Lab 7 - Non-Linear Modeling

An Introduction to Statistical Learning

We will be using the `Wage` dataset

```
pacman::p_load(ISLR, splines)
attach(Wage)
```

1. Polynomial Regression

Fitting the model

Our goal is to produce two plots, one showing `wage` vs `age` and other showing `wage>250` vs `age`.

We start by fitting a linear model with powers of `age`:

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

Making predictions

We create a grid for `age` in which we will make predictions:

```
agelims = range(age)
age.grid = seq(from=agelims[1], to=agelims[2])
```

Now we make the predictions and compute standard errors:

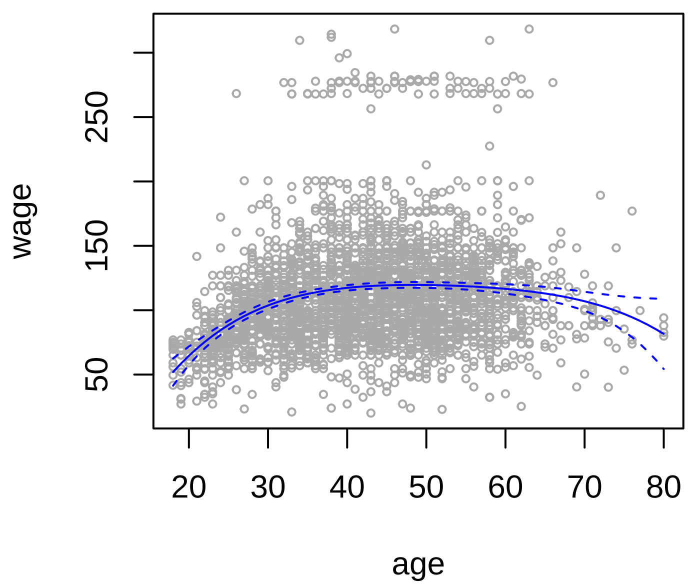
```
poly.pred = predict(poly.fit, newdata = list(age = age.grid), se.fit = TRUE)
```

We create a matrix containing the standard error intervals:

```
se.bands = cbind(poly.pred$fit - 2*poly.pred$se.fit,
                 poly.pred$fit + 2*poly.pred$se.fit)
```

We now produce the first plot, showing `wage` vs `age` with a confidence interval of 95%:

```
plot(age, wage, xlim = agelims, cex=.5, col='darkgrey')
lines(age.grid, poly.pred$fit, lwd = 1, col = 'blue')
matlines(age.grid, se.bands, lwd = 1, col = 'blue', lty = 2)
```



Creating the second plot requires a little more work. We start by selecting the degree for the polynomial on `age`. To do that we will use `anova()`:

```
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 output of the ANOVA analysis shows that the p-value comparing `fit.1` and `fit.2` is essentially 0, which means that considering both models are equivalent (H_0), the probability of obtaining the performance difference between them that ANOVA found is almost 0, so `fit.2` is better than `fit.1`. The same happens for `fit.3` and arguibly for `fit.4`.

We will use a 4-degree polynomial to fit the data to a *qualitative* model with `glm()`, in which the target is the probability of `wage>250`, so we need `family=binomial`:

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

Now we make predictions for the age grid:

