Splines with Cross-Validation

Gustavo Cepparo and Milica Cudina

First, we need some libraries:

```
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`.

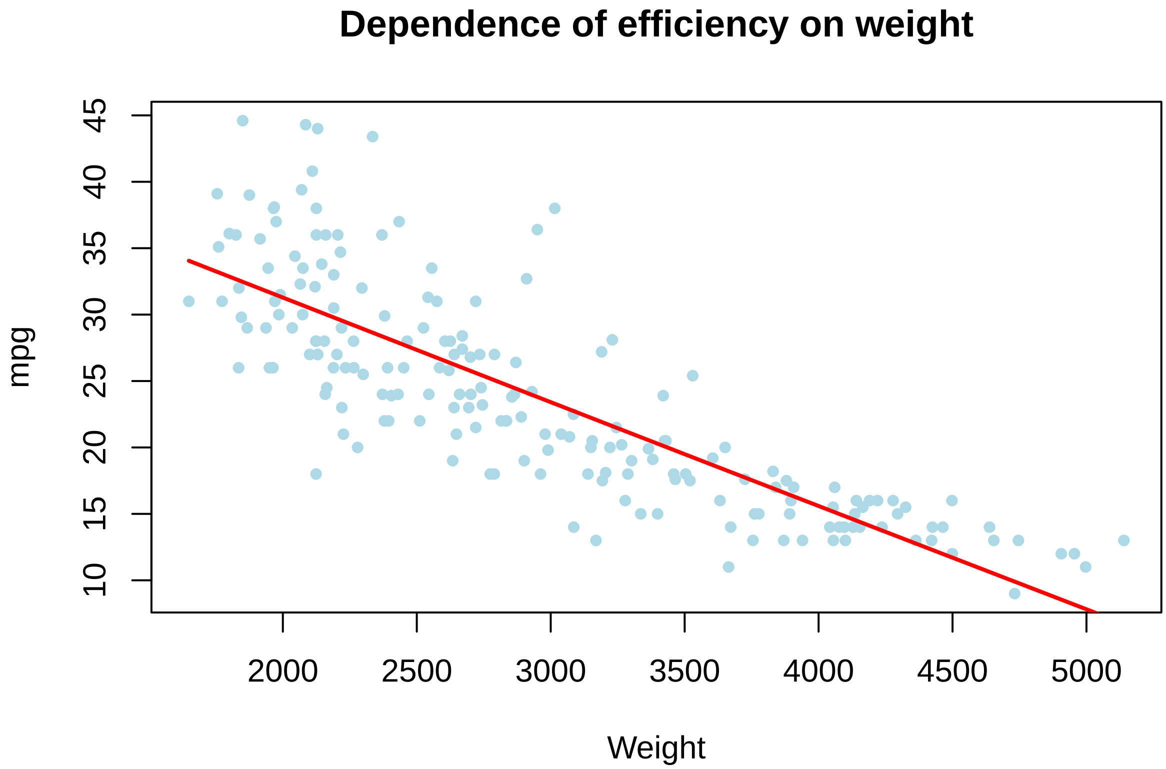
```
library(splines)
```

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
```

```r
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)
```




```r
#lm.fit=lm(training$mpg ~ training$weight)
#abline(lm.fit, col="blue")
```

