504 degrees of freedom
## Multiple R-squared:  0.5441, Adjusted R-squared:  0.5432
## F-statistic: 601.6 on 1 and 504 DF,  p-value: < 2.2e-16
```

```r
## Confidence interval
confint(lm.fit)
```

```
##                  2.5 %     97.5 %
## (Intercept) 33.448457 35.6592247
## lstat       -1.026148 -0.8739505
```

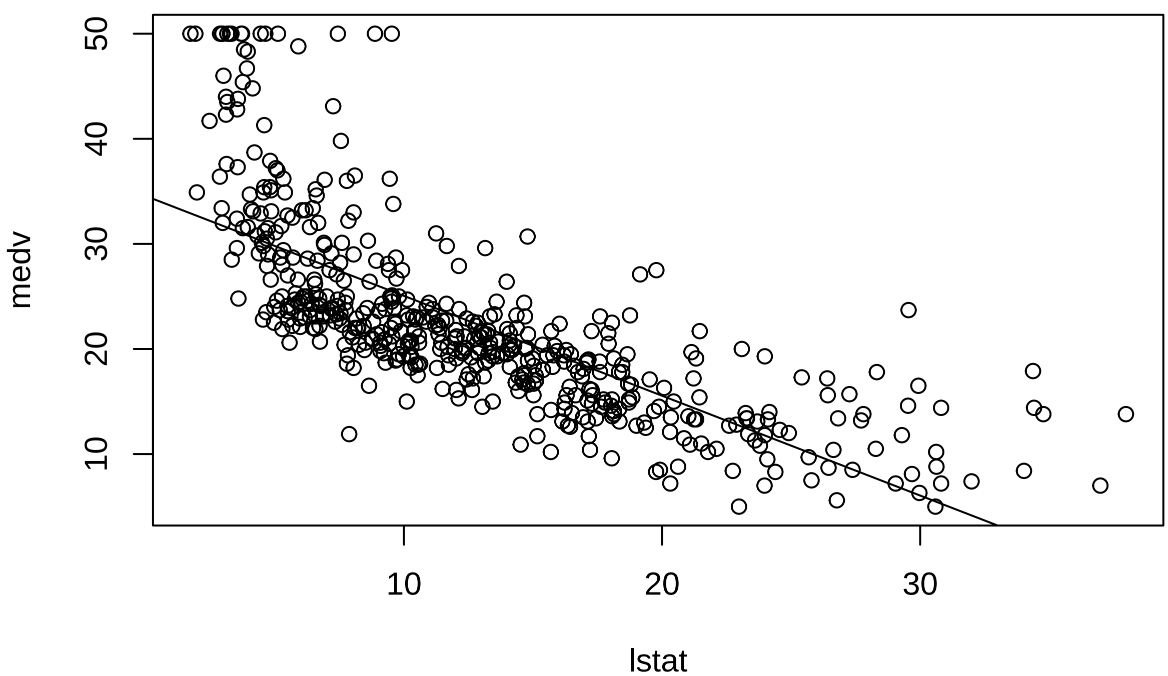
```r
## Use predict() function to produce CIs and PIs
predict(lm.fit, newdata=data.frame(lstat=c(5,10,15)), interval="confidence")
```

```
##        fit      lwr      upr
## 1 29.80359 29.00741 30.59978
## 2 25.05335 24.47413 25.63256
## 3 20.30310 19.73159 20.87461
```

```r
predict(lm.fit, newdata=data.frame(lstat=c(5,10,15)), interval="prediction")
```

```
##        fit       lwr      upr
## 1 29.80359 17.565675 42.04151
## 2 25.05335 12.827626 37.27907
## 3 20.30310  8.077742 32.52846
```

```r
plot(lstat, medv)
abline(lm.fit)
```



Note:

- To draw a line with intercept `a` and slope `b`, we type `abline(a, b)`.

Multiple Linear Regression

```r
lm.fit <- lm(medv~lstat+age, data=Boston)
coef(summary(lm.fit))
```

```
##                Estimate Std. Error    t value      Pr(>|t|)
## (Intercept) 33.22276053 0.73084711  45.457881 2.943785e-180
## lstat       -1.03206856 0.04819073 -21.416330  8.419554e-73
## age          0.03454434 0.01222547   2.825605  4.906776e-03
```

```r
lm.fit <- lm(medv~., data=Boston)
coef(summary(lm.fit))
```

```
##                  Estimate  Std. Error      t value     Pr(>|t|)
## (Intercept)  3.645949e+01 5.103458811   7.14407419 3.283438e-12
## crim        -1.080114e-01 0.032864994  -3.28651687 1.086810e-03
## zn           4.642046e-02 0.013727462   3.38157628 7.781097e-04
## indus        2.055863e-02 0.061495689   0.33431004 7.382881e-01
## chas         2.686734e+00 0.861579756   3.11838086 1.925030e-03
## nox         -1.776661e+01 3.819743707  -4.65125741 4.245644e-06
## rm           3.809865e+00 0.417925254   9.11614020 1.979441e-18
## age          6.922246e-04 0.013209782   0.05240243 9.582293e-01
## dis         -1.475567e+00 0.199454735  -7.39800360 6.013491e-13
## rad          3.060495e-01 0.066346440   4.61289977 5.070529e-06
## tax         -1.233459e-02 0.003760536  -3.28000914 1.111637e-03
## ptratio     -9.527472e-01 0.130826756  -7.28251056 1.308835e-12
## black        9.311683e-03 0.002685965   3.46679256 5.728592e-04
## lstat       -5.247584e-01 0.050715278 -10.34714580 7.776912e-23
```

```r
lm.fit <- lm(medv~.-age, data=Boston)
coef(summary(lm.fit))
```

```
##                  Estimate  Std. Error     t value     Pr(>|t|)
## (Intercept)  36.436926648 5.080119139   7.1724551 2.715464e-12
## crim         -0.108005604 0.032831554  -3.2896891 1.074747e-03
## zn            0.046333661 0.013613376   3.4035393 7.193806e-04
## indus         0.020562177 0.061433423   0.3347067 7.379887e-01
## chas          2.689026199 0.859597742   3.1282379 1.862634e-03
## nox         -17.713539860 3.679308218  -4.8143669 1.967110e-06
## rm            3.814393564 0.408479812   9.3380222 3.365945e-19
## dis          -1.478611555 0.190611263  -7.7572098 5.027955e-14
## rad           0.305785940 0.066088613   4.6269081 4.750539e-06
## tax          -0.012328692 0.003755046  -3.2832334 1.099120e-03
## ptratio      -0.952211173 0.130294221  -7.3081612 1.099178e-12
## black         0.009320653 0.002677793   3.4807225 5.444689e-04
## lstat        -0.523851840 0.047625285 -10.9994479 2.569688e-25
```

Interaction Terms

```r
lm.fit <- lm(medv~lstat*age, data=Boston)
coef(summary(lm.fit))
```

```
##                  Estimate  Std. Error     t value     Pr(>|t|)
## (Intercept) 36.0885359346 1.469835463 24.55277263 4.907116e-88
## lstat       -1.3921168406 0.167455532 -8.31335236 8.780730e-16
## age         -0.0007208595 0.019879171 -0.03626205 9.710878e-01
## lstat:age    0.0041559518 0.001851795  2.24428275 2.524911e-02
```

Non-linear Transformations of the Predictors

```r
lm.fit <- lm(medv~lstat+I(lstat^2), data=Boston)
coef(summary(lm.fit))
```

```
##                Estimate  Std. Error   t value      Pr(>|t|)
## (Intercept) 42.86200733 0.872084393  49.14892 3.500749e-194
## lstat       -2.33282110 0.123803305 -18.84296  2.548861e-60
## I(lstat^2)   0.04354689 0.003745149  11.62755  7.630116e-28
```

```r
## Hypothesis test
anova(lm(medv~lstat, data=Boston), lm(medv~lstat+I(lstat^2), data=Boston))
```

```
## Analysis of Variance Table
##
## Model 1: medv ~ lstat
## Model 2: medv ~ lstat + I(lstat^2)
##   Res.Df   RSS Df Sum of Sq     F    Pr(>F)
## 1    504 19472
## 2    503 15347  1    4125.1 135.2 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The `anova()` function performs a hypothesis test comparing the two models. The null hypothesis is that the two models fit the data equally well, and alternative hypothesis is that the full model is superior.

```r
lm.fit5 <- lm(medv~poly(lstat, 5))
summary(lm.fit5)
```

```
##
## Call:
## lm(formula = medv ~ poly(lstat, 5))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.5433  -3.1039  -0.7052   2.0844  27.1153
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)       22.5328     0.2318  97.197  < 2e-16 ***
## poly(lstat, 5)1 -152.4595     5.2148 -29.236  < 2e-16 ***
## poly(lstat, 5)2   64.2272     5.2148  12.316  < 2e-16 ***
## poly(lstat, 5)3  -27.0511     5.2148  -5.187 3.10e-07 ***
## poly(lstat, 5)4   25.4517     5.2148   4.881 1.42e-06 ***
## poly(lstat, 5)5  -19.2524     5.2148  -3.692 0.000247 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.215 on 500 degrees of freedom
## Multiple R-squared:  0.6817, Adjusted R-squared:  0.6785
## F-statistic: 214.2 on 5 and 500 DF,  p-value: < 2.2e-16
```

```r
## Log-transformation
summary(lm(medv~log(rm), data=Boston))
```

```
##
## Call:
## lm(formula = medv ~ log(rm), data = Boston)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -19.487  -2.875  -0.104   2.837  39.816
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -76.488      5.028  -15.21   <2e-16 ***
## log(rm)       54.055      2.739   19.73   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.915 on 504 degrees of freedom
## Multiple R-squared:  0.4358, Adjusted R-squared:  0.4347
## F-statistic: 389.3 on 1 and 504 DF,  p-value: < 2.2e-16
```

Qualitative Predictors

```r
data("Carseats")
names(Carseats)
```

```
##  [1] "Sales"       "CompPrice"   "Income"      "Advertising" "Population"
##  [6] "Price"       "ShelveLoc"   "Age"         "Education"   "Urban"
## [11] "US"
```

```r
lm.fit <- lm(Sales~.+Income:Advertising+Price:Age, data=Carseats)
coef(summary(lm.fit))
```

```
##                        Estimate   Std. Error     t value      Pr(>|t|)
## (Intercept)        6.5755654389 1.0087469829   6.5185478  2.223618e-10
## CompPrice          0.0929371187 0.0041183079  22.5668216  1.640774e-72
```

```
## Income                0.0108939611 0.0026044403   4.1828416  3.566527e-05
## Advertising           0.0702462284 0.0226091318   3.1069848  2.029896e-03
## Population            0.0001592453 0.0003678575   0.4328994  6.653296e-01
## Price                -0.1008063583 0.0074398929 -13.5494367  1.738295e-34
## ShelveLocGood         4.8486762073 0.1528378349  31.7243189 1.384764e-109
## ShelveLocMedium       1.9532619948 0.1257681879  15.5306523  1.336388e-42
## Age                  -0.0579465902 0.0159505783  -3.6328833  3.181359e-04
## Education            -0.0208524907 0.0196131499  -1.0631893  2.883608e-01
## UrbanYes              0.1401597237 0.1124018985   1.2469516  2.131713e-01
## USYes                -0.1575571432 0.1489233659  -1.0579746  2.907286e-01
## Income:Advertising    0.0007510392 0.0002784092   2.6976091  7.290232e-03
## Price:Age             0.0001067598 0.0001333371   0.8006763  4.238116e-01
```

```
attach(Carseats)
contrasts(ShelveLoc)
```

```
##        Good Medium
## Bad       0      0
## Good      1      0
## Medium    0      1
```

The `contrasts()` function returns the coding for the dummy variables.

Classification

The Stock Market

```
library(ISLR)
names(Smarket)
```

```
## [1] "Year"      "Lag1"      "Lag2"      "Lag3"      "Lag4"      "Lag5"
## [7] "Volume"    "Today"     "Direction"
```

```
dim(Smarket)
```

```
## [1] 1250    9
```

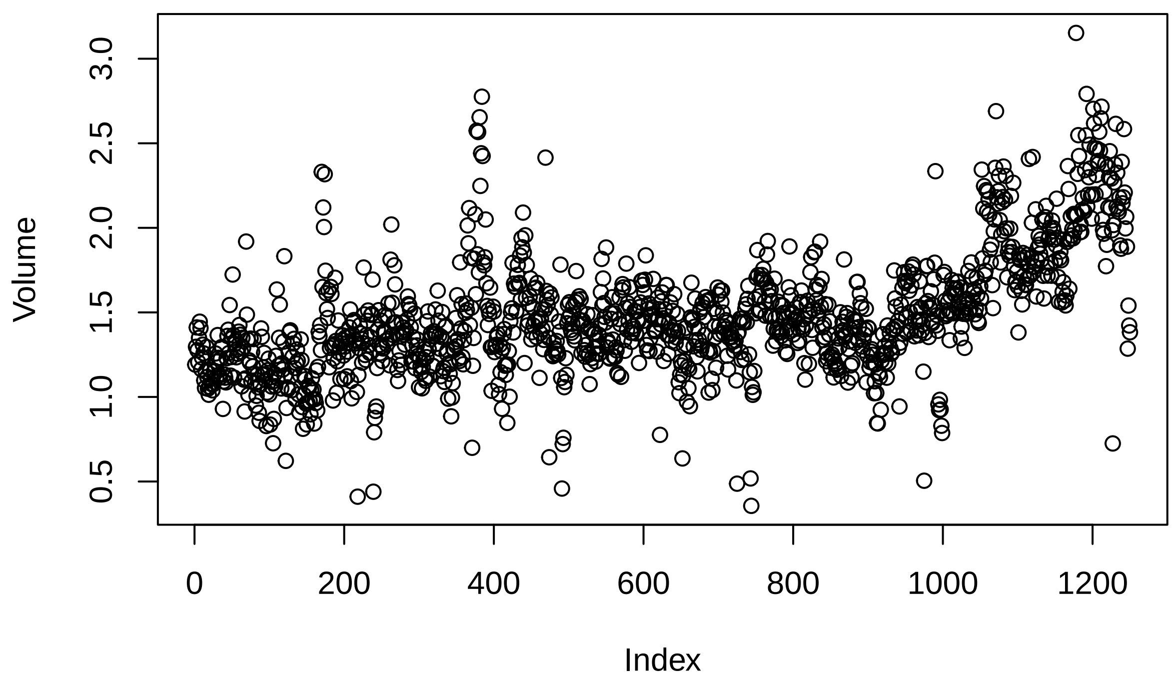
```
str(Smarket)
```

```
## 'data.frame':    1250 obs. of  9 variables:
##  $ Year     : num  2001 2001 2001 2001 2001 ...
##  $ Lag1     : num  0.381 0.959 1.032 -0.623 0.614 ...
##  $ Lag2     : num  -0.192 0.381 0.959 1.032 -0.623 ...
##  $ Lag3     : num  -2.624 -0.192 0.381 0.959 1.032 ...
##  $ Lag4     : num  -1.055 -2.624 -0.192 0.381 0.959 ...
##  $ Lag5     : num  5.01 -1.055 -2.624 -0.192 0.381 ...
##  $ Volume   : num  1.19 1.3 1.41 1.28 1.21 ...
##  $ Today    : num  0.959 1.032 -0.623 0.614 0.213 ...
##  $ Direction: Factor w/ 2 levels "Down","Up": 2 2 1 2 2 2 1 2 2 2 ...
```

```r
cor(Smarket[,-9])
```

```
##              Year         Lag1         Lag2         Lag3         Lag4
## Year   1.00000000  0.029699649  0.030596422  0.033194581  0.035688718
## Lag1   0.02969965  1.000000000 -0.026294328 -0.010803402 -0.002985911
## Lag2   0.03059642 -0.026294328  1.000000000 -0.025896670 -0.010853533
## Lag3   0.03319458 -0.010803402 -0.025896670  1.000000000 -0.024051036
## Lag4   0.03568872 -0.002985911 -0.010853533 -0.024051036  1.000000000
## Lag5   0.02978799 -0.005674606 -0.003557949 -0.018808338 -0.027083641
## Volume 0.53900647  0.040909908 -0.043383215 -0.041823686 -0.048414246
## Today  0.03009523 -0.026155045 -0.010250033 -0.002447647 -0.006899527
##                Lag5      Volume        Today
## Year    0.029787995  0.53900647  0.030095229
## Lag1   -0.005674606  0.04090991 -0.026155045
## Lag2   -0.003557949 -0.04338321 -0.010250033
## Lag3   -0.018808338 -0.04182369 -0.002447647
## Lag4   -0.027083641 -0.04841425 -0.006899527
## Lag5    1.000000000 -0.02200231 -0.034860083
## Volume -0.022002315  1.00000000  0.014591823
## Today  -0.034860083  0.01459182  1.000000000
```

```r
attach(Smarket)
plot(Volume)
```



