Lab 7 - Nonlinear Regression

Your name here

10/18/2023

This lab will work with the `Wage` data used as an example throughout the chapter.

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

1. Polynomial Regression and Step Functions

First, we will fit a polynomial of degree 4 to replicate Figure 7.1 from the text.

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

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

The syntax fits a linear model predicting `wage` with a fourth degree polynomial in age, `poly(age, 4)`. The function returns a matrix whose columns are a basis of orthogonal polynomials, which means that each column is a linear combination of the variables `age`, `age^2`, `age^3` and `age^4`. This means that each column of the design matrix is orthogonal to the other columns. This allows us to more easily assess if adding an additional power to the polynomial is significant and prevents the polynomial terms from being highly correlated with each other (i.e. the term `age^2` captures only the quadratic part of the relationship that is not captured by `age`.)

Alternatively, we can use `poly()` to obtain `age`, `age^2`, `age^3` and `age^4` directly. We can do this by using the `raw = TRUE` argument to the `poly()` function. 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.

See the first answer at https://stackoverflow.com/questions/29999900/poly-in-lm-difference-between-raw-vs-orthogonal for a nice further discussion comparing orthogonal vs. the raw input for polynomial regression.

```
fit2 <- lm(wage~poly(age, 4, raw=TRUE), 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 = TRUE)1  2.124552e+01 5.886748e+00  3.609042 0.0003123618
## poly(age, 4, raw = TRUE)2 -5.638593e-01 2.061083e-01 -2.735743 0.0062606446
## poly(age, 4, raw = TRUE)3  6.810688e-03 3.065931e-03  2.221409 0.0263977518
## poly(age, 4, raw = TRUE)4 -3.203830e-05 1.641359e-05 -1.951938 0.0510386498
```

Other equivalent ways to fit this model include:

```
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
```

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

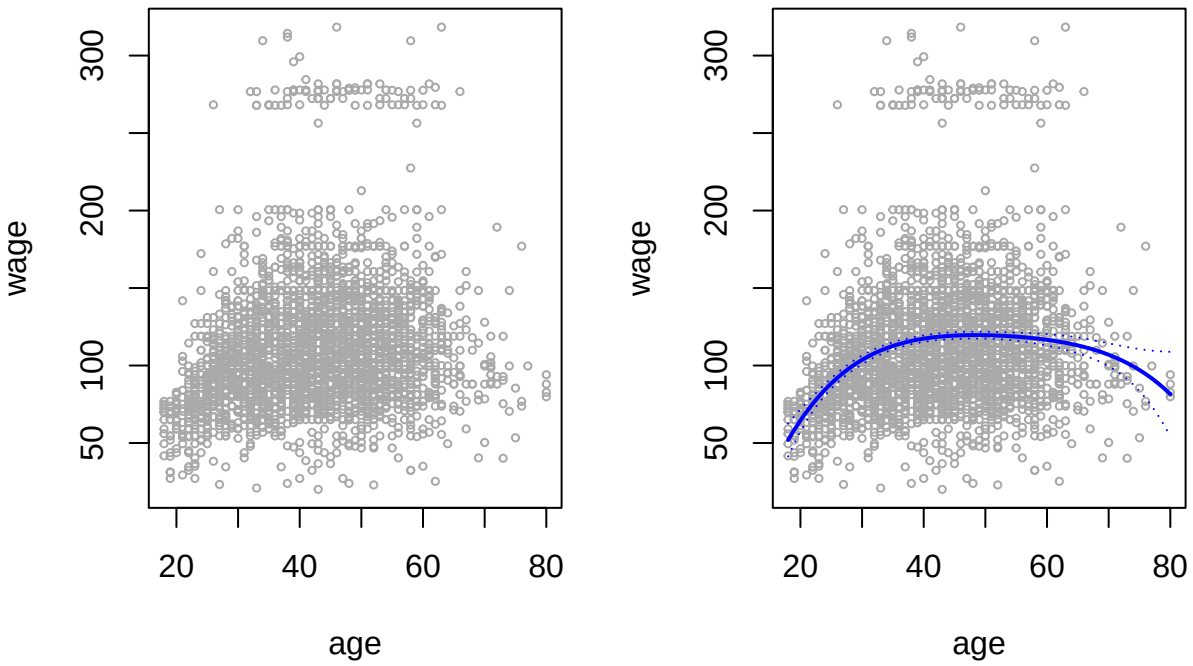
The second method above uses the `cbind()` function to build a matrix from a collection of vectors.

We can now create a grid of values for `age` at which we want predictions, and then 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]) #grid over observed range of values
preds <- predict(fit, newdata = list(age = age.grid), se = TRUE) # return standard errors
se.bands <- cbind(preds$fit + 2*preds$se.fit, preds$fit - 2*preds$se.fit)
```

We can finally 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 data only
plot(age, wage, xlim = agelims, cex = 0.5, col = "darkgrey")
#Plot fit and prediction standard error intervals with data
plot(age, wage, xlim = agelims, cex = 0.5, col = "darkgrey")
title("Degree-4 Polynomial", outer = TRUE)
lines(age.grid, preds$fit, lwd = 2, col = "blue")
matlines(age.grid, se.bands, lwd = 1, col = "blue", lty = 3)
```



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

To check that the choice of whether or not to use orthogonal basis functions does not affect predictions, we can compare the difference in predictions from the two models. The fitted values are identical in either case:

```
preds2 <- predict(fit2, newdata = list(age = age.grid), se = TRUE)
max(abs(preds$fit - preds2$fit)) ## check max difference between predictions
```

```
## [1] 6.88658e-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 perfoms an *analysis of variance* (ANOVA, using an F-test) in order to test the null hypothesis that a model M_1 is sufficient to explain the data against the alternative hypothesis that a more complex model M_2 is required. In order to use the `anova()` function, M_1 and M_2 must be *nested* models: the predictors in M_1 must be a subset of the predictions 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
```

Interpreting the `anova()` output: The p-value comparing the linear `Model 1` to the quadratic `Model 2` is essentially 0, indicating that the linear fit is not sufficient. Similarly, the p-value comparing the quadratic `Model 2` to the cubic `Model 3` is very low, 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 0.37. 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.

Alternatively, instead of using the `anova()` function, we could have obtained these p-values more succinctly by exploiting the fact that `poly()` creates orthogonal polynomials, which allows us to isolate the effect of adding additional powers.

```
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 method works whether or not we use orthogonal polynomials and also works when we have other terms in the model. For example:

```
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
```

Alternatively, we could use cross validation to select the degree of the polynomial.

Next, we will look at predicting whether an individual earns more that \$250,000 per year. We proceed much 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)
```

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

The `predict()` can again be used to make predictions.

```
preds <- predict(fit, newdata = data.frame(age = age.grid),
                 se = TRUE)
```

However, calculating confidence intervals is more involved here 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

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

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 $P(Y=1|X)$, we use the transformation

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

```
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))
```

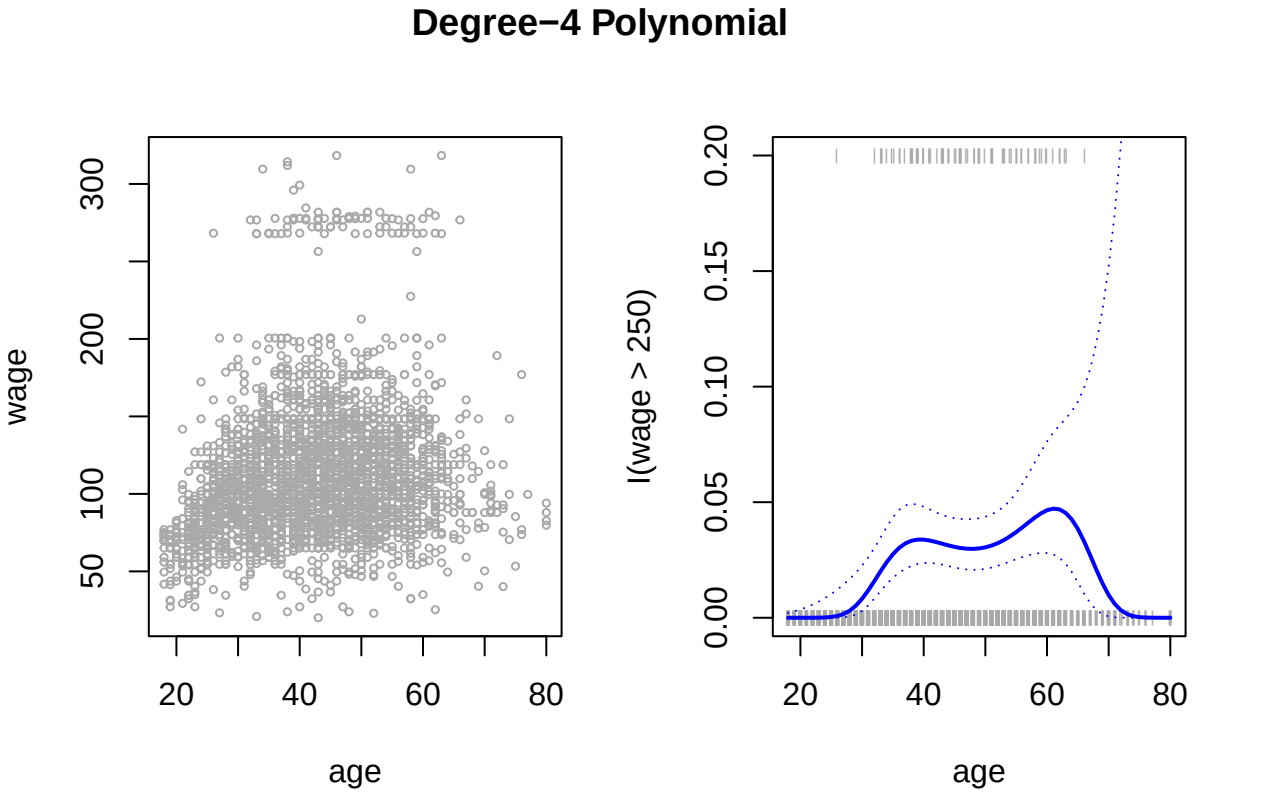
Note that we could have directly computed the probabilities by selecting the `type = "response"` option in the `predict()` function.

```
preds <- predict(fit, newdata = data.frame(age = age.grid),
                 type = "response",
                 se = TRUE)
```

However, the corresponding confidence intervals would not have made sense, since we would have gotten negative probabilities!

