Non linear modeling

```
options(warn=-1)
```

In this notebook we'll use the Wage data from the ISLR library to explore the realm of non linear models.

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

```
head(Wage)
```

```
##        year age           maritl     race       education             region
## 231655 2006  18 1. Never Married 1. White    1. < HS Grad 2. Middle Atlantic
## 86582  2004  24 1. Never Married 1. White 4. College Grad 2. Middle Atlantic
## 161300 2003  45       2. Married 1. White 3. Some College 2. Middle Atlantic
## 155159 2003  43       2. Married 3. Asian 4. College Grad 2. Middle Atlantic
## 11443  2005  50      4. Divorced 1. White      2. HS Grad 2. Middle Atlantic
## 376662 2008  54       2. Married 1. White 4. College Grad 2. Middle Atlantic
##              jobclass          health health_ins  logwage      wage
## 231655  1. Industrial      1. <=Good      2. No 4.318063  75.04315
## 86582  2. Information 2. >=Very Good      2. No 4.255273  70.47602
## 161300  1. Industrial      1. <=Good     1. Yes 4.875061 130.98218
## 155159 2. Information 2. >=Very Good     1. Yes 5.041393 154.68529
## 11443  2. Information      1. <=Good     1. Yes 4.318063  75.04315
## 376662 2. Information 2. >=Very Good     1. Yes 4.845098 127.11574
```

Polynomial models

Is the wage a 4 order polynomial of the age of the person, considering Gaussian noise in it?

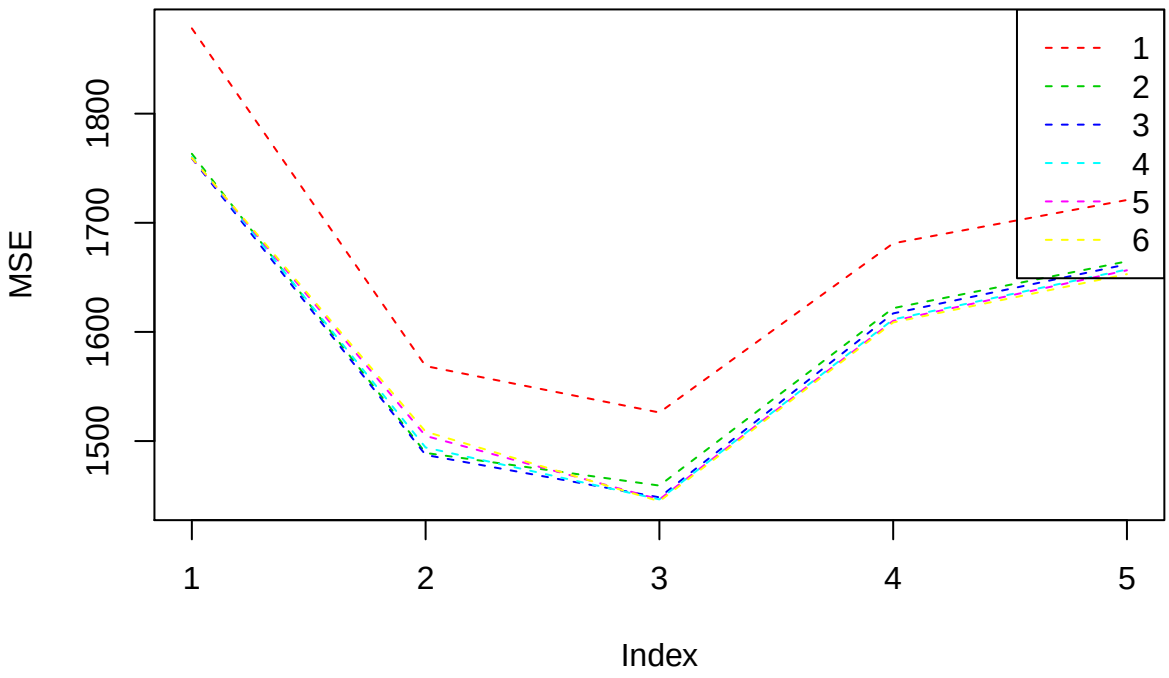
```
fit=lm(wage~poly(age ,4) ,data=Wage)
summary(fit)
```

```
##
## Call:
## lm(formula = wage ~ poly(age, 4), data = Wage)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -98.707 -24.626  -4.993  15.217 203.693
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    111.7036     0.7287 153.283  < 2e-16 ***
## poly(age, 4)1  447.0679    39.9148  11.201  < 2e-16 ***
## poly(age, 4)2 -478.3158    39.9148 -11.983  < 2e-16 ***
## poly(age, 4)3  125.5217    39.9148   3.145  0.00168 **
## poly(age, 4)4  -77.9112    39.9148  -1.952  0.05104 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 39.91 on 2995 degrees of freedom
## Multiple R-squared:  0.08626,    Adjusted R-squared:  0.08504
## F-statistic: 70.69 on 4 and 2995 DF,  p-value: < 2.2e-16
```

The answer to the above question seems to be partially positive, because the four order polynomial doesn't seem statistically significant to predict the response. Let's use cross-validation to evaluate the different models.

```
ncv <- 5
n <- dim(Wage)[1]
#shuffling
indices <- sample(1:n, size = n, replace=F)
#splitting
folds <- cut(indices, breaks = ncv, labels = F)
#poly order
od <- c(1,2,3,4,5,6)
res <- matrix(nrow=ncv, ncol=6)
for(order in od){
  for(i in 1:ncv){
    test <- indices[folds==i]
    fit<-lm(wage~poly(age ,order) ,data=Wage, subset=-test)
    preds<-predict(fit, newdata=Wage[test,])
    error<-sum((preds-Wage$wage[test])**2)/length(test)
    res[i,order]<-error
  }
}
plot(res[,1], type="l", lty="dashed", col=2, ylim =c(min(res),max(res)), ylab="MSE")
lines(res[,2], lty="dashed", col=3)
lines(res[,3], lty="dashed", col=4)
lines(res[,4], lty="dashed", col=5)
lines(res[,5], lty="dashed", col=6)
lines(res[,6], lty="dashed", col=7)
legend("topright",legend=c("1","2","3","4","5","6"), col=c(2,3,4,5,6,7), lty="dashed")
```




```
colMeans(res)
```

```
## [1] 1675.014 1599.598 1594.726 1593.914 1595.433 1594.930
```

```
which.min(colMeans(res))
```

```
## [1] 4
```

The fourth order degree seems to be the best one according to our cross validation! Let's see what the glm automatic cross validation would say.

```
library(boot)
res.glm <- numeric(6)
for(order in od){
  fit.glm <- glm(wage~poly(age ,order) ,data=Wage)
  res.glm[order] <- cv.glm(Wage, fit.glm, K=ncv)$delta[2]
}
res.glm
```

```
## [1] 1677.011 1599.135 1595.340 1594.591 1595.482 1593.127
```

```
which.min(res.glm)
```

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

The glm cross-validation and our cross validation seem to agree. What about the ANOVA test?

```
fit1 <- lm(wage~poly(age ,1) ,data=Wage)
fit2 <- lm(wage~poly(age ,2) ,data=Wage)
fit3 <- lm(wage~poly(age ,3) ,data=Wage)
```

```r
fit4 <- lm(wage~poly(age ,4) ,data=Wage)
fit5 <- lm(wage~poly(age ,5) ,data=Wage)
fit6 <- lm(wage~poly(age ,6) ,data=Wage)
anova(fit1,fit2,fit3,fit4,fit5,fit6)
```

```
## Analysis of Variance Table
##
## Model 1: wage ~ poly(age, 1)
## Model 2: wage ~ poly(age, 2)
## Model 3: wage ~ poly(age, 3)
## Model 4: wage ~ poly(age, 4)
## Model 5: wage ~ poly(age, 5)
## Model 6: wage ~ poly(age, 6)
##   Res.Df     RSS Df Sum of Sq        F    Pr(>F)
## 1   2998 5022216
## 2   2997 4793430  1    228786 143.6636 < 2.2e-16 ***
## 3   2996 4777674  1     15756   9.8936  0.001675 **
## 4   2995 4771604  1      6070   3.8117  0.050989 .
## 5   2994 4770322  1      1283   0.8054  0.369565
## 6   2993 4766389  1      3932   2.4692  0.116201
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note that the p-values obtained with the ANOVA are the same we obtain from the T-test in the biggest model.

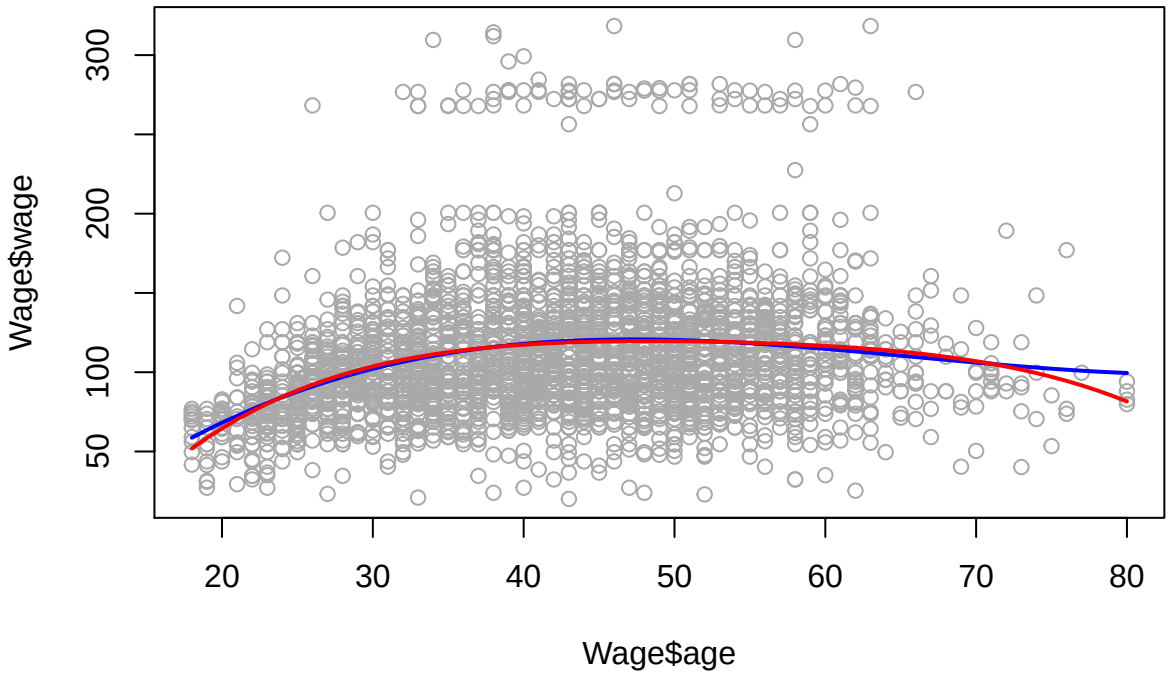
```r
summary(fit6)
```

