

Splines with Cross-Validation

Gustavo Cepparo and Milica Cudina

First, we need our textbook's library:

```
library(ISLR2)
```

Next, we look at the included data set:

```
data<-Auto
attach(Auto)
dim(Auto)
```

```
## [1] 392 9
```

We will split the data set into training and validation.

```
#setting the seed for comparable results
set.seed(1)
#first we create the set of indices of what will go into the training set
train <- sample(length(mpg), floor(length(mpg)/2))
#now we put the data with the above indices
#into the training set
training<-data[train,]
dim(training)
```

```
## [1] 196 9
```

```
#the complement of the above indices designates
#the validation set
val<-data[-train,]
dim(val)
```

```
## [1] 196 9
```

Next, we import the library `splines`.

```
library(splines)
```

Making a **linear** fit is the same thing as fitting a smooth spline with

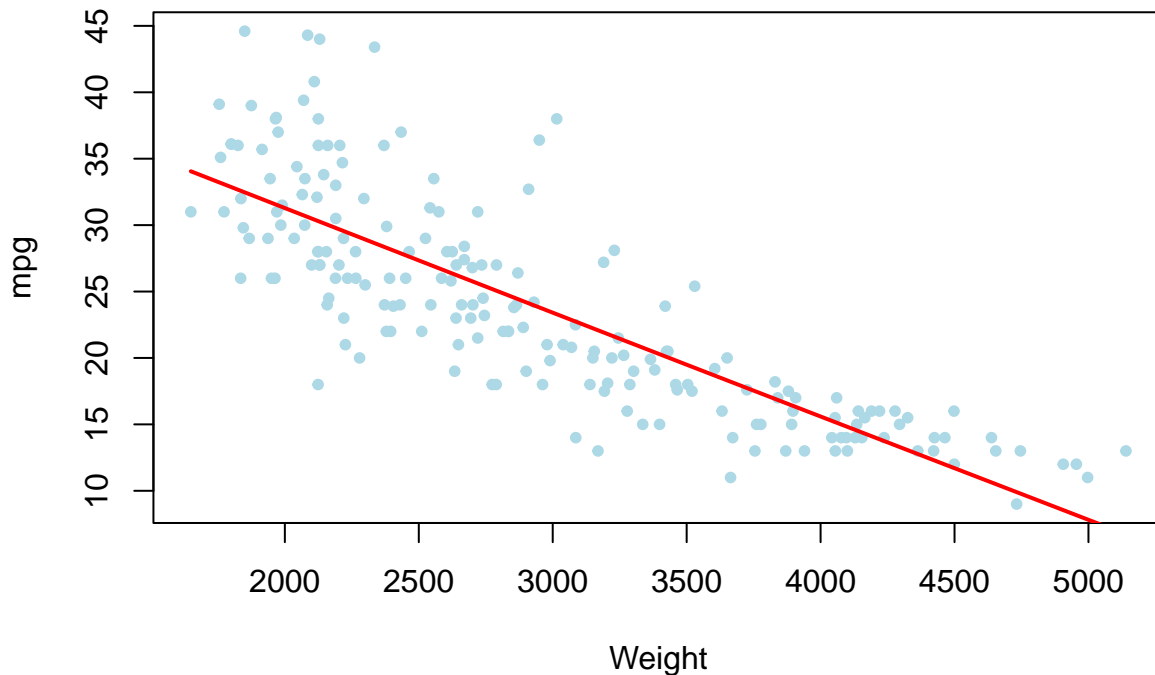
2 degrees of freedom (corresponding to parameters β_0 and β_1).

```
lin.fit=smooth.spline(training$weight,training$mpg,df=2)
lin.fit
```

```
## Call:
## smooth.spline(x = training$weight, y = training$mpg, df = 2)
##
## Smoothing Parameter spar= 1.499947 lambda= 16.04335 (30 iterations)
## Equivalent Degrees of Freedom (Df): 2.019589
## Penalized Criterion (RSS): 3827.151
## GCV: 21.48418
```

```
plot(training$weight,training$mpg,
      col="lightblue", pch=20,
      main="Dependence of efficiency on weight",
      xlab="Weight", ylab="mpg")
lines(lin.fit,col="red", lwd=2)
```

