Lab6

Dr. Purna Gamage

3/3/2021

```
library(ISLR)
library(gam)
```

```
## Loading required package: splines
```

```
## Loading required package: foreach
```

```
## Loaded gam 1.16.1
```

```
library(splines)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.3     v dplyr   1.0.4
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------------------ tidyverse_conflicts() --
## x purrr::accumulate() masks foreach::accumulate()
## x dplyr::filter()     masks stats::filter()
## x dplyr::lag()        masks stats::lag()
## x purrr::when()       masks foreach::when()
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

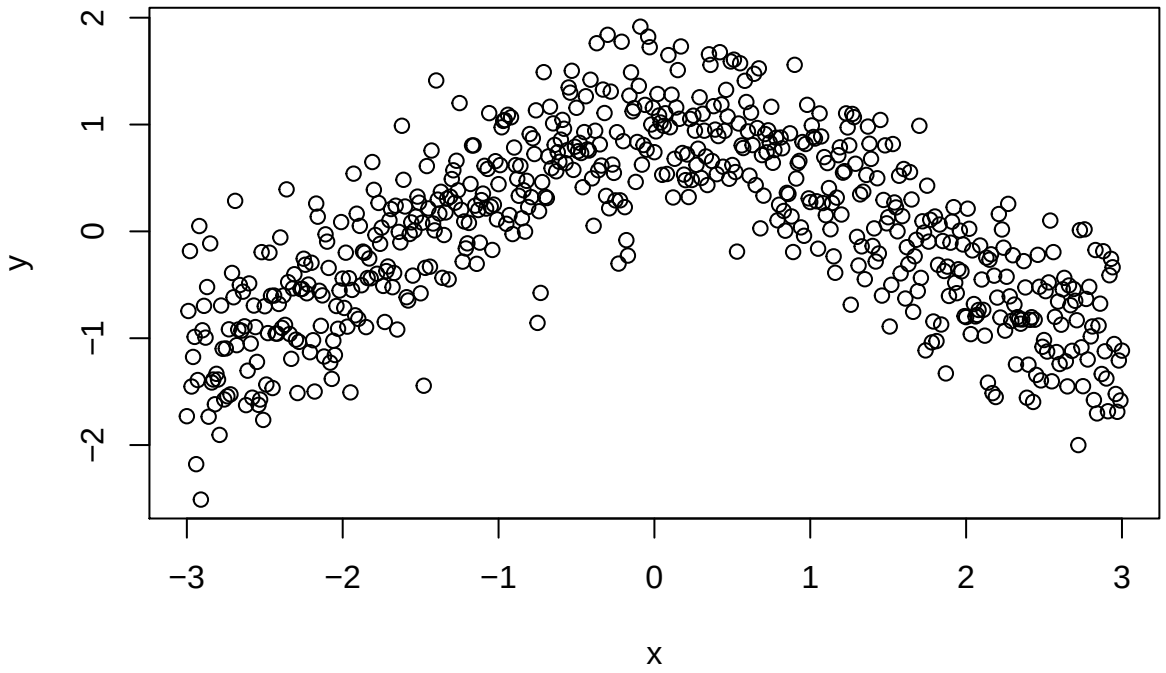
```
## The following object is masked from 'package:purrr':
##
##     lift
```

Polynomial Regression

Example 1

Make a sequence of x values to build model matrices. Also make a nonlinear response variable and store everything in a data frame.

```
x = seq(-3,3,by=.01) #random numbers
y = cos(x) + rnorm(length(x))/2 #non-linear function
mydf = data.frame(x = x, y = y)
plot(mydf)
```



Make the model matrix for simple linear regression.

```
X1 = model.matrix(y ~ x) #create a design matrix
head(X1)
```

```
##   (Intercept)     x
## 1           1 -3.00
## 2           1 -2.99
## 3           1 -2.98
## 4           1 -2.97
## 5           1 -2.96
## 6           1 -2.95
```

Make a model matrix for polynomial regression. The basis functions are the powers of the X variable.

```
X6 = model.matrix(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6))
head(X6)
```

```
##   (Intercept)     x I(x^2)    I(x^3)   I(x^4)    I(x^5)   I(x^6)
## 1           1 -3.00 9.0000 -27.00000 81.00000 -243.0000 729.0000
## 2           1 -2.99 8.9401 -26.73090 79.92539 -238.9769 714.5410
## 3           1 -2.98 8.8804 -26.46359 78.86150 -235.0073 700.3217
## 4           1 -2.97 8.8209 -26.19807 77.80828 -231.0906 686.3390
## 5           1 -2.96 8.7616 -25.93434 76.76563 -227.2263 672.5898
## 6           1 -2.95 8.7025 -25.67238 75.73351 -223.4138 659.0708
```

Also make a similar model matrix using the poly() function. This also makes basis functions which are polynomials of increasing degree. However, these basis functions have better numerical and statistical properties as we will see shortly.

```
X6p = model.matrix(y ~ poly(x,6))
head(X6)
```

```
##   (Intercept)     x I(x^2)    I(x^3)   I(x^4)    I(x^5)   I(x^6)
## 1           1 -3.00 9.0000 -27.00000 81.00000 -243.0000 729.0000
## 2           1 -2.99 8.9401 -26.73090 79.92539 -238.9769 714.5410
```

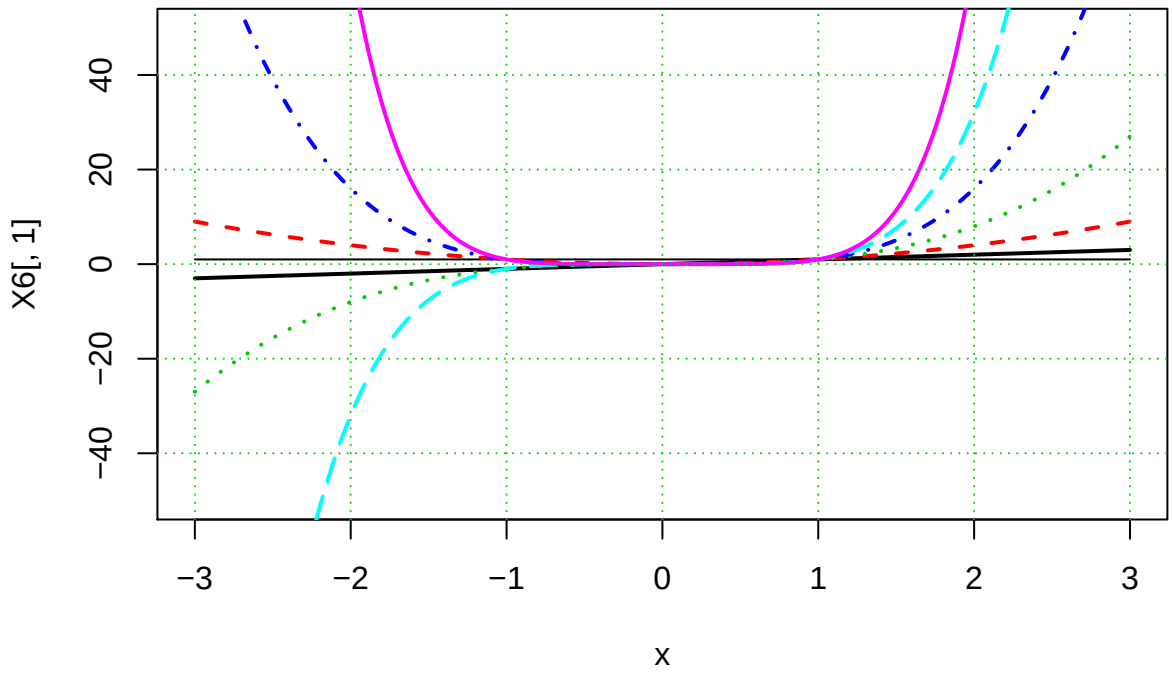
```
## 3           1 -2.98 8.8804 -26.46359 78.86150 -235.0073 700.3217
## 4           1 -2.97 8.8209 -26.19807 77.80828 -231.0906 686.3390
## 5           1 -2.96 8.7616 -25.93434 76.76563 -227.2263 672.5898
## 6           1 -2.95 8.7025 -25.67238 75.73351 -223.4138 659.0708
```

```
head(X6p) #values are different
```

```
##   (Intercept) poly(x, 6)1 poly(x, 6)2 poly(x, 6)3 poly(x, 6)4 poly(x, 6)5
## 1           1 -0.07053437  0.09075695 -0.10685037   0.1203532  -0.1319531
## 2           1 -0.07029925  0.08984938 -0.10471337   0.1163415  -0.1253555
## 3           1 -0.07006414  0.08894484 -0.10259420   0.1123900  -0.1189120
## 4           1 -0.06982902  0.08804333 -0.10049281   0.1084983  -0.1126207
## 5           1 -0.06959391  0.08714486 -0.09840913   0.1046659  -0.1064795
## 6           1 -0.06935879  0.08624941 -0.09634312   0.1008924  -0.1004863
##   poly(x, 6)6
## 1  0.14202315
## 2  0.13208153
## 3  0.12247185
## 4  0.11318745
## 5  0.10422175
## 6  0.09556826
```

Make a plot of the basis functions x, x^2, . . . , x^6. Since these values are fairly large near the ends of the interval, we need a large ylim.

```
plot(x,X6[,1], type = 'l', ylim = c(-50,50))
matlines(x,X6[,2:7],lwd = 2)
grid(col = 3) #plotting the polynomials with degree 1,..,6
```



Compute also the correlation coefficients. As you can see, even powers are correlated with each other and odd powers are also correlated with each other.

```
round(cor(X6[,2:7]),2) #what can you say about the correlation?high?low? misfitting? if the features ar
```

```
##           x I(x^2) I(x^3) I(x^4) I(x^5) I(x^6)
```

```
## x      1.00   0.00   0.92   0.00   0.82   0.00
## I(x^2) 0.00   1.00   0.00   0.96   0.00   0.90
## I(x^3) 0.92   0.00   1.00   0.00   0.97   0.00
## I(x^4) 0.00   0.96   0.00   1.00   0.00   0.98
## I(x^5) 0.82   0.00   0.97   0.00   1.00   0.00
## I(x^6) 0.00   0.90   0.00   0.98   0.00   1.00
```

By contrast, the basis function generated by poly() are not correlated at all. They are "orthogonal polynomials".

```
round(cor(X6p[,2:7]),2) #what do you notice? poly() invokes othogonal polynomials.
```

```
##             poly(x, 6)1 poly(x, 6)2 poly(x, 6)3 poly(x, 6)4 poly(x, 6)5
## poly(x, 6)1           1           0           0           0           0
## poly(x, 6)2           0           1           0           0           0
## poly(x, 6)3           0           0           1           0           0
## poly(x, 6)4           0           0           0           1           0
## poly(x, 6)5           0           0           0           0           1
## poly(x, 6)6           0           0           0           0           0
##             poly(x, 6)6
## poly(x, 6)1           0
## poly(x, 6)2           0
## poly(x, 6)3           0
## poly(x, 6)4           0
## poly(x, 6)5           0
## poly(x, 6)6           1
```

```
#donot want linear combinations of features that create other features, we want it to be othonormal bas
```

The basis functions x, x^2, ... also have less desirable numerical properties. In the matrix formulation of linear regression, the following matrix and its inverse appear:

```
A = t(X6)%*%X6 #X^TX (XtransposeX) %*% - matrix multiplication
B = solve(A) #matrix inverse
```

Then A*B should be the identity matrix. Let's compute the product and check how this differs from the identity matrix:

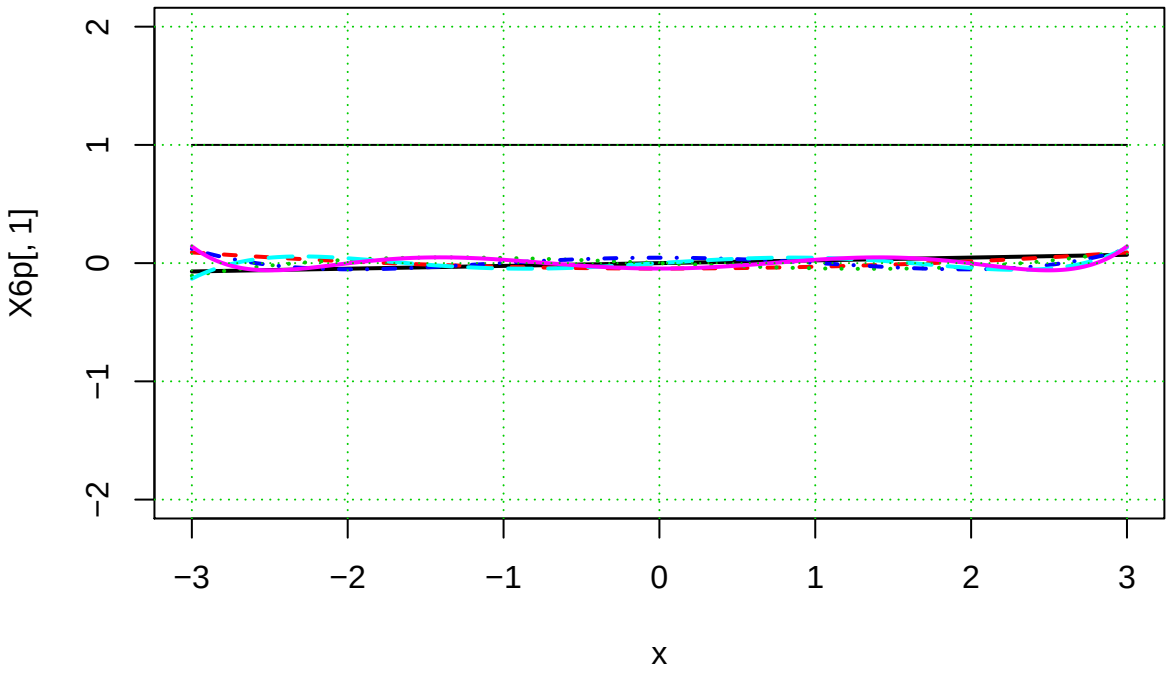
```
max(abs(A%*%B -diag(rep(1,7)))) #this should be 0 if it is equal to the identity matrix
```