Finally, we can make the right hand plot from Figure 7.1:

```
par(mfrow = c(1,2), mar = c(4.5, 4.5, 1, 1), oma = c(0,0,4,0))
#Plot fit and prediction standard error intervals with data
plot(age, wage, xlim = agelims, cex = 0.5, col = "darkgrey")
title("Degree-4 Polynomial", outer = TRUE)
lines(age.grid, preds$fit, lwd = 2, col = "blue")
matlines(age.grid, se.bands, lwd = 1, col = "blue", lty = 3)

plot(age, I(wage > 250), xlim = agelims, type = "n", ylim = c(0, 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 plotted the `age` values that correspond to observations with `wage` values above 250 on the top of the plot, and those with `wage` values below 250 are shown as gray marks on the bottom of the plot. We used `jitter()` to move around the `age` values a little bit so that observations with the same `age` values do not cover each other up. This is often called a *rug plot*.

In order to fit a step function, we need to 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
```

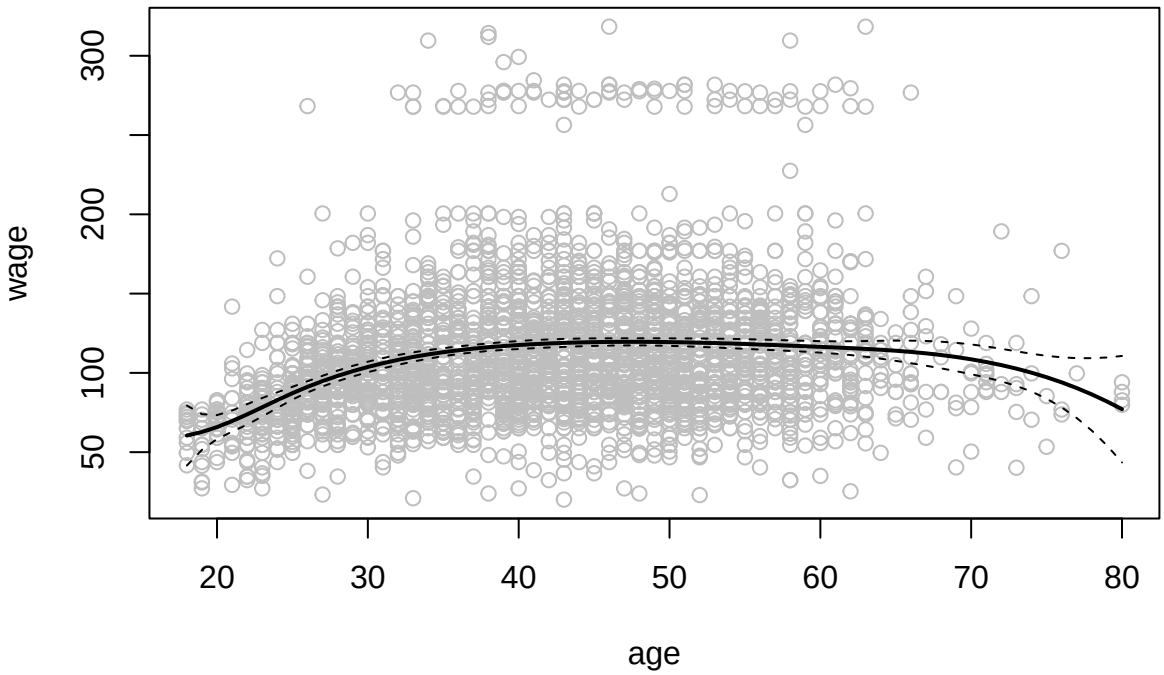
For this example, `cut()` automatically picked the cutpoints at 33.5, 49 and 64.5 years of age. We could also have specified our own cutpoints directly using the `breaks` option. The function `cut()` returns an ordered categorical variable; the `lm()` function then creates a set of dummy variables for use in the regression. The `age < 33.5` category is left out, so the intercept coefficient of $94,160 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 in the other age groups. We can produce predictions and plots just like we did with the polynomial fit.

2. Splines

We can use the `splines` library to fit splines in `R`. We saw in Section 7.4 of the text that regression splines can be fit by constructing an appropriate matrix of basis functions. Please review that section in the text before continuing with the lab.

The `bs()` function generates the entire matrix of basis functions for splines with the specified set of knots. By default, cubic splines are produced. We can fit `wage` to `age` using a regression spline as follows:

```
library(splines)
fit <- lm(wage ~ bs(age, knots = c(25, 40, 60)), data = Wage)
pred <- predict(fit, newdata = list(age = age.grid), se = TRUE)
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 pre-specified knots at ages 25, 40 and 60. This produces a spline with six basis functions. Recall that a cubic spline with three knots has seven degrees of freedom; these degrees of freedom are used by an intercept, plus six basis functions. We could use the `df` option to produce a spline with knots at uniform quantiles of the data.

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

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

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

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

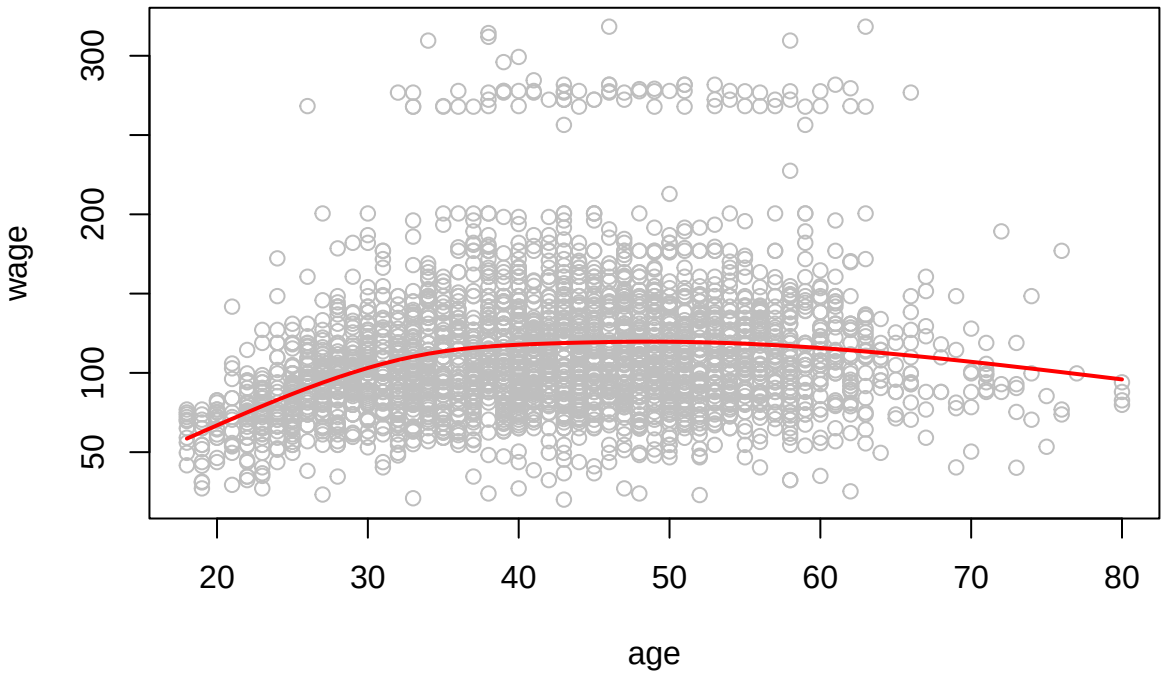
```
attr(bs(age, df = 6), "knots")
```

```
## [1] 33.75 42.00 51.00
```

In this case `R` chooses knots at ages 33.8, 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 results in a cubic spline).

In order to fit a natural spline, we use the `ns()` function. We can 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 = TRUE)
plot(age, wage, col = "gray")
lines(age.grid, pred2$fit, col = "red", lwd = 2)
```



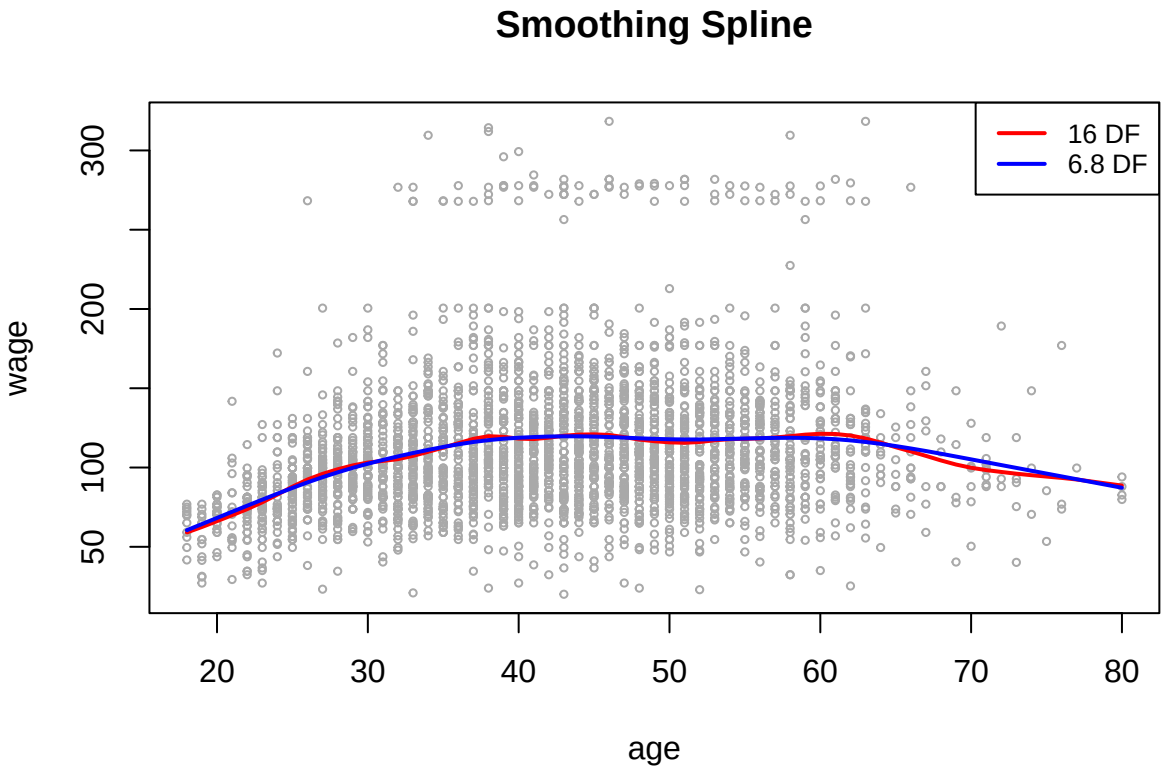
Just like with the `bs()` function, we can instead specify the knots directly using the `knots` option.

