Non-Linear Statistical Analysis

Laura Cline

16/10/2021

Non-Linear Modeling

In this lab, we will analyze the `Wage` data We begin by loading the `ISLR` library, which contains the data.

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

Polynomial Regression and Step Functions

We first fit the model using the following command:

```
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
```

This syntax fits a linear model, using the `lm()` function in order to predict `wage` using a fourth-degree polynomial in `age:poly(age,4)`. The `poly()` command allows us to avoid having to write out a long formula with powers of `age`. The function returns a matrix whose columns are a basis of *orthogonal polynomials*, which essentially means that each column is a linear combination of the 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, if we prefer. We can do this by using the `raw=TRUE`argument to the `poly()` function. Later we see that 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.

```
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
```

There are several other equivalent ways of fitting this model, which showcase the flexibility of the formula language in `R`. For example:

```
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
```

This simply creates the polynomial basis function on the fly, taking care to protect terms like `age^2` via the *wrapper* function `I()` (the ^ symbol has a special meaning in the formulas).

```
fit2b = lm(wage~cbind(age, age^2, age^3, age^4), data=Wage)
```

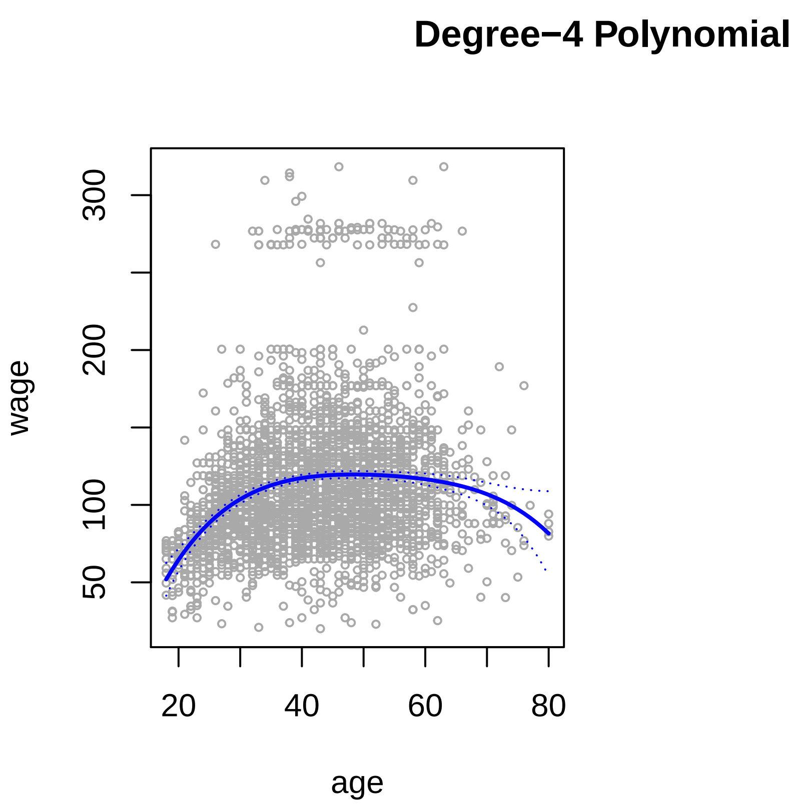
This does the same more compactly, using the `cbind()` function for building a matrix from a collection of vectors; any function call such as `cbind()` inside a formula also serves as a wrapper.

We now create a grid of values for `age` at which we want predictions, and then we call the generic `predict()` function, specifying that we want standard errors as well.

```
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)
```

Finally, we plot the data and add the fit from the degree-4 polynomial.

```
par(mfrow=c(1,2), mar=c(4.5, 4.5, 1, 1), oma = c(0,0,4,0))
plot(age, wage, xlim=agelims, cex=0.5, col='darkgrey')
title("Degree-4 Polynomial", outer=T)
lines(age.grid, preds$fit, lwd=2, col="blue")
matlines(age.grid, se.bands, lwd=1, col="blue", lty=3)
```



Here the `mar` and `oma` arguments to `par()` allow us to control the margins of the plot, and the `title()` function creates a figure title that spans both subplots.

We mentioned earlier that whether or not an orthogonal set of basis functions is produced in the `poly()` function will not affect the model obtained in a meaningful way. What do we mean by this? The fitted values obtained in either case are identical:

```
preds2 = predict(fit2, newdata=list(age=age.grid), se=TRUE)
max(abs(preds$fit - preds2$fit))
```

```
## [1] 7.81597e-11
```

In performing a polynomial regression we must decide on the degree of the polynomial to use. One way to do this is by using hypothesis tests. We now fit models ranging from linear to a degree-5 polynomial and seek to determine the simplest model which is sufficient to explain the relationship between `wage` and `age`. We use the `anova()` function, which performs an *analysis of variance* (ANOVA, using an F-test) in order to test the null hypothesis that the model M_1 is sufficient to explain the data against the alternative hypothesis that a more complex M_2 is required. In order to use the `anova()` function, M_1 and M_2 must be *nested* models: the predictors of M_1 must be a subset of the predictors in M_2. In this case, we fit five different models and sequentially compare the simpler model to the more complex model.

```
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
```

The p-value comparing the linear `Model 1` to the quadratic `Model 2` is essentially zero ($< 10^{-15}$) indicating that a linear fit is not sufficient. Similarly, the p-value comparing the quadratic `Model 2` to the cubic `Model 3` is very low (0.0016), so the quadratic fit is also insufficient. The p-value comparing the cubic and degree-4 polynomials, `Model 3` and `Model 4`, is approximately 5% while the degree-5 polynomial `Model 5` seems unnecessary because its p-value is 0.36. Hence, either a cubic or a quartic polynomial appear to provide a reasonable fit to the data, but lower- or higher-order models are not justified.

In this case, instead of using the `anova()` function, we could have obtained these p-values more succiently by exploiting the fact that `poly()` creates orthogonal polynomials.

```
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
```

Notice that the p-values are the same, and in fact the square of the t-statistics are equal to the F-statistics from the anova()‘ function; for example:

```
(-11.983)^2
```

```
## [1] 143.5923
```

However, the ANOVA methods works whether or not we used orthogonal polynomials; it also works when we have other terms in the model as well. For example, we can use `anova()` to compare these three models:

```
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
```

As an alternative to using hypothesis tests and ANOVA, we could choose the polynomial degree using cross-validation.

Next we consider the task of predicting whether an individual earns more than $250,000 per year. We proceed such as before, except that first we create the appropriate response vector, and then apply the `glm()` function using `family="binomial"` in order to fit a polynomial logistic regression model.

```
fit = glm(I(wage>250)~poly(age,4), data=Wage, family=binomial)
```

Note that we again use the wrapper `I()` to create this binary response variable on the fly. The expression `wage>250` evaluates the logical variable containing the `TRUE`s and `FALSE`s, which `glm()` coerces to binary by setting the `TRUE`s to 1 and `FALSE`s to 0.

Once again, we make predictions using the `predict()` function.

```
preds = predict(fit, newdata=list(age=age.grid), se=T)
```

However, calculating the confidence intervals is slightly more involved than in the linear regression case. The default prediction type for a `glm()` model is `type="link"`, which is what we use here. This means that we get predictions for the *logit*: that is, we have fit a model of the form:

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

and the predictions given are of the form $X\hat{\beta}$. The standard errors given are also of this form. 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))
```

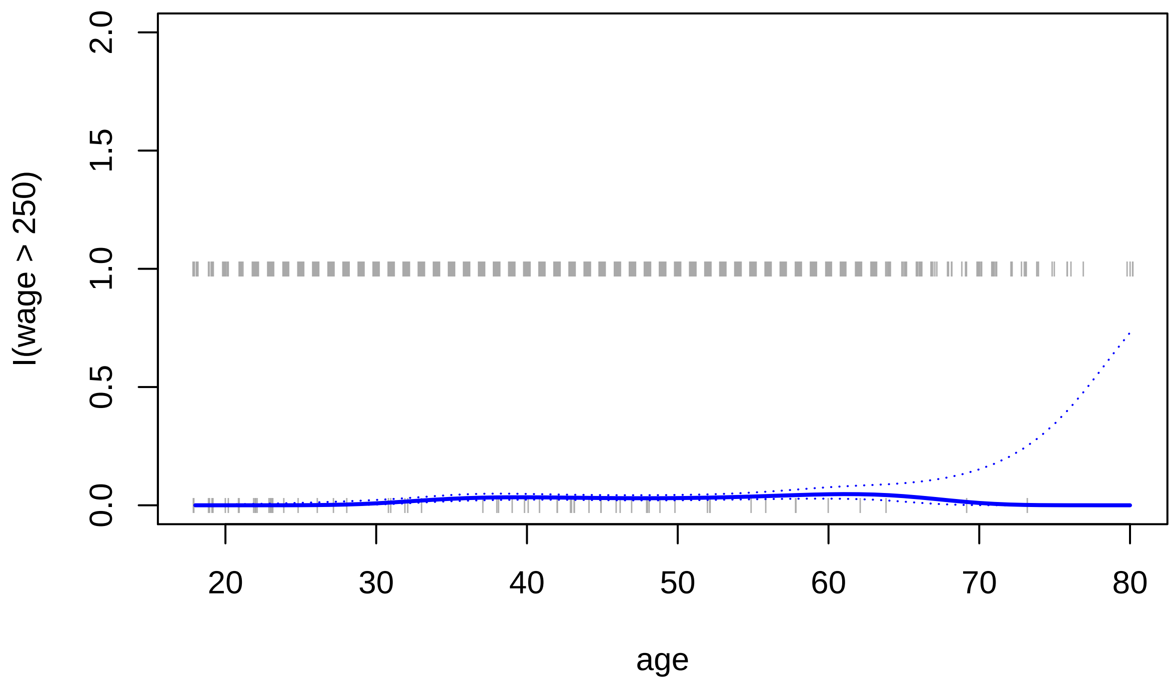
Note that we could have directly computed the probabilities by selecting the `type="response"` option in the `predict()` function.

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

However, the corresponding confidence intervals would not have been sensible because we would end up with negative probabilities!

Finally, we make a plot.

```
plot(age, I(wage>250), xlim=agelims, type="n", ylim=c(0,2))
points(jitter(age), I((wage>250/5)), cex=0.5, pch="|", col="darkgrey")
lines(age.grid, pfit, lwd=2, col="blue")
matlines(age.grid, se.bands, lwd=1, col="blue", lty=3)
```



We have drawn the `age` values corresponding to the observations with `wage` values above 250 as gray marks on the top of the plot, and those with `wage` values below 250 are shown as gray marks at the bottom of the plot. We used the `jitter()` function to jitter the `age` values a bit so that observations with the same `age` value do not cover each other up. This is often called a *rug plot*.

In order to fit a step function, we use the `cut()` function.

```
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))
```

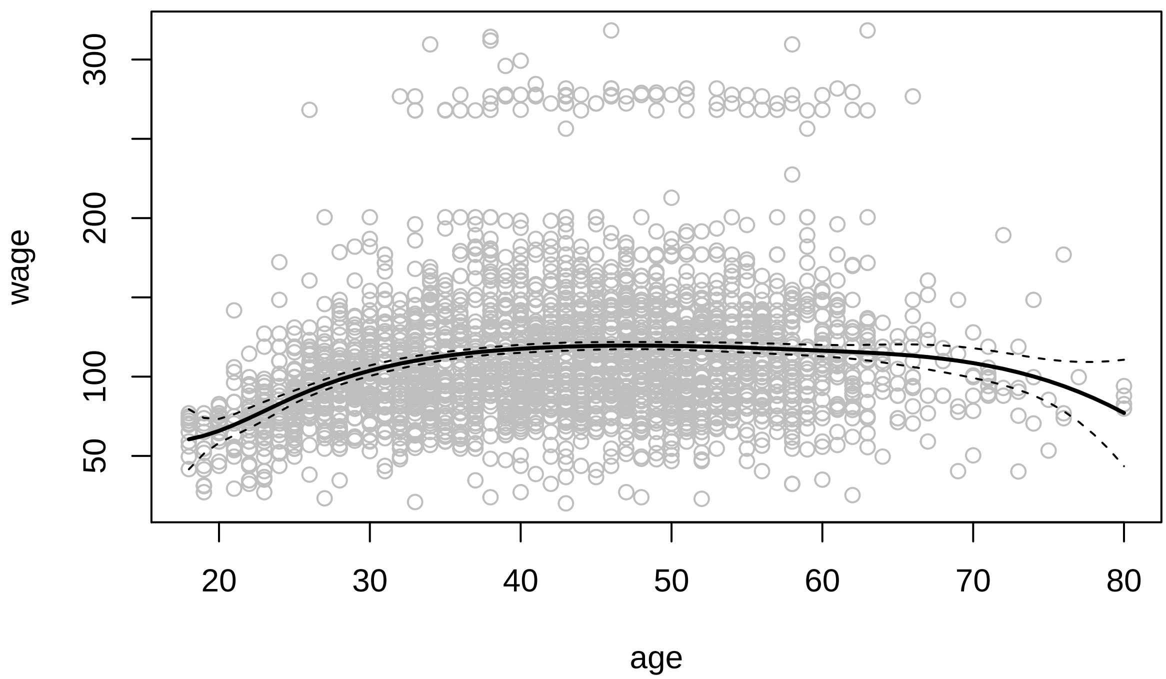
```
##                         Estimate Std. Error   t value     Pr(>|t|)
## (Intercept)            94.158392   1.476069 63.789970 0.000000e+00
## cut(age, 4)(33.5,49]   24.053491   1.829431 13.148074 1.982315e-38
## cut(age, 4)(49,64.5]   23.664559   2.067958 11.443444 1.040750e-29
## cut(age, 4)(64.5,80.1]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

Here `cut()` automatically picked up the cutpoints 33.5, 49, and 64.5 years of age. We could also have specified our own cutpoints directly using the `breaks` option. The function `cut()` returns an ordered categorical variable; the `lm()` function then creates a set of dummy variables for use in the regression. The `age<33.5` category is left out, so the intercept coefficient of $94,158 can be interpreted as the average salary for those under 33.5 years of age, and the other coefficients can be interpreted as the average additional salary for those other age groups. We can produce predictions and plots just as we did in the same of the polynomial fit.

Splines

In order to fit regression splines in `R`, we use the `splines` library. We know that regression splines can be fit by constructing an appropriate matrix of basis functions. The `bs()` function generates an entire matrix of basis functions for splines with the specified set of knots. By default, cubic splines are produced. Fitting `wage` and `age` using a regression spline is simple:

```
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")
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")
```



Here we have prespecified knots at ages 25, 40 and 60. This produces a splines with six basis functions. Recall that a cubic splines 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 data.