```
##
## Call:
## lm(formula = wage ~ poly(age, 6), data = Wage)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -98.521 -24.536  -4.848  15.471 202.108
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    111.7036     0.7286 153.316  < 2e-16 ***
## poly(age, 6)1  447.0679    39.9063  11.203  < 2e-16 ***
## poly(age, 6)2 -478.3158    39.9063 -11.986  < 2e-16 ***
## poly(age, 6)3  125.5217    39.9063   3.145  0.00167 **
## poly(age, 6)4  -77.9112    39.9063  -1.952  0.05099 .
## poly(age, 6)5  -35.8129    39.9063  -0.897  0.36956
## poly(age, 6)6   62.7077    39.9063   1.571  0.11620
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 39.91 on 2993 degrees of freedom
## Multiple R-squared:  0.08726,    Adjusted R-squared:  0.08543
## F-statistic: 47.69 on 6 and 2993 DF,  p-value: < 2.2e-16
```

This happens because the poly function automatically builds orthogonal coordinates, hence the p-value associated with one predictor cannot be influenced by the presence/absence of other predictors. The Anova hence, like the T-test, doesn't see the fourth order term as statistically significant. Let's have a look at the

third and fourth order fits.

```
plot(Wage$age,Wage$wage, col="darkgray")
agelims <- range(Wage$age)
age.grid <- seq(from=agelims[1],to=agelims[2])
preds3 <- predict(fit3, newdata = data.frame(age=age.grid))
preds4 <- predict(fit4, newdata = data.frame(age=age.grid))
lines(age.grid, preds3, col="blue",lwd=2)
lines(age.grid, preds4, col="red",lwd=2)
```



Step functions

We now want to fit a step function to predict wage using age, and perform cross-validation to choose the optimal number of cuts.

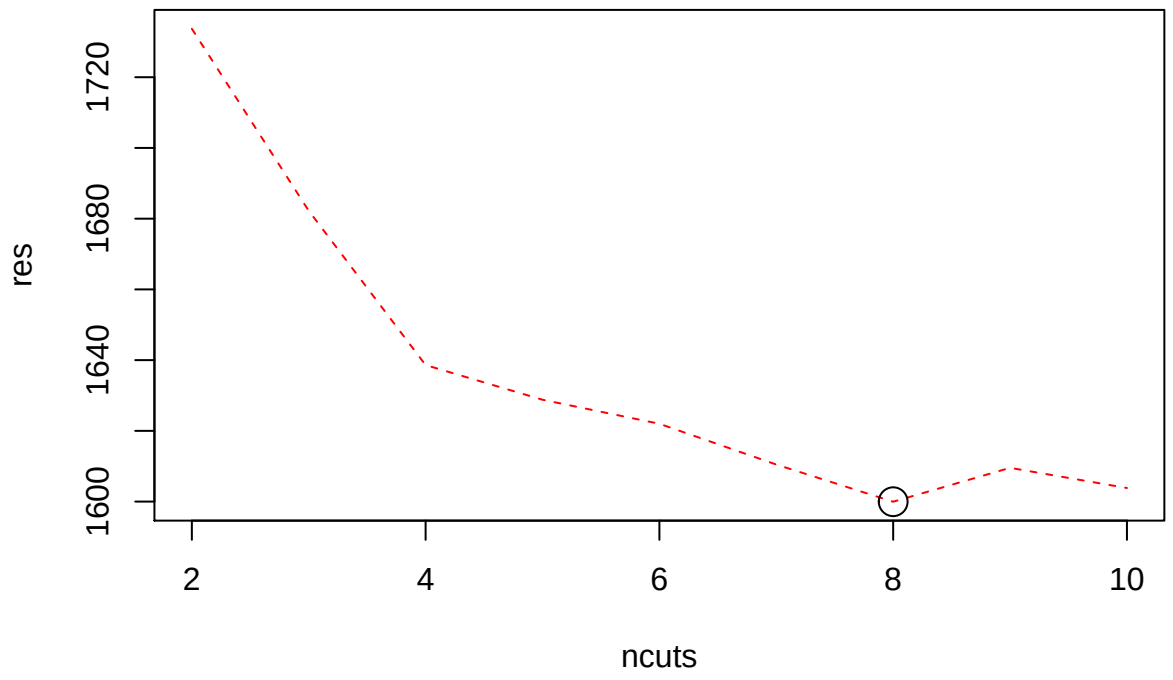
```
# The number of cuts we want to experiment with
ncuts <- c(2,3,4,5,6,7,8,9,10)
res <- numeric(length(ncuts))
for(j in 1:length(ncuts)){
  nc <- ncuts[j]
  # saving the new factor variable in the dataframe
  Wage$age.cut <- cut(age,nc)
  # fit step function to the train data
  fit.step <- glm(wage~age.cut, data=Wage)
  # evaluate the fit on the test data
  cv.res <- cv.glm(Wage, fit.step, K=ncv)$delta[2]
  res[j] <- cv.res
```

```
}

res
```

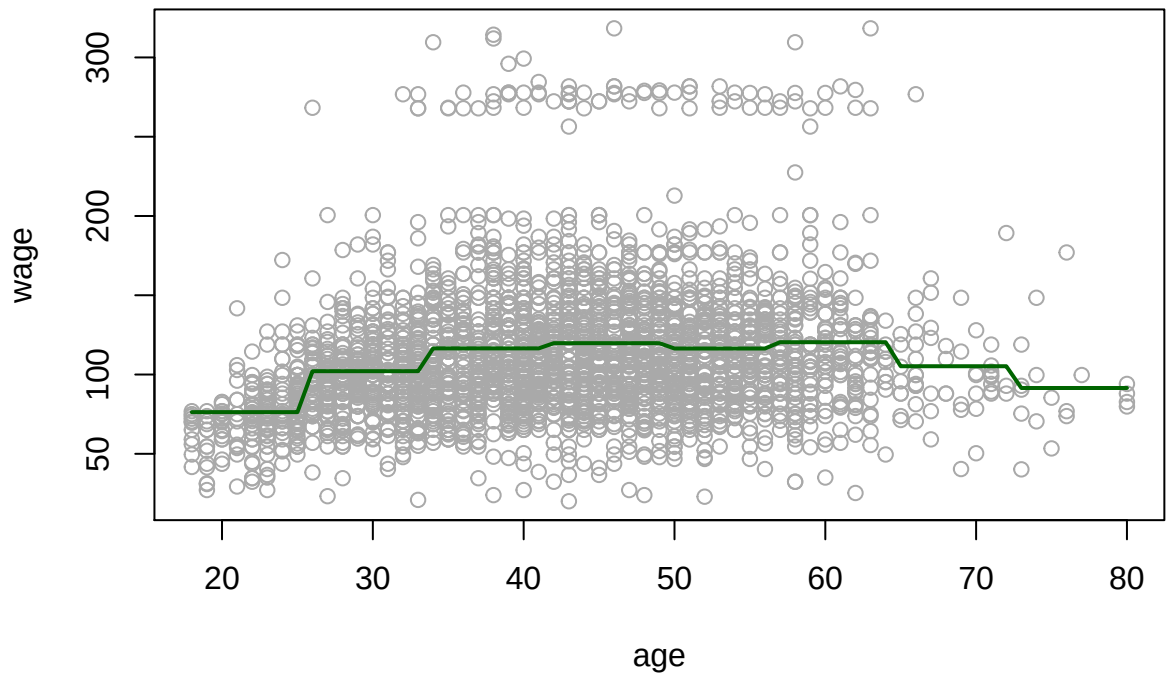
```
## [1] 1733.682 1682.234 1638.599 1628.793 1622.000 1610.472 1599.964 1609.536
## [9] 1603.835
```

```
plot(ncuts, res, type="l", lty="dashed", col="red")
points(ncuts[which.min((res))], min(res), cex=2)
```



Hence the minimum is obtained with 8 cuts. Let's have a look at the fitted function.

