# Splines, model complexity, and prediction error

September 16, 2021

## Fitting, predicting, and plotting with splines

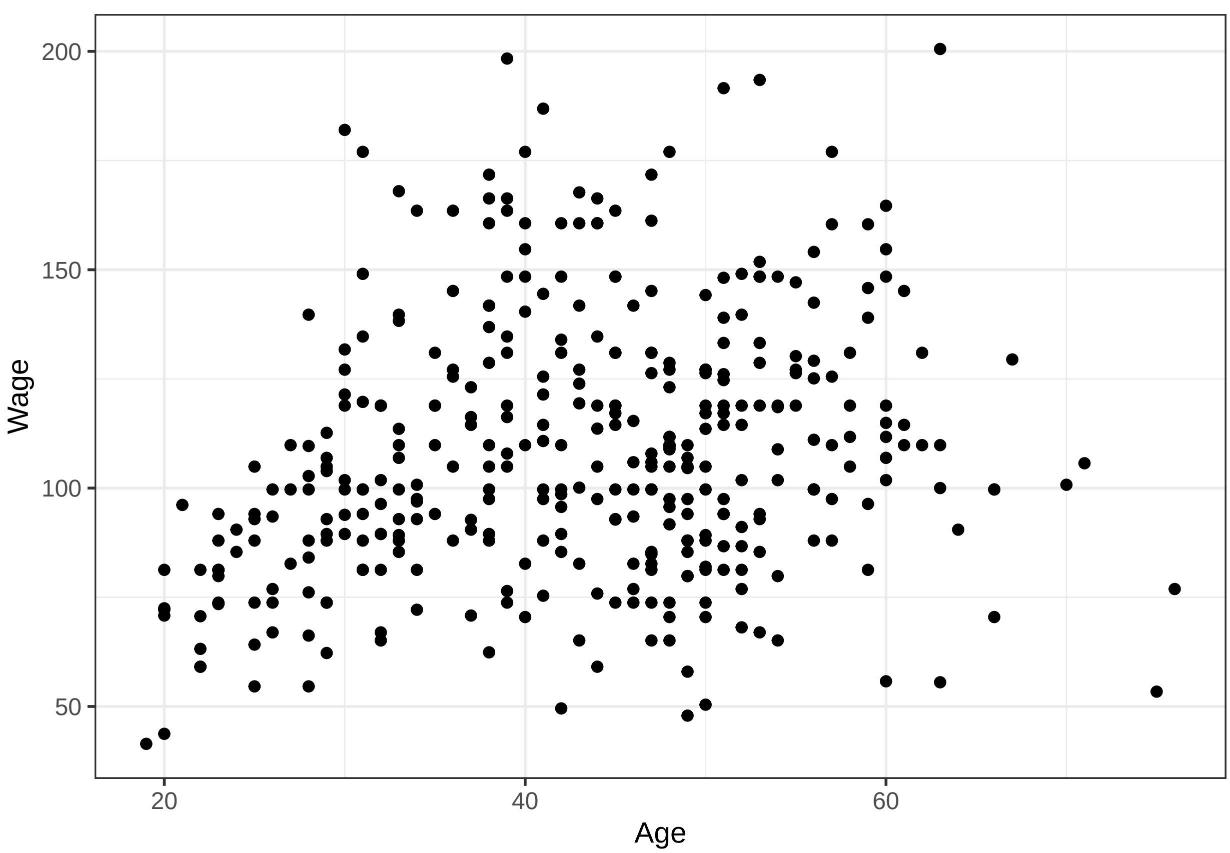
Let's first see how some of the plots from the lecture slides were generated. These plots are based on the `Wage` data from the `ISLR2` package, which accompanies the ISLRv2 course textbook.

```
Wage = ISLR2::Wage %>%
  as_tibble() %>%
  filter(wage < 225) # remove some outliers
```

Extract data from 2007:

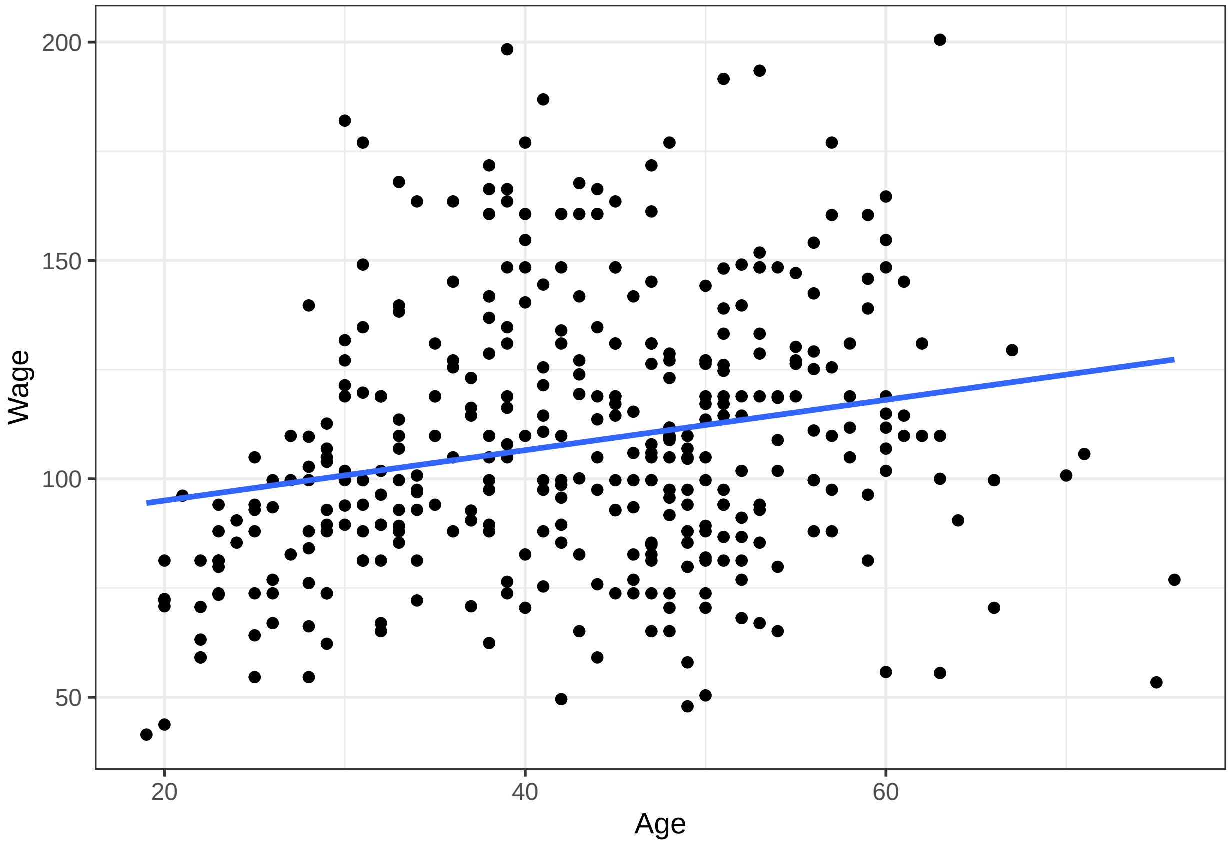
```
Wage_2007 = Wage %>% filter(year == 2007)

Wage_2007 %>%
  ggplot(aes(x = age, y = wage)) +
  geom_point() +
  theme_bw() +
  labs(x = "Age",
       y = "Wage")
```



