Smoothing

Spline modelling

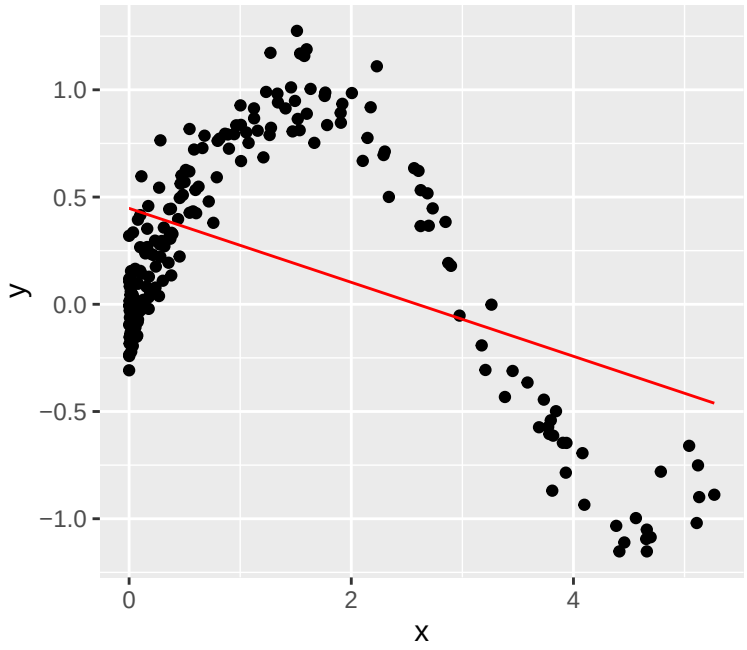
In this section we will look at some extensions of the linear model:

- Loess smoothing
- Regression splines
- Smoothing splines
- Generalized additive models

Polynomial regression

Consider some fake data.

```
set.seed(123)
n <- 200
x <- runif(n,0,1.75)^3
fx <- sin(x)
y <- fx + rnorm(100, sd = .15)
d <- data.frame(y = y, x = x)

f1 <- lm(y ~ x, data=d)

d %>%
  ggplot(aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(f1)),
            color = "red")
```



We fit a straight line to this:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

where ϵ_i is assumed to be iid Normal(0, σ^2)

The data has curvature which is not captured by this fit, ie the assumptions do not hold.

We could fit a quadratic curve

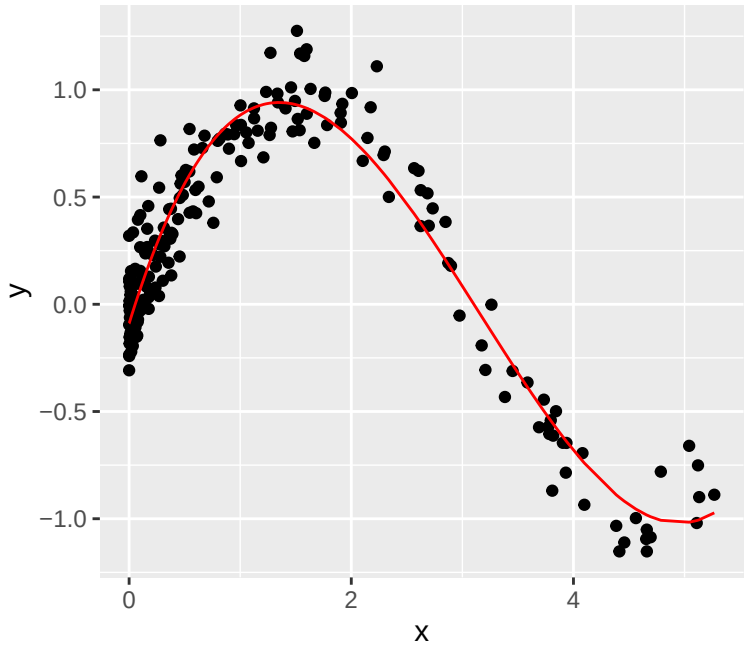
$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \epsilon_i$$

or a cubic

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \epsilon_i$$

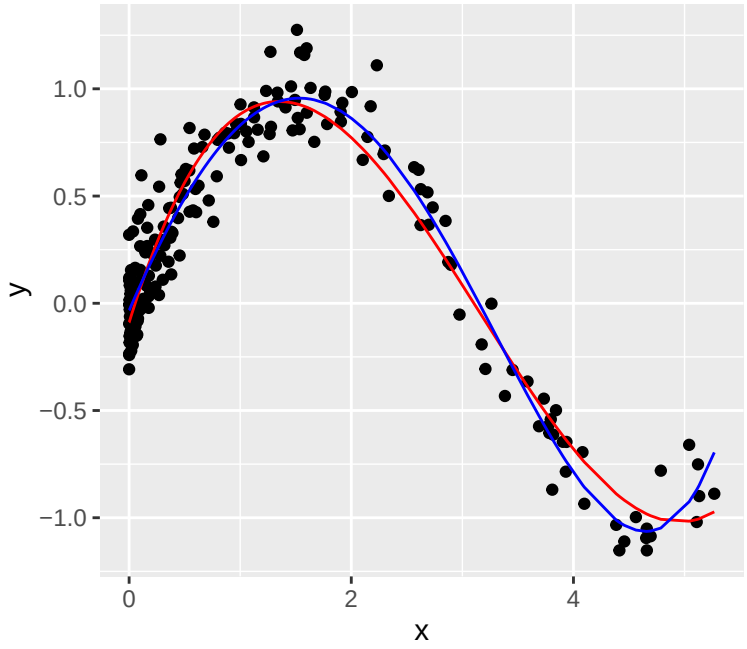
Here is the cubic fit:

```
f3 <- lm(y ~ x + I(x^2) + I(x^3), data = d)


d %>%
  ggplot(aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(f3)),
            color = "red")
```



compared with a quartic:

```
f4 <- lm(y ~ x + I(x ^ 2) + I(x ^ 3)+I(x ^ 4), data = d)


d %>%
  ggplot(aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(f3)),
            color = "red") +
  geom_line(aes(y = fitted(f4)),
            color = "blue")
```



```
mean(residuals(f3)^2)

[1] 0.0251

mean(residuals(f4)^2)

[1] 0.0208
```

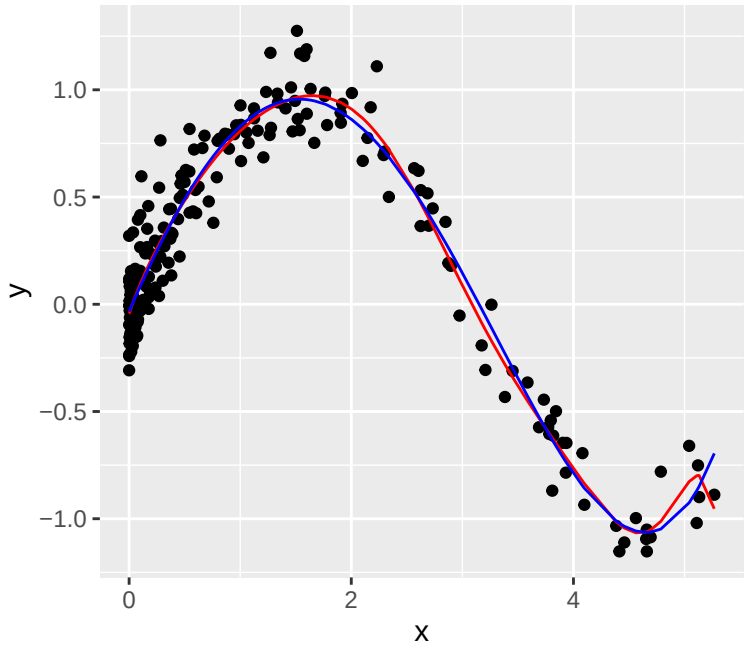
We can use an F-test to compare the two fits using

```
anova(f3, f4)

Analysis of Variance Table


Model 1: y ~ x + I(x^2) + I(x^3)
Model 2: y ~ x + I(x^2) + I(x^3) + I(x^4)
  Res.Df  RSS Df Sum of Sq    F  Pr(>F)
1    196 5.02
2    195 4.15  1     0.866 40.6 1.3e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This tells us that the x^4 term is significant in the fit.

We could keep adding higher powers of x

```
f10 <- lm(y ~ poly(x,10), data = d)


d %>%
  ggplot(aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(f10)),
            color = "red") +
  geom_line(aes(y = fitted(f4)),
            color = "blue")
```




```
mean(residuals(f10)^2)

[1] 0.02
```

The red curve is probably too flexible, see the dip at the end.