```
## [1] 1.136868e-13
```

This should really be much smaller. However, for this particular problem, it does not cause trouble.

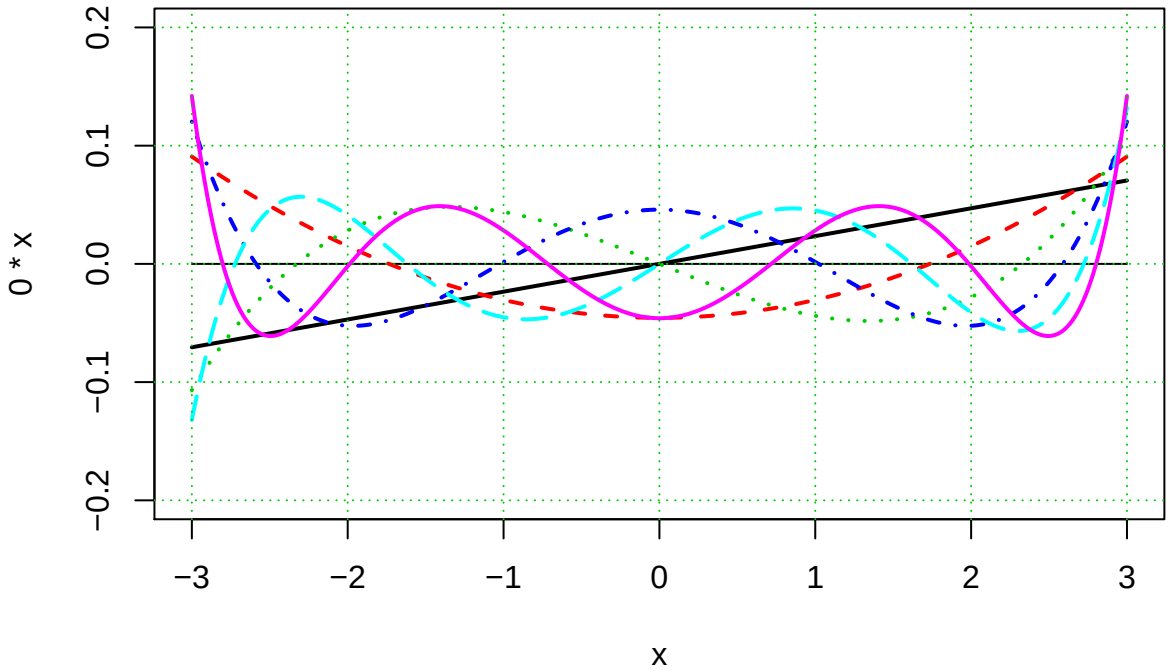
Now plot the orthogonal basis functions generated by poly().

```
plot(x,X6p[,1], type = 'l', ylim = c(-2,2))
matlines(x,X6p[,2:7],lwd = 2)
grid(col = 3)
```



All basis functions are much more similar in magnitude. Zoom in:

```
plot(x,0*x,ylim=c(-.2,.2), type = 'l')
matlines(x,X6p[,2:7],lwd = 2)
grid(col = 3) #6 polynomials
```




```
#othornormal basis set ==> much numerical stability for our model
```

These basis functions are all uncorrelated. That is a consequence (and in fact essentially equivalent) to being orthogonal.

```
round(cor(X6p[,2:7]),2)
```

```
##             poly(x, 6)1 poly(x, 6)2 poly(x, 6)3 poly(x, 6)4 poly(x, 6)5
## poly(x, 6)1           1           0           0           0           0
## poly(x, 6)2           0           1           0           0           0
## poly(x, 6)3           0           0           1           0           0
## poly(x, 6)4           0           0           0           1           0
## poly(x, 6)5           0           0           0           0           1
## poly(x, 6)6           0           0           0           0           0
##             poly(x, 6)6
## poly(x, 6)1           0
## poly(x, 6)2           0
## poly(x, 6)3           0
## poly(x, 6)4           0
## poly(x, 6)5           0
## poly(x, 6)6           1
```

Their numerical properties are also better.

```
A = t(X6p)%*%X6p
B = solve(A)
max(abs(A%*%B -diag(rep(1,7)))) #should be close to 0
```

```
## [1] 5.703642e-31
```

```
fit=lm(y ~ poly(x,6))
summary(fit) #what do you notice? Which predictors are worth keeping in the model?remeber what the data
```

```
##
## Call:
## lm(formula = y ~ poly(x, 6))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.50589 -0.32334  0.00562  0.31033  1.32103
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -8.795e-04  1.952e-02  -0.045   0.9641
## poly(x, 6)1  8.817e-01  4.785e-01   1.843   0.0659 .
## poly(x, 6)2 -1.678e+01  4.785e-01 -35.073  < 2e-16 ***
## poly(x, 6)3 -1.604e-01  4.785e-01  -0.335   0.7375
## poly(x, 6)4  3.821e+00  4.785e-01   7.987 7.22e-15 ***
## poly(x, 6)5  5.144e-01  4.785e-01   1.075   0.2827
## poly(x, 6)6 -3.667e-01  4.785e-01  -0.766   0.4438
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4785 on 594 degrees of freedom
## Multiple R-squared:  0.6862, Adjusted R-squared:  0.6831
## F-statistic: 216.5 on 6 and 594 DF,  p-value: < 2.2e-16
```

Example 1

We'll use the Boston data set [in MASS package], for predicting the median house value (mdev), in Boston Suburbs, based on the predictor variable lstat (percentage of lower status of the population).

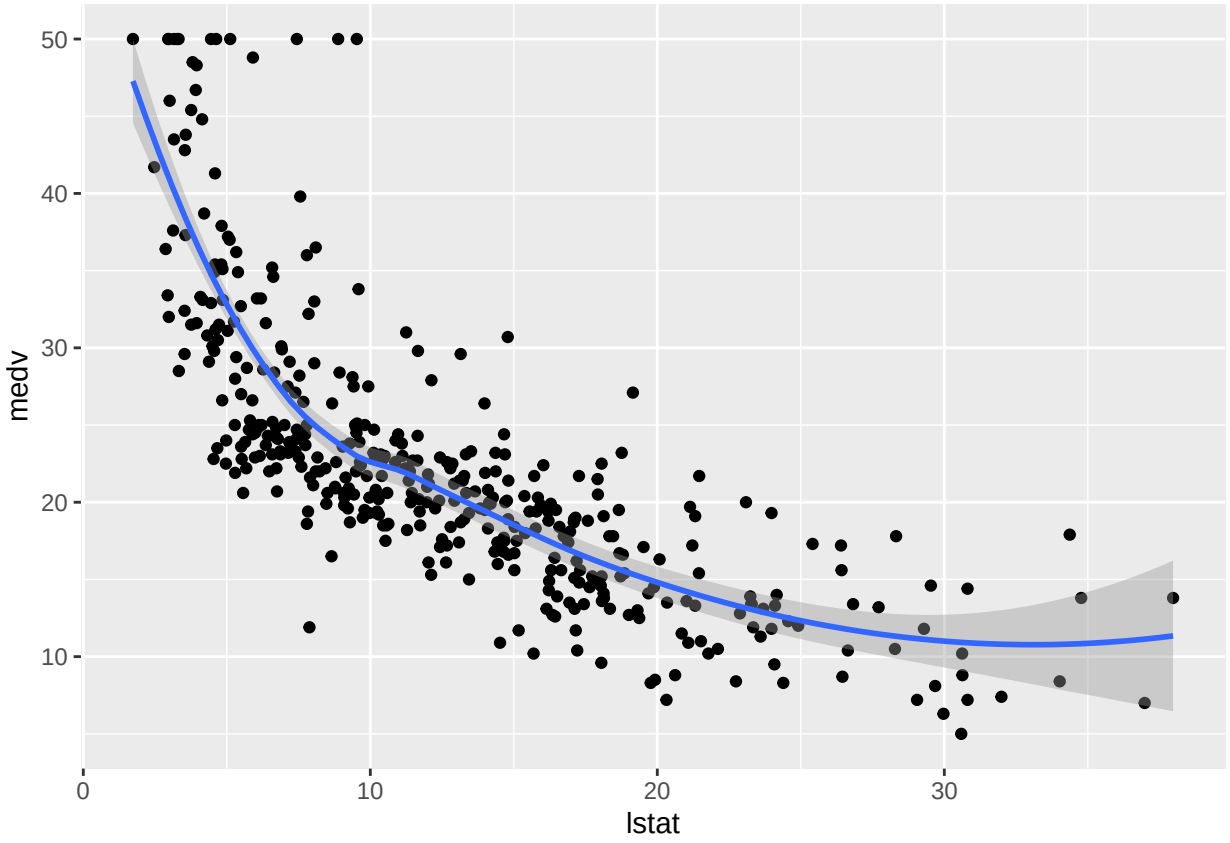
We'll randomly split the data into training set (80% for building a predictive model) and test set (20% for evaluating the model). Make sure to set seed for reproducibility.

```
# Load the data
data("Boston", package = "MASS")
# Split the data into training and test set
set.seed(123)
training.samples <- Boston$medv %>%
  createDataPartition(p = 0.8, list = FALSE)
train.data  <- Boston[training.samples, ]
test.data <- Boston[-training.samples, ]
```

First, visualize the scatter plot of the medv vs lstat variables as follow:

```
ggplot(train.data, aes(lstat, medv) ) +
  geom_point() +
  stat_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Let's compare the different models in order to choose the best one for our data.

The standard linear regression model equation can be written as medv = b0 + b1*lstat.

(a). Compute linear regression model:

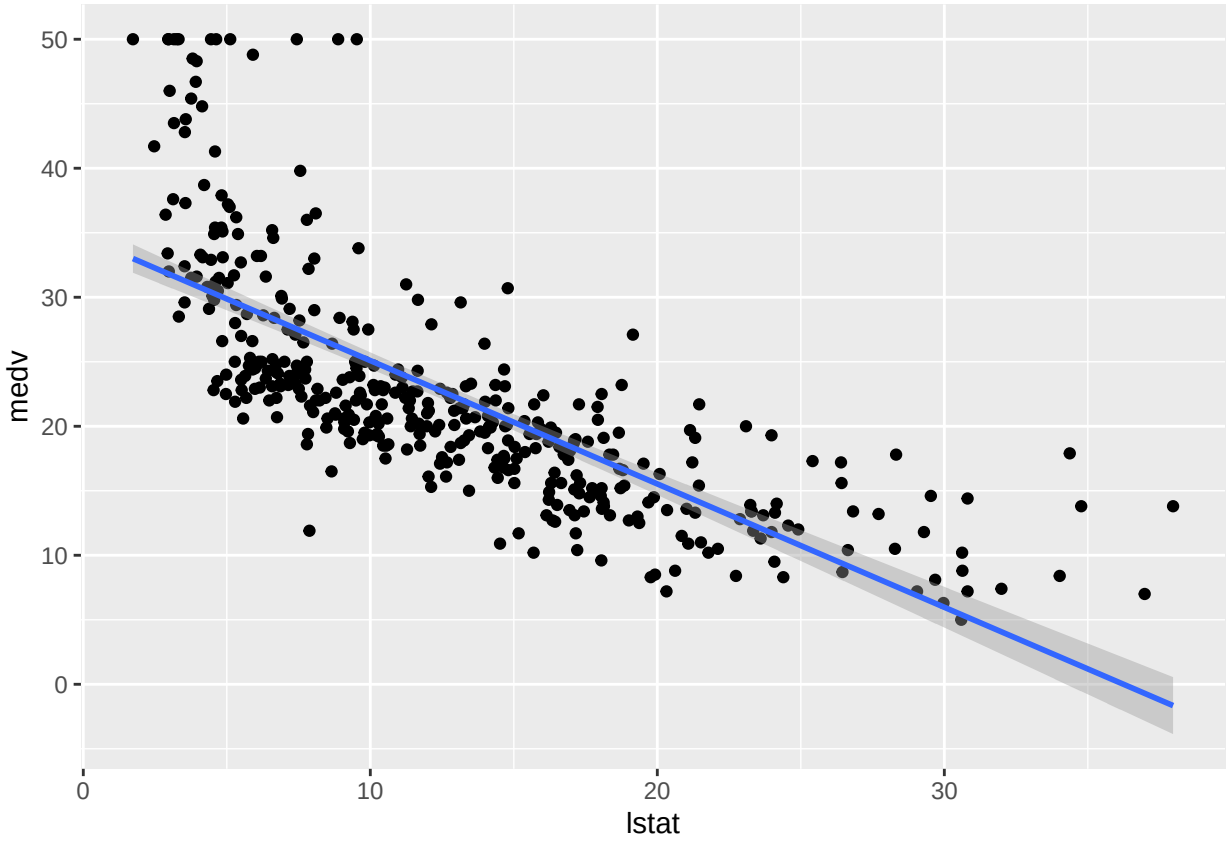
```
# Build the model
model <- lm(medv ~ lstat, data = train.data)
# Make predictions
predictions <- model %>% predict(test.data)
# Model performance
data.frame(
  RMSE = RMSE(predictions, test.data$medv),
```

```
  R2 = R2(predictions, test.data$medv)
)
```

```
##       RMSE       R2
## 1 6.503817 0.513163
```