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

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

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

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

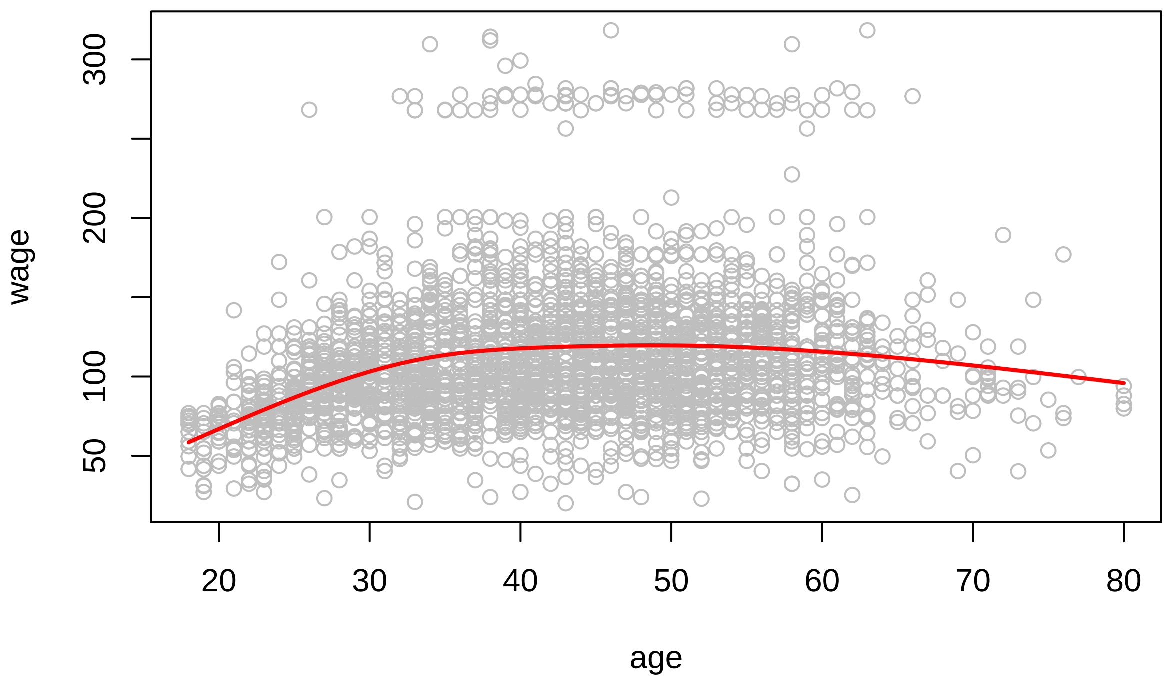
```
attr(bs(age, df=6), "knots")
```

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

In this case R chooses knots at ages 33.75, 42.0 and 51.0, which correspond to the 25th, 50th, and 75th percentiles of `age`. The function `bs()` also has a `degree` argument, so we can fit splines of any degree, rather than the default degree of 3 (which yields a cubic spline).

In order to instead fit a natural spline, we use the `ns()` function. Here we fit a natural spline with four degrees of freedom.

```
fit2 = lm(wage~ns(age, df=4), data=Wage)
pred2 = predict(fit2, newdata=list(age=age.grid), se=T)
plot(age, wage, col="gray")
lines(age.grid, pred2$fit, col="red", lwd=2)
```



As with the `bs()` function, we could instead specify the knots directly using the `knots` option.

In order to fit a smoothing spline, we use the `smooth.spline()` function.

```
plot(age, wage, xlim=agelims, cex=0.5, col="darkgrey")
title("Smoothing Spline")
fit = smooth.spline(age, wage, df=16)
fit2 = smooth.spline(age, wage, cv=TRUE)
```

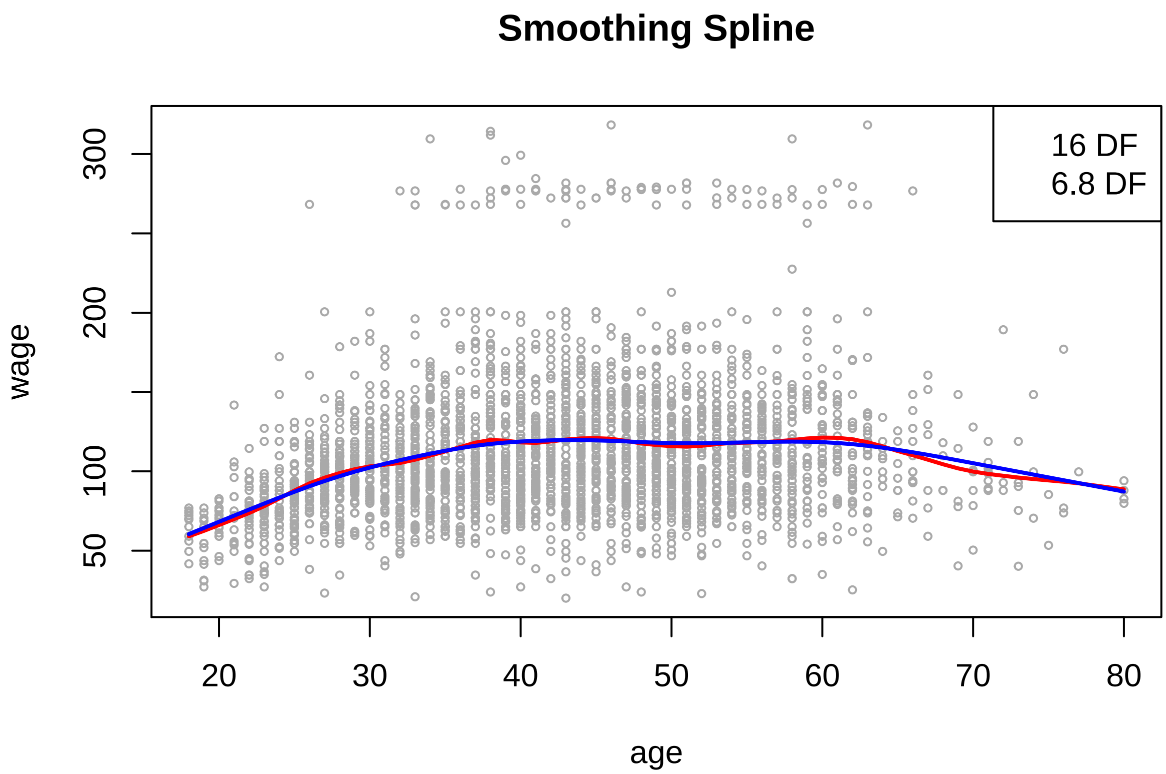
```
## Warning in smooth.spline(age, wage, cv = TRUE): cross-validation with non-unique
## 'x' values seems doubtful
```

```
fit2$df
```

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

```
lines(fit, col="red", lwd=2)
lines(fit2, col="blue", lwd=2)
legend("topright", legend=c("16 DF", "6.8 DF"),
```

```
       col=c("red", "blue", lty=1, lwd=2, cex=0.8))
```



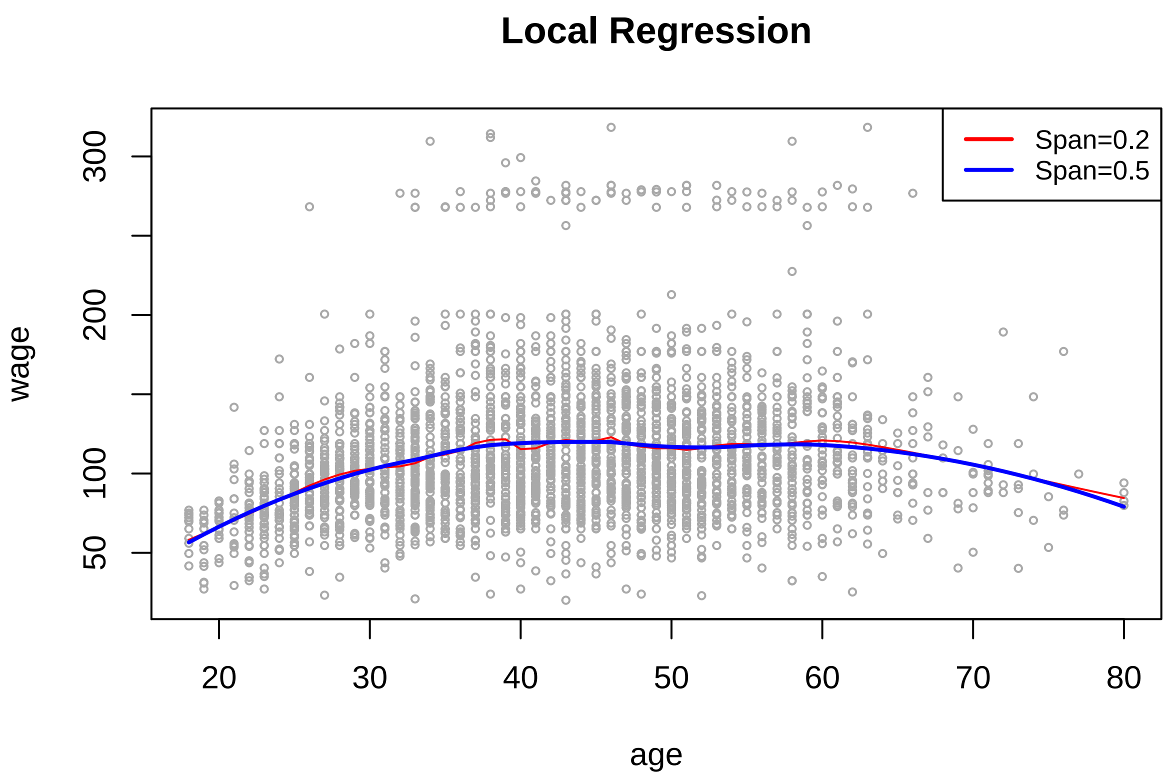
Notice that in the first call to `smooth.spline()`, we specified `df=16`. The function then determines which value of λ leads to 16 degrees of freedom. In the second call to `smooth.spline()`, we select the smoothness level by cross-validation; this results in a value of λ that yields 6.8 degrees of freedom.

In order to perform local regression, we use the `loess()` function.

```
plot(age, wage, xlim=agelims, cex=0.5, col="darkgrey")
title("Local Regression")
fit = loess(wage~age, span=0.2, data=Wage)
fit2 = loess(wage~age, span=0.5, data=Wage)
lines(age.grid, predict(fit, data.frame(age=age.grid)), col="red", lwf=2)
```

```
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "lwf" is not a graphical
## parameter
```

```
lines(age.grid, predict(fit2, data.frame(age=age.grid)), col="blue", lwd=2)
legend("topright", legend=c("Span=0.2", "Span=0.5"), col=c("red", "blue"), lty=1, lwd=2, cex=0.8)
```



Here we performed local linear regression using the spans 0.2 and 0.5: that is, each neighbourhood consists of 20% and 50% of the observations. The larger the span, the smoother the fit. The `locfit` library can also be used for fitting local regression models in `R`.

GAMs

We now fit a GAM to predict `wage` using natural spline functions of `year` and `age`, treating `education` as a qualitative predictor. Since this is just a big linear regression model using an appropriate choice of basis functions, we can somply do this using the `lm()` function.

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

We now fit the model using smoothing splines rather than natural splines. In order to fit more general sorts of GAMs, using smoothing splines or other components that cannot be expressed in terms of basis functions and then fit using least squares regression, we will need to use the `gam` library in `R`.

The `s()` function, which is part of the `gam` library, is used to indicate that we would like to use a smoothing spline. We specify that the function of `year` should have 4 degrees of freedom, and that the function of `age` will have 5 degrees of freedom. Since `education` is qualitative, we leave it as is, and it is converted into four dummy variables. We use the `gam()` function in order to fit a GAM using these components. All of the terms are fit simulataneously, taking each other into account to explain the response.

```
#install.packages("gam")
library(gam)
```

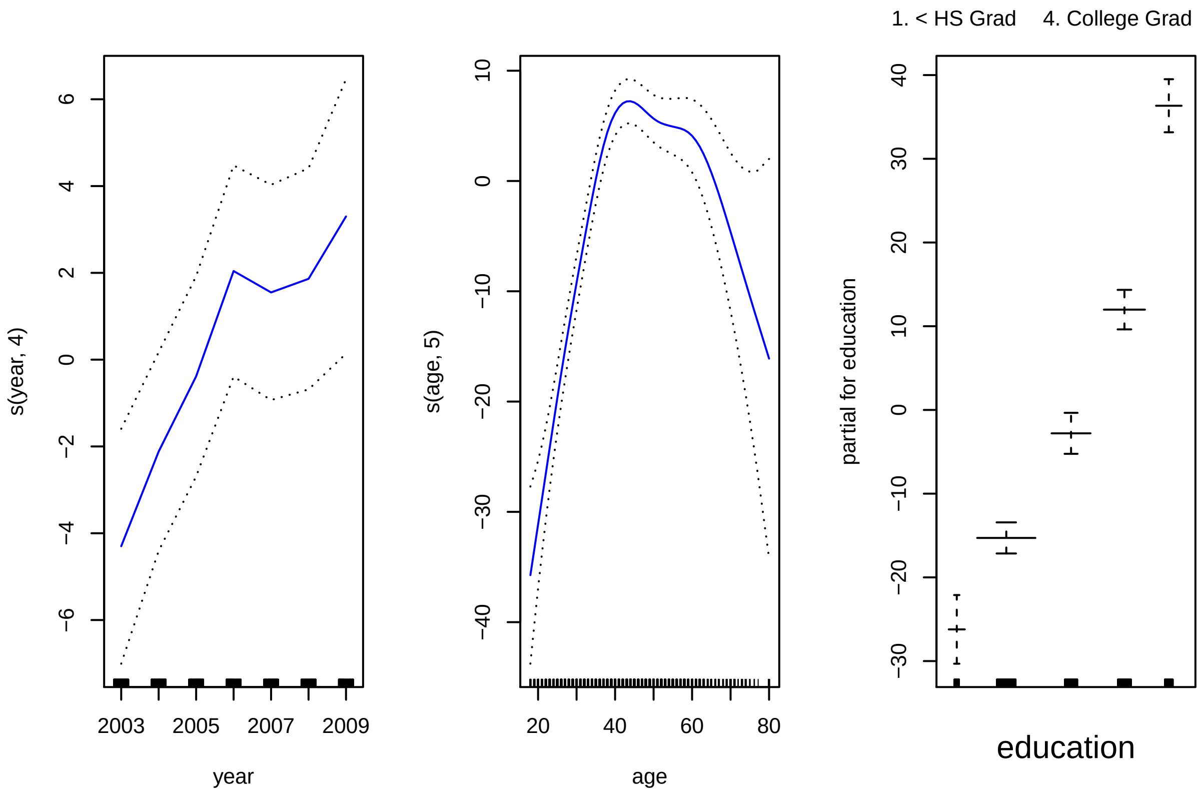
```
## Loading required package: foreach
```

```
## Loaded gam 1.20
```

