MLStats: Moving Beyond Linearity

Darshan Patel

2/19/2019

In this assignment, mimic the lab exercises from ISLR Chapter 7: Moving Beyond Linearity.

Libraries

Load the following libraries.

```
rm(list = ls())
library(MASS)
library(tidyverse)
```

```
## Warning: package 'tibble' was built under R version 3.4.4
## Warning: package 'tidyr' was built under R version 3.4.4
## Warning: package 'purrr' was built under R version 3.4.4
## Warning: package 'dplyr' was built under R version 3.4.4
```

```
library(gridExtra)
library(boot)
library(GGally)
```

```
## Warning: package 'GGally' was built under R version 3.4.4
## Warning: replacing previous import 'ggplot2::empty' by 'plyr::empty' when
## loading 'GGally'
```

```
library(gam)
```

```
## Warning: package 'gam' was built under R version 3.4.4
```

```
library(splines)
library(leaps)
```

Dataset

In this assignment, the dataset that will be used is.. none. Instead, a synthetic dataset will be made using sampling and decimal degrees.

The reason for this is that I have looked through many datasets where only linear relationships exists and if nonlinear methods were used, results did not seem to differ from method to method.

```
set.seed(2019)
n = 50
x1 = sample(1:200, size = n, replace = FALSE)
x2 = sample(1:200, size = n, replace = FALSE)
x3 = sample(c("BS", "MS", "PhD", size = n, replace = TRUE))
noise = runif(n, 0, 15)
y = rnorm(n, 13 + 54*x1^(-0.2) + 2.6*x1^(-3.1) + 1.3*x2^(-2.1), 1) + noise
df = data.frame(y, x1, x2, x3)
df$x3 = as.factor(df$x3)
```

In this dataset, there are

```r
n
```

```
## [1] 50
```

observations. The variables are:

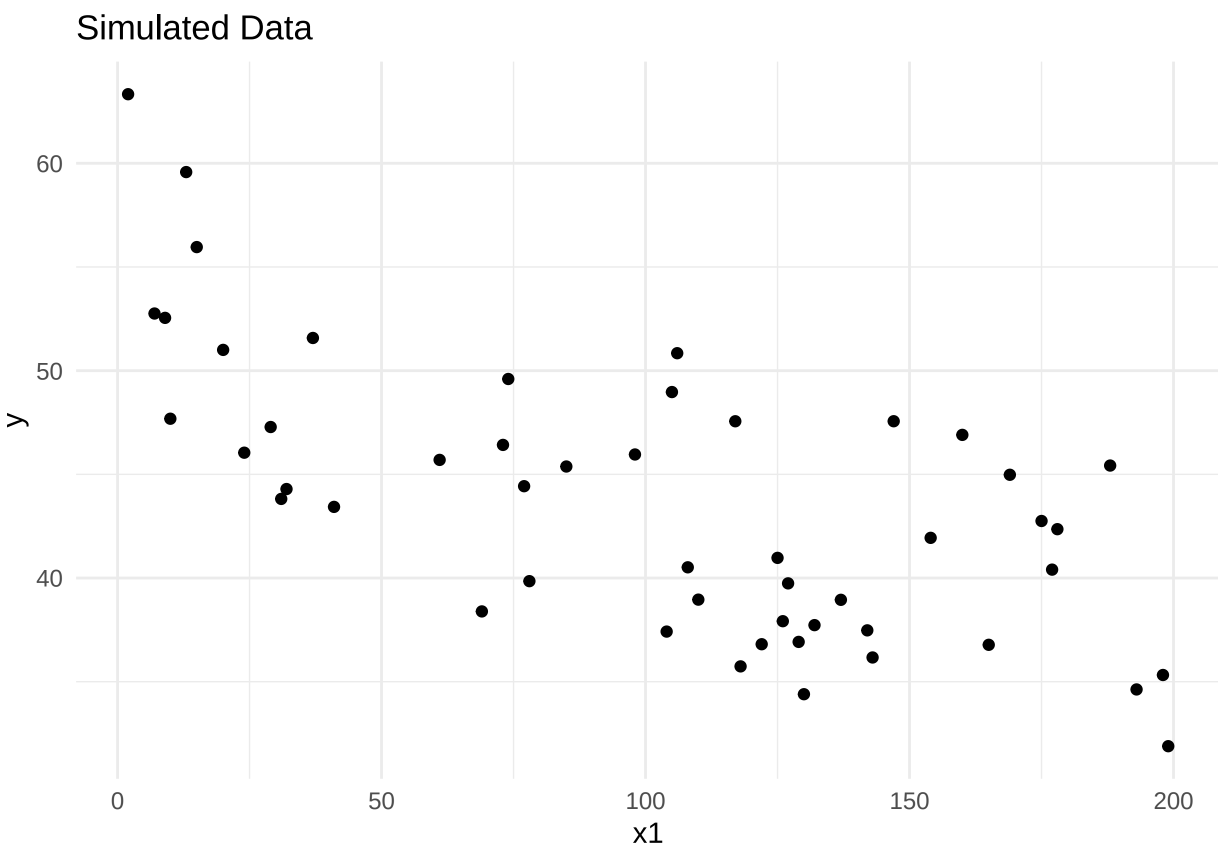
```r
colnames(df)
```

```
## [1] "y"  "x1" "x2" "x3"
```

Pretty basic. x1 will be used for polynomial regression, step functions and splines, whereas x2 and x3 will come with GAMs.