Dependence of efficiency on weight



```
#lm.fit=lm(training$mpg ~ training$weight)
#abline(lm.fit, col="blue")
#what about the fit on the training set?
pred.t<-predict(lin.fit, training$weight)
#what is the structure of `pred.t` here?
pred.t
```

```
## $x
## [1] 2085 2639 2451 3420 2745 4190 2556 4100 1990 3465 1845 3205 2145 2189 3870
## [16] 3288 4654 1825 2575 3169 3605 2585 3777 4054 1755 2950 3015 4955 2124 1835
## [31] 5140 2065 4422 3399 4154 2545 2648 3365 3651 4464 3190 2990 3897 3086 1836
## [46] 2962 2430 3150 4325 3085 4906 2525 3430 1875 1760 4997 2910 3761 3530 2160
## [61] 2694 3245 2379 2100 4732 1963 3155 4060 2279 2670 2634 2215 2391 3830 1950
## [76] 2125 2158 2110 2789 4135 3520 2620 2074 2335 1970 4638 2295 2130 3265 1649
## [91] 3725 2702 3139 2155 2670 2234 2464 3755 3425 3039
## [ reached 'max' / getOption("max.print") -- omitted 96 entries ]
##
## $y
## [1] 30.613641 26.247461 27.728022 20.118214 25.413403 14.112622 26.900928
## [8] 14.813091 31.362994 19.766286 32.506809 21.801551 30.140402 29.793390
## [15] 16.604660 21.151326 10.504439 32.664578 26.751312 22.083724 18.672246
## [22] 26.672573 17.329807 15.171223 33.216772 23.802147 23.291788 8.165163
## [29] 30.306031 32.585694 6.727539 30.771395 12.308027 20.282493 14.392777
## [36] 26.987555 26.176622 20.548532 18.313047 11.981457 21.919112 23.488049
```

```
## [43] 16.394218 22.734627 32.577805 23.707907 27.893491 22.232684 13.062377
## [50] 22.742472 8.545948 27.145072 20.039996 32.270154 33.177330 7.838780
## [57] 24.116343 17.454611 19.258183 30.022099 25.814622 21.488130 28.295407
## [64] 30.495328 9.898196 31.575976 22.193482 15.124505 29.083707 26.003478
## [71] 26.286817 29.588353 28.200830 16.916494 31.678524 30.298144 30.037873
## [78] 30.416454 25.067367 14.540653 19.336335 26.397022 30.700405 28.642224
## [85] 31.520758 10.628803 28.957560 30.258708 21.331461 34.052948 17.735475
## [92] 25.751677 22.318936 30.061533 26.003478 29.438527 27.625597 17.501417
## [99] 20.079104 23.103416
## [ reached 'max' / getOption("max.print") -- omitted 96 entries ]
```

```
pred.t$x
```

```
## [1] 2085 2639 2451 3420 2745 4190 2556 4100 1990 3465 1845 3205 2145 2189 3870
## [16] 3288 4654 1825 2575 3169 3605 2585 3777 4054 1755 2950 3015 4955 2124 1835
## [31] 5140 2065 4422 3399 4154 2545 2648 3365 3651 4464 3190 2990 3897 3086 1836
## [46] 2962 2430 3150 4325 3085 4906 2525 3430 1875 1760 4997 2910 3761 3530 2160
## [61] 2694 3245 2379 2100 4732 1963 3155 4060 2279 2670 2634 2215 2391 3830 1950
## [76] 2125 2158 2110 2789 4135 3520 2620 2074 2335 1970 4638 2295 2130 3265 1649
## [91] 3725 2702 3139 2155 2670 2234 2464 3755 3425 3039
## [ reached 'max' / getOption("max.print") -- omitted 96 entries ]
```

```
pred.t$y
```

```
## [1] 30.613641 26.247461 27.728022 20.118214 25.413403 14.112622 26.900928
## [8] 14.813091 31.362994 19.766286 32.506809 21.801551 30.140402 29.793390
## [15] 16.604660 21.151326 10.504439 32.664578 26.751312 22.083724 18.672246
## [22] 26.672573 17.329807 15.171223 33.216772 23.802147 23.291788 8.165163
## [29] 30.306031 32.585694 6.727539 30.771395 12.308027 20.282493 14.392777
## [36] 26.987555 26.176622 20.548532 18.313047 11.981457 21.919112 23.488049
## [43] 16.394218 22.734627 32.577805 23.707907 27.893491 22.232684 13.062377
## [50] 22.742472 8.545948 27.145072 20.039996 32.270154 33.177330 7.838780
## [57] 24.116343 17.454611 19.258183 30.022099 25.814622 21.488130 28.295407
## [64] 30.495328 9.898196 31.575976 22.193482 15.124505 29.083707 26.003478
## [71] 26.286817 29.588353 28.200830 16.916494 31.678524 30.298144 30.037873
## [78] 30.416454 25.067367 14.540653 19.336335 26.397022 30.700405 28.642224
## [85] 31.520758 10.628803 28.957560 30.258708 21.331461 34.052948 17.735475
## [92] 25.751677 22.318936 30.061533 26.003478 29.438527 27.625597 17.501417
## [99] 20.079104 23.103416
## [ reached 'max' / getOption("max.print") -- omitted 96 entries ]
```

```
#the mean squared error of the predicted values
```

```
#in the training set
```