```
poly.4.pred = predict(poly.4.fit, newdata = list(age = age.grid), se.fit = TRUE)
```

The default prediction type is `link`, which for a default `binomial` model are *log-odds*, probabilities on *logit* scale:

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

Using `type="response"`, which gives the actual predicted probabilities, seems the correct choice here, but the confidence intervals we obtained this way would have negative values.

We need to convert the *logit* probabilities to actual probabilities:

$$p(Y=1 \mid X) = \frac{\exp{(X\beta)}}{1+\exp{(X\beta)}}$$

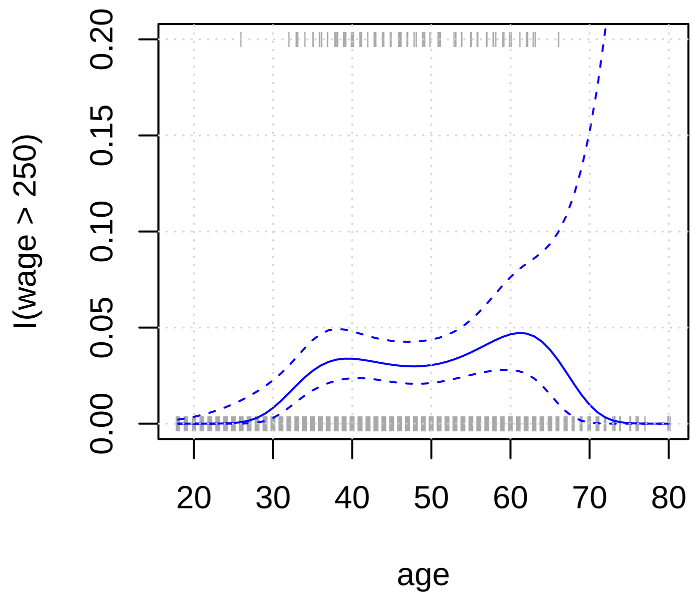
```
pfit = exp(poly.4.pred$fit) / (1 + exp(poly.4.pred$fit))
```

And for the confidence interval:

```
se.bands.logit = cbind(poly.4.pred$fit - 2 * poly.4.pred$se.fit,
                       poly.4.pred$fit + 2 * poly.4.pred$se.fit)
se.bands = exp(se.bands.logit) / (1 + exp(se.bands.logit))
```

We can now create the second plot:

```
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 = 1, col = 'blue')
matlines(age.grid, se.bands, lwd = 1, lty = 2, col = 'blue')
grid()
```



2. Step Functions

The `cut()` function divides the range of the data into intervals and assigns each data point to one of those intervals, returning an ordered *categorical* variable:

```
cut(age, 4)[1:5]
```

```
## [1] (17.9,33.5] (17.9,33.5] (33.5,49]   (33.5,49]   (49,64.5]
## Levels: (17.9,33.5] (33.5,49] (49,64.5] (64.5,80.1]
```

Labels can be passed to `cut()` to name the different levels or intervals.

To get the total count of observations for each interval `table()` is used:

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

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

We can fit a linear model using these levels that creates *dummy* variables:

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

3. Splines

B-Splines

The `bs()` method, included in the `splines` library, creates an entire matrix of basis functions for splines with the given set of knots. In this case we will specify fixed knots for 3 values of `age`: 25, 40 and 60:

```
bs(age, knots=c(25, 40, 60))[1:6,]
```

```
##              1           2          3         4          5 6
## [1,] 0.0000000 0.000000000 0.00000000 0.0000000 0.00000000 0
## [2,] 0.5599113 0.403778040 0.03339518 0.0000000 0.00000000 0
## [3,] 0.0000000 0.114795918 0.61856447 0.2627334 0.00390625 0
## [4,] 0.0000000 0.167108844 0.63316713 0.1988803 0.00084375 0
## [5,] 0.0000000 0.034013605 0.50819419 0.4265422 0.03125000 0
## [6,] 0.0000000 0.007346939 0.37667904 0.5302240 0.08575000 0
```

Fitting the model

A linear model is fitted using these basis expansions as predictors:

```
spline.fit = lm(wage ~ bs(age, knots = c(25, 40, 60)), data = Wage)
coef(summary(spline.fit))
```