The data for x1 and y is plotted below.

```r
basic_plot = ggplot(df, aes(x1,y)) + geom_point() +
  ggtitle("Simulated Data") +
  theme_minimal()
basic_plot
```



The relationship between x and y is pretty nonlinear.

The first non-linear fitting procedure that will be looked at is polynomial regression.

Polynomial Regression and Step Functions

First fit a polynomial regression model of degree 3.

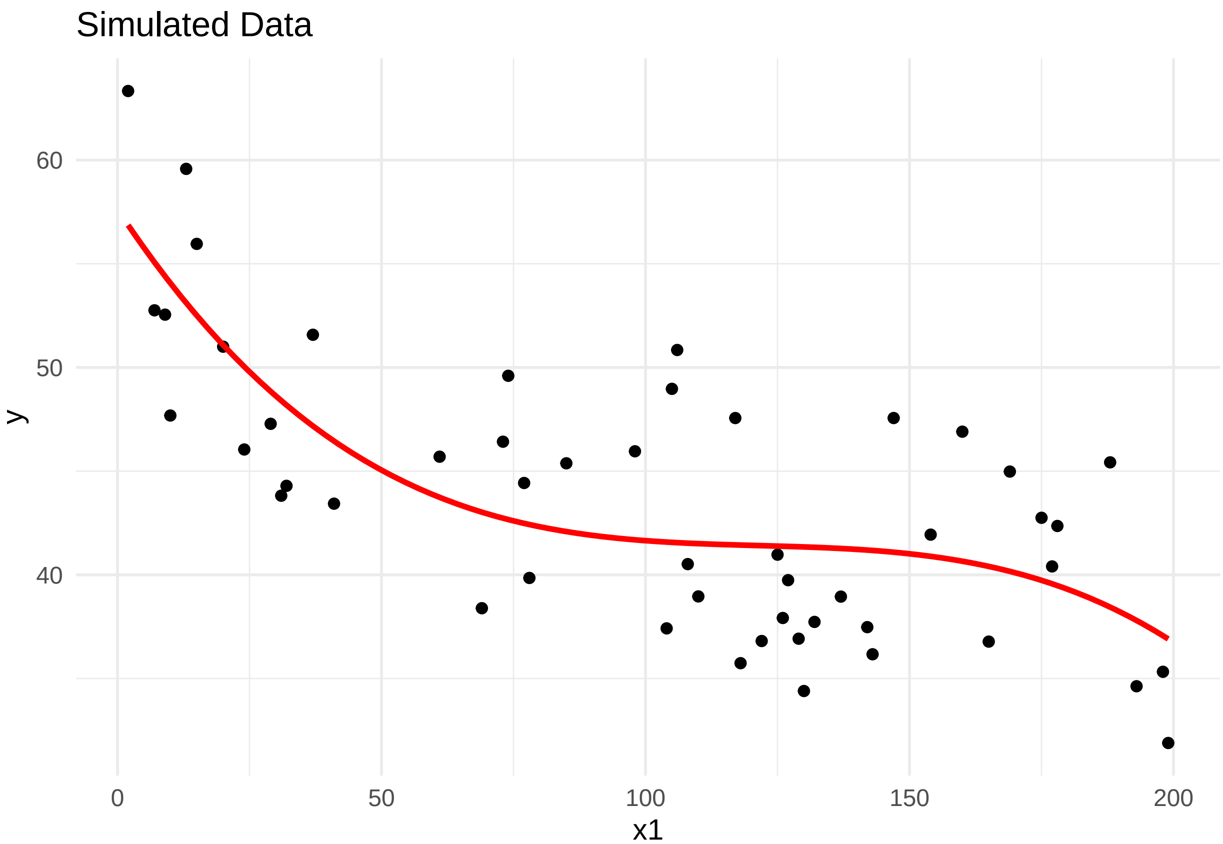
```r
model1 = lm(y ~ poly(x1, 3), data = df)
summary(model1)
```

```
##
## Call:
## lm(formula = y ~ poly(x1, 3), data = df)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -6.948 -3.694 -1.594  3.630  9.291
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    43.660      0.651  67.068  < 2e-16 ***
## poly(x1, 3)1  -31.949      4.603  -6.941 1.12e-08 ***
## poly(x1, 3)2   11.132      4.603   2.418   0.0196 *
## poly(x1, 3)3   -9.038      4.603  -1.963   0.0557 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.603 on 46 degrees of freedom
## Multiple R-squared:  0.5572, Adjusted R-squared:  0.5283
## F-statistic: 19.29 on 3 and 46 DF,  p-value: 3.048e-08
```

According to the output, the adjusted R^2 statistic for this model is 0.52, meaning that only 52% of the variation in y is explained by the model. In addition, two of the three coefficient estimates are statistically significant.