```
fit.8.steps <- glm(wage~cut(age,8), data=Wage)
preds.steps <- predict(fit.8.steps, newdata=data.frame(age=age.grid))
plot(age, wage, col="darkgray")
lines(age.grid, preds.steps, lwd=2, col="darkgreen")
```



Let's now explore other the non-linear relationships between wage and other variables.

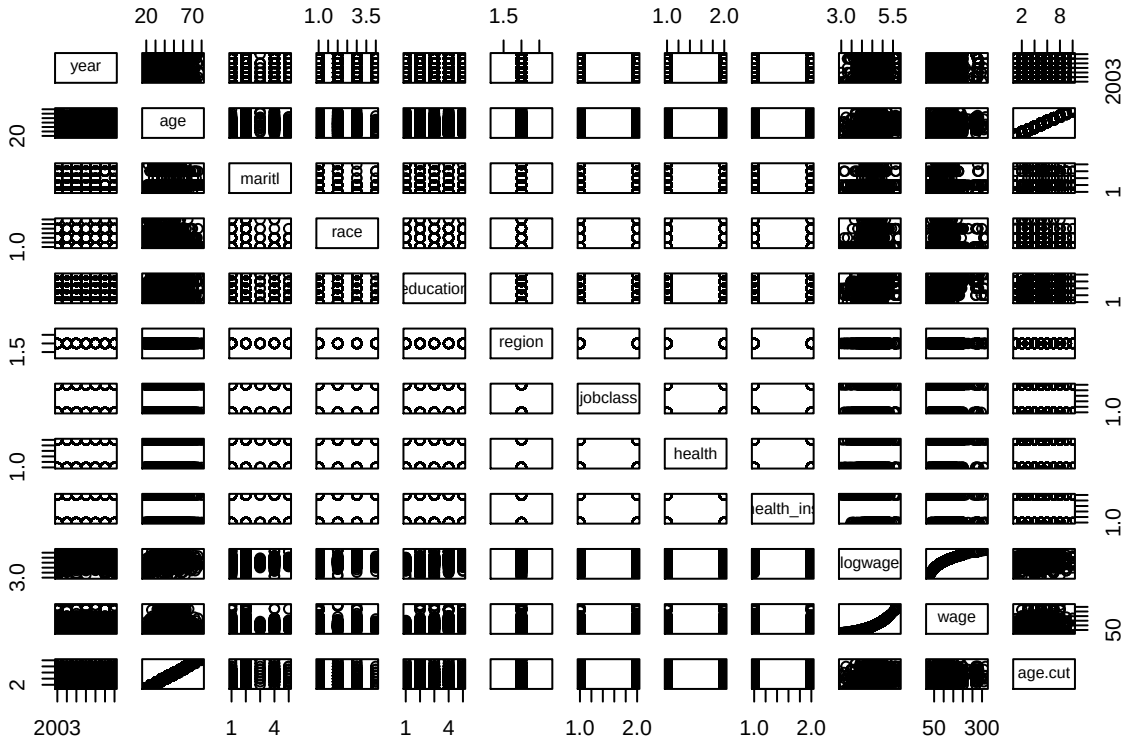
```
p <- dim(Wage)[2]
p
```

```
## [1] 12
```

```
head(Wage)
```

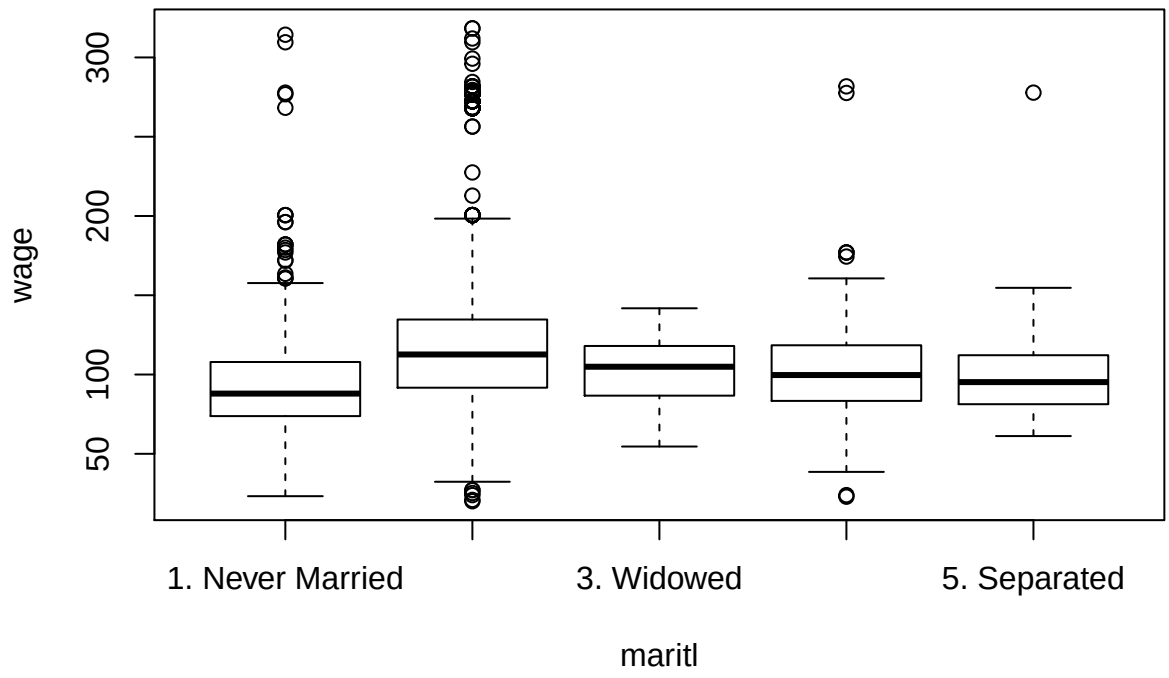
```
##        year age           maritl     race       education             region
## 231655 2006  18 1. Never Married 1. White    1. < HS Grad 2. Middle Atlantic
## 86582  2004  24 1. Never Married 1. White 4. College Grad 2. Middle Atlantic
## 161300 2003  45       2. Married 1. White 3. Some College 2. Middle Atlantic
## 155159 2003  43       2. Married 3. Asian 4. College Grad 2. Middle Atlantic
## 11443  2005  50      4. Divorced 1. White      2. HS Grad 2. Middle Atlantic
## 376662 2008  54       2. Married 1. White 4. College Grad 2. Middle Atlantic
##              jobclass          health health_ins  logwage      wage     age.cut
## 231655  1. Industrial      1. <=Good      2. No 4.318063  75.04315 (17.9,24.2]
## 86582  2. Information 2. >=Very Good      2. No 4.255273  70.47602 (17.9,24.2]
## 161300  1. Industrial      1. <=Good     1. Yes 4.875061 130.98218   (42.8,49]
## 155159 2. Information 2. >=Very Good     1. Yes 5.041393 154.68529   (42.8,49]
## 11443  2. Information      1. <=Good     1. Yes 4.318063  75.04315   (49,55.2]
## 376662 2. Information 2. >=Very Good     1. Yes 4.845098 127.11574   (49,55.2]
```

```
# 11 predictors
pairs(Wage)
```



We'll now build a generalized additive model to predict the Wage based on the age, the marital status and the eductation. We'll first do some exploratory analysis to evaluate the relationships between wage and marital status and education.

```
plot(maritl,wage, xlab="maritl", ylab="wage")
```



Since the main difference seems to be between never married, married and the the other 3 categories together let's try models where we use this categorical variable with some of the levels.

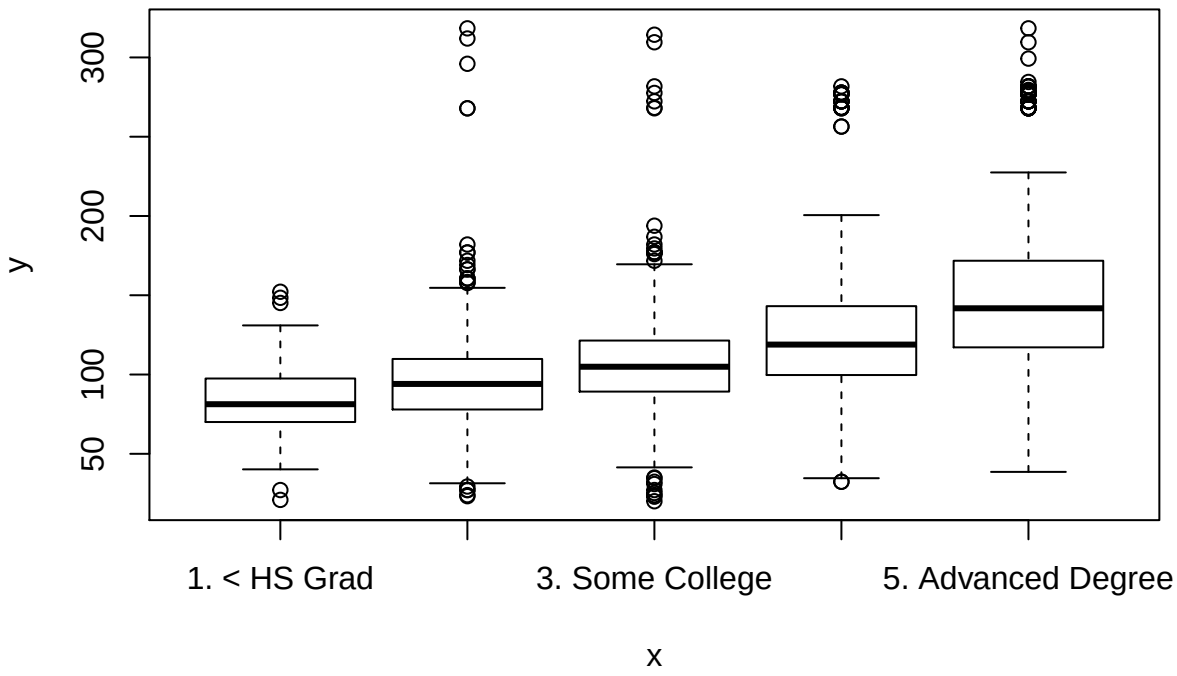
```
lvl1 <- maritl == levels(as.factor(maritl))[1]
lvl2 <- maritl == levels(as.factor(maritl))[2]
fit1 <- lm(wage~lvl2)
fit2 <- lm(wage~lvl2+lvl1)
fit3 <- lm(wage~maritl)
anova(fit1,fit2,fit3)
```