The `smooth.spline()` function can be used to fit a smoothing spline. We can reproduce Figure 7.8 using the following code:

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

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

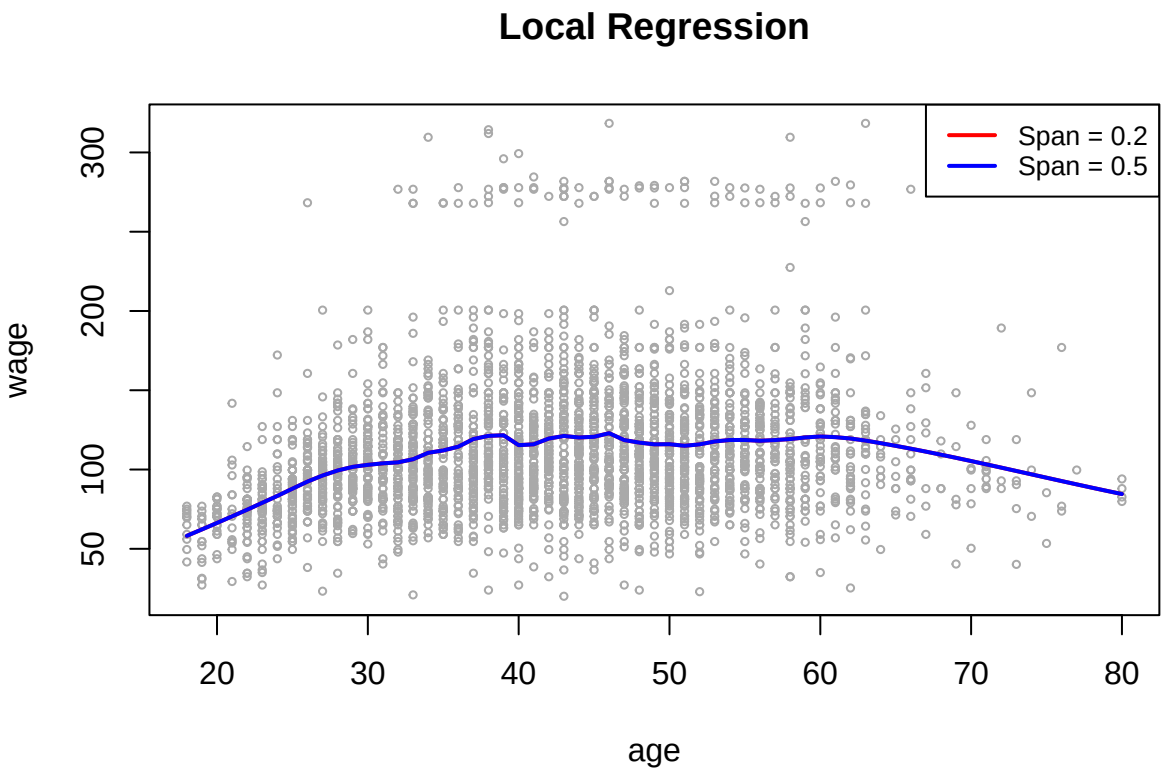
```
lines(fit, col = "red", lwd = 2)
lines(fit2, col = "blue", lwd = 2)
legend("topright", legend = c("16 DF", "6.8 DF"), col = c("red", "blue"),
       lty = 1, lwd = 2, cex = 0.8)
```



For 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 `fit2`, 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", lwd = 2)
lines(age.grid, predict(fit, 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 have performed local linear regression using spans of 0.2 and 0.5: that is, each neighborhood consists of 20% or 50% of the observations. The larger the span, the smoother the fit. The `locfit` library can also be used for fitting local regression models in `R`.

3. GAMs

We now fit a GAM to predict `wage` using natural spline functions of `year` and `age`, treating `education` as a qualitative predictor.

$$\text{wage} = \beta_0 + f_1(year) + f_2(age) + f_3(education) + \epsilon.$$

Since this is just a big linear regression model using an appropriate choice of basis functions, we can simply do this using the `lm()` function.

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

We now fit the model specified above 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`.

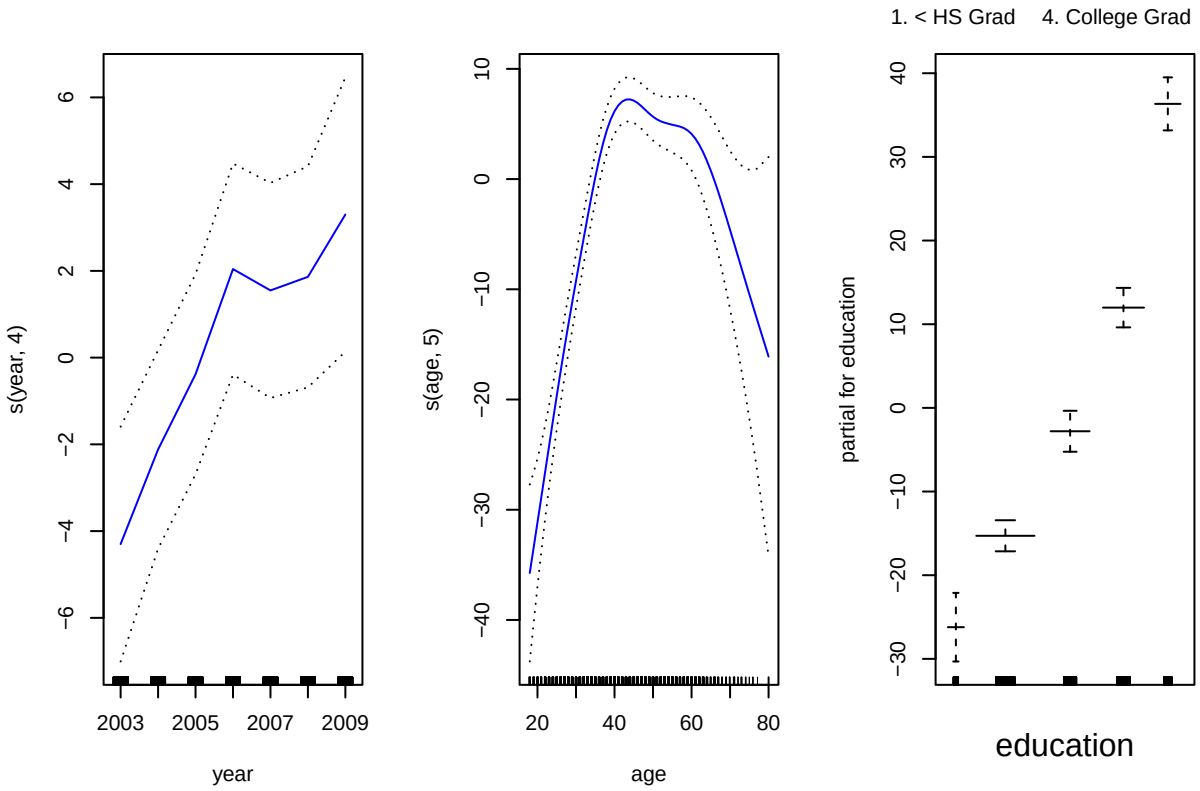
The `s()` function, which is part of the `gam` library, is used to indicate that we would like to use a smoothing spline. We specify that the function of `year` should have 4 degrees of freedom and that the function of `age` will have 5 degrees of freedom.

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 in the above model are fit simultaneously, taking each other into account to explain the response.