We can add the least squares line to these data using `geom_smooth()`:

```
Wage_2007 %>%
  ggplot(aes(x = age, y = wage)) +
  geom_point() +
  theme_bw() +
  geom_smooth(method = "lm",
              formula = "y ~ x", # note use y and x instead of wage and age
              se = FALSE) +
  labs(x = "Age",
       y = "Wage")
```




Fitting a spline is almost as simple as fitting a linear regression, since since a spline fit is after all a regression fit! We can use the following syntax:

```
spline_fit = lm(wage ~ splines::ns(age, df = 5), data = Wage_2007)
```

Typically we inspect the results of a regression using `summary()`:

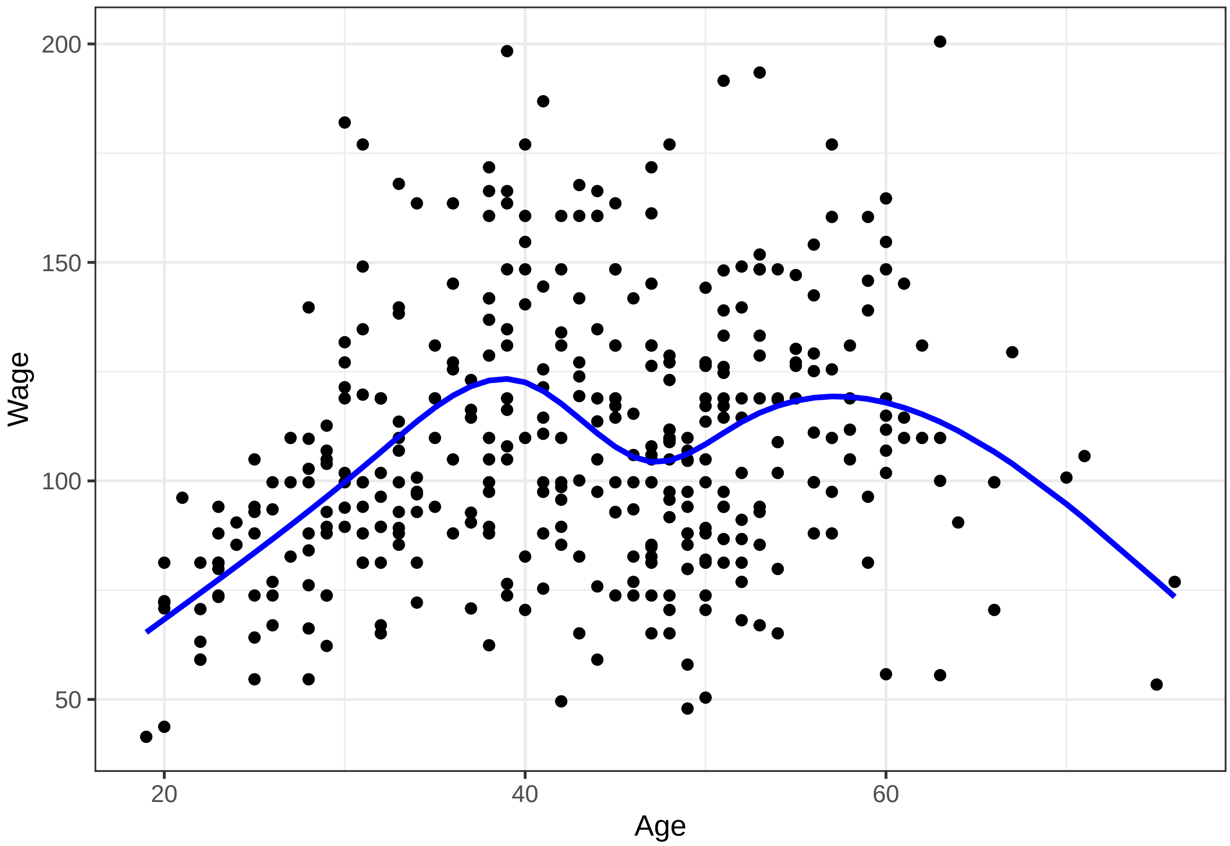
```
summary(spline_fit)
```

```
##
## Call:
## lm(formula = wage ~ splines::ns(age, df = 5), data = Wage_2007)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -68.134 -19.344  -0.782  15.851  87.041
##
```

```
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 65.348      8.230   7.940 2.44e-14 ***
## splines::ns(age, df = 5)1   69.939      8.783   7.963 2.08e-14 ***
## splines::ns(age, df = 5)2   29.958     10.006   2.994  0.00294 **
## splines::ns(age, df = 5)3   59.448      8.904   6.677 8.96e-11 ***
## splines::ns(age, df = 5)4   67.113     20.702   3.242  0.00130 **
## splines::ns(age, df = 5)5   -8.112     15.051  -0.539  0.59022
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 27.74 on 370 degrees of freedom
## Multiple R-squared:  0.1746, Adjusted R-squared:  0.1634
## F-statistic: 15.65 on 5 and 370 DF,  p-value: 5.583e-14
```