Visualize the data:

```
ggplot(train.data, aes(lstat, medv) ) +
  geom_point() +
  stat_smooth(method = lm, formula = y ~ x)
```



(b). Polynomial regression

The polynomial regression adds polynomial or quadratic terms to the regression equation as follow:

$medv = b0 + b1 * lstat + b2 * lstat^2$

In R, to create a predictor x^2 you should use the function I(), as follow: I(x^2). This raise x to the power 2.

```
lm(medv ~ poly(lstat, 2, raw = TRUE), data = train.data)
```

```
##
## Call:
## lm(formula = medv ~ poly(lstat, 2, raw = TRUE), data = train.data)
##
## Coefficients:
##                  (Intercept)  poly(lstat, 2, raw = TRUE)1
##                      42.5736                      -2.2673
## poly(lstat, 2, raw = TRUE)2
##                       0.0412
```

The output contains two coefficients associated with lstat : one for the linear term (lstat^1) and one for the quadratic term (lstat^2).

The following example computes a sixfth-order polynomial fit:

```
lm(medv ~ poly(lstat, 6, raw = TRUE), data = train.data) %>%
  summary()
```

```
##
## Call:
## lm(formula = medv ~ poly(lstat, 6, raw = TRUE), data = train.data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.1962  -3.1527  -0.7655   2.0404  26.7661
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   7.788e+01  6.844e+00  11.379  < 2e-16 ***
## poly(lstat, 6, raw = TRUE)1 -1.767e+01  3.569e+00  -4.952 1.08e-06 ***
## poly(lstat, 6, raw = TRUE)2  2.417e+00  6.779e-01   3.566 0.000407 ***
## poly(lstat, 6, raw = TRUE)3 -1.761e-01  6.105e-02  -2.885 0.004121 **
## poly(lstat, 6, raw = TRUE)4  6.845e-03  2.799e-03   2.446 0.014883 *
## poly(lstat, 6, raw = TRUE)5 -1.343e-04  6.290e-05  -2.136 0.033323 *
## poly(lstat, 6, raw = TRUE)6  1.047e-06  5.481e-07   1.910 0.056910 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.188 on 400 degrees of freedom
## Multiple R-squared:  0.6845, Adjusted R-squared:  0.6798
## F-statistic: 144.6 on 6 and 400 DF,  p-value: < 2.2e-16
```

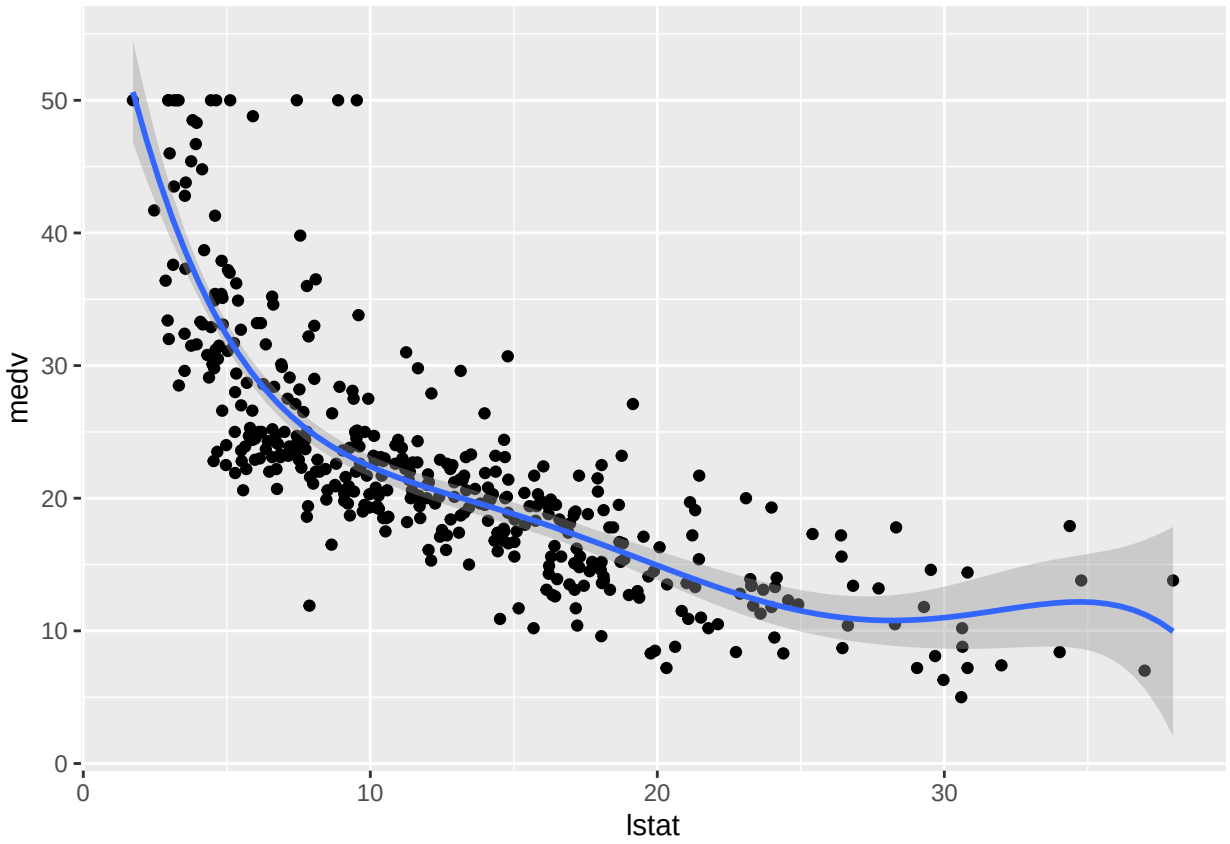
From the output above, it can be seen that polynomial terms beyond the fith order are not significant. So, just create a fith polynomial regression model as follow:

```
# Build the model
model <- lm(medv ~ poly(lstat, 5, raw = TRUE), data = train.data)
# Make predictions
predictions <- model %>% predict(test.data)
# Model performance
data.frame(
  RMSE = RMSE(predictions, test.data$medv),
  R2 = R2(predictions, test.data$medv)
)
```

```
##       RMSE        R2
## 1 5.270374 0.6829474
```

Visualize the fith polynomial regression line as follow:

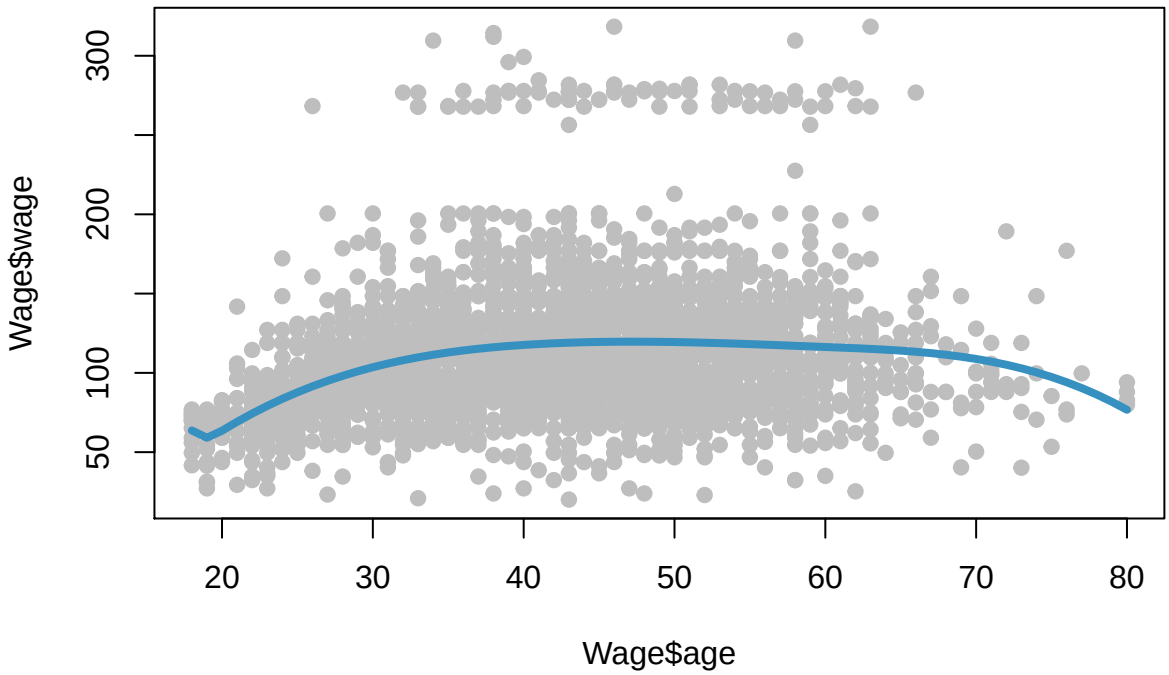
```
ggplot(train.data, aes(lstat, medv) ) +
  geom_point() +
  stat_smooth(method = lm, formula = y ~ poly(x, 5, raw = TRUE))
```



Cubic Splines and Smoothing Splines

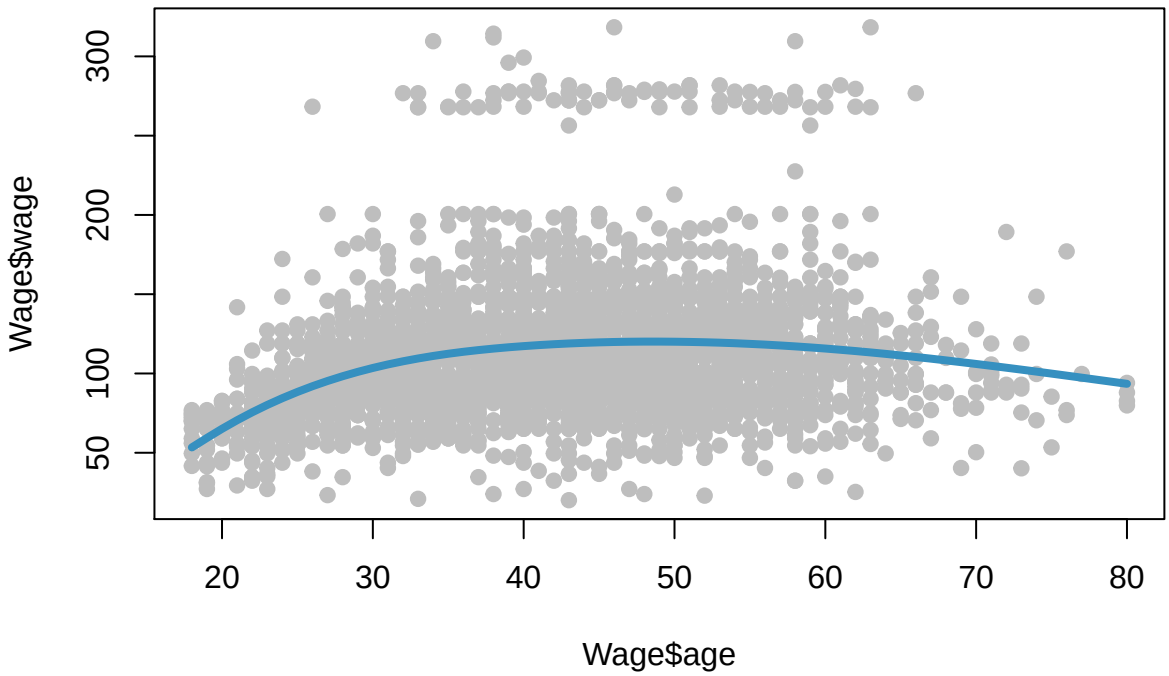
Cubic spline with fixed knots.

```
#library(splineDesign)
attach(Wage)
agelims=range(Wage$age)
age.grid=seq(from=agelims[1],to=agelims[2])
spline_fit <- lm(wage ~ bs(age, knots=c(20,40,60)), data=Wage)

plot(Wage$age, Wage$wage, pch=19, col='grey')
pred = predict(spline_fit,list(age=age.grid), se=T)
lines(age.grid,pred$fit, col="#3690C0",lwd=4)
```