```
## Analysis of Variance Table
##
## Model 1: wage ~ lvl2
## Model 2: wage ~ lvl2 + lvl1
## Model 3: wage ~ maritl
##   Res.Df     RSS Df Sum of Sq       F    Pr(>F)
## 1   2998 4877943
## 2   2997 4859287  1   18655.6 11.4991 0.0007053 ***
## 3   2995 4858941  2     345.8  0.1066 0.8989166
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The anova test confirms our intuitions: there's no statistically significant difference between Widowed Divorced and Separated, however there is a difference between these three together and Married or Never Married.

Let's now look at education:

```
plot(education, wage)
```



If we do the same we've done before for marital status here to education we'll probably obtain opposite results: all the categories are statistically significantly different. But let's put it into numbers.

```
edu1 <- education == levels(as.factor(education))[1]
edu3 <- education == levels(as.factor(education))[3]
edu5 <- education == levels(as.factor(education))[5]
fit1 <- lm(wage~edu3)
fit2 <- lm(wage~edu3+edu1)
fit3 <- lm(wage~edu3+edu1+edu5)
fit4 <- lm(wage~education)
anova(fit1,fit2,fit3,fit4)
```

```
## Analysis of Variance Table
##
## Model 1: wage ~ edu3
## Model 2: wage ~ edu3 + edu1
## Model 3: wage ~ edu3 + edu1 + edu5
## Model 4: wage ~ education
##   Res.Df     RSS Df Sum of Sq      F    Pr(>F)
## 1   2998 5209152
## 2   2997 4960140  1    249012 186.65 < 2.2e-16 ***
## 3   2996 4325281  1    634860 475.86 < 2.2e-16 ***
## 4   2995 3995721  1    329559 247.02 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Our intution was right. Let's now turn to the variable age again, but this time let's use splines.

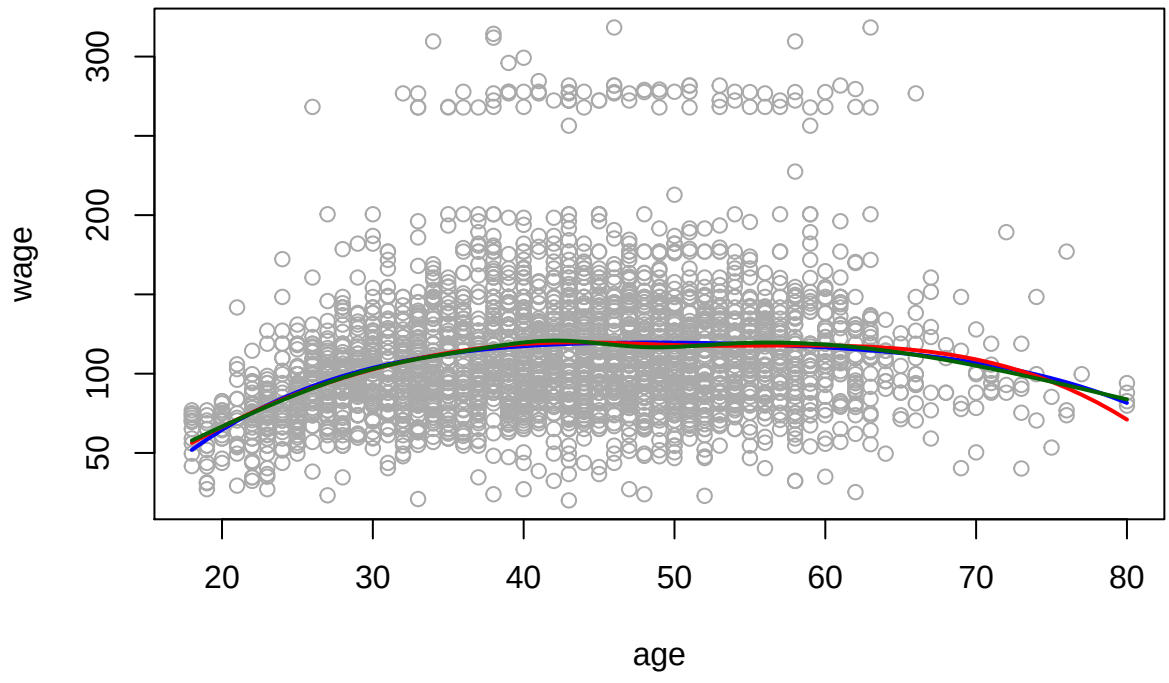
```
library(gam)
```

```
## Loading required package: splines
```

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

```
## Loaded gam 1.16.1
```

```
poly.fit <- glm(wage~poly(age,4), data=Wage)
spline.fit <- glm(wage~bs(age, df = 6))
nat.spline.fit <- glm(wage~ns(age, df = 6))
plot(age,wage, col="darkgray")
preds.poly <- predict(poly.fit, newdata=data.frame(age=age.grid))
preds.spline <- predict(spline.fit, newdata=data.frame(age=age.grid))
preds.nat.spline <- predict(nat.spline.fit, newdata=data.frame(age=age.grid))
lines(age.grid, preds.poly, lwd=2, col="blue")
lines(age.grid, preds.spline, lwd=2, col="red")
lines(age.grid, preds.nat.spline, lwd=2, col="darkgreen")
```



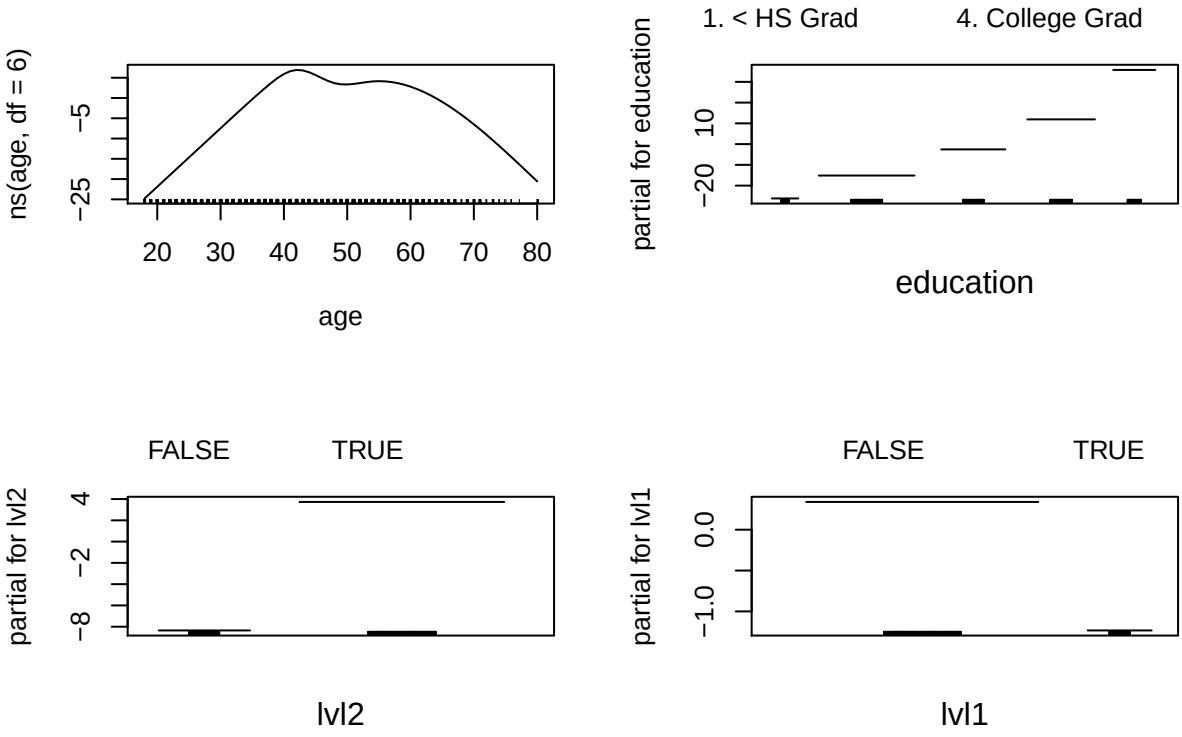
Note that the natural spline and the polynomial fit are almost identical. Due to its robustness at the boundary we choose the natural spline, and proceed to fit a generalized additive model.