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

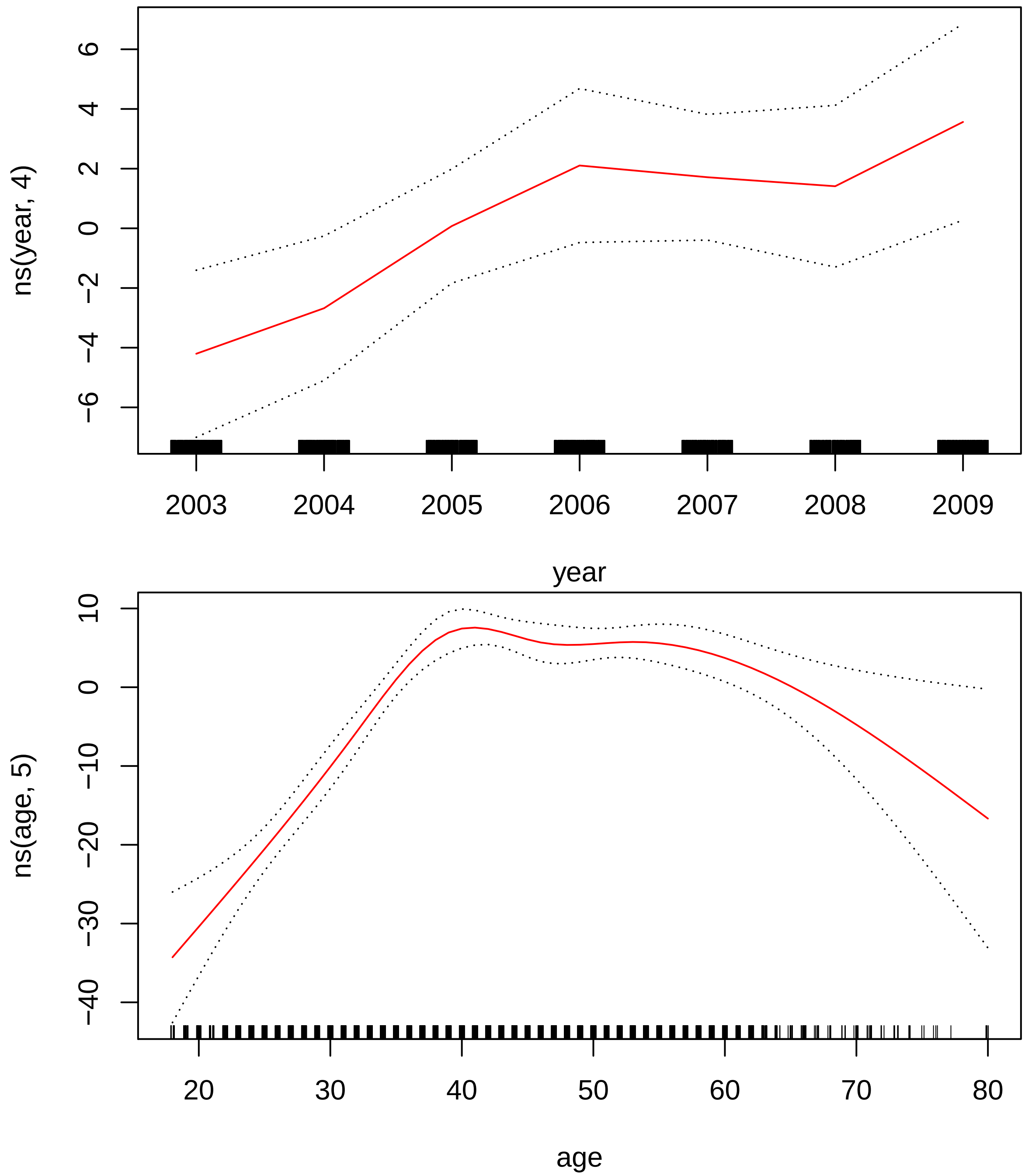
In order to produce the figure, we simply call the `plot()` function.

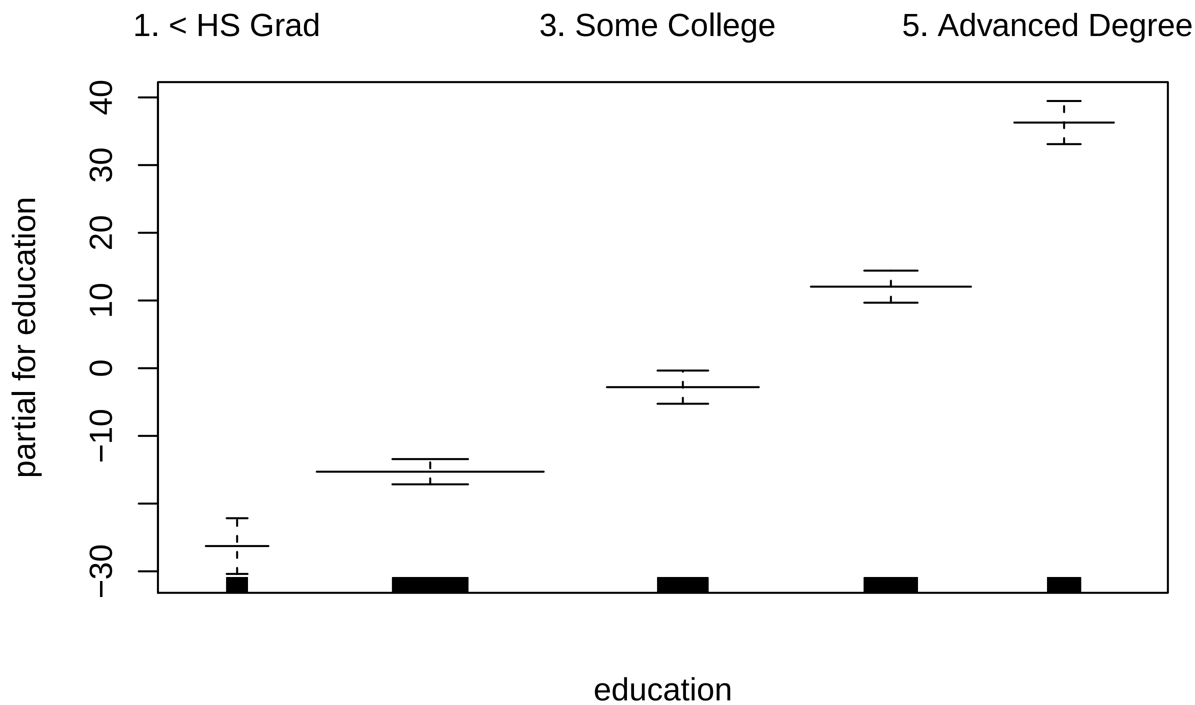
```
par(mfrow = c(1,3))
plot(gam.m3, se=TRUE, col="blue")
```



The generic `plot()` function recognizes that `gam.m3` is an object og class `gam`, and invokes the appropriate `plot.gam()` method. Conveniently, even though `gam1` is not a class of `gam` but rather a class of `lm`, we can *still* use `plot.gam()` on it. We can produce the figure using the following expression:

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


ns(year, 4)
6
4
2
0
−2
−4
−6
2003
2004
2005
2006
2007
2008
2009
year
ns(age, 5)
10
0
−10
−20
−30
−40
20
30
40
50
60
70
80
age



Notice here we had to use the `plot.GAM()` rather than the *generic* `plot()` function.

In these plots, the function of `year` looks rather linear. We can perform a series of ANOVA tests to determine which of these three modles is best: a GAM that excludes `year` ($Model_1$), a GAM that uses a linear function of `year` ($Model_2$), or a GAM that uses a spline function of `year` ($Model_3$).

```
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
```

We find that there is compelling evidence that a GAM with a linear function of `year` is better than a GAM that does not include `year` at all (p-value = 0.00014). However, there is no evidence that a non-linear function of `year` is needed (p-value = 0.348). In other words, based on the results of this ANOVA, M_2 is preferred.

The `summary()` function produces a summary of the gam fit.

```
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
```

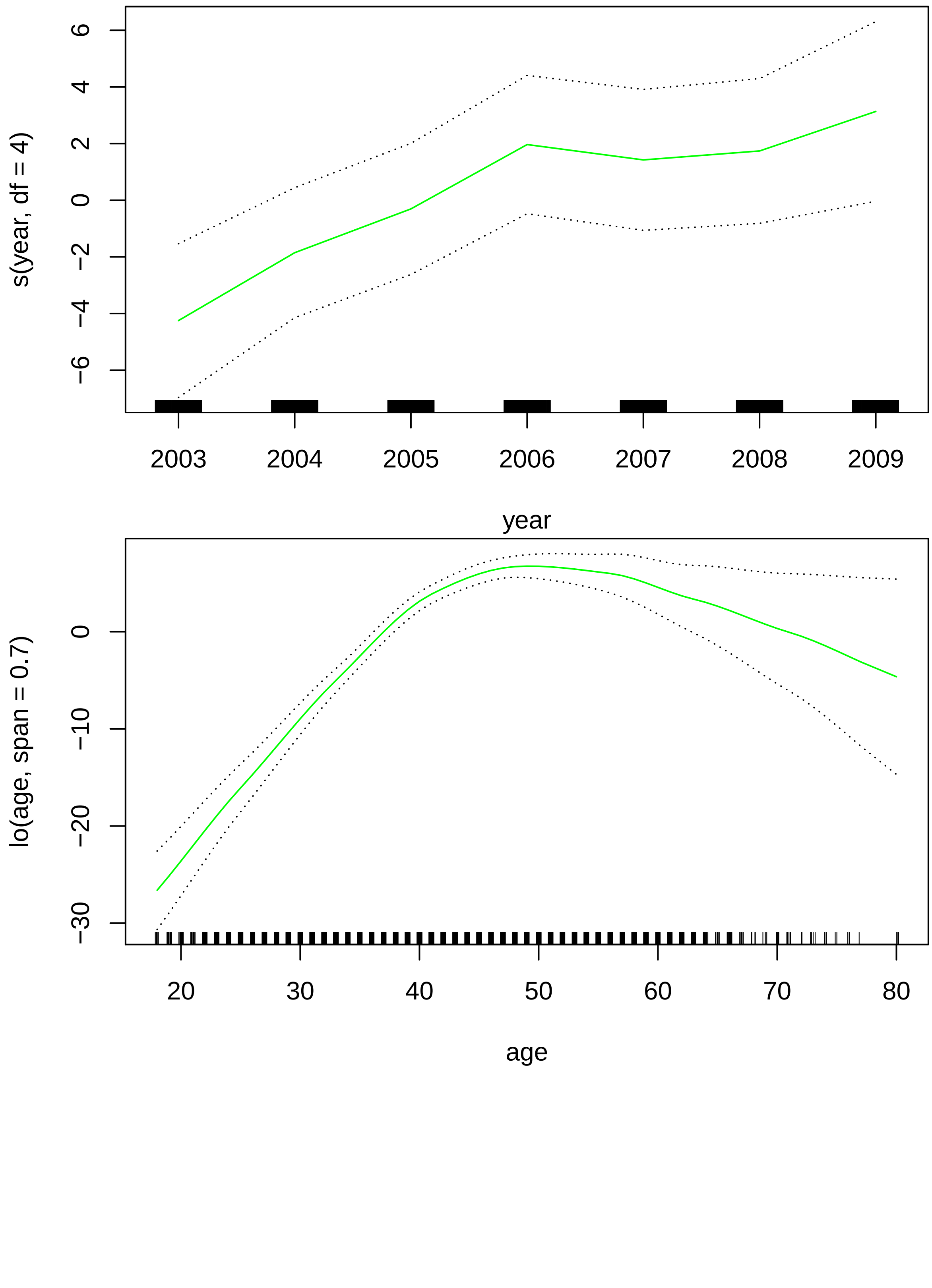
The p-values for `year` and `age` correspond to the null hypothesis of a linear relationship versus the alternative of a non-linear relationship. The large p-value for `year` reinforces our conclusion from the ANOVA test that a linear function is adequate for this term. However, there is a very clear evidence that a non-linear term is required for `age`.

We can make predictions from `gam` objects, just like from `lm` objects, using the `predict()` method for the class `gam`. Here we make predictions on the training set.

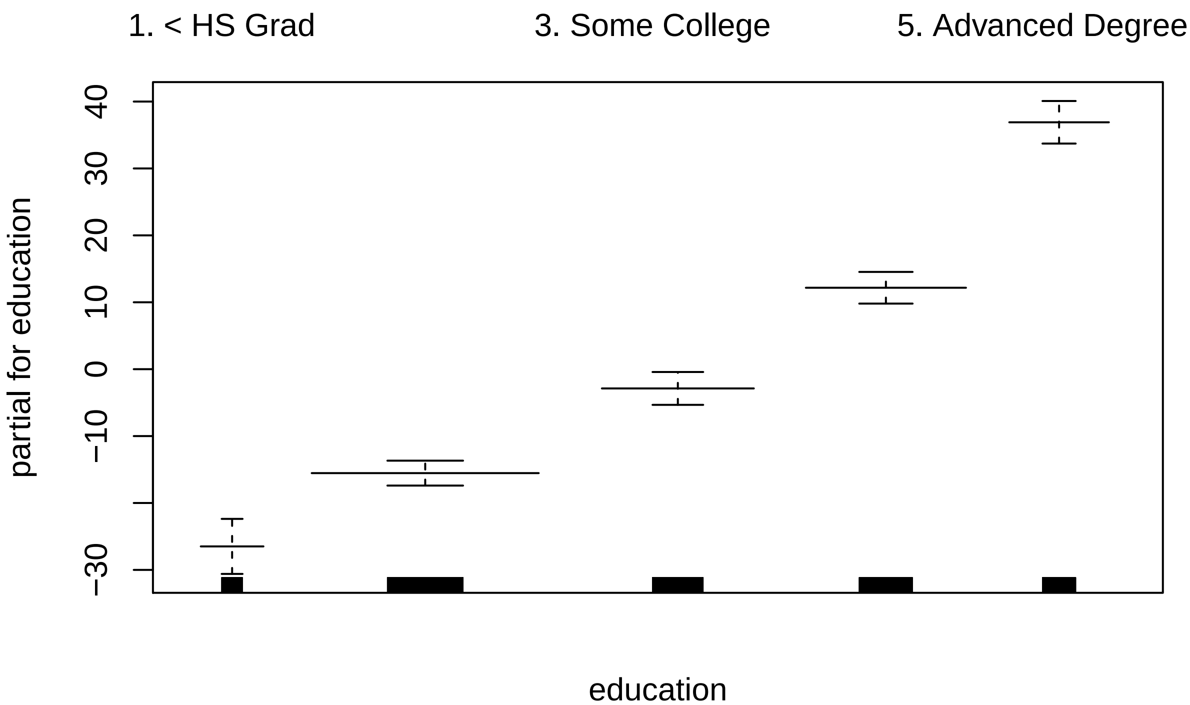
```
preds = predict(gam.m2, newdata=Wage)
```

We can also use local regression fits as building blocks in a GAM, using the `lo()` functiton.

```
gam.lo = gam(wage~s(year, df=4) + lo(age,span=0.7) + education, data=Wage)
plot.Gam(gam.lo, se=TRUE, col='green')
```



s(year, df = 4)
6
4
2
0
−2
−4
−6
2003
2004
2005
2006
2007
2008
2009
year
lo(age, span = 0.7)
0
−10
−20
−30
20
30
40
50
60
70
80
age



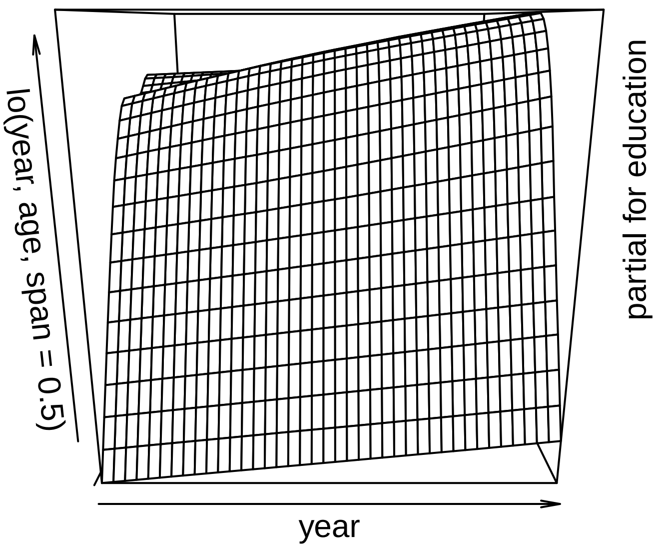
Here we have used local regression for the `age` term, with a span of 0.7. We can also use the `lo()` function to create interactions before calling the `gam()` function. For example,

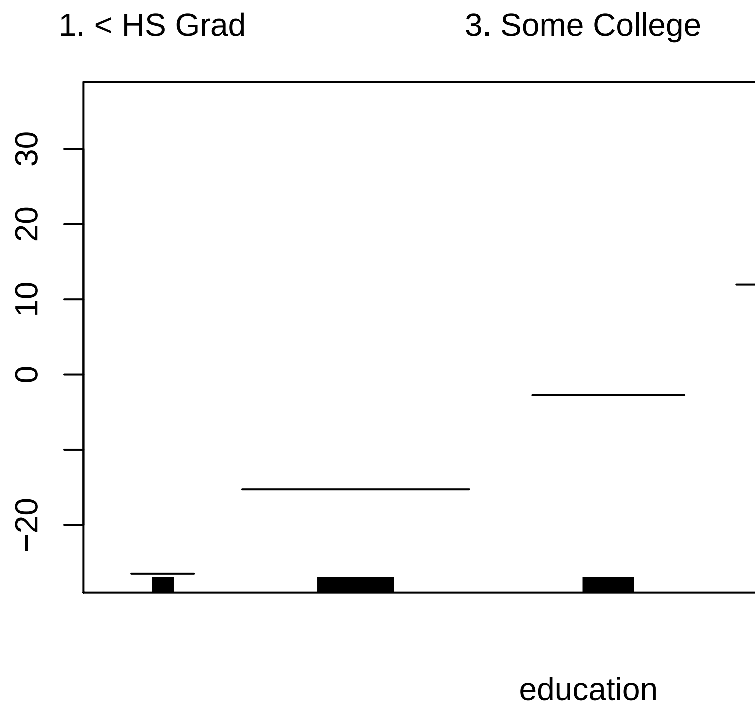
```
gam.lo.i = gam(wage~lo(year, age, span=0.5)+education, data=Wage)
```

```
## 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)
```

fits two-term model, in which the first term is an interaction between `year` and `age`, fit by a local regression surface. We can plot the resulting two-dimensional surfance if we first install the `akima` package.