```
anova(f4, f10)

Analysis of Variance Table


Model 1: y ~ x + I(x^2) + I(x^3) + I(x^4)
Model 2: y ~ poly(x, 10)
  Res.Df  RSS Df Sum of Sq   F Pr(>F)
1    195 4.15
2    189 3.99  6     0.164 1.3   0.26
```

This F-test tells us that f10 is not significantly better that f4.

Polynomial regression models are special cases of the **basis function** approach. We have a family of functions $b_1(), b_2(), b_3(), ...$ that can be applied to a variable X. Then we fit a linear model:

$$y_i = \beta_0 + \beta_1 b_1(x_1) + \beta_2 b_2(x_i) + ... + \epsilon_i.$$

The coefficients can be estimated in the usual way and all the inference tools from linear

models are available.

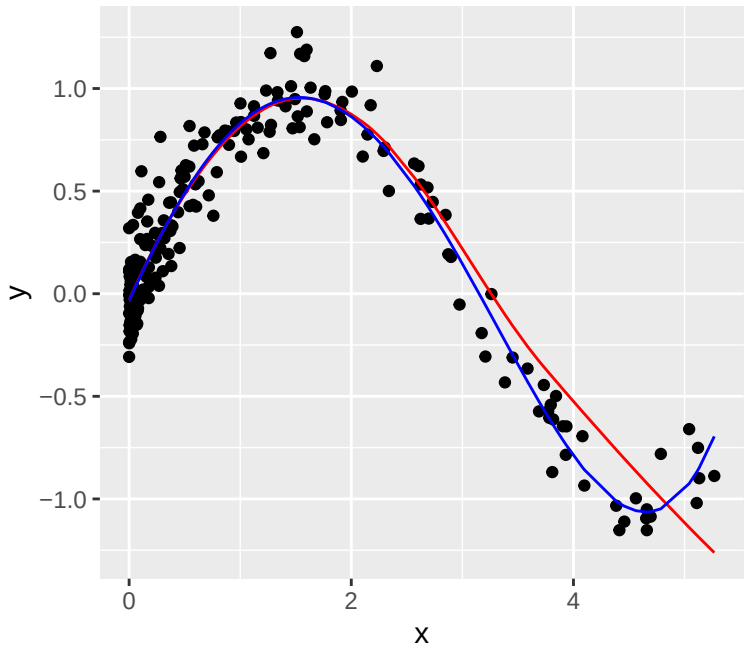
The basis functions are fixed and known (i.e. selected ahead of time). For polynomial regression, the basis functions are : $b_j(x_i) = x_i^j$.

Many alternatives are possible though, regression splines, wavelets, Fourier series etc.

Local regression with loess

Local regression is a very different approach to modelling. Here is the result of loess on this data.

```
flo <- loess(y ~  x, data = d)


d %>%
  ggplot(aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(flo)),
            color = "red") +
  geom_line(aes(y = fitted(f4)),
            color = "blue")
```



```
mean(residuals(flo)^2)

[1] 0.0292
```

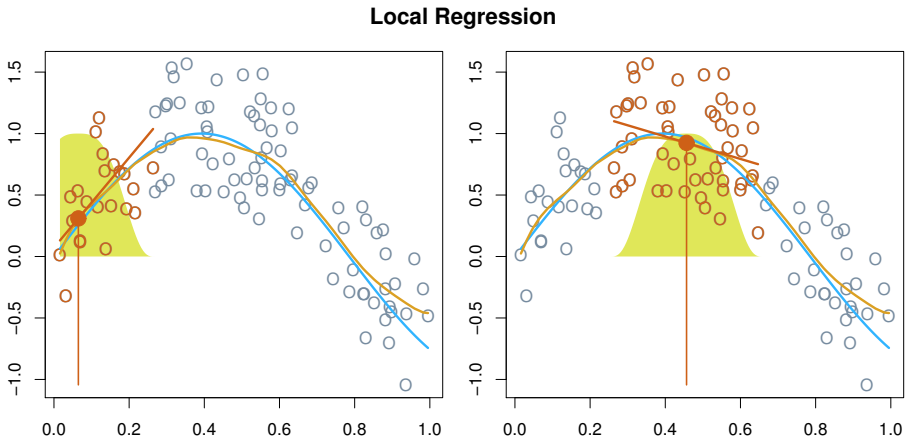
For this set of data loess (in red) does not do that well. There is clear bias on the right hand side.

The loess algorithm does the following: To produces the fitted value at x_0

1. Find the k observations whose x values are closest to x_0
2. Assign each of the k observations a weight w_{i0} according to their distance from x_0. The farthest point has a weight of 0, the closes point has the highest weight. All other points get a weight of 0.
3. Fit a regression to y and x using the weights w_{i0}.
4. The fitted value at x_0 provides $\hat{f}(x_0)$.

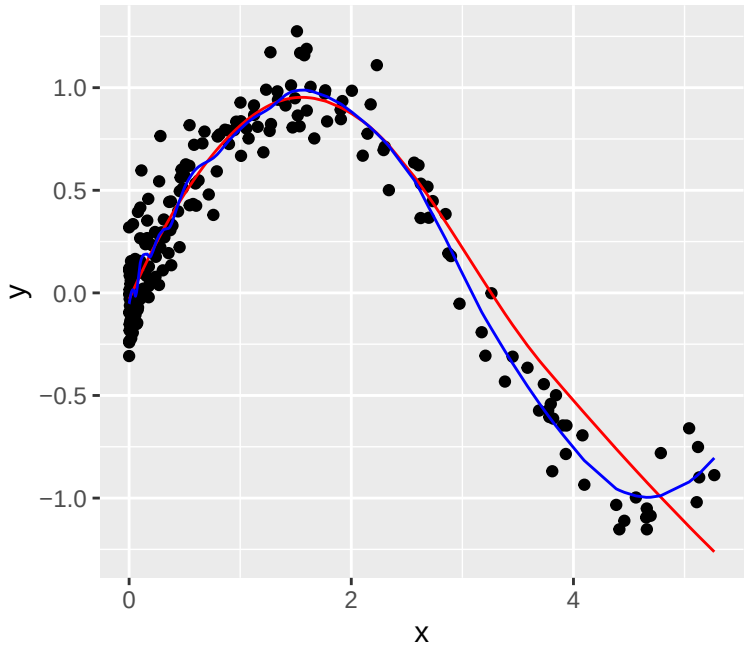
The loess algorithm takes the span s as input, then $k = s \times n$. The default span is 0.75.

The regression calculated at step 3 could be linear, constant or quadratic. The default is quadratic.



This figure illustrates the loess process. The true curve is blue, the loess curve is red. We can see from the description of the algorithm why loess is not working so well on the right hand side. The x values are sparse, so the k nearest neighbours are further away. Next we try using a lower span:

```
flo1 <- loess(y ~  x, data = d, span = .2)


d %>%
  ggplot(aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(flo)),
            color = "red") +
  geom_line(aes(y = fitted(flo1)),
            color = "blue")
```




```
mean(residuals(flo1)^2)

[1] 0.0192
```

This produces a curve which follows the data more closely but might be too wiggly.

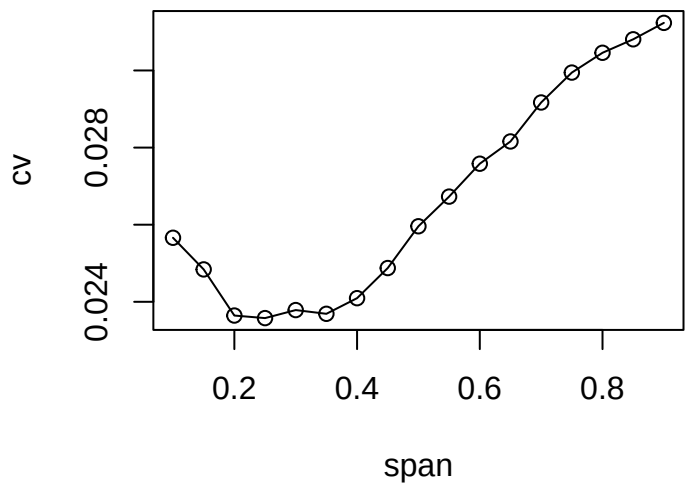
Probably for this data we could use a span that decreases from left to right.

Cross validation could be used to choose the best span.

