Chapter 7 Problem 8

Andira Putri

Fit some of the non-linear models investigated in this chapter to the `Auto` data set. Is there evidence for non-linear relationships in this data set? Create informative plots to justify your answer.

The methods introduced in this chapter are polynomial regression, step functions, regression splines, smoothing splines, local regression, and generalized additive models (GAMs).

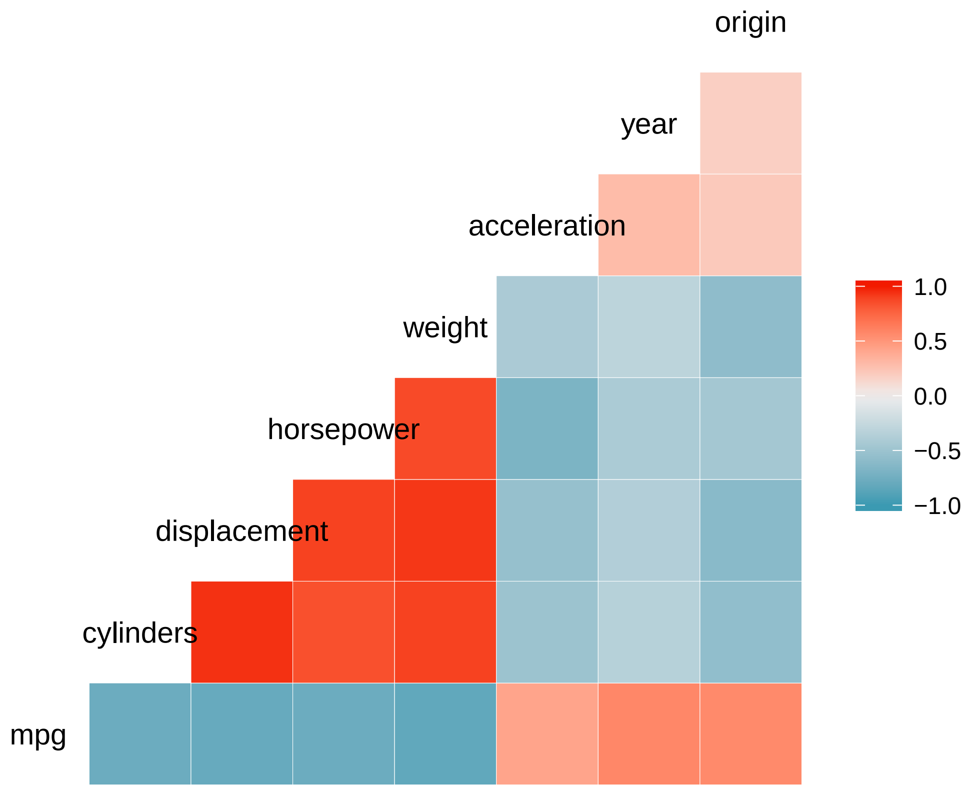
I choose to apply polynomial regression, step functions, regression splines, and GAMs to the `Auto` data set, with `mpg` as a response variable.

```
set.seed(1)
library(ISLR)
#exclude name variable in Auto data set
auto=subset(Auto,select=c(-name))
#exploratory analysis of data with heatmap
#red-->positive correlation,blue-->neg corr
require(GGally)
```

```
## Loading required package: GGally
```

```
## Loading required package: ggplot2
```

```
ggcorr(data=auto)
```



We look at the base of the heatmap for pairwise correlations associated with `mpg`. From the heatmap, `mpg` is negatively correlated with `cylinders`,`displacement`,`horsepower`, and `weight`, so these would be interesting predictors to use.