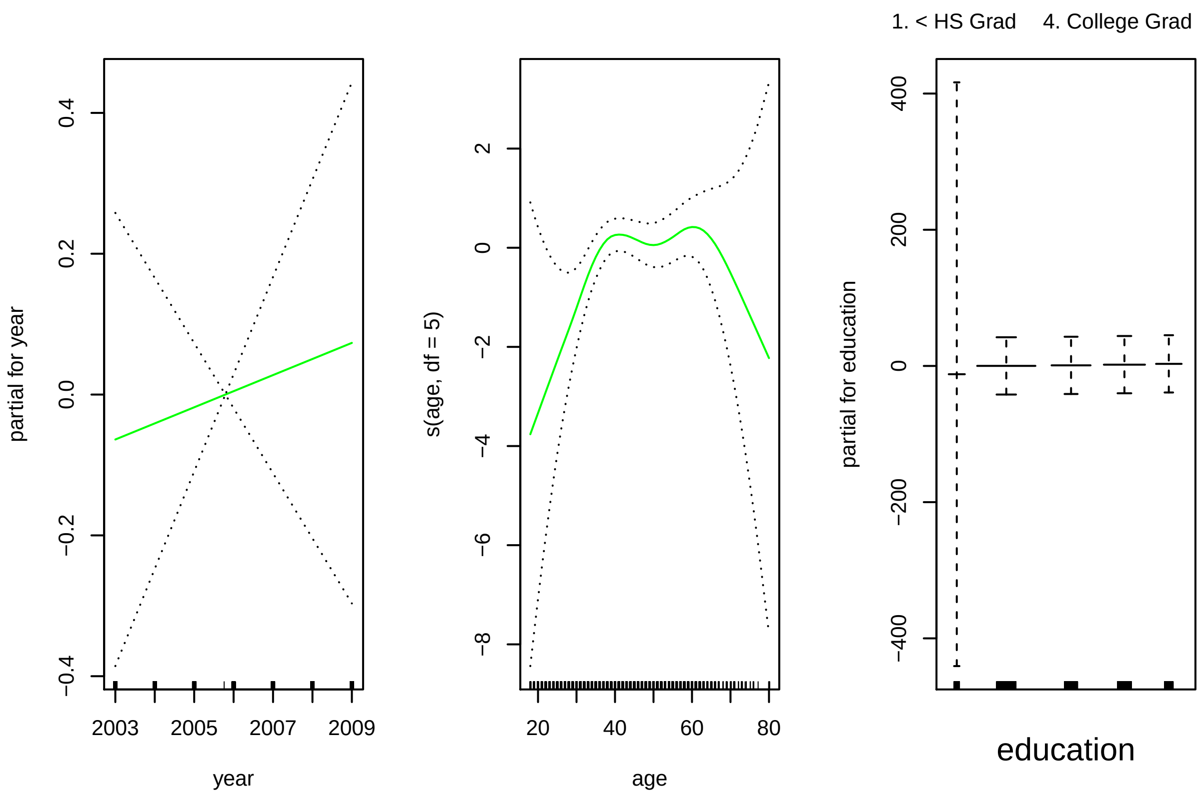
```
#install.packages("akima")
library(akima)
plot(gam.lo.i)
```





In order to fit a logistic regression GAM, we once again use the `I()` function in constructing the binary response variable, and set `family=binomial`.

```
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')
```



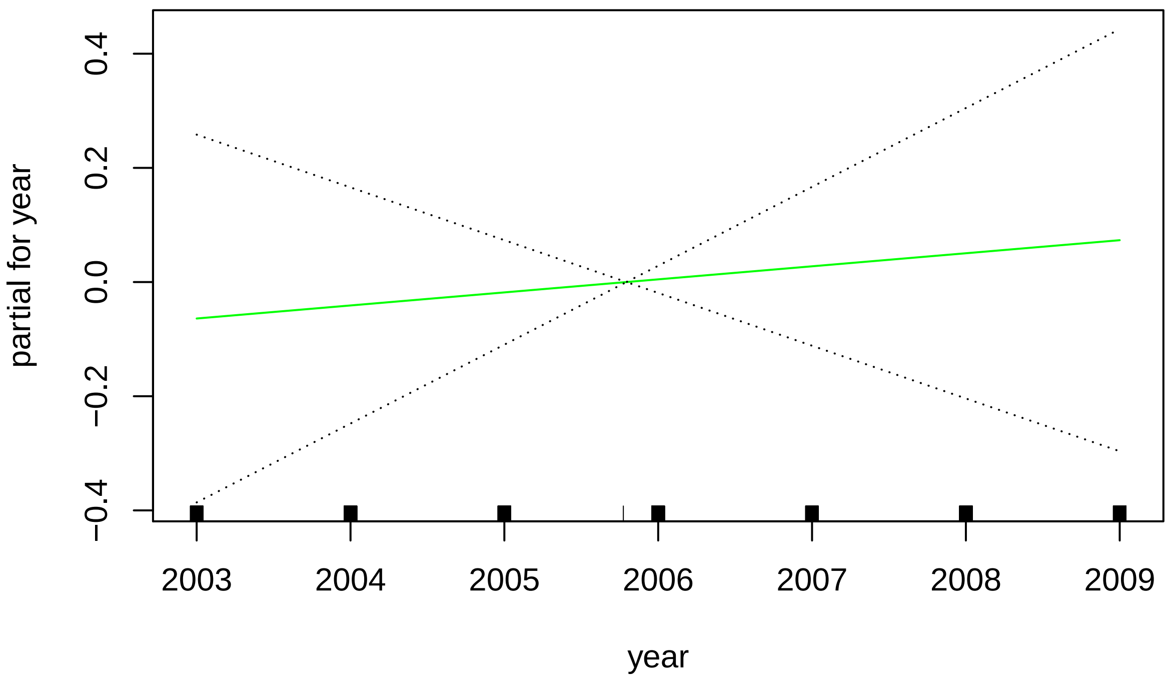
It is easy to see that there are no high earners in the `<HS` category:

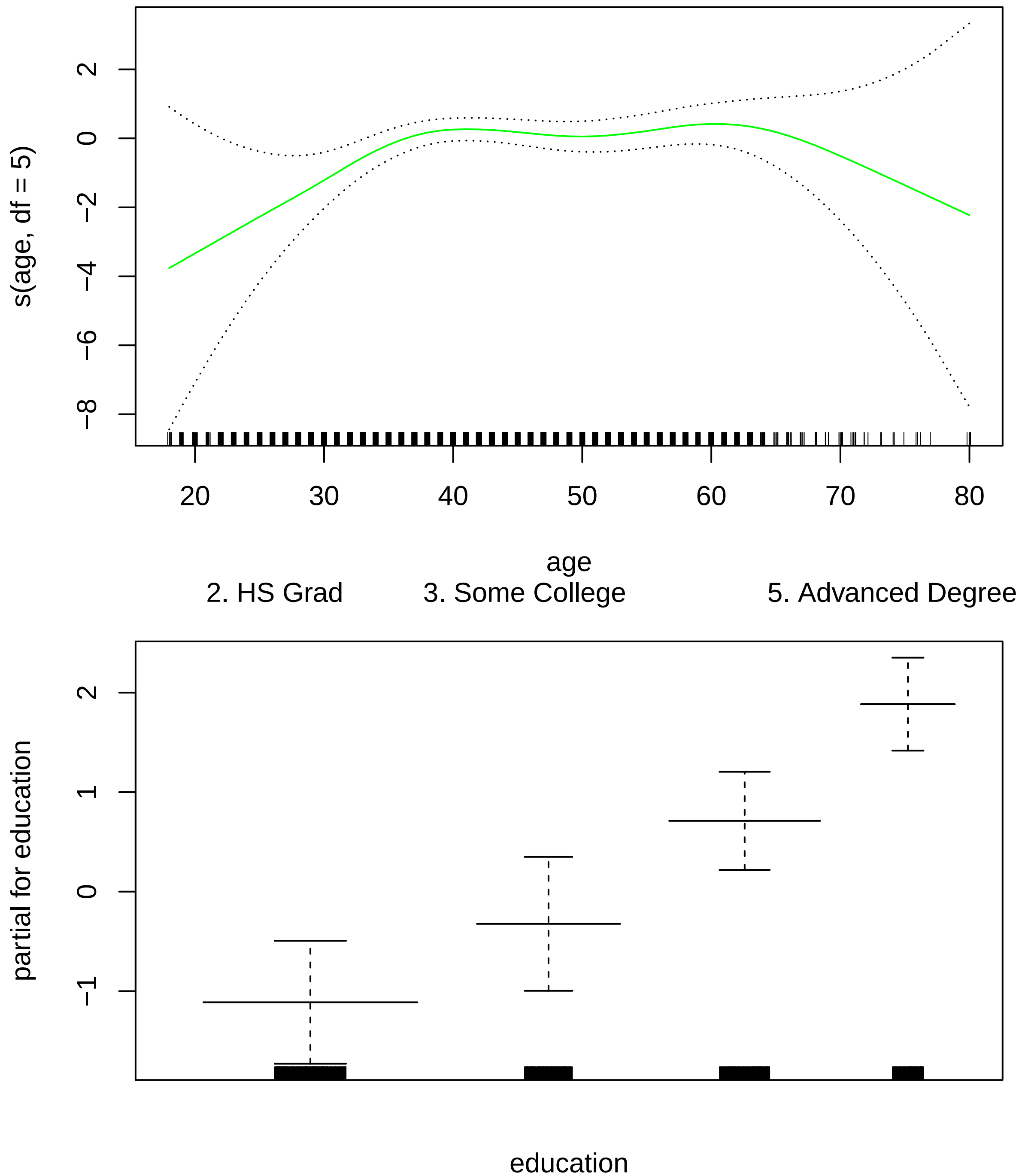
```
table(education, I(wage>250))
```

```
##
## 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
```

Hence, we fit a logistic regression GAM using all but this category. This provides more sensible results.

```
gam.lr.s = gam(I(wage>250)~year+s(age, df=5)+education, family=binomial, data=Wage, subset=(education!=
plot(gam.lr.s, se=T, col='green')
```





Excercises

Question Three

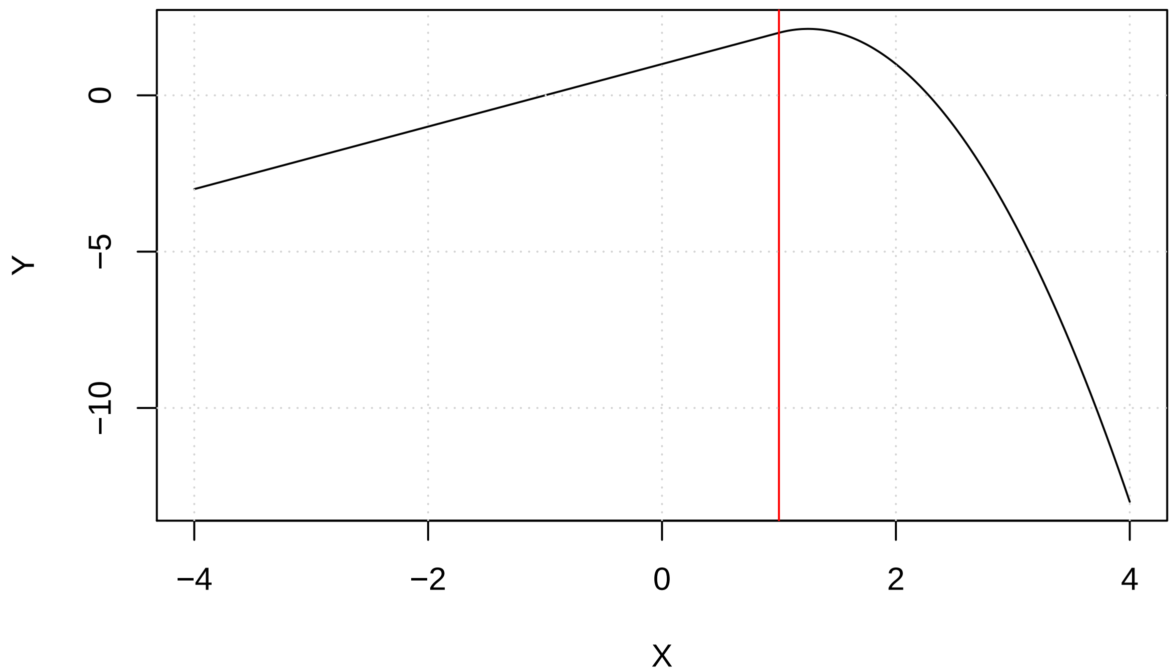
Suppose we fit a curve with a basis function $b_1(X) = X, b_2(X) = (X-1)_2 I(X \geq 1)$. Note that $I(X \geq 1)$ equals 1 for $X \geq 1$ and 0 otherwise. We fit the linear regression model

$Y = \beta_0 + \beta_1 b_1(X) + \beta_2 b_2(X) + \epsilon$

and obtain the coefficient estimates $\hat{\beta}_0 = 1, \hat{\beta}_1 = 1, \hat{\beta}_2 = -2$. Sketch the estimated curve between X = -2 and X = 2. Note the intercepts, slopes and other relevant information.

```r
X = seq(from=-4, to=+4, length.out=500)
Y = 1 + X - 2 * (X-1)^2 * (X >= 1)
```

```r
plot(X, Y, type="l")
abline(v=1, col='red')
grid()
```



Question Four

Suppose we fit a curve with basis functions $b_1 X = I(0 \leq X \leq 2) - (X-1)I(1 \leq X \leq 2), b_2(X) = (X-3)I(3 \leq X \leq 4) + I(4 \leq X \leq 5)$. We fit the linear regression model