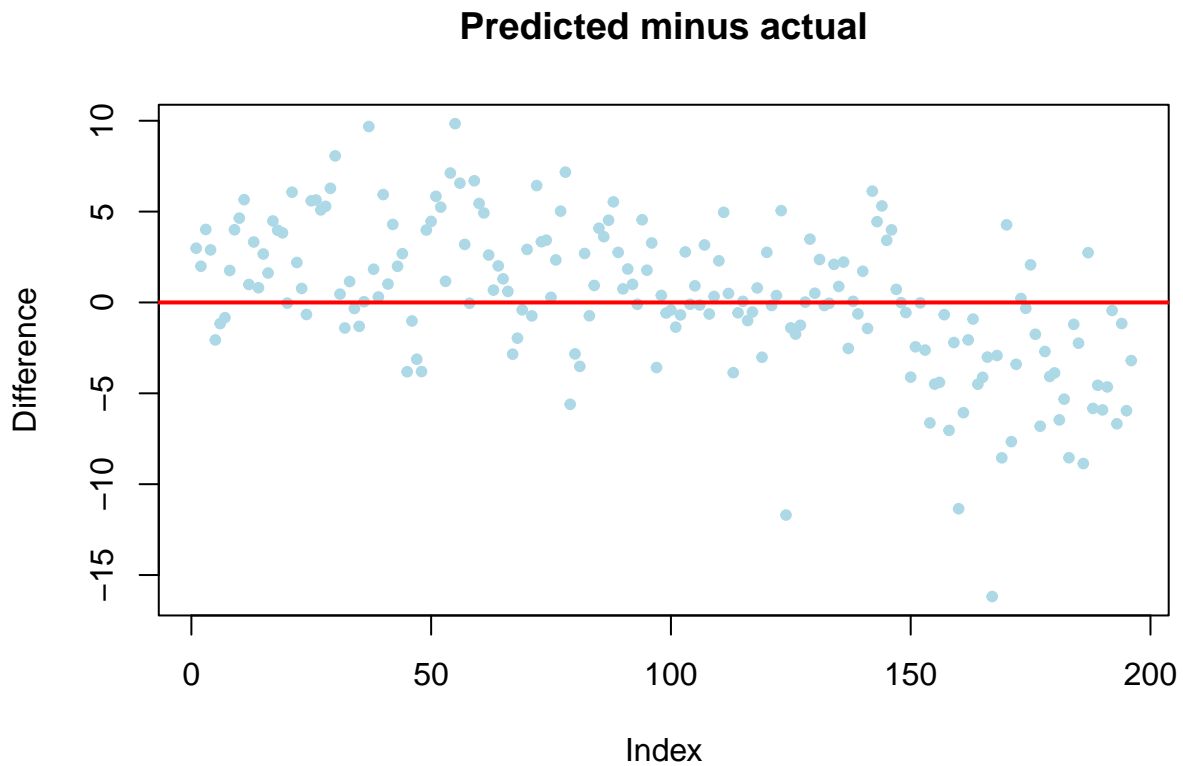
```
mean((pred.t$y-training$mpg)^2)
```

```
## [1] 21.04371
```