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

```r
pred <- predict(lin.fit, val$weight)
#what does `predict` give us in this case?
pred
```

```
## $x
##   [1] 3693 3436 3433 3449 4341 4354 4312 3850 3563 3609 2587 2672 2375 4615 4376
##  [16] 4382 2228 3439 3329 4209 2408 3282 2065 1613 1834 1955 2278 2126 2254 2408
##  [31] 4274 4385 3672 4633 4502 4456 2330 4098 4294 2933 2288 2506 2164 3988 4952
##  [46] 4464 4735 4951 3821 3121 2945 3021 2904 2401 2310 2472 2265 4082 2582 2868
##  [61] 2807 3102 1950 2542 3781 3613 4699 4457 4257 2300 2003 2108 2246 2489 2000
##  [76] 3264 3432 3158 4668 4440 4657 3730 3785 2171 2914 2592 2223 2984 3211 2957
##  [91] 2945 2671 1795 2220 2572 2255 4215 3962 4215 3233
##  [ reached getOption("max.print") -- omitted 96 entries ]
##
## $y
##   [1] 17.985195 19.993068 20.016532 19.891400 12.937935 12.836830 13.163492
##   [8] 16.760566 19.000327 18.641006 26.656824 25.987738 28.326932 10.807582
##  [15] 12.665736 12.619076 29.485838 19.969605 20.830307 13.964779 28.066854
##  [22] 21.198314 30.771393 34.336930 32.593580 31.639080 29.091590 30.290255
```

```
## [29] 29.280822 28.066854 13.459079 12.595747 18.149107 10.667668 11.686016
## [36] 12.043659 28.681637 14.828661 13.303503 23.935668 29.012747 27.294727
## [43] 29.990551 15.685213  8.188477 11.981458  9.874881  8.196248 16.986668
## [50] 22.460092 23.841415 23.244691 24.163480 28.122019 28.839303 27.562567
## [57] 29.194088 14.953220 26.696193 24.446352 24.925837 22.609115 31.678522
## [64] 27.011181 17.298608 18.609767 10.154678 12.035884 13.591328 28.918140
## [71] 31.260446 30.432227 29.343902 27.428642 31.284110 21.339293 20.024353
## [78] 22.169960 10.395623 12.168065 10.481122 17.696461 17.267410 29.935344
## [85] 24.084918 26.617457 29.525266 23.535157 21.754530 23.747172 23.841415
## [92] 25.995608 32.901231 29.548923 26.774933 29.272937 13.918094 15.887747
## [99] 13.918094 21.582144
## [ reached getOption("max.print") -- omitted 96 entries ]
```

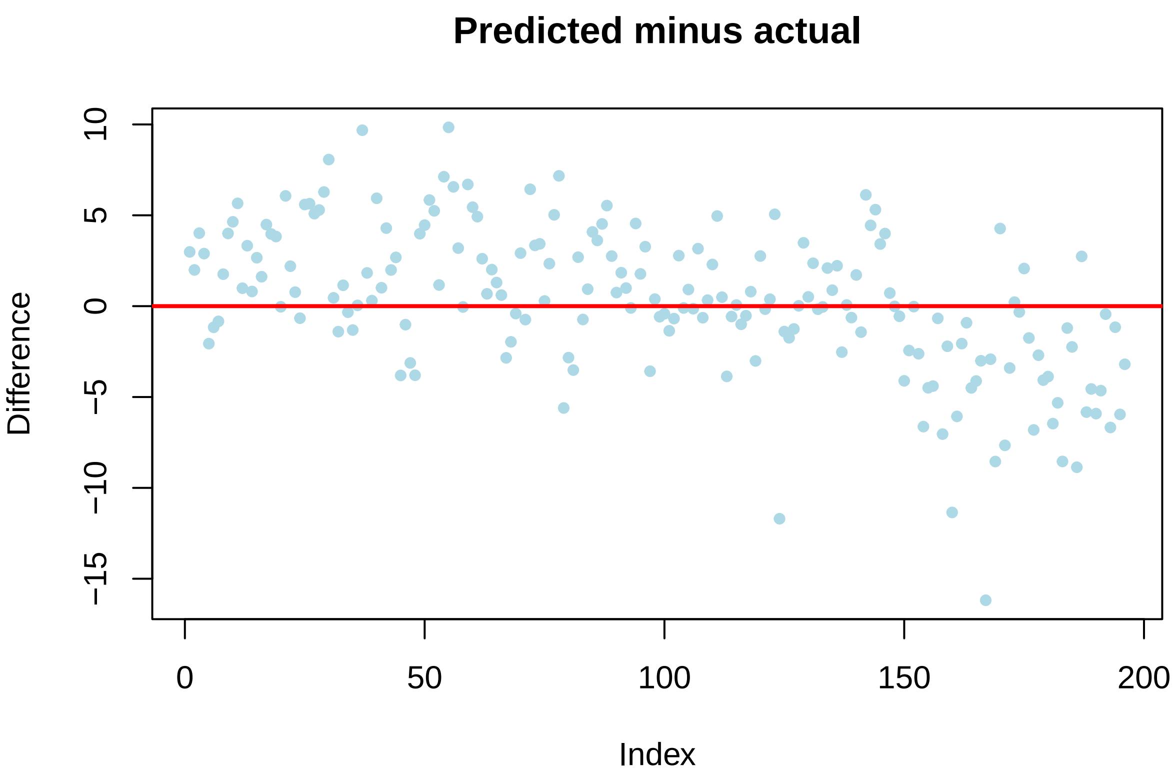
```
pred$x
```

```
##   [1] 3693 3436 3433 3449 4341 4354 4312 3850 3563 3609 2587 2672 2375 4615 4376
##  [16] 4382 2228 3439 3329 4209 2408 3282 2065 1613 1834 1955 2278 2126 2254 2408
##  [31] 4274 4385 3672 4633 4502 4456 2330 4098 4294 2933 2288 2506 2164 3988 4952
##  [46] 4464 4735 4951 3821 3121 2945 3021 2904 2401 2310 2472 2265 4082 2582 2868
##  [61] 2807 3102 1950 2542 3781 3613 4699 4457 4257 2300 2003 2108 2246 2489 2000
##  [76] 3264 3432 3158 4668 4440 4657 3730 3785 2171 2914 2592 2223 2984 3211 2957
##  [91] 2945 2671 1795 2220 2572 2255 4215 3962 4215 3233
## [ reached getOption("max.print") -- omitted 96 entries ]
```

```
pred$y
```

```
##   [1] 17.985195 19.993068 20.016532 19.891400 12.937935 12.836830 13.163492
##   [8] 16.760566 19.000327 18.641006 26.656824 25.987738 28.326932 10.807582
##  [15] 12.665736 12.619076 29.485838 19.969605 20.830307 13.964779 28.066854
##  [22] 21.198314 30.771393 34.336930 32.593580 31.639080 29.091590 30.290255
##  [29] 29.280822 28.066854 13.459079 12.595747 18.149107 10.667668 11.686016
##  [36] 12.043659 28.681637 14.828661 13.303503 23.935668 29.012747 27.294727
##  [43] 29.990551 15.685213  8.188477 11.981458  9.874881  8.196248 16.986668
##  [50] 22.460092 23.841415 23.244691 24.163480 28.122019 28.839303 27.562567
##  [57] 29.194088 14.953220 26.696193 24.446352 24.925837 22.609115 31.678522
##  [64] 27.011181 17.298608 18.609767 10.154678 12.035884 13.591328 28.918140
##  [71] 31.260446 30.432227 29.343902 27.428642 31.284110 21.339293 20.024353
##  [78] 22.169960 10.395623 12.168065 10.481122 17.696461 17.267410 29.935344
##  [85] 24.084918 26.617457 29.525266 23.535157 21.754530 23.747172 23.841415
##  [92] 25.995608 32.901231 29.548923 26.774933 29.272937 13.918094 15.887747
##  [99] 13.918094 21.582144
## [ reached getOption("max.print") -- omitted 96 entries ]
```

```
#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)
```