Logistic Regression

```
glm.fits <- glm(Direction~Lag1+Lag2+Lag3+Lag4+Lag5+Volume,
                data=Smarket, family=binomial)
summary(glm.fits)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
##     Volume, family = binomial, data = Smarket)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.446  -1.203   1.065   1.145   1.326
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.126000   0.240736  -0.523    0.601
## Lag1        -0.073074   0.050167  -1.457    0.145
## Lag2        -0.042301   0.050086  -0.845    0.398
## Lag3         0.011085   0.049939   0.222    0.824
## Lag4         0.009359   0.049974   0.187    0.851
## Lag5         0.010313   0.049511   0.208    0.835
## Volume       0.135441   0.158360   0.855    0.392
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1731.2  on 1249  degrees of freedom
## Residual deviance: 1727.6  on 1243  degrees of freedom
## AIC: 1741.6
##
## Number of Fisher Scoring iterations: 3
```

```
glm.probs <- predict(glm.fits, type="response")
glm.probs[1:10]
```

```
##         1         2         3         4         5         6         7         8
## 0.5070841 0.4814679 0.4811388 0.5152224 0.5107812 0.5069565 0.4926509 0.5092292
##         9        10
## 0.5176135 0.4888378
```

The `type="response"` argument in `predict()` function tells `R` to output probabilities of the form $P(Y = 1|X)$, as opposed to other information such as the logit.

```
glm.pred <- rep("Down", nrow(Smarket))
glm.pred[glm.probs>.5] <- "Up"

## Confusion matrix
table(glm.pred, Direction)
```

```
##         Direction
```

```
## glm.pred Down  Up
##     Down  145 141
##     Up    457 507
```

```r
## Training error rate
mean(glm.pred!=Direction)
```

```
## [1] 0.4784
```

```r
train <- (Year<2005)
Smarket.2005 <- Smarket[!train, ]
dim(Smarket.2005)
```

```
## [1] 252   9
```

```r
Direction.2005 <- Direction[!train]
```

```r
glm.fits <- glm(Direction~Lag1+Lag2+Lag3+Lag4+Lag5+Volume,
                data=Smarket, family=binomial, subset=train)
glm.probs <- predict(glm.fits, Smarket.2005, type="response")
glm.pred <- rep("Down", nrow(Smarket.2005))
glm.pred[glm.probs>.5] <- "Up"

## Confusion matrix
table(glm.pred, Direction.2005)
```

```
##         Direction.2005
## glm.pred Down Up
##     Down   77 97
##     Up     34 44
```

```r
## Test error rate
mean(glm.pred!=Direction.2005)
```

```
## [1] 0.5198413
```

Linear Discriminant Analysis

```r
library(MASS)
lda.fit <- lda(Direction~Lag1+Lag2, data=Smarket, subset=train)
lda.fit
```

```
## Call:
## lda(Direction ~ Lag1 + Lag2, data = Smarket, subset = train)
##
## Prior probabilities of groups:
##     Down       Up
## 0.491984 0.508016
##
```

```
## Group means:
##             Lag1        Lag2
## Down  0.04279022  0.03389409
## Up   -0.03954635 -0.03132544
##
## Coefficients of linear discriminants:
##             LD1
## Lag1 -0.6420190
## Lag2 -0.5135293
```

The *coefficients of linear discriminants* output provides the linear combination of `Lag1` and `Lag2` that are used to form the LDA decision rule.

```
lda.pred <- predict(lda.fit, Smarket.2005)
names(lda.pred)
```

```
## [1] "class"     "posterior" "x"
```

```
lda.class <- lda.pred$class
table(lda.class, Direction.2005)
```

```
##          Direction.2005
## lda.class Down  Up
##      Down   35  35
##      Up     76 106
```

```
mean(lda.class==Direction.2005)
```

```
## [1] 0.5595238
```

```
sum(lda.pred$posterior[,1]>=.5)
```

```
## [1] 70
```

```
sum(lda.pred$posterior[,1]<.5)
```

```
## [1] 182
```

```
sum(lda.pred$posterior[,1]>=.9)
```

```
## [1] 0
```

```
sum(lda.pred$posterior[,1]<.9)
```

```
## [1] 252
```

Quadratic Discriminant Analysis

```r
qda.fit <- qda(Direction~Lag1+Lag2, data=Smarket, subset=train)
qda.fit
```

```
## Call:
## qda(Direction ~ Lag1 + Lag2, data = Smarket, subset = train)
##
## Prior probabilities of groups:
##     Down       Up
## 0.491984 0.508016
##
## Group means:
##             Lag1        Lag2
## Down  0.04279022  0.03389409
## Up   -0.03954635 -0.03132544
```

```r
qda.pred <- predict(qda.fit, Smarket.2005)
qda.class <- qda.pred$class

## Confusion matrix
table(qda.class, Direction.2005)
```

```
##          Direction.2005
## qda.class Down  Up
##      Down   30  20
##      Up     81 121
```

```r
mean(qda.class==Direction.2005)
```

```
## [1] 0.5992063
```

Resampling Methods

The Validation Set Approach

```r
library(ISLR)
data(Auto)
dim(Auto)
```

```
## [1] 392   9
```

```r
set.seed(1)
train <- sample(nrow(Auto), nrow(Auto)/2)
```

```r
attach(Auto)
lm.fit <- lm(mpg~horsepower, data=Auto, subset=train)
mean((mpg-predict(lm.fit, Auto))[-train]^2)
```

```
## [1] 23.26601
```

```r
lm.fit2 <- lm(mpg~poly(horsepower, 2), data=Auto, subset=train)
mean((mpg-predict(lm.fit2, Auto))[-train]^2)
```

```
## [1] 18.71646
```

Leave-One-Out Cross-Validation

```r
glm.fit <- glm(mpg~horsepower, data=Auto)
coef(glm.fit)
```

```
## (Intercept)  horsepower
##  39.9358610  -0.1578447
```

```r
lm.fit <- lm(mpg~horsepower, data=Auto)
coef(lm.fit)
```

```
## (Intercept)  horsepower
##  39.9358610  -0.1578447
```

```r
library(boot)
glm.fit <- glm(mpg~horsepower, data=Auto)
cv.err <- cv.glm(Auto, glm.fit)
cv.err$delta
```

```
## [1] 24.23151 24.23114
```

```r
cv.error <- rep(0, 5)
for (i in 1:5) {
  glm.fit <- glm(mpg~poly(horsepower, i), data=Auto)
  cv.error[i] <- cv.glm(Auto, glm.fit)$delta[1]
}
cv.error
```

```
## [1] 24.23151 19.24821 19.33498 19.42443 19.03321
```

k-Fold Cross-Validation

```r
## 10-fold cross-validation
set.seed(17)
cv.error.10 <- rep(0, 10)
for (i in 1:10) {
  glm.fit <- glm(mpg~poly(horsepower, i), data=Auto)
  cv.error.10[i] <- cv.glm(Auto, glm.fit, K=10)$delta[1]
}
cv.error.10
```

```
##  [1] 24.27207 19.26909 19.34805 19.29496 19.03198 18.89781 19.12061 19.14666
##  [9] 18.87013 20.95520
```

When we perform k-fold CV, the two numbers associated with `delta` differ slightly. The first is the standard k-fold CV estimate. The second is a bias-corrected version.

Bootstrap

```
boot.fn <- function(data, index)
  return(coef(lm(mpg~horsepower, data=data, subset=index)))

boot.fn(Auto, 1:nrow(Auto))
```

```
## (Intercept)  horsepower
##  39.9358610  -0.1578447
```

```
set.seed(1)
boot.fn(Auto, sample(nrow(Auto), nrow(Auto), replace=T))
```

```
## (Intercept)  horsepower
##  40.3404517  -0.1634868
```

```
boot.fn(Auto, sample(nrow(Auto), nrow(Auto), replace=T))
```

```
## (Intercept)  horsepower
##  40.1186906  -0.1577063
```

```
boot(Auto, boot.fn, R=1000)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Auto, statistic = boot.fn, R = 1000)
##
##
## Bootstrap Statistics :
##       original        bias    std. error
## t1* 39.9358610  0.0544513229 0.841289790
## t2* -0.1578447 -0.0006170901 0.007343073
```

```
coef(summary(lm(mpg~horsepower, data=Auto)))
```

```
##               Estimate  Std. Error   t value      Pr(>|t|)
## (Intercept) 39.9358610 0.717498656  55.65984 1.220362e-187
## horsepower  -0.1578447 0.006445501 -24.48914  7.031989e-81
```

Linear Model Selection and Regularization

Best Subset Selection

```r
library(ISLR)
data(Hitters)
sapply(Hitters, function(x) sum(is.na(x)))
```

```
##     AtBat      Hits     HmRun      Runs       RBI     Walks     Years    CAtBat
##         0         0         0         0         0         0         0         0
##     CHits    CHmRun     CRuns      CRBI    CWalks    League  Division   PutOuts
##         0         0         0         0         0         0         0         0
##   Assists    Errors    Salary NewLeague
##         0         0        59         0
```

```r
Hitters <- na.omit(Hitters)
```

```r
library(leaps)
regfit.full <- regsubsets(Salary~., Hitters)
summary(regfit.full)
```

```
## Subset selection object
## Call: regsubsets.formula(Salary ~ ., Hitters)
## 19 Variables  (and intercept)
##            Forced in Forced out
## AtBat          FALSE      FALSE
## Hits           FALSE      FALSE
## HmRun          FALSE      FALSE
## Runs           FALSE      FALSE
## RBI            FALSE      FALSE
## Walks          FALSE      FALSE
## Years          FALSE      FALSE
## CAtBat         FALSE      FALSE
## CHits          FALSE      FALSE
## CHmRun         FALSE      FALSE
## CRuns          FALSE      FALSE
## CRBI           FALSE      FALSE
## CWalks         FALSE      FALSE
## LeagueN        FALSE      FALSE
## DivisionW      FALSE      FALSE
## PutOuts        FALSE      FALSE
## Assists        FALSE      FALSE
## Errors         FALSE      FALSE
## NewLeagueN     FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: exhaustive
##          AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI
## 1  ( 1 ) " "   " "  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 2  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 3  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 4  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
```

```
## 5  ( 1 ) "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 6  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    " "   "*"
## 7  ( 1 ) " "   "*"  " "   " "  " " "*"   " "   "*"    "*"   "*"    " "   " "
## 8  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "   "*"    "*"   " "
##          CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1  ( 1 ) " "    " "     " "       " "     " "     " "    " "
## 2  ( 1 ) " "    " "     " "       " "     " "     " "    " "
## 3  ( 1 ) " "    " "     " "       "*"     " "     " "    " "
## 4  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
## 5  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
## 6  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
## 7  ( 1 ) " "    " "     "*"       "*"     " "     " "    " "
## 8  ( 1 ) "*"    " "     "*"       "*"     " "     " "    " "
```

Note:

By default, `regsubsets()` only reports results up to the best eight-variable model. But the `nvmax` option can be used to return as many variables as desired.

```r
regfit.full <- regsubsets(Salary~., data=Hitters, nvmax=19)
reg.summary <- summary(regfit.full)
names(reg.summary)
```

```
## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

```r
reg.summary$rsq
```

```
##  [1] 0.3214501 0.4252237 0.4514294 0.4754067 0.4908036 0.5087146 0.5141227
##  [8] 0.5285569 0.5346124 0.5404950 0.5426153 0.5436302 0.5444570 0.5452164
## [15] 0.5454692 0.5457656 0.5459518 0.5460945 0.5461159
```

```r
# par(mfrow=c(2, 2))
plot(reg.summary$rss, xlab="Number of Variables", ylab="RSS", type="l")
plot(reg.summary$adjr2, xlab="Number of Variables", ylab="Adjusted RSq", type="l")
which.max(reg.summary$adjr2)
```