Polynomial Regression

Polynomial regression is just an extension of linear regression; coefficients can be easily estimated from least squares. The degree of polynomial functions is usually not greater than 3 or 4 because the model becomes too flexible and can take on strange shapes. Some examples of polynomial function used in this regression method are:

Linear model (linear regression): $Y = \beta_0 + \beta_1 X$

Quadratic model: $Y = \beta_0 + \beta_1 X + \beta_2 X^2$

Quartic model: $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \beta_4 X^4$

We test models up to degree 5 and use `weight` as a predictor. We analyze the results using ANOVA tables, cross-validation, and visualizations.

```
fit.1=lm(mpg~weight,data=auto) #linear regression
fit.2=lm(mpg~poly(weight,2),data=auto) #quadratic
fit.3=lm(mpg~poly(weight,3),data=auto) #cubic
fit.4=lm(mpg~poly(weight,4),data=auto) #quartic
fit.5=lm(mpg~poly(weight,5),data=auto) #quintic
#Generate ANOVA table
anova(fit.1,fit.2,fit.3,fit.4,fit.5)
```

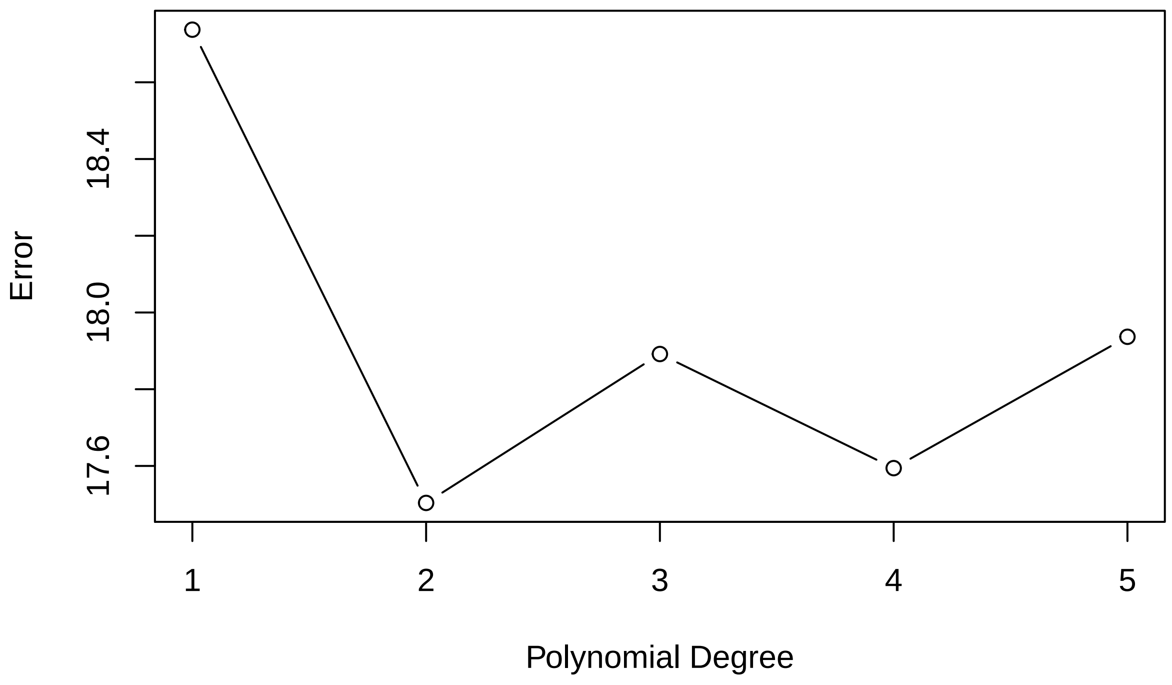
```
## Analysis of Variance Table
##
## Model 1: mpg ~ weight
## Model 2: mpg ~ poly(weight, 2)
## Model 3: mpg ~ poly(weight, 3)
## Model 4: mpg ~ poly(weight, 4)
## Model 5: mpg ~ poly(weight, 5)
##   Res.Df    RSS Df Sum of Sq       F    Pr(>F)
## 1    390 7321.2
## 2    389 6784.9  1    536.34 30.6147 5.817e-08 ***
## 3    388 6784.8  1      0.05  0.0028    0.9580
## 4    387 6777.0  1      7.88  0.4500    0.5027
## 5    386 6762.3  1     14.67  0.8374    0.3607
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

From the ANOVA table, the quadratic model seems to be the best out of the five due to the low p-value. We check using cross-validation and plots.