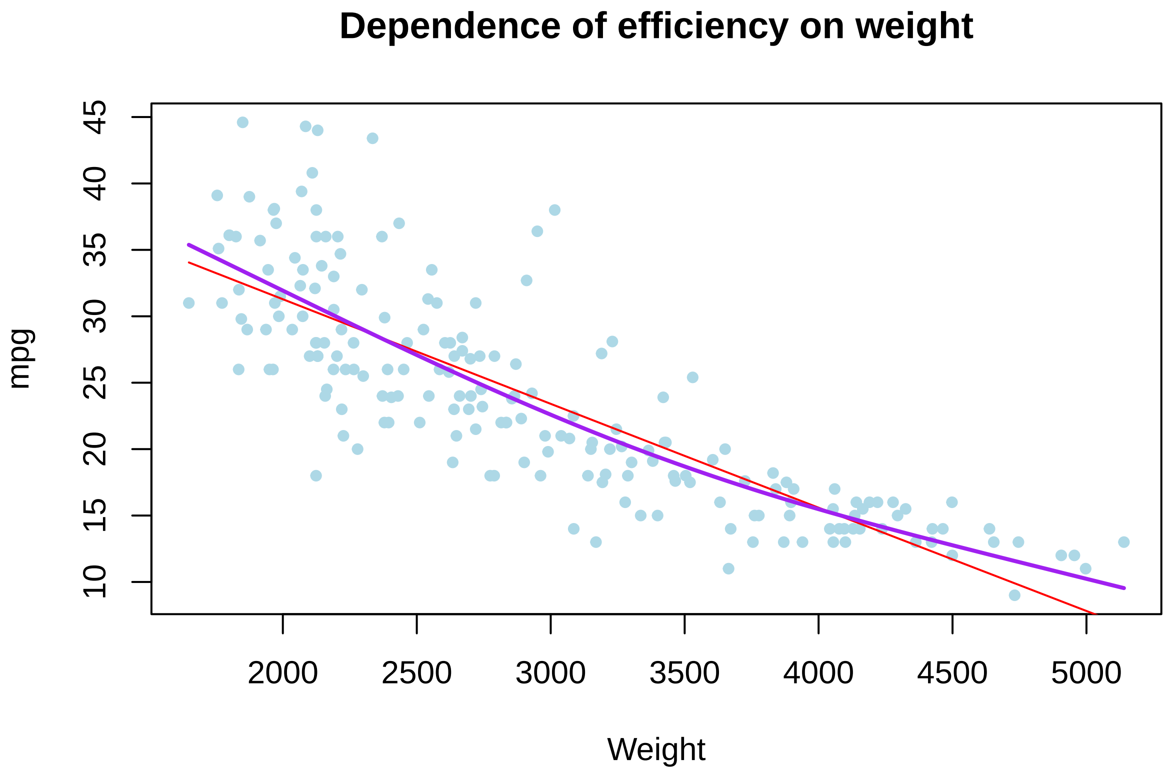

```
#the mean squared error of the predicted values
#in the validation set
mean((pred$y-val$mpg)^2)
```

```
## [1] 16.32752
```