```
##                                 Estimate Std. Error   t value     Pr(>|t|)
## (Intercept)                     60.49371   9.460394 6.3944180 1.863268e-10
## bs(age, knots = c(25, 40, 60))1  3.98050  12.537648 0.3174838 7.508987e-01
## bs(age, knots = c(25, 40, 60))2 44.63098   9.626267 4.6363744 3.697569e-06
## bs(age, knots = c(25, 40, 60))3 62.83879  10.755235 5.8426233 5.691460e-09
## bs(age, knots = c(25, 40, 60))4 55.99083  10.706284 5.2297167 1.814593e-07
## bs(age, knots = c(25, 40, 60))5 50.68810  14.401846 3.5195557 4.387142e-04
## bs(age, knots = c(25, 40, 60))6 16.60614  19.126431 0.8682300 3.853380e-01
```

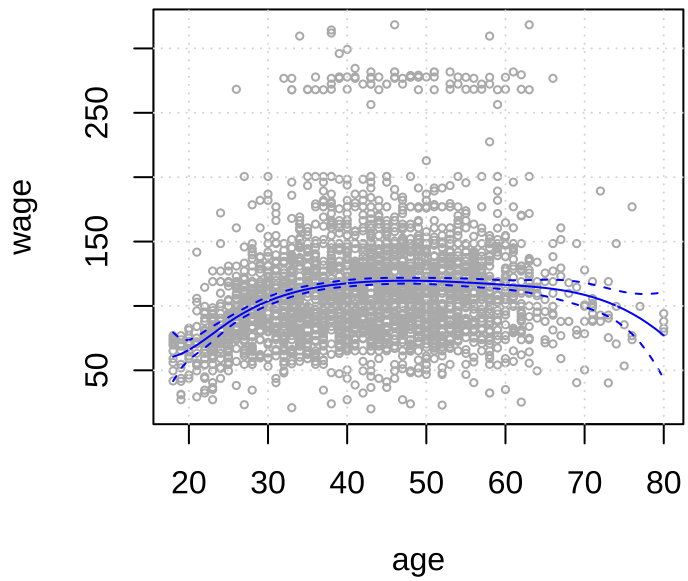
Degrees of freedom can be specified instead of knots, using `df`. This generates a spline with wknots at uniform quantiles of the data.

Making predictions

```
spline.pred = predict(spline.fit, newdata = list(age = age.grid), se.fit = TRUE)
```

Let's plot the results:

```
lwd = 1.0
plot(age, wage, col='darkgray', cex=.5)
lines(age.grid, spline.pred$fit, lwd=lwd, col='blue')
lines(age.grid, spline.pred$fit - 2*spline.pred$se.fit,
      lty = 'dashed', col = 'blue')
lines(age.grid, spline.pred$fit + 2*spline.pred$se.fit,
      lty = 'dashed', col = 'blue')
grid()
```



Natural Splines

To fit a *natural spline* the `ns()` function is used:

```
nat.fit = lm(wage ~ ns(age, knots = c(25, 40, 60)), data = Wage)
nat.pred = predict(nat.fit, newdata = list(age = age.grid), se.fit = TRUE)
```

Smoothing Splines

The `smooth.spline()` method is used. The syntax is different than before.

The number of degrees of freedom can be specified using `df`, or the built-in *LOOCV* method can be used to select the best value for `df`:

```
smooth.fit = smooth.spline(age, wage, cv=TRUE)
```

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

There's no need to make predictions when using a smoothing spline, as they are already computed in the `y` component of the fitted spline.

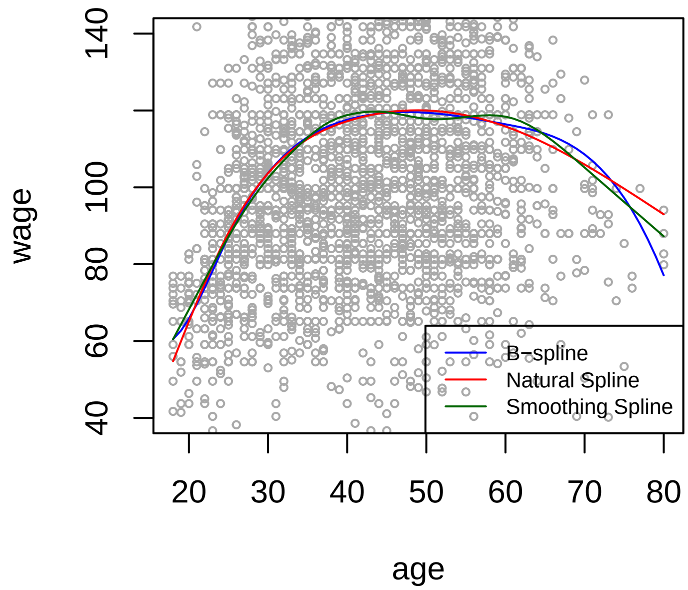
Comparing the results

Let's plot all the splines together:

```
lwd = 1.0
plot(age, wage, col='darkgray', cex=.5, ylim = c(40, 140))
# B-Spline
lines(age.grid, spline.pred$fit, lwd=lwd, col='blue')
# Natural Spline
lines(age.grid, nat.pred$fit, col='red', lwd=lwd)
```

```r
# Smoothing Spline
lines(smooth.fit, col='darkgreen', lwd=lwd)

legend('bottomright',
       legend = c('B-spline', 'Natural Spline', 'Smoothing Spline'),
       col = c('blue', 'red', 'darkgreen'),
       lty=1, cex=.7)
```



4. Local Regression

The `loess()` function (included in the `stats` library) performs local regression.

Fitting the data

Let's fit two local models, with `span` values of 0.2 and 0.5; this means that each neighborhood consists of 20% and 50% of the observations:

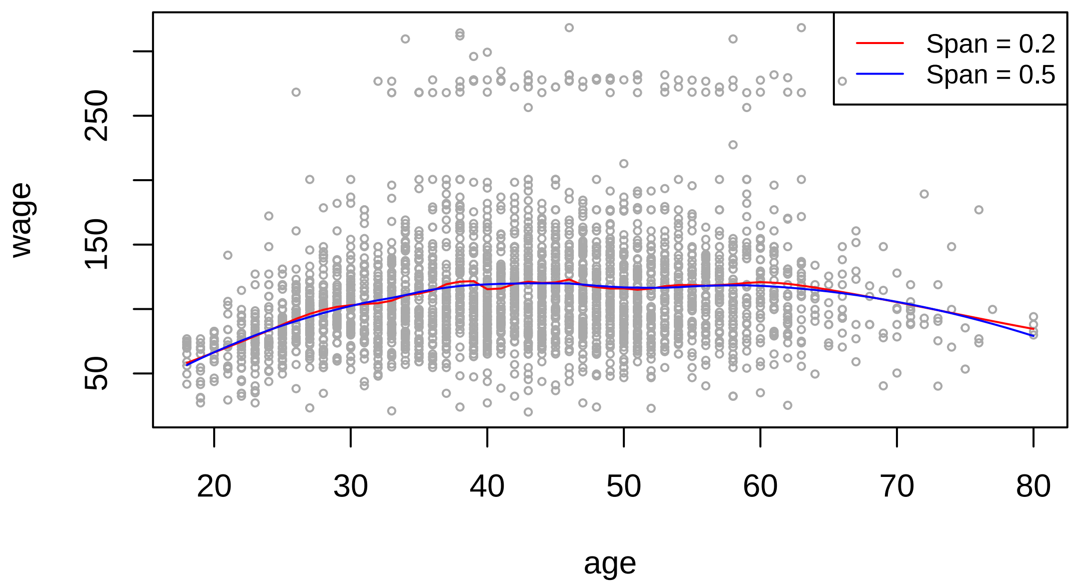
```r
local.fit1 = loess(wage ~ age, data = Wage, span = 0.2)
local.fit2 = loess(wage ~ age, data = Wage, span = 0.5)
```

Making predictions

```r
local.pred1 = predict(local.fit1, newdata = data.frame(age=age.grid))
local.pred2 = predict(local.fit2, newdata = data.frame(age=age.grid))
```

Plotting the results

```
plot(age, wage, xlim = agelims, cex = .5, col = 'darkgrey')
lines(age.grid, local.pred1, col = 'red', lwd = 1)
lines(age.grid, local.pred2, col = 'blue', lwd = 1)
legend('topright',
       legend = c('Span = 0.2', 'Span = 0.5'),
       col = c('red', 'blue'),
       lty = 1, lwd = 1, cex = 0.8)
```



The `locfit` library can also be used for fitting local regression models.

4. Generalized Additive Models

GAMs are linear regression models using an appropiate choice of basis functions, so they can be fitted using `lm()`. Here a GAM is fitted using a natural spline with 4 dof for `year`, another natural spline wit h5 dof for `age` and the `education` variable as is, because it's a qualitative variable:

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

To fit more general GAMs, using smoothing splines and other components that can't be expressed in terms of basis functions, the `gam` library is used.