```
k <- 5
fold <- sample(k, nrow(d), replace = TRUE)
```

For each span from .1 to .9 we can calculate the CV test error:

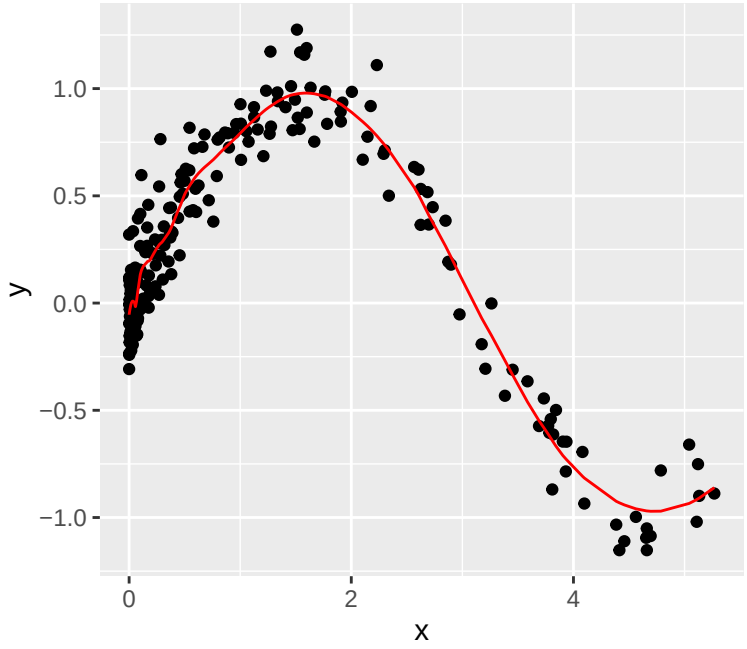
```
mse <- vector(length = k)
span <- seq(.1, .9, by = .05)
cv <- vector(length = length(span))

for(j in 1:length(span)) {
  for(i in 1:k) {
    foldi <- d[fold == i,]
    foldOther <- d[fold != i,]
    f <- loess(y ~ x, data = foldOther, span = span[j])
    pred <- predict(f, foldi)
    mse[i] <- mean((pred - foldi$y)^2, na.rm = TRUE) # MSEi
  }
  cv[j]<- mean(mse)
}

plot(span, cv)
lines(span,cv)
```




```
span[which.min(cv)] # produces the lowest CV

[1] 0.25
```

Refit with the best span and plot the result, which looks much better than before:

```
flo <- loess(y ~  x, data = d, span=span[which.min(cv)])


d %>%
  ggplot(aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(flo)),
            color = "red")
```

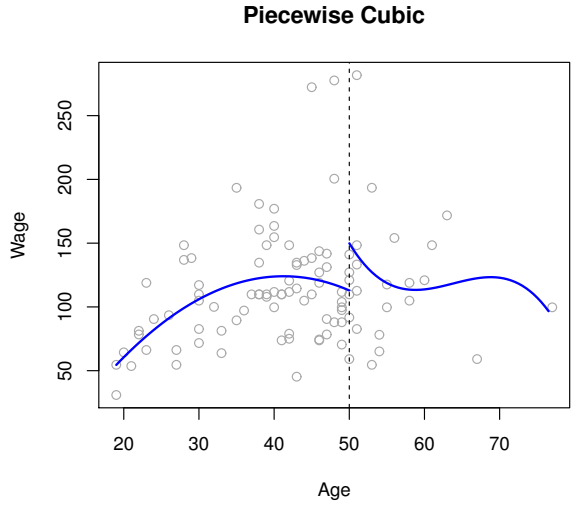



```
mean(residuals(flo)^2)

[1] 0.0199
```

Regression splines

One way to make polynomial regression more flexible is to fit separate polynomial regressions for different regions of x.

Here is an example of fitting two cubics:



There is a cubic fit to data with $x < 50$ and another for $x \geq 50$.

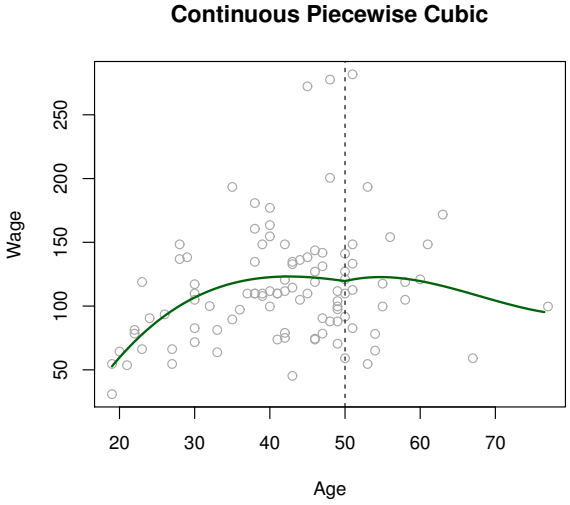
As each polynomial uses 4 parameters, overall 8 parameters are used.

The split point $x = 50$ is called a **knot**.

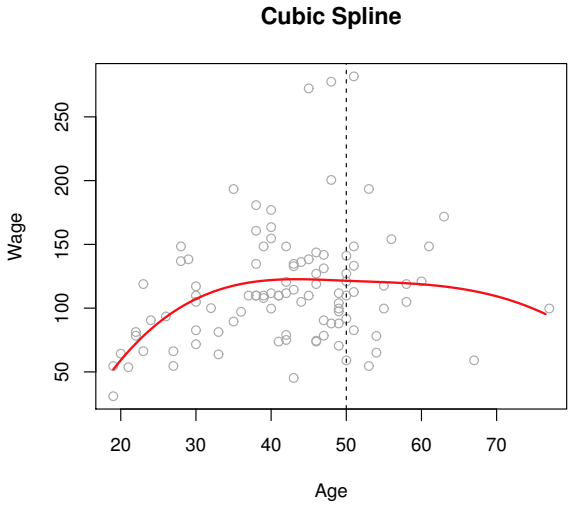
Using more knots makes the fit more flexible.

The main downside of this approach is that the fit is discontinuous as $x = 50$ which is not a good idea.

We could require the two cubic polynomials to be continuous at $x = 50$. This gives the following result:



Going even further, require the two polynomials to have continuous first and second derivatives. This gets rid of the kink and makes the curve smoother:



Even though there are two cubics the continuity constraints reduced the number of parameters from 8 to 5. This is the so-called regression spline.
In general, a cubic spline with K knots uses $K + 4$ parameters.

The general definition of a degree d spline is that it is a degree d polynomial, with continuity in derivatives up to degree $d - 1$ at each knot.

A cubic regression spline with K knots can be written as:

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \ldots..\beta_{K+3} b_{K+3}(x_i) + \epsilon_i$$

As usual β_i are the parameters of the cubic spline.
The b_j are the basis functions.

For a cubic, the basis functions are:

- basis for a cubic polynomial: x, x^2, x^3
- For a knot at $x = \xi$, $b(x, \xi) = (x - \xi)^3_+$, i.e. $b(x, \xi) = (x - \xi)^3$ if $x > \xi$ and $b(x, \xi) = 0$ otherwise.

Thus $K + 3$ basis functions are needed for K knots.

We could rewrite the model with K knots as:

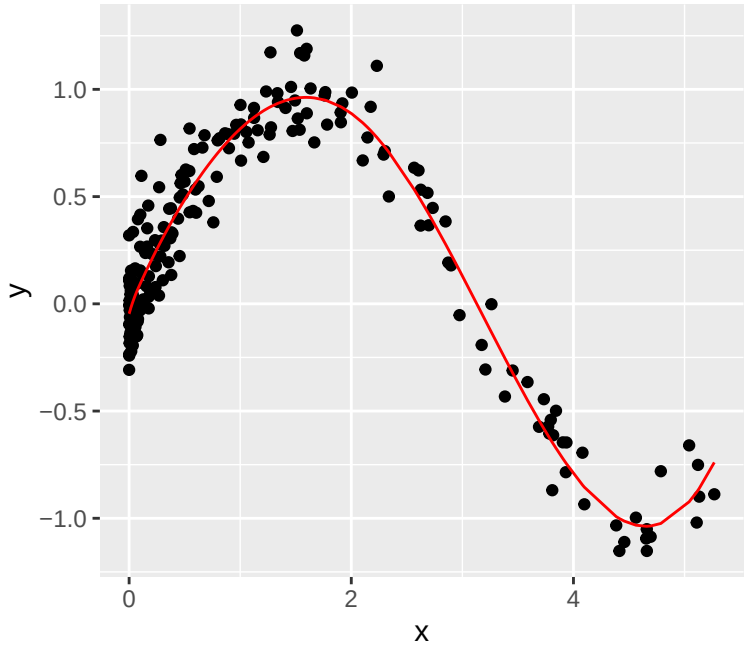
$$y_i = \sum_{j=0}^{3} \beta_j x_i^j + \sum_{k=1}^{K} \beta_{k+3} (x_i - \xi_k)^3_+ + \epsilon_i$$

The basis representation looks just like a linear model, the calculation is just linear multiple regression.

Fitting a regression spline in R

With the same data as above, we will fit a regression spline with 3 knots. By default the function uses degree 3 (cubic) so this means the fit uses 6 basis functions.
By default the 3 knots are positioned at the 25th, 50th and 75th percentiles of x.