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

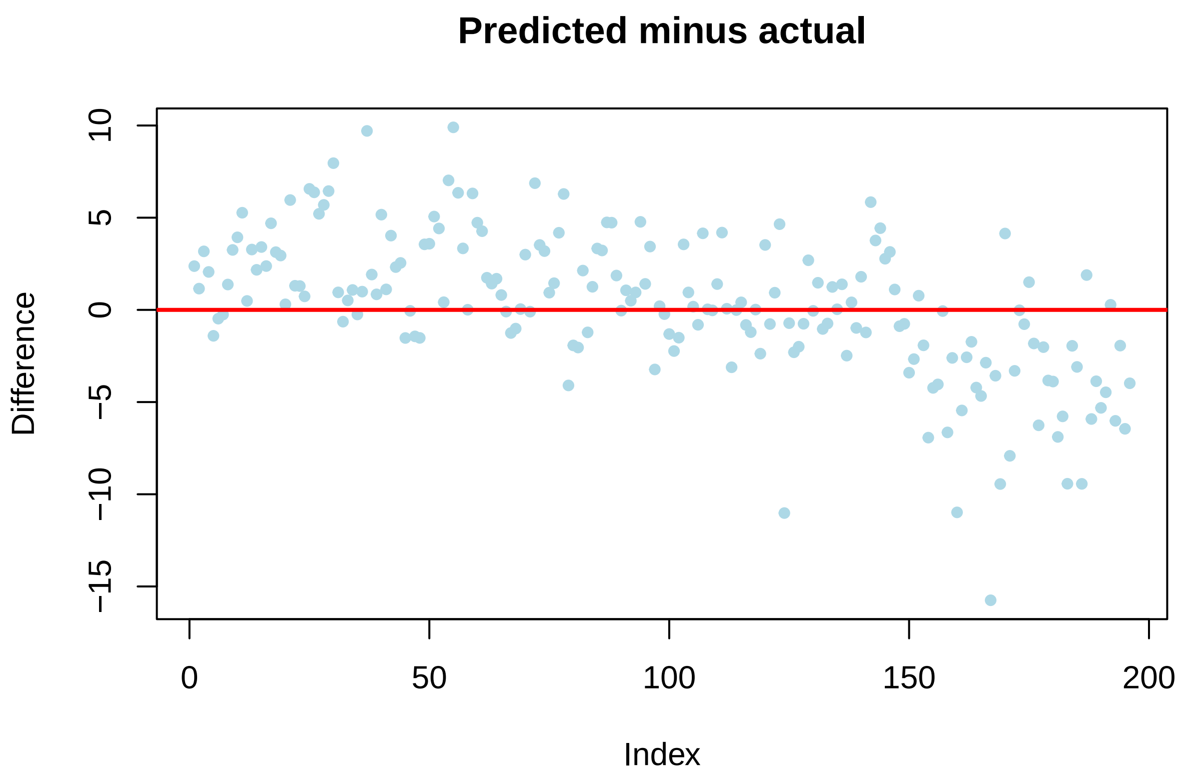
```
q.fit=smooth.spline(training$weight,training$mpg,df=3)

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)
```