```
pacman::p_load(gam)
```

Fitting the model

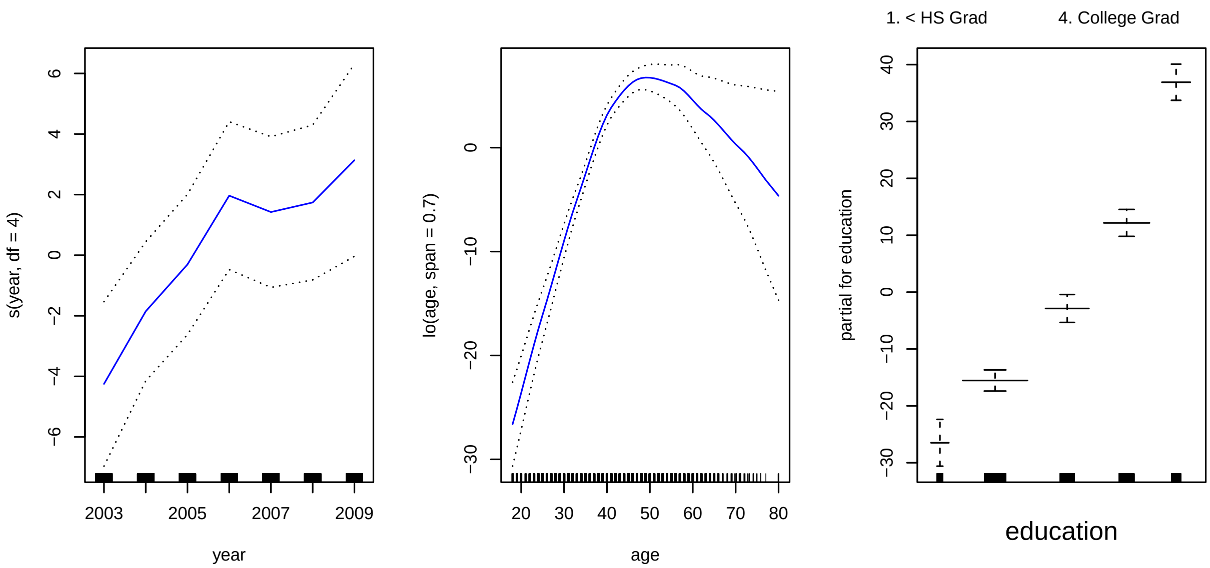
The `s()` function from `gam` is used to use smoothing splines, whereas `lo()` performs local regression. We create a smoothing spline for `year` with 4 dof and we use local regression for `age` with a `span` value of 0.7:

```
gam.fit = gam(wage ~ s(year, df = 4) + lo(age, span=.7) + education, data = Wage)
```

Plotting the fitted model

Results can be plotted:

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



Exploring linearity for year

In the plots the function of `year` shows some linearity. An ANOVA test can be used to determine which of the following models is best:

- A GAM that excludes `year`.
- A GAM that uses a linear function of `year`.
- A Gam that uses a spline function of `year` (our fitted model)

```
gam.mod.1 = gam(wage ~ lo(age, span = 0.7) + education, data = Wage)
gam.mod.2 = gam(wage ~ year + lo(age, span = 0.7) + education, data = Wage)
gam.mod.3 = gam.fit
```

Let's do the ANOVA test:

```
anova(gam.mod.1, gam.mod.2, gam.mod.3, test = 'F')
```

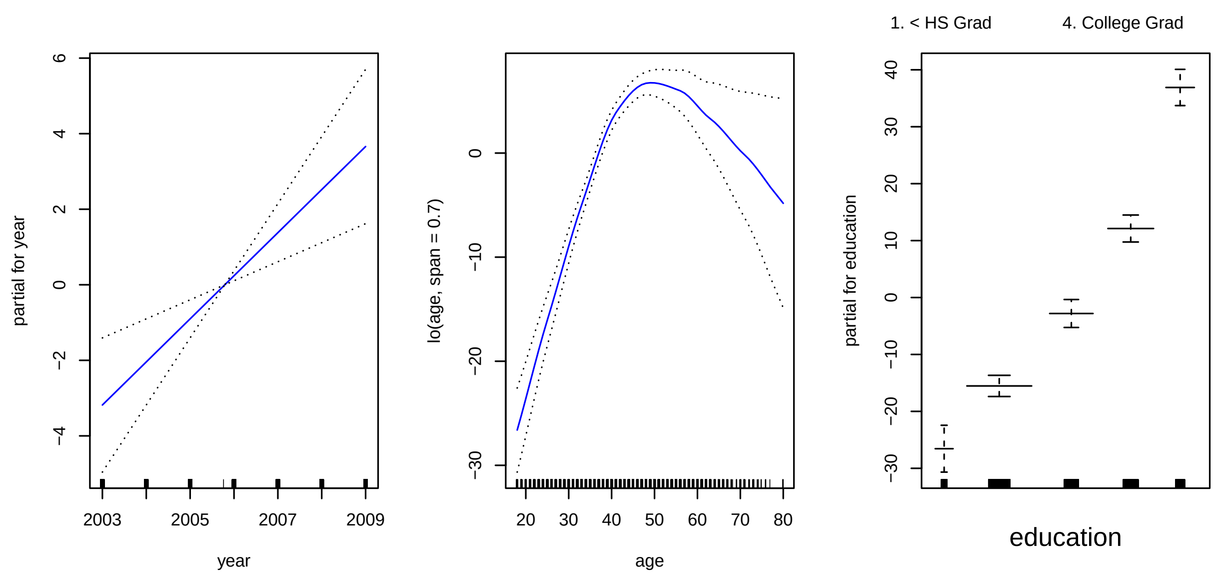
```
## Analysis of Deviance Table
##
## Model 1: wage ~ lo(age, span = 0.7) + education
## Model 2: wage ~ year + lo(age, span = 0.7) + education
## Model 3: wage ~ s(year, df = 4) + lo(age, span = 0.7) + education
##   Resid. Df Resid. Dev     Df Deviance       F    Pr(>F)
## 1    2992.8    3737372
## 2    2991.8    3720625 1.0000  16746.3 13.4667 0.0002471 ***
## 3    2988.8    3716672 2.9996   3953.2  1.0598 0.3649366
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The result shows evidence that a GAM with a linear function of `year` is better than a GAM that does not include `year` at all, but doesn't show evidence that a non-linear function of `year` is needed.

Let's plot the second model:

```
par(mfrow=c(1,3))
plot(gam.mod.2, se=TRUE, col='blue')
```



Creating interactions with `lo()`

`lo()` can be used to create interactions before calling `gam()`. Here a model is fitted creating an interaction between `year` and `age`:

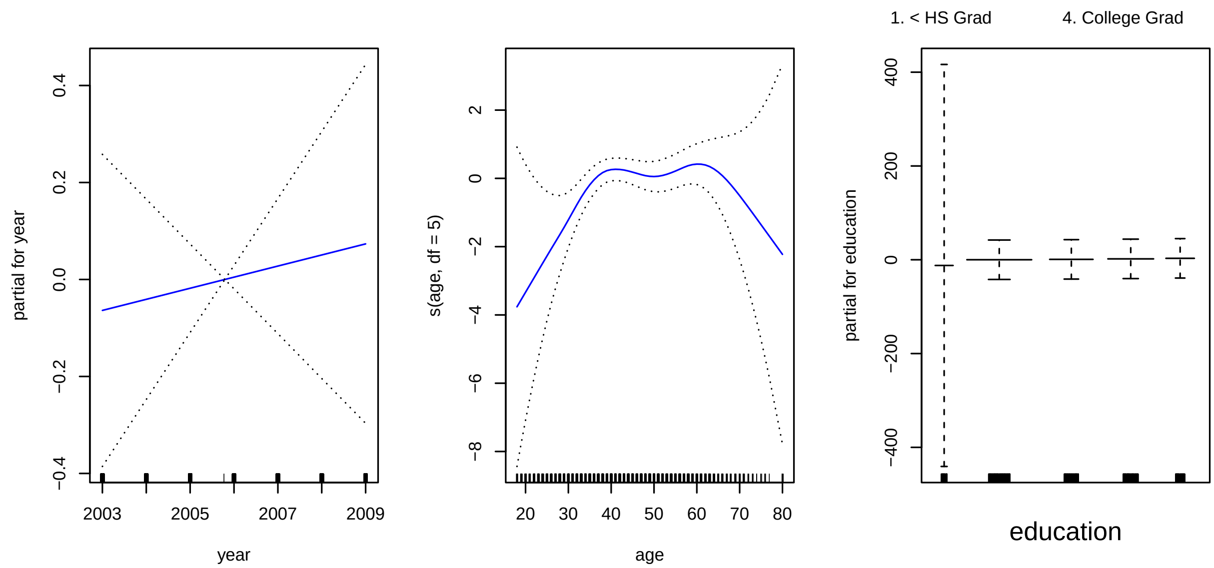
```
gam.lo.inter = gam(wage ~ s(year, df = 4) + lo(year, age, span = 0.5) + education,
                   data = Wage)
```