```
gam.fit <- gam(wage~ns(age, df = 6)+education+lvl2+lvl1)
summary(gam.fit)
```

```
##
## Call: gam(formula = wage ~ ns(age, df = 6) + education + lvl2 + lvl1)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
```

```
## -116.075  -19.287   -2.713   14.217  212.816
##
## (Dispersion Parameter for gaussian family taken to be 1212.358)
##
##     Null Deviance: 5222086 on 2999 degrees of freedom
## Residual Deviance: 3621314 on 2987 degrees of freedom
## AIC: 29829.57
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##                   Df  Sum Sq Mean Sq  F value Pr(>F)
## ns(age, df = 6)    6  458327   76388  63.0076 <2e-16 ***
## education          4 1052120  263030 216.9572 <2e-16 ***
## lvl2               1   89922   89922  74.1708 <2e-16 ***
## lvl1               1     403     403   0.3324 0.5643
## Residuals       2987 3621314    1212
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
par(mfrow=c(2,2))
plot.Gam(gam.fit)
```



The curse of dimensionality

We're now going to practically face what is known as *the curse of dimensionality* in the context of three different linear models: Knn, multiple linear regression and GAMs. We will consider the specific case when the underlying signal is sparse (only one predictor has an effect on the response) and the signal is non-linear.

```
library(gam)
library(FNN)
# Generating artificial dataset
set.seed(1)
# Train set
xtrain <- matrix(rnorm(20*100),ncol=20, nrow = 100)
# Note: dimensionality of predictors = 20
ytrain <- sin(2*xtrain[,1]) + 0.3*rnorm(100)
dtrain <- data.frame(xtrain,y = ytrain)
# Test set
xtest <- matrix(rnorm(20*100),ncol=20, nrow = 100)
ytest <- sin(2*xtest[,1]) + 0.3*rnorm(100)
dtest <- data.frame(xtest,y = ytest)
```

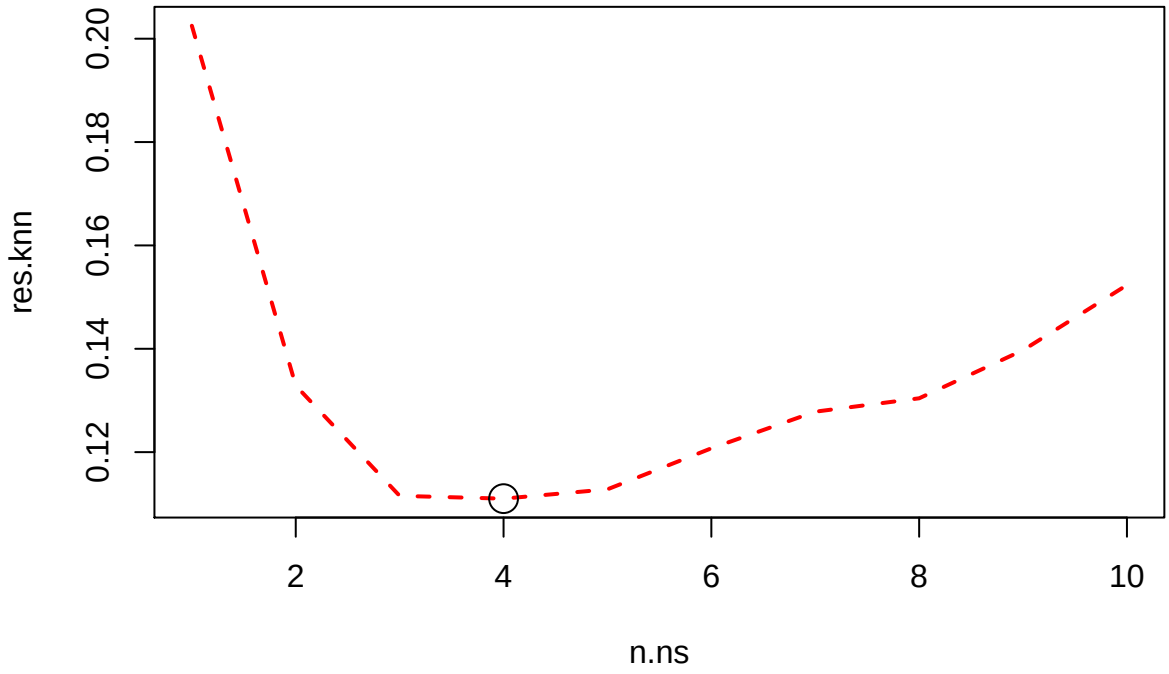
```
head(dtrain)
```

```
##           X1          X2         X3         X4         X5          X6
## 1 -0.6264538 -0.62036668  0.4094018  0.8936737  1.0744410  0.07730312
## 2  0.1836433  0.04211587  1.6888733 -1.0472981  1.8956548 -0.29686864
## 3 -0.8356286 -0.91092165  1.5865884  1.9713374 -0.6029973 -1.18324224
## 4  1.5952808  0.15802877 -0.3309078 -0.3836321 -0.3908678  0.01129269
## 5  0.3295078 -0.65458464 -2.2852355  1.6541453 -0.4162220  0.99160104
## 6 -0.8204684  1.76728727  2.4976616  1.5122127 -0.3756574  1.59396745
##           X7          X8          X9        X10         X11        X12
## 1 -0.3410670 -0.70756823 -1.08690882 -1.5414026  1.13496509  0.2418959
## 2  1.5024245  1.97157201 -1.82608301  0.1943211  1.11193185 -1.1327594
## 3  0.5283077 -0.08999868  0.99528181  0.2644225 -0.87077763  1.4899074
## 4  0.5421914 -0.01401725 -0.01186178 -1.1187352  0.21073159 -0.2482471
## 5 -0.1366734 -1.12345694 -0.59962839  0.6509530  0.06939565  0.1835837
## 6 -1.1367339 -1.34413012 -0.17794799 -1.0329002 -1.66264885  0.4048710
##          X13        X14        X15        X16         X17        X18        X19
## 1 -1.5570357  0.3412484  1.5468813  0.8500435  0.34419403  1.6212029  0.7140855
## 2  1.9231637  1.3161672  0.1789210 -0.9253130  0.01271984 -0.3291028  0.5813846
## 3 -1.8568296 -0.9597765 -0.2825466  0.8935812 -0.87345013 -2.3264095 -0.1467239
## 4 -2.1061184 -1.2055752 -0.7672988 -0.9410097  0.34280028  2.1929980  1.5069818
## 5  0.6976485  1.5675731 -0.5764042  0.5389521 -0.17738775 -1.0824800 -0.2795326
## 6  0.9074444  0.2252858 -0.9148558 -0.1819744  0.92143325 -0.5063610  2.0277387
##           X20          y
## 1 -0.57099429 -1.2157423
## 2  0.28653902 -0.2175921
## 3  1.14761986 -0.5090478
## 4  0.13955870  0.1068316
## 5  0.08892661  0.5955839
## 6 -2.63015932 -0.7886159
```

We'll now fit on the training set each of the three models (KNN, multiple linear regression and GAM with splines) with only the first predictor, and then evaluate their test mean squared error.

```
# KNN first
# 10 different number of neighbors
```

```
n.ns <- 1:10
res.knn <- numeric(length(n.ns))
for(n.n in n.ns){
  # Fit KNN model with n.n number of neighbors
  fit <- knn.reg(train=as.matrix(xtrain[,1]),
              test=as.matrix(xtest[,1]),
              y=ytrain, k = n.n)
  error <- sum((fit$pred-ytest)**2)/length(ytest)
  res.knn[n.n] <- error
}
plot(n.ns, res.knn, type="l", lty="dashed", lwd=2, col="red")
min.nn <- which.min(res.knn)
points(min.nn, res.knn[min.nn], cex=2)
```




```
#saving best KNN model
fit.knn <- knn.reg(train=as.matrix(xtrain[,1]),
              test=as.matrix(xtest[,1]),
              y=ytrain, k = 4)
```

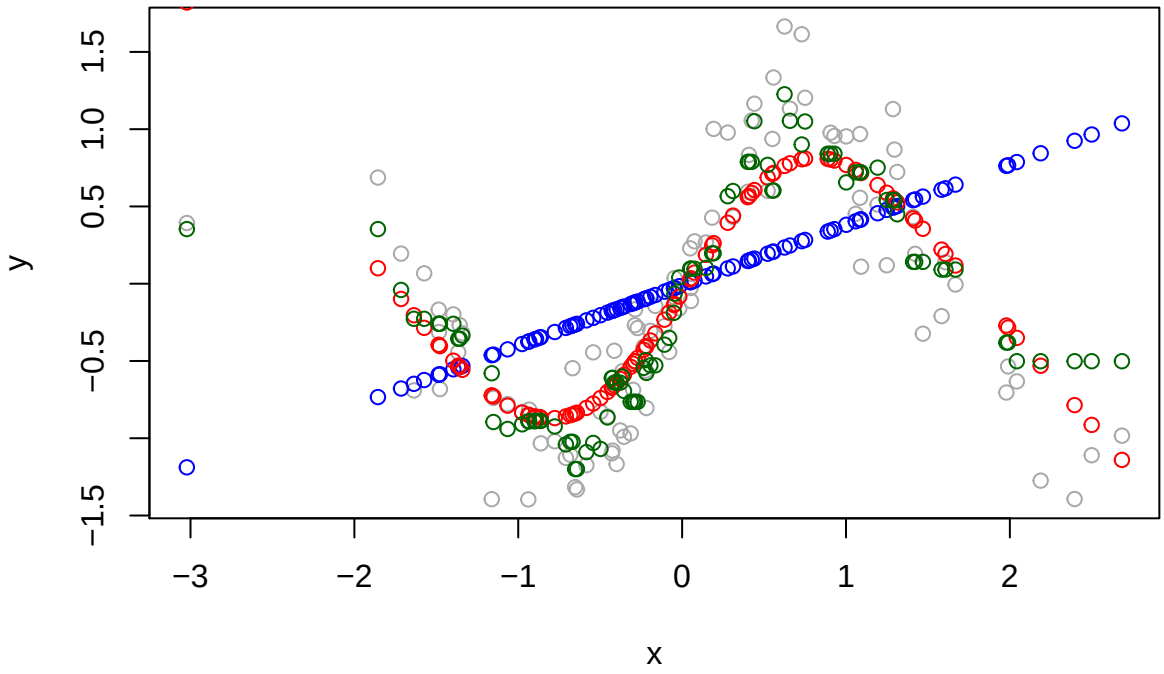
```
# Now multiple linear regression
lm.fit <- lm(y~X1, data=dtrain)
preds.lm <- predict(lm.fit, dtest)
error <- mean((preds.lm-dtest$y)^2)
error
```