Let's see how this fit performs on the validation set.

```
pred <- predict(lin.fit, val$weight)
#what does `predict` give us in this case?
#pred
#plot of how "off" the predictions are
plot(pred$y-val$mpg,
     col="lightblue", pch=20,
     main="Predicted minus actual",
     xlab="Index", ylab="Difference")
```

```
abline(0,0, col="red", lwd=2)
```



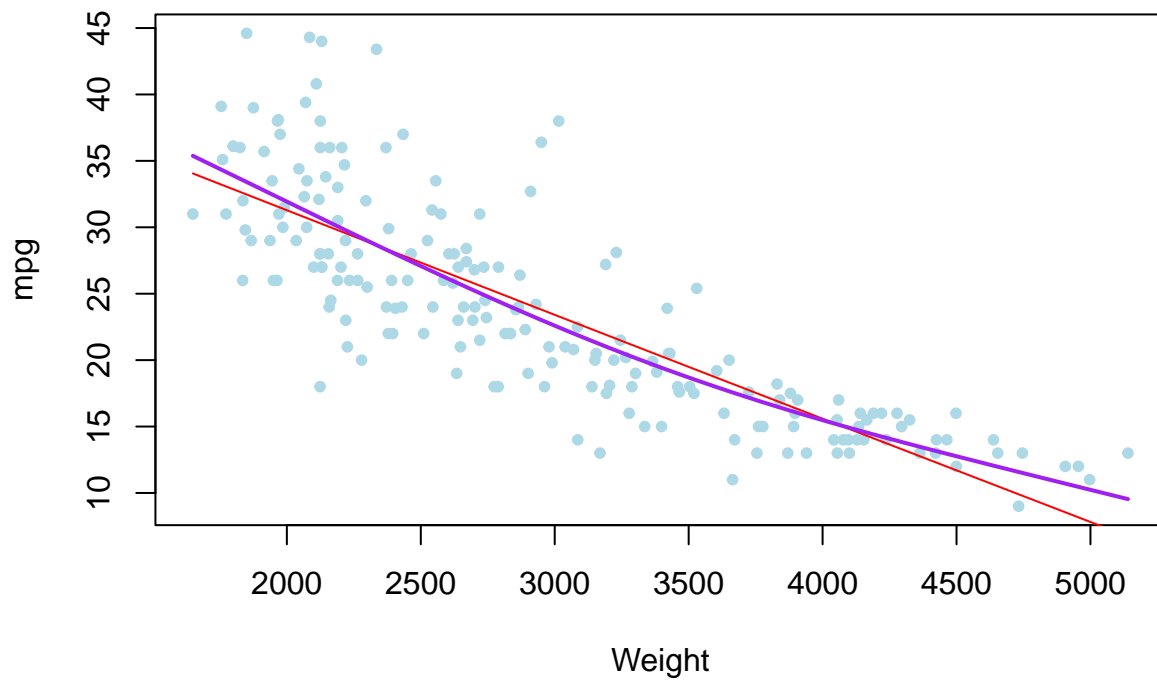
```
#the mean squared error of the predicted values  
#on the validation set  
mean((pred$y-val$mpg)^2)
```

```
## [1] 16.32753
```

How would the **quadratic** fit work? That's what we get with 3 degrees of freedom.

```
q.fit=sMOOTH.spline(training$weight,training$mpg,df=3)  
  
plot(training$weight,training$mpg,  
      col="lightblue", pch=20,  
      main="Dependence of efficiency on weight",  
      xlab="Weight", ylab="mpg")  
lines(lin.fit, col="red")  
lines(q.fit,col="purple", lwd=2)
```