Plot this on the data.

```r
basic_plot + stat_smooth(method = "lm", se = FALSE,
                         formula = y ~ poly(x, 3),
                         color = "red")
```



The model can be improved on. Perform hypothesis testing using ANOVA to find the simplest model to determine `rings` using `whole.weight`.

```
fit1 = lm(y ~ poly(x1, 1), data = df)
fit2 = lm(y ~ poly(x1, 2), data = df)
fit3 = lm(y ~ poly(x1, 3), data = df)
fit4 = lm(y ~ poly(x1, 4), data = df)
fit5 = lm(y ~ poly(x1, 5), data = df)
fit6 = lm(y ~ poly(x1, 6), data = df)
anova(fit1, fit2, fit3, fit4, fit5, fit6)
```

```
## Analysis of Variance Table
##
## Model 1: y ~ poly(x1, 1)
## Model 2: y ~ poly(x1, 2)
## Model 3: y ~ poly(x1, 3)
## Model 4: y ~ poly(x1, 4)
## Model 5: y ~ poly(x1, 5)
## Model 6: y ~ poly(x1, 6)
##   Res.Df     RSS Df Sum of Sq       F   Pr(>F)
## 1     48 1180.33
## 2     47 1056.40  1   123.923  7.0064 0.011305 *
## 3     46  974.72  1    81.684  4.6183 0.037305 *
## 4     45  974.68  1     0.038  0.0022 0.963132
## 5     44  783.61  1   191.072 10.8029 0.002023 **
## 6     43  760.55  1    23.061  1.3038 0.259838
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

According to an analysis of variance using an F-test, the p-value comparing the linear model to the quadratic model is next to 0, meaning that the linear fit is not sufficient. Likewise, the p-value comparing the quadratic model to the tertiary model is also small, However, the p-value comparing the tertiary model to the degree 4 model is significantly large (greater than $\alpha = 0.05$), meaning that the degree 3 model will provide the best fit to the data.

The coefficient of the degree 3 model are

```r
coef(summary(fit3))[,1]
```

```
##  (Intercept) poly(x1, 3)1 poly(x1, 3)2 poly(x1, 3)3
##    43.660446   -31.948985    11.132065    -9.037937
```

Another way to find the best polynomial fit is by using cross validation.

```r
set.seed(2019)
indices = sample(1:nrow(df), size = 0.7*nrow(df))
train = df[indices,]
test = df[indices,]

sse = c()
for(i in 1:10){
  model_temp = lm(y ~ poly(x1, i), data = train)
  sse = c(sse, mean((predict(model_temp) - test$y)^2))
}
```

By cross validation, the best polynomial fit is made using degree

```r
which.min(sse)
```

```
## [1] 10
```

That's a lot of degrees!

Now let's try regressing y using a step function.