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

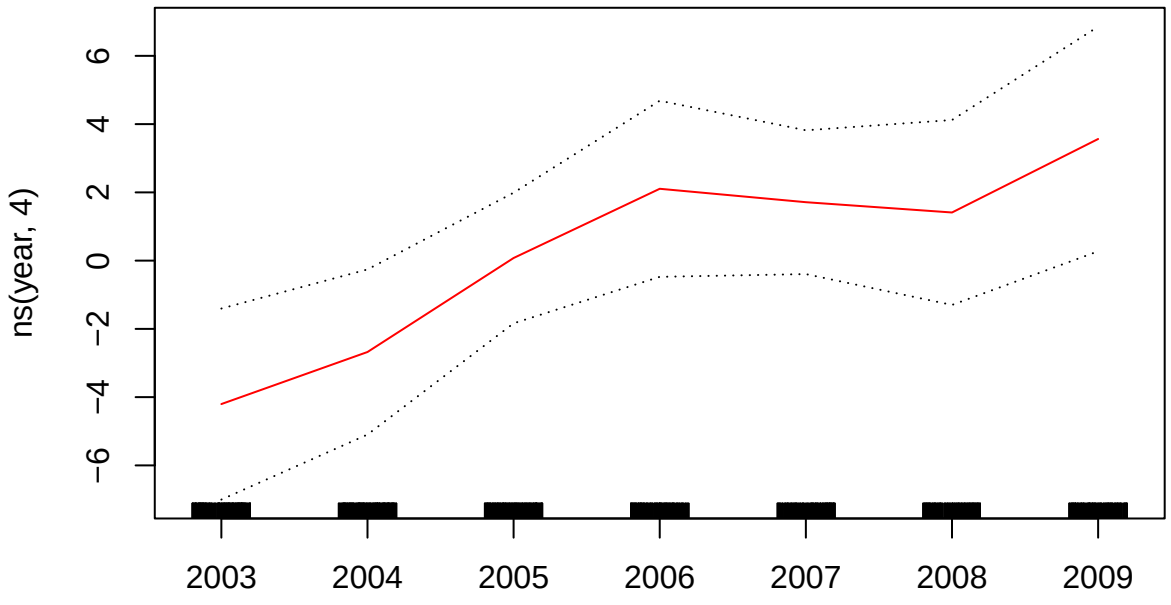
We can reproduce Figure 7.12 using the `plot()` function:

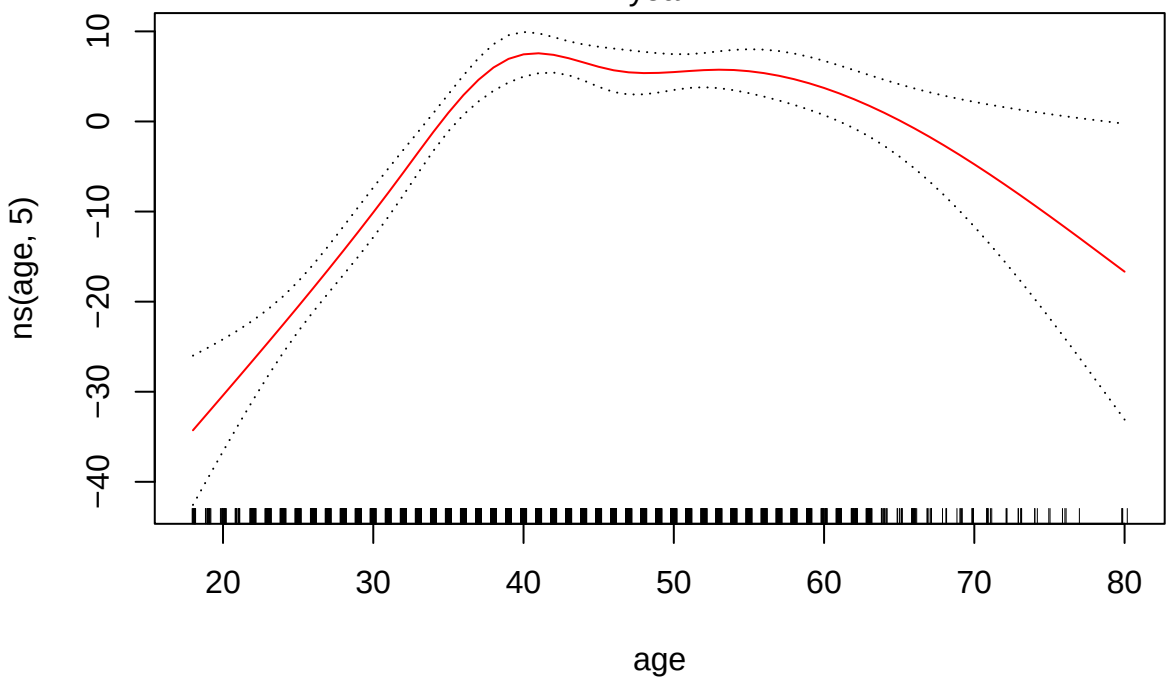
```
par(mfrow = c(1,3))
plot(gam.m3, se = TRUE, col = "blue")
```

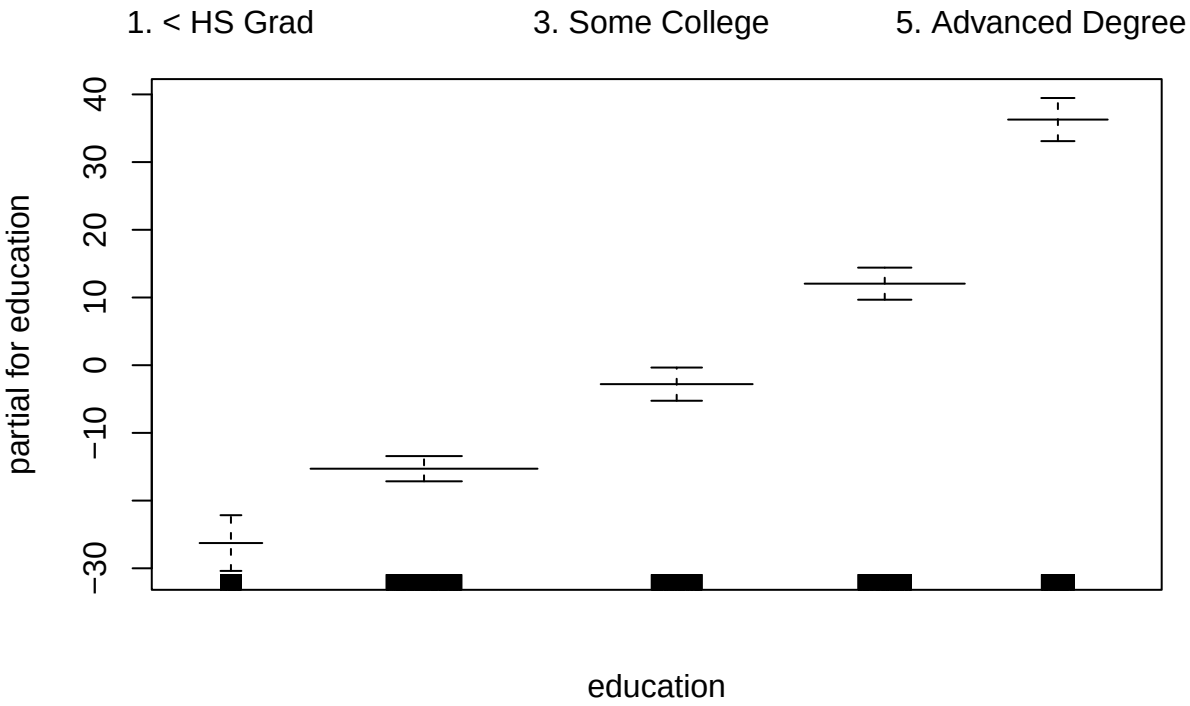


The generic `plot()` function recognizes that `gam2` is an object of class `gam`, and invokes the appropriate `plot.gam()` method. Conveniently, even though `gam1` is of class `lm()` and not of class `gam`, we can *still* use `plot.gam()` on it. Figure 7.11 was produced using the following expression:

```
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



Note: here we use `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 in order to determine which of these three models is best: a GAM that excludes `year` (M_1), a GAM that uses a linear function of `year` (M_2) or a GAM that uses a spline function of `year` (M_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 compelling evidence that a GAM with a linear function of `year` is better than a GAM that does not include `year` at all. However, there is no evidence that a non-linear function of `year` is needed, looking at the p-values. Based on the results of the ANOVA,then, we can select M_2.

The `summary()` function can be used to produce 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
```

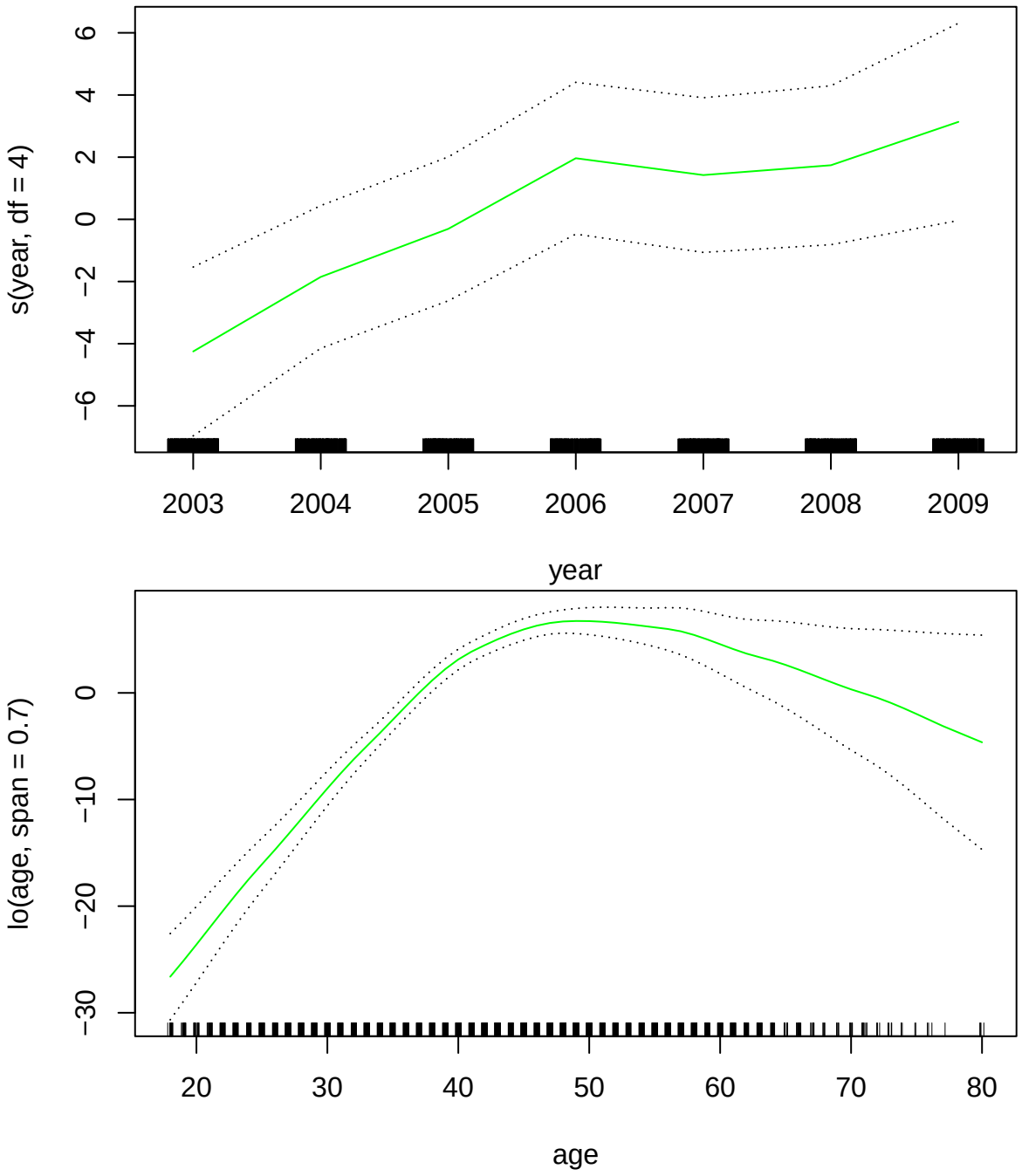
The p-values for `year` and `age` correspond to a null hypothesis of a linear relationship versus the alternative of a non-linear relationship. The large p-value for `year` reinforces our conclusion from the ANOVA test that a linear function is adequate for this term. However, there is very clear evidence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