A natural spline has better behavior at the boundary points.

```
agelims=range(Wage$age)
age.grid=seq(from=agelims[1],to=agelims[2])

spline_fit <- lm(wage ~ ns(age, knots=c(20,40,60)), data=Wage)
plot(Wage$age, Wage$wage, pch=19, col='grey')
pred = predict(spline_fit,list(age=age.grid), se=T)
lines(age.grid,pred$fit, col="#3690C0",lwd=4)
```



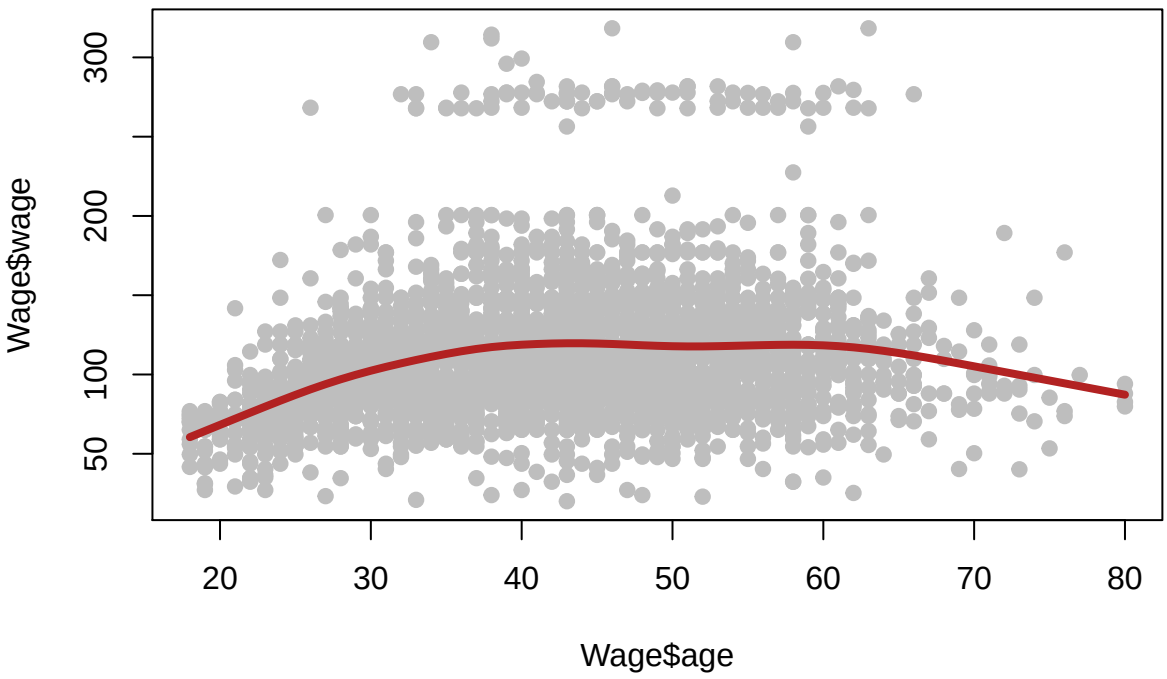
(d) Smoothing Spline

A smoothing spline is a cubic spline with a knot at every observed x but also a penalization to encourage smoothness.

```
plot(Wage$age, Wage$wage, pch=19, col='grey')
ss_fit <- smooth.spline(Wage$age,Wage$wage,cv=TRUE)
```

```
## Warning in smooth.spline(Wage$age, Wage$wage, cv = TRUE): cross-validation with
## non-unique 'x' values seems doubtful
```

```
lines(ss_fit,col="firebrick", lwd=4)
```



Example 2

For our previous example, we'll place the knots at the lower quartile, the median quartile, and the upper quartile:

```
knots <- quantile(train.data$lstat, p = c(0.25, 0.5, 0.75))
```

We'll create a model using a cubic spline (degree = 3):

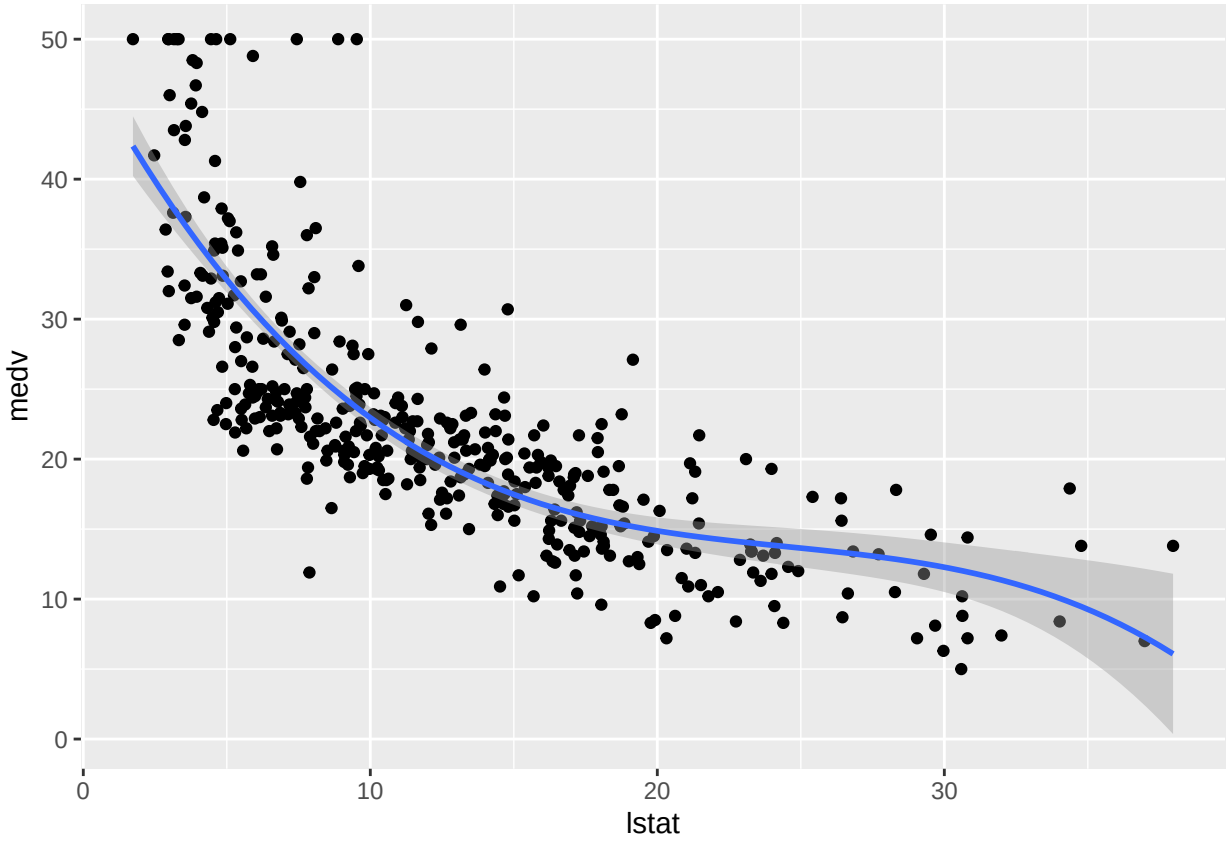
```
# Build the model
knots <- quantile(train.data$lstat, p = c(0.25, 0.5, 0.75))
model <- lm (medv ~ bs(lstat, knots = knots), data = train.data)
# Make predictions
predictions <- model %>% predict(test.data)
# Model performance
data.frame(
  RMSE = RMSE(predictions, test.data$medv),
  R2 = R2(predictions, test.data$medv)
)
```

```
##       RMSE        R2
## 1 5.317372 0.6786367
```

Note that, the coefficients for a spline term are not interpretable.

Visualize the cubic spline as follow:

```
ggplot(train.data, aes(lstat, medv) ) +
  geom_point() +
  stat_smooth(method = lm, formula = y ~ splines::bs(x, df = 3))
```



GAM

```
# Build the model
model <- gam(medv ~ s(lstat), data = train.data)
```

```
## Warning in model.matrix.default(mt, mf, contrasts): non-list contrasts argument
## ignored
```

```
# Make predictions
predictions <- model %>% predict(test.data)
# Model performance
data.frame(
  RMSE = RMSE(predictions, test.data$medv),
  R2 = R2(predictions, test.data$medv)
)
```

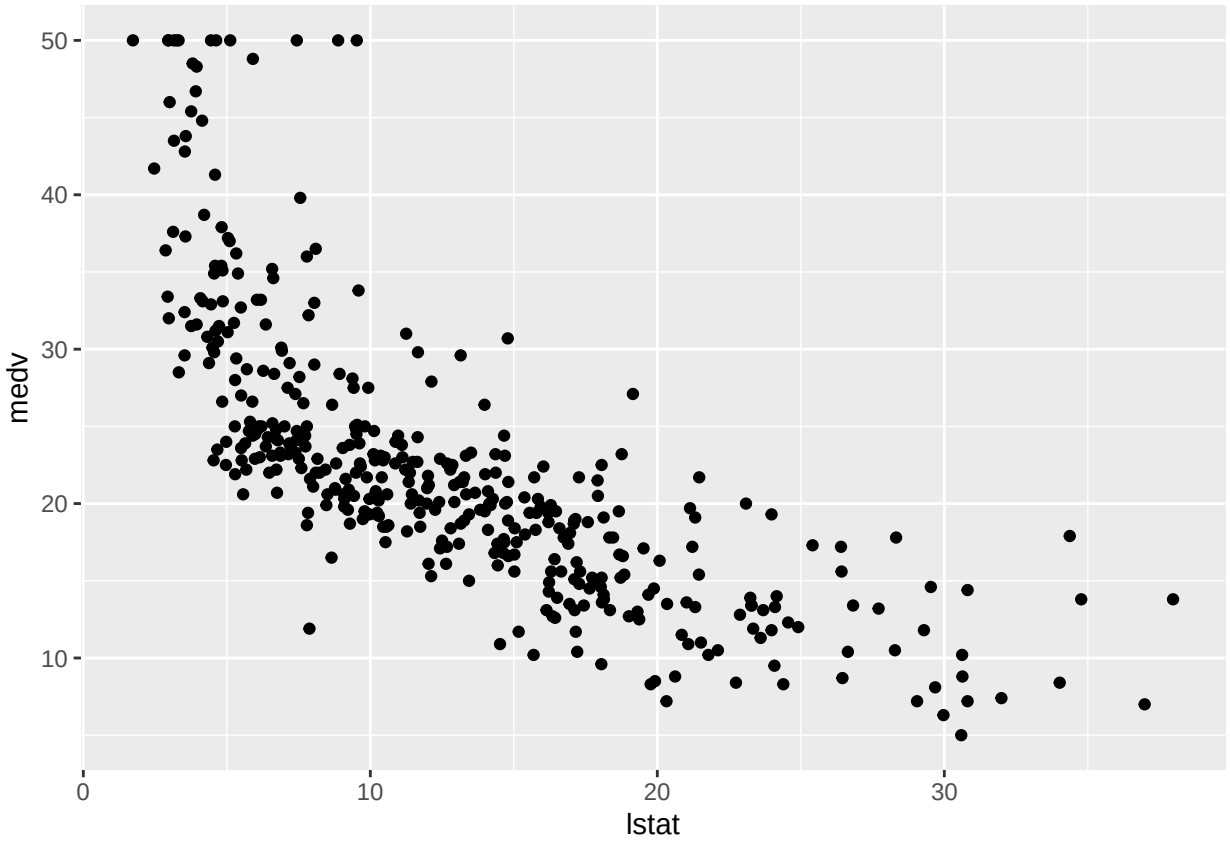
```
##      RMSE        R2
## 1 5.42684 0.6610243
```

```
ggplot(train.data, aes(lstat, medv) ) +
  geom_point() +
  stat_smooth(method = gam, formula = y ~ s(x))
```

```
## Warning in model.matrix.default(mt, mf, contrasts): non-list contrasts argument
## ignored
```

```
## Warning in newdata.predict.Gam(object, newdata, type, dispersion, se.fit, : No
## standard errors (currently) for gam predictions with newdata
```

```
## Warning: Computation failed in `stat_smooth()`:
## $ operator is invalid for atomic vectors
```



From analyzing the RMSE and the R2 metrics of the different models, it can be seen that the polynomial regression, the spline regression and the generalized additive models outperform the linear regression model and the log transformation approaches.