```
library(splines)
f1 <- lm(y ~ bs(x, df = 6), data = d) # df is the number of basis functions


d %>%
  ggplot(aes(x = x, y = y)) +
  geom_point() +
  geom_line(aes(y = fitted(f1)),
            color = "red")
```



The knots are positioned at

```
attr(bs(d$x, df = 6), "knots")

  25%   50%   75%
0.108 0.601 2.114
```

The `bs()` call constructs what is called a B-spline X matrix necessary for `lm`.

As f1 is just a linear model, all the usual plots and summaries for regression apply:

```
f1


Call:
lm(formula = y ~ bs(x, df = 6), data = d)

Coefficients:
   (Intercept)  bs(x, df = 6)1  bs(x, df = 6)2  bs(x, df = 6)3  bs(x, df = 6)4
       -0.0465          0.0670          0.3008          0.9550          1.4049
bs(x, df = 6)5  bs(x, df = 6)6
       -1.7685         -0.6930

summary(f1)


Call:
lm(formula = y ~ bs(x, df = 6), data = d)

Residuals:
    Min      1Q  Median      3Q     Max
-0.2982 -0.1046 -0.0164  0.0850  0.4906

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)     -0.0465     0.0360   -1.29     0.20
bs(x, df = 6)1   0.0670     0.0620    1.08     0.28
bs(x, df = 6)2   0.3008     0.0515    5.84  2.2e-08 ***
bs(x, df = 6)3   0.9550     0.0696   13.72  < 2e-16 ***
bs(x, df = 6)4   1.4049     0.1083   12.97  < 2e-16 ***
bs(x, df = 6)5  -1.7685     0.1078  -16.40  < 2e-16 ***
bs(x, df = 6)6  -0.6930     0.0809   -8.57  3.4e-15 ***
---
```

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

Residual standard error: 0.146 on 193 degrees of freedom
Multiple R-squared:  0.934,Adjusted R-squared:  0.932
F-statistic:  459 on 6 and 193 DF,  p-value: <2e-16
```

We could try another regression spline where we specify the position of knots:

```
f2 <- lm(y ~ bs(x, knots = 1:3), data = d)
```

The fitted function looks similar to f1.

How many knots should the function use?

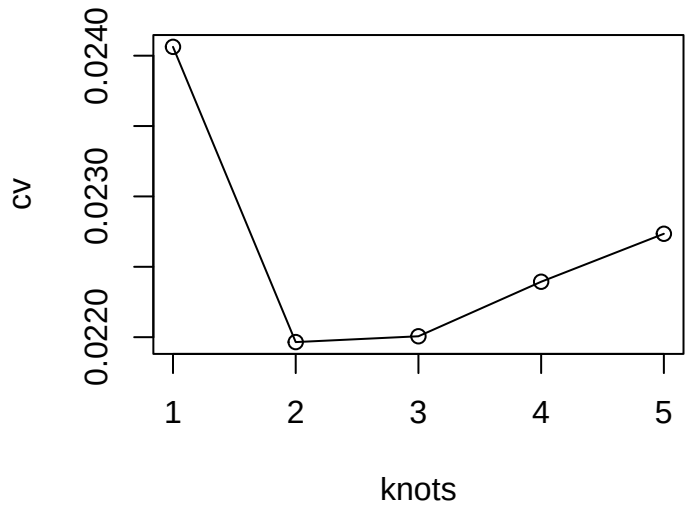
We can answer this by trying out different choices and picking the one that looks best. Cross validation gives a more objective answer:

```
k <- 5
fold <- sample(k, nrow(d), replace = TRUE)
fsize <- table(fold)
```

For each number of knots from 1 to 5 we can calculate the CV test error:

```
options(warn = -1)
mse <- vector(length = k)
knots <- 1:5
cv <- vector(length = length(knots))

for(j in 1:length(knots)) {
  for (i in 1:k) {
    foldi <- d[fold == i,]
    foldOther <- d[fold != i,]
    f <- lm(y ~ bs(x, df = knots[j] + 3), data = foldOther)
    pred <- predict(f, foldi)
    mse[i] <- mean((pred - foldi$y)^2, na.rm = TRUE) # MSEi
  }
  cv[j] <- weighted.mean(mse, fsize)
```

```
  # weighted as each fold not the same size.
}


plot(knots, cv)
lines(knots,cv)
```




```
knots[which.min(cv)] # produces the lowest CV

[1] 2
```

When we plot the 2-knot regression spline along with the original 3-knot fit, the results are similar.

```
f2 <- lm(y ~ bs(x, df = 5), data = d)
```

Natural splines

The regression splines that we have described (i.e. the so-called b - splines) often have high variance at outer range of the predictors.

A **natural spline** is a regression spline with additional boundary constraints: the function is linear for $x < \xi_{min}$ and $x > \xi_{max}$. It uses fewer parameters, K instead of $K + 4$.

For a natural spline with K knots, the model becomes:

$$y_i = \sum_{j=0}^{1} \beta_j x_i^j + \sum_{k=3}^{K} \beta_k \left(d_{k-2}(x_i) - d_{K-1}(x_i) \right) + \epsilon_i$$

where

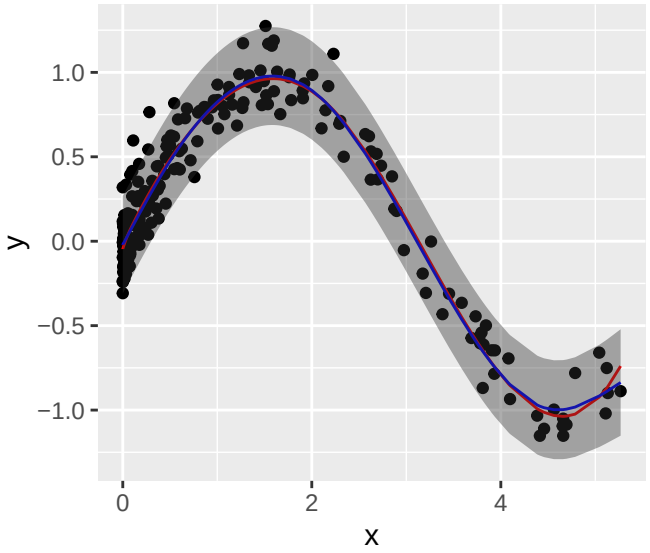
$$d_k(x_i) = \frac{(x_i - \xi_k)_+^3 - (x_i - \xi_K)_+^3}{\xi_K - \xi_k}.$$

In matrix form we can write:

$$\mathbf{y} = \mathbf{N}\boldsymbol{\beta}$$

where $\mathbf{N}$ is the design matrix whose columns are made up of the basis functions. At K knots, it is a matrix of K columns (the first column is a column of 1s) and n rows.

In R, for natural splines use ns() instead of bs().



Regression splines for logistic regression

Example:

Y = wage (in thousands) $>$ 250 (1) or not (0)

X = age

We could model this using logistic regression.

The model is

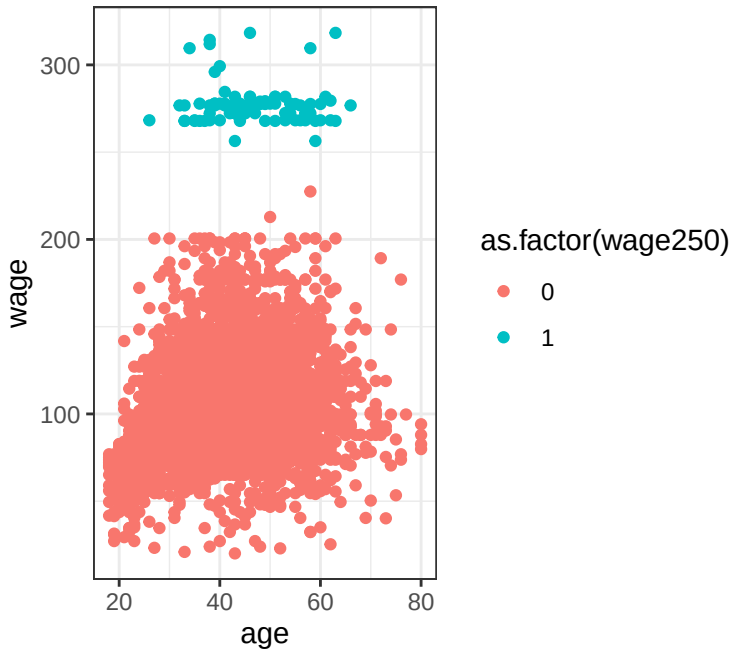
$$P(Y = 1|X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

or equivalently

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X$$

where where $p = P(Y = 1|X)$.