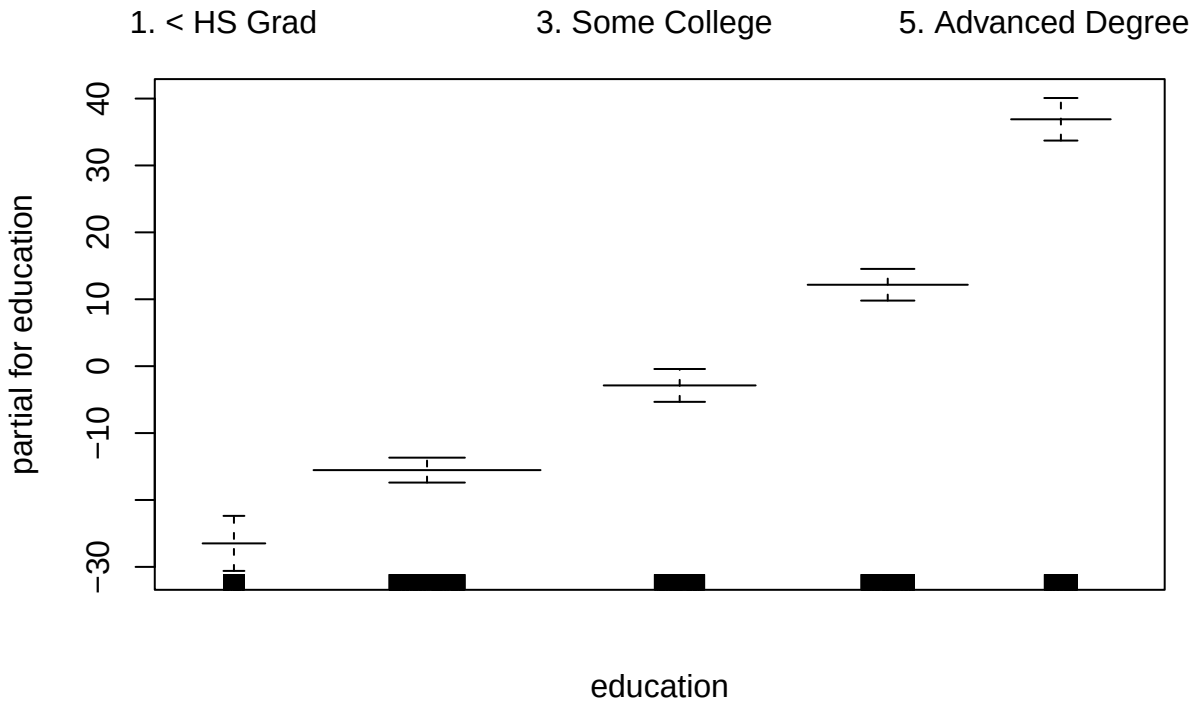
We can make predictions on `gam` objects, just like from `lm` objects, using the `predict()` method for the class `gam`. Here, we make predictions on the training set.

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

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

```
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

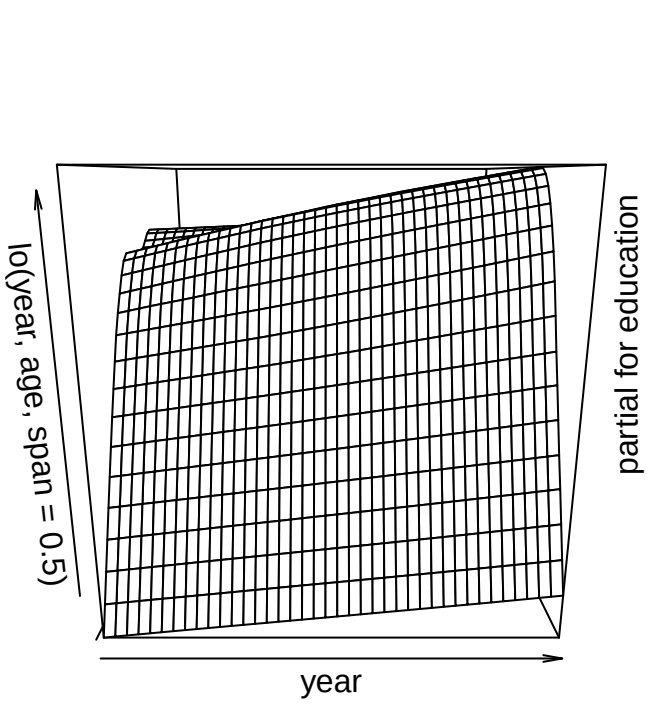


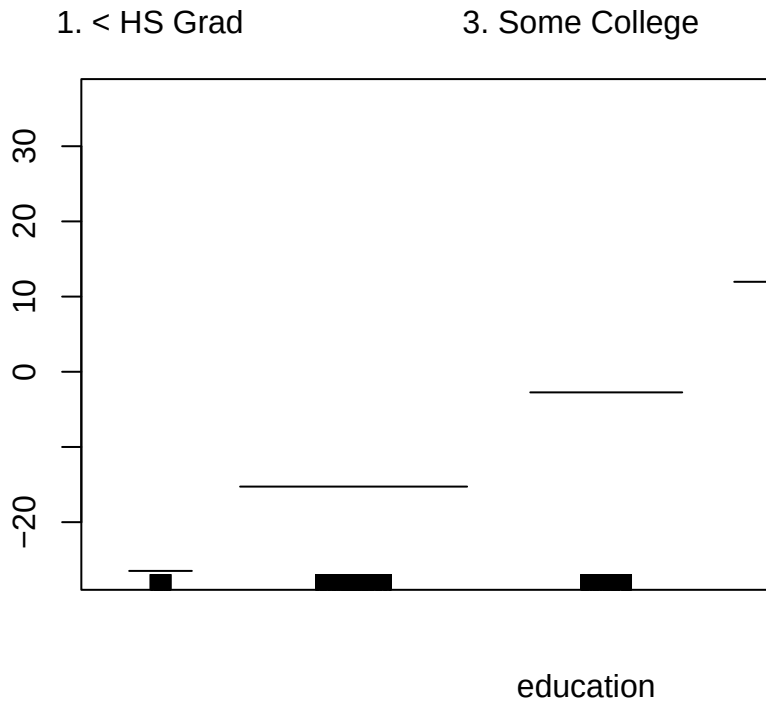
Here, we have used local regression for the `age` term, with a span of 0.7. We can also use the `lo()` function to create interactions before calling the `gam()` function. For example,

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

fits a 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 surface if we first install the `akima` package.

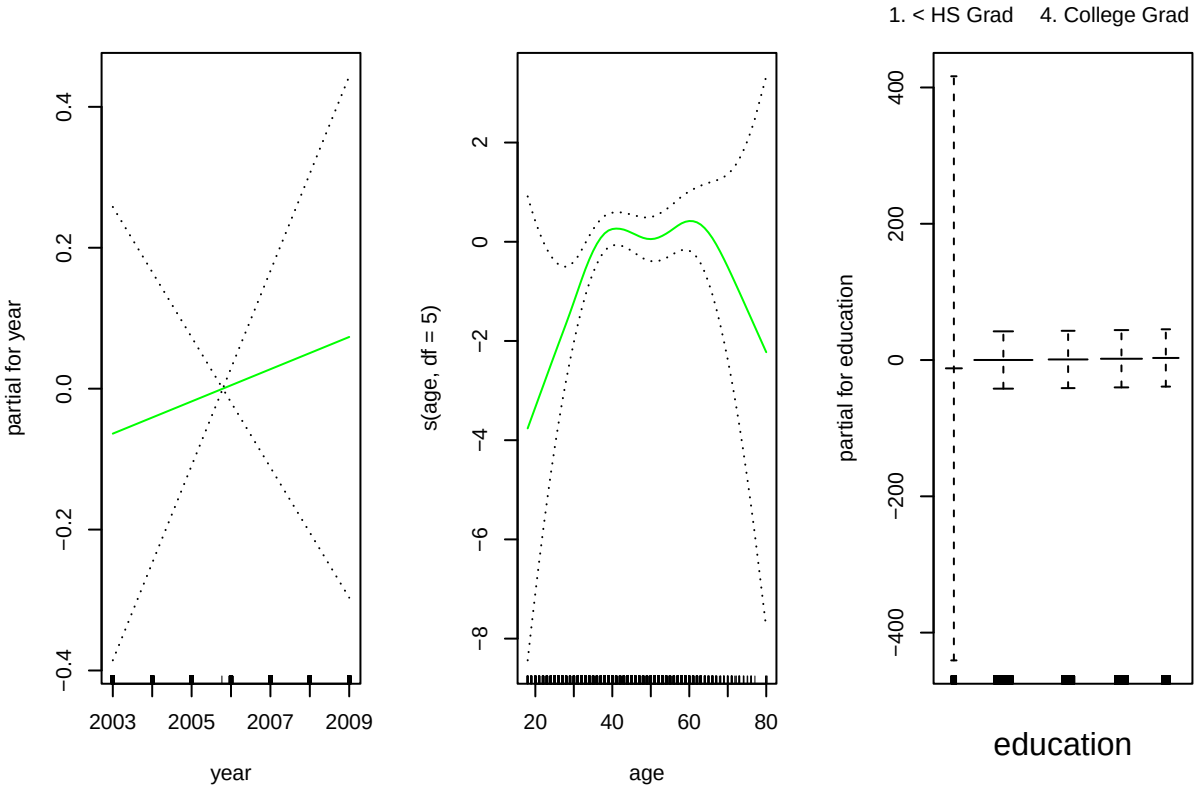
```
library(akima)
plot(gam.lo.i)
```





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 = TRUE, col = "green")
```



