ZZCS8506 — Regression analysis for data Scientists

Atefeh Zamani

Based on the notes provided by Boris Beranger

Deparment of Statistics

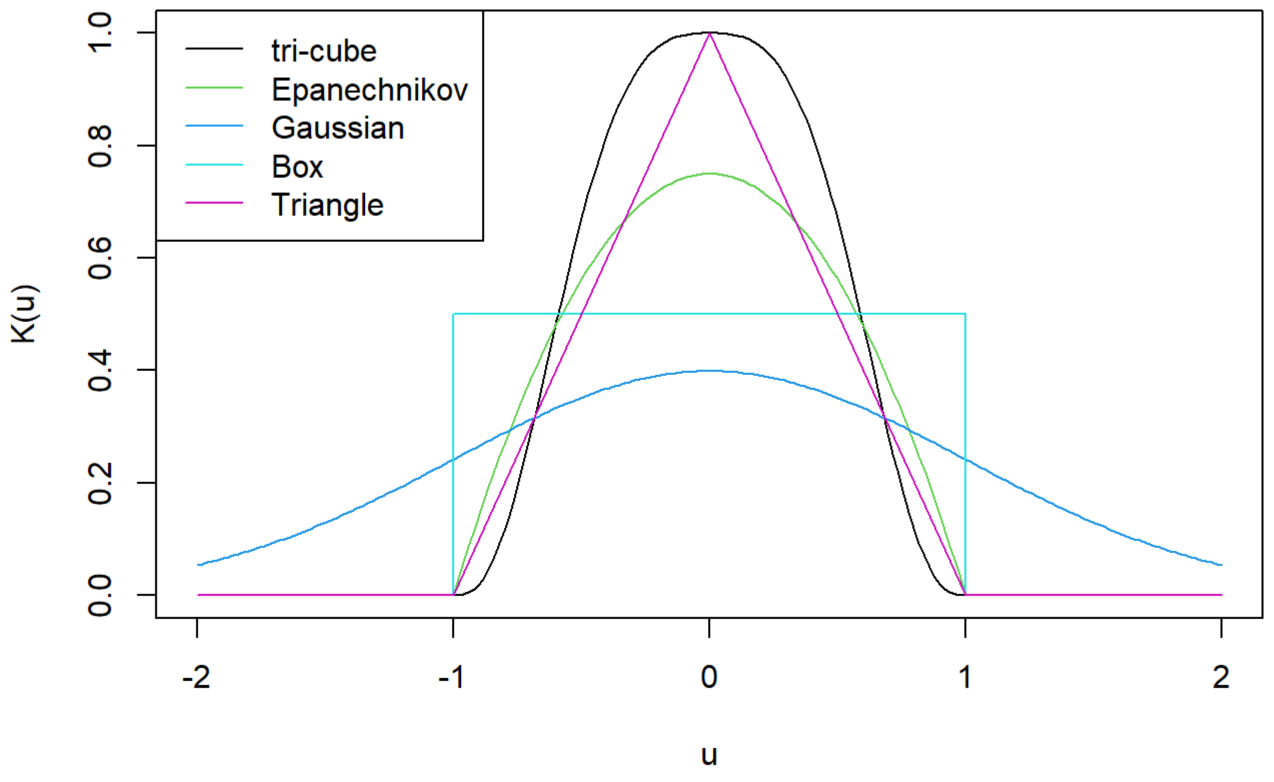
University of New South Wales

5.1. Loess Smoother

Different kinds of weighting functions

Below, we visualise some weighting functions that can be used in the loess algorithm. Note that they all integrate to 1. They mathematical definition is provided in the teaching materilas.

```
xx1=seq(from = -1,to = 1,length.out = 100)
# tri-cube Kernel
yy=(1-abs(xx1)^3)^3
plot(xx1,yy,xlim = c(-2,2),type='l',xlab='u',ylab='K(u)'); lines(x=c(-2,-1),y=c(0,
0)); lines(c(1,2),c(0,0))
# Epanechikov kernel
yy=3/4*(1-xx1^2)
lines(xx1,yy,col=3)
xx2=seq(from = -2,to = 2,length.out = 200)
# Gaussian kernel
yy=dnorm(xx2)
lines(xx2,yy,col=4)
#Box kernel
lines(c(-2,-1,-1,1,1,2),c(0,0,1/2,1/2,0,0),col=5)
#Triangular kernel
lines(c(-2,-1,0,1,2),c(0,0,1,0,0),col=6)
legend("topleft", legend=c("tri-cube", "Epanechnikov", "Gaussian", "Box","Triangl
e"),lty=1, col=c(1,3,4,5,6))
```



Example: Scatterplot smoothers (Loess)

In this example, we are going to work with the diabetes data set from the start of this section. This is data from a study of the factors affecting patterns of insulin-dependent diabetes mellitus in children, Sockett et al. 1987.

