Beyond Linearity

Simone Collier

Getting Started

Start by loading the packages that have the data we need. If you need to install the packages first then run `install.packages("PACKAGENAME")` in your console before running the code chunk.

```
library(ISLR2)
library(ggplot2)
library(splines)
library(gam)
```

We will be making use of the `Wage` data set in the `ISLR2` package. This data set contains information about the wage of 3000 men in the Mid-Atlantic region. There are many other variables included such as `age`, the `year` the information was recorded, etc.

```
attach(Wage)
names(Wage)
```

```
##  [1] "year"       "age"        "maritl"     "race"       "education"
##  [6] "region"     "jobclass"   "health"     "health_ins" "logwage"
## [11] "wage"
```

We want to create models that use the other variables in the data set to predict the `wage` of an individual.

Polynomial Regression

We can fit a polynomial regression using the `lm()` function by performing multiple linear regression with powers of a single predictor. In our case we will use the variable `age`, so our predictors for a fourth degree polynomial regression are `age`, `age^2`, `age^3`, and `age^4`.

```
poly.fit <- lm(wage ~ cbind(age, age^2, age^3, age^4), data = Wage)
coef(summary(poly.fit))
```

```
##                                         Estimate   Std. Error   t value
## (Intercept)                        -1.841542e+02 6.004038e+01 -3.067172
## cbind(age, age^2, age^3, age^4)age  2.124552e+01 5.886748e+00  3.609042
## cbind(age, age^2, age^3, age^4)    -5.638593e-01 2.061083e-01 -2.735743
## cbind(age, age^2, age^3, age^4)     6.810688e-03 3.065931e-03  2.221409
## cbind(age, age^2, age^3, age^4)    -3.203830e-05 1.641359e-05 -1.951938
##                                        Pr(>|t|)
## (Intercept)                        0.0021802539
```

```
## cbind(age, age^2, age^3, age^4)age 0.0003123618
## cbind(age, age^2, age^3, age^4)    0.0062606446
## cbind(age, age^2, age^3, age^4)    0.0263977518
## cbind(age, age^2, age^3, age^4)    0.0510386498
```

We can produce the same result using the `poly()` function which gives us the powers of `age` when the argument `raw = TRUE`. Otherwise it gives a linear combination of the powers of `age` which will affect the fitted coefficients but not the model predictions.

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

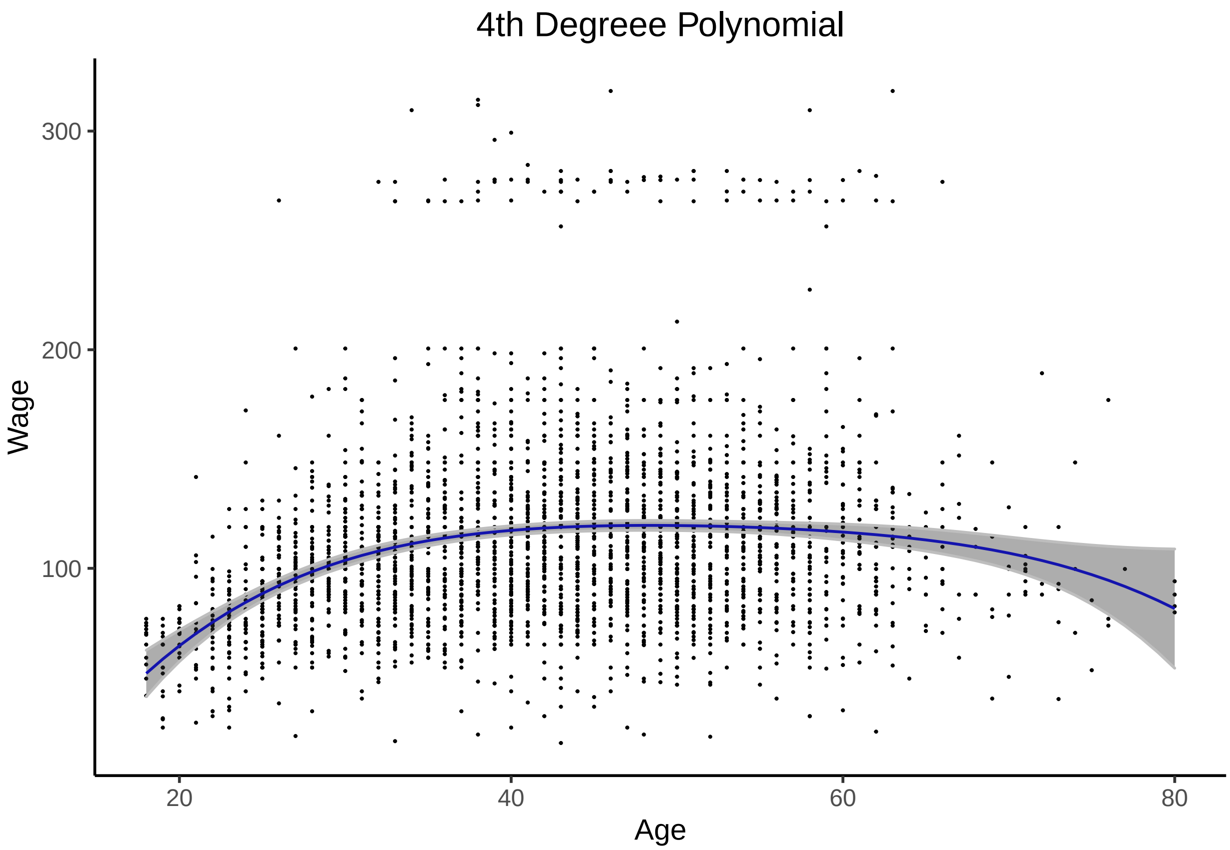
```
##                                Estimate   Std. Error   t value     Pr(>|t|)
## (Intercept)               -1.841542e+02 6.004038e+01 -3.067172 0.0021802539
## poly(age, 4, raw = TRUE)1  2.124552e+01 5.886748e+00  3.609042 0.0003123618
## poly(age, 4, raw = TRUE)2 -5.638593e-01 2.061083e-01 -2.735743 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
```