$Y = \beta_0 + \beta_1 b_1(X) + \beta_2 b_2(X) + \epsilon$

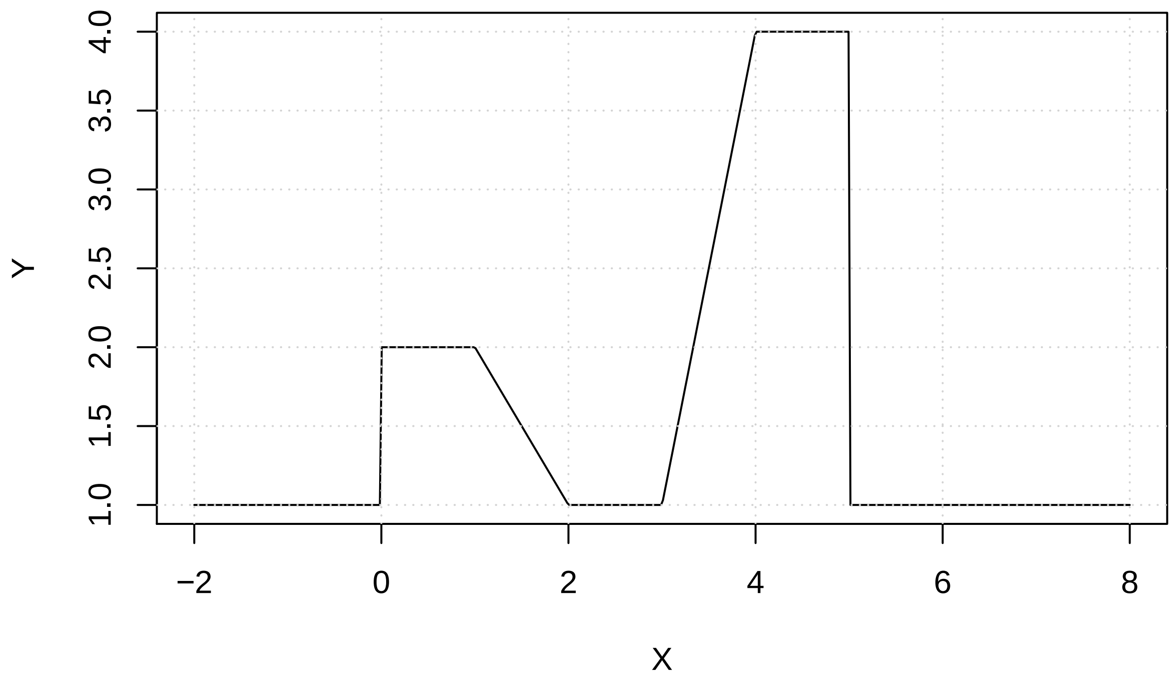
and obtain the coefficient estimates $\hat{\beta}_0 = 1, \hat{\beta}_1 = 1, \hat{\beta}_2 = 3$. Sketch the estimated curve between X = -2 and X = 2. Note the intercepts, slopes, and other relevant information.

```r
X = seq(from=-2, to=+8, length.out=500)
```

```r
# Compute some auxiliary indicator functions:
I_1 = (X >= 0) & (X <= 2)
I_2 = (X >= 1) & (X <= 2)
I_3 = (X >= 3) & (X <= 4)
I_4 = (X >= 4) & (X <= 5)
```

```r
Y = 1 + (I_1 - (X - 1) * I_2) + 3 * ((X - 3) * I_3 + I_4)
```

```r
plot(X, Y, type='l')
grid()
```



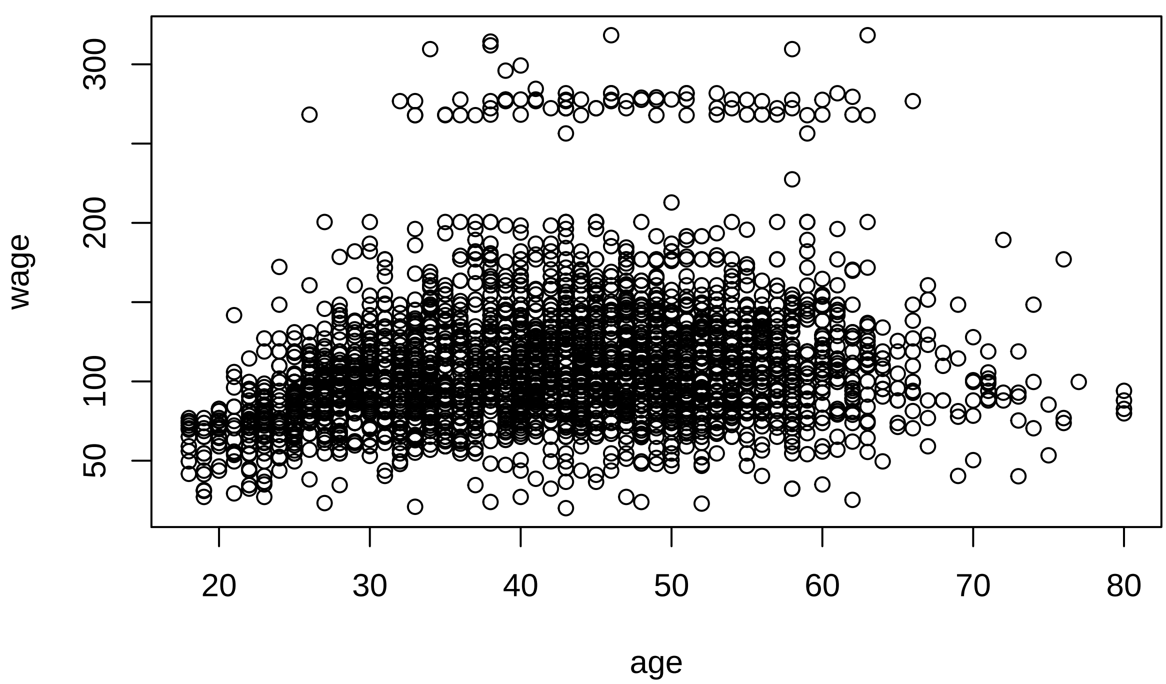
Question Six

In this excercise, you will further analyze the `Wage` dataset.

A. Perform polynomial regression to predict `wage` using `age`. Use cross-validation to select the optimal degree d for the polynomial. What degree was chosen, and how does this compare to the results of hypothesis testing using ANOVA? Make a plot of the resulting polynomial fit to the data.

```
set.seed(0)
```

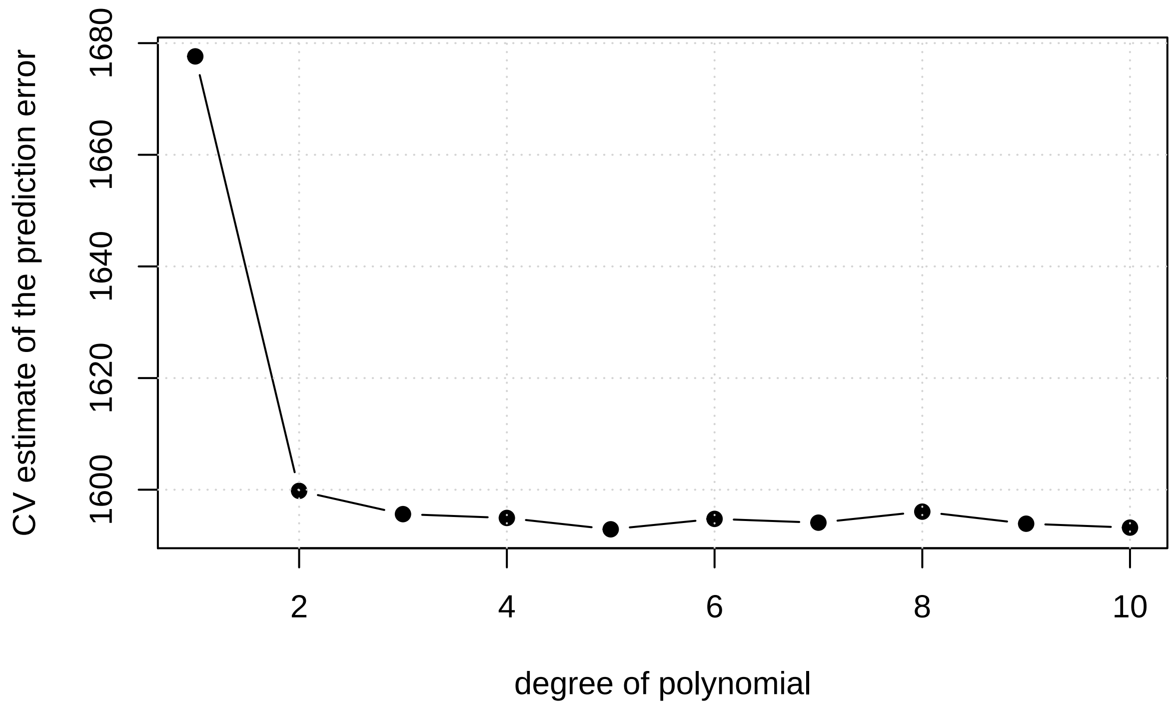
```
# Plot the data to see what it looks like
with(Wage, plot(age,wage))
```



```
library(boot)
```

```
# Perform polynomial regression for various polynomial degrees:
cv.error = rep(0,10)
for( i in 1:10 ){ # fit polynomial models of various degrees
  glm.fit = glm( wage ~ poly(age,i), data=Wage )
  cv.error[i] = cv.glm( Wage, glm.fit, K=10 )$delta[1]
}
```

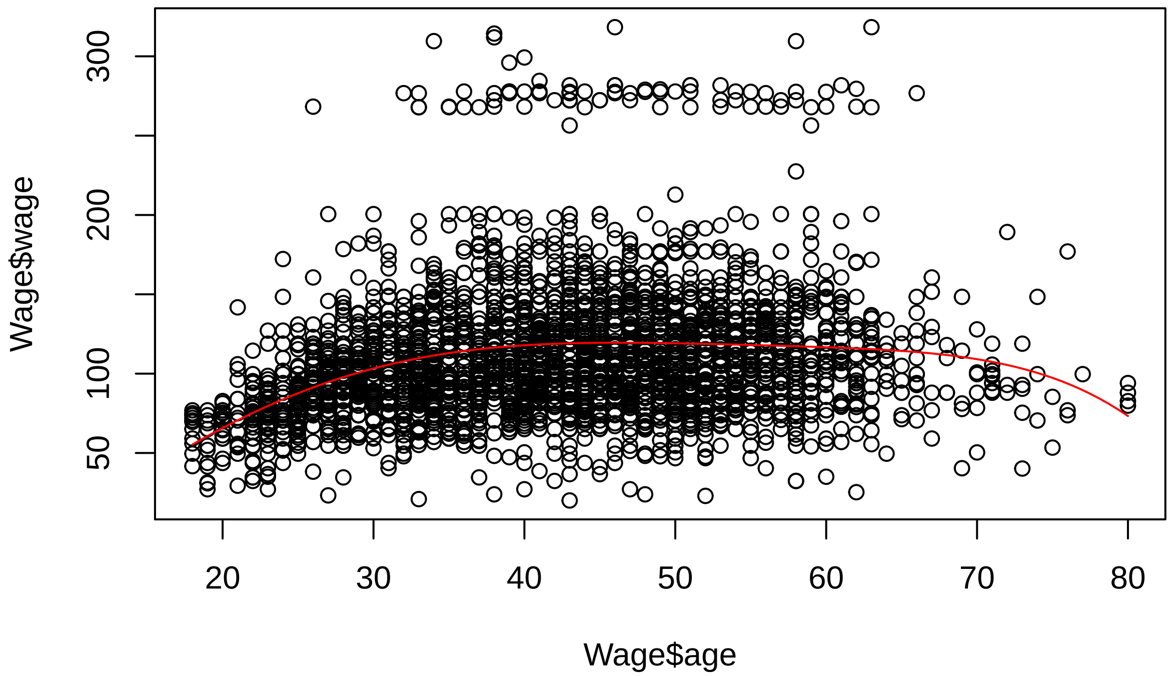
```
plot(1:10, cv.error, pch=19, type='b', xlab='degree of polynomial', ylab='CV estimate of the prediction
grid()
```




```
# Using the minimal value for the CV error gives the value 10 which seems like too much polynomial (i.e
# From the plot, 5 is the point where the curve stops decreasing and starts increasing so we will consi
me = which.min(cv.error)
me = 5
```

```
m = glm(wage ~ poly(age, me), data=Wage)
```

```
plot(Wage$age, Wage$wage)

aRng = range(Wage$age)

a_predict = seq(from=aRng[1], to=aRng[2], length.out=100)
w_predict = predict(m, newdata=list(age=a_predict))
lines(a_predict, w_predict, col='red')
```




```r
# Lets consider the ANOVA approach (i.e., a sequence of nested linear models)
m0 = lm(wage ~ 1, data=Wage)
m1 = lm(wage ~ poly(age,1), data=Wage)
m2 = lm(wage ~ poly(age,2), data=Wage)
m3 = lm(wage ~ poly(age,3), data=Wage)
m4 = lm(wage ~ poly(age,4), data=Wage)
m5 = lm(wage ~ poly(age,5), data=Wage)
anova(m0, m1, m2, m3, m4, m5)
```

```
## Analysis of Variance Table
##
## Model 1: wage ~ 1
## Model 2: wage ~ poly(age, 1)
## Model 3: wage ~ poly(age, 2)
## Model 4: wage ~ poly(age, 3)
## Model 5: wage ~ poly(age, 4)
## Model 6: wage ~ poly(age, 5)
##   Res.Df     RSS Df Sum of Sq        F    Pr(>F)
## 1   2999 5222086
## 2   2998 5022216  1    199870 125.4443 < 2.2e-16 ***
## 3   2997 4793430  1    228786 143.5931 < 2.2e-16 ***
## 4   2996 4777674  1     15756   9.8888  0.001679 **
## 5   2995 4771604  1      6070   3.8098  0.051046 .
## 6   2994 4770322  1      1283   0.8050  0.369682
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

B. Fit a step function to predict `wage` using `age`, and perform cross-validation to choose the optimal number of cuts. Make a plot of the fit obtained.