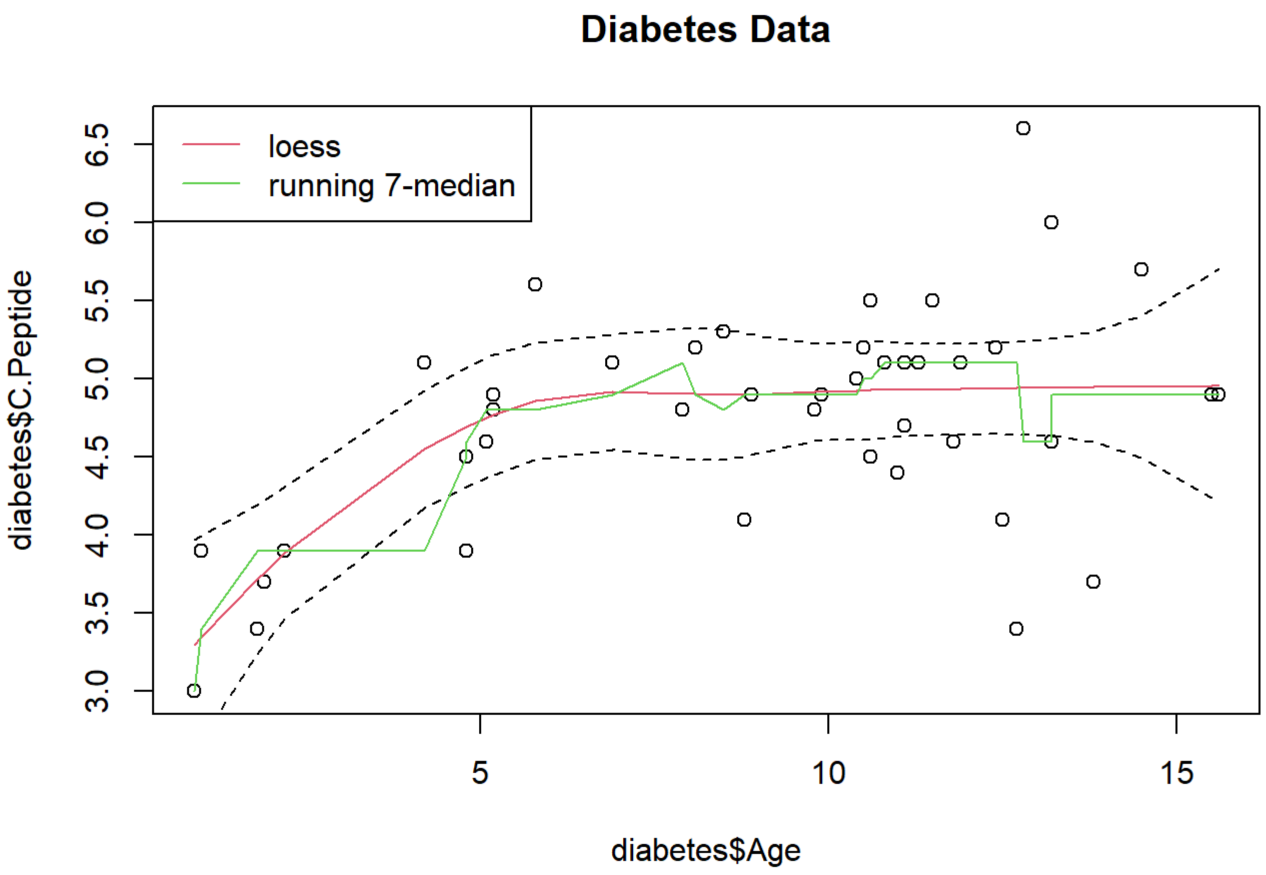
```
diabetes <- read.table("diabetes.dat", header=TRUE, quote="\"")
diabetes <- diabetes[order(diabetes$Age),] #put the age in ascending order (not neces
sary here)
head(diabetes)
```

	Age <dbl>	Base.Deficit <dbl>	C.Peptide <dbl>
15	0.9	-11.6	3.0
24	1.0	-8.2	3.9
6	1.8	-19.2	3.4
10	1.9	-25.0	3.7
11	2.2	-3.1	3.9
36	4.2	-17.0	5.1

6 rows

The scatterplot smoothers for the diabetes data set were obtained as follows:

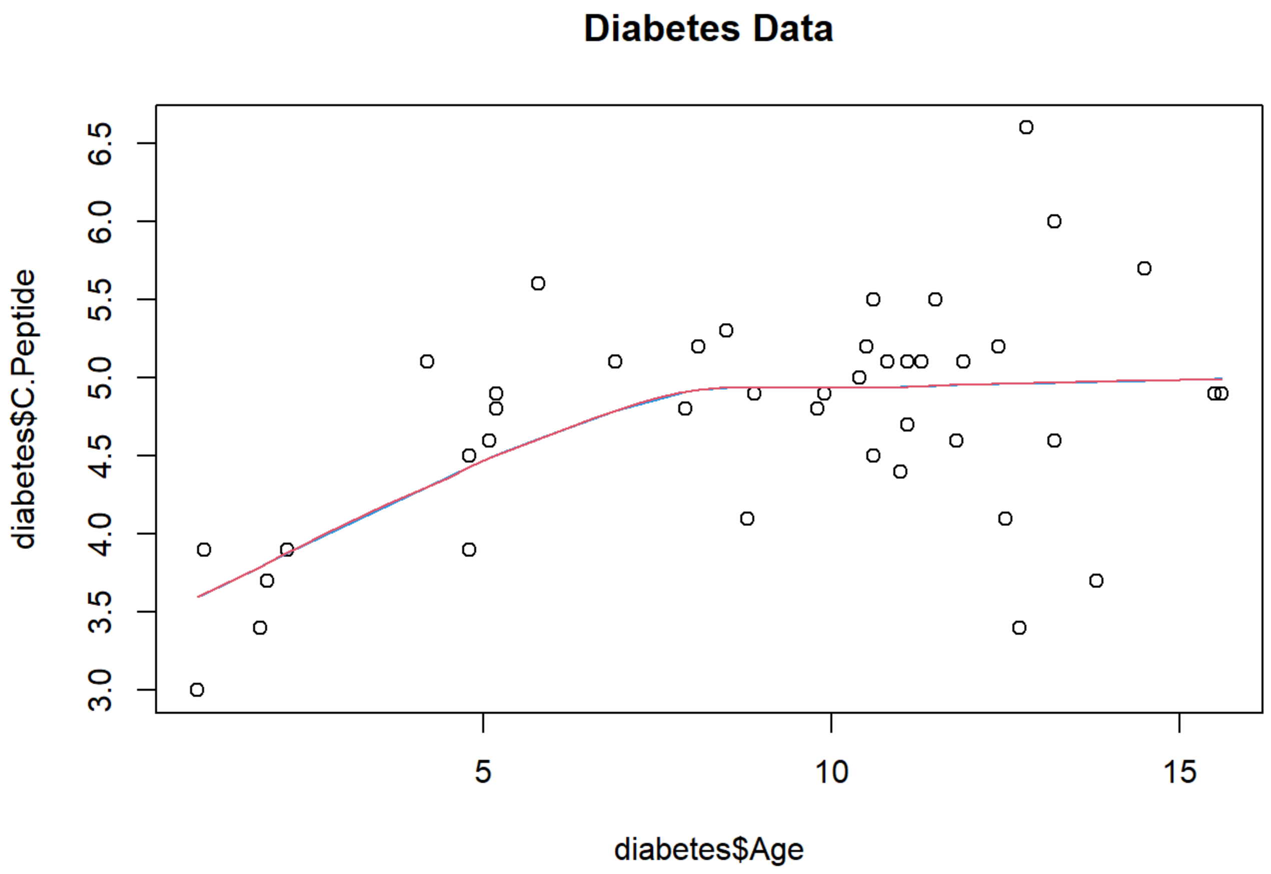
```
plot(diabetes$Age,diabetes$C.Peptide,main="Diabetes Data")
# loess: Fit a polynomial surface determined by one or more numerical predictors, usi
ng local fitting.
#The size of the neighborhood can be controlled using the span argument, which ranges
between 0 to 1. It controls the degree of smoothing. So, the greater the value of spa
n, more smooth is the fitted curve. Default span=0.75
fit=loess(diabetes$C.Peptide~diabetes$Age,diabetes)
# predict: Predictions from a loess fit, optionally with standard errors,
# se: should standard errors be computed?
pred=predict(fit,diabetes$Age,se=T)
# df in the output stands for effective degrees of freedom (Page 278-279 ISLR)
lines(diabetes$Age,predict(fit),col=2)
# Confidence interval
lines(diabetes$Age,pred$fit+2*pred$se.fit,lty=2)
lines(diabetes$Age,pred$fit-2*pred$se.fit,lty=2)
# runmed: Compute running medians of odd span
lines(diabetes$Age,runmed(x = diabetes$C.Peptide,k = 7),col=3)
legend("topleft", legend = c("loess", "running 7-median"),lty=c(1,1), col = 2:3)
```



```
# lines(loess.smooth(diabetes$Age,diabetes$C.Peptide, span=0.75),col=6)
# The function loess.smooth: returns a list with two components, x and y. x contains
a set of predictor values and y is the vector of smoothed values given by loess.smoot
h. The default specification of k in loess.smooth is a "span" of 2/3 of all data valu
es.
#lines(loess.smooth(diabetes$Age,diabetes$C.Peptide),col=2)
```

If you set the parameters of `loess.smooth` function and `loess` function the same, you will end up with smooth carves very close to each other. Please run the following code:

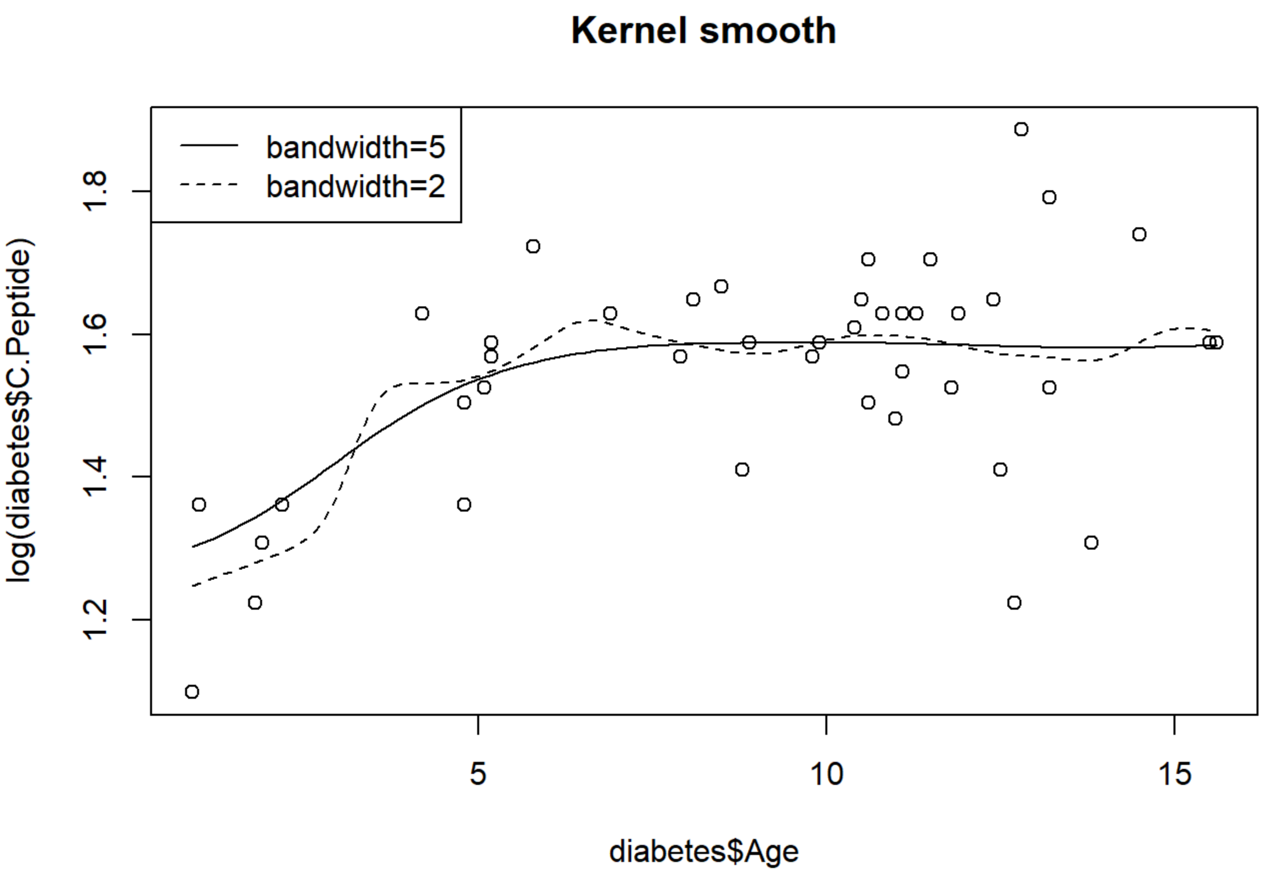
```
plot(diabetes$Age,diabetes$C.Peptide,main="Diabetes Data")

fit=loess(C.Peptide~Age,data= diabetes, span = 0.75,family = "symmetric", degree=1)
# in loess function the degree of the polynomials are normally 1 or 2.
pred=predict(fit,diabetes$Age,se=T)
lines(diabetes$Age,predict(fit),col=4)

fit1=loess.smooth(diabetes$Age, diabetes$C.Peptide,  span = 0.75,family = "symmetri
c", degree=1)
lines(fit1,col=2)
```



Example: Kernel smoothing

In this example we illustrate kernel smoothing for the diabetes data (with $\log(\mathtt{C.peptide})$ as the response and $\mathtt{age}$ as the predictor):

```
# ksmooth: The Nadaraya–Watson kernel regression estimate.
plot(diabetes$Age,log(diabetes$C.Peptide))
lines(ksmooth(diabetes$Age,log(diabetes$C.Peptide), bandwidth=5.0, kernel="normal"))
lines(ksmooth(diabetes$Age,log(diabetes$C.Peptide), bandwidth=2.0, kernel="normal"),
      lty=2)
title("Kernel smooth")
legend("topleft",c("bandwidth=5", "bandwidth=2"),lty=1:2)
```



As for *loess.smooth()*, the output of the function *ksmooth()* is a list with components `x` and `y` (numeric vectors, by default `x` is the set of observed predictors and `y` gives corresponding values of the kernel smooth).

The argument `bandwidth` controls the degree of smoothing (the default value is usually not sensible) and `kernel` allows specification of the kernel (options `box` and `normal`, see help for details).