From the book

```
gam1=lm(wage~ns(year,4)+ns(age,5)+education,data=Wage)

gam.m3=gam(wage~s(year,4)+s(age,5)+education,data=Wage)
```

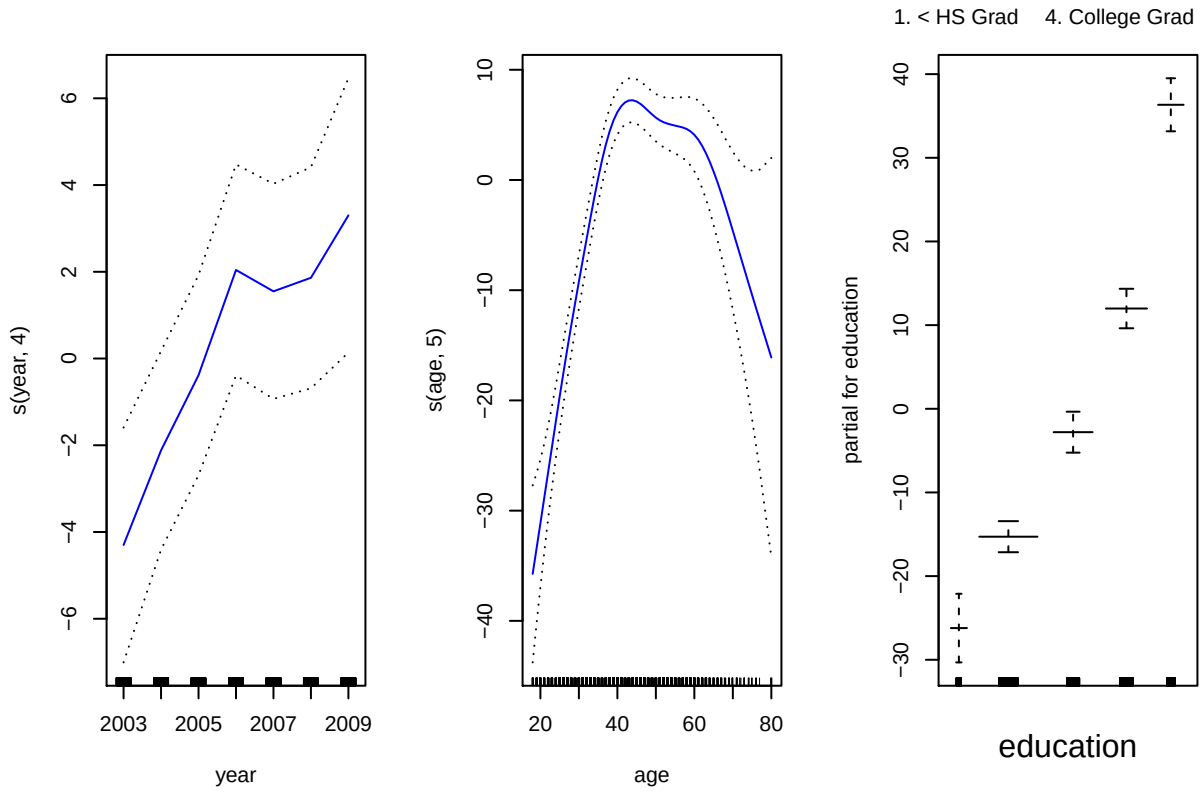
```
## Warning in model.matrix.default(mt, mf, contrasts): non-list contrasts argument
## ignored
```

```
summary(gam.m3)
```

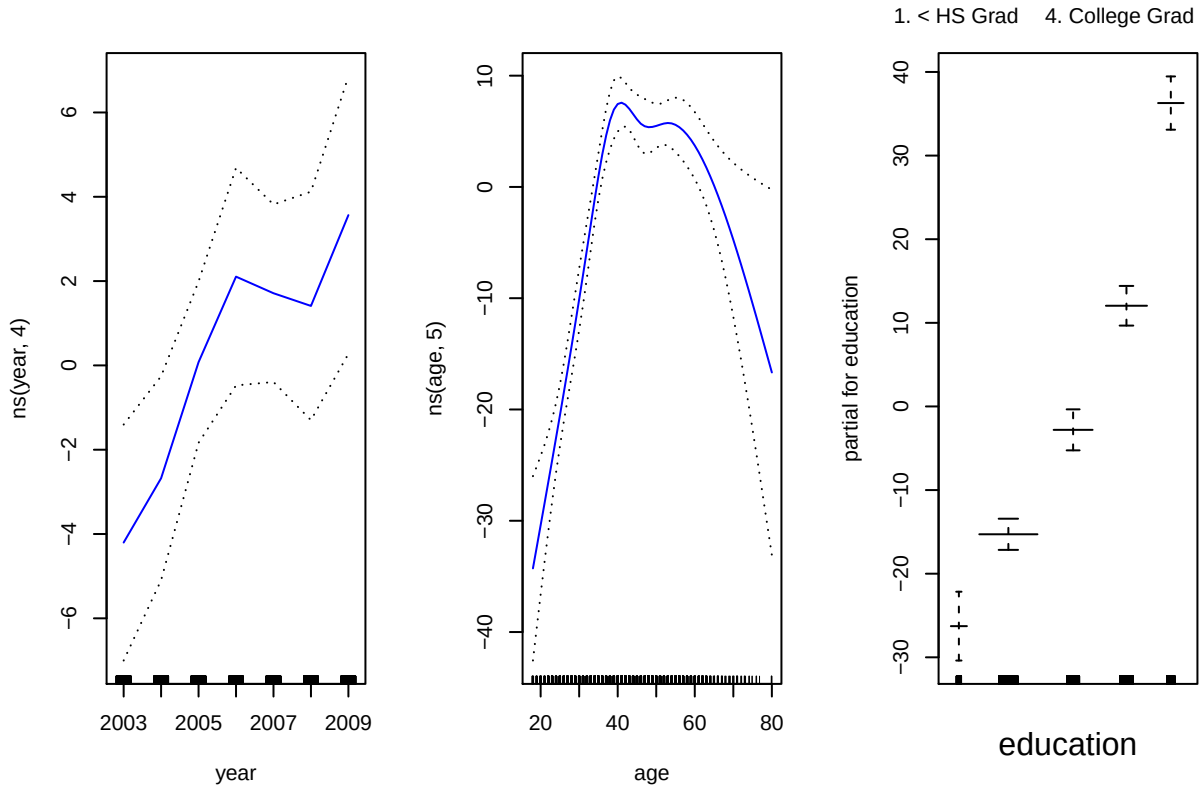
```
##
## Call: gam(formula = wage ~ s(year, 4) + s(age, 5) + education, data = Wage)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -119.43  -19.70   -3.33   14.17  213.48
```

```
##
## (Dispersion Parameter for gaussian family taken to be 1235.69)
##
##     Null Deviance: 5222086 on 2999 degrees of freedom
## Residual Deviance: 3689770 on 2986 degrees of freedom
## AIC: 29887.75
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##              Df  Sum Sq Mean Sq F value    Pr(>F)
## s(year, 4)    1   27162   27162  21.981 2.877e-06 ***
## s(age, 5)     1  195338  195338 158.081 < 2.2e-16 ***
## education     4 1069726  267432 216.423 < 2.2e-16 ***
## Residuals  2986 3689770    1236
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##             Npar Df Npar F  Pr(F)
## (Intercept)
## s(year, 4)        3  1.086 0.3537
## s(age, 5)         4 32.380 <2e-16 ***
## education
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
par(mfrow=c(1,3))

plot(gam.m3, se=TRUE,col="blue")
```




```
plot.Gam(gam1, se=TRUE,col="blue")
```



```r
gam.m1=gam(wage~s(age,5)+education,data=Wage)
```

```
## Warning in model.matrix.default(mt, mf, contrasts): non-list contrasts argument
## ignored
```

```r
gam.m2=gam(wage~year+s(age,5)+education,data=Wage)
```

```
## Warning in model.matrix.default(mt, mf, contrasts): non-list contrasts argument
## ignored
```

```r
gam.m3=gam(wage~s(year,4)+s(age,5)+education,data=Wage)
```

```
## Warning in model.matrix.default(mt, mf, contrasts): non-list contrasts argument
## ignored
```

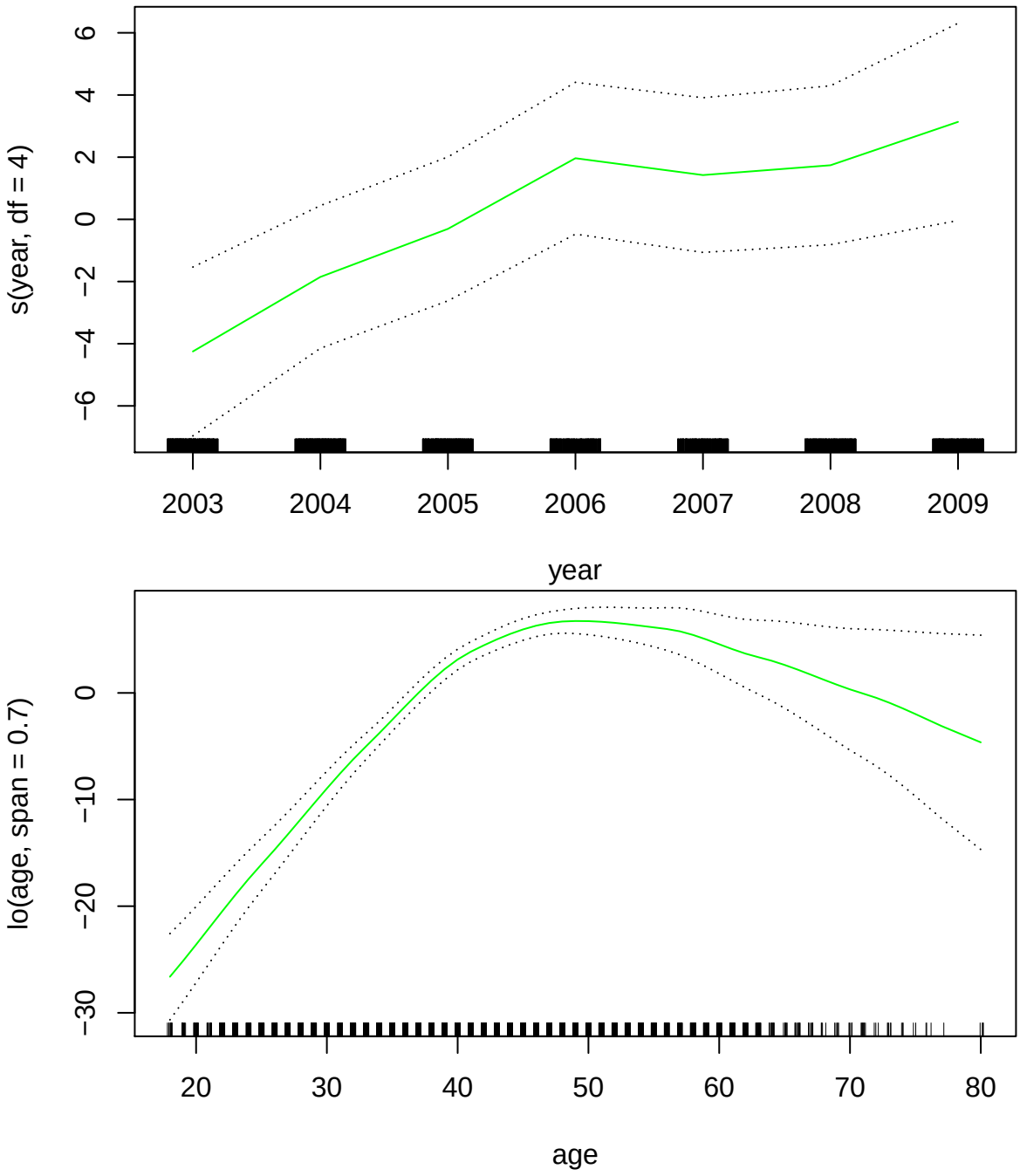
```r
anova(gam.m1,gam.m2,gam.m3,test="F")
```