We can pick a range of ages that we want predictions of `wage` and set the argument `se = TRUE` to include the standard errors in the output. We can then create a 95% confidence intervals by adding and subtracting 2 standard errors from the predictions.

```
age.range <- seq(from = min(age), to = max(age))
pred <- predict(poly.fit, newdata = list(age = age.range), se = TRUE)
conf.int <- cbind(pred$fit + 2 * pred$se.fit, pred$fit - 2 * pred$se.fit)
```

We can create some plots to visualize the results of the fit.

```
predictions <- data.frame(AGE = age.range, WAGE = pred$fit,
                          upper = conf.int[, 1], lower = conf.int[, 2])
ggplot() +
  theme_classic() +
  ggtitle("4th Degreee Polynomial") +
  theme(plot.title = element_text(hjust = 0.5)) +
  geom_point(data = Wage, aes(x = age, y = wage), size = 0.05) +
  geom_line(data = predictions, aes(x = AGE, y = WAGE), col = "blue") +
  geom_ribbon(data = predictions, aes(x = age.range, ymin = lower, ymax = upper), col = 'grey', alpha =
  xlab("Age") + ylab("Wage")
```



Now that we know how to fit a polynomial regression we can think about what is the best degree polynomial for our data. We will determine this by fitting a bunch of polynomials from linear to degree 5 and then performing analysis of variance (ANOVA) to test the null hypothesis that a model is sufficient to explain the data. The alternative hypothesis would be that a more complex model does a better job. We can use the `anova()` function to sequentially compare more and more complex models.

```
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-values from the ANOVA indicate that we can reject the null hypothesis when compare model 1 to model 2 and model 2 to model 3. This means that both the linear and quadratic models are not sufficient to explain the data. The p-value for the comparison of model 3 and 4 is almost significant so we could say that a 3rd or 4th degree polynomial is sufficient to explain the data. The p-value for the comparison of model 4 and 5 is not significant so there is no justification for using a 5th degree polynomial in this case.

An alternative to the anova method is to use cross-validation. ***Describe the steps you would take to perform cross-validation in this case to choose the degree of the polynomial.***

Step Function

We will now try to fit a step function to the data in order to again predict `wage` based on `age`. We use the `cut()` function to split the data into a specified number equal ranges.