```
## [1] 0.6329264
```

```
# And finally let's use splines
GAM.fit <- gam(y~s(X1, df = 4), data=dtrain)
preds.GAM <- predict(GAM.fit,dtest)
error <- mean((preds.GAM-dtest$y)^2)
error
```

```
## [1] 0.1385982
```

```
#Plotting
plot(xtest[,1], ytest, col="darkgray", xlab="x",ylab="y")
points(xtest[,1], preds.lm, col="blue")
points(xtest[,1], preds.GAM, col="red")
points(xtest[,1], fit.knn$pred, col="darkgreen")
```



As we may have expected, the two most flexible models give the best results, while linear regression struggles to fit the data. Now what happens if we increase the dimensionality?

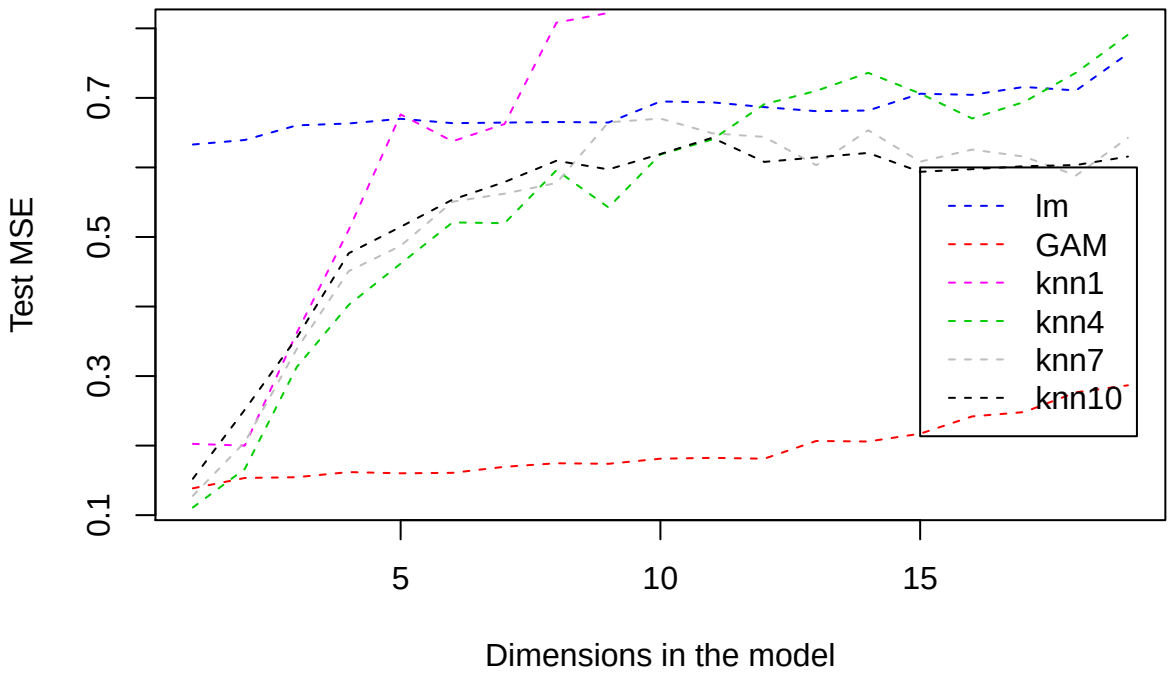
```
# For each additional predictor we'll record the test MSE
res.dim <- matrix(nrow=19, ncol=12)
predictors <- names(dtrain)[1:19]
for(d in 1:19){
  # FIRST KNNs
  for(n.n in n.ns){
  # Fit KNN model with n.n number of neighbors
  fit <- knn.reg(train=as.matrix(xtrain[,1:d]),
              test=as.matrix(xtest[,1:d]),
              y=ytrain, k = n.n)
  error <- mean((fit$pred-ytest)**2)
```

```
  res.dim[d, n.n] <- error
  }
  # multiple linear regression
  lm.formula <- as.formula(paste("y~",paste(predictors[1:d],collapse="+"), sep = ""))
  lm.fit <- lm(lm.formula, data=dtrain)
  preds.lm <- predict(lm.fit, dtest)
  res.dim[d,11] <- mean((preds.lm-dtest$y)^2)
  # GAMS
  gams.formula <- as.formula(paste("y~s(",paste(predictors[1:d],
                                                collapse=",4)+s(",",4)",sep = ""))
  GAM.fit <- gam(gams.formula, data=dtrain)
  preds.GAM <- predict(GAM.fit,dtest)
  res.dim[d,12] <- mean((preds.GAM-dtest$y)^2)
}
```

```
res.dim <- data.frame(res.dim)
names(res.dim)<- c("knn1","knn2","knn3","knn4","knn5","knn6","knn7","knn8","knn9","knn10","lm","GAM")
head(res.dim)
```

```
##        knn1      knn2      knn3      knn4      knn5      knn6      knn7
## 1 0.2025036 0.1328669 0.1115425 0.1110039 0.1128088 0.1207373 0.1278271
## 2 0.1999919 0.1723791 0.1670061 0.1664203 0.1744254 0.1913617 0.2049146
## 3 0.3620731 0.2618467 0.2724886 0.3128575 0.3117892 0.3152308 0.3388934
## 4 0.5101438 0.3968565 0.4097525 0.4022746 0.3989410 0.4130885 0.4508353
## 5 0.6762065 0.5112015 0.4660836 0.4616575 0.4730929 0.4803426 0.4872387
## 6 0.6375652 0.5267985 0.4894428 0.5209776 0.5208555 0.5306474 0.5506706
##        knn8      knn9     knn10        lm       GAM
## 1 0.1304200 0.1397241 0.1523280 0.6329264 0.1385982
## 2 0.2246237 0.2428624 0.2515996 0.6394764 0.1534681
## 3 0.3386655 0.3361963 0.3550551 0.6605180 0.1545241
## 4 0.4531481 0.4605854 0.4766057 0.6631865 0.1619465
## 5 0.4957665 0.5123646 0.5145941 0.6699414 0.1601889
## 6 0.5504020 0.5455623 0.5537569 0.6636671 0.1608692
```

```
plot(1:19, res.dim$lm, type="l", col="blue", lty="dashed", ylim=c(0.12,0.8), ylab="Test MSE", xlab="Dim
lines(1:19, res.dim$GAM, type="l", col="red", lty="dashed")
lines(1:19, res.dim$knn1, type="l", col=6, lty="dashed")
lines(1:19, res.dim$knn4, type="l", col=11, lty="dashed")
lines(1:19, res.dim$knn7, type="l", col=8, lty="dashed")
lines(1:19, res.dim$knn10, type="l", col=9, lty="dashed")
legend(15,0.6, legend=c("lm","GAM","knn1","knn4","knn7","knn10"), lty="dashed", col=c("blue", "red",6,1
```



It is clear from the above results that the KNN approach suffers from the curse of dimensionality the most, while both multiple linear regression and GAMs models slowly increase with the dimension.

The backfitting algorithm

We'll now focus on a topic specific to GAMs, which is that of model fitting. GAMs do not always present a closed form solution (for instance in presence of splines or loess curves). To fit these models the *backfitting algorithm* can be used. We'll now implement it for a multiple regression model, which can be seen as a special cas of a GAM.