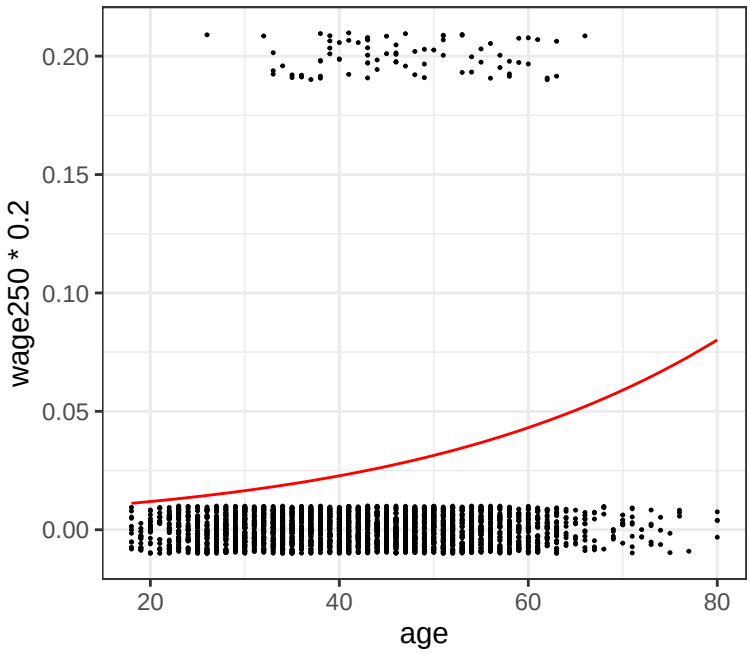
```
library(ISLR)


Wage$wage250 <- as.numeric(Wage$wage>250)


Wage %>%
  ggplot(aes(x = age, y = wage, colour = as.factor(wage250))) +
  theme_bw() +
  geom_point()
```




```
f1 <- glm(wage250 ~ age,
          family = "binomial",
          data = Wage)
```

We can plot the fit as

```
pred1 <- predict(f1, Wage, type = "response")


Wage %>%
  ggplot(aes(x = age, y = wage250 * .2)) +
  theme_bw() +
  geom_jitter(width = 0, height = 0.01, size = .2) +
  geom_line(aes(x = age, y = pred1),
            color = "red")
```

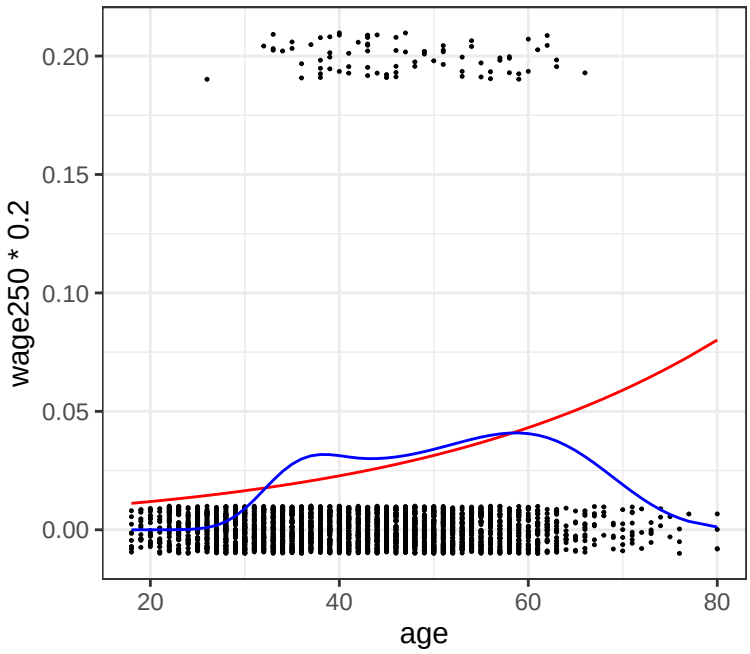


(Wage was multiplied by .2 to reduce the scale and improve visualisation only)

If we want to use a regression spline with K knots instead of logistic regression the model becomes

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 b_1(x) + \beta_2 b_2(x) + \ldots..\beta_{K+3} b_{K+3}(x)$$

```
f2 <- glm(wage250 ~ bs(age, 4),
          family = "binomial",
          data = Wage)

pred2 <- predict(f2, Wage, type = "response")

Wage %>%
  ggplot(aes(x = age, y = wage250 * .2)) +
  theme_bw() +
  geom_jitter(width = 0, height = 0.01, size = .2) +
  geom_line(aes(x = age, y = pred1),
            color = "red") +
  geom_line(aes(x = age, y = pred2),
            color = "blue")
```

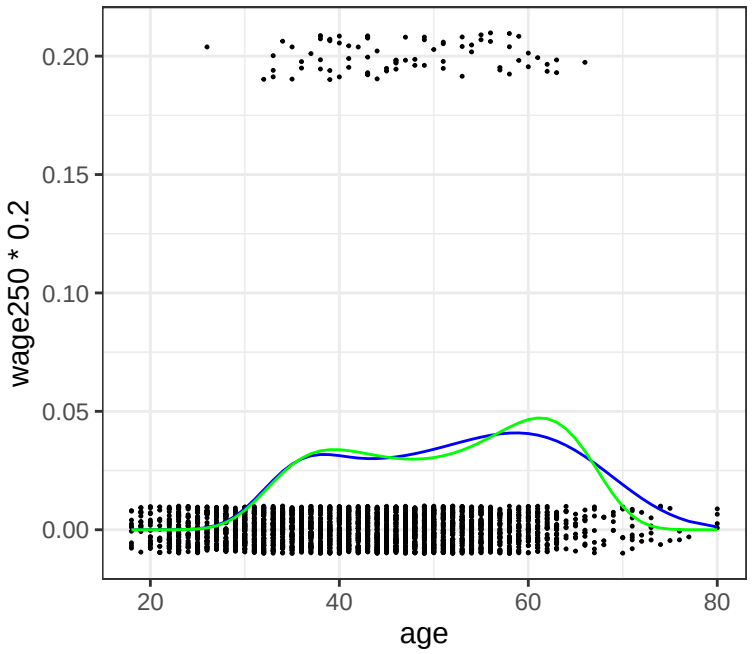


In this example the spline has just 1 knot, which has 4 degrees of freedom.

You can see from the plot that f2 fits the data much better. f1 can only go up or down, whereas for this data it is the middle age groups that have the highest probability of a wage over 250.

We could alternatively fit a quartic polynomial in age.

```
f3 <- glm(wage250 ~ poly(age, 4),
          family = "binomial",
          data = Wage)
pred3 <- predict(f3, Wage, type = "response")


Wage %>%
  ggplot(aes(x = age, y = wage250 * .2)) +
  theme_bw() +
  geom_jitter(width = 0, height = 0.01, size=.2) +
  geom_line(aes(x = age, y = pred2),
            color = "blue") +
  geom_line(aes(x = age, y = pred3),
            color = "green")
```



Regression splines versus polynomial regression

Regression splines often give better results than polynomials.

Polynomials need high degree to get flexible fits, whereas with regression splines more knots give greater flexibility.

High degree polynomials can show wild behaviour, especially near tails.

With regression splines extra knots can be used in regions where f seems to be changing rapidly, and fewer knots in areas where f is more stable.

Smoothing splines

Smoothing splines take a different approach to produce a spline from the regression splines.

When fitting a curve to a set of data, we would like to find a function $\hat{f} = g$ such that

$$g(x_i) \approx y_i$$

Equivalently, the squared residuals $(y_i - g(x_i))^2$ are small for all observations i suggesting

we should minimize the residual sum of squares

$$\text{SSE} = \sum_{1}^{n}(y_i - g(x_i))^2$$

(Note, I'm using g instead of $\hat{f}$ for ease of notation here.)
This criterion in minimised by setting $g(x_i) = y_i$. The resulting curve g interpolates the data.
This g would be way too flexible. Usually it would be wiggly and overfit the data.

Instead we would like a function g that produces small residual sum of squares and that is also **smooth**.

This can be achieved by finding g to minimize

$$\sum_{1}^{n}(y_i - g(x_i))^2 + \lambda \int g''(t)^2 dt \tag{1}$$

where $\lambda > 0$ can be selected by the user.

- Here g'' refers to the second derivative of the function g.
- The integrated second derivative $\int g''(t)^2 dt$ is a measure of the **roughness** of the function g.
- The quantity $\lambda \int g''(t)^2 dt$ penalises rough choices of g.
- If $\lambda = 0$ then the penalty term has no effect, g will just interpolated the data.
- The larger λ is the bigger the smoother the resulting g will be.
- As $\lambda \to \infty$, g becomes perfectly smooth, i.e. gives a linear fit.
- Term $\sum_1^n (y_i - g(x_i))^2$ is often called loss function.

The criterion (1) is defined on an infinite-dimensional function space. However, the function $g = \hat{f}$ minimising the criterion (1) above turns out to be a natural cubic spline with knots at the observed x_i. It is called a **smoothing spline**.
The smoothing spline is not the same as the natural regression spline discussed previously. I.e. we don't get the same spline if we take the basis function approach with knots at x_i. It is in fact a shrunken version version of such a spline.

Parameter λ controls the degree of shrinkage.

For a particular choice of lambda, the solution to the cubic spline can be written as:

$$\hat{\boldsymbol{\beta}}_\lambda = (\mathbf{N}^T\mathbf{N} + \lambda\boldsymbol{\Omega})^{-1}\mathbf{N}^T\mathbf{y}$$

where $\boldsymbol{\Omega}_{jk} = \int N_j''(t)N_k''(t)dt$.

The effective degrees of freedom is the sum of the diagonal elements of $\mathbf{N}(\mathbf{N}^T\mathbf{N}+\lambda\boldsymbol{\Omega})^{-1}\mathbf{N}^T$ matrix.

It is possible to show that as λ increases from 0 to ∞, the effective degrees of freedom df_λ decrease from n to 2.

The user can select λ.

Automatic choice is commonly done using leave one out cross-validation, which is where cross-validation is done leaving out one observation at a time (PRESS statistic),

In R, the `smooth.spline` function calculates smoothing splines.