Example: Local likelihood

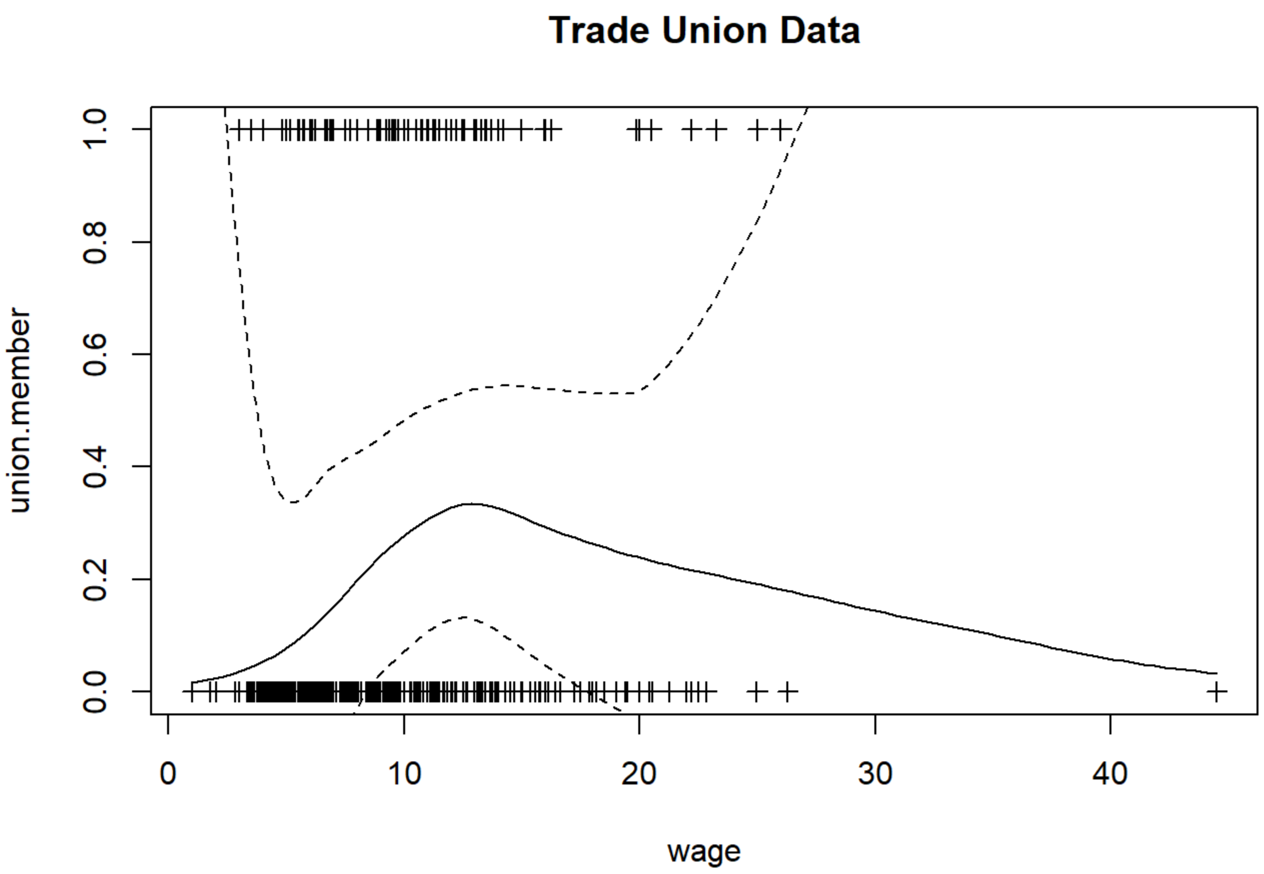
Recall the trade union data set, which we have used as an example for logistic regression. The package `locfit` fits a nonparametric logistic regression model as follows:

```r
library(SemiPar)

data("trade.union")
attach(trade.union)
require(locfit)
```

```
## Loading required package: locfit
```

```
## locfit 1.5-9.8    2023-06-11
```

```r
#locfit: is the model formula-based interface for fitting local regression and likeli
hood models.
# lp: is a local polynomial model term for Locfit models. Usually, it will be the onl
y term on the RHS of the model formula.
  #nn: Nearest neighbor component of the smoothing parameter. Default value is 0.7, u
nless either h or adpen are provided, in which case the default is 0.
  # h   : The constant component of the smoothing parameter. Default: 0.

fit=locfit(union.member~lp(wage,nn=0.7,deg=2), data=trade.union,family="binomial")
plot(fit,ylim = c(0,1),type='n',main="Trade Union Data")
newxx=seq(from = min(wage),to = max(wage),length.out = 100)
newdat=data.frame(wage=newxx)
ypred=predict(fit,newdat,se.fit = T)
lines(newxx,ypred$fit)
lines(newxx,ypred$fit-ypred$se.fit,lty=2)
lines(newxx,ypred$fit+ypred$se.fit,lty=2)
# jitter: Add a small amount of noise to a numeric vector
points(jitter(wage), union.member, main = "trade union data set", pch = 3)
```



Activity in R 1: Loess Smoother

Download the data set `ethanol`. The `ethanol` data frame contains 88 measurements generated in an experiment in which ethanol was burned in a single cylinder automobile test engine. The variables are `NOx` (a measure of the nitrogen oxides in the exhaust), `C` (compression ratio of an engine) and `E` (a measure of the richness of the air/ethanol mix used).

```
ethanol<-read.table("ethanol.txt",header=T)
# data in SemiPar package
print(dim(ethanol))
```

```
## [1] 88  3
```

```
head(ethanol)
```