```
set.seed(1)
# Generating 2 artificial datasets
# One with completely indipendent variables
# One with dependent variables
n <- 100
x1 <- rnorm(n)
x2 <- rnorm(n)
x3 <- rnorm(n)
x4 <- rnorm(n)
y <- 0.7 + x1 + 2*x2 + 0.5*x3 -3*x4 + rnorm(n)
# second dataset
x1_cor <- rnorm(n)
x2_cor <- rnorm(n)
x3_cor <- rnorm(n)
x4_cor <- x1_cor + 0.4*rnorm(n)
y_cor <- x1_cor + 2*x2_cor + 0.5*x3_cor -3*x4_cor + rnorm(n)
```

```
niter <- 10
nvar <- 4
# one step (one variable modified)
backfit.onestep<-function(xj, y,j, gs){
  other.gs <- gs[,setdiff(1:dim(gs)[2],j)]
  new.y <- y - mean(y) - rowSums(other.gs)
  fitj <- lm(new.y~xj)
  return(fitj)
}
do.backfit <- function(x,y,niter, eps){
  # Initialization
  fits <- matrix(nrow=nvar, ncol=2)
  gs <- matrix(rep(0, n*nvar), nrow=n, ncol=nvar)
  # Iterations
  for(i in 1:niter){
    old.gs <- gs
    for(j in 1:dim(x)[2]){
      new.fitj <- backfit.onestep(x[,j],y,j, gs)
      fits[j,] <- new.fitj$coefficients
      gs[,j]<-new.fitj$fitted.values
    }
    # check for convergence
    conv <- all(abs(old.gs - gs) < eps)
    if(conv){
      break
      }
  }
  if(conv){print(paste("Converged after ",paste(i,"iterations")))}
  else{print(paste("Not converged: update = ",max(abs(old.gs - gs))))}
  return(fits)
}
```

```
x <- cbind(x1,x2,x3,x4)
x.cor <- cbind(x1_cor, x2_cor, x3_cor, x4_cor)
x.fits <- do.backfit(x,y,niter=10,eps=0.05)
```

```
## [1] "Converged after  3 iterations"
```

```
x.cor.fits <- do.backfit(x.cor,y_cor,niter=10,eps=0.05)
```

```
## [1] "Not converged: update =  0.279119488585477"
```

Let's give the correlation data more iterations:

```
x.cor.fits <- do.backfit(x.cor,y_cor,niter=100,eps=0.05)
```

```
## [1] "Converged after  21 iterations"
```

Let's now compare the estimate we get from backfitting with those from standard multiple linear regression: we expect to see no difference for the indipendent dimensions data, while we expect to see differences with the second dataset.

```
lm.fit.x <- lm(y~x)
lm.fit.x.cor <- lm(y_cor~x.cor)
```

```
beta0.x <- sum(x.fits[,1])+mean(y)
beta0.x
```

```
## [1] 0.6429992
```

```
x.fits[,2]
```

```
## [1]  1.1934712  2.1693271  0.5336871 -2.9573220
```

```
summary(lm.fit.x)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.5593 -0.7365 -0.0915  0.6819  3.7101
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.6430     0.1180   5.449 3.97e-07 ***
## xx1           1.1933     0.1309   9.113 1.30e-14 ***
## xx2           2.1696     0.1229  17.651  < 2e-16 ***
## xx3           0.5337     0.1144   4.666 1.00e-05 ***
## xx4          -2.9573     0.1195 -24.750  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.168 on 95 degrees of freedom
## Multiple R-squared:  0.9194, Adjusted R-squared:  0.916
## F-statistic:   271 on 4 and 95 DF,  p-value: < 2.2e-16
```

We were right about the indipendent dimensions dataset. What about the second one?

```
beta0.xcor <- sum(x.cor.fits[,1])+mean(y)
beta0.xcor
```

```
## [1] 0.8510059
```

```
x.cor.fits[,2]
```

```
## [1]  0.2760864  1.9113279  0.5758315 -2.3900377
```

```
summary(lm.fit.x.cor)
```

```
##
## Call:
## lm(formula = y_cor ~ x.cor)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.73871 -0.70229 -0.01988  0.67990  2.16482
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.0198     0.1062  -0.186    0.853
## x.corx1_cor   0.3872     0.2825   1.371    0.174
## x.corx2_cor   1.9115     0.0975  19.605  < 2e-16 ***
## x.corx3_cor   0.5754     0.0972   5.920 5.11e-08 ***
## x.corx4_cor  -2.4787     0.2437 -10.172  < 2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.043 on 95 degrees of freedom
## Multiple R-squared:  0.9148, Adjusted R-squared:  0.9112
## F-statistic: 254.8 on 4 and 95 DF,  p-value: < 2.2e-16
```

Note how only the estimate of x1 and x4 are off: the reason for this is that x1 and x4 are correlated, while x2 and x3 are indipendend from all the others.

Tree

Finally we'll turn to a kind of models completely different from what we've seen so far: trees. We'll now apply a Tree to the Carseats dataset to predict the value of Sales given the other variables.

```
library(ISLR)
attach(Carseats)
head(Carseats)
```

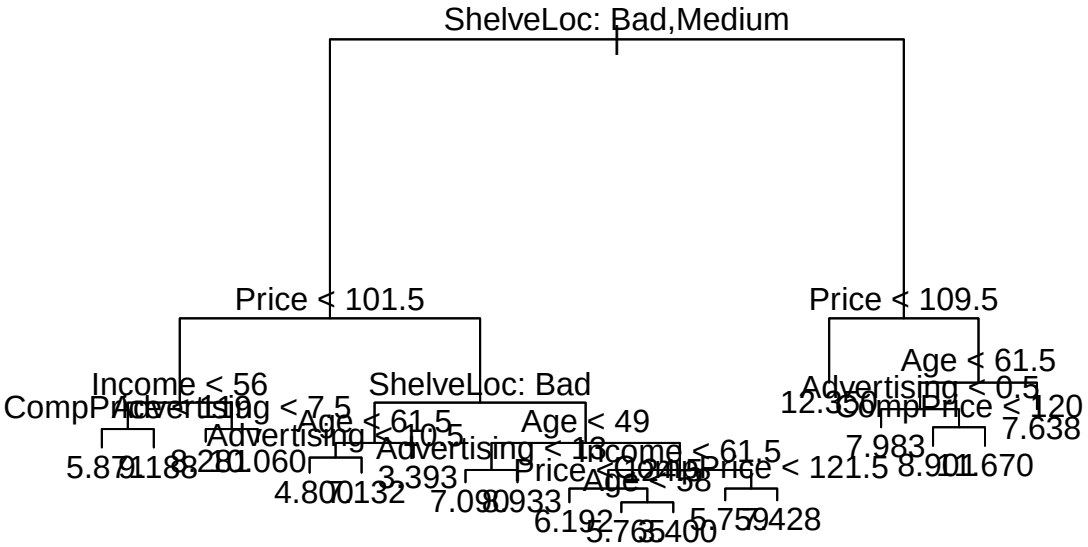
```
##   Sales CompPrice Income Advertising Population Price ShelveLoc Age Education
## 1  9.50       138     73          11        276   120       Bad  42        17
## 2 11.22       111     48          16        260    83      Good  65        10
## 3 10.06       113     35          10        269    80    Medium  59        12
## 4  7.40       117    100           4        466    97    Medium  55        14
## 5  4.15       141     64           3        340   128       Bad  38        13
## 6 10.81       124    113          13        501    72       Bad  78        16
##   Urban  US
## 1   Yes Yes
## 2   Yes Yes
## 3   Yes Yes
## 4   Yes Yes
## 5   Yes  No
## 6    No Yes
```

We'll first fit a regression tree to the data, using a validation set approach to estimate its test performance.

```
library(tree)
# Let's first split the data into training and test sets
n <- dim(Carseats)[1]
indices <- sample(1:n, size=n, replace=F)
test <- indices[1:n/3]
length(test)
```

```
## [1] 398
```

```
tree.full <- tree(Sales~., data=Carseats, subset=-test)
plot(tree.full)
text(tree.full, pretty=0)
```




```
summary(tree.full)
```

```
##
## Regression tree:
## tree(formula = Sales ~ ., data = Carseats, subset = -test)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Income"      "CompPrice"   "Advertising"
## [6] "Age"
## Number of terminal nodes:  19
## Residual mean deviance:  2.239 = 555.2 / 248
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -4.0580 -1.0940  0.1095  0.0000  1.0540  3.9210
```

We have used only 5 of the 10 predictors available, obtaining 17 terminal nodes in total. What test MSE do we get with this fully grown tree?

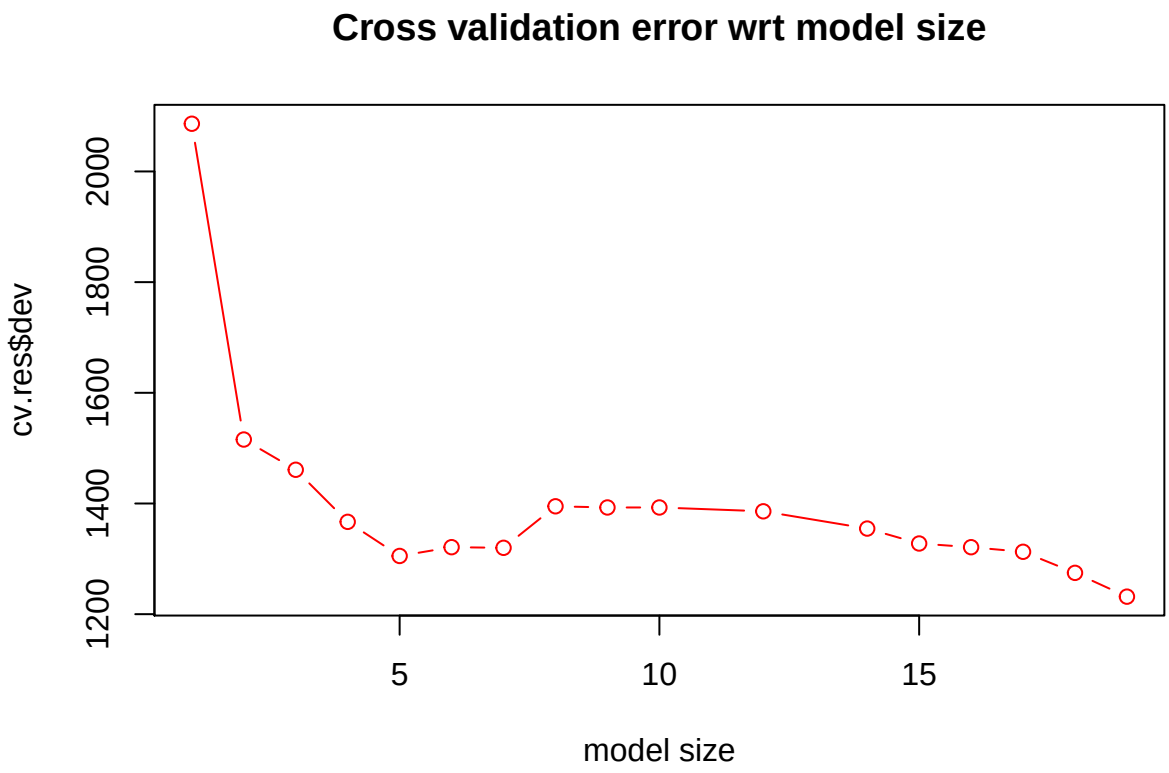
```
preds <- predict(tree.full, newdata = Carseats[test,])
test.MSE <- mean((Carseats$Sales[test]-preds)^2)
test.MSE
```