```r
model2 = lm(y ~ cut(x1, 3), data = df)
summary(model2)
```

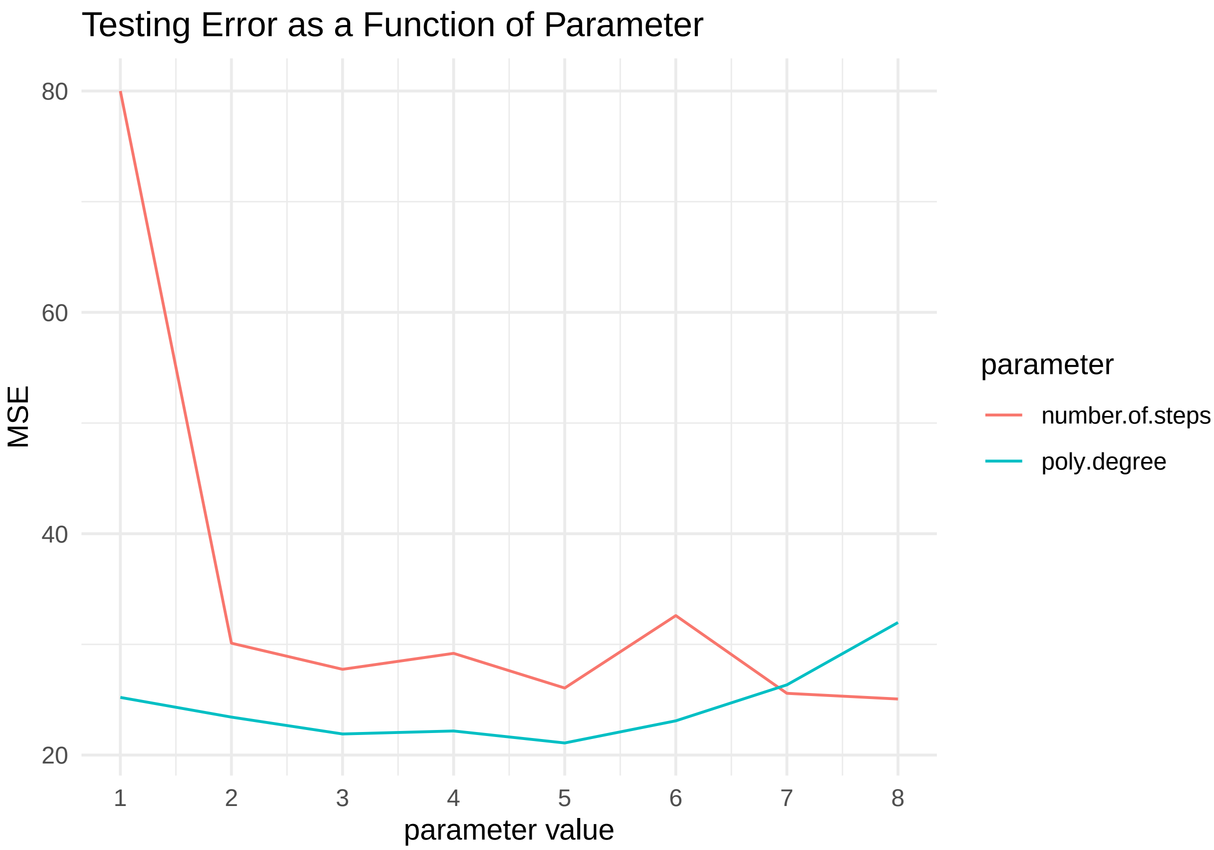
```
##
## Call:
## lm(formula = y ~ cut(x1, 3), data = df)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -8.345 -4.186 -1.206  4.169 12.975
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)            50.356      1.412  35.662  < 2e-16 ***
## cut(x1, 3)(67.7,133]   -8.713      1.823  -4.780 1.77e-05 ***
## cut(x1, 3)(133,199]   -10.121      1.963  -5.155 4.98e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.283 on 47 degrees of freedom
## Multiple R-squared:  0.4039, Adjusted R-squared:  0.3786
## F-statistic: 15.92 on 2 and 47 DF,  p-value: 5.242e-06
```

This model performed worse than the polynomial regression model of degree 3 since the RSE here is 5.283 where before it is 4.603. In addition, the R^2 statistic also went down.

Try using polynomial functions and step functions to find the best model using k-fold cross validation.

```
set.seed(3)
poly_mses = c()
for(i in 1:8){
  model = glm(data = df, y ~ poly(x1, i))
  poly_mses = c(poly_mses, cv.glm(df, model, K = 5)$delta[1])
}

cut_mses = c(80)
for(i in 2:8){
  df$cutted = cut(df$x1, i)
  model = glm(data = df, y ~ cutted)
  cut_mses = c(cut_mses, cv.glm(df, model, K=5)$delta[1])
}
mses_df = data.frame(x = 1:8,
                     "poly degree" = poly_mses,
                     "number of steps" = cut_mses)
mses_df %>% gather(parameter, value,
                   poly.degree, number.of.steps) %>%
  ggplot(aes(x = x, y = value,
             color = parameter)) +
  geom_path() +
  ggtitle("Testing Error as a Function of Parameter") +
  labs(x = "parameter value", y = "MSE") +
  scale_x_continuous(limits = c(1,8),
                     breaks = 1:8) +
  theme_minimal()
```



By using cross validation, it can be seen that adding many steps helped to lower MSE. After a number of steps, MSE then rose up, suggesting overfitting. As for the degrees of polynomial, a high degree does not help with lowering error; after 5 degrees, MSE went up. In fact, the best number of step is