How well did we do on the validation set?

```
#as before, we create the predicted values
pred <- predict(q.fit, val$weight)
#plot of how "off" the predictions are
plot(pred$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
#in the validation set
mean((pred$y-val$mpg)^2)
```

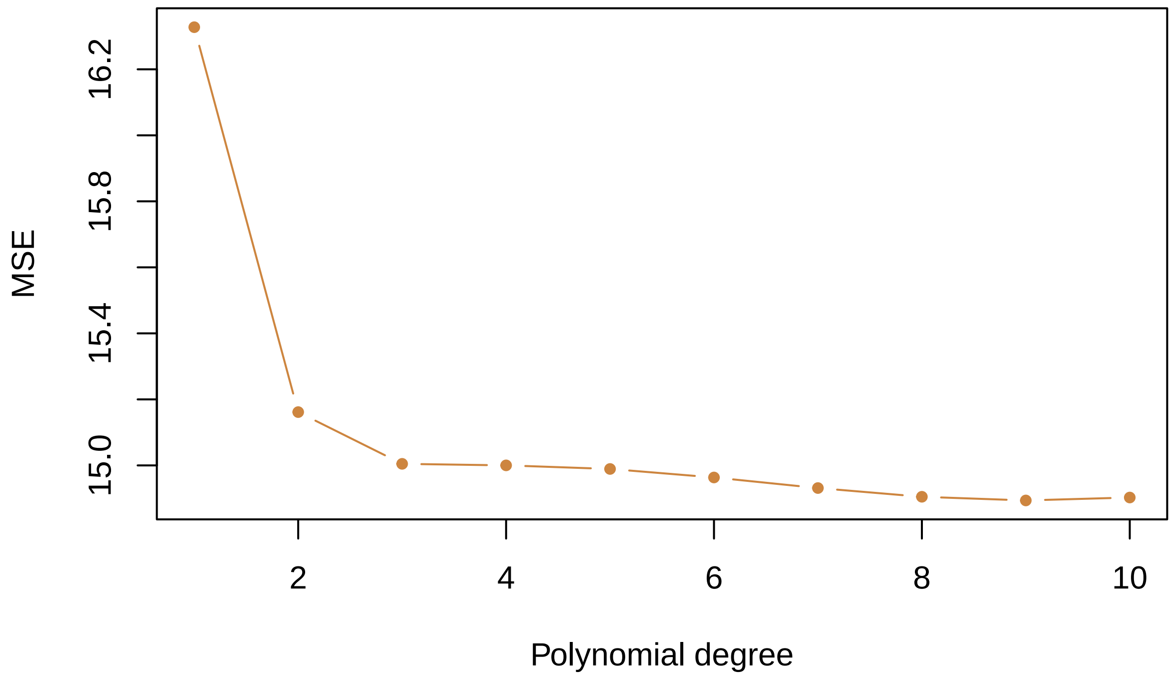
```
## [1] 15.1615
```

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] 16.32752 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?