```
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
```

The intercept coefficient can be interpreted as the average salary for individuals `age` < 33.5. The remaining coefficients are the average additional salary for those in the other age groups.

Make wage predictions for a range of ages and plot the results including the 95% confidence intervals.

Splines

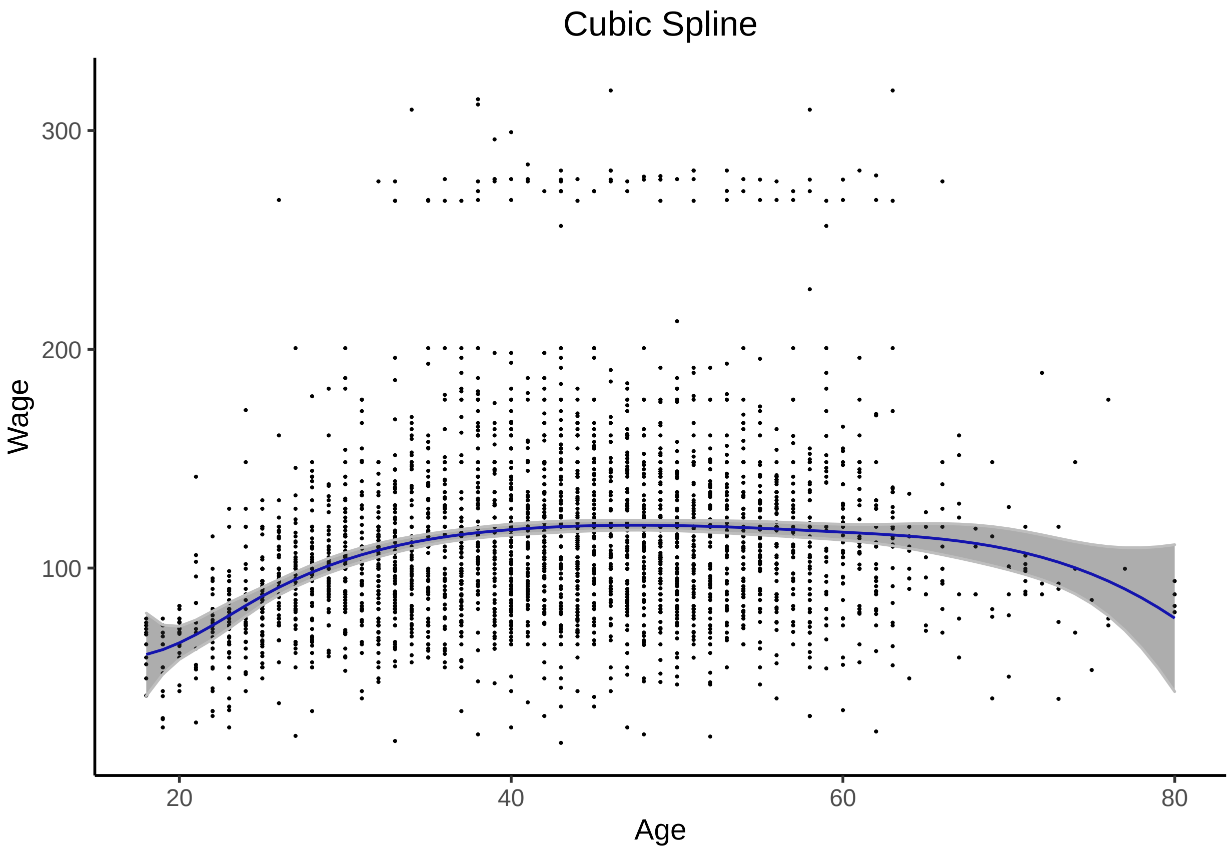
We want to fit a regression spline with `wage` as the response and `age` as the predictor. We will fit regression splines using the `splines` library. The `bs()` function can generate a variety of functions for splines with a specified set of knots. The default spline is cubic although we can change this with the argument `degree`.

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

We can make predictions on our age range we defined previously using the fitted spline and get the standard errors as well. Then plottinh the data, the fit, and the confidence intervals is the same as before.

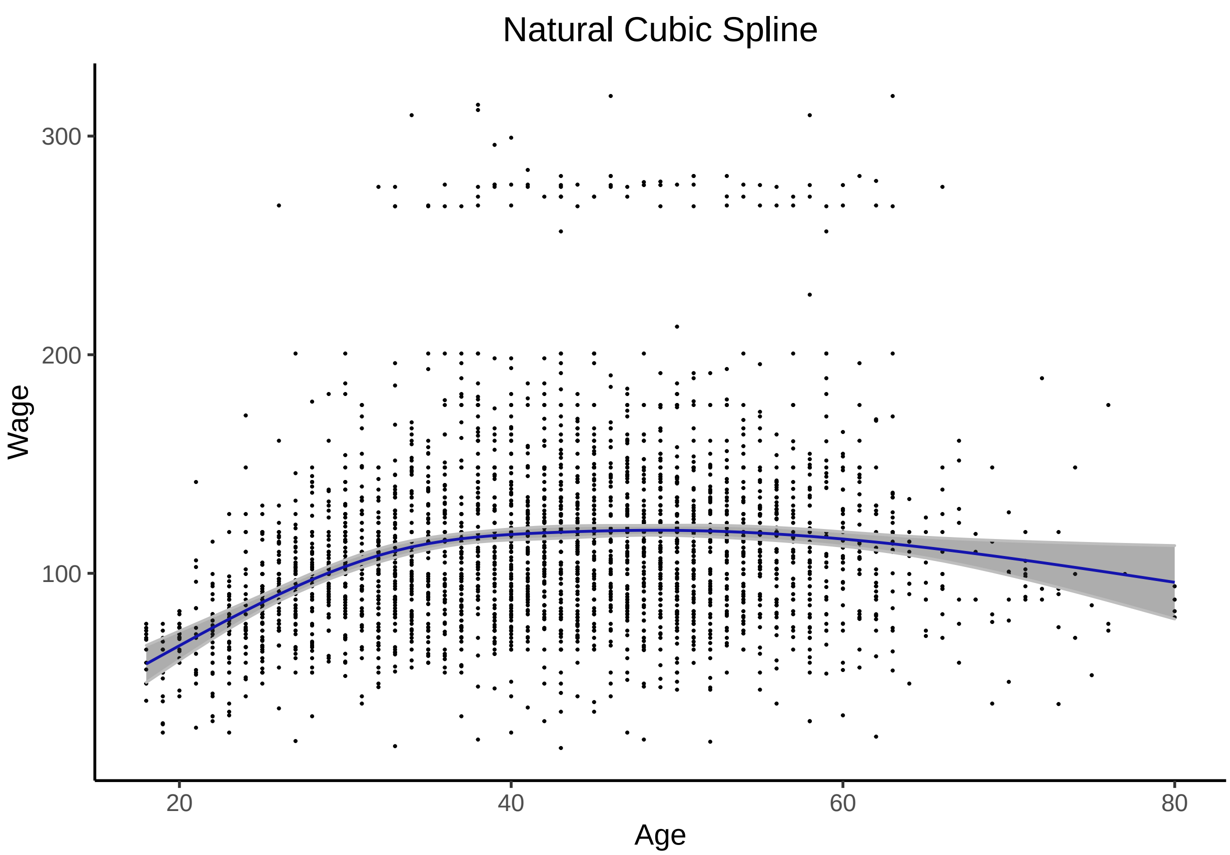
```
pred <- predict(spline.fit, newdata = list(age = age.range), se = TRUE)
conf.int <- cbind(pred$fit + 2 * pred$se.fit, pred$fit - 2 * pred$se.fit)
predictions <- data.frame(AGE = age.range, WAGE = pred$fit,
                          upper = conf.int[, 1], lower = conf.int[, 2])
ggplot() +
  theme_classic() +
  ggtitle("Cubic Spline") +
  theme(plot.title = element_text(hjust = 0.5)) +
```