```
which.min(cut_mses)
```

```
## [1] 8
```

and the best degree of polynomial is

```
which.min(poly_mses)
```

```
## [1] 5
```

According to k-fold cross validation, the best number of steps is 8 and the best number of degree of polynomial is 5.

Try using regression splines to improve the model.

Splines

First fit a cubic spline using three knots assigned by hand.

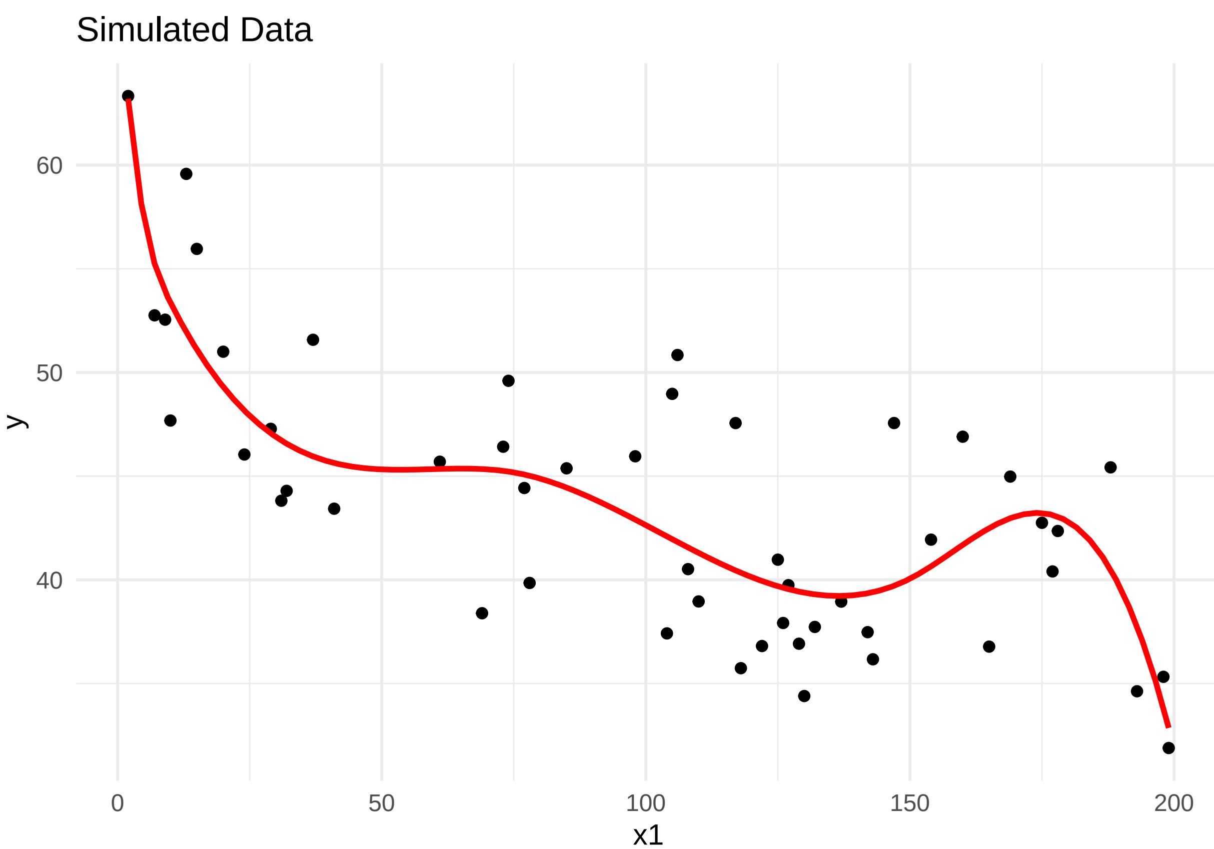
```
model3 = lm(y ~ bs(x1, knots = c(10, 75, 150)), data = df)
summary(model3)
```

```
##
## Call:
## lm(formula = y ~ bs(x1, knots = c(10, 75, 150)), data = df)
##
```

```
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.9538 -2.5051 -0.6724  1.6525  9.0150
##
## Coefficients:
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                       63.197      4.175  15.136  < 2e-16 ***
## bs(x1, knots = c(10, 75, 150))1   -6.920      5.142  -1.346 0.185370
## bs(x1, knots = c(10, 75, 150))2  -21.182      5.611  -3.775 0.000486 ***
## bs(x1, knots = c(10, 75, 150))3  -12.819      5.865  -2.185 0.034349 *
## bs(x1, knots = c(10, 75, 150))4  -31.149      5.485  -5.679 1.07e-06 ***
## bs(x1, knots = c(10, 75, 150))5  -14.402      5.372  -2.681 0.010363 *
## bs(x1, knots = c(10, 75, 150))6  -30.338      4.997  -6.072 2.88e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.179 on 43 degrees of freedom
## Multiple R-squared:  0.6588, Adjusted R-squared:  0.6112
## F-statistic: 13.84 on 6 and 43 DF,  p-value: 1.097e-08
```

This model appears to have one statistically significant coefficient estimates after six degrees. Plot the splines on the data.