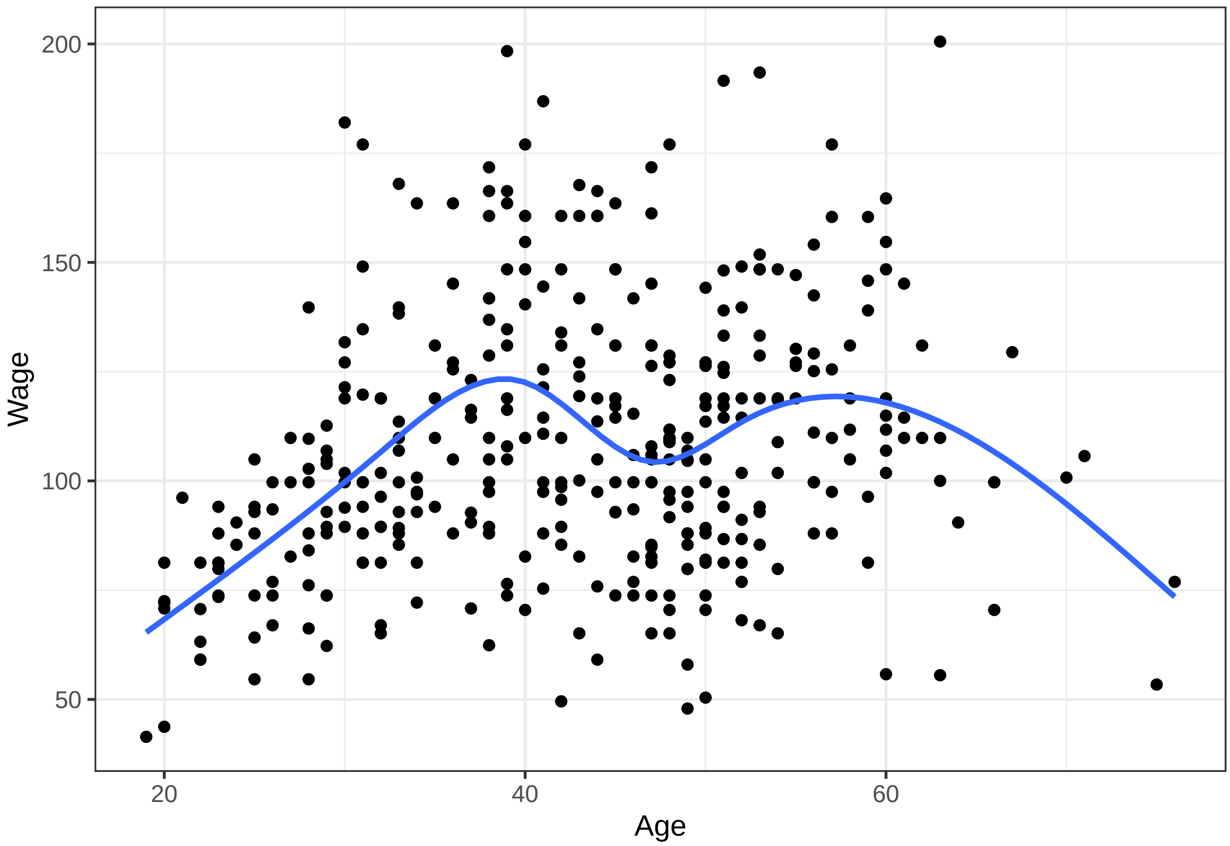
Actually in this case, we're less interested in the coefficients (or p-values) of the linear regression and more interested in the fitted function. We can extract the fitted values and add them to the plot as follows:

```
fitted_values = spline_fit$fitted.values
Wage_2007 %>%
  mutate(fitted_values = fitted_values) %>%
  ggplot(aes(x = age)) +
  geom_point(aes(y = wage)) +
  geom_line(aes(y = fitted_values),
            colour = "blue",
            size = 1) +
  theme_bw() +
  labs(x = "Age",
       y = "Wage")
```




There's actually a simpler way of making the same plot via `geom_smooth()`, just substituting a different `formula` argument!

```
Wage_2007 %>%
  mutate(fitted_values = fitted_values) %>%
  ggplot(aes(x = age, y = wage)) +
  geom_point() +
  geom_smooth(method = "lm",
              formula = "y ~ splines::ns(x, df = 5)", # note use y and x instead of wage and age
              se = FALSE) +   theme_bw() +
  labs(x = "Age",
       y = "Wage")
```




Now, in order to make predictions for a test dataset (e.g. the data from the year 2008), we can use the convenient `predict()` function.

```
Wage_2008 = Wage %>% filter(year == 2008)
predictions = predict(spline_fit, newdata = Wage_2008)
head(predictions)
```

```
##         1         2         3         4         5         6
## 117.17964  99.72272 119.32551 110.16106 113.49437  91.36378
```

```
length(predictions)
```

```
## [1] 377
```

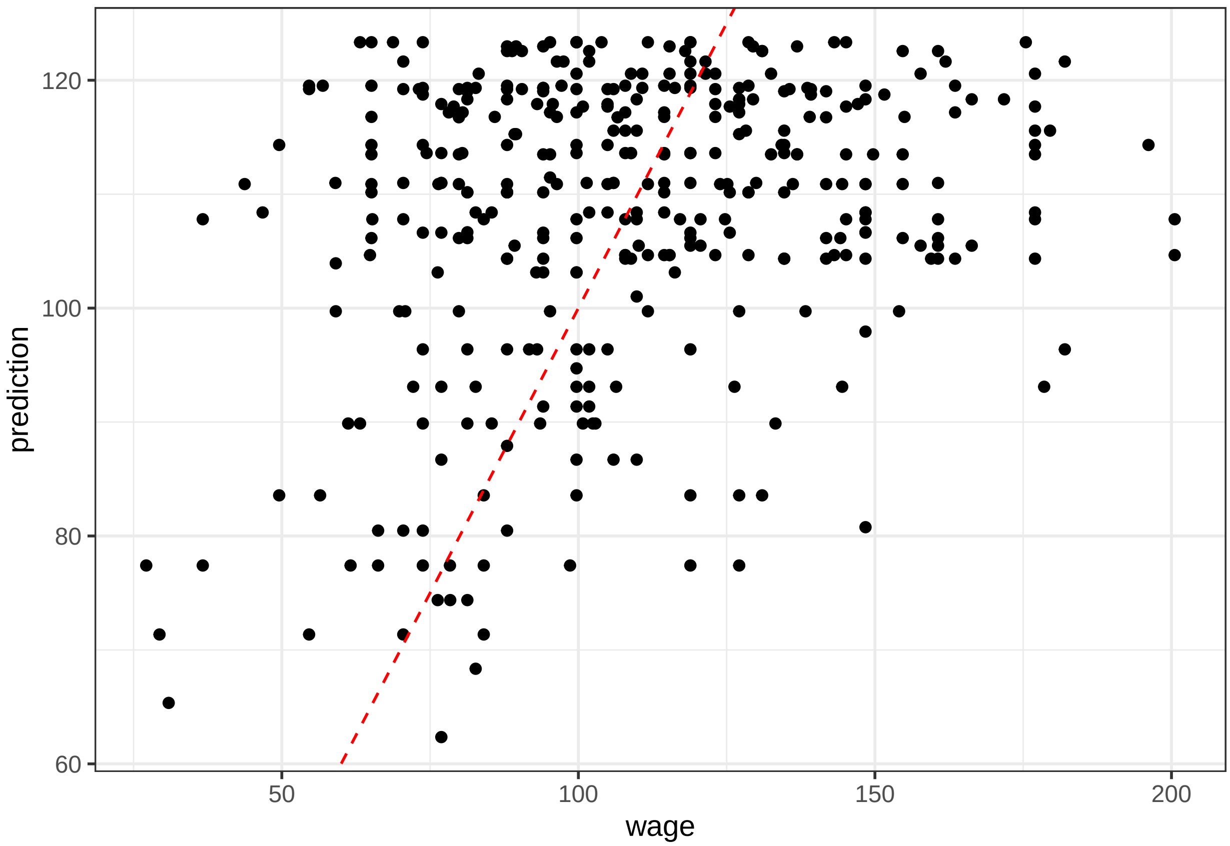
We can then add these predictions back to the data frame and compute the (root) mean squared test error:

```
Wage_2008 = Wage_2008 %>% mutate(prediction = predictions)
Wage_2008
```

```
## # A tibble: 377 x 12
##     year   age maritl  race  education region jobclass health health_ins logwage
##    <int> <int> <fct>   <fct> <fct>     <fct>  <fct>    <fct>  <fct>        <dbl>
##  1  2008    54 2. Mar~ 1. W~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes        4.85
##  2  2008    30 1. Nev~ 3. A~ 3. Some ~ 2. Mi~ 2. Info~ 1. <=~ 1. Yes        4.72
##  3  2008    57 2. Mar~ 1. W~ 2. HS Gr~ 2. Mi~ 1. Indu~ 2. >=~ 2. No         4.76
##  4  2008    33 1. Nev~ 1. W~ 5. Advan~ 2. Mi~ 1. Indu~ 2. >=~ 2. No         4.40
##  5  2008    52 2. Mar~ 1. W~ 5. Advan~ 2. Mi~ 1. Indu~ 2. >=~ 1. Yes        5.04
##  6  2008    71 2. Mar~ 1. W~ 3. Some ~ 2. Mi~ 1. Indu~ 2. >=~ 1. Yes        4.62
##  7  2008    21 1. Nev~ 2. B~ 2. HS Gr~ 2. Mi~ 1. Indu~ 2. >=~ 2. No         4.26
##  8  2008    44 1. Nev~ 1. W~ 1. < HS ~ 2. Mi~ 1. Indu~ 1. <=~ 2. No         4.48
```

```
## 9  2008    37 2. Mar~ 1. W~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes        4.80
## 10  2008    50 1. Nev~ 1. W~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes        4.74
## # ... with 367 more rows, and 2 more variables: wage <dbl>, prediction <dbl>
```

```
Wage_2008 %>%
  ggplot(aes(x = wage, y = prediction)) +
  geom_point() +
  geom_abline(slope = 1, colour = "red", linetype = "dashed") +
  theme_bw()
```




```
Wage_2008 %>%
  summarise(test_error = sqrt(mean((wage-prediction)^2)))
```

```
## # A tibble: 1 x 1
##   test_error
##        <dbl>
## 1       31.6
```