	NOx <dbl>	C <dbl>	E <dbl>
1	3.741	12	0.907
2	2.295	12	0.761
3	1.498	12	1.108

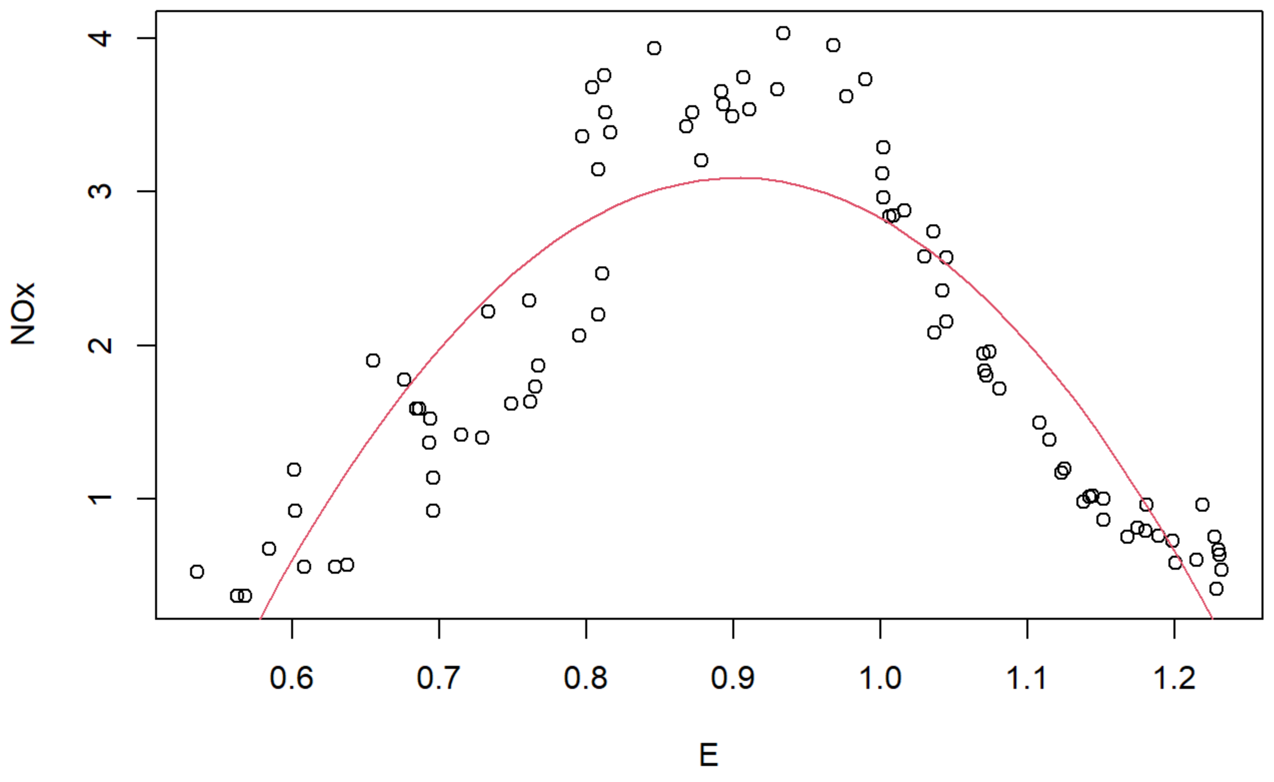
	NOx <dbl>	C <dbl>	E <dbl>
4	2.881	12	1.016
5	0.760	12	1.189
6	3.120	9	1.001

6 rows

```
attach(ethanol)
```

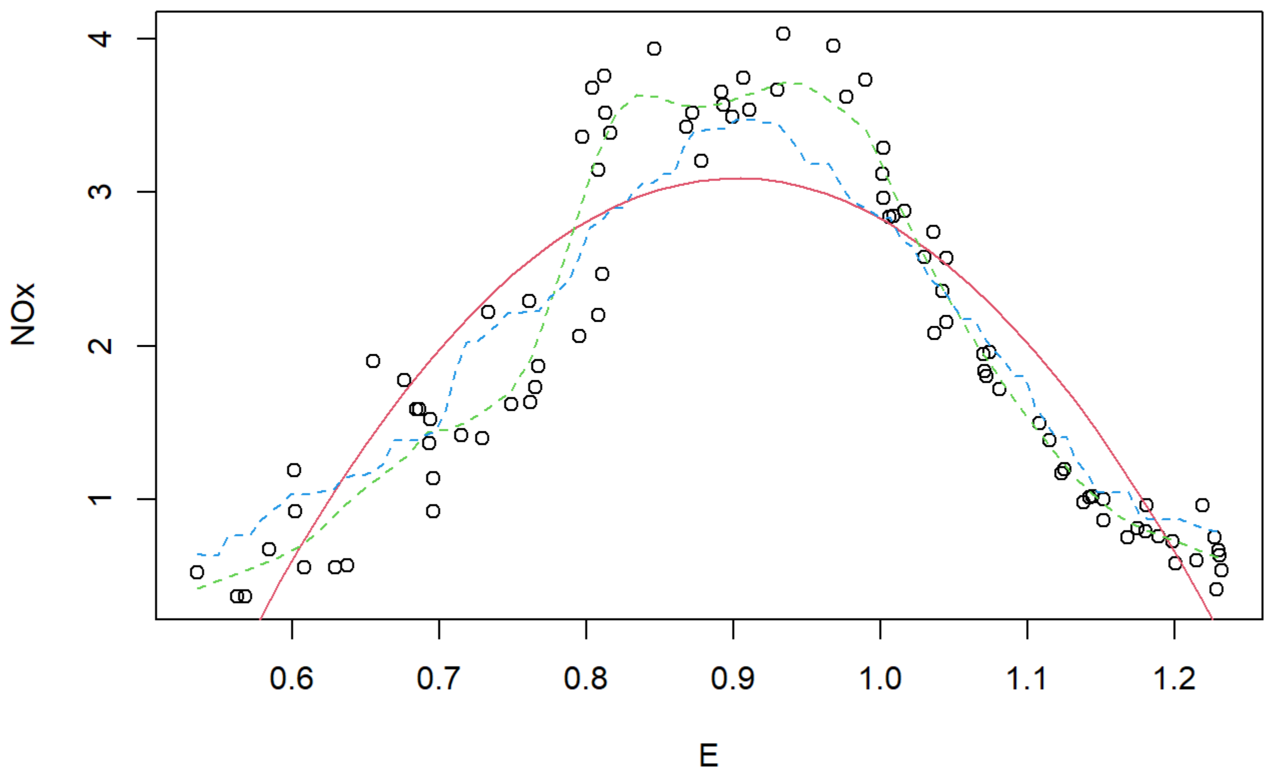
a. Do a scatterplot of `NOx` against `E`. After looking at the scatterplot you might suggest a polynomial regression model to capture the relationship between the response `NOx` and the predictor `E`. Fit a quadratic regression model and superimpose the fitted curve on the scatterplot.

```
plot(E, NOx)
Esq<-E*E
ethanol.lm<-lm(NOx~E+Esq)
Egrid<-seq(from=min(E),to=max(E),length=50)
new.data<-data.frame(E=Egrid,Esq=Egrid*Egrid)
ethanol.pred<-predict(ethanol.lm,newdata=new.data)
lines(Egrid,ethanol.pred,col=2)
```