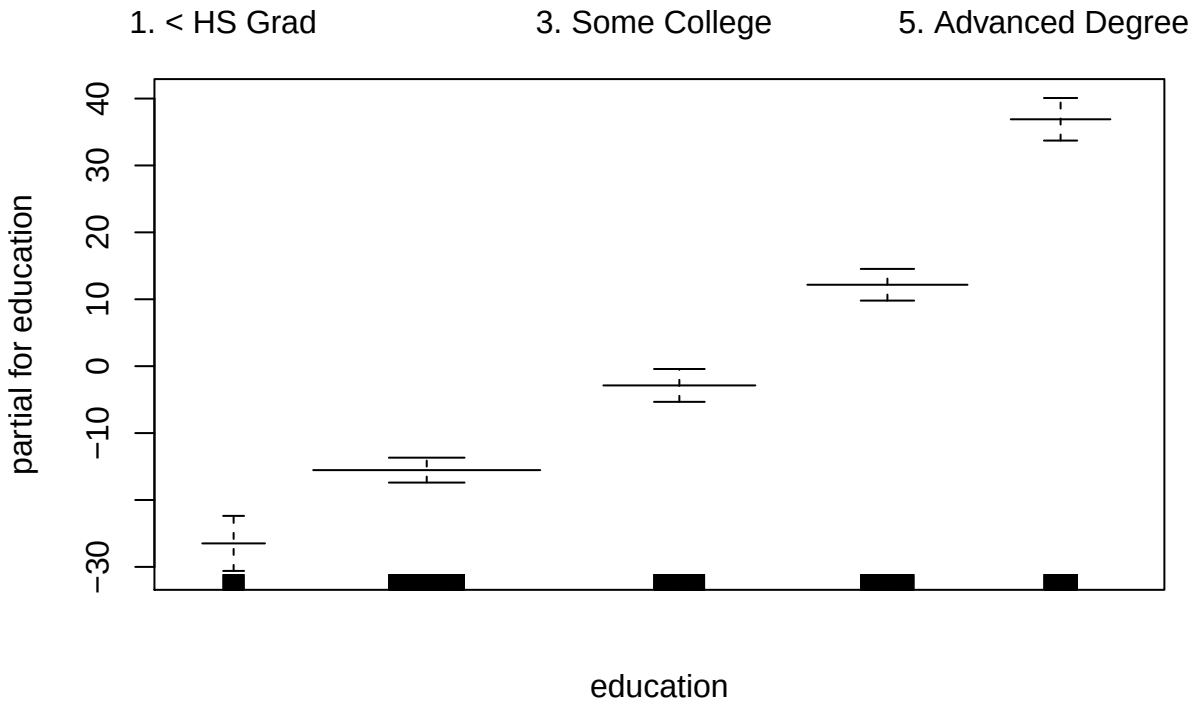
```
## Analysis of Deviance Table
##
## Model 1: wage ~ s(age, 5) + education
## Model 2: wage ~ year + s(age, 5) + education
## Model 3: wage ~ s(year, 4) + s(age, 5) + education
##   Resid. Df Resid. Dev Df Deviance       F    Pr(>F)
## 1      2990    3711731
## 2      2989    3693842  1  17889.2 14.4771 0.0001447 ***
## 3      2986    3689770  3   4071.1  1.0982 0.3485661
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
preds=predict(gam.m2,newdata=Wage)

gam.lo=gam(wage~s(year,df=4)+lo(age,span=0.7)+education,data=Wage)
```

```
## Warning in model.matrix.default(mt, mf, contrasts): non-list contrasts argument
## ignored
```

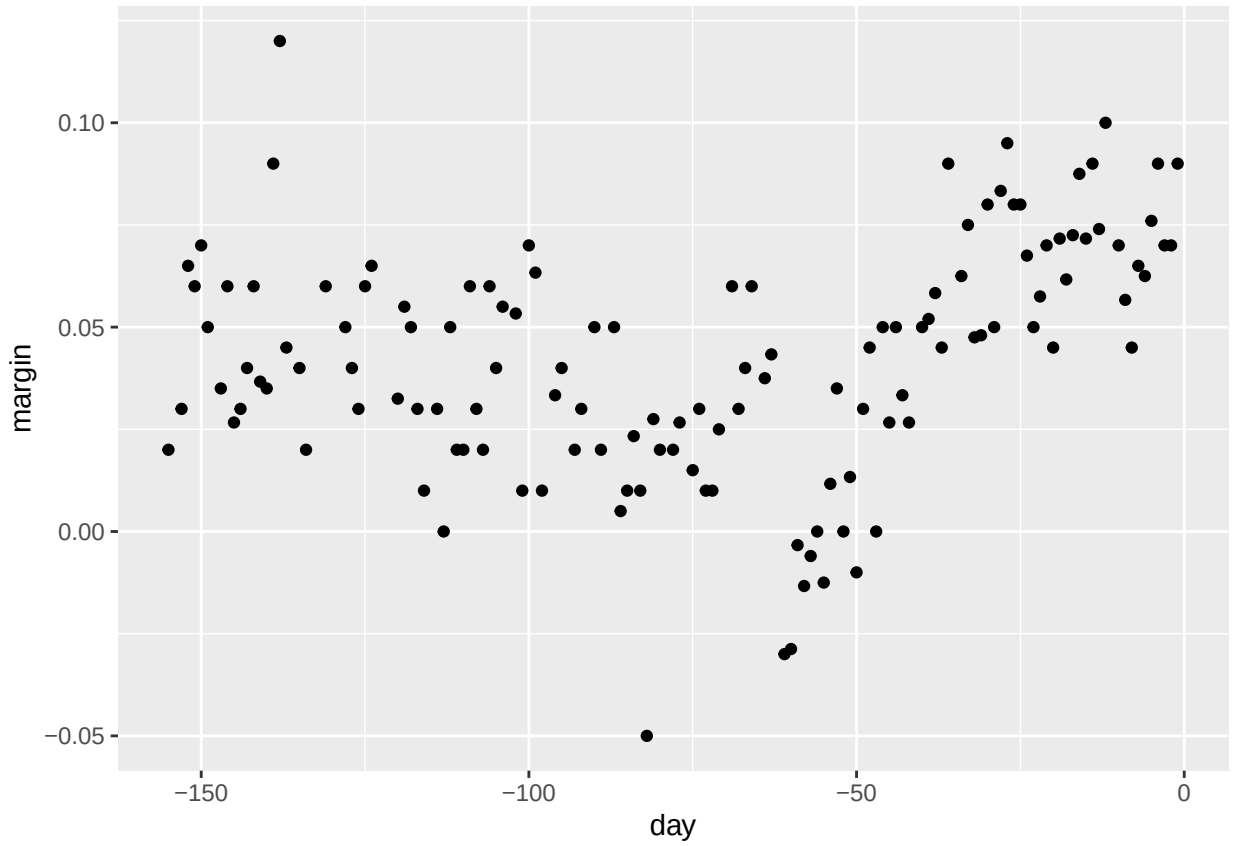
```r
plot(gam.lo, se=TRUE, col="green")
```


s(year, df = 4)
6
4
2
0
−2
−4
−6
2003
2004
2005
2006
2007
2008
2009
year
lo(age, span = 0.7)
0
−10
−20
−30
20
30
40
50
60
70
80
age

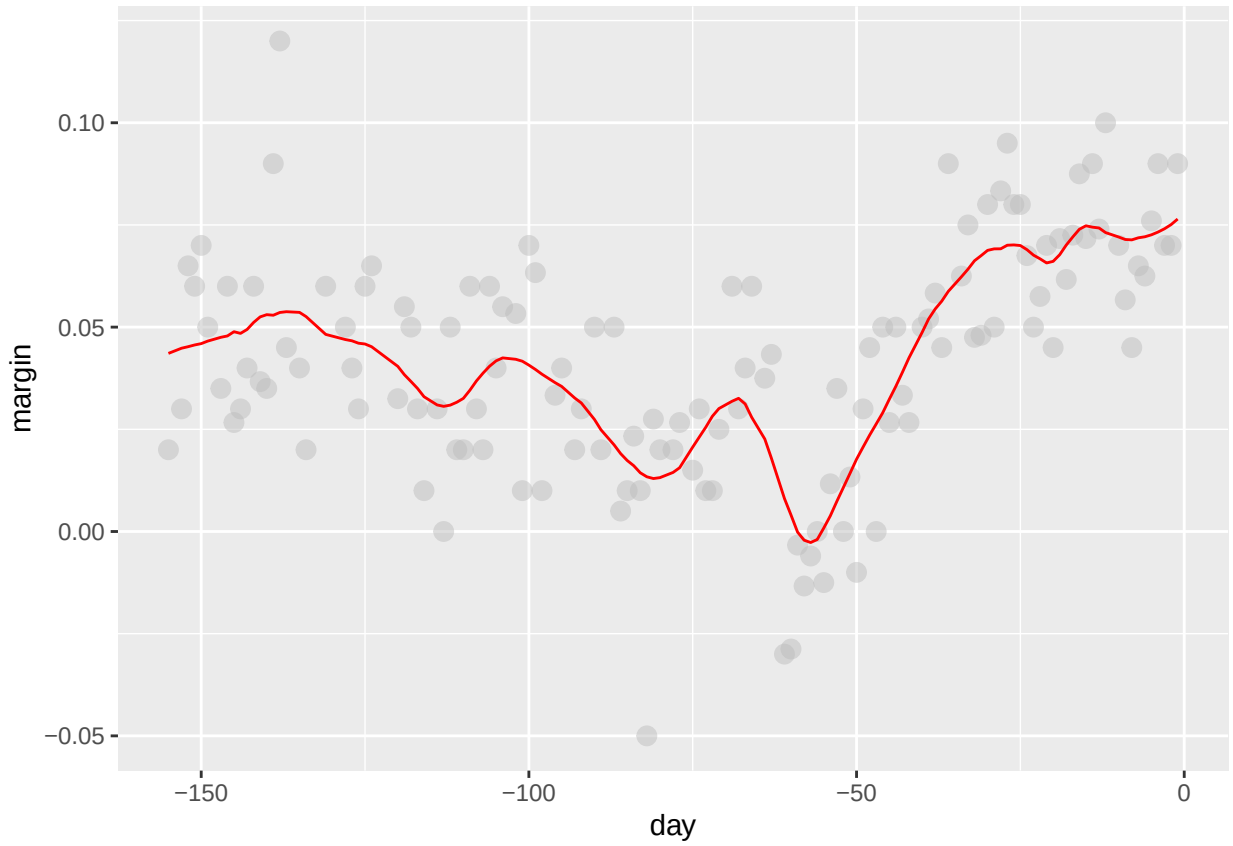


LOESS

```
library(tidyverse)
library(dslabs)
library(caret)

data("polls_2008")
qplot(day, margin, data = polls_2008)
```




```
total_days <- diff(range(polls_2008$day))
span <- 21/total_days

fit <- loess(margin ~ day, degree=1, span = span, data=polls_2008)

polls_2008 %>% mutate(smooth = fit$fitted) %>%
  ggplot(aes(day, margin)) +
  geom_point(size = 3, alpha = .5, color = "grey") +
  geom_line(aes(day, smooth), color="red")
```

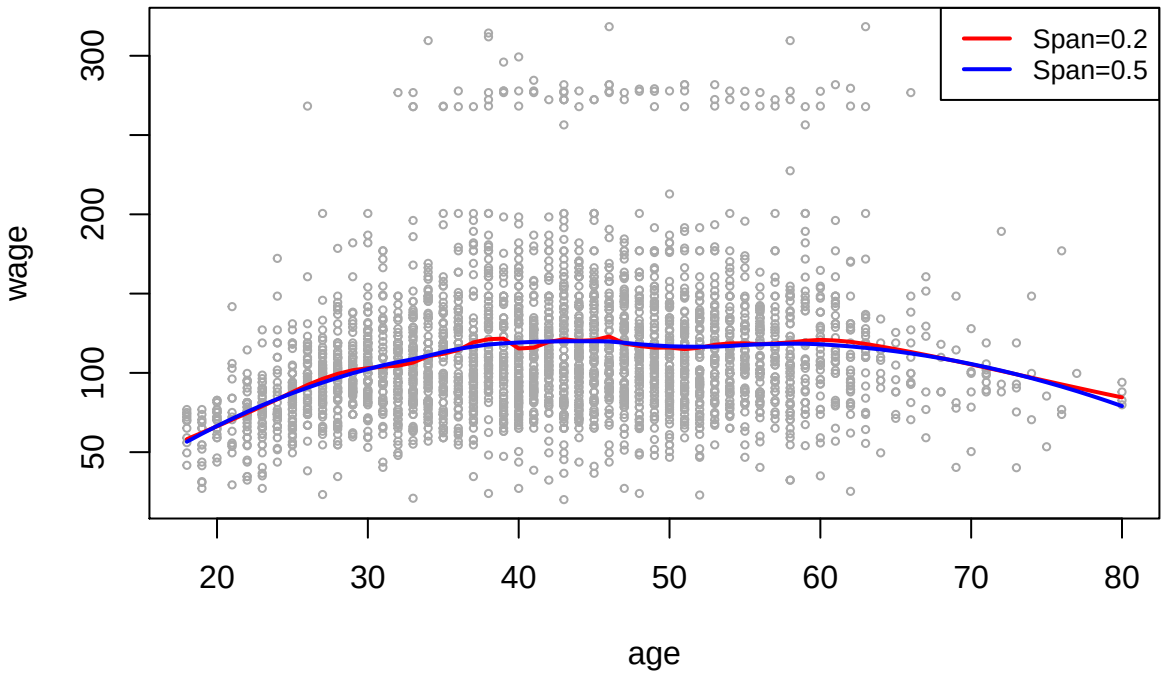


From the book

```
attach(Wage)
```

```
## The following objects are masked from Wage (pos = 4):
##
##     age, education, health, health_ins, jobclass, logwage, maritl,
##     race, region, wage, year
```

```
plot(age,wage,xlim=agelims ,cex=.5,col="darkgrey")
fit=loess(wage~age,span=.2,data=Wage)
fit2=loess(wage~age,span=.5,data=Wage)
lines(age.grid,predict(fit,data.frame(age=age.grid)),col="red",lwd=2)
lines(age.grid,predict(fit2,data.frame(age=age.grid)),col="blue",lwd=2)
legend("topright",legend=c("Span=0.2","Span=0.5"),col=c("red","blue"),lty=1,lwd=2,cex=.8)
```