```
  geom_point(data = Wage, aes(x = age, y = wage), size = 0.05) +
  geom_line(data = predictions, aes(x = AGE, y = WAGE), col = "blue") +
  geom_ribbon(data = predictions, aes(x = age.range, ymin = lower, ymax = upper), col = 'grey', alpha =
  xlab("Age") + ylab("Wage")
```



We could instead fit a natural spline using the `ns()` function. We could also try specifying the degrees of freedom we would like.

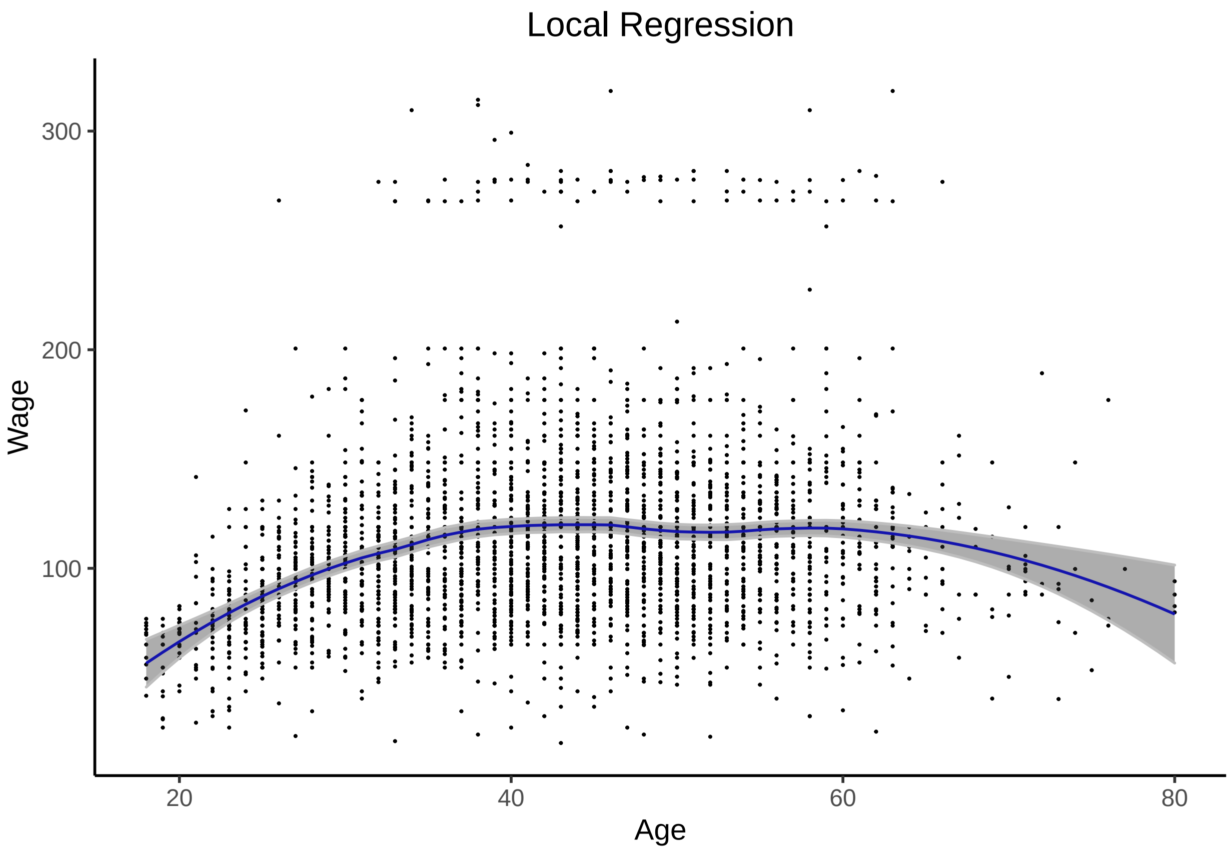
```
nspline.fit <- lm(wage ~ ns(age, df = 4), data = Wage)
# Make predictions
pred <- predict(nspline.fit, newdata = list(age = age.range), se = TRUE)
conf.int <- cbind(pred$fit + 2 * pred$se.fit, pred$fit - 2 * pred$se.fit)
predictions <- data.frame(AGE = age.range, WAGE = pred$fit,
                          upper = conf.int[, 1], lower = conf.int[, 2])
# Plot the natural spline
ggplot() +
  theme_classic() +
  ggtitle("Natural Cubic Spline") +
  theme(plot.title = element_text(hjust = 0.5)) +
  geom_point(data = Wage, aes(x = age, y = wage), size = 0.05) +
  geom_line(data = predictions, aes(x = AGE, y = WAGE), col = "blue") +
  geom_ribbon(data = predictions, aes(x = age.range, ymin = lower, ymax = upper), col = 'grey', alpha =
  xlab("Age") + ylab("Wage")
```



Local Regression

We can perform local regression using the `loess()` function with a chosen `span`.