```
## [1] 11
```

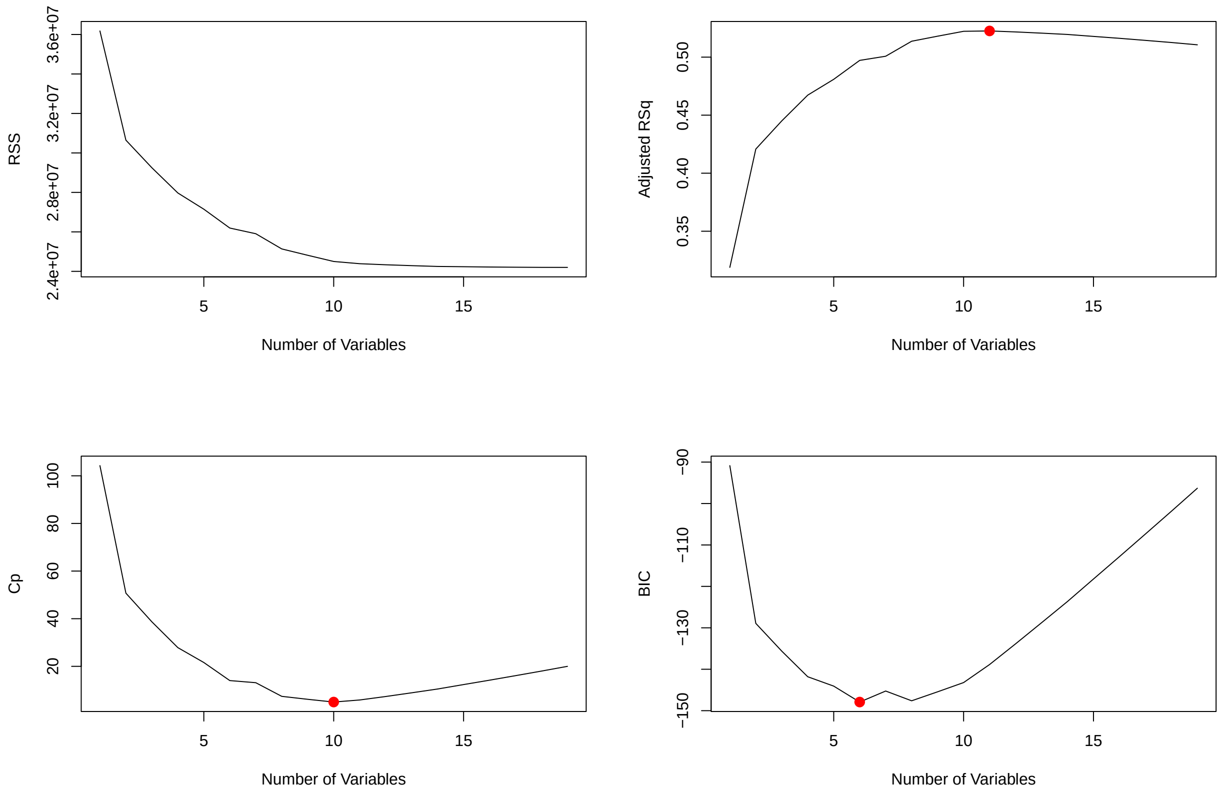
```r
points(11, reg.summary$adjr2[11], col="red", cex=2, pch=20)

plot(reg.summary$cp, xlab="Number of Variables", ylab="Cp", type="l")
which.min(reg.summary$cp)
```

```
## [1] 10
```

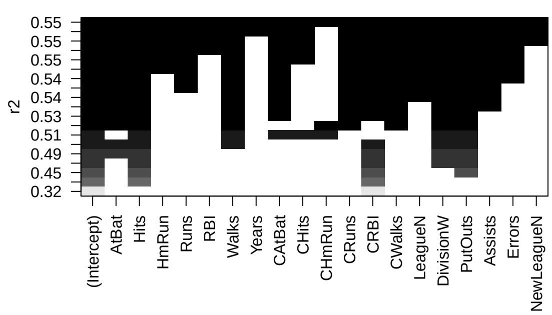
```r
points(10, reg.summary$cp[10], col="red", cex=2, pch=20)

plot(reg.summary$bic, xlab="Number of Variables", ylab="BIC", type="l")
which.min(reg.summary$bic)
```

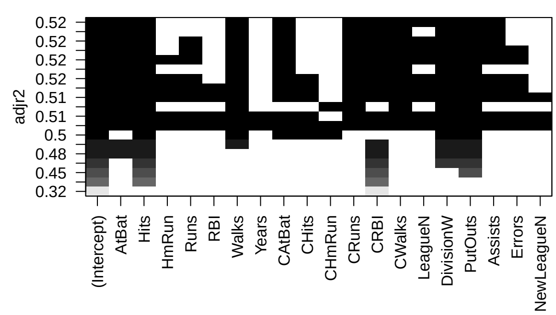
```
## [1] 6
```

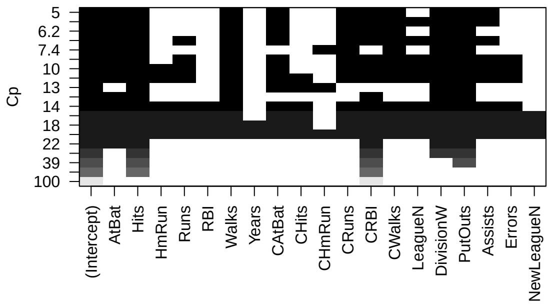
```r
points(6, reg.summary$bic[6], col="red", cex=2, pch=20)
```

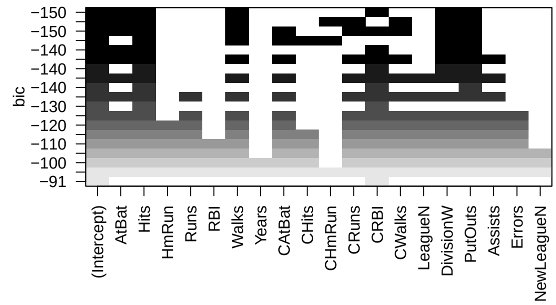



```r
plot(regfit.full, scale="r2")
plot(regfit.full, scale="adjr2")
plot(regfit.full, scale="Cp")
plot(regfit.full, scale="bic")
```










```r
coef(regfit.full, 6)
```

```
##  (Intercept)        AtBat         Hits        Walks         CRBI    DivisionW 
##   91.5117981   -1.8685892    7.6043976    3.6976468    0.6430169 -122.9515338 
##      PutOuts 
##    0.2643076
```

Forward and Backward Stepwise Selection

```r
regfit.fwd <- regsubsets(Salary~., data=Hitters, nvmax=19, method="forward")
regfit.bwd <- regsubsets(Salary~., data=Hitters, nvmax=19, method="backward")
```

```r
coef(regfit.full, 7)
```

```
##  (Intercept)         Hits        Walks       CAtBat        CHits       CHmRun 
##   79.4509472    1.2833513    3.2274264   -0.3752350    1.4957073    1.4420538 
##    DivisionW      PutOuts 
## -129.9866432    0.2366813
```

```r
coef(regfit.fwd, 7)
```

```
##  (Intercept)        AtBat         Hits        Walks         CRBI       CWalks 
##  109.7873062   -1.9588851    7.4498772    4.9131401    0.8537622   -0.3053070 
##    DivisionW      PutOuts 
## -127.1223928    0.2533404
```

```r
coef(regfit.bwd, 7)
```

```
##  (Intercept)        AtBat         Hits        Walks        CRuns       CWalks 
##  105.6487488   -1.9762838    6.7574914    6.0558691    1.1293095   -0.7163346 
##    DivisionW      PutOuts 
## -116.1692169    0.3028847
```

Choosing Among Models Using Cross-Validation

```r
set.seed(1)
train <- sample(c(TRUE, FALSE), nrow(Hitters), rep=TRUE)
test <- (!train)
```

```r
regfit.best <- regsubsets(Salary~., data=Hitters[train,], nvmax=19)
```

```r
test.mat <- model.matrix(Salary~., data=Hitters[test,])

val.errors <- rep(NA, 19)
for (i in 1:19) {
  coefi <- coef(regfit.best, id=i)
  pred <- test.mat[, names(coefi)]%*%coefi
  val.errors[i] <- mean((Hitters$Salary[test]-pred)^2)
}
val.errors
```

```
##  [1] 164377.3 144405.5 152175.7 145198.4 137902.1 139175.7 126849.0 136191.4
##  [9] 132889.6 135434.9 136963.3 140694.9 140690.9 141951.2 141508.2 142164.4
## [17] 141767.4 142339.6 142238.2
```

```r
which.min(val.errors)
```

```
## [1] 7
```

```r
coef(regfit.best, 7)
```

```
##  (Intercept)        AtBat         Hits        Walks        CRuns       CWalks
##   67.1085369   -2.1462987    7.0149547    8.0716640    1.2425113   -0.8337844
##    DivisionW      PutOuts
## -118.4364998    0.2526925
```

```r
## Define predict() method for regsubsets()
predict.regsubsets <- function(object, newdata, id, ...){
  form <- as.formula(object$call[[2]])
  mat <- model.matrix(form, newdata)
  coefi <- coef(object, id=id)
  xvars <- names(coefi)
  mat[, xvars]%*%coefi
}
```

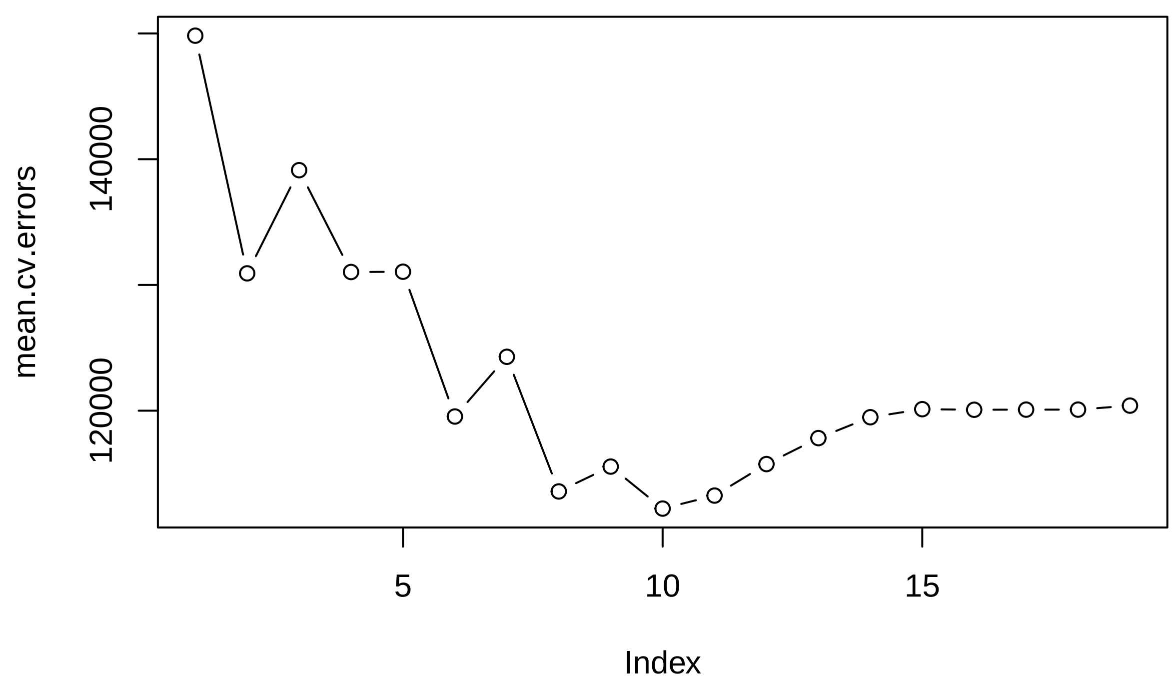
```r
k <- 10
set.seed(1)
folds <- sample(1:k, nrow(Hitters), replace=T)
cv.errors <- matrix(NA, k, 19, dimnames=list(NULL, paste(1:19)))

for (i in 1:k) {
  best.fit <- regsubsets(Salary~., data=Hitters[folds!=i,], nvmax=19)
  for (j in 1:19) {
```

```r
    pred <- predict(best.fit, Hitters[folds==i,], id=j)
    cv.errors[i, j] <- mean((Hitters$Salary[folds==i]-pred)^2)
  }
}
```

```r
## Use apply() function to average over the columns
mean.cv.errors <- apply(cv.errors, 2, mean)
mean.cv.errors
```

```
##        1        2        3        4        5        6        7        8
## 149821.1 130922.0 139127.0 131028.8 131050.2 119538.6 124286.1 113580.0
##        9       10       11       12       13       14       15       16
## 115556.5 112216.7 113251.2 115755.9 117820.8 119481.2 120121.6 120074.3
##       17       18       19
## 120084.8 120085.8 120403.5
```

```r
par(mfrow=c(1, 1))
plot(mean.cv.errors, type="b")
```




```r
reg.best <- regsubsets(Salary~., data=Hitters, nvmax=19)
coef(reg.best, which.min(mean.cv.errors))
```

```
##  (Intercept)        AtBat         Hits        Walks       CAtBat        CRuns
##  162.5354420   -2.1686501    6.9180175    5.7732246   -0.1300798    1.4082490
```

```
##         CRBI       CWalks     DivisionW      PutOuts      Assists
##    0.7743122   -0.8308264 -112.3800575    0.2973726    0.2831680
```

Ridge Regression and the Lasso

```r
x <- model.matrix(Salary~., Hitters)[,-1]
y <- Hitters$Salary
```

```r
library(glmnet)
```

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

```
## Loaded glmnet 4.1-2
```

```r
grid <- 10^seq(10, -2, length=100)

## Ridge regression
ridge.mod <- glmnet(x, y, alpha=0, lambda=grid)
```

```r
dim(coef(ridge.mod))
```

```
## [1]  20 100
```

```r
predict(ridge.mod, s=50, type="coefficients")[1:20,]
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
##  4.876610e+01 -3.580999e-01  1.969359e+00 -1.278248e+00  1.145892e+00
##           RBI         Walks         Years        CAtBat         CHits
##  8.038292e-01  2.716186e+00 -6.218319e+00  5.447837e-03  1.064895e-01
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##  6.244860e-01  2.214985e-01  2.186914e-01 -1.500245e-01  4.592589e+01
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
## -1.182011e+02  2.502322e-01  1.215665e-01 -3.278600e+00 -9.496680e+00
```

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

```r
ridge.mod <- glmnet(x[train,], y[train], alpha=0, lambda=grid, thresh=1e-12)
ridge.pred <- predict(ridge.mod, s=4, newx=x[test,])
mean((ridge.pred-y.test)^2)
```

```
## [1] 142199.2
```

```r
mean((mean(y[train])-y.test)^2)
```

```
## [1] 224669.9
```

```r
ridge.pred <- predict(ridge.mod, s=1e10, newx=x[test,])
mean((ridge.pred-y.test)^2)
```

```
## [1] 224669.8
```

```r
ridge.pred <- predict(ridge.mod, s=0, newx=x[test,], exact=T,
                      x=x[train,], y=y[train])
mean((ridge.pred-y.test)^2)
```