b. Superimpose on the scatterplot in a) a suitable loess smooth. Do you think the smooth captures structure in the data that is missed by the quadratic regression model?

```
plot(E, NOx)
Esq<-E*E
ethanol.lm<-lm(NOx~E+Esq)
Egrid<-seq(from=min(E),to=max(E),length=50)
new.data<-data.frame(E=Egrid,Esq=Egrid*Egrid)
ethanol.pred<-predict(ethanol.lm,newdata=new.data)
lines(Egrid,ethanol.pred,col=2)
lines(loess.smooth(E,NOx,span=0.2),lty=2,col=3)
lines(ksmooth(E,NOx,bandwidth=0.2),lty=2,col=4)
```



From the above figure, it seems that loess smooth capture more bell-shaped structure of the data. This is slightly different from the quadratic regression fit.

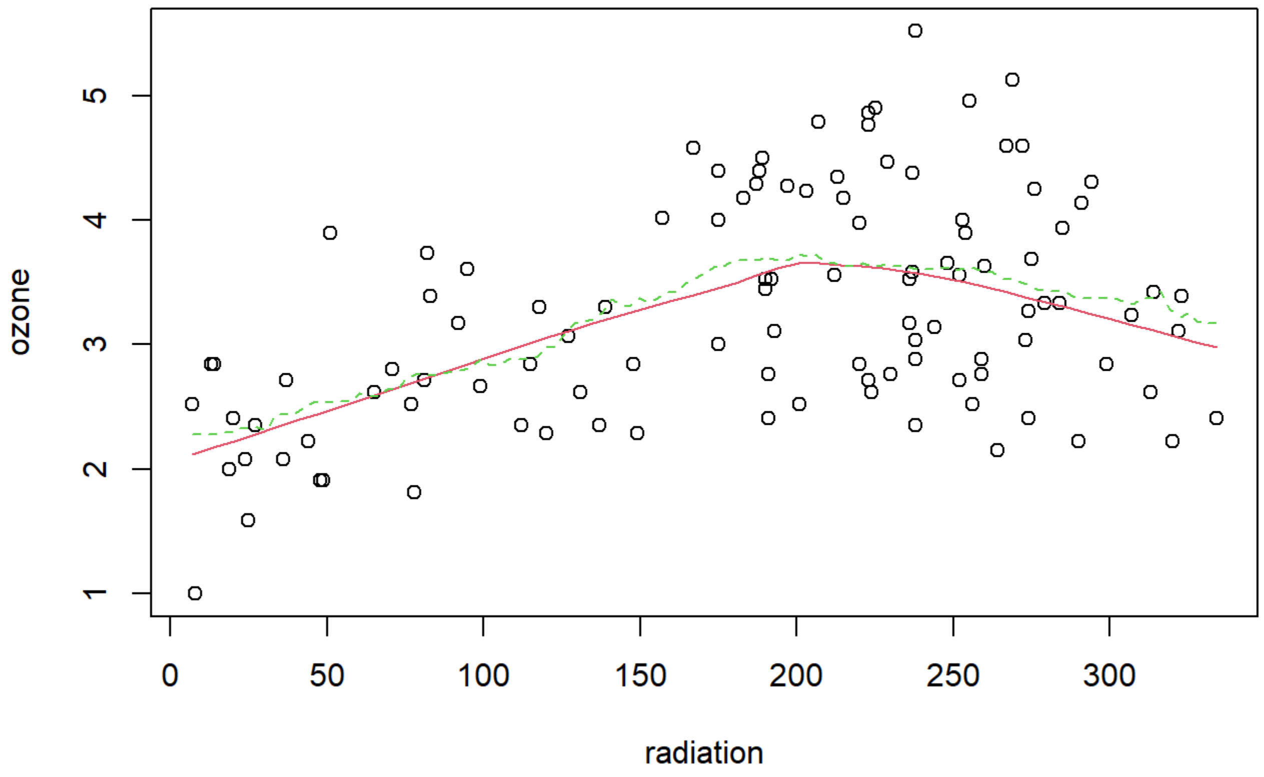
```
# library(fANCOVA)
# degree: the degree of the local polynomials to be used. It can ben 0, 1 or 2
# Fit a local polynomial regression with automatic smoothing parameter selection.
# fit_loess <- loess.as(E,NOx, degree = 1, criterion = "gcv", user.span = NULL, plot
= T)
# summary(fit_loess)
```

Activity in R: Kernel Smoother

Download the data set air (there are four columns in this data frame, ozone, radiation, temperature and wind). The data set records New York Air Quality Measurements, May to September 1973.