```
basic_plot + stat_smooth(method = "lm", se = FALSE,
                         formula = y ~ bs(x, knots = c(10, 75, 150)),
                         color = "red")
```



Instead of supplying the knots itself, use quantiles to fit a natural spline on the data.

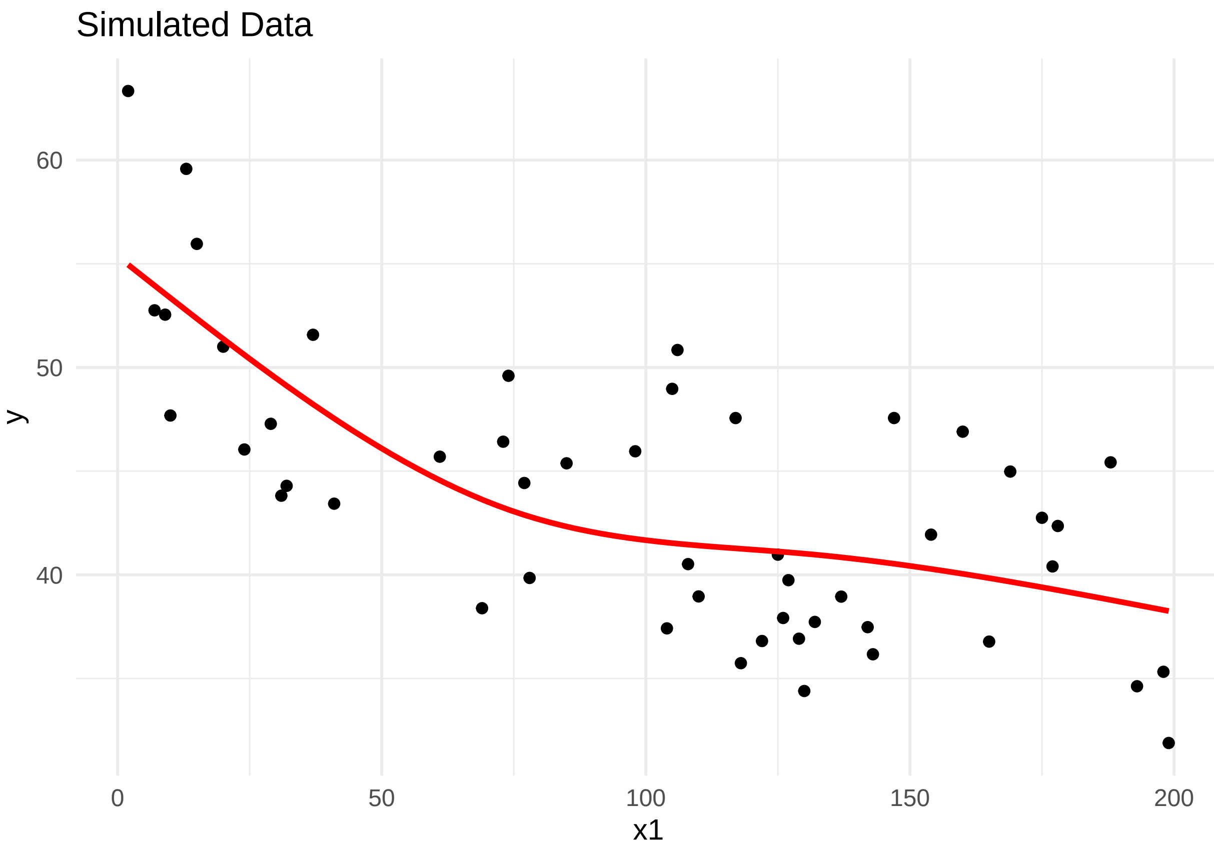
```r
model4 = lm(y ~ ns(x1, df = 3), data = df)
summary(model4)
```

```
##
## Call:
## lm(formula = y ~ ns(x1, df = 3), data = df)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -6.623 -4.060 -1.099  3.351  9.338
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)        54.954      1.994  27.555  < 2e-16 ***
## ns(x1, df = 3)1    -9.422      2.656  -3.547 0.000909 ***
## ns(x1, df = 3)2   -27.101      5.266  -5.146 5.38e-06 ***
## ns(x1, df = 3)3    -9.305      2.325  -4.003 0.000226 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.736 on 46 degrees of freedom
## Multiple R-squared:  0.5312, Adjusted R-squared:  0.5006
## F-statistic: 17.37 on 3 and 46 DF,  p-value: 1.105e-07
```

This model performs slightly worse than the previous regression spline model. R^2 went down to 0.50. The RSE went up to 4.736 from 4.179.

Plot the splines on the data.