```
set.seed(1)
library(boot)
#initialize vector
cv=rep(0,5)
#k-fold cross validation with k=5
for (i in 1:5) {
  fit=glm(mpg~poly(weight,i),data=auto)
  cv[i]=cv.glm(auto,fit,K=5)$delta
}
```

```
## Warning in cv[i] <- cv.glm(auto, fit, K = 5)$delta: number of items to
## replace is not a multiple of replacement length

## Warning in cv[i] <- cv.glm(auto, fit, K = 5)$delta: number of items to
## replace is not a multiple of replacement length

## Warning in cv[i] <- cv.glm(auto, fit, K = 5)$delta: number of items to
## replace is not a multiple of replacement length

## Warning in cv[i] <- cv.glm(auto, fit, K = 5)$delta: number of items to
## replace is not a multiple of replacement length

## Warning in cv[i] <- cv.glm(auto, fit, K = 5)$delta: number of items to
## replace is not a multiple of replacement length
```

```
#plot error of all degrees
plot(cv,type="b",xlab="Polynomial Degree",ylab="Error")
```



Cross-validation also supports that the best model is quadratic, with the quartic model being a close second. This is interesting because in the ANOVA table, the second best model was the degree 5 model. The test MSE approximation for the quadtratic model is a little over 17.5. Just throwing it out there now that since the test error for degree 1 is so high, there is indeed a non-linear relationship in the `Auto` datset. Now, let's step over to the step function!

Step Functions

Step functions break the range of X into bins. It specifically creates cutpoints $c_1, ..., c_k$ in the range of X and constructs k+1 new variables. Visually, you can think of step functions as stairs, where the cutpoints are where you take a next step up or down. The cutpoints are represented in a piecewise way:

$c_o(x) = I(x < c_1)$

$c_1(x) = I(c_1 \leq x < c_2)$

...

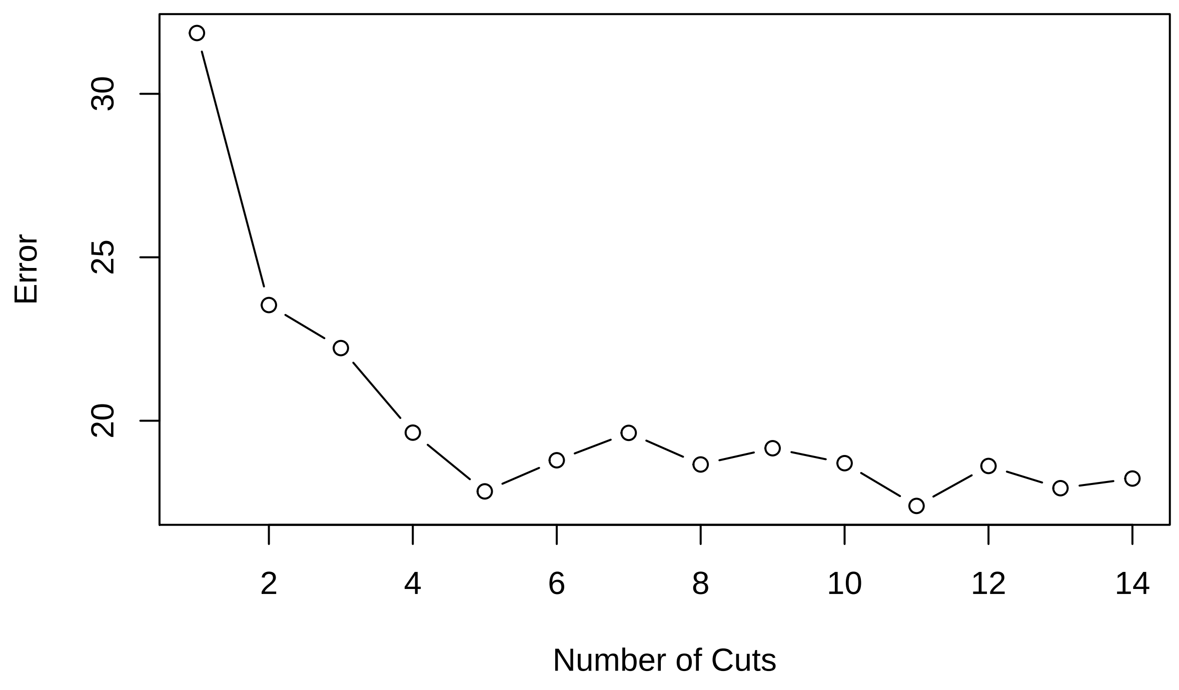
$c_{k-1}(x) = Ic_{k-1} \leq x < c_k)$

$c_k(x) = I(c_k \leq x)$

where a 1 is returned when the indicator function I is true for a particular value of x. x must be in exactly one of $k + 1$ variables. The step function model is fit using least squares with our cutpoints above as predictors. Our model is represented as:

$y_i = \beta_0 + \beta_1 c_1(x) + ... + \beta_k c_k(x) + \epsilon_i$