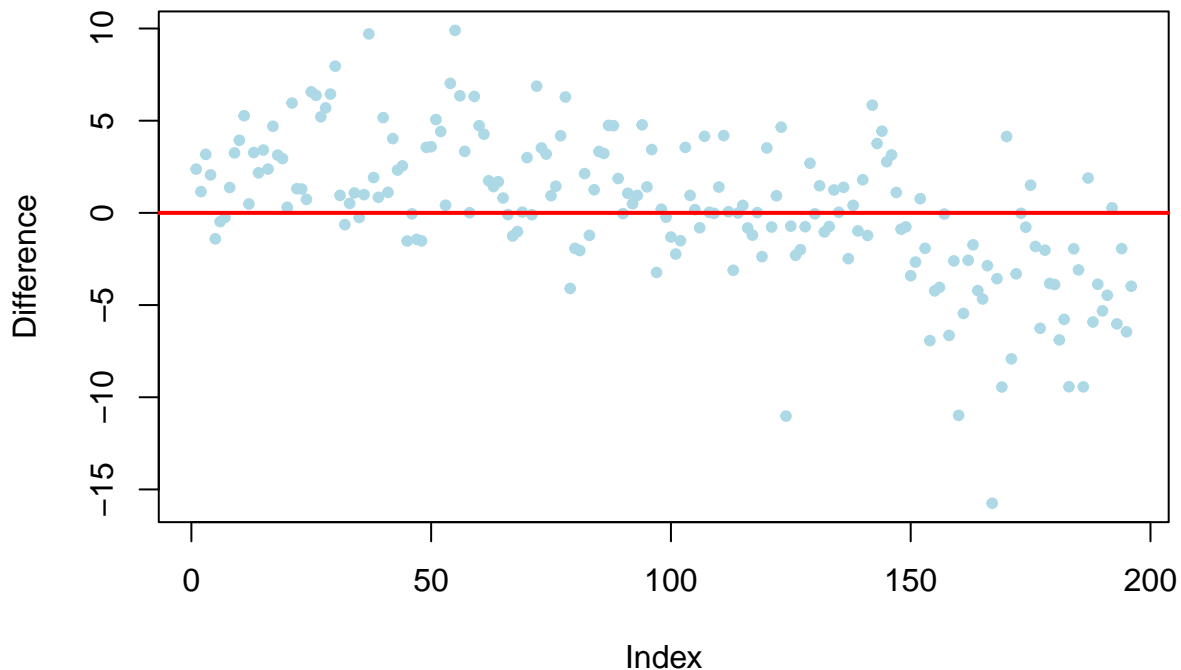
Dependence of efficiency on weight



How well did we do on the validation set?

```
#as before, we create the predicted values
pred <- predict(q.fit, val$weight)
#plot of how "off" the predictions are
plot(pred$val$mpg,
     col="lightblue", pch=20,
     main="Predicted minus actual",
     xlab="Index", ylab="Difference")
abline(0,0, col="red", lwd=2)
```

Predicted minus actual



```
#the mean squared error of the predicted values
#in the validation set
mean((pred$y-val$mpg)^2)
```