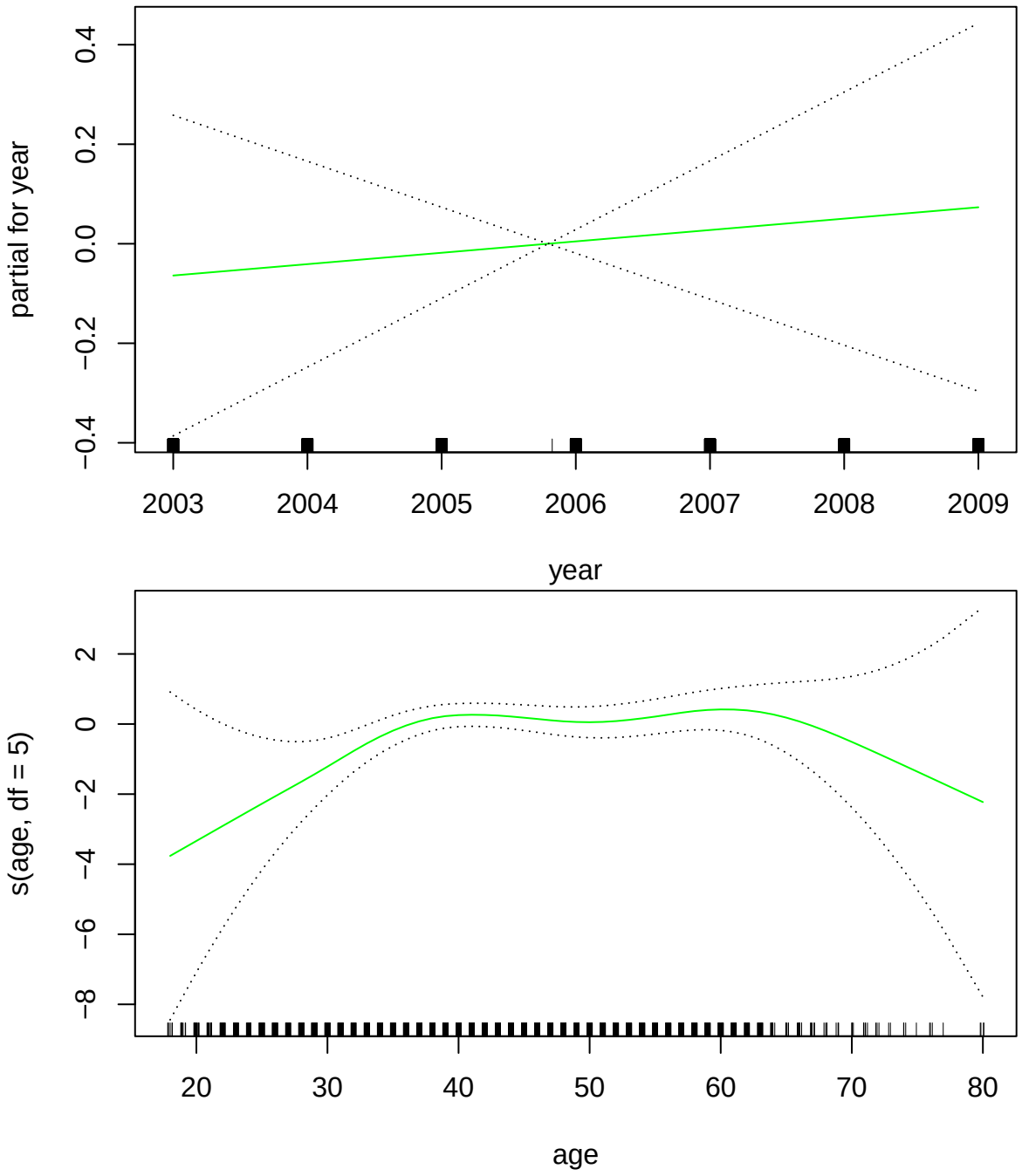
It is easy to see that there are no high earners in the <HS category:

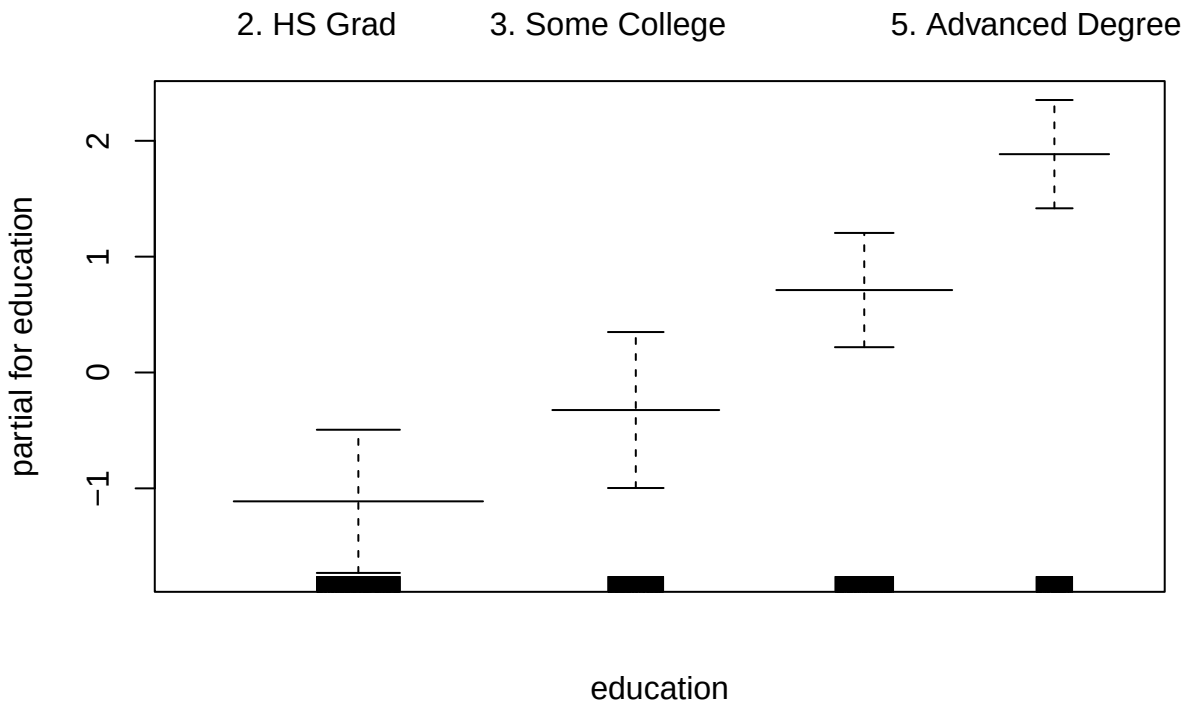
```
table(education, I(wage > 250))
```

```
##
## 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!="1. < HS Grad"))
plot(gam.lr.s, se = TRUE, col = "green")
```


partial for year
0.4
0.2
0.0
−0.2
−0.4
2003
2004
2005
2006
2007
2008
2009
year
s(age, df = 5)
2
0
−2
−4
−6
−8
20
30
40
50
60
70
80
age



Problems

1. Polynomial Regression

We will work with the `iris` dataset again for this question.

```
data("iris")
```

Suppose that we want to predict `Sepal.Length` using the other predictors and `Species` as predictors.