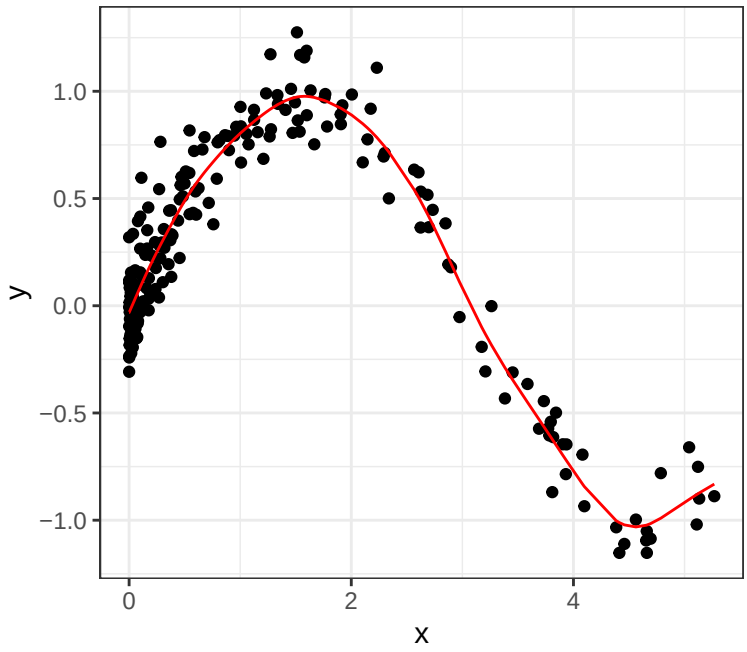
Going back to the fake data we used to demonstrate polynomial regression and regression splines:

```
f3 <- smooth.spline(x, y, cv = TRUE)
f3

Call:
smooth.spline(x = x, y = y, cv = TRUE)

Smoothing Parameter  spar= 1.5  lambda= 0.000274 (25 iterations)
Equivalent Degrees of Freedom (Df): 10.4
Penalized Criterion (RSS): 3.98
PRESS(l.o.o. CV): 0.0218
```

You can see the value of lambda used, which was chosen by cross-validation and equates to a degree of freedom of 10.37.

We can plot the fit using:

```
d %>%
  ggplot(aes(x = x, y = y)) +
  theme_bw() +
  geom_point() +
  geom_line(aes(y = fitted(f3)),
            color = "red")
```



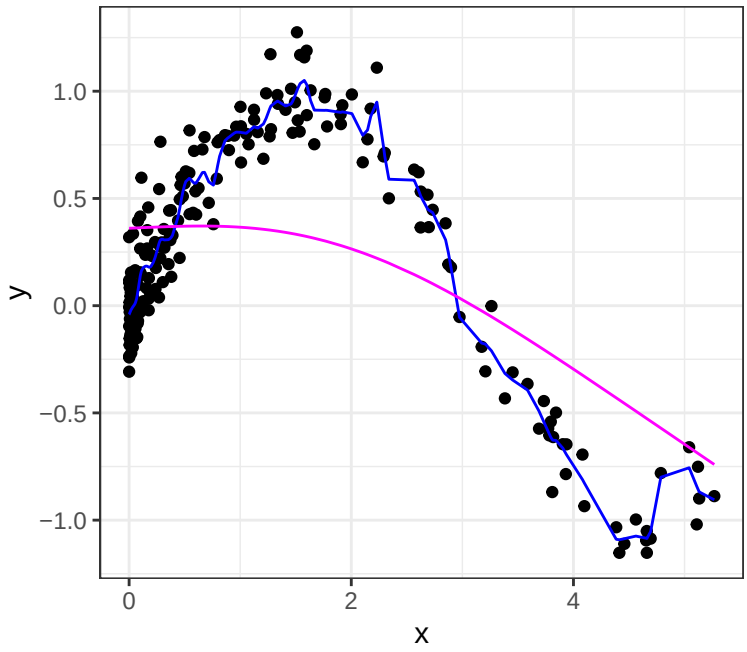
As you can see, it does well.

We can experiment with what happens if we increase or decrease the amount of smoothing. In R we can do this by specifying the parameter spar (related to lambda).

```
f4 <- smooth.spline(x, y, spar = 1.1, cv = TRUE) # decrease smoothing
f4

Call:
smooth.spline(x = x, y = y, spar = 1.1, cv = TRUE)

Smoothing Parameter  spar= 1.1  lambda= 3.53e-07
Equivalent Degrees of Freedom (Df): 41.5
Penalized Criterion (RSS): 3.26
PRESS(l.o.o. CV): 0.0248
```

```
f5 <- smooth.spline(x, y, spar = 2, cv = TRUE) # increase smoothing
f5

Call:
smooth.spline(x = x, y = y, spar = 2, cv = TRUE)

Smoothing Parameter  spar= 2  lambda= 1.12
Equivalent Degrees of Freedom (Df): 2.29
Penalized Criterion (RSS): 33.5
PRESS(l.o.o. CV): 0.172
```

```
d %>%
  ggplot(aes(x = x, y = y)) +
  theme_bw() +
  geom_point() +
  geom_line(aes(y = fitted(f4)),
            color = "blue")+
  geom_line(aes(y = fitted(f5)),
            color = "magenta")
```



For f5 (magenta) the effective df is 2.3. If the effective df were 2, the fitted function would be linear.

Generalized Additive Models

Dealing with multiple predictors in a flexible way

We have looked at smoothing methods for a single predictor.

In principle these methods extend to multiple predictors but often perform poorly beyond $p = 3$ or 4, because of the sparsity of data in higher dimensions.

The loess implementation in R accepts up to 4 predictors.
In this case the model is

$$y = f(x_1, x_2, x_3, \ldots, x_p) + \epsilon$$

Generalization of b-splines involves tensor product basis, so going from, say 1d to 2d with 1 knot and 4 basis functions increases to 16 basis functions etc.

One dimensional smoothing splines generalize to higher dimensions using regularization context with more general penalty functions and have finite dimensional solutions known as thin-plate splines.

A general class of problems (under which smoothing splines and thin plate splines fall under) has the form:

$$\min_{f \in H} \{L(y_i, f(x_i)) + \lambda P(f)\}$$

where $P(f)$ is a penalty function and H is a space of functions over which $P(f)$ is defined. H is infinite-dimensional, but solutions are finite - dimensional. An important subclass of the problems of the above form are generated by kernels. The loss function is optimized over $f(x)$ that have infinite-dimensional basis but the solution can be represented by n - dimensional basis (regardless of p). We will see an example of their use later in the module.

Multiple predictors - additive models

The linear regression model is

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p + \epsilon$$

A generalised additive model (GAM) extends this by allowing a function f_j for every predictor x_j.

The GAM is then

$$y = \beta_0 + f_1(x_1) + f_2(x_2) + \ldots f_p(x_p) + \epsilon$$

Each function f_j can be fit using any of the methods discussed for a single predictor:

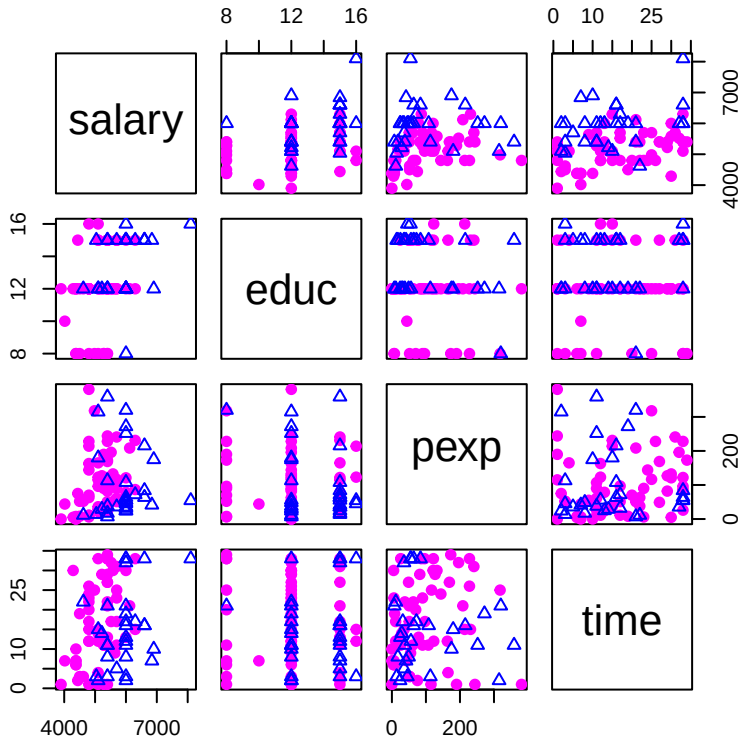
1. A polynomial in that predictor
2. A regression spline using bs() or ns()
3. A smoothing spline
4. Local regression: loess

Methods (1) and (2) are straightforward, because the model is just a regression with a few powers of x_j or basis functions included for each predictor.