By contrast, let's take a look at the RMS training error:

```
Wage_2007 = Wage_2007 %>%
  mutate(fitted_value = spline_fit$fitted.values)
Wage_2007 %>%
  summarise(training_error = sqrt(mean((wage-fitted_value)^2)))
```

```
## # A tibble: 1 x 1
##   training_error
##            <dbl>
## 1           27.5
```

Now, let's calculate the training and test errors for `df` = 1,2,3,4,5:

```
df_values = 1:5
train_errors = numeric(length(df_values))
test_errors = numeric(length(df_values))
for(i in 1:length(df_values)){
  df = df_values[i]
  lm_fit = lm(wage ~ splines::ns(age, df = df),
              data = Wage_2007)
  fitted_wages_2007 = lm_fit$fitted.values
  predicted_wages_2008 = predict(lm_fit, newdata = Wage_2008)
  actual_wages_2007 = Wage_2007 %>% pull(wage)
  actual_wages_2008 = Wage_2008 %>% pull(wage)
  train_errors[i] = sqrt(mean((fitted_wages_2007 - actual_wages_2007)^2))
  test_errors[i] = sqrt(mean((predicted_wages_2008 - actual_wages_2008)^2))
}
```

Let's take a look at these errors:

```
train_errors
```

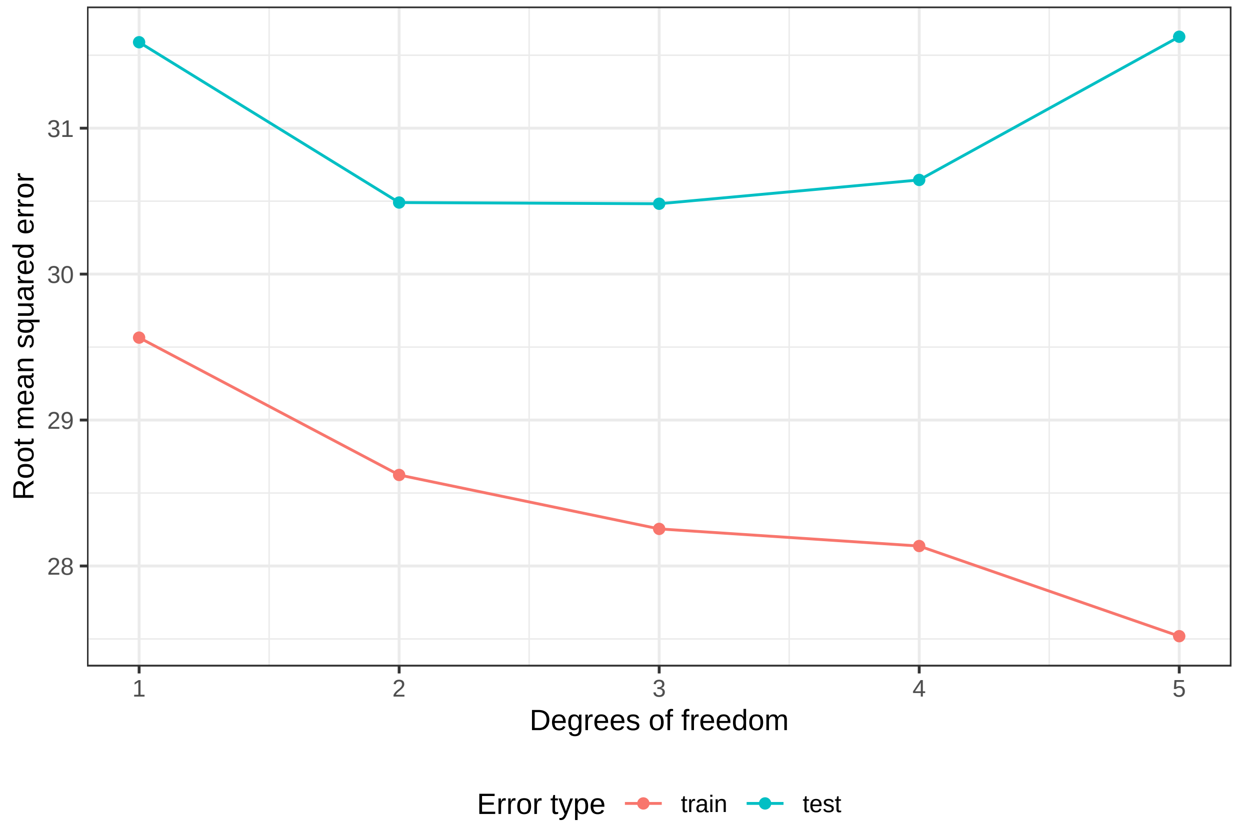
```
## [1] 29.56470 28.62369 28.25417 28.13682 27.51901
```

```
test_errors
```

```
## [1] 31.58897 30.49050 30.48212 30.64534 31.62660
```