(a) Explore what power of `Sepal.Width` predicts `Sepal.Length` well. Explore polynomial degrees up to 5 and make sure to use an orthogonal basis for the polynomials. Use `anova` to determine which degree polynomial to use. Print the coefficients and their estimates (i.e. using `summary()`) once you have decided on a model.

```
degrees <- 1:5

# Create a list to store the models
models <- list()

# Fit polynomial regression models for different degrees
for (degree in degrees) {
  model <- lm(Sepal.Length ~ poly(Sepal.Width, degree, raw = FALSE), data = iris)
  models[[degree]] <- model
}

# Use anova to compare models
anova_results <- anova(models[[1]], models[[2]], models[[3]], models[[4]], models[[5]])

print(anova_results)
```

```
## Analysis of Variance Table
##
## Model 1: Sepal.Length ~ poly(Sepal.Width, degree, raw = FALSE)
## Model 2: Sepal.Length ~ poly(Sepal.Width, degree, raw = FALSE)
## Model 3: Sepal.Length ~ poly(Sepal.Width, degree, raw = FALSE)
## Model 4: Sepal.Length ~ poly(Sepal.Width, degree, raw = FALSE)
## Model 5: Sepal.Length ~ poly(Sepal.Width, degree, raw = FALSE)
##   Res.Df     RSS Df Sum of Sq      F  Pr(>F)
## 1    148 100.756
## 2    147  98.752  1    2.0044 3.1031 0.08026 .
## 3    146  95.052  1    3.6998 5.7278 0.01799 *
## 4    145  95.044  1    0.0081 0.0125 0.91106
## 5    144  93.015  1    2.0287 3.1407 0.07848 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

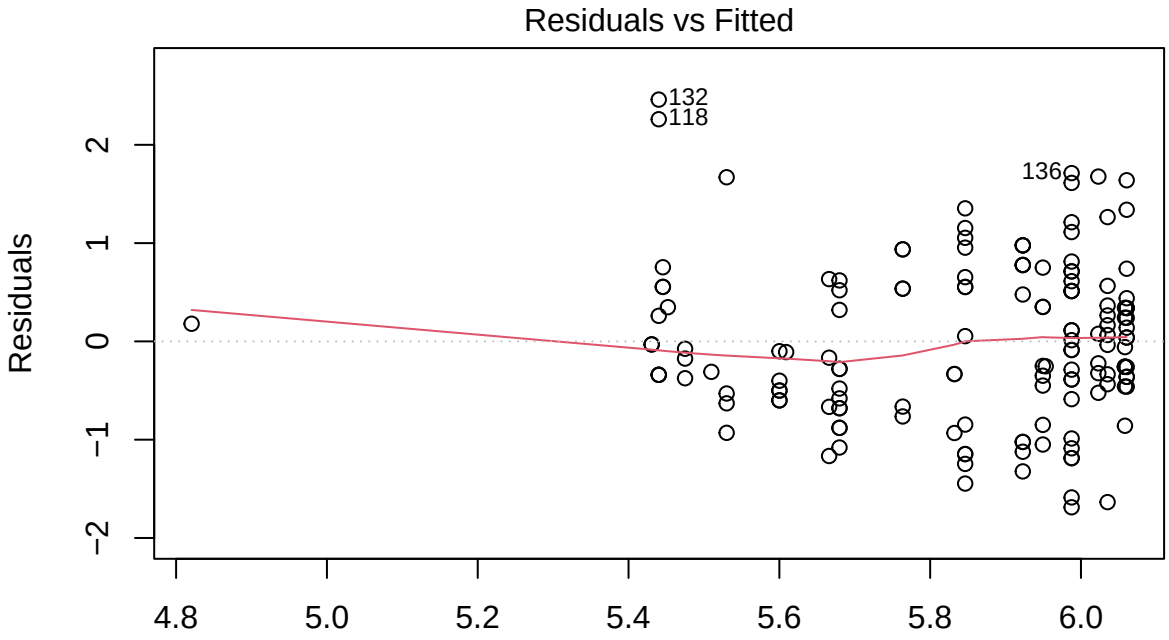
It seems polynomial degree 3 is appropriate as its p-value is still statistically significant, whereas going beyond 3 doesn't yield statistically significant models.

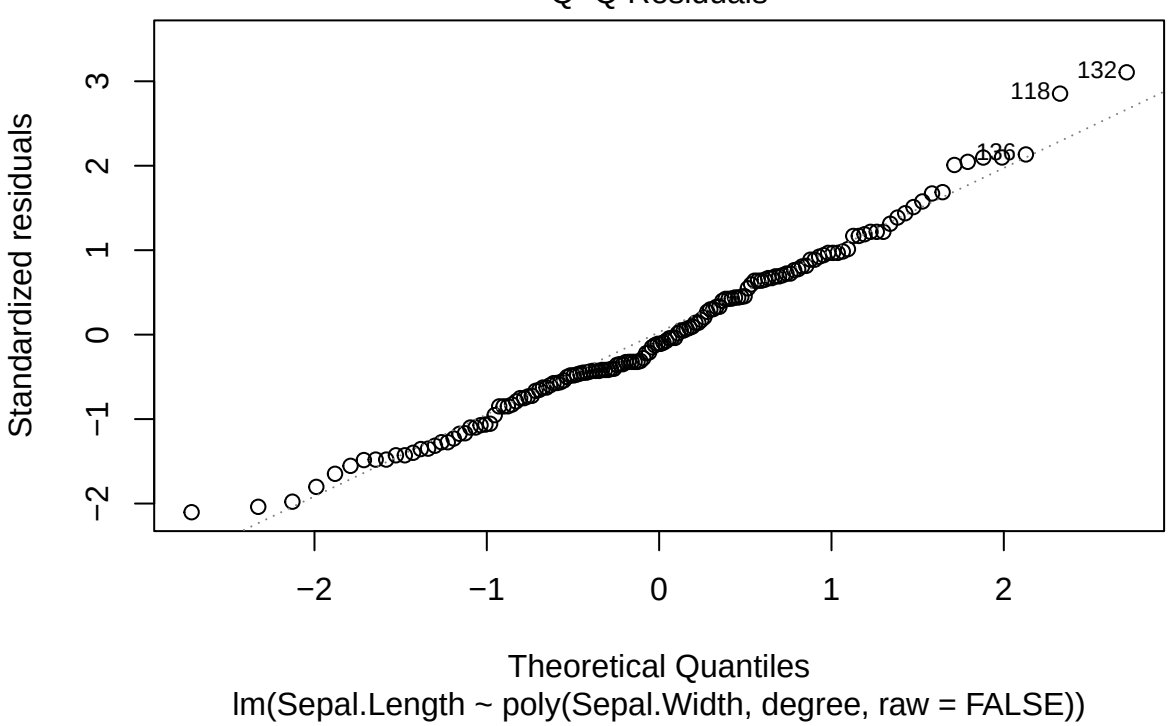
```
best.model <- models[[3]]
summary(best.model)
```

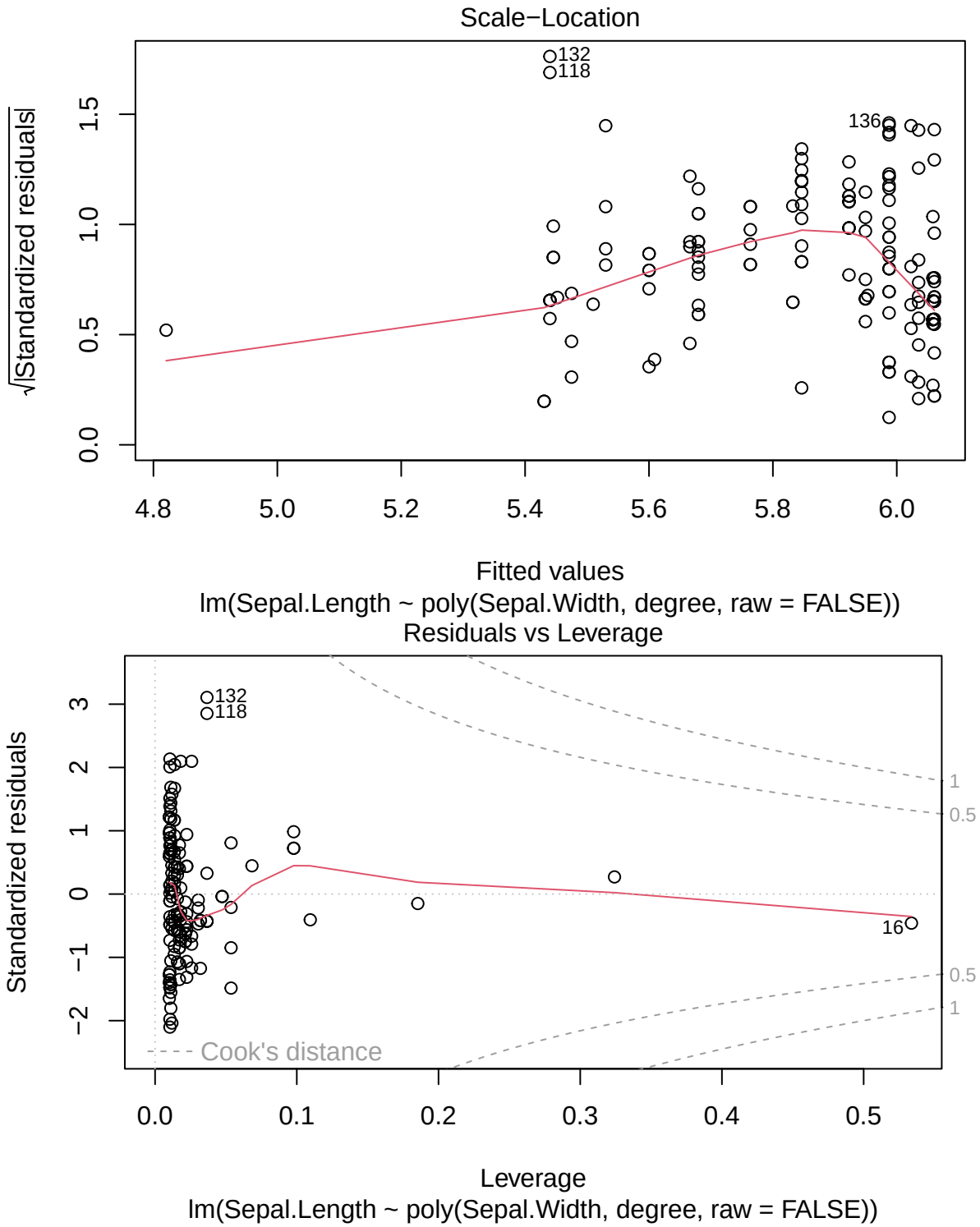
```
##
## Call:
## lm(formula = Sepal.Length ~ poly(Sepal.Width, degree, raw = FALSE),
##     data = iris)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.6876 -0.5001 -0.0876  0.5493  2.4600
##
## Coefficients:
##                                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)                           5.84333    0.06588  88.696   <2e-16
## poly(Sepal.Width, degree, raw = FALSE)1 -1.18838    0.80687  -1.473   0.1430
## poly(Sepal.Width, degree, raw = FALSE)2 -1.41578    0.80687  -1.755   0.0814
## poly(Sepal.Width, degree, raw = FALSE)3  1.92349    0.80687   2.384   0.0184
##
## (Intercept)                          ***
## poly(Sepal.Width, degree, raw = FALSE)1
## poly(Sepal.Width, degree, raw = FALSE)2 .
## poly(Sepal.Width, degree, raw = FALSE)3 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8069 on 146 degrees of freedom
## Multiple R-squared:  0.06965,    Adjusted R-squared:  0.05054
## F-statistic: 3.644 on 3 and 146 DF,  p-value: 0.01425
```

(b) Once you have decided on a model, perform model diagnostics and assess the quality of the model fit. Are there any outliers or high leverage points? How do the residuals look?