```r
#Unlike polynomial regression, we do model fitting and CV in one for loop
#initialize vector
cv=rep(0,10)
#model fitting and CV
for (i in 2:15) {
  auto$weight.cut=cut(auto$weight,i)
  step=glm(mpg~weight.cut,data=auto)
  cv[i-1]=cv.glm(auto,step,K=5)$delta[2]
}
#analyze plot and CV
plot(cv,type="b",xlab="Number of Cuts",ylab="Error")
```



```r
which.min(cv)
```

```
## [1] 11
```

```r
#MSE of best # of cuts
auto$weight.cut=cut(auto$weight,11)
cv.glm(auto,step,K=5)$delta[2]
```

```
## [1] 18.02088
```

CV gives us that 11 cuts is the best for our step function model, however the approximated test error is higher than that of the quadratic model in our polynomial regression analysis. Still, this does suggest non-linearity is in our `Auto` data set.

Regression Splines

Regression splines are kind of like a hybrid of polynomial regression and stepwise functions. Like stepwise functions, regression splines feature piecewise representations; low-degree polynomials are popular choices for each piece. The coefficients of the polynomial changes at a point called the knot.

For example, a piecewise cubic polynomial with one knot at c is represented as:

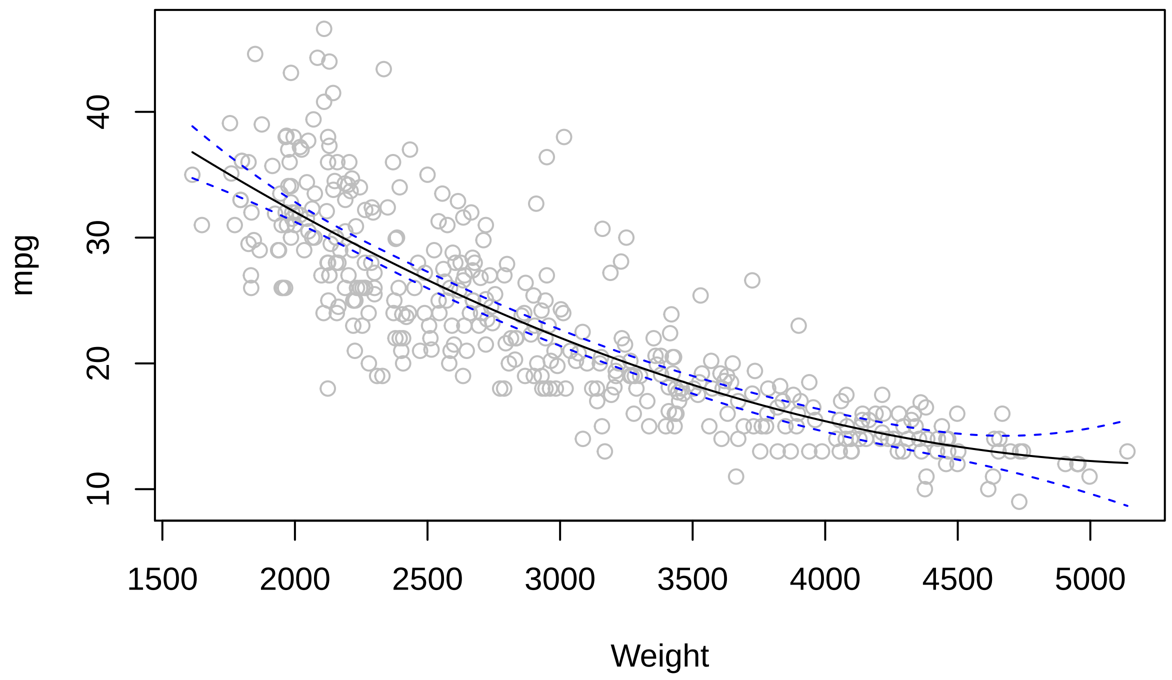
$$y_i = \begin{cases} \beta_{01} + \beta_{11}x_i + \beta_{21}x_i^2 + \beta_{31}x_i^3 & x_i < c \\ \beta_{02} + \beta_{12}x_i + \beta_{22}x_i^2 + \beta_{32}x_i^3 & x_i \geq c \end{cases}$$

Of course, not all functions are going to be polynomial. One can choose any known function for the different regions of x; these functions are called basis functions. Be careful because some models can lead to discontinuity, which won't make much sense. The point of the regression spline is to join together piecewise polynomials and create a smooth curve. This bypasses the weird effects of high-degree polynomials while giving more flexibility than step functions.