```
local.fit <- loess(wage ~ age, span = 0.5, data = Wage)
# Make predictions
pred <- predict(local.fit, newdata = data.frame(age = age.range), se = TRUE)
conf.int <- cbind(pred$fit + 2 * pred$se.fit, pred$fit - 2 * pred$se.fit)
predictions <- data.frame(AGE = age.range, WAGE = pred$fit,
                          upper = conf.int[, 1], lower = conf.int[, 2])
# Plot the local regression
ggplot() +
  theme_classic() +
  ggtitle("Local Regression") +
  theme(plot.title = element_text(hjust = 0.5)) +
  geom_point(data = Wage, aes(x = age, y = wage), size = 0.05) +
  geom_line(data = predictions, aes(x = AGE, y = WAGE), col = "blue") +
  geom_ribbon(data = predictions, aes(x = age.range, ymin = lower, ymax = upper), col = 'grey', alpha =
  xlab("Age") + ylab("Wage")
```



Redo the fitting and plotting but this time choose a smaller span. What differences do you see? Explain why choosing a smaller span causes this.

Generalised Additive Models

We will fit a GAM to predict `Wage` using the predictors `year`, `age`, and `education`. The quantitative predictors will be a natural spline functions and `education` will naturally be fit with a step function.

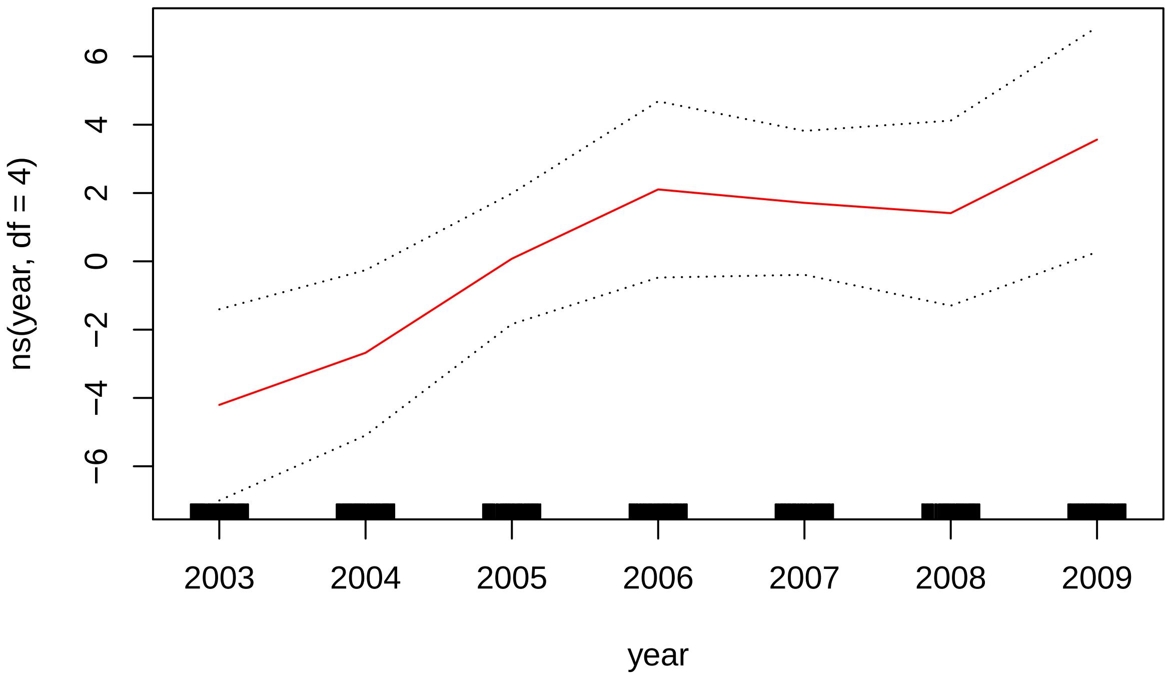
```
gam.fit <- lm(wage ~ ns(year, df = 4) + ns(age, df = 5) + education, data = Wage)
```

We can use our GAM to make predictions on the whole training set.

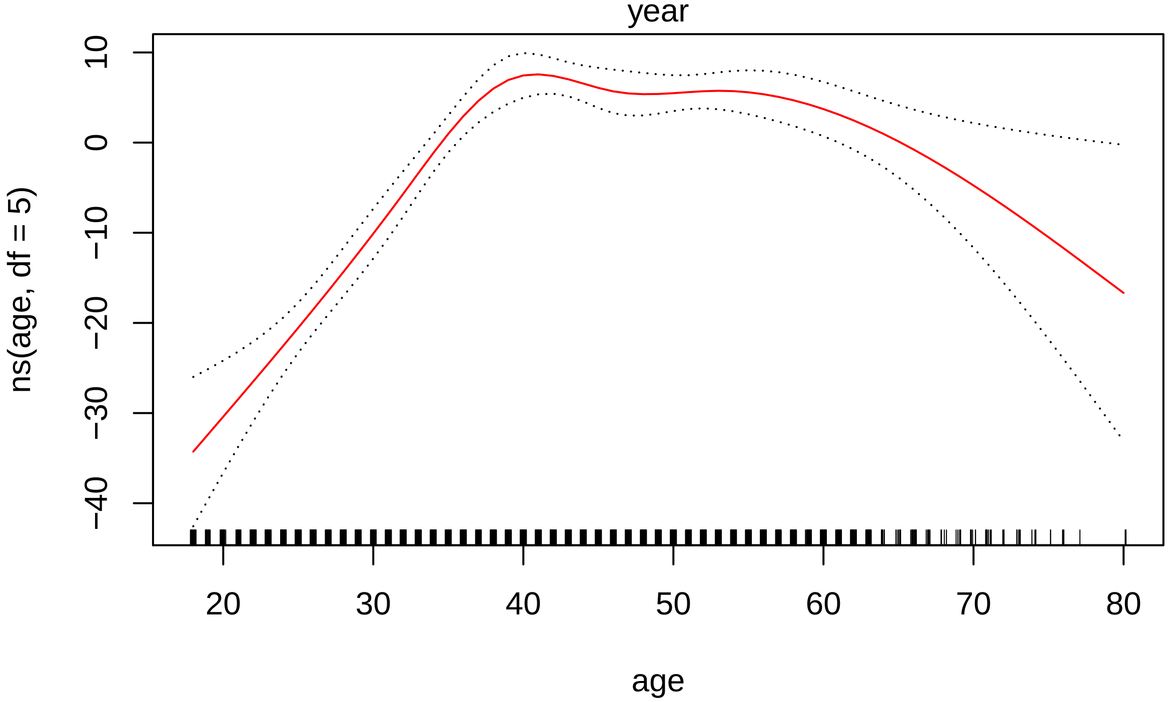
```
pred <- predict(gam.fit, newdata = Wage)
```

We can use a special plotting function `plot.Gam()` from the `gam` library to make plots to summarise GAMs. Each plot shows the fitted relationship between one of the predictors and the response `wage`

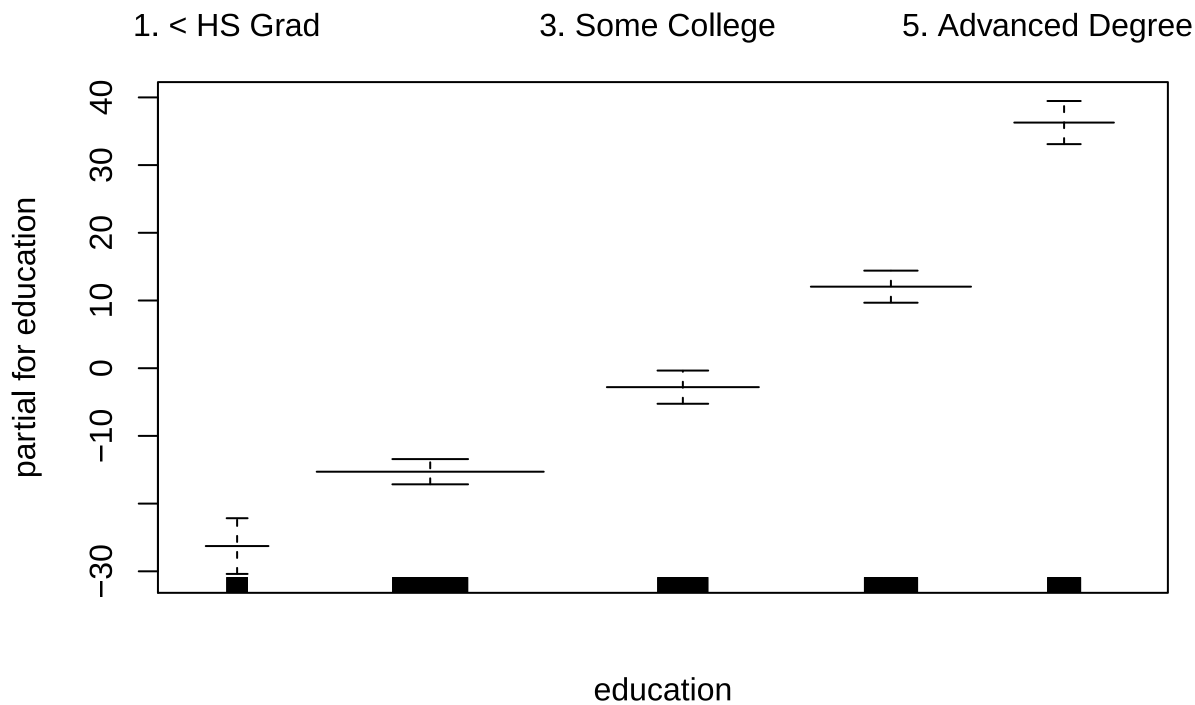
```
plot.Gam(gam.fit, se = TRUE, col = 'red')
```



ns(year, df = 4)
6
4
2
0
−2
−4
−6
2003
2004
2005
2006
2007
2008
2009
year



ns(age, df = 5)
10
0
−10
−20
−30
−40
20
30
40