```r
basic_plot + stat_smooth(method = "lm", se = FALSE,
                         formula = y ~ ns(x, df = 3),
                         color = "red")
```



This plot looks similar to the polynomial regression model of degree 3 where the right-hand side is somewhat flat. In the previous spline model, this section of the plot was more wavy. This makes sense, more variation in y was explained in the previous spline model than this model.

Now onto GAMs.

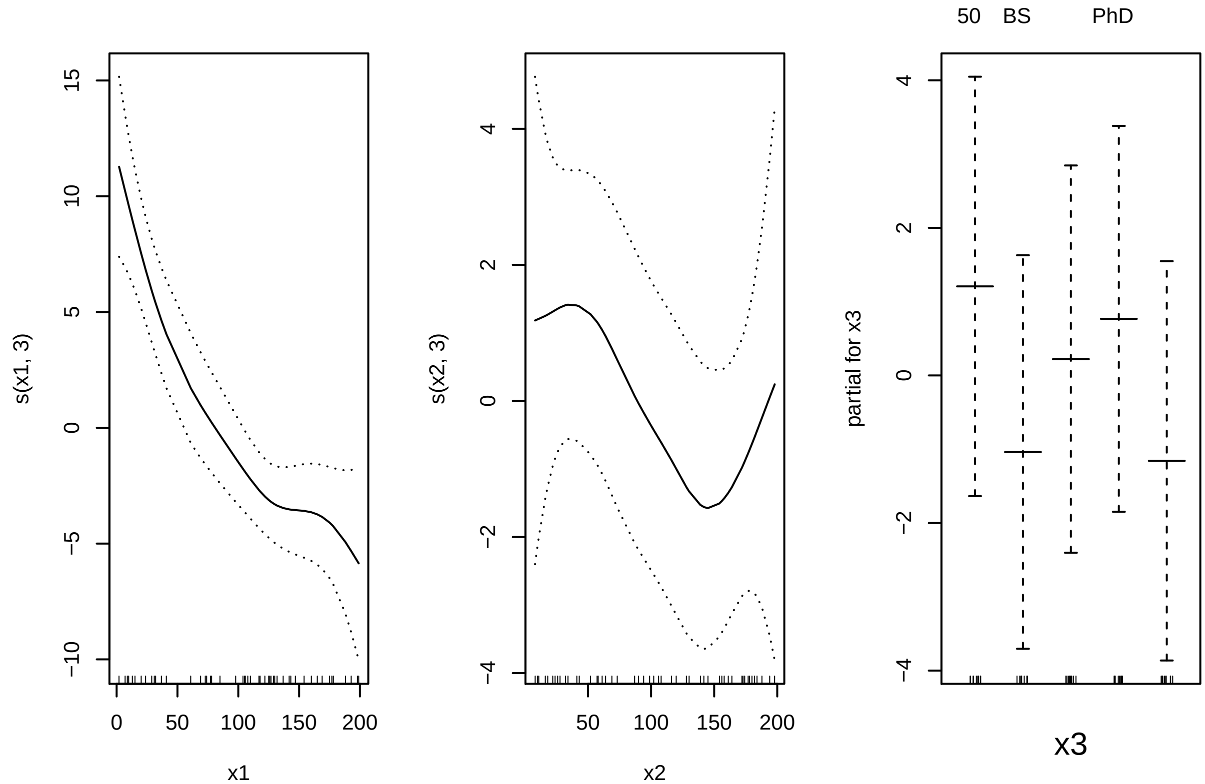
GAMs

Produce a GAM using all 3 predictors and smoothing splines of degree 3.

```
model5 = gam(y ~ s(x1, 3) + s(x2, 3) + x3, data = df)
```

The model can visualized below.

```
par(mfrow = c(1,3))
plot(model5, se = TRUE)
```



The first plot is nonlinear with respect to `x1`, as well as the second plot for `x2`. The third plot shows that as `x3` increases, or education increases, `y` increases. (This is a spurious correlation, `x3` was generated randomly.)

A series of ANOVA tests can be done to see which model of increasing variables is the best.

```
gam1 = gam(y ~ s(x1, 3), data = df)
gam2 = gam(y ~ s(x1, 3) + s(x2, 3), data = df)
anova(gam1, gam2, model5)
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ s(x1, 3)
## Model 2: y ~ s(x1, 3) + s(x2, 3)
## Model 3: y ~ s(x1, 3) + s(x2, 3) + x3
##   Resid. Df Resid. Dev     Df Deviance Pr(>Chi)
## 1        46     968.81
## 2        43     859.73 2.9999  109.079   0.1635
## 3        39     831.64 4.0000   28.093   0.8584
```

There is compeling evidene that a GAM with `x1`, `x2` and `x3` would be infavorable since its *p*-value is large. In fact, even just `x1` and `x2` is infavorable.

The summary of the model with all variables is shared below.