```r
library(splines)
weightlims=range(auto$weight)
weight.grid=seq(from=weightlims[1],to=weightlims[2])
#fit cubic polynomial spline
spl.fit=lm(mpg~bs(weight,knots=c(25,40,60)),data=auto)
pred=predict(spl.fit,newdata=list(weight=weight.grid),se=T)
```

```
## Warning in predict.lm(spl.fit, newdata = list(weight = weight.grid), se =
## T): prediction from a rank-deficient fit may be misleading
```

```r
#plot spline and standard error
plot(auto$weight,auto$mpg,col="gray",xlab="Weight",ylab="mpg")
lines(weight.grid,pred$fit)
lines(weight.grid,pred$fit+2*pred$se,col="blue",lty="dashed")
lines(weight.grid,pred$fit-2*pred$se,col="blue",lty="dashed")
```



Generalized Additive Models

GAMs extend the standard linear model by allowing non-linear functions for each of the variables while still maintaining additivity. There is no need to manually transform the variables since we can model non-linear relationships, we can make more accurate predictions than linear regression, and we can still examine the effect of each observation on Y individually while fixing other predictors. However, we cannot account for any interactions unless we manually add them in.

For our GAM on the `Auto` data set, I use three predictors: `weight`, `displacement`, and `horsepower`. Each predictor has 4 degrees of freedom.

```
library(gam)
```

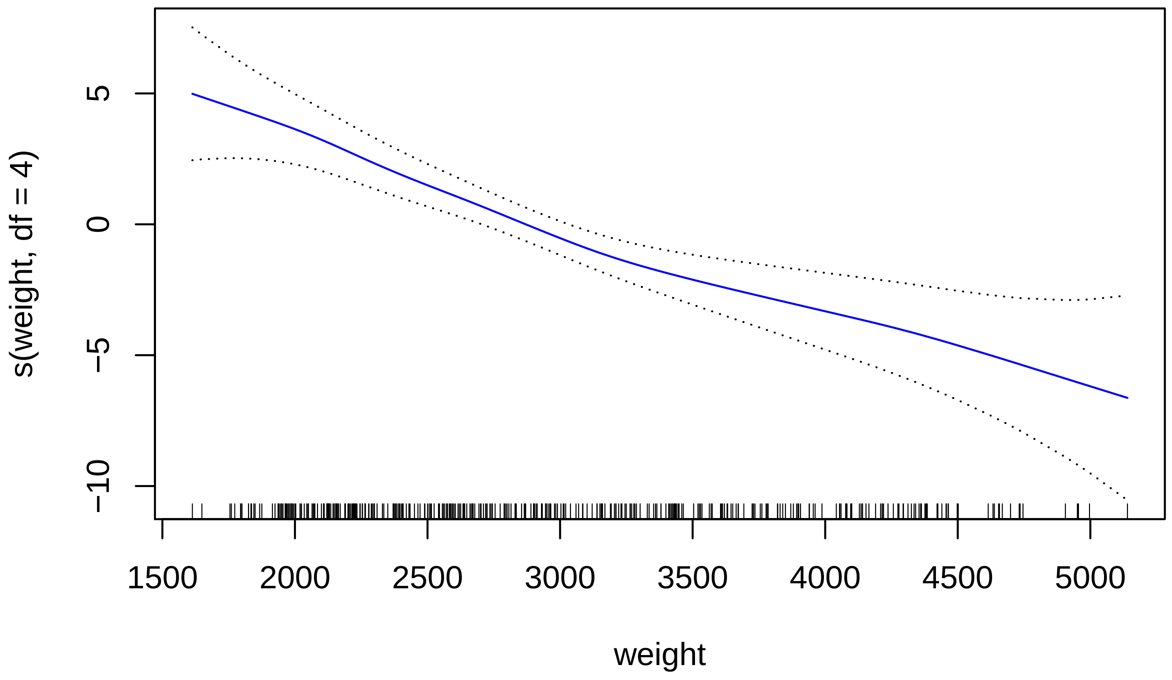
```
## Loading required package: foreach