Case study: salary data

The following salary data were used in 1978 in a sex discrimination lawsuit against a Chicago bank. The data gave beginning salaries and measures of job qualifications for 61 female and 32 male skilled entry-level clerical employees. The purpose was to compare the male and female salaries taking qualifications into account. If after adjusting for qualifications there is still a real Difference between the beginning salaries for males and females, then sex discrimination could justly be claimed. The variables are
sal : beginning salary
educ: years of education
pexp: previous experience in months
time: time of hire (months after Jan 1 1969)
female : gender (1 = female 0 = male)

```r
sal <- read.table("https://raw.githubusercontent.com/rafamoral/courses/main/intro_stat



cols <- c("blue", "magenta")[sal$gender + 1]
pchs <- c(2,19)[sal$gender+1]
sal$gender <- factor(ifelse(sal$gender == 1, "F", "M"))

pairs(sal[,-5], col = cols, pch = pchs)
```



The average salary of males is higher than that for females but this difference may not persist (or could be greater) when we adjust for the other predictors.

The plot for pexp suggests a transformation of pexp may be in order, but for now we go ahead without any transformations.

As one might expect, there are not strong predictor relationships here.

Linear model

$$\log(\text{sal}) = \beta_0 + \beta_1 \text{educ} + \beta_2 \text{pexp} + \beta_3 \text{time} + \beta_4 \text{genderM} + \epsilon$$

```
fit1 <- lm(log(salary) ~ . , data = sal)
summary(fit1)
```

```

Call:
lm(formula = log(salary) ~ ., data = sal)

Residuals:
     Min       1Q   Median       3Q      Max
-0.22436 -0.06179  0.00002  0.05454  0.21306

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 8.240911   0.061079  134.92  < 2e-16 ***
educ        0.016526   0.004604    3.59  0.00055 ***
pexp        0.000260   0.000107    2.43  0.01731 *
time        0.004254   0.000963    4.42  2.9e-05 ***
genderM     0.126730   0.021640    5.86  8.4e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.0925 on 86 degrees of freedom
Multiple R-squared:   0.5,Adjusted R-squared:  0.476
F-statistic: 21.5 on 4 and 86 DF,  p-value: 2.63e-12
```

The summary shows that all predictors are significant. The 95% confidence interval for gender is

```
confint(fit1, "genderM") # for log(sal)
```

```
         2.5 % 97.5 %
genderM 0.0837   0.17
```

```
exp(confint(fit1, "genderM")) # sal
```

```
        2.5 % 97.5 %
genderM  1.09   1.19
```

The residuals show a clear non-constant variance in the residuals vs fit plot.

The residuals versus pexp plot shows clear curvature, suggesting a need for transforming pexp.

```
library(gridExtra)
```

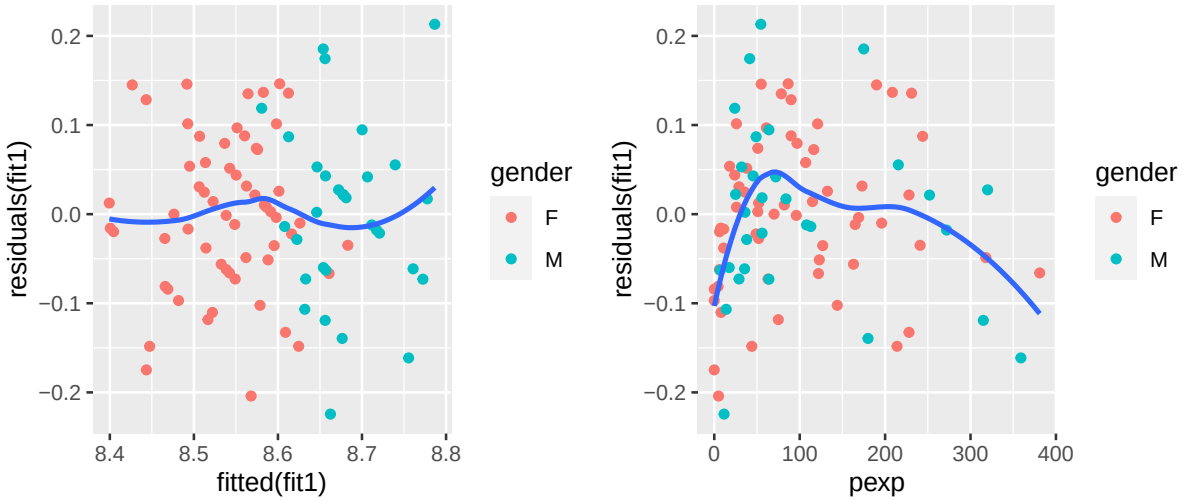
```
Attaching package:  'gridExtra'
The following object is masked from 'package:dplyr':
    combine
```

```
p1 <- ggplot(data=sal, aes(x=fitted(fit1), y= residuals(fit1))) +
  geom_point(aes(color=gender)) + geom_smooth(se=F)
```

```
p2 <- ggplot(data=sal, aes(x=pexp, y= residuals(fit1))) +
  geom_point(aes(color=gender)) + geom_smooth(se=F)
```

```
grid.arrange(p1,p2, ncol=2)
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Additive model

We will use splines for predictors pexp and time.

Gender is binary so no function is needed.

educ takes just a few values, so a cubic spline is not appropriate.

The model is

$$\log(\text{sal}) = \beta_0 + \beta_1 \text{educ} + \text{f}_2(\text{pexp}) + \text{f}_3(\text{time}) + \beta_4 \text{genderM} + \epsilon$$

```
library(splines)
fit2 <- lm(log(salary) ~ educ + ns(pexp, 4) + ns(time, 4) + gender,
           data = sal)
#summary(fit2)
```

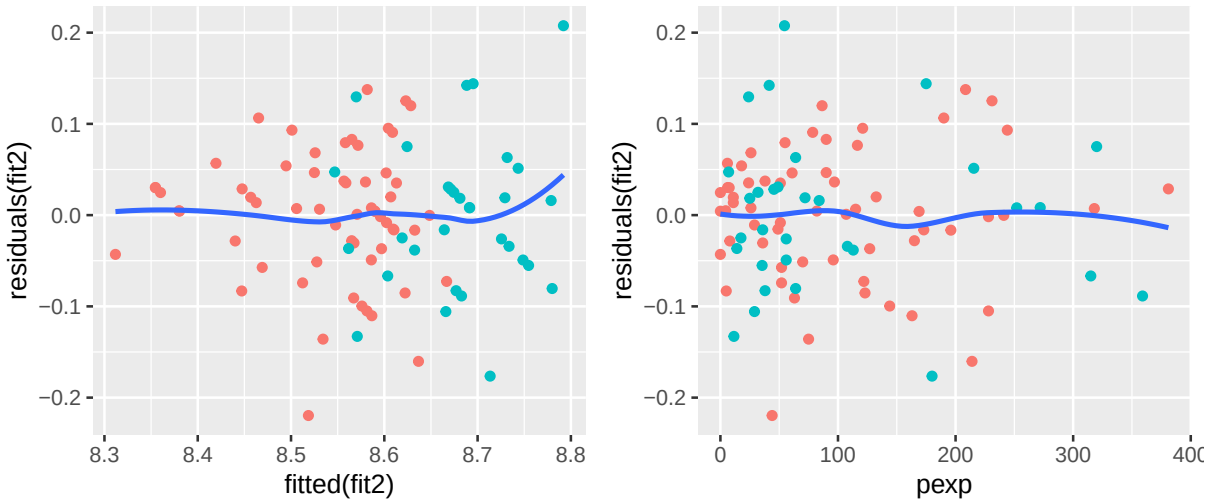
We can compare the two fits via anova (because they are nested)

```
anova(fit1,fit2)

Analysis of Variance Table

Model 1: log(salary) ~ educ + pexp + time + gender
Model 2: log(salary) ~ educ + ns(pexp, 4) + ns(time, 4) + gender
  Res.Df   RSS Df Sum of Sq    F  Pr(>F)
1     86 0.737
2     80 0.519  6     0.217 5.58 7.1e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This tells you that the extra parameters added for the splines on each variable are not all zero.

```
p1 <- ggplot(data=sal, aes(x=fitted(fit2), y= residuals(fit2))) +
  geom_point(aes(color=gender)) + geom_smooth(se=F) + guides(color=F)


p2 <- ggplot(data=sal, aes(x=pexp, y= residuals(fit2))) +
  geom_point(aes(color=gender)) + geom_smooth(se=F) + guides(color=F)

grid.arrange(p1,p2, ncol=2)
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



The residual plots for fit2 (at least the ones shown) do not show much curvature and are better behaved than the fit1 plots.

A simpler additive model

The model is

$$\log(\text{sal}) = \beta_0 + \beta_1 \text{educ} + \text{f}_2(\text{pexp}) + \beta_3 \text{time} + \beta_4 \text{genderM} + \epsilon$$

In this fit, we use a spline function for pexp and not for time. The anova tells us that the spline for time is not necessary.