Now let's plot them:

```
tibble(df = df_values,
       error_test = test_errors,
       error_train = train_errors) %>%
  pivot_longer(-df,
               names_to = "Error type",
               names_prefix = "error_",
               values_to = "error") %>%
  mutate(`Error type` = factor(`Error type`, levels = c("train", "test"))) %>%
  ggplot(aes(x = df, y = error, colour = `Error type`)) +
  geom_point() + geom_line() +
  labs(x = "Degrees of freedom",
       y = "Root mean squared error") +
  theme_bw() +
  theme(legend.position = "bottom")
```




## Exercise: Exploring model complexity in a simulation

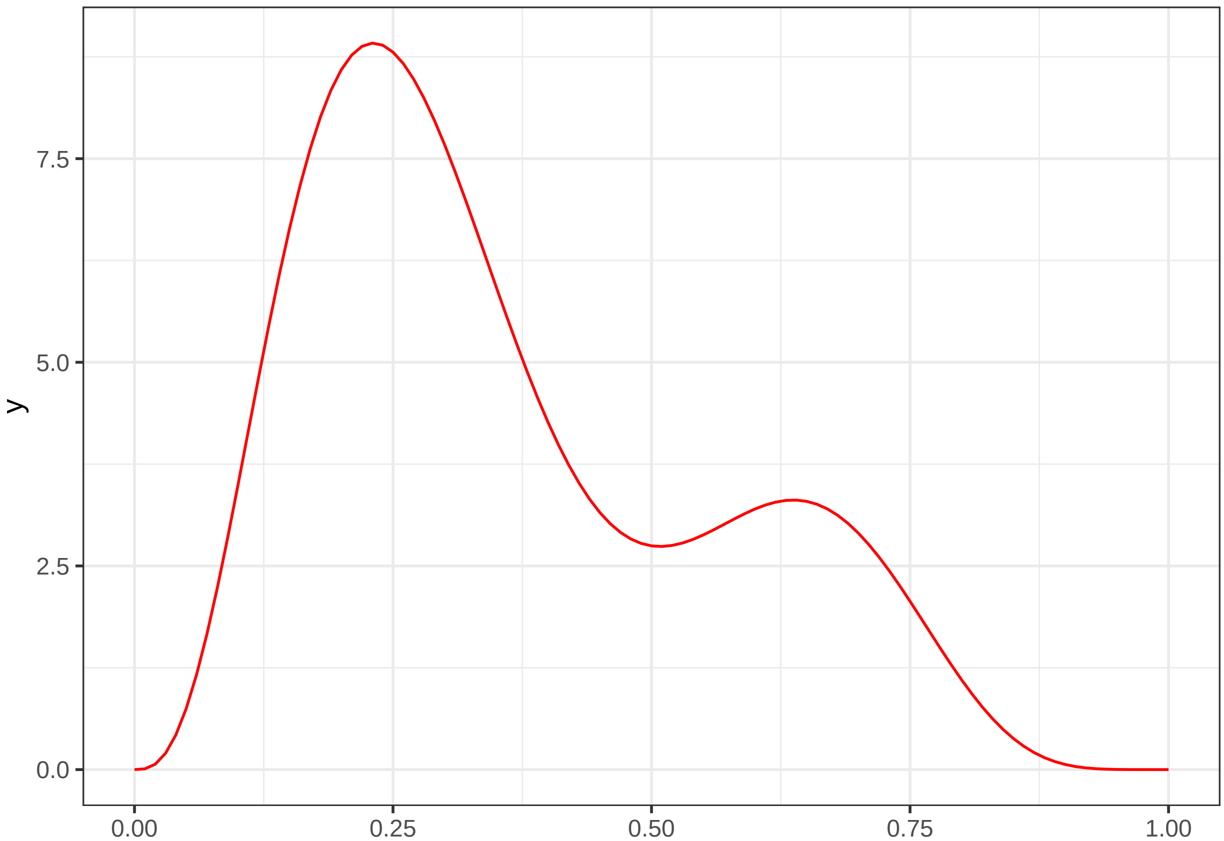
**Thank you to Leontij Potupin for contributing the solutions below.**

In the `Wage` data, we don't know what the "true" trend is. It may be illuminating, therefore, to replace this data with *simulated* data (i.e. data we generate ourselves). Suppose that $Y = f(X) + \epsilon$, and let us assume $f$ take the following form:

```
# credit: this function comes from
# https://gist.github.com/rudeboybert/752f7aa1e42faa2174822dd29bfaf959
f <- function(x){
  0.2*x^11*(10*(1-x))^6+10*(10*x)^3*(1-x)^10
}
```

It's hard to understand what this function is but we can plot it:

```
ggplot() +
  stat_function(fun = f, colour = "red") +
  theme_bw()
```