```
## [1] 168588.6
```

```r
lm(y~x, subset=train)
```

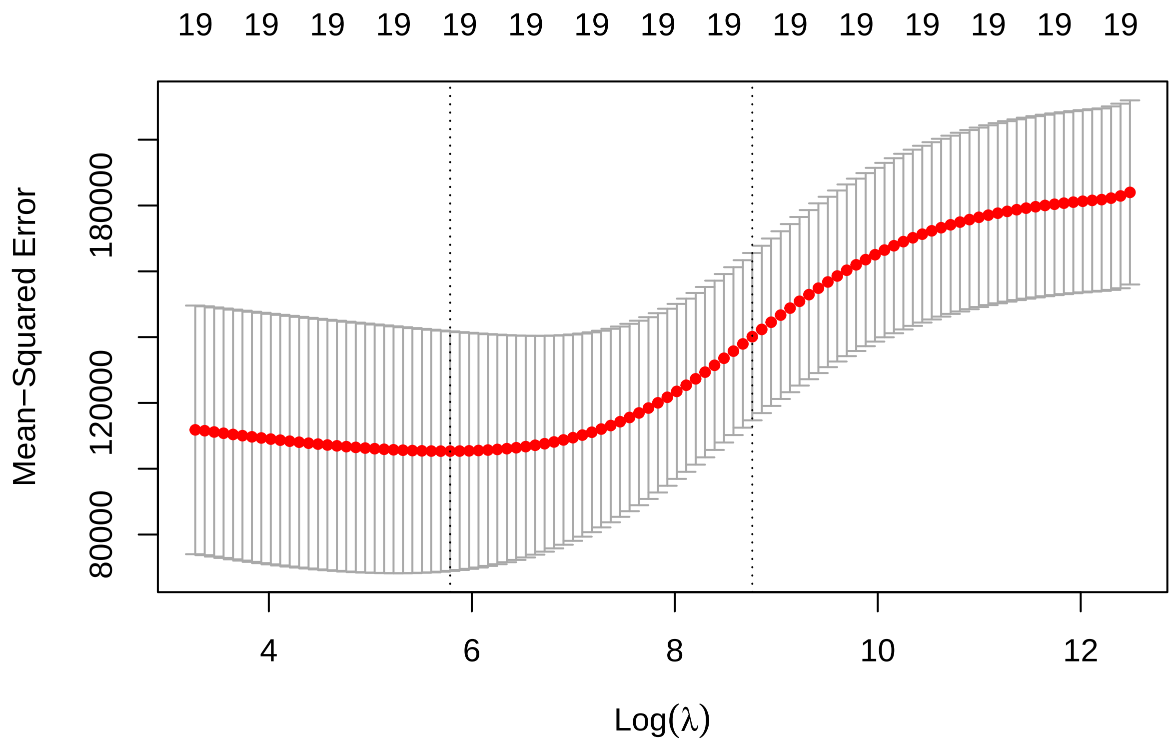
```
##
## Call:
## lm(formula = y ~ x, subset = train)
##
## Coefficients:
## (Intercept)       xAtBat        xHits       xHmRun        xRuns         xRBI
##    274.0145      -0.3521      -1.6377       5.8145       1.5424       1.1243
##      xWalks       xYears      xCAtBat       xCHits      xCHmRun       xCRuns
##      3.7287     -16.3773      -0.6412       3.1632       3.4008      -0.9739
##       xCRBI      xCWalks     xLeagueN   xDivisionW     xPutOuts     xAssists
##     -0.6005       0.3379     119.1486    -144.0831       0.1976       0.6804
##     xErrors  xNewLeagueN
##     -4.7128     -71.0951
```

```r
predict(ridge.mod, s=0, exact=T, type="coefficients",
        x=x[train,], y=y[train])[1:20,]
```

```
##  (Intercept)        AtBat         Hits        HmRun         Runs          RBI
##  274.0200994   -0.3521900   -1.6371383    5.8146692    1.5423361    1.1241837
##        Walks        Years       CAtBat        CHits       CHmRun        CRuns
##    3.7288406  -16.3795195   -0.6411235    3.1629444    3.4005281   -0.9739405
##         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists
##   -0.6003976    0.3378422  119.1434637 -144.0853061    0.1976300    0.6804200
##       Errors   NewLeagueN
##   -4.7127879  -71.0898914
```

```
## [1] 168593.3
```

```r
set.seed(1)
cv.out <- cv.glmnet(x[train,], y[train], alpha=0)
plot(cv.out)
```




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

```
## [1] 326.0828
```

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

```
## [1] 139856.6
```

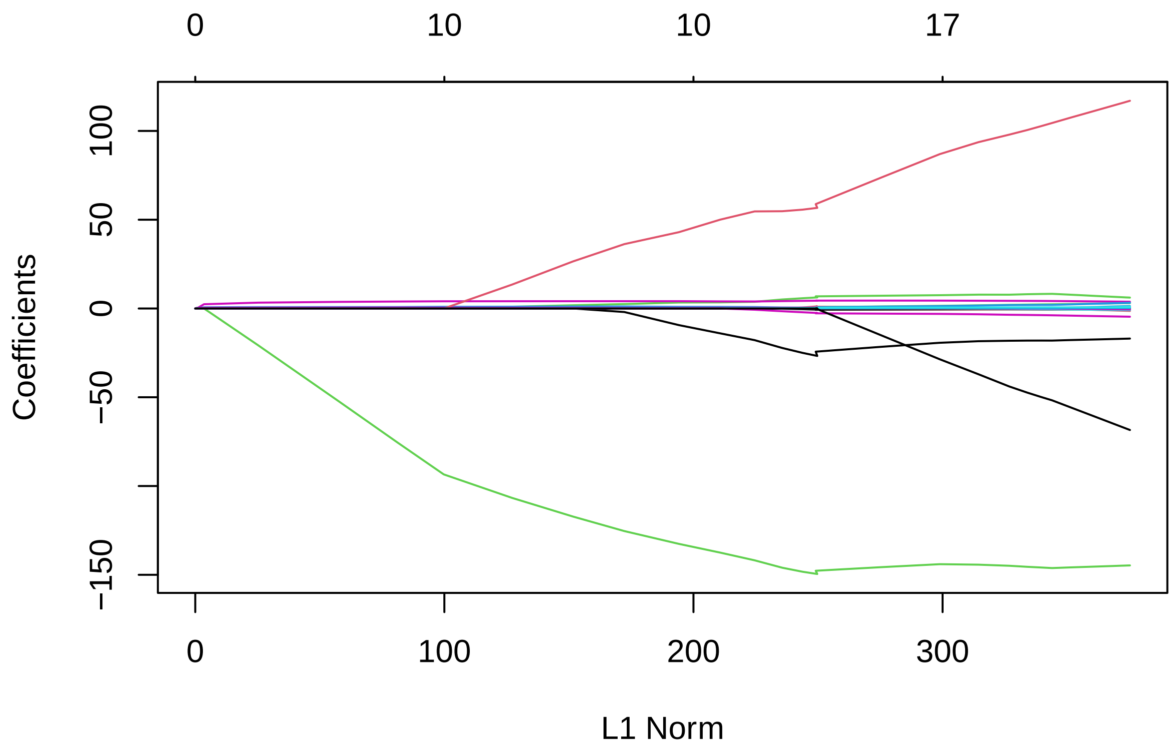
```
out <- glmnet(x, y, alpha=0)
predict(out, type="coefficients", s=bestlam)[1:20, ]
```

```
##  (Intercept)        AtBat         Hits        HmRun         Runs          RBI
##  15.44383135   0.07715547   0.85911581   0.60103107   1.06369007   0.87936105
##        Walks        Years       CAtBat        CHits       CHmRun        CRuns
##   1.62444616   1.35254780   0.01134999   0.05746654   0.40680157   0.11456224
##         CRBI       CWalks      LeagueN    DivisionW      PutOuts      Assists
##   0.12116504   0.05299202  22.09143189 -79.04032637   0.16619903   0.02941950
##       Errors   NewLeagueN
##  -1.36092945   9.12487767
```

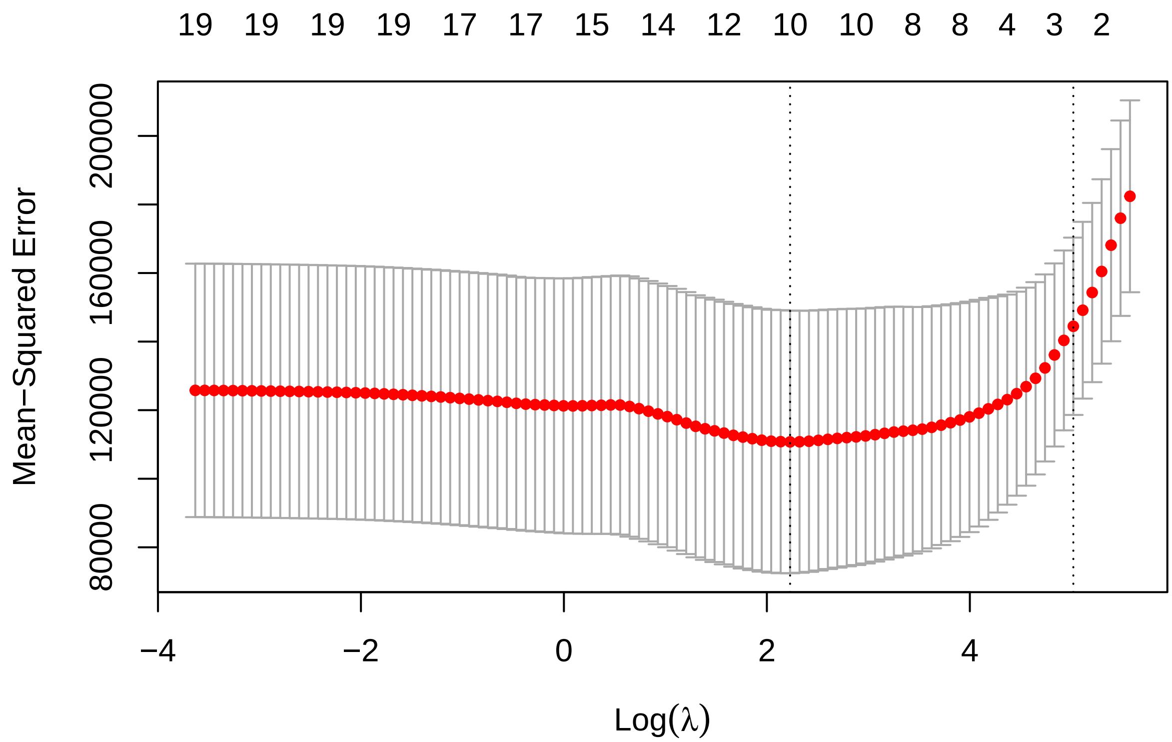
The Lasso

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

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```




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




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

```
## [1] 143673.6
```

```
out <- glmnet(x, y, alpha=1, lambda=grid)
lasso.coef <- predict(out, type="coefficients", s=bestlam)[1:20,]
lasso.coef
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
##    1.27479059   -0.05497143    2.18034583    0.00000000    0.00000000
##           RBI         Walks         Years        CAtBat         CHits
##    0.00000000    2.29192406   -0.33806109    0.00000000    0.00000000
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##    0.02825013    0.21628385    0.41712537    0.00000000   20.28615023
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
## -116.16755870    0.23752385    0.00000000   -0.85629148    0.00000000
```

```
lasso.coef[lasso.coef!=0]
```

```
##   (Intercept)         AtBat          Hits         Walks         Years
##    1.27479059   -0.05497143    2.18034583    2.29192406   -0.33806109
##        CHmRun         CRuns          CRBI       LeagueN     DivisionW
```

```
##    0.02825013    0.21628385    0.41712537   20.28615023 -116.16755870
##       PutOuts        Errors
##    0.23752385   -0.85629148
```

Note: the lasso has a substantial advantage over ridge regression in that the resulting coefficient estimates are sparse.

Moving Beyond Linearity

```r
library(ISLR)
attach(Wage)
```

```
## The following object is masked from Auto:
##
##     year

## The following object is masked from Boston:
##
##     age
```

Polynomial Regression and Step Functions

```r
fit <- lm(wage~poly(age, 4), data=Wage)
coef(summary(fit))
```

```
##                  Estimate Std. Error    t value     Pr(>|t|)
## (Intercept)     111.70361  0.7287409 153.283015 0.000000e+00
## poly(age, 4)1   447.06785 39.9147851  11.200558 1.484604e-28
## poly(age, 4)2  -478.31581 39.9147851 -11.983424 2.355831e-32
## poly(age, 4)3   125.52169 39.9147851   3.144742 1.678622e-03
## poly(age, 4)4   -77.91118 39.9147851  -1.951938 5.103865e-02
```

The `poly()` function returns a matrix whose columns are a basis of *orthogonal polynomials*, which essentially means that each column is a linear combination of variables `age`, `age^2`, `age^3` and `age^4`.

However, we can also use `poly()` to obtain `age`, `age^2`, `age^3` and `age^4` directly. We can do this by using the `raw=TRUE` argument to the `poly()` function. This does not affect the model in a meaningful way. Though the choice of basis clearly affects the coefficient estimates, it does not affect the fitted values obtained.

```r
fit2 <- lm(wage~poly(age, 4, raw=T), data=Wage)
coef(summary(fit2))
```

```
##                             Estimate   Std. Error   t value     Pr(>|t|)
## (Intercept)            -1.841542e+02 6.004038e+01 -3.067172 0.0021802539
## poly(age, 4, raw = T)1  2.124552e+01 5.886748e+00  3.609042 0.0003123618
## poly(age, 4, raw = T)2 -5.638593e-01 2.061083e-01 -2.735743 0.0062606446
## poly(age, 4, raw = T)3  6.810688e-03 3.065931e-03  2.221409 0.0263977518
## poly(age, 4, raw = T)4 -3.203830e-05 1.641359e-05 -1.951938 0.0510386498
```

```r
## Equivalent ways of fitting this model
fit2a <- lm(wage~age+I(age^2)+I(age^3)+I(age^4), data=Wage)
coef(fit2a)
```

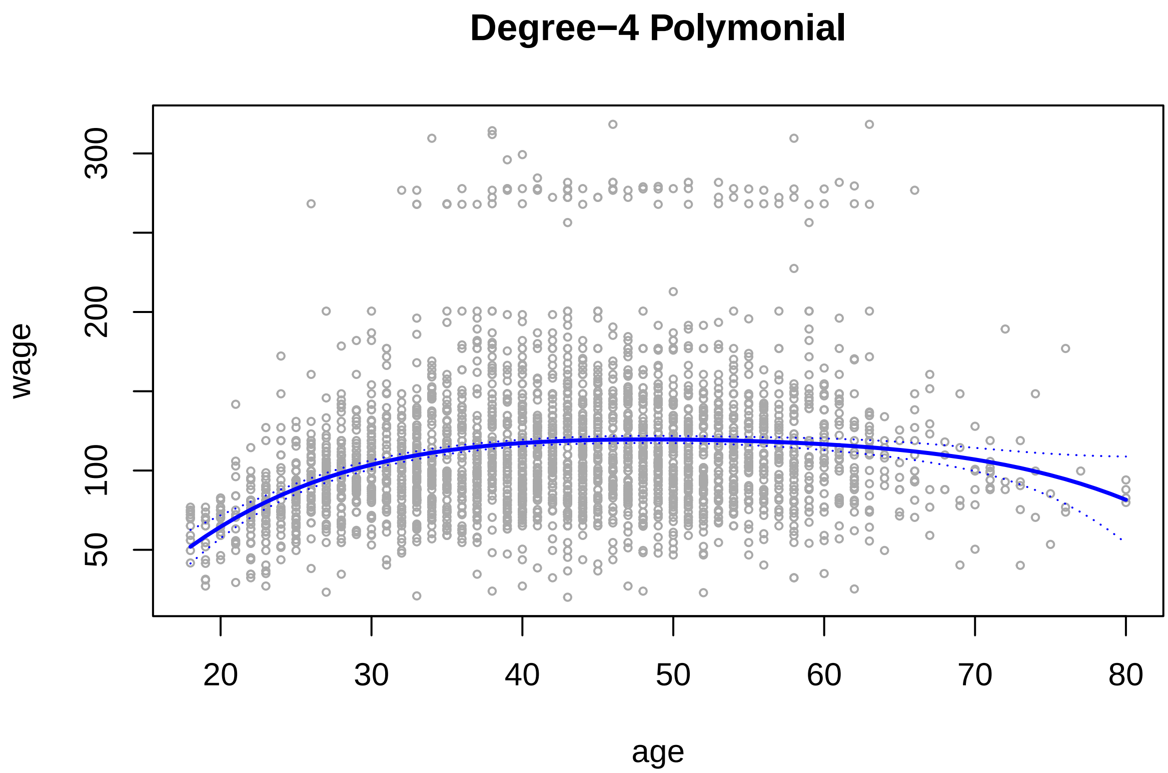
```
##   (Intercept)           age      I(age^2)      I(age^3)      I(age^4)
## -1.841542e+02  2.124552e+01 -5.638593e-01  6.810688e-03 -3.203830e-05
```