LAB Assignment

S&P Data: Discuss this with your group members

We will working with the weekly S&P500 data.

```
names(Weekly)
```

```
## [1] "Year"      "Lag1"      "Lag2"      "Lag3"      "Lag4"      "Lag5"
## [7] "Volume"    "Today"     "Direction"
```

```
head(Weekly)
```

```
##   Year   Lag1   Lag2   Lag3   Lag4   Lag5    Volume  Today Direction
## 1 1990  0.816  1.572 -3.936 -0.229 -3.484 0.1549760 -0.270      Down
## 2 1990 -0.270  0.816  1.572 -3.936 -0.229 0.1485740 -2.576      Down
## 3 1990 -2.576 -0.270  0.816  1.572 -3.936 0.1598375  3.514        Up
## 4 1990  3.514 -2.576 -0.270  0.816  1.572 0.1616300  0.712        Up
## 5 1990  0.712  3.514 -2.576 -0.270  0.816 0.1537280  1.178        Up
## 6 1990  1.178  0.712  3.514 -2.576 -0.270 0.1544440 -1.372      Down
```

```
 #time series data
attach(Weekly)
```

We want to predict weekly Volume from performance in previous weeks.

Make an additive model that uses polynomial regression for each of the predictors.

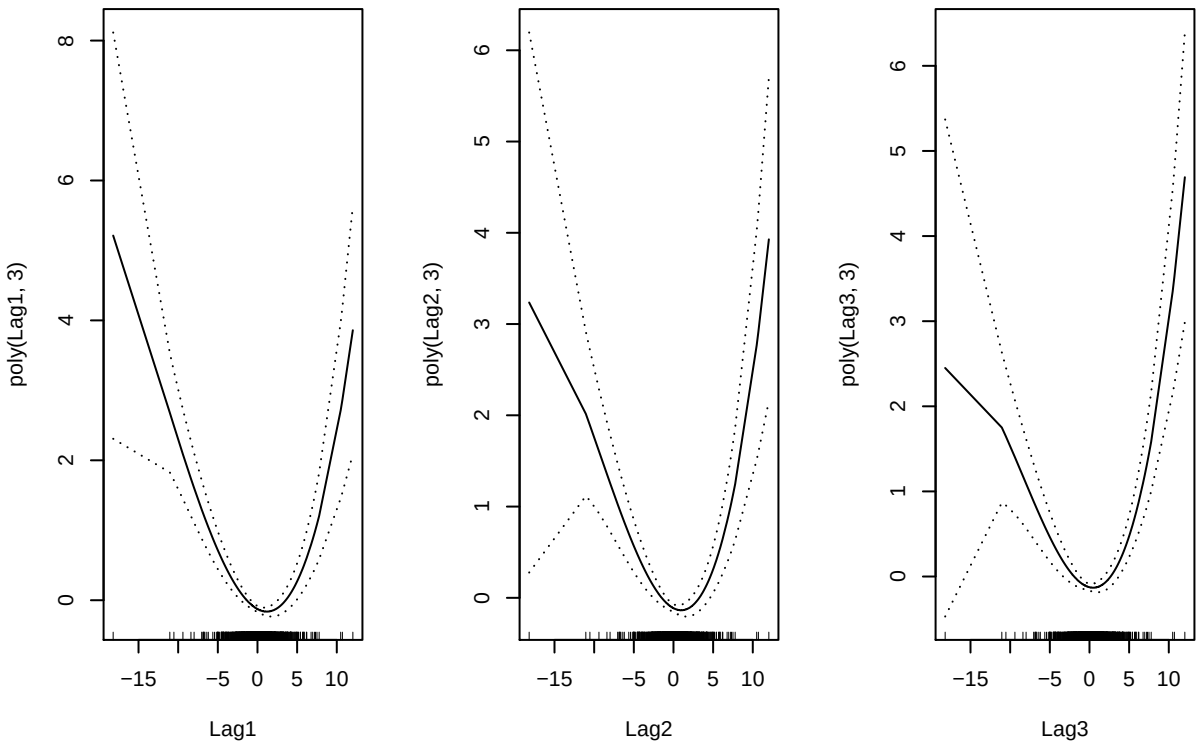
```
library(gam)
fit4 = gam(Volume ~ poly(Lag1,3 ) + poly(Lag2,3 ) + poly(Lag3,3 )) #polynomials of lag features
```

```
## Warning in model.matrix.default(mt, mf, contrasts): non-list contrasts argument
## ignored
```

```
summary(fit4)
```

```
##
## Call: gam(formula = Volume ~ poly(Lag1, 3) + poly(Lag2, 3) + poly(Lag3,
##     3))
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -3.5228 -1.0298 -0.6724  0.6498  7.8760
##
## (Dispersion Parameter for gaussian family taken to be 2.4057)
##
##     Null Deviance: 3095.08 on 1088 degrees of freedom
## Residual Deviance: 2595.744 on 1079 degrees of freedom
## AIC: 4058.368
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##                  Df  Sum Sq Mean Sq F value    Pr(>F)
## poly(Lag1, 3)     3  275.25  91.749  38.139 < 2.2e-16 ***
## poly(Lag2, 3)     3  132.33  44.109  18.335 1.317e-11 ***
## poly(Lag3, 3)     3   91.76  30.587  12.714 3.604e-08 ***
## Residuals      1079 2595.74   2.406
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

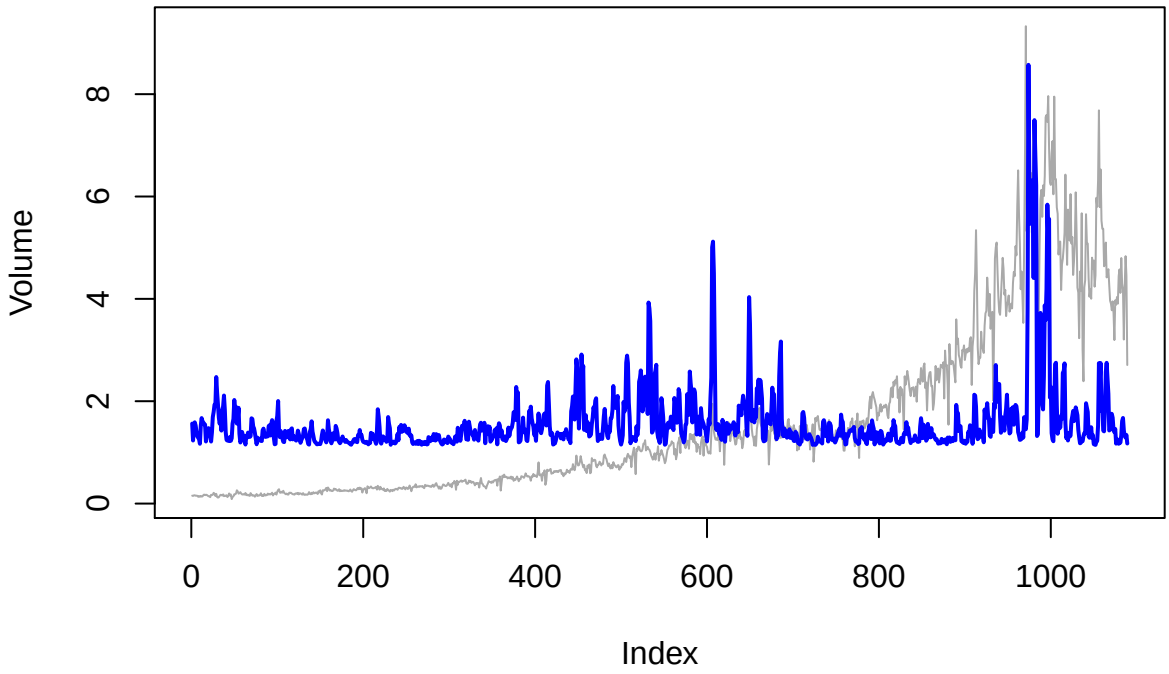
```
par(mfrow=c(1,3)) #to partition the Plotting Window
plot(fit4,se = TRUE)
```



Plot the data and the fits.

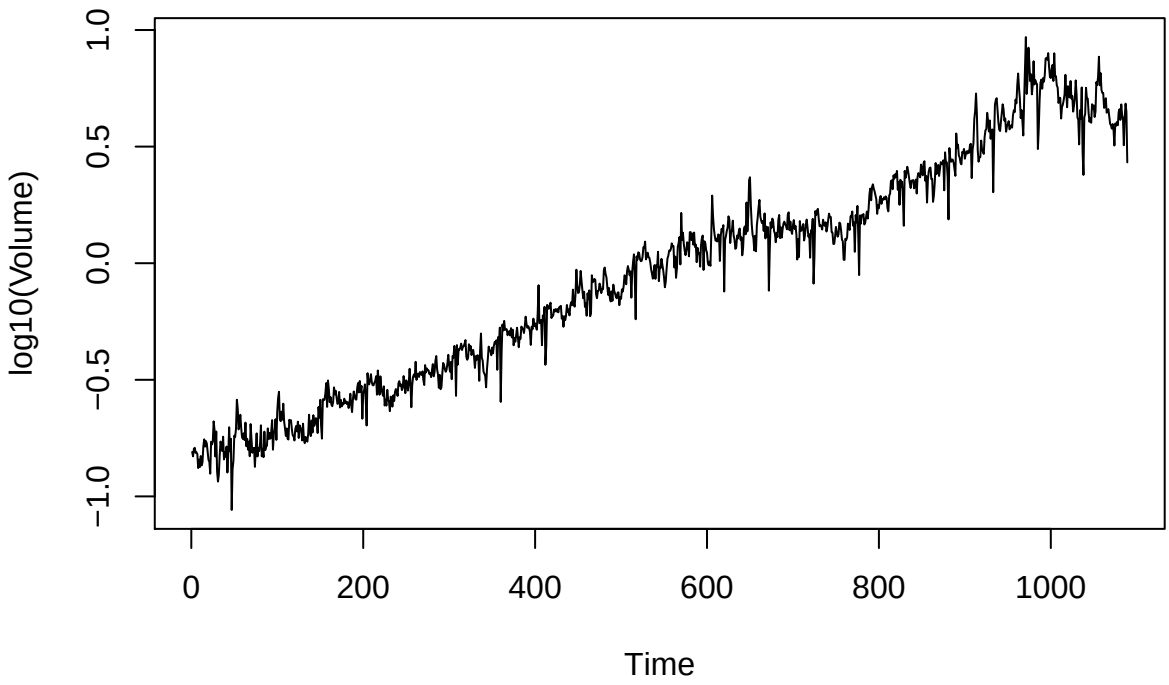
```
plot(Volume,col="darkgrey",type="l")
preds.Weekly=predict(fit4,se=TRUE)
```

```
lines(preds.Weekly$fit,lwd=2,col="blue") #need to treat this as time series data
```



Clearly the fits have nothing to do with the data. It is essential to include time in the fit. Also, the exponential increase of the trading volume suggests that one should look at the log of the volume.