## Loaded gam 1.16
```

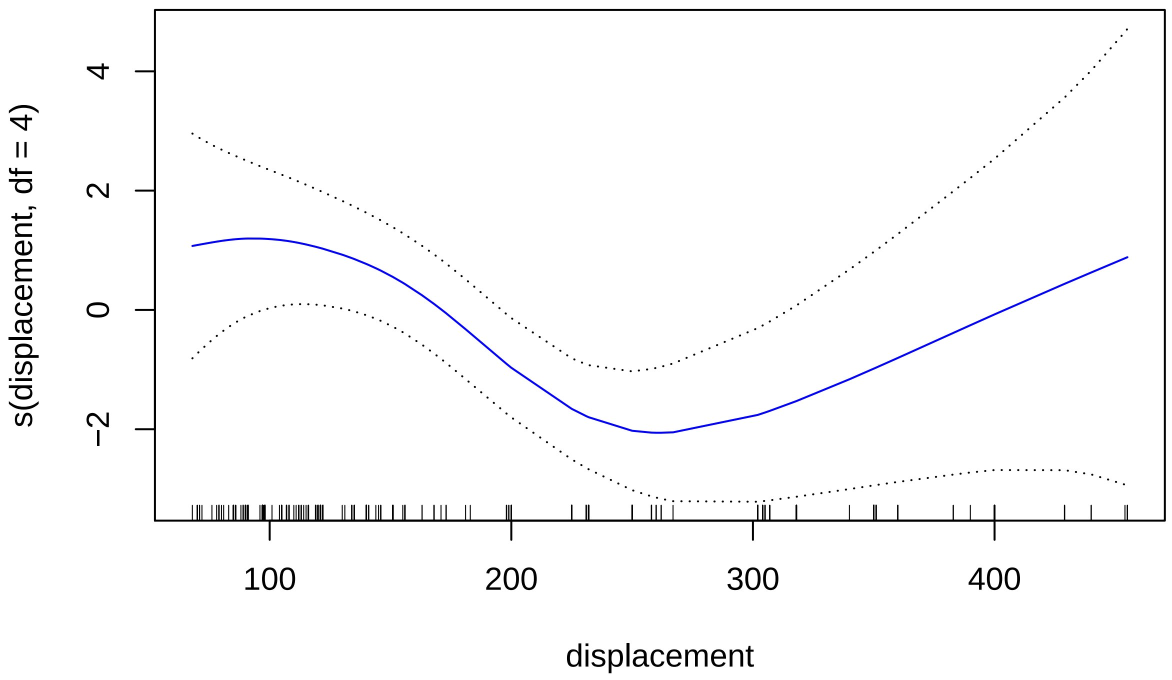
```
#fit GAM
gam.fit = gam(mpg~s(weight,df=4)+s(displacement,df=4)+s(horsepower,df=4),data=auto)
summary(gam.fit)
```

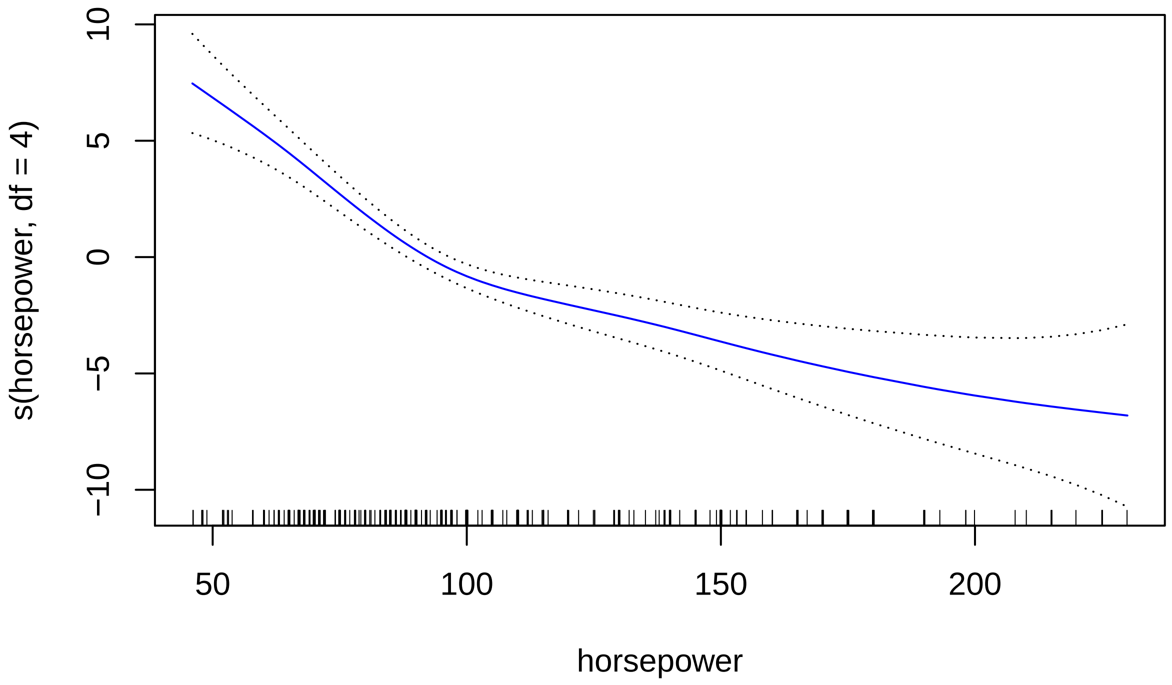
```
##
## Call: gam(formula = mpg ~ s(weight, df = 4) + s(displacement, df = 4) +
##     s(horsepower, df = 4), data = auto)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -10.4139  -2.1021  -0.2776   1.8289  16.1705
##
## (Dispersion Parameter for gaussian family taken to be 14.6736)
```

```
##
##     Null Deviance: 23818.99 on 391 degrees of freedom
## Residual Deviance: 5561.308 on 378.9999 degrees of freedom
## AIC: 2180.16
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##                            Df  Sum Sq Mean Sq  F value    Pr(>F)
## s(weight, df = 4)           1 15546.9 15546.9 1059.510 < 2.2e-16 ***
## s(displacement, df = 4)     1   459.7   459.7   31.331 4.181e-08 ***
## s(horsepower, df = 4)       1   683.7   683.7   46.592 3.482e-11 ***
## Residuals                 379  5561.3    14.7
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                           Npar Df  Npar F     Pr(F)
## (Intercept)
## s(weight, df = 4)               3  1.3419 0.2604245
## s(displacement, df = 4)         3  6.9181 0.0001519 ***
## s(horsepower, df = 4)           3 11.2166 4.564e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
#plot each fit
plot(gam.fit,se=T,col="blue")
```




s(displacement, df = 4)
4
2
0
−2
100
200
300
400
displacement



The small p-values resulting from each fit suggest non-linearity in the data.