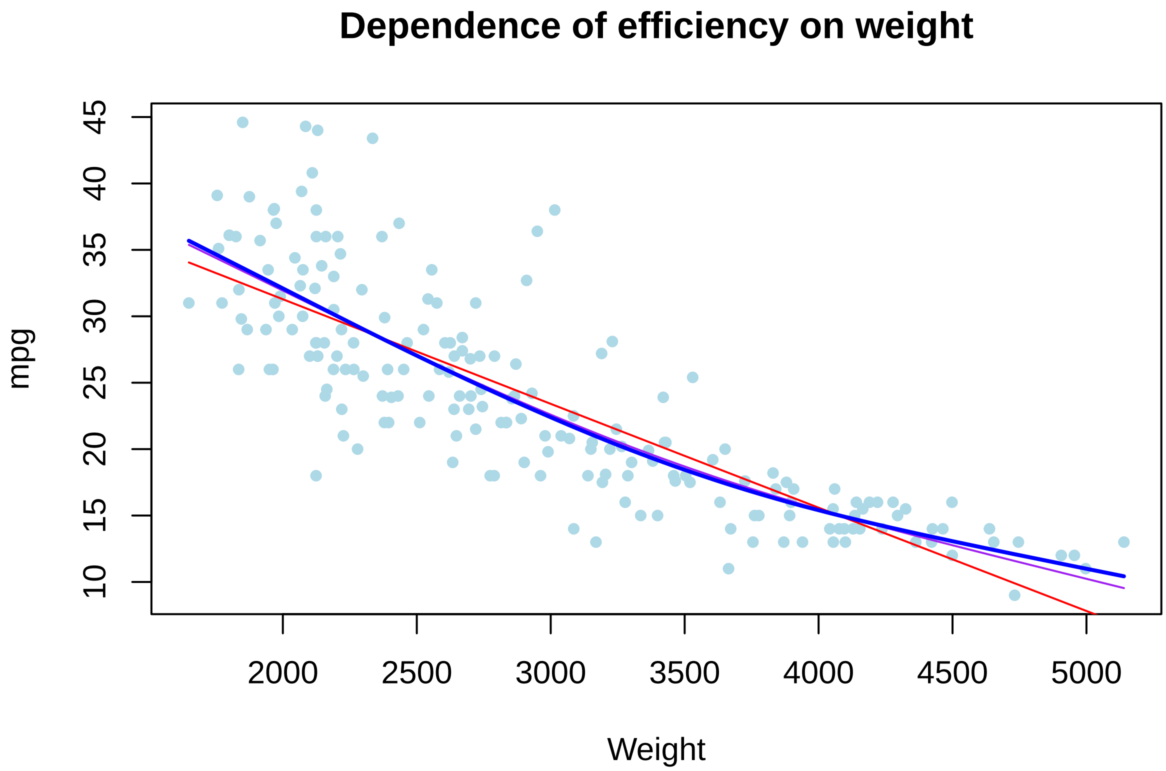
```
fit.cv=smooth.spline(training$weight,training$mpg,cv=T)
```

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

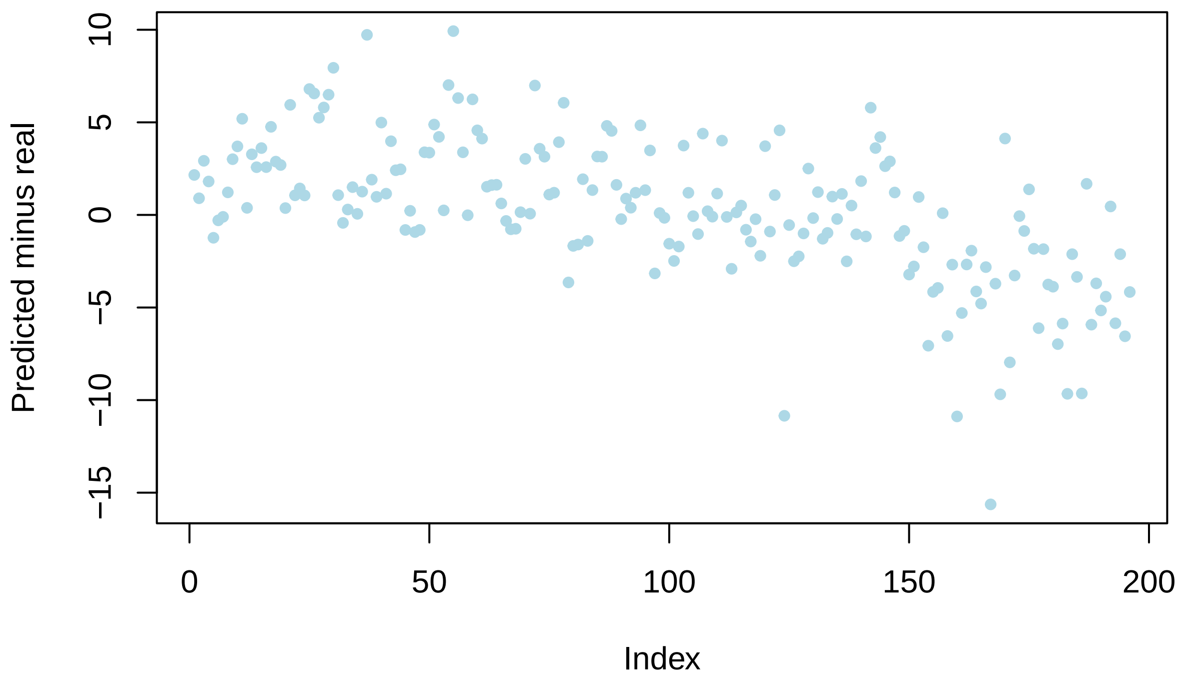
```
fit.cv$df
```

```
## [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)
```




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




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

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