50
60
70
80
age



We can see from the plot that wage is increasing somewhat linearly with year. ***Use an ANOVA test to determine which of these three models is the best:***

- The same GAM as before except exclude `year`.
- The same GAM as before except it instead has a linear function of `year` (written simply `+ year`)
- The GAM we have already fit.

Recall that the models must be supplied to the `anova()` function in order of increasing complexity.

Once you have determined which model is the best, plot the relationships as we did before.

Fitting GAMs using the `gam()` function instead of `lm()` allows us more flexibility in the choice of our building block functions and will yield the same result otherwise. The `lo()` function can be used to fit local regression as one of the functions in a GAM.

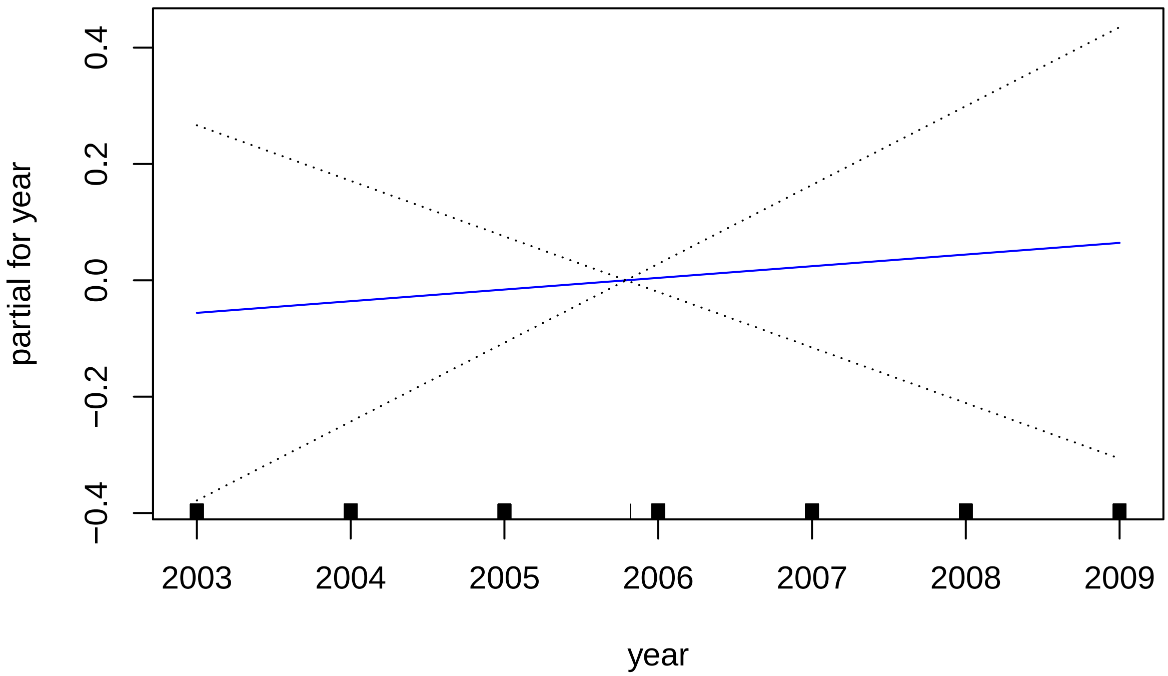
```
lo.gam.fit <- gam(wage ~ ns(year, df = 4) + lo(age, span = 0.7) + education, data = Wage)