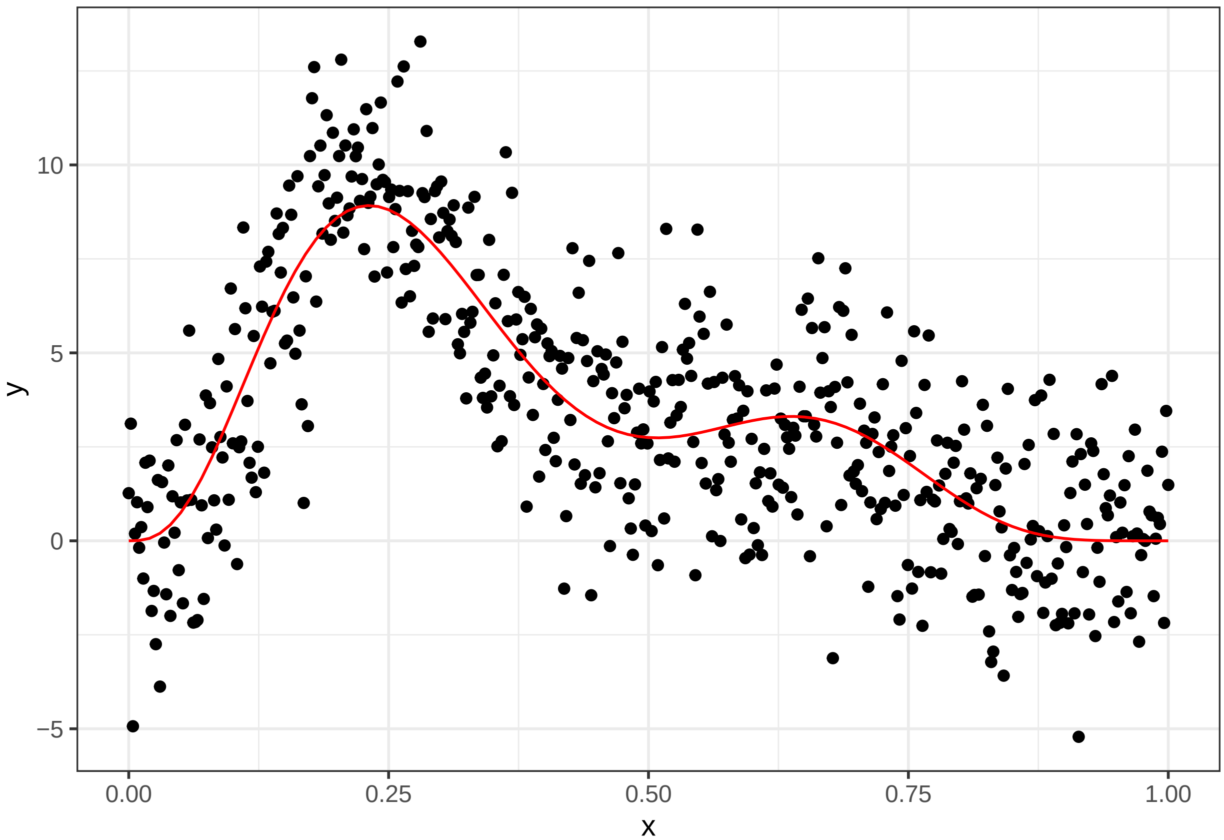

1. Create 500 training data points $(X, Y)$ as follows. First, generate 500 equally spaced values of `x` between 0 and 1 (using the `seq` function). Then, sample a vector `y` based on $Y = f(X) + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$ and $\sigma = 2$. Put `x` and `y` into a tibble called `data`.

```
x = seq.int(0,1,length.out = 500)
y = f(x) + rnorm(n = 500, mean = 0, sd = 2)
data = tibble(x,y)
data
```

```
## # A tibble: 500 x 2
##          x      y
##      <dbl>  <dbl>
##   1 0        1.27
##   2 0.00200  3.12
##   3 0.00401 -4.93
##   4 0.00601  0.190
##   5 0.00802  1.03
##   6 0.0100  -0.184
##   7 0.0120   0.364
##   8 0.0140  -1.00
##   9 0.0160   2.08
##  10 0.0180   0.897
## # ... with 490 more rows
```

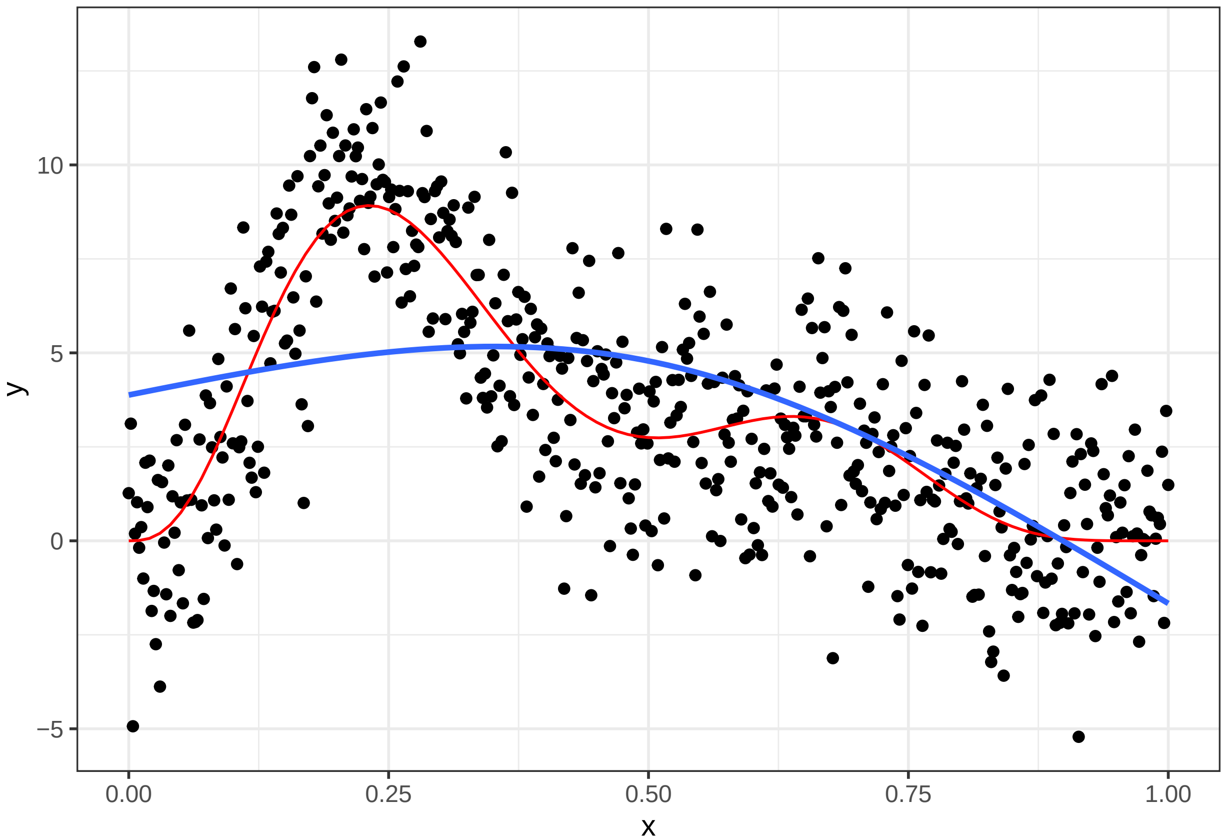
2. Create a scatter plot of these data, overlaying the function `f` in red.

```
data %>%
  ggplot(aes(x=x,y=y)) + geom_point() + theme_bw() + stat_function(fun = f, colour = "red")
```