Fitting a Logistic Regression GAM

It can be done using `family="binomial"`. Here `I()` is used again to create a binary response variable:

```
gam.log = gam(I(wage > 250) ~ year + s(age, df = 5) + education,
              data = Wage, family = 'binomial')

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



In the `1. < HS Grad` category the error bar is huge, due to the fact that there are no high earners for that category:

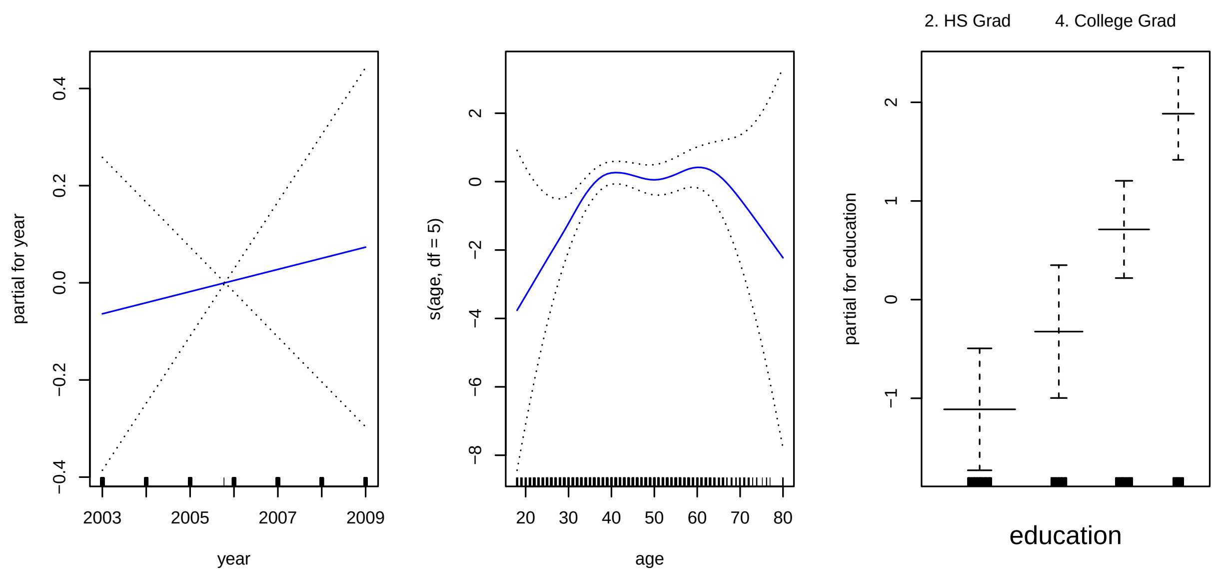
```
table(education, I(wage > 250))
```

```
##
## 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
```

It's best then to fit the GAM using all the `education` categories but that one:

```
gam.log.noHS = gam(I(wage > 250) ~ year + s(age, df = 5) + education,
                   data = Wage, family = 'binomial',
                   subset = (education != "1. < HS Grad"))

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


partial for year
0.4
0.2
0.0
−0.2
−0.4
2003
2005
2007
2009
year
s(age, df = 5)
2
0
−2
−4
−6
−8
20
30
40
50
60
70
80
age
2. HS Grad
4. College Grad
partial for education
2
1
0
−1
education