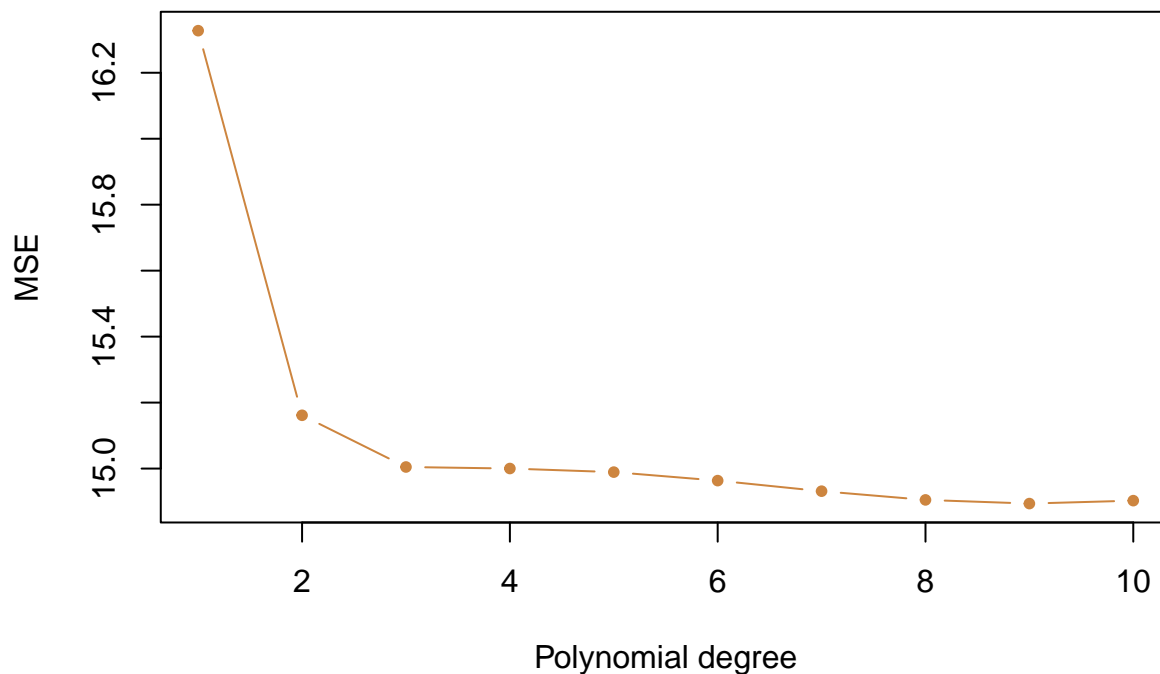
```
## [1] 15.1615
```

It might be fun to go through a for loop and see what the MSEs on the validation set are as we increase the number of degrees of freedom.

```
#first we choose the maximal number of degrees of freedom
max.deg=10
#now, we create a numeric vector which will contain
#validation set MSEs
MSEs=numeric(max.deg)
#next, we repeat the procedures we had before `max.deg` times
for (deg in 1:max.deg){
  fit=smooth.spline(training$weight,training$mpg,df=deg+1)
  pred <- predict(fit, val$weight)
  MSEs[deg]=mean((pred$y-val$mpg)^2)
}
print(MSEs)
```

```
## [1] 16.32753 15.16150 15.00475 15.00011 14.98925 14.96347 14.93146 14.90495
## [9] 14.89386 14.90271
```

```
#let's plot the MSEs as they depend
#on the degree of the polynomial
plot(MSEs, type="b",
     xlab="Polynomial degree",
     ylab="MSE",
     col="peru", pch=20)
```



What would we get if we let R optimize the number of degrees of freedom using LOOCV?

```
fit.cv=smooth.spline(training$weight,training$mpg,cv=T)
```

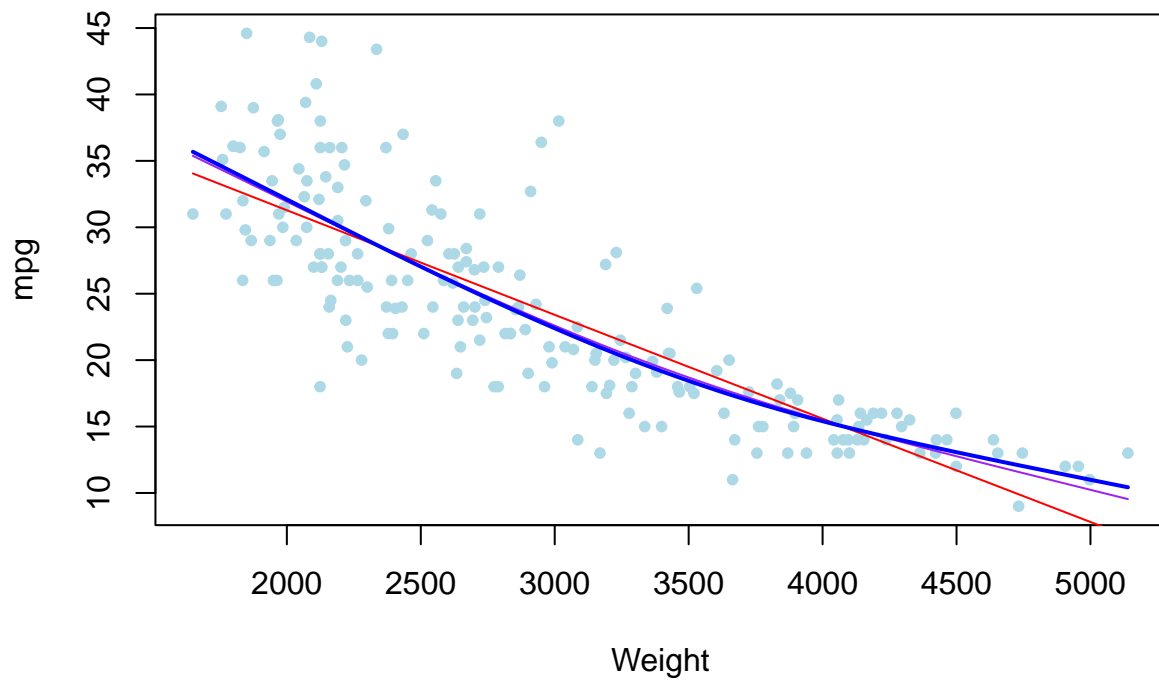
```
## Warning in smooth.spline(training$weight, training$mpg, cv = T):  
## cross-validation with non-unique 'x' values seems doubtful
```

```
#summary(fit.cv)  
fit.cv$df
```

```
## [1] 3.548739
```