```
plot.ts(log10(Volume), type="l")
```

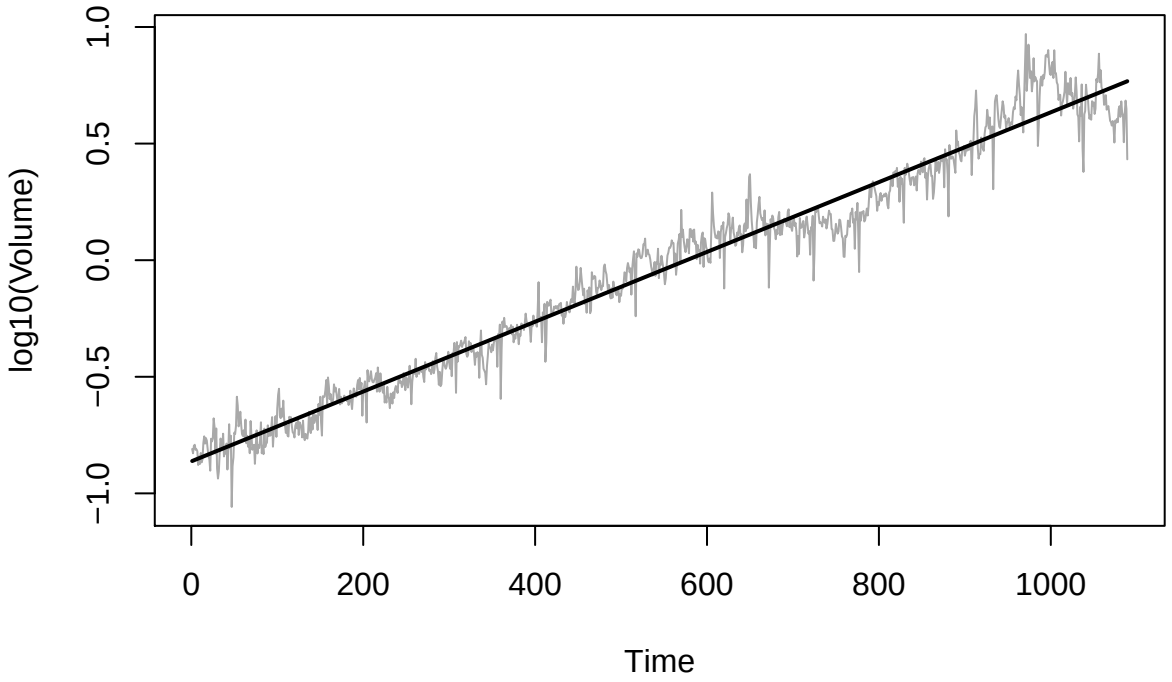


There is a general linear trend, the magnitude of the variation is about the same for all years, there are deviations from the linear trend around the years 2002 and 2008, and there may be seasonal variation.

Smoothing splines

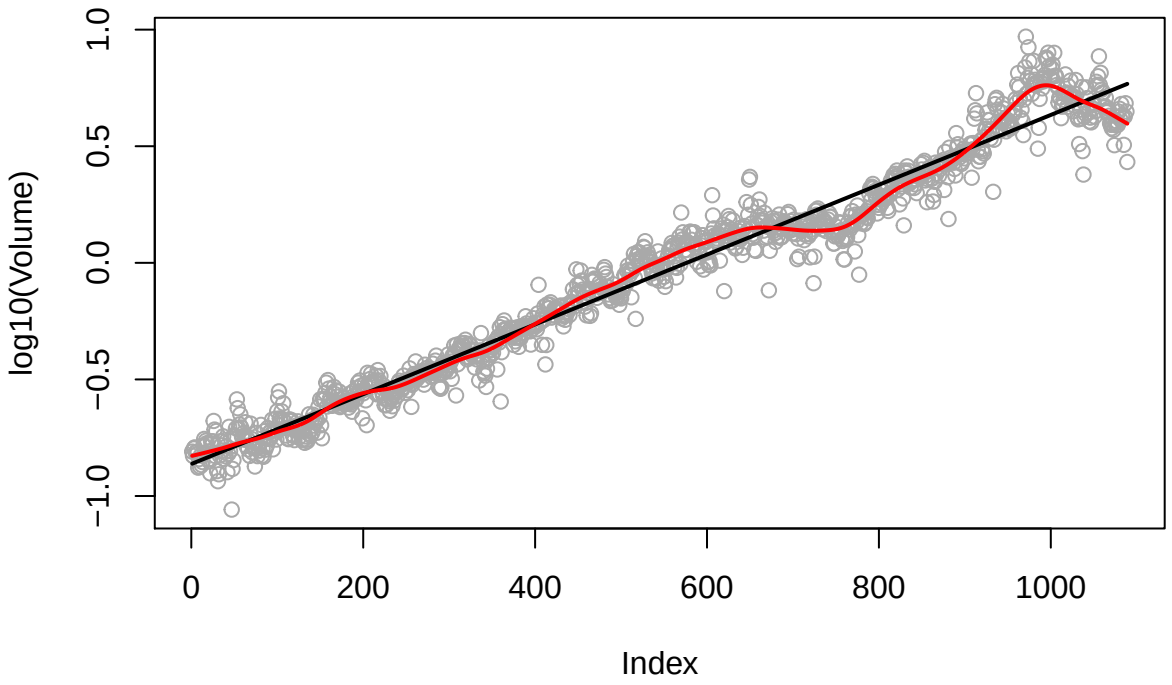
We can use smoothing splines to summarize these data. For very large penalty parameters λ or equivalently $df = 2$, a smoothing spline is essentially a straight line. We'll make such a smoothing spline, compute predictions, and plot it. Choosing a large lambda is equivalent to using two degrees of freedom (two parameters are fitted).

```
fit.s1 = smooth.spline(log10(Volume) ~  1:1089, df = 2) #constraint or flexible with df=2? linear?

#dim(Weekly)
#1089  9
preds.s1 = predict(fit.s1)
plot.ts(log10(Volume), col = "darkgrey")
lines(preds.s1$y, lwd =2 ) #high bias?low bias?
```



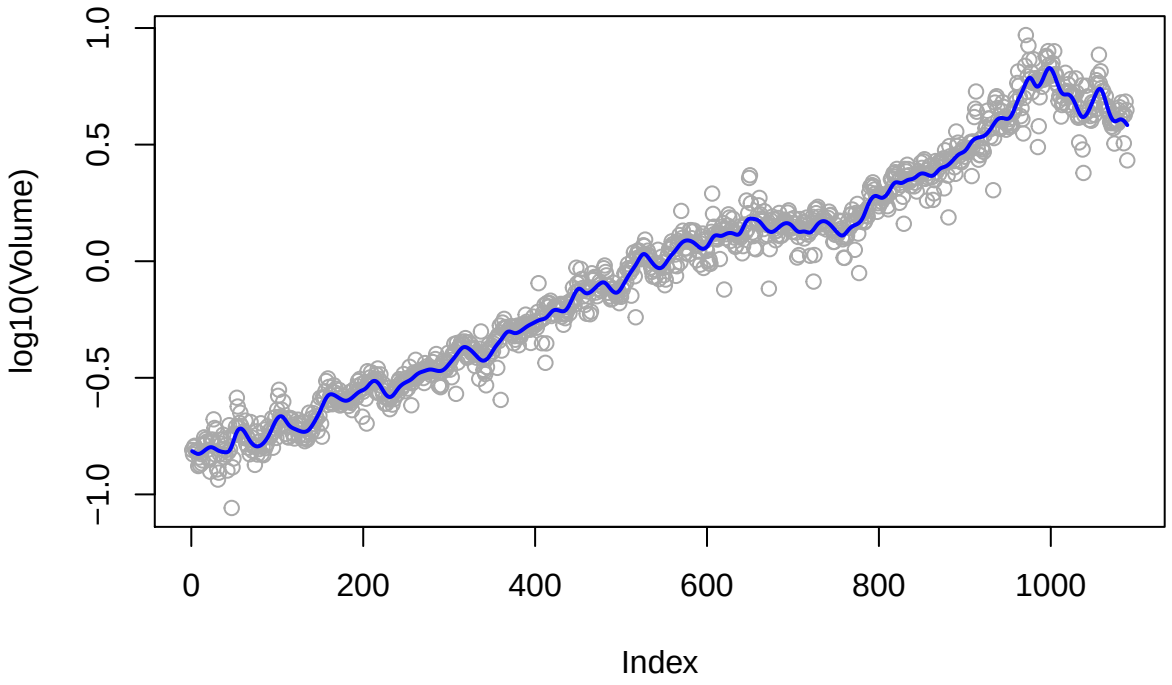
To capture year-to-year variation, increase the number of degrees of freedom. Let's use one df per year, plus one for the intercept. This is plotted in red.

```
fit.s1 = smooth.spline(log10(Volume) ~  1:1089, df = 2)
preds.s1 = predict(fit.s1)
fit.s22 = smooth.spline(log10(Volume) ~  1:1089, df = 22)
preds.s22 = predict(fit.s22)

plot(log10(Volume), col = "darkgrey")
lines(preds.s1$y, lwd =2 )
lines(preds.s22$y, lwd =2, col = 2)
```



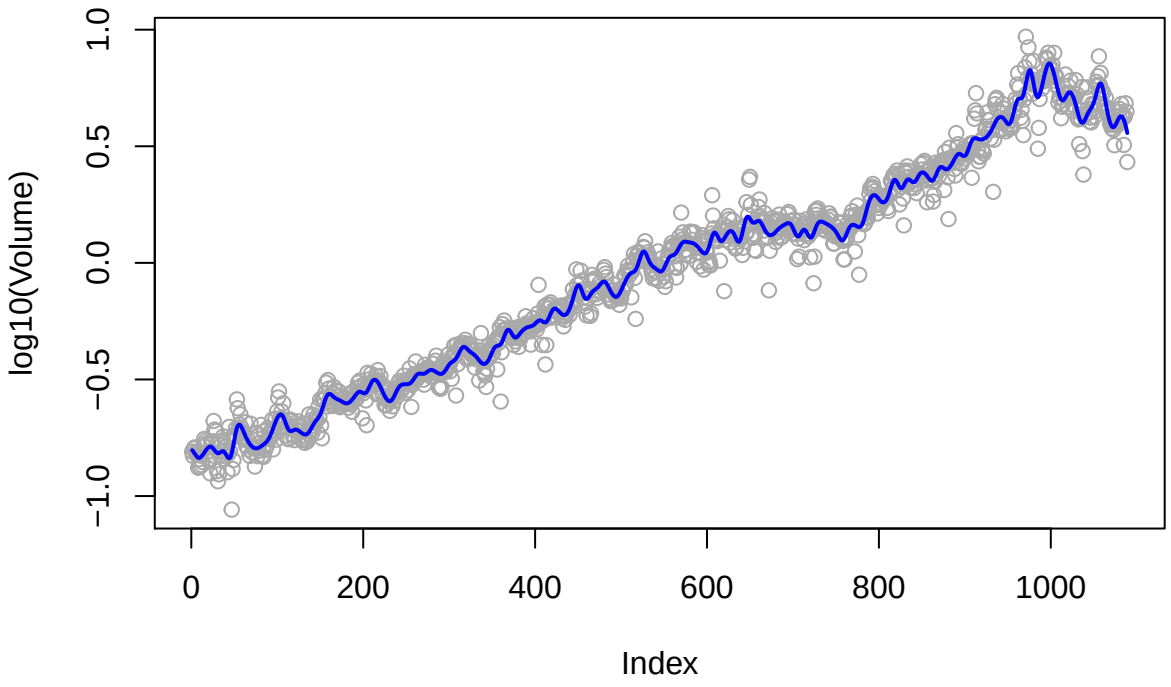
To capture also seasonal variation, increase the number of degrees of freedom further. We use four df per year, plus one for the intercept. This is plotted in blue.

```
fit.s85 = smooth.spline(log10(Volume) ~  1:1089, df = 85)
preds.s85 = predict(fit.s85)

plot(log10(Volume), col = "darkgrey")
lines(preds.s85$y, lwd =2, col = 4) #what can you say about the overall market trend?
```



When the number of degrees of freedom is not specified, `gam` chooses one. Here is the resulting plot. This fit uses about 120 degrees of freedom.

```
fit.s = smooth.spline(log10(Volume) ~  1:1089)
preds.s = predict(fit.s)

plot(log10(Volume), col = "darkgrey")
lines(preds.s$y, lwd =2, col = 4)
```




```
fit.s
```

```
## Call:
## smooth.spline(x = log10(Volume) ~ 1:1089)
##
## Smoothing Parameter  spar= 0.1833952  lambda= 3.62849e-08 (12 iterations)
## Equivalent Degrees of Freedom (Df): 120.1739
## Penalized Criterion (RSS): 2.895513
## GCV: 0.0033594
```

2. This question relates to the College data set.

(a) Split the data into a training set and a test set. Using out-of-state tuition as the response and the other variables as the predictors, perform forward stepwise selection on the training set in order to identify a satisfactory model that uses just a subset of the predictors. plot the BIC to pick the best model (cross validation could also be used).

(b) Fit a GAM on the training data, using out-of-state tuition as the response and the features selected in the previous step as the predictors. Plot the results, and explain your findings.

(c) Evaluate the model obtained on the test set, and explain the results obtained.

(d) For which variables, if any, is there evidence of a non-linear relationship with the response? (make scatterplots of the response against the five numerical predictors.)

We can make scatterplots of the response against the five numerical predictors.