```r
fit2b <- lm(wage~cbind(age, age^2, age^3, age^4), data=Wage)
coef(fit2b)[1:5]
```

```
##                        (Intercept) cbind(age, age^2, age^3, age^4)age
##                      -1.841542e+02                       2.124552e+01
##    cbind(age, age^2, age^3, age^4)    cbind(age, age^2, age^3, age^4)
##                      -5.638593e-01                       6.810688e-03
##    cbind(age, age^2, age^3, age^4)
##                      -3.203830e-05
```

```r
agelims <- range(age)
age.grid <- seq(from=agelims[1], to=agelims[2])
preds <- predict(fit, newdata=list(age=age.grid), se=TRUE)
se.bands <- cbind(preds$fit+2*preds$se.fit, preds$fit-2*preds$se.fit)
```

```r
# par(mfrow=c(1, 2))
plot(age, wage, xlim=agelims, cex=.5, col="darkgrey")
title("Degree-4 Polymonial")
lines(age.grid, preds$fit, lwd=2, col="blue")
matlines(age.grid, se.bands, lwd=1, col="blue", lty=3)
```




```
fit.1 <- lm(wage~age, data=Wage)
fit.2 <- lm(wage~poly(age, 2), data=Wage)
fit.3 <- lm(wage~poly(age, 3), data=Wage)
fit.4 <- lm(wage~poly(age, 4), data=Wage)
fit.5 <- lm(wage~poly(age, 5), data=Wage)
anova(fit.1, fit.2, fit.3, fit.4, fit.5)
```

```
## Analysis of Variance Table
##
## Model 1: wage ~ age
## Model 2: wage ~ poly(age, 2)
## Model 3: wage ~ poly(age, 3)
## Model 4: wage ~ poly(age, 4)
## Model 5: wage ~ poly(age, 5)
##   Res.Df     RSS Df Sum of Sq        F    Pr(>F)
## 1   2998 5022216
## 2   2997 4793430  1    228786 143.5931 < 2.2e-16 ***
## 3   2996 4777674  1     15756   9.8888  0.001679 **
## 4   2995 4771604  1      6070   3.8098  0.051046 .
## 5   2994 4770322  1      1283   0.8050  0.369682
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
coef(summary(fit.5))
```

```
##                   Estimate Std. Error     t value     Pr(>|t|)
## (Intercept)    111.70361  0.7287647 153.2780243 0.000000e+00
## poly(age, 5)1  447.06785 39.9160847  11.2001930 1.491111e-28
## poly(age, 5)2 -478.31581 39.9160847 -11.9830341 2.367734e-32
## poly(age, 5)3  125.52169 39.9160847   3.1446392 1.679213e-03
## poly(age, 5)4  -77.91118 39.9160847  -1.9518743 5.104623e-02
## poly(age, 5)5  -35.81289 39.9160847  -0.8972045 3.696820e-01
```

```
fit.1 <- lm(wage~education+age, data=Wage)
fit.2 <- lm(wage~education+poly(age, 2), data=Wage)
fit.3 <- lm(wage~education+poly(age, 3), data=Wage)
anova(fit.1, fit.2, fit.3)
```

```
## Analysis of Variance Table
##
## Model 1: wage ~ education + age
## Model 2: wage ~ education + poly(age, 2)
## Model 3: wage ~ education + poly(age, 3)
##   Res.Df     RSS Df Sum of Sq        F Pr(>F)
## 1   2994 3867992
## 2   2993 3725395  1    142597 114.6969 <2e-16 ***
## 3   2992 3719809  1      5587   4.4936 0.0341 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
## Logistic regression
fit <- glm(I(wage>250)~poly(age,4), data=Wage, family=binomial)
preds <- predict(fit, newdata=list(age=age.grid), se=T)
```

The default prediction type for a `glm()` model is `type="link"`, which is what we use here. This means we get predictions for the *logit*: that is, we have fit a model of the form

$$\log\left(\frac{\Pr(Y=1|X)}{1-\Pr(Y=1|X)}\right) = X\beta$$

In order to obtain confidence intervals for $\Pr(Y=1|X)$, we use the transformation

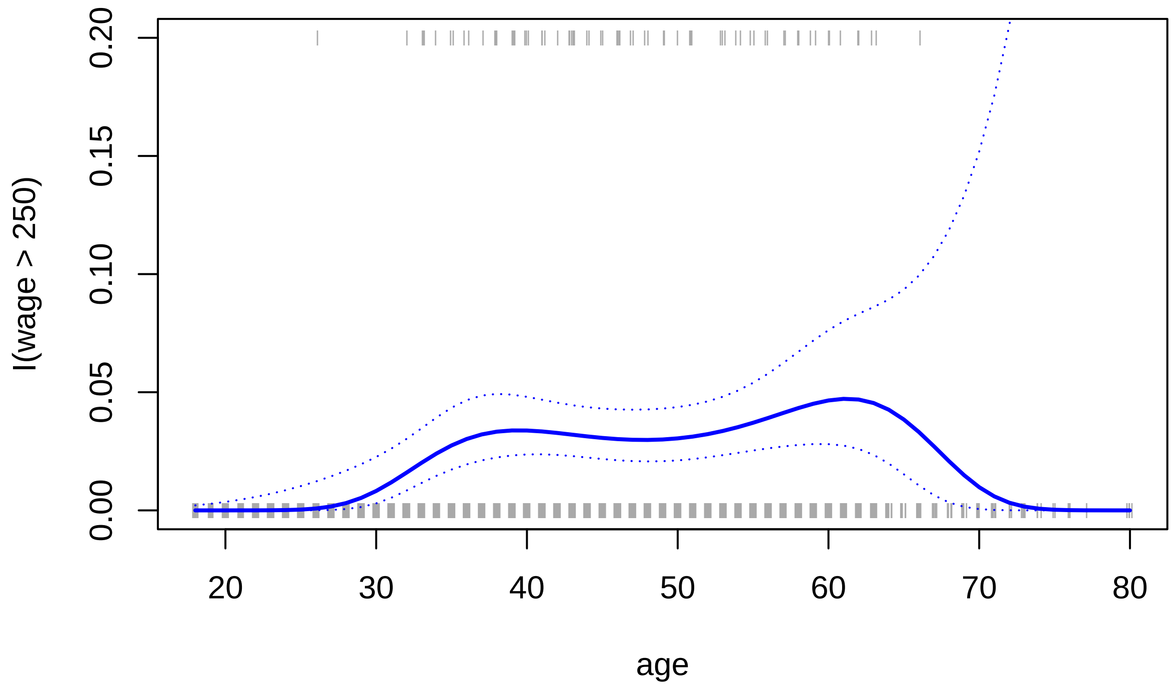
$$\Pr(Y=1|X) = \frac{\exp(X\beta)}{1+\exp(X\beta)}$$

```
pfit <- exp(preds$fit)/(1+exp(preds$fit))
se.bands.logit <- cbind(preds$fit+2*preds$se.fit, preds$fit-2*preds$se.fit)
se.bands <- exp(se.bands.logit)/(1+exp(se.bands.logit))
```

Note that we could have directly computed the probabilities by selecting the `type="response"` option in the `predict()` function.

```
preds <- predict(fit, newdata=list(age=age.grid), type="response", se=T)
```

```
plot(age, I(wage>250), xlim=agelims, type="n", ylim=c(0, .2))
points(jitter(age), I((wage>250)/5), cex=.5, pch="|", col="darkgrey")
lines(age.grid, pfit, lwd=2, col="blue")
matlines(age.grid, se.bands,lwd=1, col="blue", lty=3)
```




```
table(cut(age,4))
```

```
##
## (17.9,33.5]   (33.5,49]   (49,64.5] (64.5,80.1]
##         750        1399         779          72
```

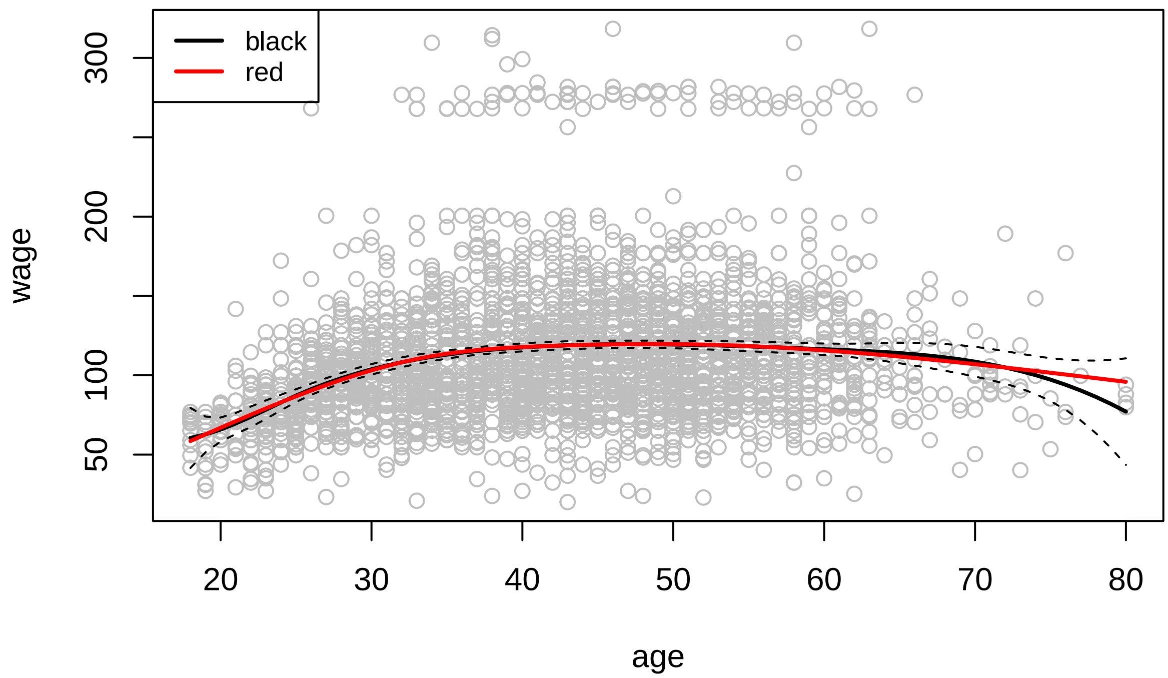
```
fit <- lm(wage~cut(age, 4), data=Wage)
coef(summary(fit))
```

```
##                         Estimate Std. Error   t value     Pr(>|t|)
## (Intercept)            94.158392   1.476069 63.789970 0.000000e+00
## cut(age, 4)(33.5,49]   24.053491   1.829431 13.148074 1.982315e-38
## cut(age, 4)(49,64.5]   23.664559   2.067958 11.443444 1.040750e-29
## cut(age, 4)(64.5,80.1]  7.640592   4.987424  1.531972 1.256350e-01
```

Splines

```
library(splines)

fit <- lm(wage~bs(age, knots=c(25,40,60)), data=Wage)
pred <- predict(fit, newdata=list(age=age.grid), se=T)
plot(age, wage, col="gray")
```

```r
lines(age.grid, pred$fit, lwd=2)
lines(age.grid, pred$fit+2*pred$se, lty="dashed")
lines(age.grid, pred$fit-2*pred$se, lty="dashed")

## Natural spline
fit2 <- lm(wage~ns(age, df=4), data=Wage)
pred2 <- predict(fit2, newdata=list(age=age.grid), se=T)
lines(age.grid, pred2$fit, col="red", lwd=2)
legend("topleft", legend=c("black", "red"), lwd=2, lty=1,
       cex=.8, col=c("black", "red"))
```



Here we have prespecified knots at ages 25, 40, and 60. This produces a spline with six basis functions. (Recall that a cubic spline with three knots has seven degrees of freedom; these degrees of freedom are used up by an intercept, plus six basis functions.) We could also use the `df` option to produce a spline with knots at uniform quantiles of the data.

```r
dim(bs(age, knots=c(25, 40, 60)))
```

```
## [1] 3000    6
```

```r
dim(bs(age, df=6))
```

```
## [1] 3000    6
```

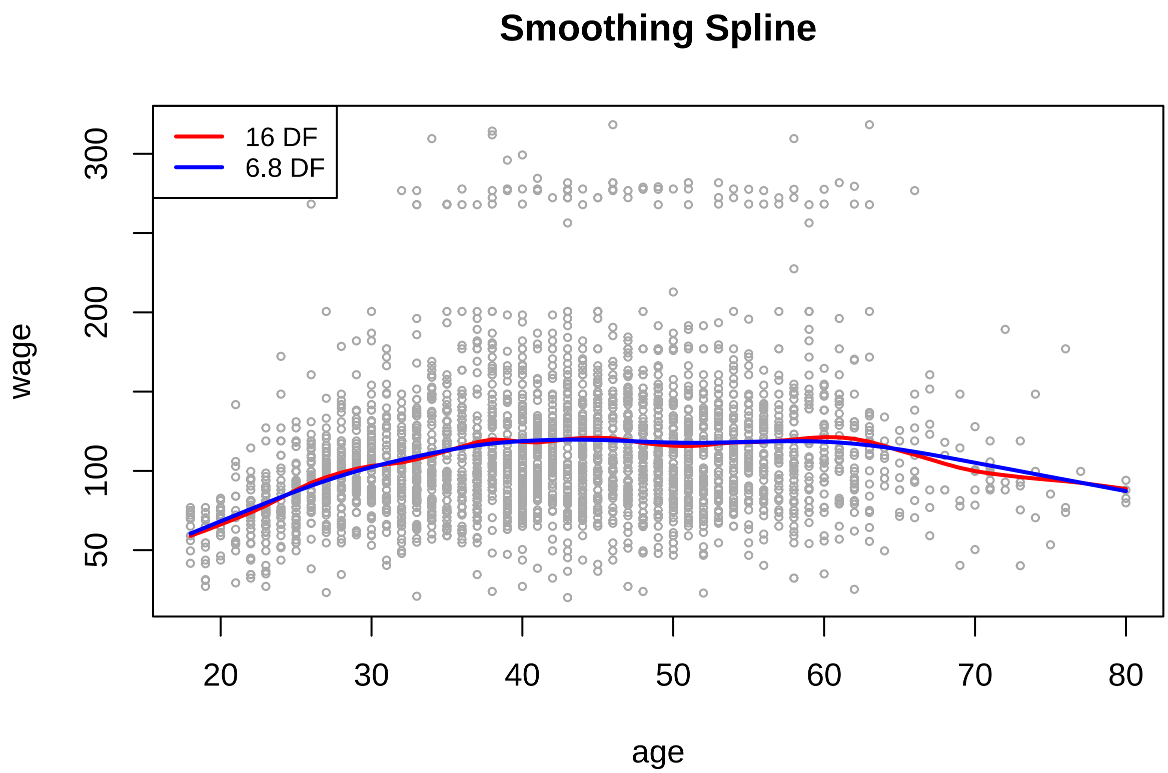
```
attr(bs(age, df=6), "knots")
```