```r
# Let's do the same thing with the cut function for fitting a piecewise constant model:
# We will do cross-validation by hand
number_of_bins = c(2, 3, 4, 5, 10)
nc = length(number_of_bins)
```

```r
k = 10
folds = sample(1:k, nrow(Wage), replace=TRUE)
cv.errors = matrix(NA, k, nc)
```

```r
# Prepare for the type of factors you might obtain (extend the age range a bit):
age_range = range(Wage$age)
age_range[1] = age_range[1] - 1
age_range[2] = age_range[2] + 1
```

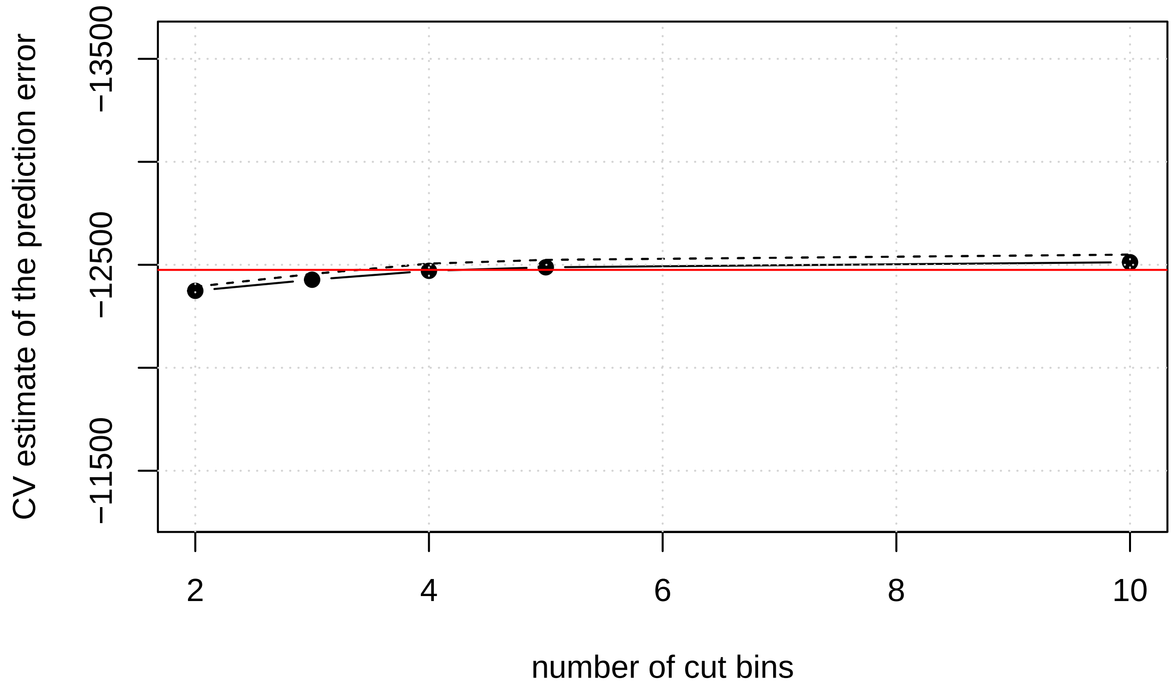
```r
for(ci in 1:nc){
  # For each number fo cuts to test
  nob = number_of_bins[ci] # Number of cuts

  for(fi in 1:k){
    # for each fold
    # In this ugly command we break the "age" variable in the subset of data Wage[folds!=fi,] into "nob
    fit = glm(wage ~ cut(age, breaks=seq(from=age_range[1], to= age_range[2], length.out=(nob+1))), data
    y_hat = predict(fit, newdata=Wage[folds==fi,])
    cv.errors[fi,ci] = mean((Wage[folds==fi,]$wage - y_hat^2))
  }
}
```

```r
cv.errors.mean = apply(cv.errors, 2, mean)
cv.errors.stderr = apply(cv.errors, 2, sd) /sqrt(k)
```

```r
min.cv.index = which.min(cv.errors.mean)
one_se_up_value = (cv.errors.mean + cv.errors.stderr)[min.cv.index]
```

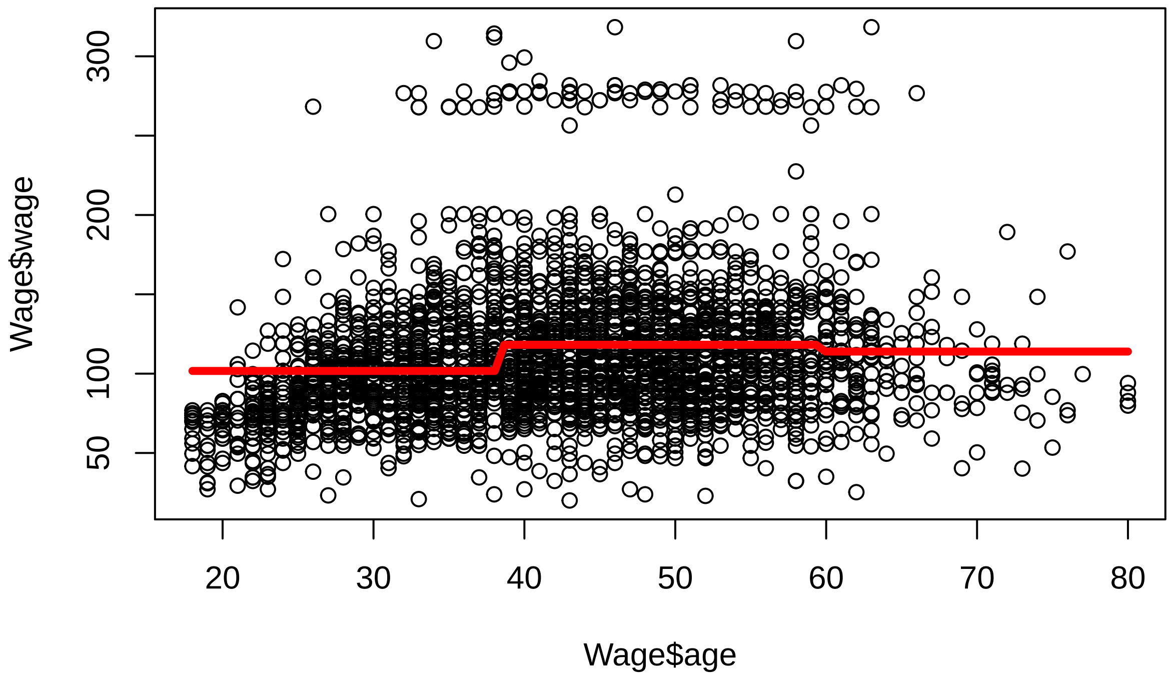
```r
# Set up the x-y limits for plotting
min_lim = min(one_se_up_value, cv.errors.mean, cv.errors.mean-cv.errors.stderr, cv.errors.mean+cv.errors

max_lim = max(one_se_up_value, cv.errors.mean, cv.errors.mean-cv.errors.stderr, cv.errors.mean+cv.errors
```

```r
plot(number_of_bins, cv.errors.mean, ylim=c(min_lim, max_lim), pch=19, type='b', xlab='number of cut bin
lines(number_of_bins, cv.errors.mean-cv.errors.stderr, lty='dashed')
lines(number_of_bins, cv.errors.mean-cv.errors.stderr, lty='dashed')
abline(h=one_se_up_value, col='red')
grid()
```




```r
# Fit the optimal model using all data
nob = 3
fit = glm(wage ~ cut(age, breaks = seq(from=age_range[1], to=age_range[2], length.out=(nob+1))), data=Wa
```

```r
plot(Wage$age, Wage$wage)

aRng = range(Wage$age)

a_predict = seq(from=aRng[1], to=aRng[2], length.out=100)
w_predict = predict(fit, newdata=list(age=a_predict))
lines(a_predict, w_predict, col='red', lw=4)
```



Question Nine

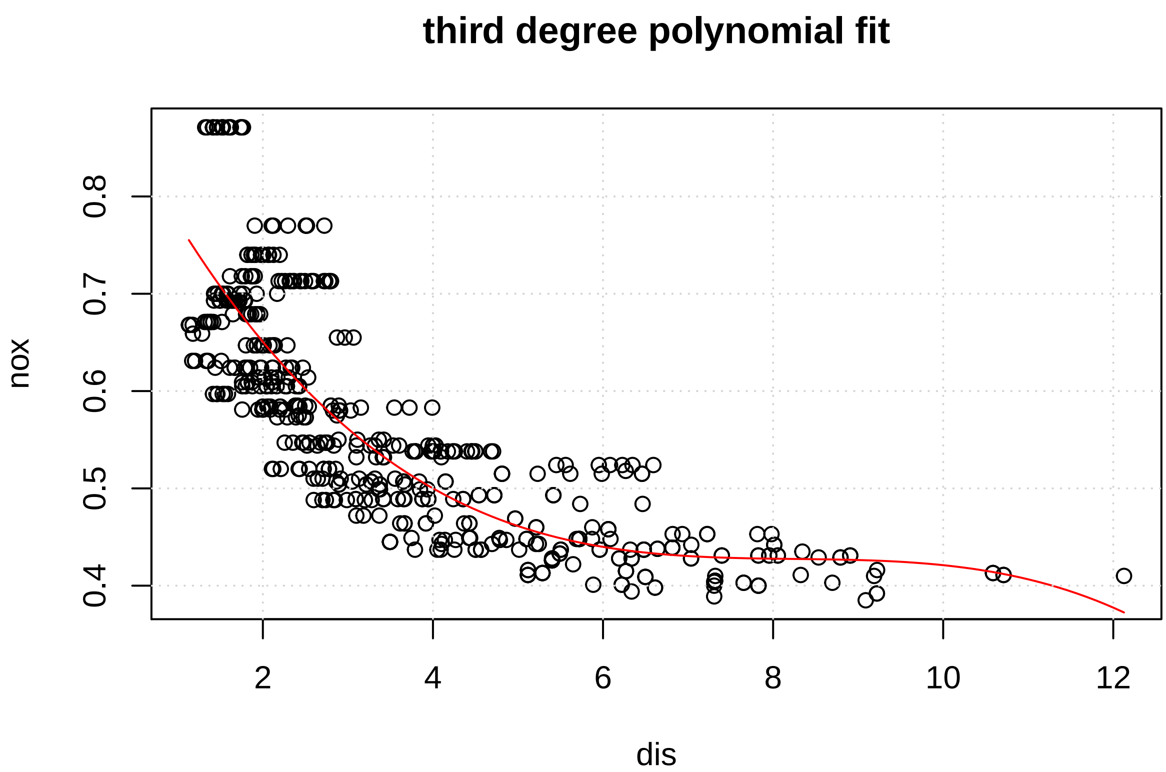
This question uses the variable `dis` (the weighted mean of distance to five Boston employment centers) and `nox` (nitrogen oxides concentration in parts per 10 million) from the `Boston` data. We will treat `dis` as the predictor and `nox` as the response.

A. Use the `poly()` function to fit a cubic polynomial regression to predict `nox` and `dis`. Report the regression output, and plot the resulting data and polynomial fits.

```
library(MASS)
set.seed(0)
```

```
m = lm(nox ~ poly(dis,3), data=Boston)
```

```
plot(Boston$dis, Boston$nox, xlab='dis', ylab= 'nox', main='third degree polynomial fit')

dis_range = range(Boston$dis)
dis_samples = seq(from=dis_range[1], to=dis_range[2], length.out=100)
y_hat = predict(m, newdata=list(dis=dis_samples))

lines(dis_samples, y_hat, col='red')
grid()
```



B. Plot the polynomial fit for a range of different polynomial degrees (say, from 1 to 10), and report the associated residual sum of squares.

C. Perform cross-validation or another approach to select the optimal degree for the polynomial, and explain your results.

```
d_max = 10
```

```
# The training RSS:
training_rss = rep(NA,d_max)
for( d in 1:d_max ){
  m = lm( nox ~ poly(dis,d), data=Boston )
  training_rss[d] = sum( ( m$residuals )^2 )
}
```

```
# The RSS estimated using cross-valdiation:
k = 10
folds = sample( 1:k, nrow(Boston), replace=TRUE )
cv.rss.test = matrix( NA, k, d_max )
cv.rss.train = matrix( NA, k, d_max )
```

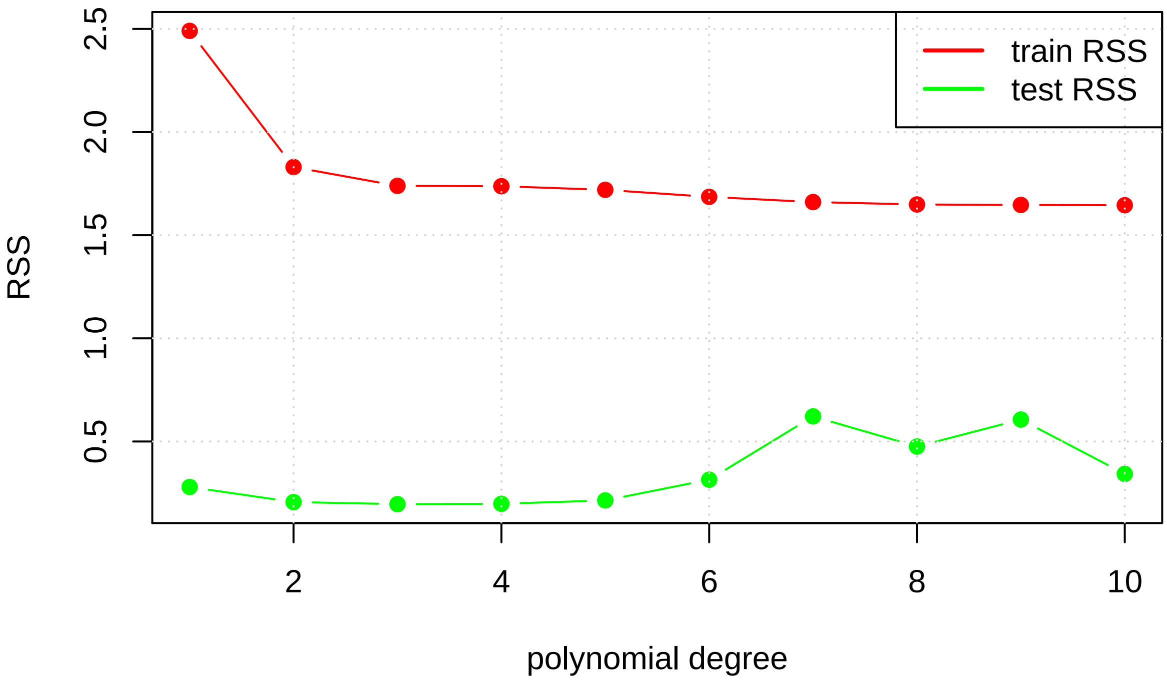
```
for( d in 1:d_max ){
  for( fi in 1:k ){ # for each fold
    fit = lm( nox ~ poly(dis,d), data=Boston[folds!=fi,] )

    y_hat = predict( fit, newdata=Boston[folds!=fi,] )
    cv.rss.train[fi,d] = sum( ( Boston[folds!=fi,]$nox - y_hat )^2 )

    y_hat = predict( fit, newdata=Boston[folds==fi,] )
    cv.rss.test[fi,d] = sum( ( Boston[folds==fi,]$nox - y_hat )^2 )
  }
}
```

```
cv.rss.train.mean = apply(cv.rss.train,2,mean)
cv.rss.train.stderr = apply(cv.rss.train,2,sd)/sqrt(k)
```

```
cv.rss.test.mean = apply(cv.rss.test,2,mean)
cv.rss.test.stderr = apply(cv.rss.test,2,sd)/sqrt(k)
```

```
min_value = min( c(cv.rss.test.mean,cv.rss.train.mean) )
max_value = max( c(cv.rss.test.mean,cv.rss.train.mean) )
```

```
plot( 1:d_max, cv.rss.train.mean, xlab='polynomial degree', ylab='RSS', col='red', pch=19, type='b', yl
lines( 1:d_max, cv.rss.test.mean, col='green', pch=19, type='b' )
grid()
legend( "topright", legend=c("train RSS","test RSS"), col=c("red","green"), lty=1, lwd=2 )
```



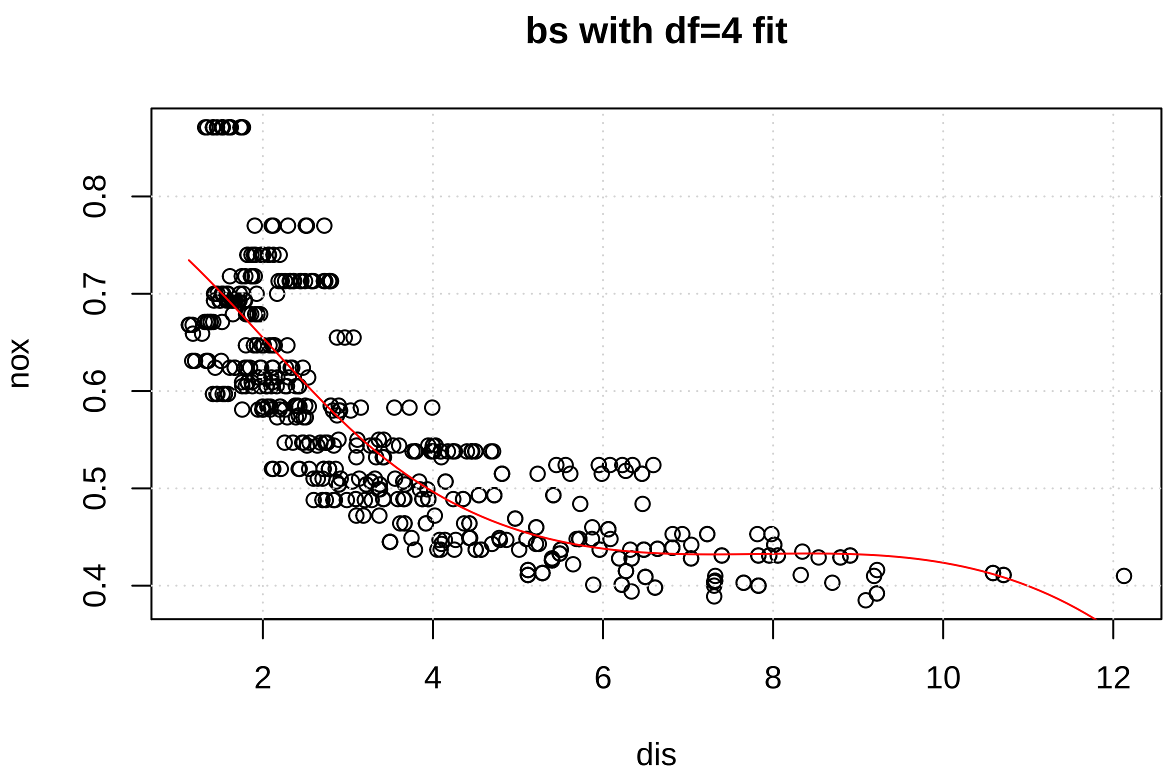
D. Use the `bs()` function to fit a regression spline to predict `nox` using `dis`. Report the output for the fit using four degrees of freedom. How did you choose the knots? Plot the resulting fit.

E. Now fit a regression spline for a range of degrees of freedom, and plot the resulting fits and report the resulting RSS. Describe the results obtained.

F. Perform cross-validation or another approach in order to select the best degrees of freedom for a regression spline on this data. Describe your results.