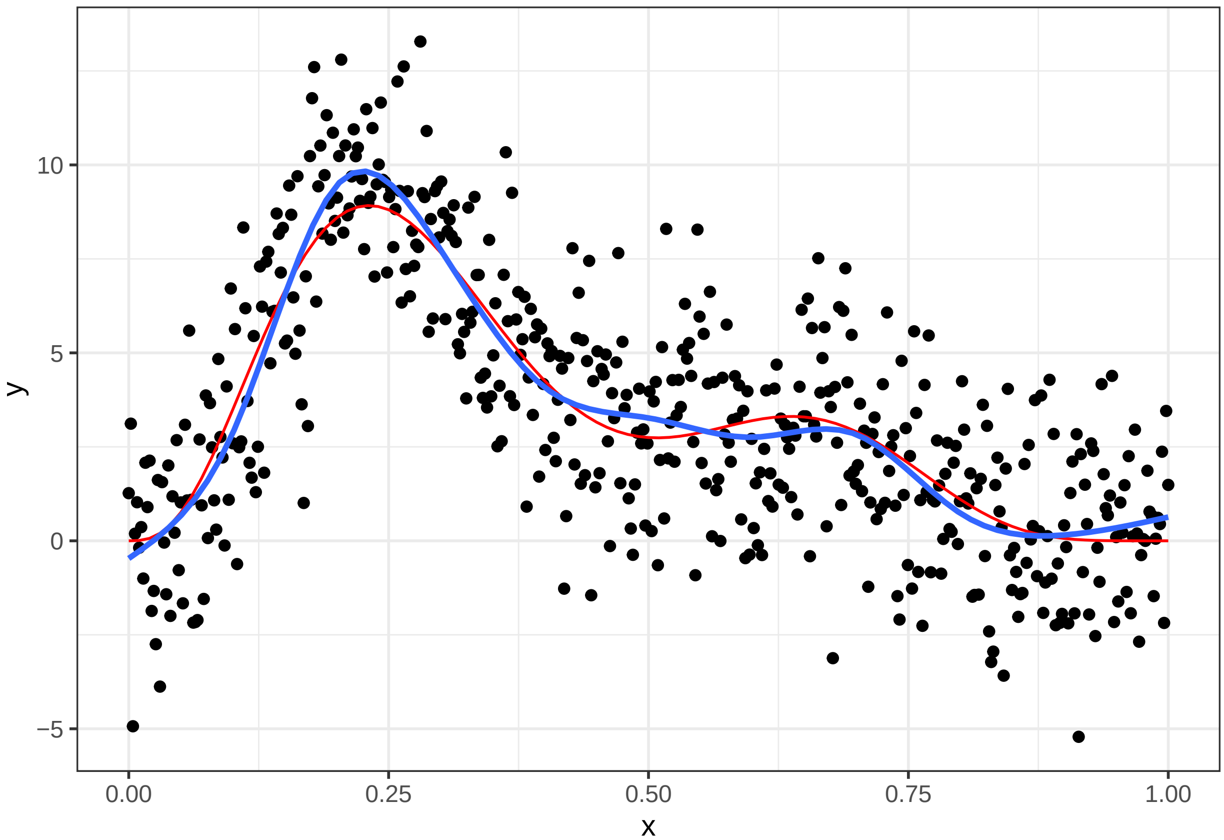



3. Take a look at what natural spline fits look like for this data, using different degrees of freedom. Using `geom_smooth()`, plot natural spline fits with degrees of freedom equal to 2,10,25,50 (use separate plots for each value of df). For each of these plots, superimpose the true trend as a red line. Comment on what happens as you increase the degrees of freedom. Which of these four values seems to fit the scatter plot best?

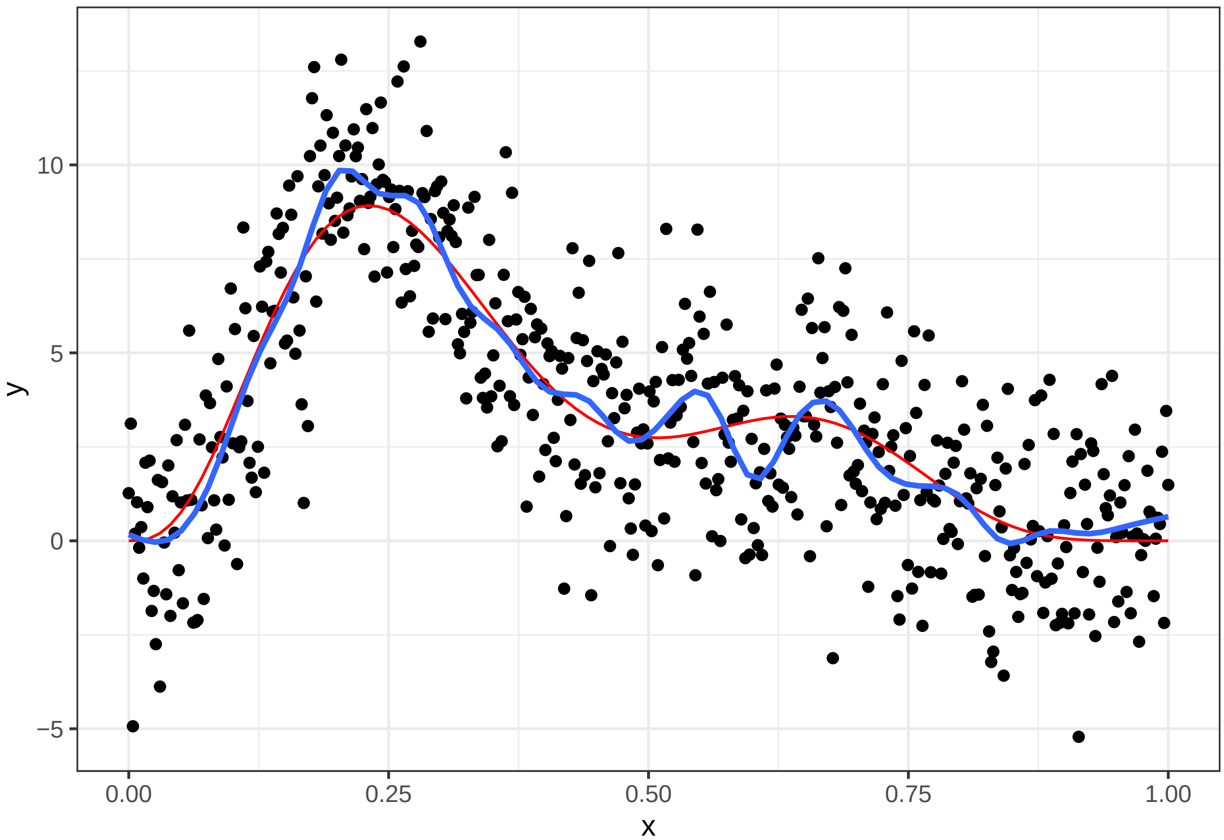
```
data %>%
  ggplot(aes(x=x,y=y)) +
  geom_point() +
  theme_bw() +
  stat_function(fun = f, colour = "red") +
  geom_smooth(method = "lm",
              formula = "y ~ splines::ns(x, df = 2)",
              se = FALSE)
```