```
##   25%   50%   75%
## 33.75 42.00 51.00
```

```
## Smoothing spline
plot(age, wage, xlim=agelims, cex=.5, col="darkgrey")
title("Smoothing Spline")
fit <- smooth.spline(age, wage, df=16)
fit2 <- smooth.spline(age, wage, cv=TRUE)
fit2$df
```

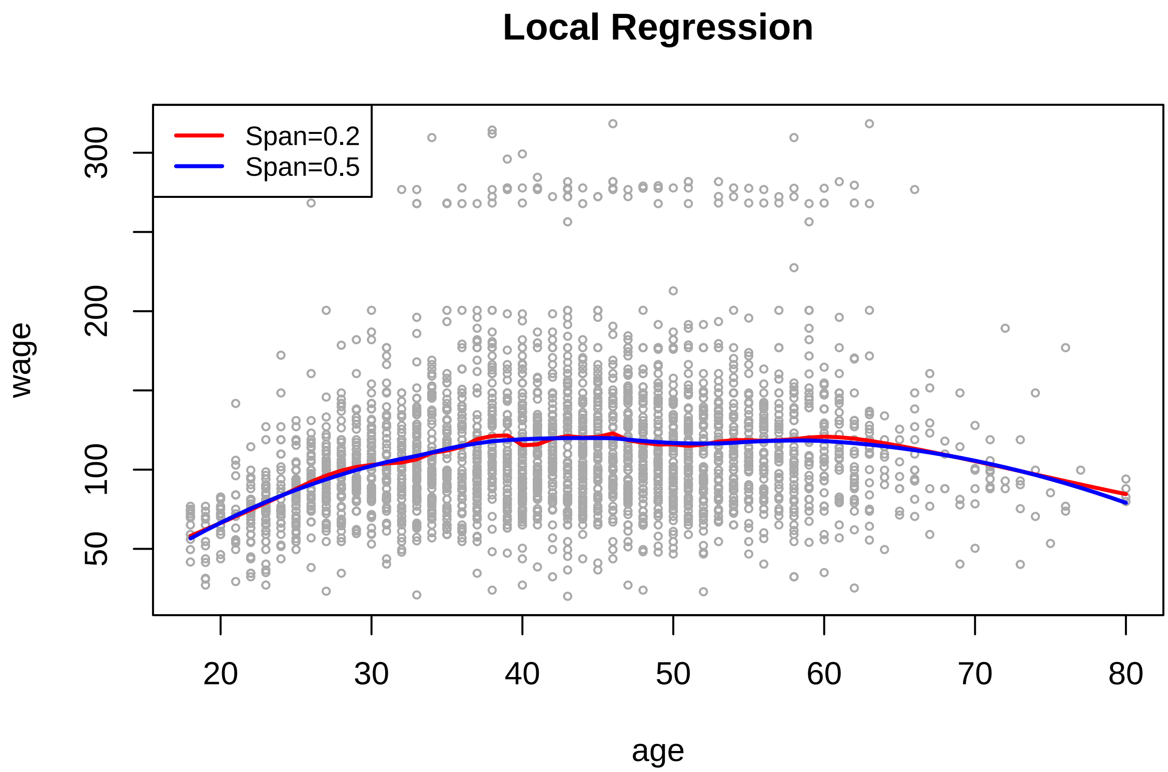
```
## [1] 6.794596
```

```
lines(fit, col="red", lwd=2)
lines(fit2, col="blue", lwd=2)
legend("topleft", legend=c("16 DF", "6.8 DF"),
       col=c("red", "blue"), lty=1, lwd=2, cex=.8)
```




```
plot(age, wage, xlim=agelims, cex=.5, col="darkgrey")
title("Local Regression")
fit <- loess(wage~age, span=.2, data=Wage)
fit2 <- loess(wage~age, span=.5, data=Wage)
```

```r
lines(age.grid, predict(fit, data.frame(age=age.grid)), col="red", lwd=2)
lines(age.grid, predict(fit2, data.frame(age=age.grid)), col="blue", lwd=2)
legend("topleft", legend=c("Span=0.2", "Span=0.5"),
       col=c("red", "blue"), lty=1, lwd=2, cex=.8)
```



Here we have performed local linear regression using spans of 0.2 and 0.5: that is, each neighborhood consists of 20% or 50% of the observations. The larger the span, the smoother the fit.

GAMs

```r
gam1 <- lm(wage~ns(year,4)+ns(age,5)+education, data=Wage)
```

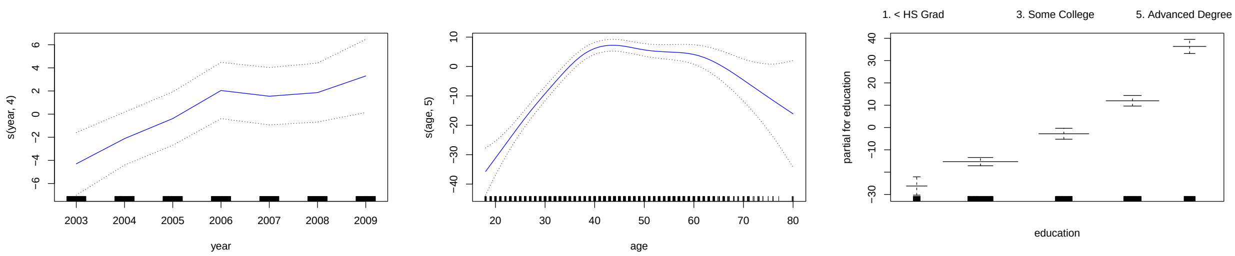
```r
library(gam)
```

```
## Loading required package: foreach
```

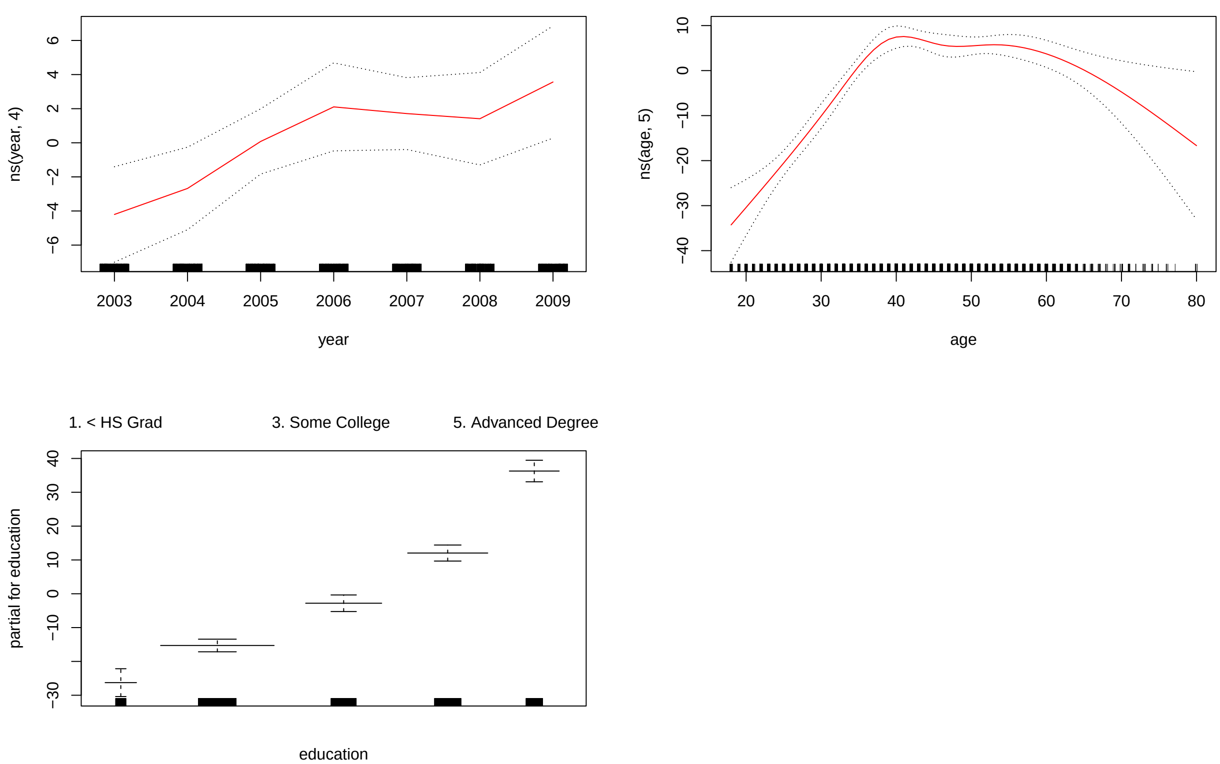
```
## Loaded gam 1.20
```

```r
gam.m3 <- gam(wage~s(year, 4)+s(age, 5)+education, data=Wage)
```

```r
plot(gam.m3, se=TRUE, col="blue")
```




```r
plot.Gam(gam1, se=TRUE, col="red")
```



In these plots, the function of `year` looks rather linear. We can perform a series of ANOVA tests in order to determine which of three models is best: a GAM that excludes `year` ($\mathcal{M}_1$), a GAM that uses a linear function of `year` ($\mathcal{M}_2$), or a GAM that uses a spline function of `year` ($\mathcal{M}_3$)

```r
gam.m1 <- gam(wage~s(age, 5)+education, data=Wage)
gam.m2 <- gam(wage~year+s(age, 5)+education, data=Wage)
anova(gam.m1, gam.m2, gam.m3, test="F")
```

```
## Analysis of Deviance Table
##
```

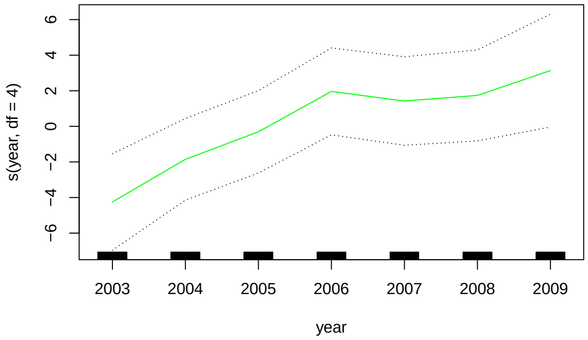
```
## Model 1: wage ~ s(age, 5) + education
## Model 2: wage ~ year + s(age, 5) + education
## Model 3: wage ~ s(year, 4) + s(age, 5) + education
##   Resid. Df Resid. Dev Df Deviance       F    Pr(>F)    
## 1      2990    3711731                                  
## 2      2989    3693842  1  17889.2 14.4771 0.0001447 ***
## 3      2986    3689770  3   4071.1  1.0982 0.3485661    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

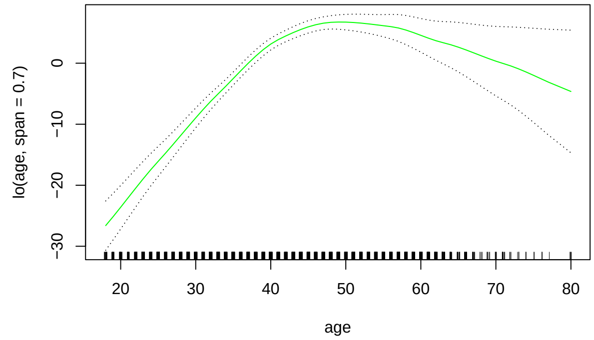
```r
summary(gam.m3)
```

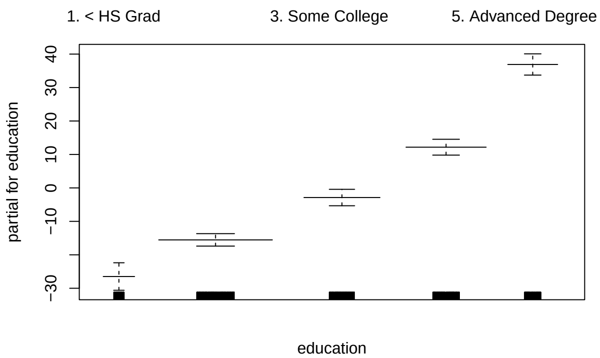
```
## 
## Call: gam(formula = wage ~ s(year, 4) + s(age, 5) + education, data = Wage)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max 
## -119.43  -19.70   -3.33   14.17  213.48 
## 
## (Dispersion Parameter for gaussian family taken to be 1235.69)
## 
##     Null Deviance: 5222086 on 2999 degrees of freedom
## Residual Deviance: 3689770 on 2986 degrees of freedom
## AIC: 29887.75 
## 
## Number of Local Scoring Iterations: NA 
## 
## Anova for Parametric Effects
##              Df  Sum Sq Mean Sq F value    Pr(>F)    
## s(year, 4)    1   27162   27162  21.981 2.877e-06 ***
## s(age, 5)     1  195338  195338 158.081 < 2.2e-16 ***
## education     4 1069726  267432 216.423 < 2.2e-16 ***
## Residuals  2986 3689770    1236                      
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Anova for Nonparametric Effects
##             Npar Df Npar F  Pr(F)    
## (Intercept)                          
## s(year, 4)        3  1.086 0.3537    
## s(age, 5)         4 32.380 <2e-16 ***
## education                            
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
preds <- predict(gam.m2, newdata=Wage)
```

```r
## Local regression
gam.lo <- gam(wage~s(year, df=4)+lo(age, span=0.7)+education, data=Wage)
plot.Gam(gam.lo, se=TRUE, col="green")
```



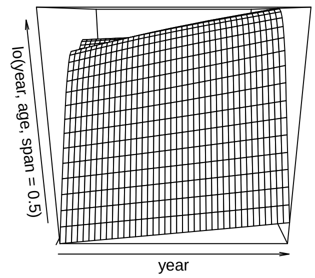


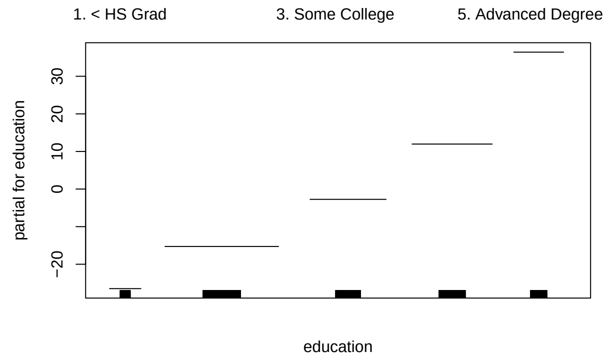



```r
## Local regression with interactions
gam.lo.i <- gam(wage~lo(year, age, span=0.5)+education, data=Wage)
```