```
m = lm(nox ~ bs(dis, df=4), data=Boston)
```

```
plot(Boston$dis, Boston$nox, xlab='dis', ylab='nox', main='bs with df=4 fit')

dis_range = range(Boston$dis)
dis_samples = seq(from=dis_range[1], to=dis_range[2], length.out=100)
y_hat = predict(m, newdata=list(dis=dis_samples))

lines(dis_samples, y_hat, col='red')
grid()
```




```
dof_choices = c(3, 4, 5, 10, 15, 20)
n_dof_choices = length(dof_choices)
```

```
# The RSS estimated using cross validation
k = 5
folds = sample(1:k, nrow(Boston), replace=TRUE)
cv.rss.test = matrix(NA, k, n_dof_choices)
cv.rss.train = matrix(NA, k, n_dof_choices)
```

```
for( di in 1:n_dof_choices ){
  for( fi in 1:k ){ # for each fold
    fit = lm( nox ~ bs(dis,df=dof_choices[di]), data=Boston[folds!=fi,] )

    y_hat = predict( fit, newdata=Boston[folds!=fi,] )
    cv.rss.train[fi,di] = sum( ( Boston[folds!=fi,]$nox - y_hat )^2 )

    y_hat = predict( fit, newdata=Boston[folds==fi,] )
    cv.rss.test[fi,di] = sum( ( Boston[folds==fi,]$nox - y_hat )^2 )
  }
}
```

```
## Warning in bs(dis, degree = 3L, knots = numeric(0), Boundary.knots = c(1.137, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`50%` = 3.1322), Boundary.knots =
## c(1.137, : some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`33.33333%` = 2.3887, `66.66667%` =
## 4.3549: some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`12.5%` = 1.7542375, `25%` =
## 2.100175, : some 'x' values beyond boundary knots may cause ill-conditioned
```

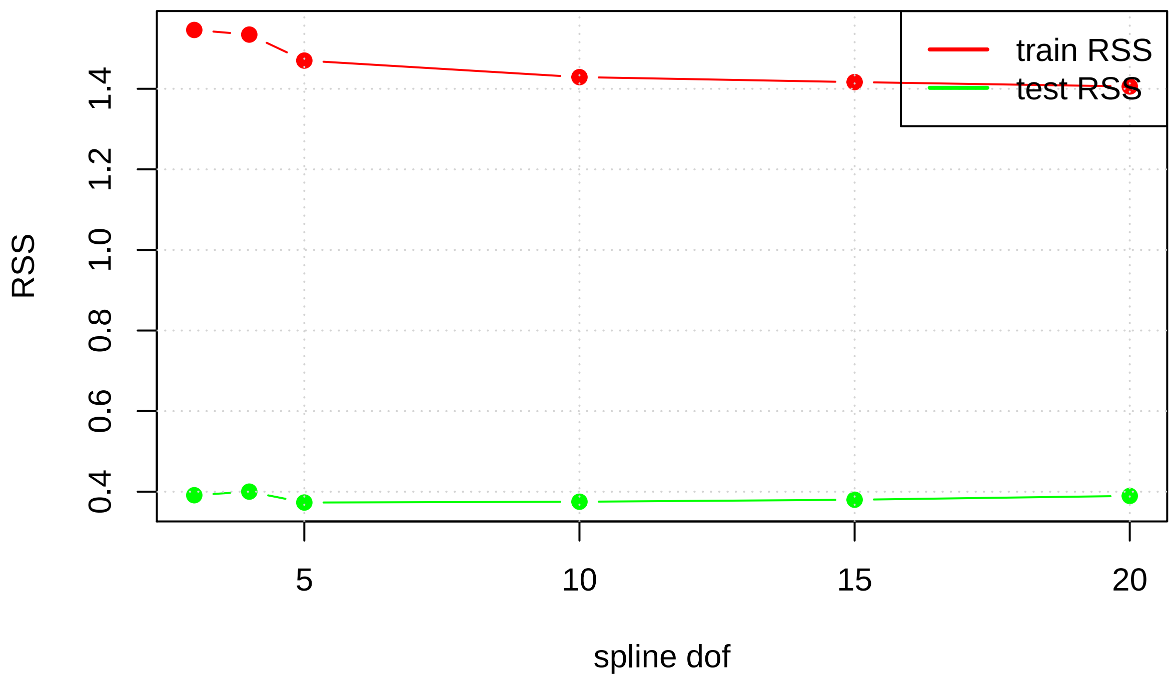
```
## bases

## Warning in bs(dis, degree = 3L, knots = c(`7.692308%` = 1.53333846153846, : some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`5.555556%` = 1.464, `11.11111%` =
## 1.6582, : some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
cv.rss.train.mean = apply(cv.rss.train,2,mean)
cv.rss.train.stderr = apply(cv.rss.train,2,sd)/sqrt(k)
```

```
cv.rss.test.mean = apply(cv.rss.test,2,mean)
cv.rss.test.stderr = apply(cv.rss.test,2,sd)/sqrt(k)
```

```
min_value = min( c(cv.rss.test.mean,cv.rss.train.mean) )
max_value = max( c(cv.rss.test.mean,cv.rss.train.mean) )
```

```
plot(dof_choices, cv.rss.train.mean, xlab='spline dof', ylab='RSS', col='red', pch=19, type='b', ylim=c
lines(dof_choices, cv.rss.test.mean, col='green', pch=19, type='b')
grid()
legend("topright", legend=c("train RSS", "test RSS"), col=c("red", "green"), lty=1, lwd=2)
```



Question Ten

This question relates to the `College` dataset.

A. Split the data into a training set and a test set. Using out-of-state tuition as the response and the other variables as predictors, perform forward stepwise selection on the training set in order to identify a satisfactory model that uses a subset of the predictors.

```
# install.packages("leaps")
library(leaps)
library(glmnet)
```

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

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

```r
set.seed(0)
```

```r
# Divide the dataset into three parts: training==1, validation==2, and test==3
dataset_part = sample(1:3, nrow(College), replace=T, prob=c(0.5, 0.25, 0.25))
```

```r
p = ncol(College)-1
```

```r
# Fit subsets of various sizes
regfit.forward = regsubsets(Outstate ~., data=College[dataset_part==1,], nvmax=p, method="forward")
print(summary(regfit.forward))
```

```
## Subset selection object
## Call: regsubsets.formula(Outstate ~ ., data = College[dataset_part ==
##     1, ], nvmax = p, method = "forward")
## 17 Variables  (and intercept)
##             Forced in Forced out
## PrivateYes      FALSE      FALSE
## Apps            FALSE      FALSE
## Accept          FALSE      FALSE
## Enroll          FALSE      FALSE
## Top10perc       FALSE      FALSE
## Top25perc       FALSE      FALSE
## F.Undergrad     FALSE      FALSE
## P.Undergrad     FALSE      FALSE
## Room.Board      FALSE      FALSE
## Books           FALSE      FALSE
## Personal        FALSE      FALSE
## PhD             FALSE      FALSE
## Terminal        FALSE      FALSE
## S.F.Ratio       FALSE      FALSE
## perc.alumni     FALSE      FALSE
## Expend          FALSE      FALSE
## Grad.Rate       FALSE      FALSE
## 1 subsets of each size up to 17
## Selection Algorithm: forward
##           PrivateYes Apps Accept Enroll Top10perc Top25perc F.Undergrad
## 1  ( 1 )  " "        " "  " "    " "    " "       " "       " "
## 2  ( 1 )  "*"        " "  " "    " "    " "       " "       " "
## 3  ( 1 )  "*"        " "  " "    " "    " "       " "       " "
## 4  ( 1 )  "*"        " "  " "    " "    " "       " "       " "
## 5  ( 1 )  "*"        " "  " "    " "    " "       " "       " "
## 6  ( 1 )  "*"        " "  " "    " "    " "       " "       " "
## 7  ( 1 )  "*"        " "  " "    " "    " "       " "       " "
## 8  ( 1 )  "*"        " "  " "    " "    " "       " "       " "
## 9  ( 1 )  "*"        " "  "*"    " "    " "       " "       " "
## 10  ( 1 ) "*"        " "  "*"    "*"    " "       " "       " "
## 11  ( 1 ) "*"        "*"  "*"    "*"    " "       " "       " "
## 12  ( 1 ) "*"        "*"  "*"    "*"    "*"       " "       " "
## 13  ( 1 ) "*"        "*"  "*"    "*"    "*"       "*"       " "
## 14  ( 1 ) "*"        "*"  "*"    "*"    "*"       "*"       " "
## 15  ( 1 ) "*"        "*"  "*"    "*"    "*"       "*"       "*"
## 16  ( 1 ) "*"        "*"  "*"    "*"    "*"       "*"       "*"
## 17  ( 1 ) "*"        "*"  "*"    "*"    "*"       "*"       "*"
##           P.Undergrad Room.Board Books Personal PhD Terminal S.F.Ratio
```

```
## 1  ( 1 )  " "         " "        " "   " "      " "  " "      " "
## 2  ( 1 )  " "         " "        " "   " "      " "  " "      " "
## 3  ( 1 )  " "         "*"        " "   " "      " "  " "      " "
## 4  ( 1 )  " "         "*"        " "   " "      " "  " "      " "
## 5  ( 1 )  " "         "*"        " "   " "      " "  "*"      " "
## 6  ( 1 )  " "         "*"        " "   " "      " "  "*"      " "
## 7  ( 1 )  " "         "*"        " "   "*"      " "  "*"      " "
## 8  ( 1 )  " "         "*"        " "   "*"      " "  "*"      "*"
## 9  ( 1 )  " "         "*"        " "   "*"      " "  "*"      "*"
## 10  ( 1 ) " "         "*"        " "   "*"      " "  "*"      "*"
## 11  ( 1 ) " "         "*"        " "   "*"      " "  "*"      "*"
## 12  ( 1 ) " "         "*"        " "   "*"      " "  "*"      "*"
## 13  ( 1 ) " "         "*"        " "   "*"      " "  "*"      "*"
## 14  ( 1 ) "*"         "*"        " "   "*"      " "  "*"      "*"
## 15  ( 1 ) "*"         "*"        " "   "*"      " "  "*"      "*"
## 16  ( 1 ) "*"         "*"        "*"   "*"      " "  "*"      "*"
## 17  ( 1 ) "*"         "*"        "*"   "*"      "*"  "*"      "*"
##           perc.alumni Expend Grad.Rate
## 1  ( 1 )  " "         "*"    " "
## 2  ( 1 )  " "         "*"    " "
## 3  ( 1 )  " "         "*"    " "
## 4  ( 1 )  " "         "*"    "*"
## 5  ( 1 )  " "         "*"    "*"
## 6  ( 1 )  "*"         "*"    "*"
## 7  ( 1 )  "*"         "*"    "*"
## 8  ( 1 )  "*"         "*"    "*"
## 9  ( 1 )  "*"         "*"    "*"
## 10  ( 1 ) "*"         "*"    "*"
## 11  ( 1 ) "*"         "*"    "*"
## 12  ( 1 ) "*"         "*"    "*"
## 13  ( 1 ) "*"         "*"    "*"
## 14  ( 1 ) "*"         "*"    "*"
## 15  ( 1 ) "*"         "*"    "*"
## 16  ( 1 ) "*"         "*"    "*"
## 17  ( 1 ) "*"         "*"    "*"
```

```r
reg.summary = summary(regfit.forward)
```

```r
# Test the trained models on the validation set
validation.mat = model.matrix(Outstate ~ ., data=College[dataset_part==2,])
val.errors = rep(NA,p)
for (ii in 1:p){
  coefi = coef(regfit.forward, id=ii)
  pred=validation.mat[,names(coefi)] %*% coefi
  val.errors[ii] = mean((College$Outstate[dataset_part==2] - pred)^2)
}
```

```r
print("forward selection validation errors")
```

```
## [1] "forward selection validation errors"
```

```r
print(val.errors)
```

```
##  [1] 9662181 7319698 5595207 4932427 4516258 4239392 4238819 4186692 4142959
## [10] 4558914 4337677 4181877 4199016 4204663 4204463 4200772 4207920
```

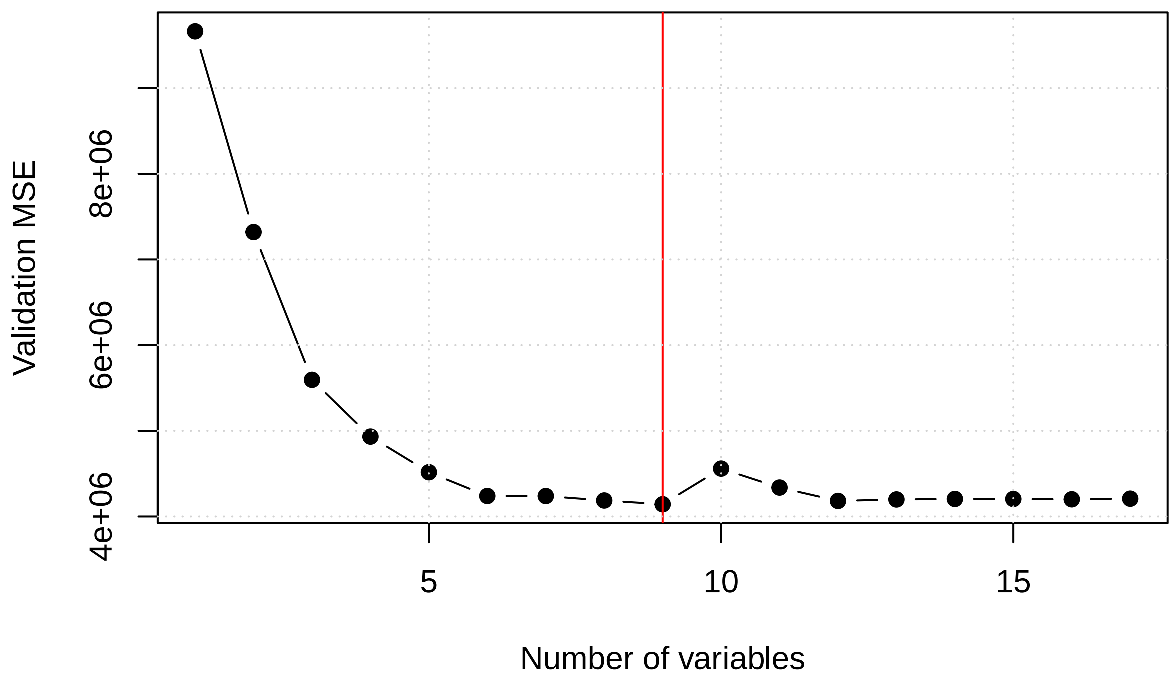
```r
k = which.min(val.errors)
print(sprintf("smallest validation error for the index = %d, with coefficients given by", k))
```