```
fit3 <- lm(log(salary) ~ educ + ns(pexp, 4) + time + gender,
           data = sal)
anova(fit3, fit2)

Analysis of Variance Table

Model 1: log(salary) ~ educ + ns(pexp, 4) + time + gender
Model 2: log(salary) ~ educ + ns(pexp, 4) + ns(time, 4) + gender
  Res.Df   RSS Df Sum of Sq   F Pr(>F)
1     83 0.541
2     80 0.519  3    0.0214 1.1   0.35
```

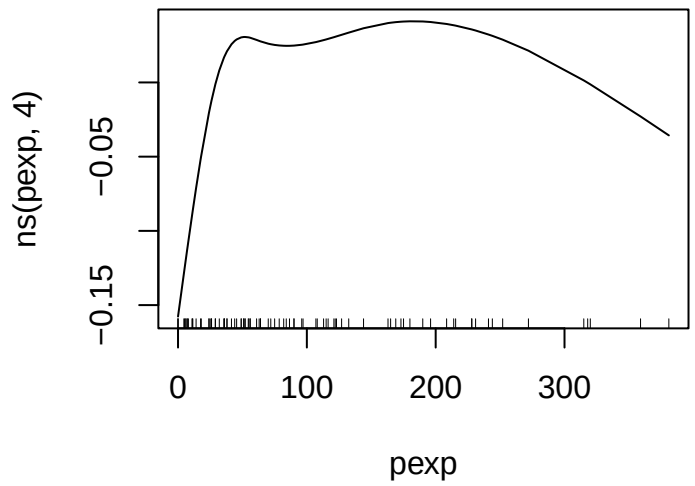
The residual plots (not shown) for this are similar to those for fit2.

The 95% confidence interval for gender is

```
exp(confint(fit3, "genderM")) # sal

            2.5 % 97.5 %
genderM  1.09    1.18
```

This plot shows the shape of the spline function fit to pexp.

```
suppressMessages(library(gam))
plot.Gam(fit3, terms = "ns(pexp, 4)")
```



This plot shows the effect of an additional year of experience on log salary, for fixed levels of other predictors.

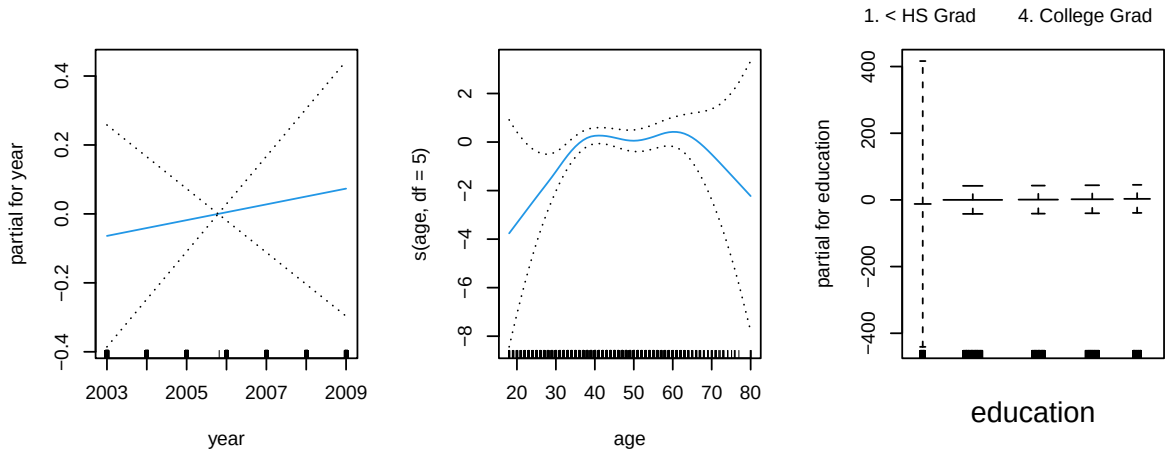
Up to about 100 months (8 years), pexp has a positive impact on log salary (and on salary). After that, more experience reduces salary.

NOTE: library gam allows for more complex fits than the lm() function, e.g. smoothing splines or other components that can't be expressed in terms of basis functions.

A classification problem

```
fit_gam <- gam(wage250 ~ year + s(age, df = 5) + education,
               family = "binomial",
               data = Wage)
```

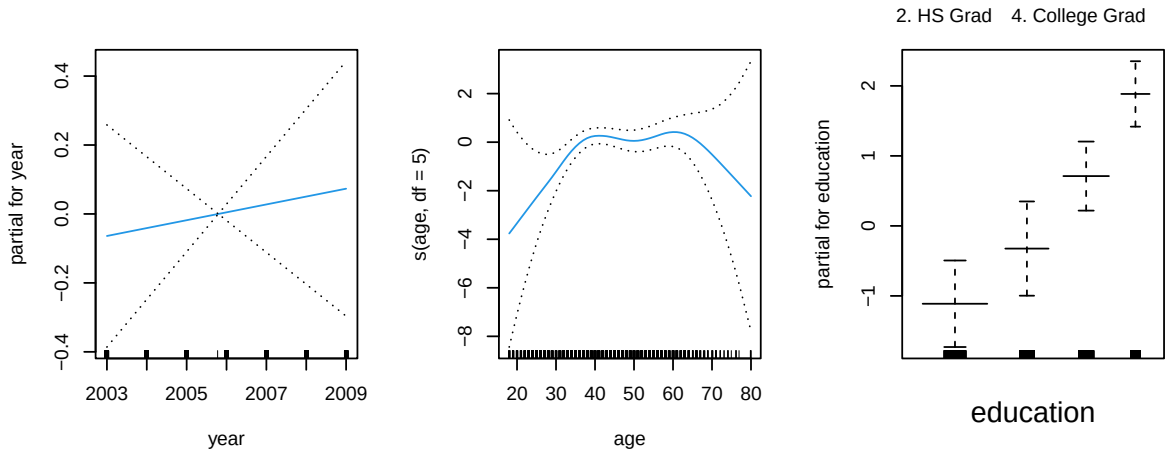
```
par(mfrow = c(1,3))
plot(fit_gam, se = TRUE, col = 4)
```




```
table(Wage$education, Wage$wage250)
```

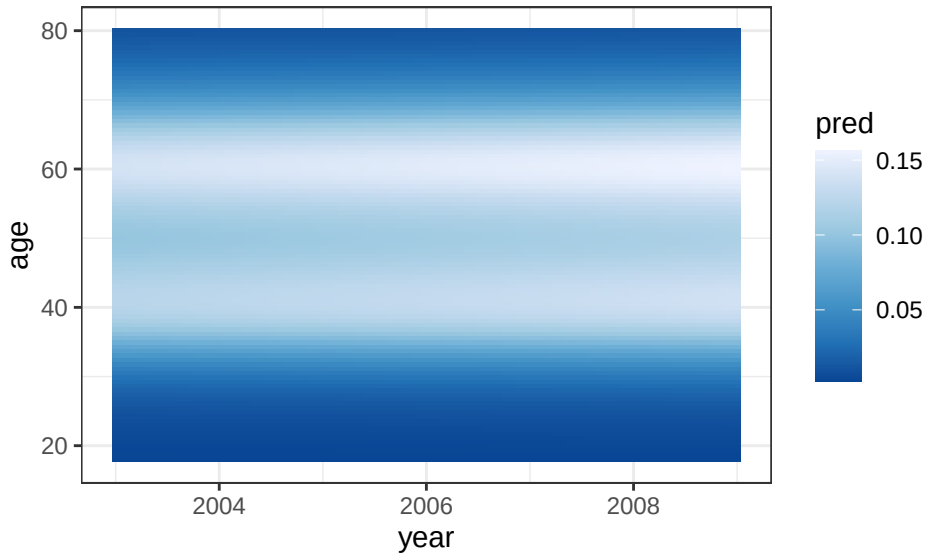
```
                      0   1
  1. < HS Grad      268   0
  2. HS Grad        966   5
  3. Some College   643   7
  4. College Grad   663  22
  5. Advanced Degree 381  45
```

```
fit_gam <- gam(wage250 ~ year + s(age, df = 5) + education,
               family = "binomial",
               data = Wage %>%
                 filter(education !="1. < HS Grad"))

par(mfrow = c(1,3))
plot(fit_gam, se = TRUE, col = 4)
```



Note: the plot method for gam objects plots the component functions that make it up on the scale of the linear predictor.

```
grid <- expand.grid(
  year = seq(2003, 2009, length = 100),
  age = seq(18, 80, length = 100)
)

grid$education <- levels(Wage$education)[5]

grid$pred <- matrix(predict(fit_gam,
                            newdata = grid,
                            type = "response"),
                    dim(grid)[1], 1)

grid %>%
  ggplot(aes(x = year, y = age, z = pred)) +
  theme_bw() +
  geom_raster(aes(fill = pred)) +
  scale_fill_distiller(type = "seq", palette = 1)
```




```
grid$pred2 <- grid$pred > 0.12


grid %>%
  ggplot(aes(x = year, y = age, fill = pred2)) +
  theme_bw() +
  geom_raster()
```