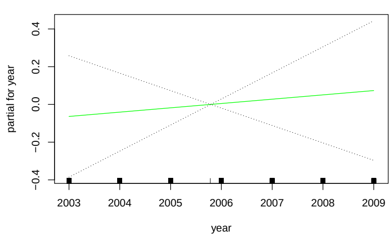
```
## Warning in lo.wam(x, z, wz, fit$smooth, which, fit$smooth.frame, bf.maxit, : liv
## too small. (Discovered by lowesd)

## Warning in lo.wam(x, z, wz, fit$smooth, which, fit$smooth.frame, bf.maxit, : lv
## too small. (Discovered by lowesd)

## Warning in lo.wam(x, z, wz, fit$smooth, which, fit$smooth.frame, bf.maxit, : liv
## too small. (Discovered by lowesd)

## Warning in lo.wam(x, z, wz, fit$smooth, which, fit$smooth.frame, bf.maxit, : lv
## too small. (Discovered by lowesd)
```

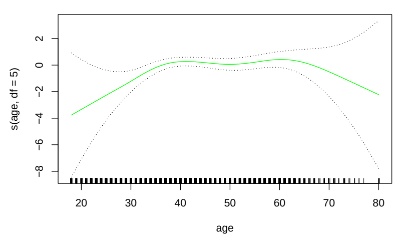
```r
library(akima)
plot(gam.lo.i)
```

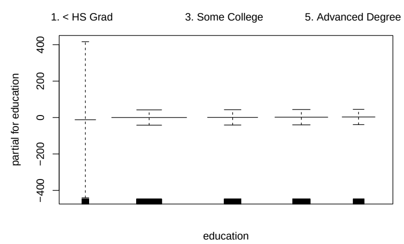





```
## Logistic regression GAM
gam.lr <- gam(I(wage>250)~year+s(age, df=5)+education, family=binomial, data=Wage)
# par(mfrow=c(1, 3))
plot(gam.lr, se=T, col="green")
```



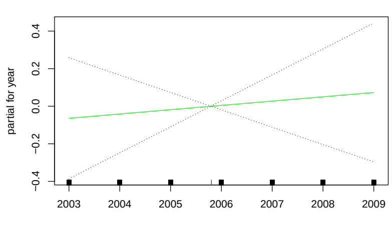


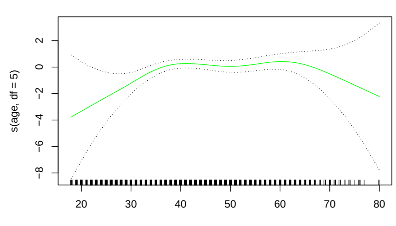


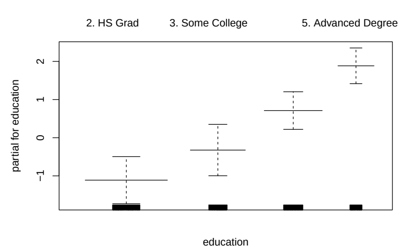

```
table(education, I(wage>250))
```

```
##
## education            FALSE TRUE
##    1. < HS Grad        268    0
##    2. HS Grad          966    5
##    3. Some College     643    7
##    4. College Grad     663   22
##    5. Advanced Degree  381   45
```

```
gam.lr.s <- gam(I(wage>250)~year+s(age, df=5)+education, family=binomial,
                data=Wage, subset=(education!="1. < HS Grad"))
plot(gam.lr.s, se=T, col="green")
```







Tree-Based Methods

```
library(tree)
library(ISLR)
attach(Carseats)
```

```
## The following objects are masked from Carseats (pos = 15):
##
##     Advertising, Age, CompPrice, Education, Income, Population, Price,
##     Sales, ShelveLoc, Urban, US
```

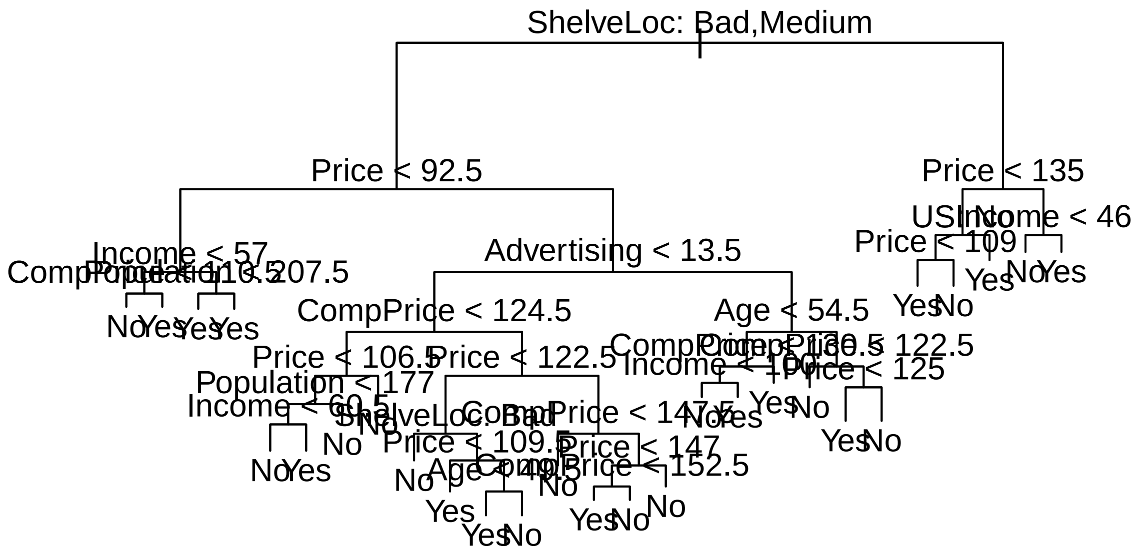
Fitting Classification Tree

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

```
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
```

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




```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
```

```r
mean(tree.pred==High.test)
```

```
## [1] 0.77
```

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

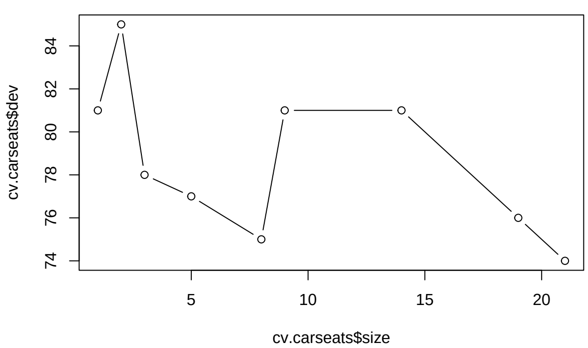
```
## [1] "size"   "dev"    "k"      "method"
```

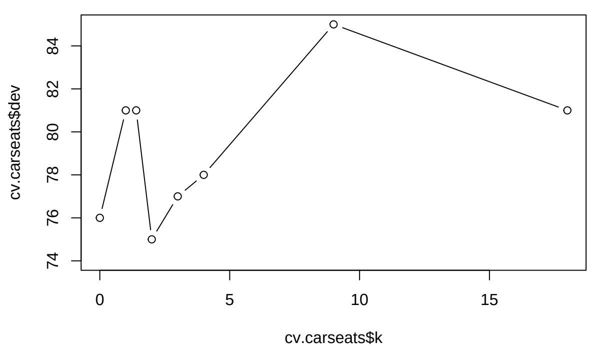
```
cv.carseats
```

```
## $size
## [1] 21 19 14  9  8  5  3  2  1
##
## $dev
## [1] 74 76 81 81 75 77 78 85 81
##
## $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"
```

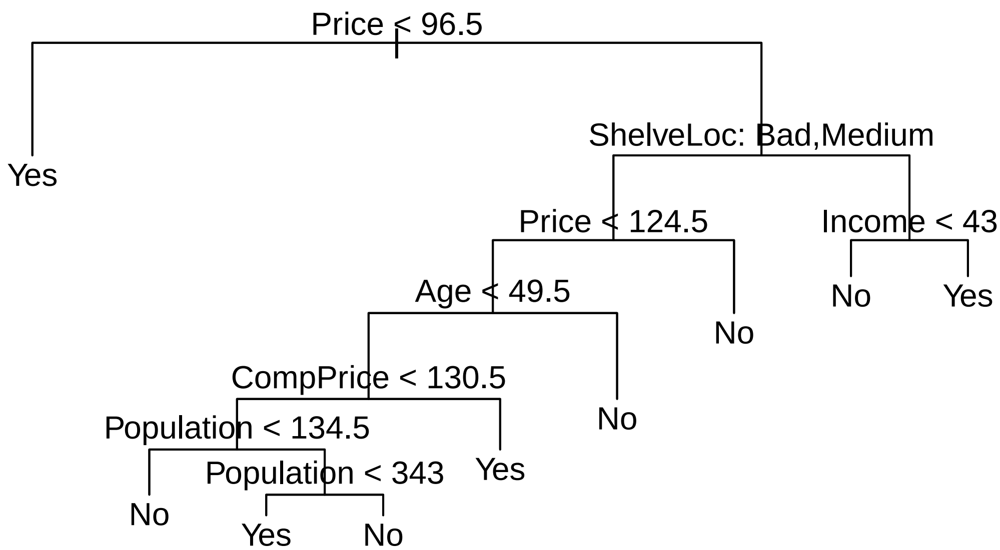
Note that despite the name, `dev` corresponds to the cross-validation error rate in this instance. `size` is the number of terminal nodes of each tree considered. `k` is the value of the cost-complexity parameter used.

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






```
prune.carseats <- prune.misclass(tree.carseats, best=which.min(cv.carseats$size))
plot(prune.carseats)
text(prune.carseats, pretty=0)
```




```r
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
```

```r
mean(tree.pred==High.test)
```

```
## [1] 0.775
```

Fitting Regression Tree

```r
library(MASS)
data(Boston)
attach(Boston)
```

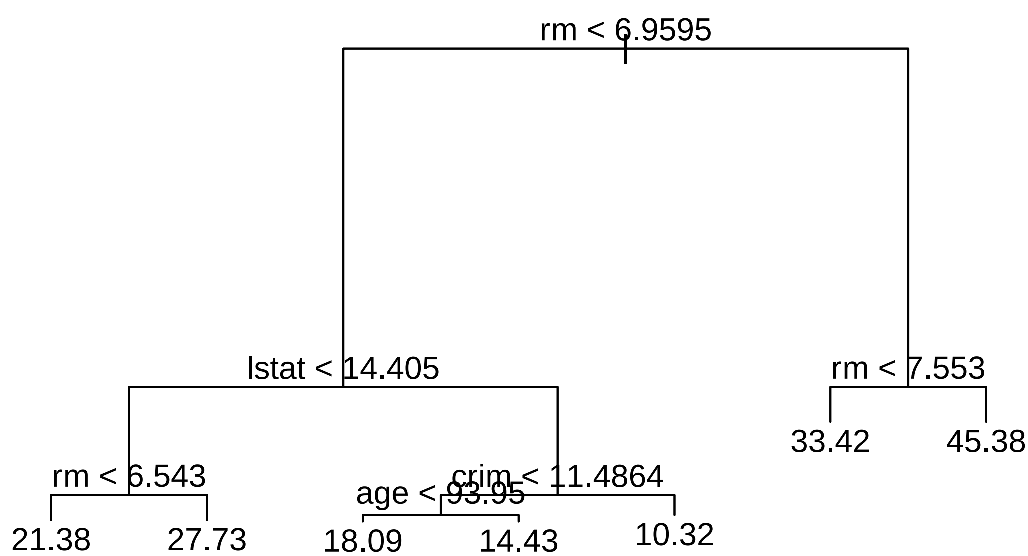
```
## The following object is masked from Wage:
##
##     age
```

```
## The following objects are masked from Boston (pos = 17):
##
##     age, black, chas, crim, dis, indus, lstat, medv, nox, ptratio, rad,
##     rm, tax, zn
```

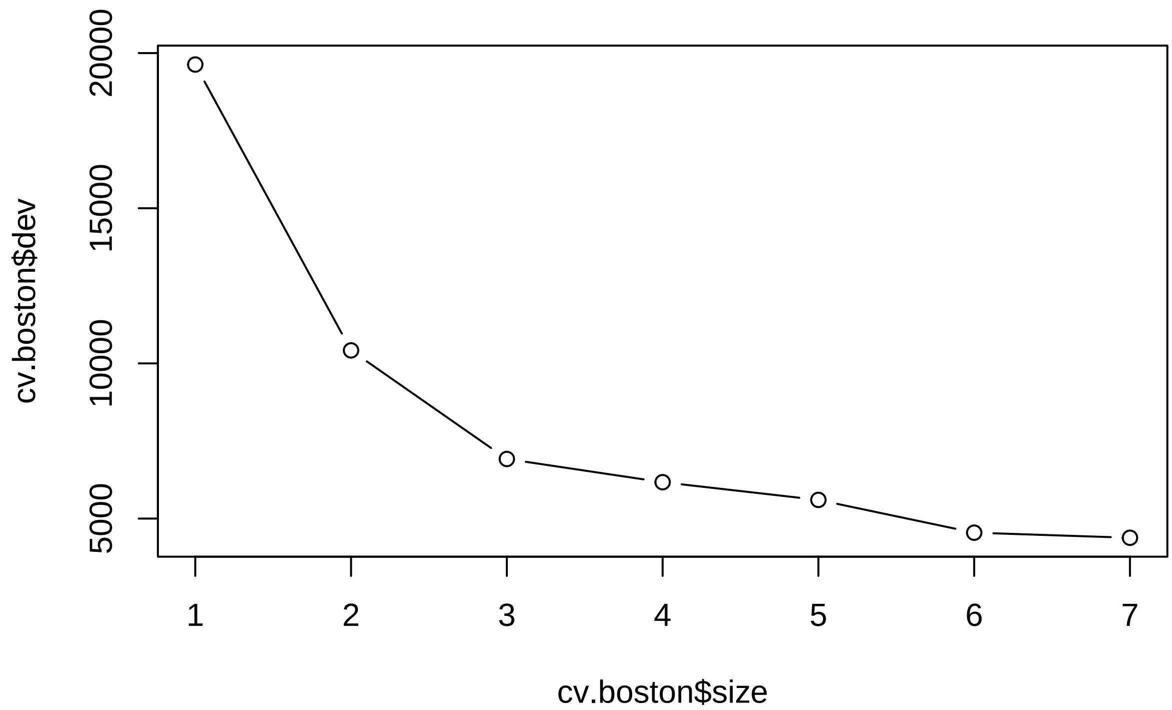
```
set.seed(1)
train <- sample(1:nrow(Boston), nrow(Boston)/2)
tree.boston <- tree(medv~., Boston, subset=train)
summary(tree.boston)
```

```
##
## 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
```