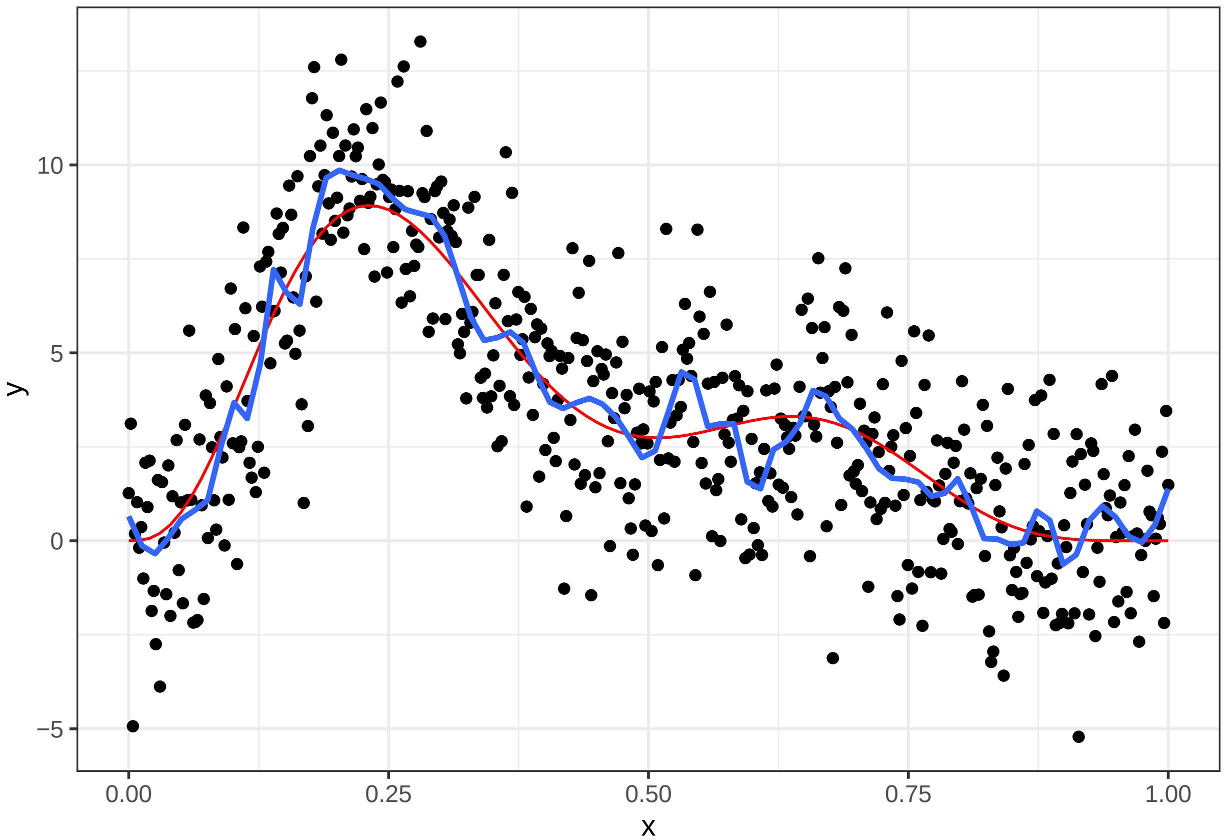

```
data %>%
  ggplot(aes(x=x,y=y)) +
  geom_point() +
  theme_bw() +
  stat_function(fun = f, colour = "red") +
  geom_smooth(method = "lm",
              formula = "y ~ splines::ns(x, df = 10)",
              se = FALSE)
```




```r
data %>%
  ggplot(aes(x=x,y=y)) +
  geom_point() +
  theme_bw() +
  stat_function(fun = f, colour = "red") +
  geom_smooth(method = "lm",
              formula = "y ~ splines::ns(x, df = 25)",
              se = FALSE)
```




```
data %>%
  ggplot(aes(x=x,y=y)) +
  geom_point() +
  theme_bw() +
  stat_function(fun = f, colour = "red") +
  geom_smooth(method = "lm",
              formula = "y ~ splines::ns(x, df = 50)",
              se = FALSE)
```




**Out of these four values, 10 degrees of freedom seem to fit the scatter plot best.**

4. (If time permits) Create a test data set with 500 additional points from the same distribution and compute the root mean square test error for each of the four spline fits above. Does the degrees of freedom that gives the closest fit to the underlying trend also lead to the lowest test error?

```
x = seq(0,1,length.out = 500)
y = f(x) + rnorm(n = 500, mean = 0, sd = 2)
additional_data = tibble(x,y)
additional_data
```

```
## # A tibble: 500 x 2
##          x       y
##      <dbl>   <dbl>
##  1 0         1.77
##  2 0.00200   0.105
##  3 0.00401  -2.84
##  4 0.00601  -1.53
##  5 0.00802   1.29
##  6 0.0100    0.850
##  7 0.0120   -2.96
##  8 0.0140   -0.0531
##  9 0.0160    1.89
## 10 0.0180   -0.237
## # ... with 490 more rows
```

```
spline_fit_2 = lm(y ~ splines::ns(x, df = 2), data = data)
spline_fit_10 = lm(y ~ splines::ns(x, df = 10), data = data)
spline_fit_25 = lm(y ~ splines::ns(x, df = 25), data = data)
```

```r
spline_fit_50 = lm(y ~ splines::ns(x, df = 50), data = data)

predictions_2 = predict(spline_fit_2, newdata = additional_data)
predictions_10 = predict(spline_fit_10, newdata = additional_data)
predictions_25 = predict(spline_fit_25, newdata = additional_data)
predictions_50 = predict(spline_fit_50, newdata = additional_data)

additional_data = additional_data %>%
  mutate(predictions_2 = predictions_2,
         predictions_10 = predictions_10,
         predictions_25 = predictions_25,
         predictions_50 = predictions_50)

additional_data %>%
  summarise(test_error_2 = sqrt(mean((y-predictions_2)^2)),
            test_error_10 = sqrt(mean((y-predictions_10)^2)),
            test_error_25 = sqrt(mean((y-predictions_25)^2)),
            test_error_50 = sqrt(mean((y-predictions_50)^2)))
```

```
## # A tibble: 1 x 4
##   test_error_2 test_error_10 test_error_25 test_error_50
##          <dbl>         <dbl>         <dbl>         <dbl>
## 1         2.75          1.98          2.01          2.04
```

**Yes, the degrees of freedom that gives the closest fit to the underlying trend also leads to the lowest test error.**