```
## [1] 4.850144
```

AN MSE of practically 4, which means we're on average off by 2000$ on each prediction. Let's see if we can lower this error using pruning.

To determine the optimal pruning level we use cross validation on our fully grown tree.

```
cv.res <- cv.tree(tree.full, K = 10)
plot(cv.res$size, cv.res$dev, main = "Cross validation error wrt model size", type="b", col="red", xlab=
```

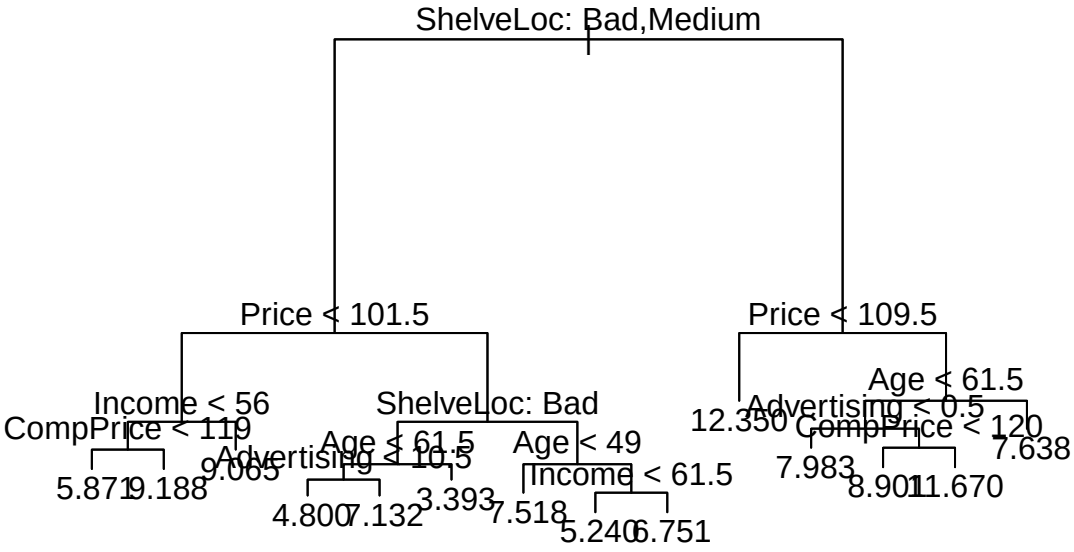



```
cv.res$size[which.min(cv.res$dev)]
```

```
## [1] 19
```

The optimal size is 13 leaf nodes, according to the cross validation results. Let's use this pruned tree to make predictions.

```
tree.pruned <- prune.tree(tree.full, best = 13)
plot(tree.pruned)
text(tree.pruned, pretty=0)
```



Let's look at the Test MSE for this pruned version.

```
preds.pruned <- predict(tree.pruned, newdata = Carseats[test, ])
error.pruned <- mean((Carseats$Sales[test]-preds.pruned)^2)
error.pruned
```

```
## [1] 5.130844
```

The test MSE increased by 0.5, hence the pruning didn't seems to improve our model. Let's now refer to more robust methods such as bagging and random forests to analyze this data.

```
library(randomForest)
```

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

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

```
tree.bagged <- randomForest(formula=Sales~., data =Carseats, mtry=10, subset = -test)
tree.bagged
```

```
##
## Call:
##  randomForest(formula = Sales ~ ., data = Carseats, mtry = 10,      subset = -test)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 10
##
##           Mean of squared residuals: 2.62575
##                     % Var explained: 65.93
```

The mean of squared residuals we see in the summary is obtained by evaluating the performance on the OOB samples for each bootstrapped dataset. If the estimate is reliable the bagged version has almost halved our error. Let's look at the test MSE.

```
preds.bagged <- predict(tree.bagged, newdata = Carseats[test,])
error.bagged <- mean((Carseats$Sales[test]-preds.bagged)^2)
error.bagged
```

```
## [1] 2.60636
```

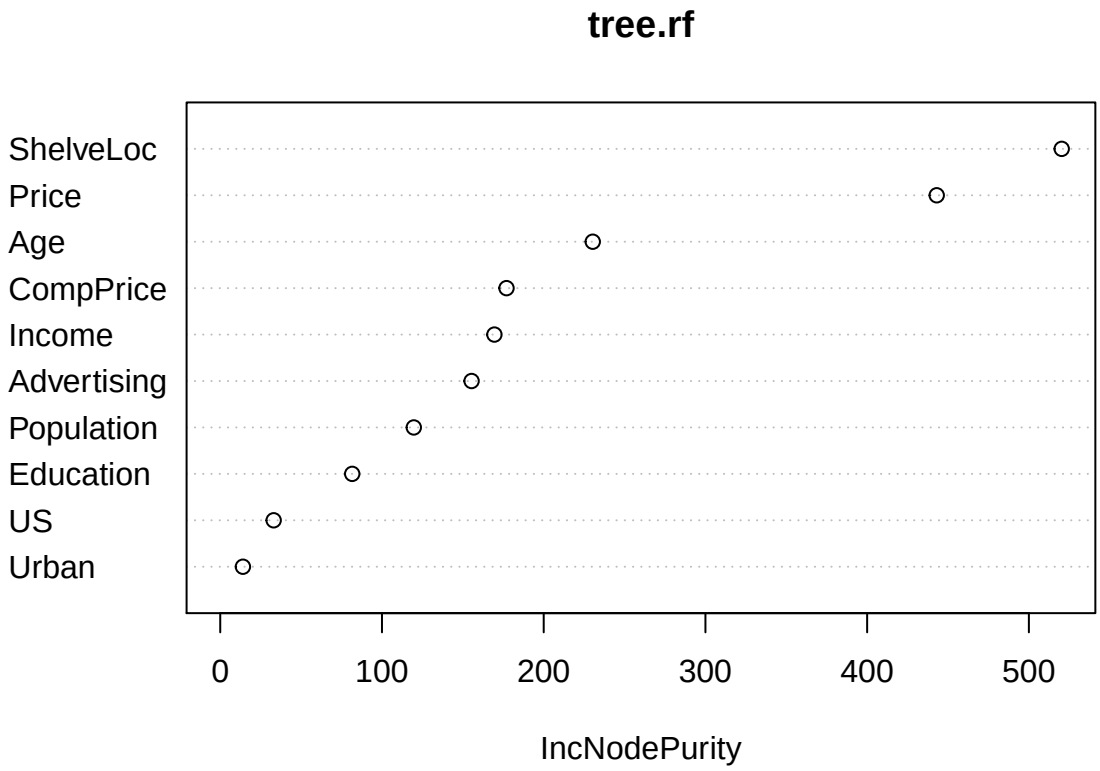
Indeed the OOB estimation of the error is in line with the test MSE: bagging has decreased the error from approximately 4 to approximately 2.3. One problem with bagging is that the predictors tend to be highly correlated. A simple way to solve this is to allow each split to be done on a random subset of the predictors: this is the random forest approach.

```
# by default mtry ) p/3 : almost 4 in our case
tree.rf <- randomForest(formula=Sales~., data =Carseats, subset = -test)
tree.rf
```

```
##
## Call:
##  randomForest(formula = Sales ~ ., data = Carseats, subset = -test)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 2.965125
##                     % Var explained: 61.52
```

The OOB estimate seems to be increased wrt bagging. What is this telling us? As the main distinctive feature of trees is the splitting on a random subset, this could mean that the available features at each cut are not enough to lead to signigicant improvements in the model. This can happen when there's not much overlapping between the dimensions in the dataset, while only few of them are predictive. In order to prove this reasoning we look at the variable importance, which is an analysis tools available in random forests.

```
varImpPlot(tree.rf)
```



The plot above seem to validate our resoning. There are only two variables that significantly determine the model performance. This leads naturally to poorer estimates when these 2 variables are not "randomly selected" for the split.

Let's look at the test MSE as we've done for the previous cases to conclude our analysis.

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

```
## [1] 3.083141
```