```
summary(model5)
```

```
##
## Call: gam(formula = y ~ s(x1, 3) + s(x2, 3) + x3, data = df)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -6.9771 -3.3412 -0.6163  3.4163  8.9403
##
```

```
## (Dispersion Parameter for gaussian family taken to be 21.324)
##
##     Null Deviance: 2201.064 on 49 degrees of freedom
## Residual Deviance: 831.6372 on 39.0001 degrees of freedom
## AIC: 306.4624
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##           Df  Sum Sq Mean Sq F value    Pr(>F)
## s(x1, 3)   1 1031.99 1031.99 48.3956 2.458e-08 ***
## s(x2, 3)   1   47.25   47.25  2.2157    0.1447
## x3         4   39.46    9.86  0.4626    0.7627
## Residuals 39  831.64   21.32
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##             Npar Df Npar F   Pr(F)
## (Intercept)
## s(x1, 3)          2 4.4727 0.01783 *
## s(x2, 3)          2 0.9602 0.39168
## x3
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

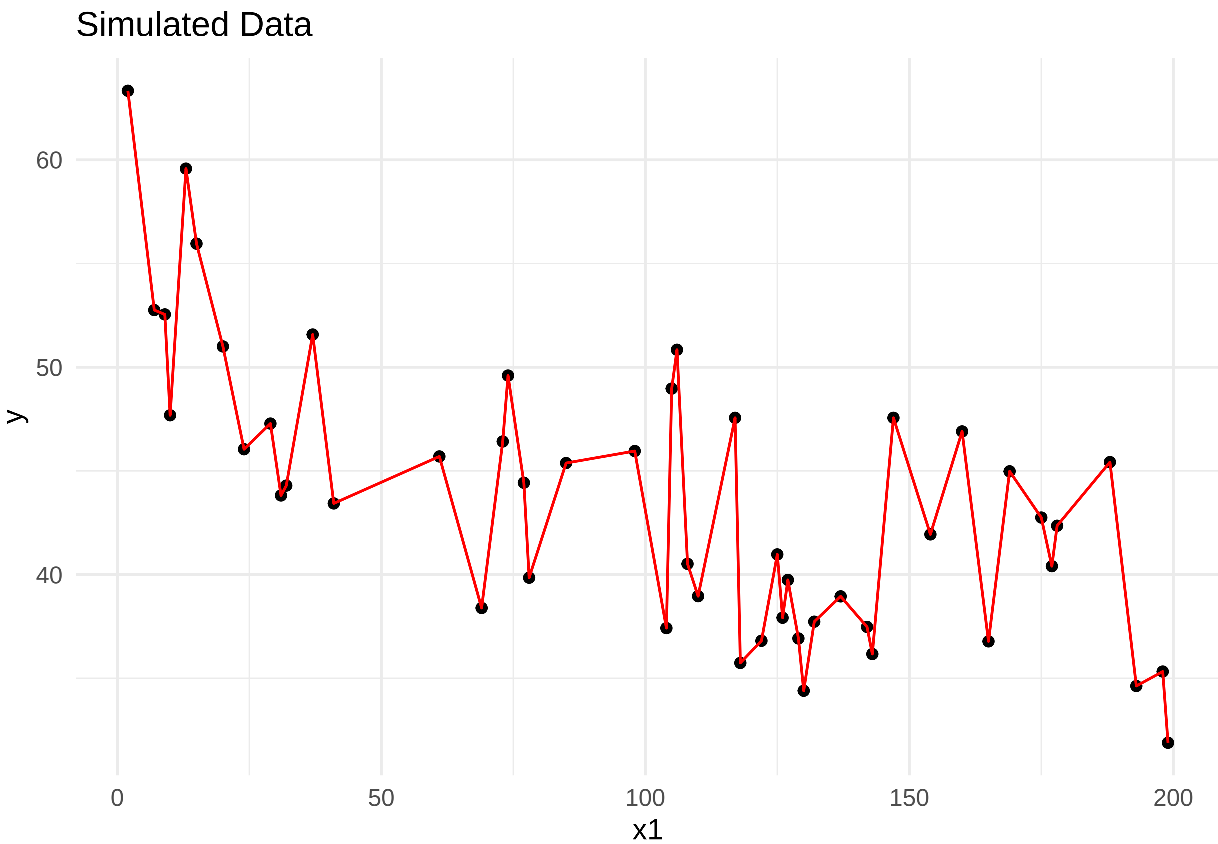
The p-value for `x1` is close to 0, meaning that the the coefficient estimate is statistically significant. For `x2`, it is statistically significant. `x3` also has a high p-value, which agrees with the ANOVA test above.

It would be best to use a GAM with just `x1` with 3 splines.

Bonus: Spline of degree n

A model of n observations can be fit with a degree n spline where the splines would go through each point. This would simple be a line between two consecutive points in increasing x value. Let's see how that looks.

```
basic_plot + geom_line(color = "red")
```



Interesting. This model would of course be overfitting the data.

All of the lab instructions in this document are taken from “An Introduction to Statistical Learning, with applications in R” (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.