```r
plot(best.model)
```







According to the model diagnostic plots, it seems the model is a good fit, though R has highlighted several potential outliers (132, 118, and 136) and a high-leverage point (16).

(c) As we increase the degree of the polynomial that we fit, what can we say about the bias of our model? What can we also say about the test MSE of the model fit? (Answer qualitatively, you don't need to run more code unless you want to!)

As we increase the degree of the polynomial, model becomes more flexible, allowing it to fit the training

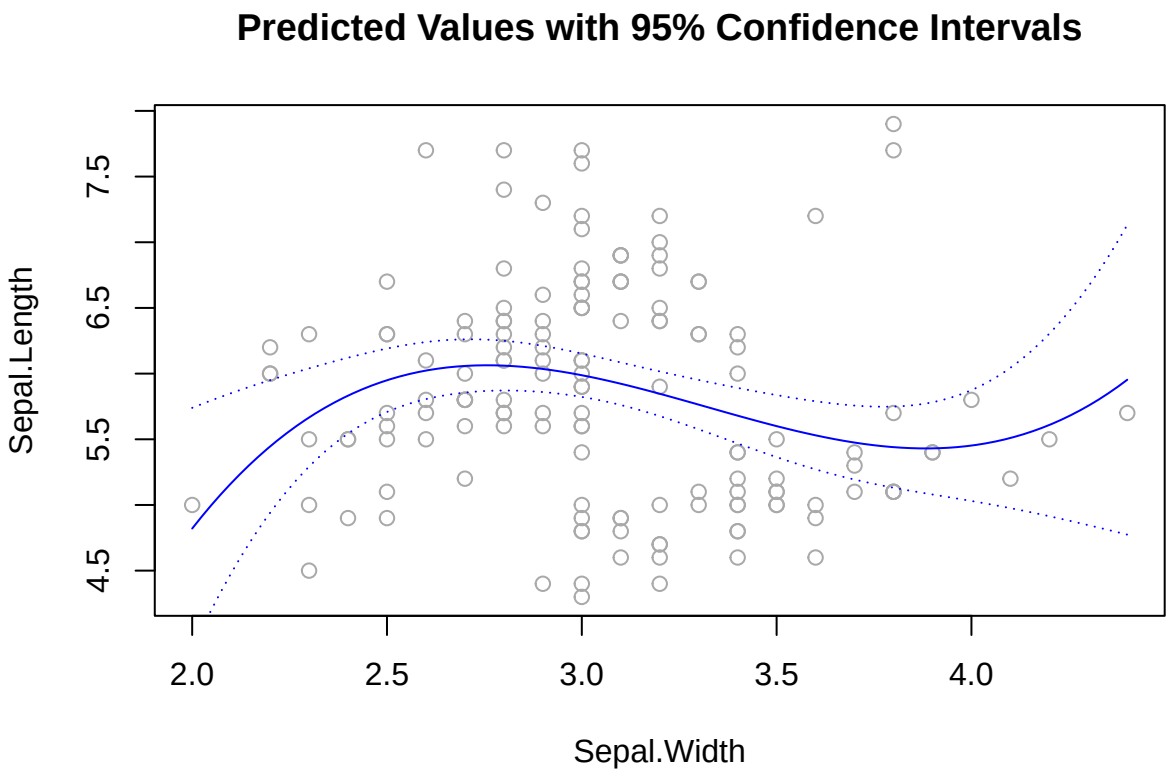
data more closely, leading to a decrease in bias. A high-degree polynomial can capture complex relationships in the data.

The test MSE follows a U-shaped curve. As we increase the degree of the polynomial, the test MSE may decrease as the model captures more underlying patterns. However, beyond a certain degree, the test MSE may start to increase due to overfitting. The model becomes too sensitive to noise, leading to poor generalization and higher test MSE.

(d) Use the predict function with standard errors to plot the data, the predicted values over a grid of points within the range of `Sepal.Width` and then the 95% standard errors of the predictions. Use the final model you selected from part (a) above.

```r
best.model <- lm(Sepal.Length ~ poly(Sepal.Width, 3, raw = FALSE), data = iris)

# Create a grid of Sepal.Width values for prediction
width.grid <- seq(min(iris$Sepal.Width), max(iris$Sepal.Width), length = 100)

# Predict using the same basis type (nmatrix.3)
preds <- predict(best.model, newdata = list(Sepal.Width = width.grid), se = TRUE)

# Create standard error bands
se.bands <- cbind(preds$fit + 2 * preds$se.fit, preds$fit - 2 * preds$se.fit)
```

```r
# Plot the data, predicted values, and 95% standard errors
plot(iris$Sepal.Width, iris$Sepal.Length, col = "darkgrey", xlab = "Sepal.Width", ylab = "Sepal.Length")
title("Predicted Values with 95% Confidence Intervals")

# Plot the predicted values and standard error bands
lines(width.grid, preds$fit, col = "blue")
matlines(width.grid, se.bands, col = "blue", lty = 3)
```



2. Splines

Using the `iris` data again, fit a smoothing spline to predict `Sepal.Length` from `Sepal.Width`. Fit two splines: one with 10 degrees of freedom, and one where the degrees of freedom are selected via cross-validation. Report the degrees of freedom found via cross validation, and plot both splines with the data. Make sure to add a legend to your plot.

```
# Fit a smoothing spline with 10 degrees of freedom
spline_10df <- smooth.spline(iris$Sepal.Width, iris$Sepal.Length, df = 10)
```

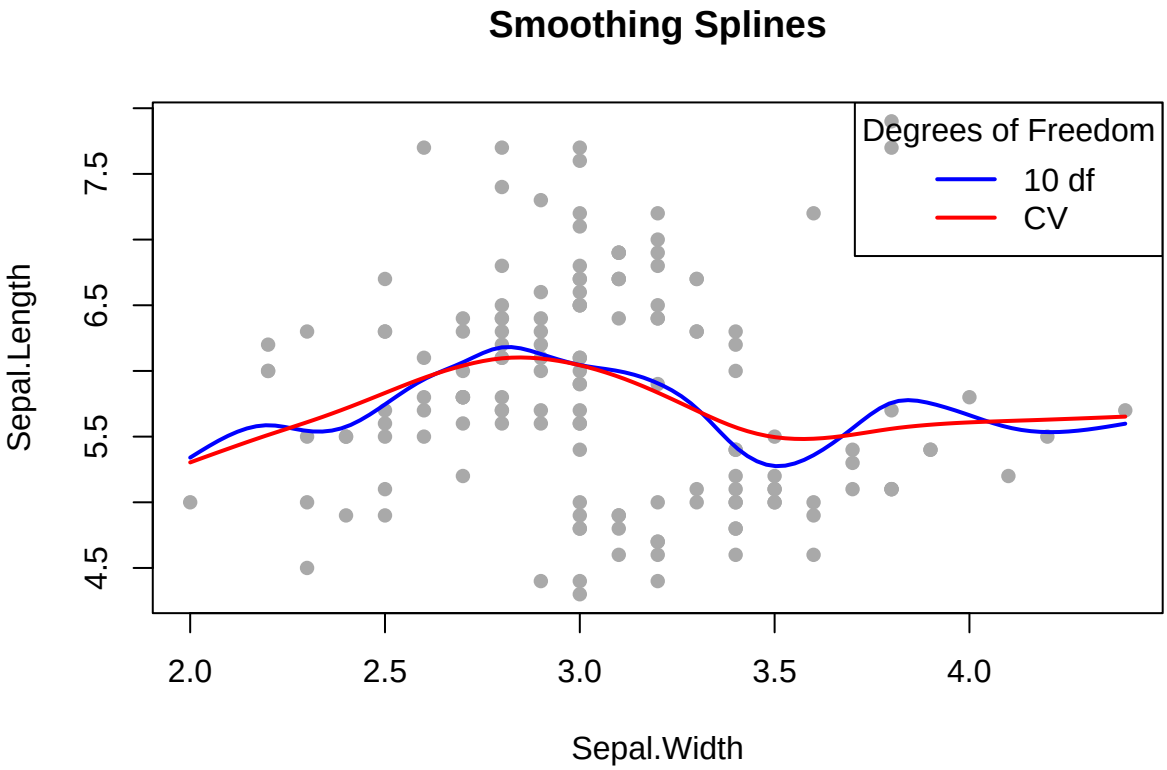
```
# Fit a smoothing spline with degrees of freedom selected via cross-validation
spline_cv <- smooth.spline(iris$Sepal.Width, iris$Sepal.Length, cv = TRUE)
```

```
## Warning in smooth.spline(iris$Sepal.Width, iris$Sepal.Length, cv = TRUE):
## cross-validation with non-unique 'x' values seems doubtful
```

```
df_cv <- spline_cv$df
print(df_cv)
```

```
## [1] 5.33148
```

```
# Create a grid of Sepal.Width values for plotting
width.grid <- seq(min(iris$Sepal.Width), max(iris$Sepal.Width), length = 100)

# Predict Sepal.Length using both splines
predicted_10df <- predict(spline_10df, x = width.grid)$y
predicted_cv <- predict(spline_cv, x = width.grid)$y

# Create a plot with the data and splines using the base R plot function
plot(iris$Sepal.Width, iris$Sepal.Length, pch = 16, col = "darkgrey",
     xlab = "Sepal.Width",
     ylab = "Sepal.Length",
     main = "Smoothing Splines")
lines(width.grid, predicted_10df, col = "blue", lwd = 2)
lines(width.grid, predicted_cv, col = "red", lwd = 2)
legend("topright", legend = c("10 df", "CV"), col = c("blue", "red"), lwd = 2, title = "Degrees of Freed
```



3. GAMS

Explain what ϵ represents in the GAM model:

$$\text{wage} = \beta_0 + f_1(year) + f_2(age) + f_3(education) + \epsilon.$$

The symbol ϵ represents the error term. It is the part of the outcome variable (in this case, "wage") that cannot be explained by the predictor variables (year, age, and education) and the smooth functions (f1, f2, and f3).