```

The `lo()` function can also be use to create interaction terms in the GAM such as between `year` and `age`.

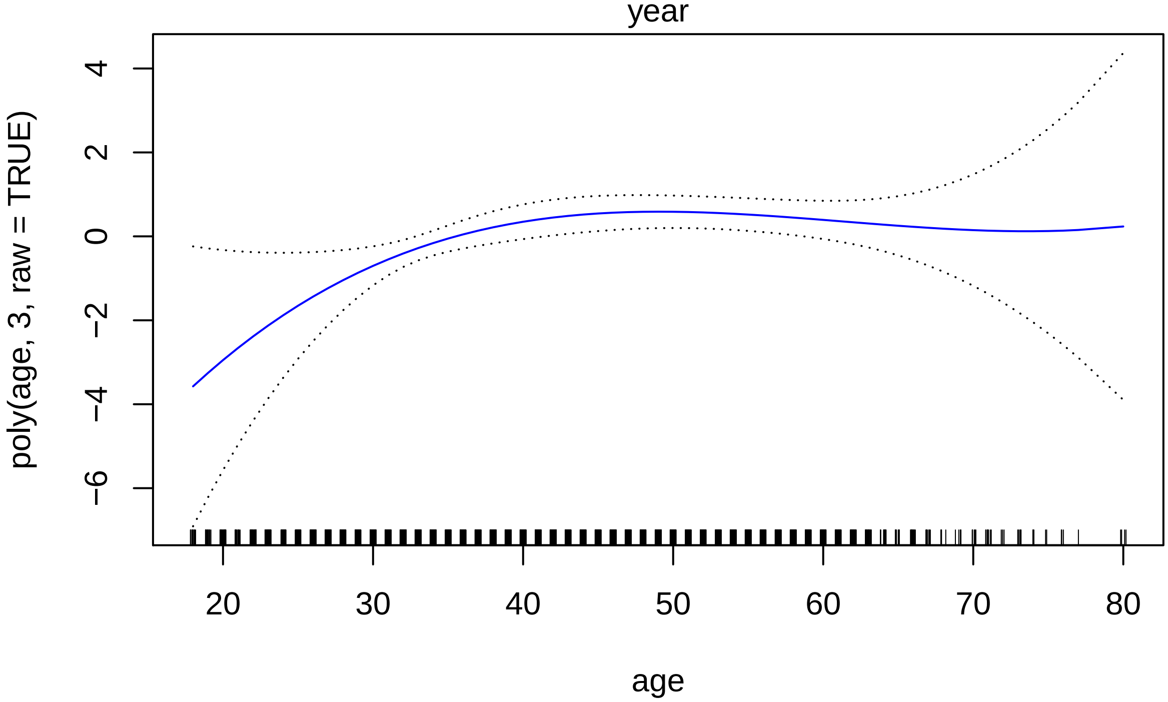
```
lo.gam.fit2 <- gam(wage ~ lo(year,age, span = 0.2) + education, data = Wage)
```

We can fit a logistic regression GAM so we can predict whether wages are greater than 250. We use the `I()` function and set the argument `family = binomial`.

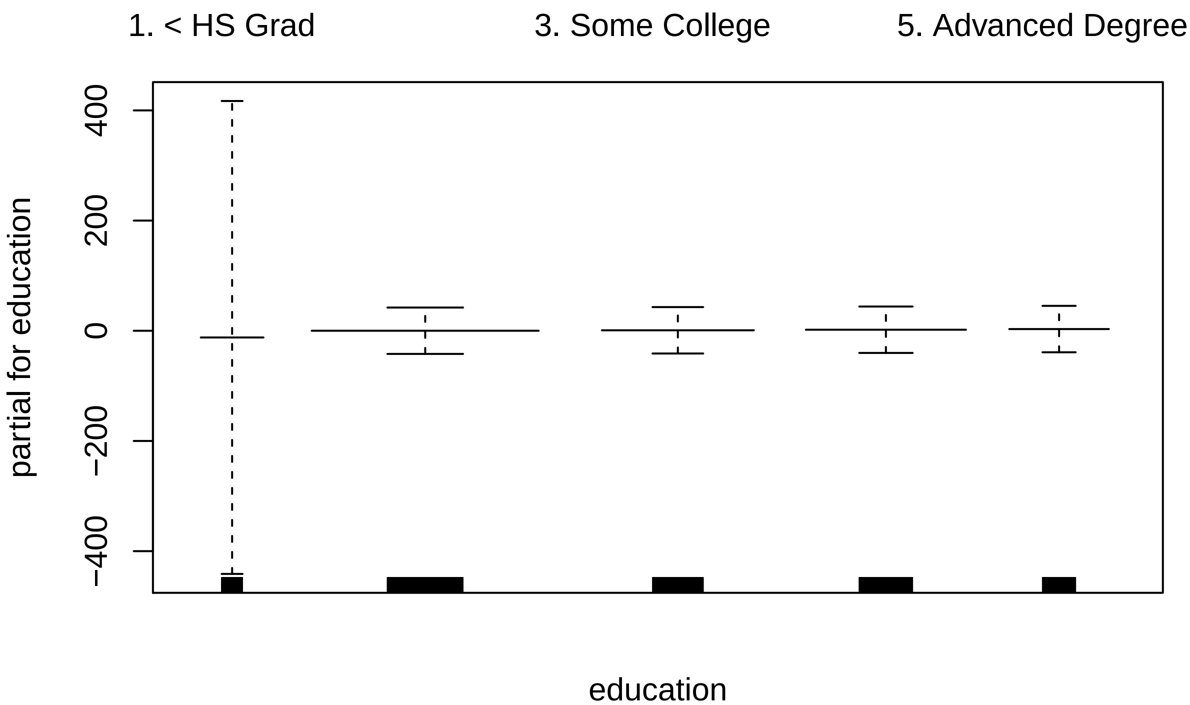
```
logis.gam.fit <- gam(I(wage > 250) ~ year + poly(age, 3, raw = TRUE) + education,
                     family = binomial, data = Wage)
plot(logis.gam.fit, se = TRUE, col = "blue")
```



partial for year
0.4
0.2
0.0
−0.2
−0.4
2003
2004
2005
2006
2007
2008
2009
year



poly(age, 3, raw = TRUE)
4
2
0
−2
−4
−6
20
30
40
50
60
70
80
age



These exercises were adapted from : James, Gareth, et al. An Introduction to Statistical Learning: with Applications in R, 2nd ed., Springer, 2021.