```
## [1] "smallest validation error for the index = 9, with coefficients given by"
```

```r
print(coef(regfit.forward, id=k))
```

```
##   (Intercept)    PrivateYes        Accept    Room.Board      Personal
## -3.423831e+03  2.852109e+03  7.135054e-02  9.408533e-01 -4.539620e-01
##      Terminal     S.F.Ratio   perc.alumni        Expend     Grad.Rate
##  4.480521e+01 -4.828169e+01  3.085511e+01  2.249079e-01  3.535524e+01
```

```r
plot(val.errors, xlab="Number of variables", ylab="Validation MSE", pch=19, type='b')
abline(v=k, col='red')
grid()
```




```r
# Predict the best model found on the testing set
test.mat = model.matrix(Outstate ~., data=College[dataset_part==3,])
coefi = coef(regfit.forward, id=k)
pred = test.mat[,names(coefi)] %*% coefi
test.error = mean((College$Outstate[dataset_part==3] - pred)^2)
print("test error on the optimal subset")
```

```
## [1] "test error on the optimal subset"
```

```r
print(test.error)
```

```
## [1] 4427377
```

```r
k = 3
coefi = coef(regfit.forward, id=k)
pred = test.mat[,names(coefi)] %*% coefi
test.error = mean((College$Outstate[dataset_part==3] - pred)^2)
print("test erro on the k=3 subset")
```

```
## [1] "test erro on the k=3 subset"
```

```r
print(test.error)
```

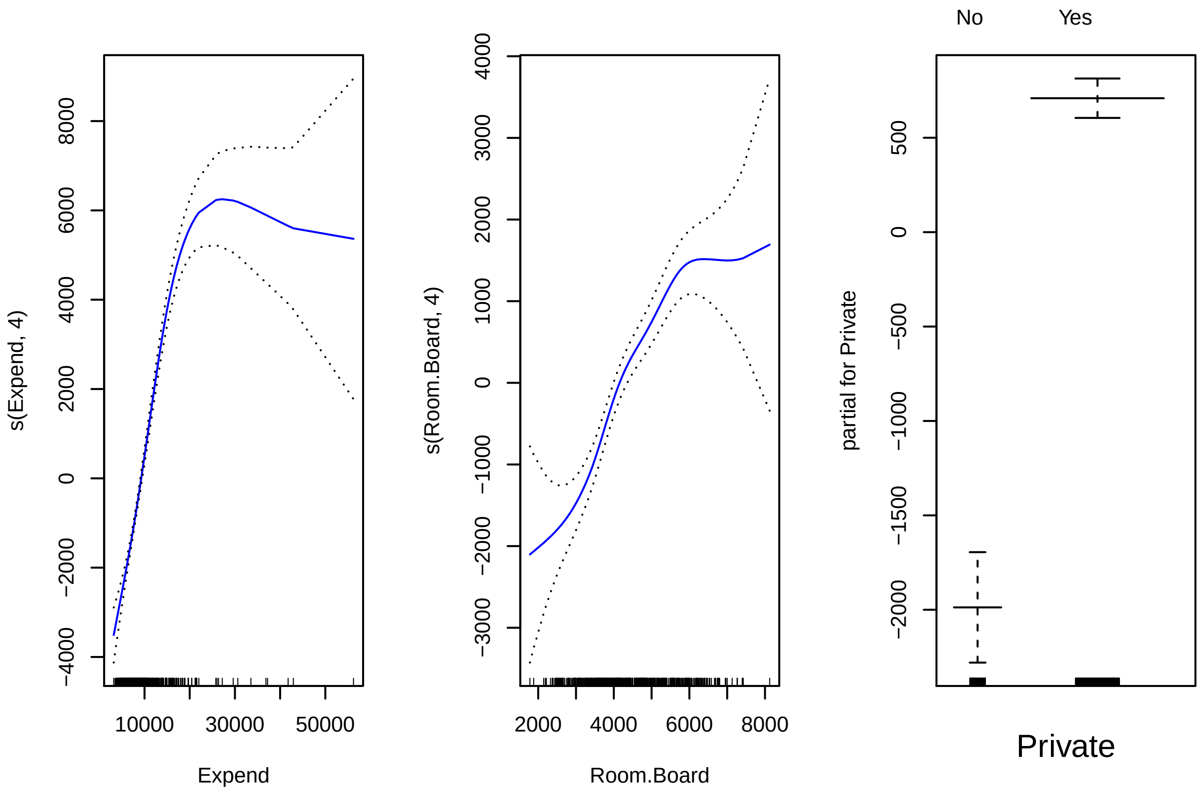
```
## [1] 5097180
```

B. Fit a GAM on the training data using out-of-state tuition as a response and the features selected in the previous step as the predictors. Plot the results and explain your findings.

```r
# Combine the training and validation into one "training" dataset
dataset_part[dataset_part==2] = 1
dataset_part[dataset_part==3] = 2
```

```r
gam.model = gam(Outstate ~ s(Expend,4) + s(Room.Board,4) + Private, data=College[dataset_part==1,])
```

```r
par(mfrow=c(1,3))
plot(gam.model, se=TRUE, col='blue')
```




```r
par(mgrow=c(1,1))
```

```
## Warning in par(mgrow = c(1, 1)): "mgrow" is not a graphical parameter
```

```r
# Predict the GAM performance on the test dataset
y_hat = predict(gam.model, newdata=College[dataset_part==2,])
MSE = mean((College[dataset_part==2,]$Outstate - y_hat)^2)
print("gam testing set (MSE) error")
```

```
## [1] "gam testing set (MSE) error"
```

```r
print(MSE)
```

```
## [1] 4570160
```

Question Eleven

GAMs are generally fit using a *backfitting* approach. The idea behind backfitting is actually quite simple. We will now explore backfitting in the context of multiple linear regression.

Suppose that we would like to perform multiple linear regression, but we do not have the software to do so. Instead, we only have software to perform simple linear regression. Therefore, we take the following iterative approach: we repeatedly hold all but one coefficient estimate fixed as its current value, and update only the coefficient estimate using a simple linear regression. The process is continued until *convergence* - that is, until the coefficient estimates stop changing.

We will now try this out on a toy example.

A. Generate a response Y and two predictors X_1 and X_2, with n = 100.

```
n = 100

X1 = rnorm(n)
X2  = rnorm(n)
```

```
# The true values of beta_i:
beta_0 = 3.0
beta_1 = 5.0
beta_2 = -0.2
```

```
Y = beta_0 + beta_1 * X1 + beta_2 * X2 + 0.1 * rnorm(n)
```

B. Initiative $\hat{\beta}_1$ to take on a value of your choice. It does not matter what value you choose.

```
beta_1_hat = -3.0
```

C. Keep $\hat{\beta}_1$ fixed, fit the model

$Y - \hat{\beta}_1 X_1 = \beta_0 + \beta_2 X_2 + \epsilon.$

D. Keeping $\hat{\beta}_2$ fixed, fit the model:

$Y - \hat{\beta}_2 X_2 = \beta_0 + \beta_1 X_1 + \epsilon.$

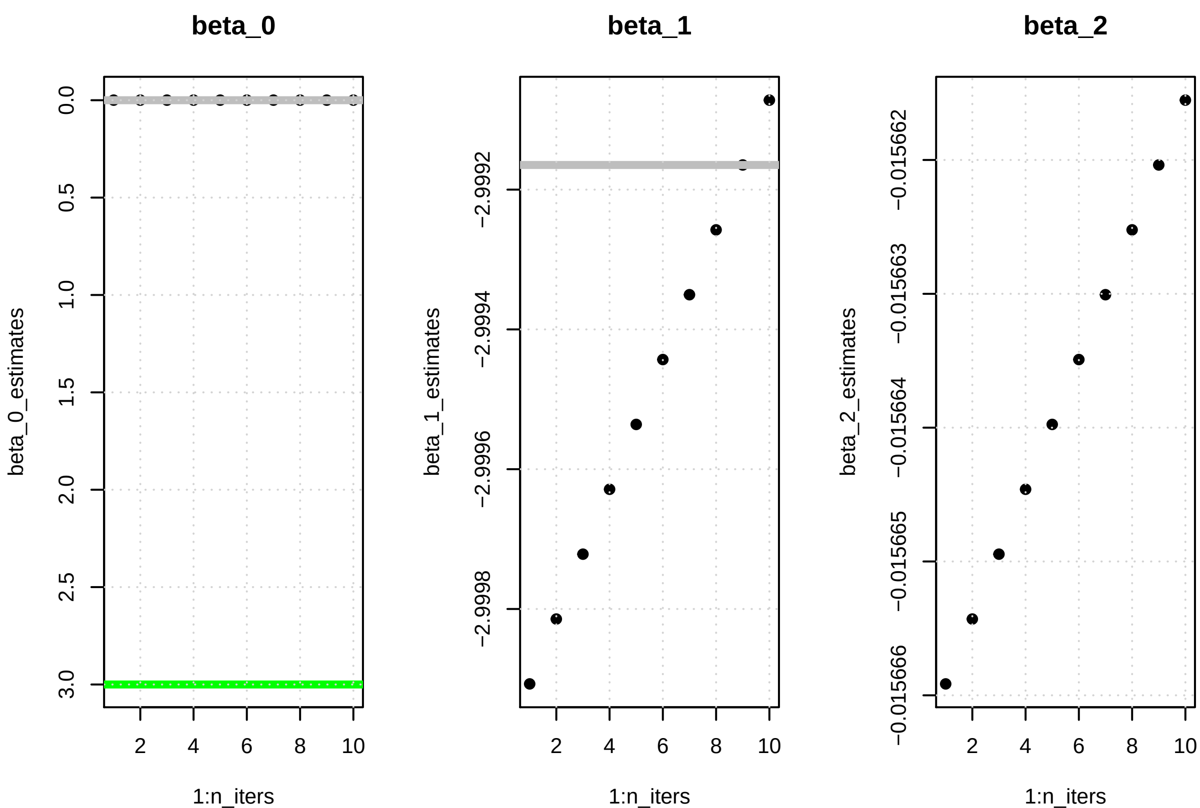
E. Write a for loop to repeat (C) and (D) 1,000 times. Report the estimates of $\hat{\beta}_0$, $\hat{\beta}_1$ and $\hat{\beta}_2$ at each iteration of the for loop. Create a plot in which each of these values is displayed, with $\hat{\beta}_0$, $\hat{\beta}_1$ and $\hat{\beta}_2$ each shown in a different colour.

```
n_iters = 10

beta_0_estimates = c()
beta_1_estimates = c()
beta_2_estimates = c()

for(ii in 1:n_iters){
  a = Y = beta_1_hat * X1
  beta_2_hat = lm(a ~ X2)$coef[2]

  a = Y - beta_2_hat * X2
  m = lm(a ~ X1)
  beta_1_hat = m$coef[2]

  beta_0_hat = m$coef[1]

  beta_0_estimates = c(beta_0_estimates, beta_0_hat)
```

```
  beta_1_estimates = c(beta_1_estimates, beta_1_hat)
  beta_2_estimates = c(beta_2_estimates, beta_2_hat)
}
```

```
# Get the coefficient estimates using lm
m = lm(Y ~ X1 + X2)
```

```
old_par = par(mfrow = c(1,3))

plot(1:n_iters, beta_0_estimates, main='beta_0', pch=19, ylim=c(beta_0*0.999, max(beta_0_estimates)))
abline(h=beta_0, col='green', lwd=4)
abline(h=m$coefficients[1], col='gray', lwd=4)
grid()

plot(1:n_iters, beta_1_estimates, main='beta_1', pch=19)
abline(h=beta_1, col='green', lwd=4)
abline(h=m$coefficients[2], col='gray', lwd=4)
grid()

plot(1:n_iters, beta_2_estimates, main='beta_2', pch=19)
abline(h=beta_2, col='green', lwd=4)
abline(h=m$coefficients[3], col='gray', lwd=4)
grid()
```




```
par(old_par)
```

Question Twelve

The problem is a continuation of the previous excercise. In a toy example with p = 100, show that one can approximate the multiple linear regression coefficient estimates by repeatedly performing simple linear regression in a backfitting procedure. How many backfitting iterations are required in order to obtain a good approximation to the multiple regression coefficient estimates? Create a plot to justify your answer.

```
p = 100
n = 100
```

```
# Generate some regression coefficients beta_0, beta_1,..., beta_p
beta_truth = rnorm( p+1 )
```

```
# Generate some data (append a column of ones)
Xs = c( rep(1,n), rnorm( n*p ) )
X = matrix( data=Xs, nrow=n, ncol=(p+1), byrow=FALSE )
```

```
# Produce the response
Y = X %*% beta_truth + 0.1 * rnorm(n)
```

```
# Get the true estimated coefficient estimate using lm
m = lm( Y ~ X - 1 )
beta_lm = m$coeff
```

```
# Estimate beta_i using backfitting
beta_hat = rnorm( p+1 ) # initial estimate of beta's is taken to be random
 # Initial estimate of beta's is taken to be random
```

```
n_iters = 10
```

```
beta_estimates = matrix( data=rep(NA,n_iters*(p+1)), nrow=n_iters, ncol=(p+1) )
beta_differences_with_truth = rep(NA,n_iters)
beta_differences_with_LS = rep(NA,n_iters)
for( ii in 1:n_iters ){
  for( pi in 0:p ){ # for beta_0, beta_1, ... beta_pi ... beta_p

    # Perform simple linear regression on the variable X_pi (assuming we know all other values of beta_
    #
    a = Y - X[,-(pi+1)] %*% beta_hat[-(pi+1)] # remove all predictors except beta_0

    if( pi==0 ){
      m = lm( a ~ 1 ) # estimate a constant
      beta_hat[pi+1] = m$coef[1]
    }else{
      m = lm( a ~ X[,pi+1] ) # estimate the slope on X_pi
      beta_hat[pi+1] = m$coef[2]
    }

  }
  beta_estimates[ii,] = beta_hat
  beta_differences_with_truth[ii] = sqrt( sum( ( beta_hat - beta_truth )^2 ) )
  beta_differences_with_LS[ii] = sqrt( sum( ( beta_hat - beta_lm )^2 ) )
}
```