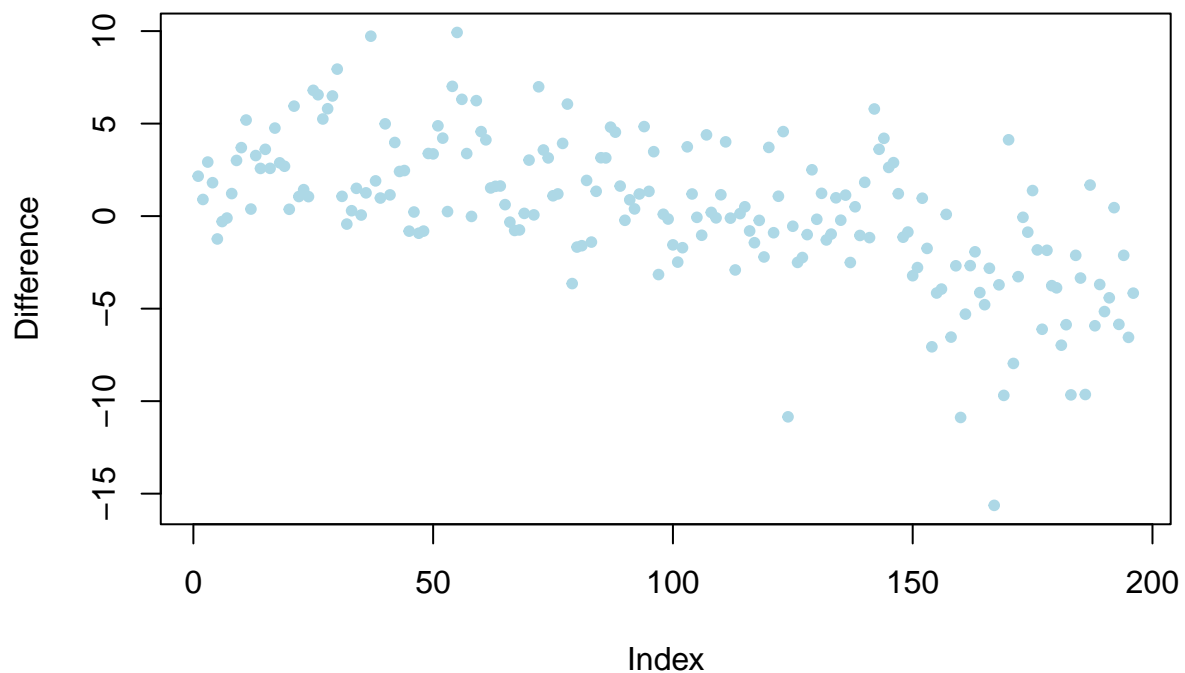
```
plot(training$weight,training$mpg,  
      col="lightblue", pch=20,  
      main="Dependence of efficiency on weight",  
      xlab="Weight", ylab="mpg")  
lines(lin.fit, col="red")  
lines(q.fit,col="purple")  
lines(fit.cv,col="blue",lwd=2)
```

Dependence of efficiency on weight



```
pred <- predict(fit.cv, val$weight)
plot(pred$y-val$mpg, pch=20, col="lightblue",
     xlab="Index", ylab="Difference",
     main="Predicted minus actual")
```

Predicted minus actual




```
mean((pred$y-val$mpg)^2)
```

```
## [1] 15.02952
```