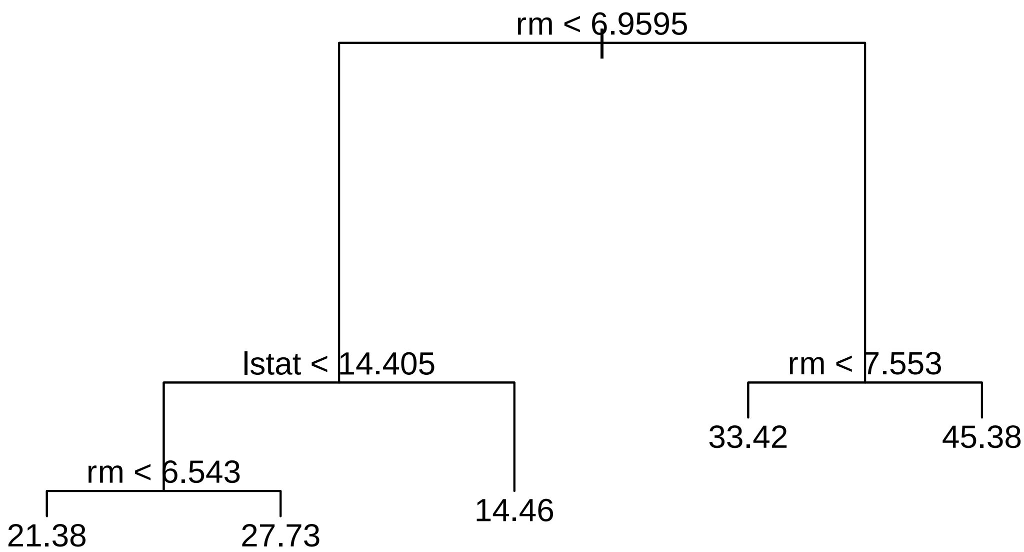
```
plot(tree.boston)
text(tree.boston, pretty=0)
```



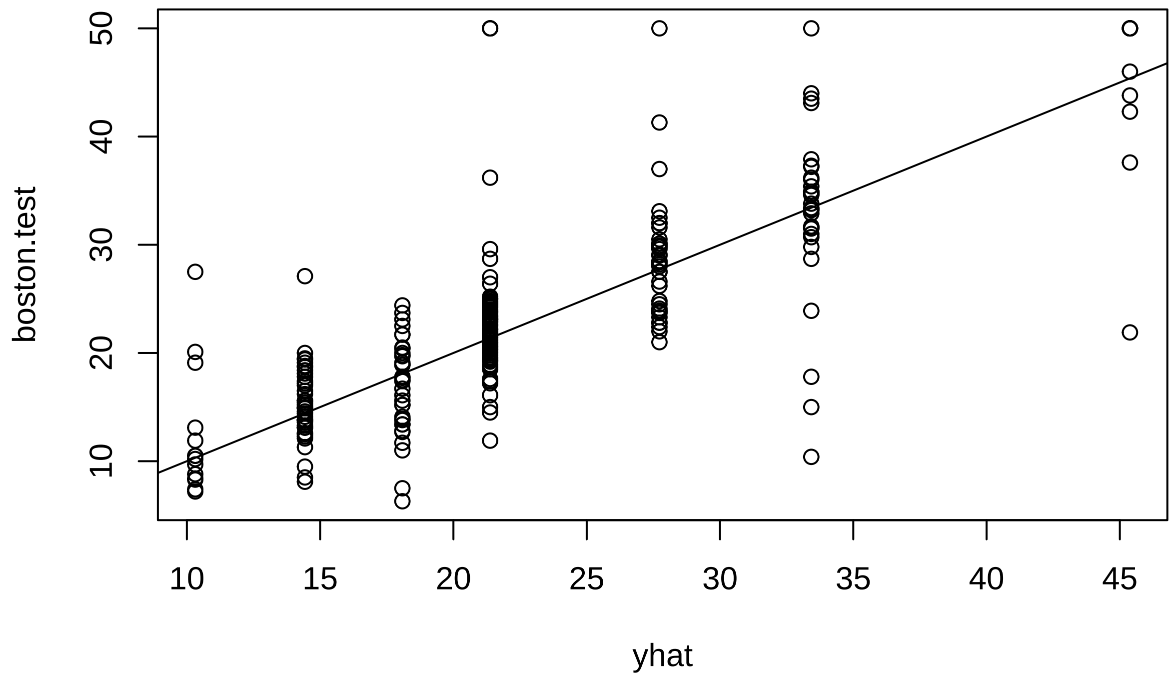
```r
cv.boston <- cv.tree(tree.boston)
plot(cv.boston$size, cv.boston$dev, type="b")
```




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




```
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
```

Bagging and Random Forests

Note that bagging is simply a special case of a random forest with $m = p$. Therefore, the `randomForest` function can be used to perform both random forests and bagging.

```
## Bagging
library(randomForest)
```

```
## randomForest 4.6-14
```

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

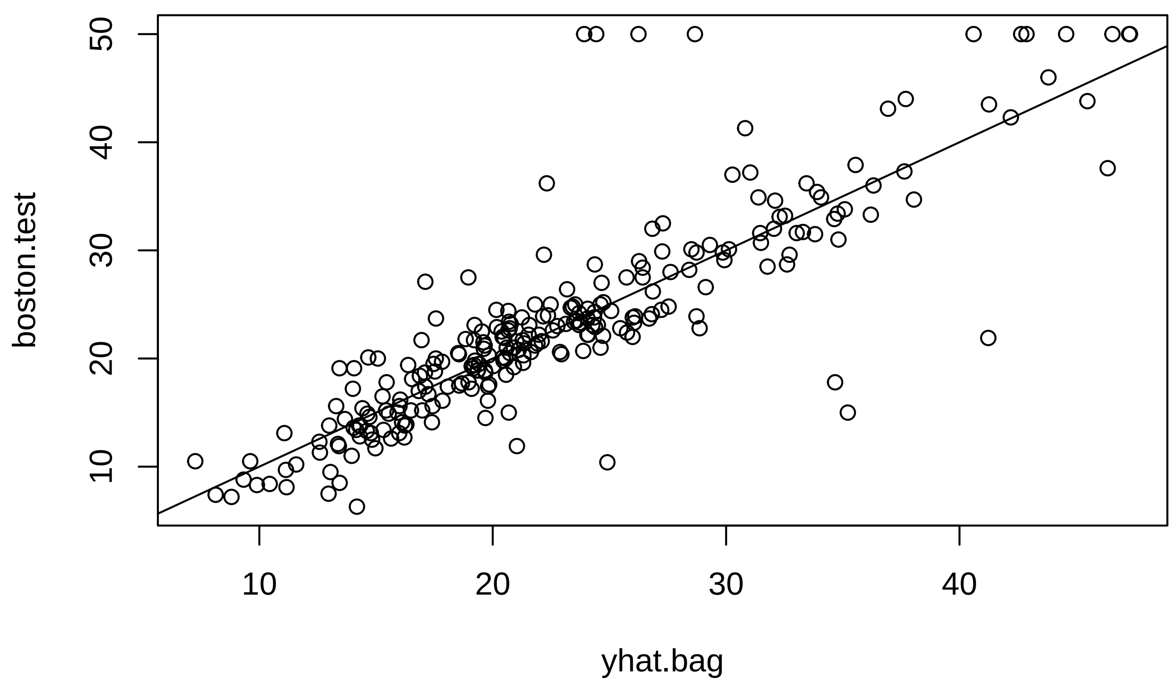
```
set.seed(1)
bag.boston <- randomForest(medv~., data=Boston, subset=train, mtry=13, importance=TRUE)
bag.boston
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = Boston, mtry = 13, importance = TRUE,      subset = train)
```

```
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 13
##
##           Mean of squared residuals: 11.39601
##                     % Var explained: 85.17
```

The argument `mtry=13` indicates that all 13 predictors should be considered for each split of the tree.

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




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

```
## [1] 23.59273
```

Growing a random forest proceeds in exactly the same way, except that we use a smaller value of the `mtry` argument. By default, `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.

```r
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)
```

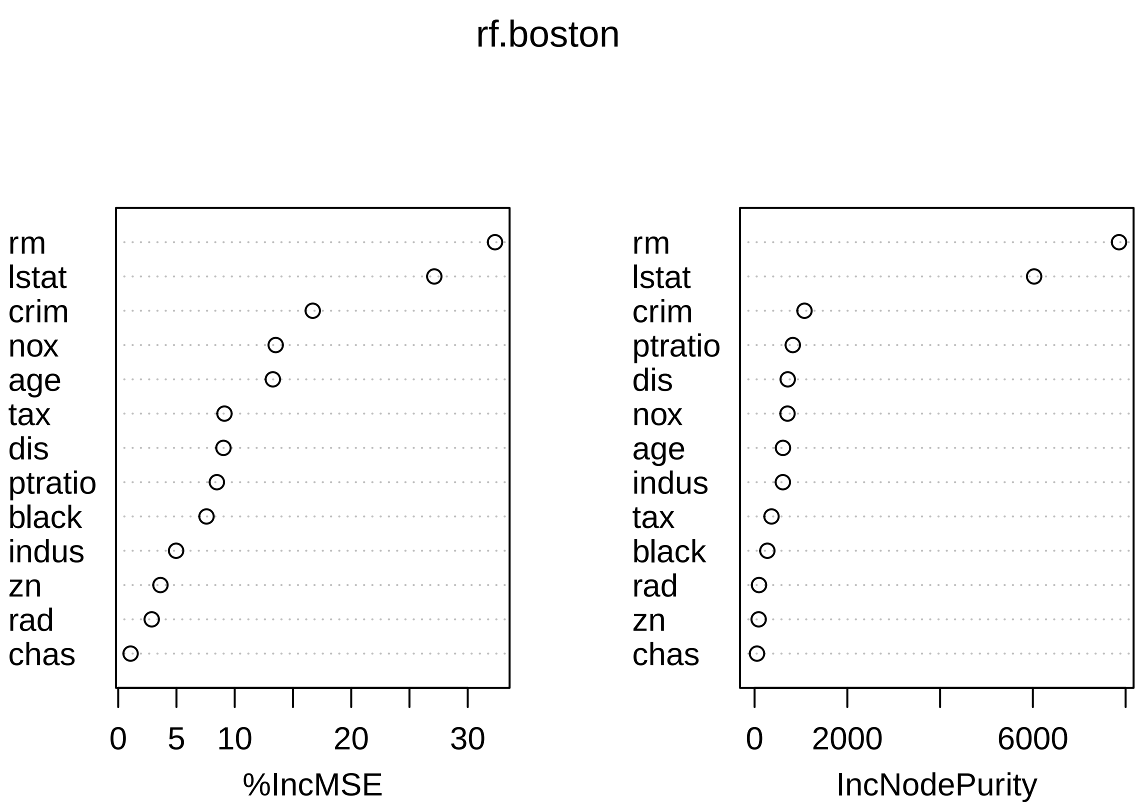
```
## [1] 19.62021
```

```
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
```

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 that results from splits over that variable, averaged over all trees. In the case of regression trees, the node impurity is measured by the train RSS, and for classification trees by the deviance.

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



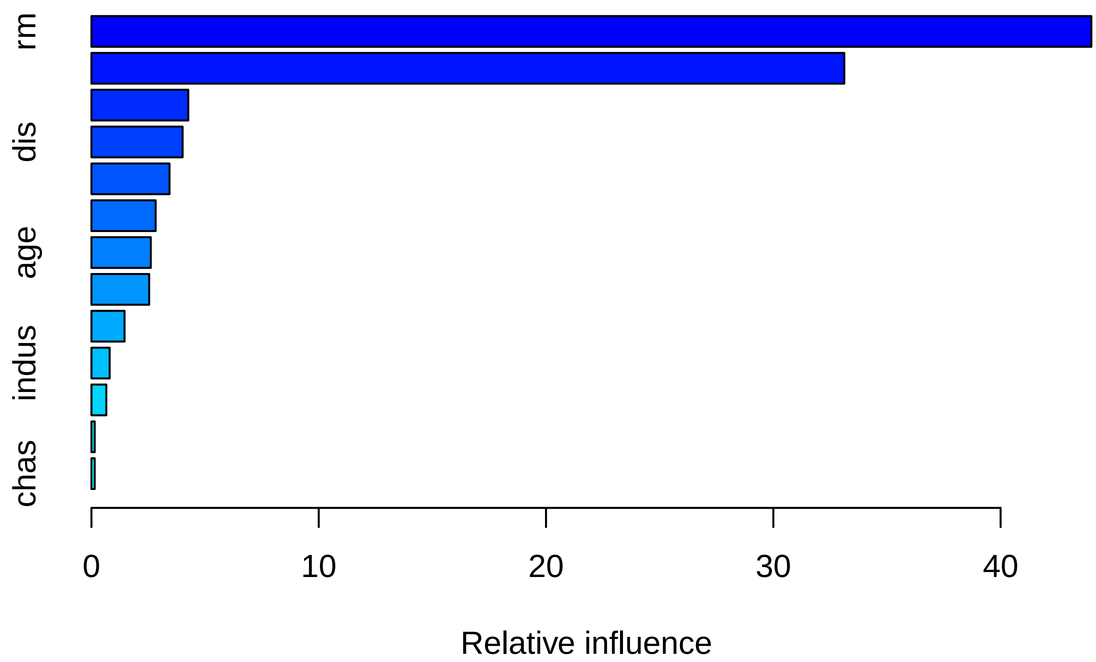
Boosting

```
library(gbm)
```

```
## Loaded gbm 2.1.8
```

```
set.seed(1)
boost.boston <- gbm(medv~., data=Boston[train,], distribution="gaussian",
                    n.trees=5000, interaction.depth=4)
```

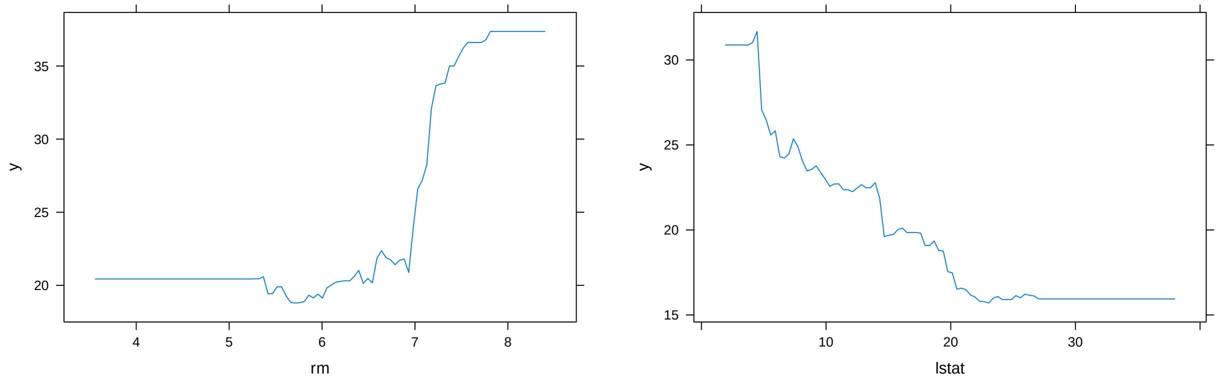
```
summary(boost.boston)
```




```
##             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
```

```
# par(mfrow=c(1, 2))
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] 18.84709
```

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

```
## [1] 18.33455
```