Plot a scatterplot of ozone against radiation and superimpose loess and kernel smooths on the scatterplot. You may need to change the default level of smoothing for the kernel and loess smooths.

```
air<-read.table("air.txt",header=T)
attach(air)
plot(radiation,ozone)
# loess.smooth:Plot and add a smooth curve computed by loess to a scatter plot. You c
an add degree as a parameter # default span=2/3
lines(loess.smooth(radiation,ozone),col=2)
lines(ksmooth(radiation,ozone,bandwidth=100),lty=2,col=3)
```



```
# bandwidth: the bandwidth. The kernels are scaled so that their quartiles (viewed as
probability densities) are at ± 0.25*bandwidth.
```

Section 5.2 Polynomial regression and step functions

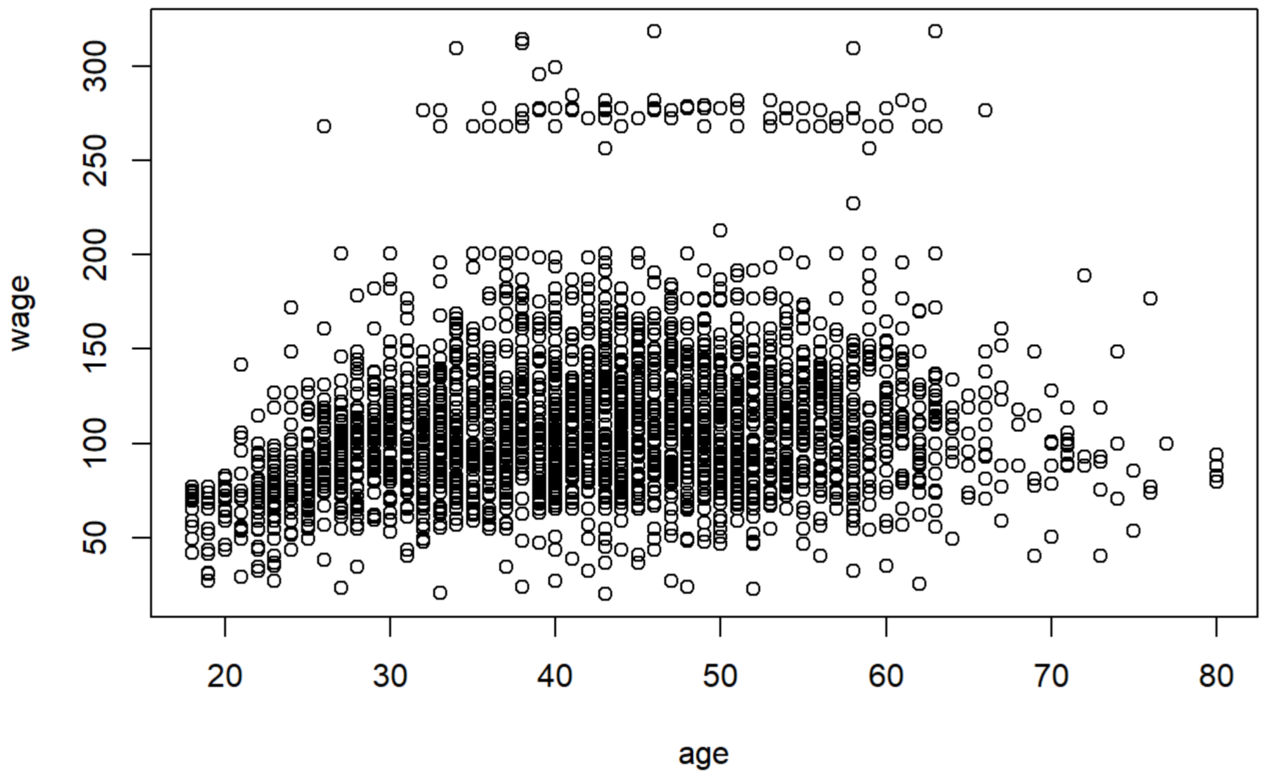
Example: Polynomial regression

In this example we analyze the `Wage` data from the `ISLR` library.

```
library(ISLR)
data("Wage")
attach(Wage)
```

```
## The following objects are masked from trade.union:
##
##     age, race, wage
```

```
plot(age,wage)
```



```
fit=lm(wage~poly(age,4,raw=T), data=Wage)
# if true, use raw and not orthogonal polynomials
coef(summary(fit))
```

```
##                            Estimate   Std. Error   t value     Pr(>|t|)
## (Intercept)            -1.841542e+02 6.004038e+01 -3.067172 0.0021802539
## poly(age, 4, raw = T)1  2.124552e+01 5.886748e+00  3.609042 0.0003123618
## poly(age, 4, raw = T)2 -5.638593e-01 2.061083e-01 -2.735743 0.0062606446
## poly(age, 4, raw = T)3  6.810688e-03 3.065931e-03  2.221409 0.0263977518
## poly(age, 4, raw = T)4 -3.203830e-05 1.641359e-05 -1.951938 0.0510386498
```

There are several other equivalent ways of fitting this model:

```
fit2=lm(wage~age+I(age^2)+I(age^3)+I(age^4),data=Wage)
coef(summary(fit2))
```

```
##                  Estimate   Std. Error   t value     Pr(>|t|)
## (Intercept) -1.841542e+02 6.004038e+01 -3.067172 0.0021802539
## age          2.124552e+01 5.886748e+00  3.609042 0.0003123618
## I(age^2)    -5.638593e-01 2.061083e-01 -2.735743 0.0062606446
## I(age^3)     6.810688e-03 3.065931e-03  2.221409 0.0263977518
## I(age^4)    -3.203830e-05 1.641359e-05 -1.951938 0.0510386498
```

```
## https://stackoverflow.com/questions/24192428/what-does-the-capital-letter-i-in-r-l
inear-regression-formula-mean
## https://stackoverflow.com/questions/8055508/in-r-formulas-why-do-i-have-to-use-th
e-i-function-on-power-terms-like-y-i
```

or

```
fit3=lm(wage~cbind(age,age^2,age^3,age^4),data=Wage)
coef(summary(fit3))
```

```
##                                          Estimate   Std. Error   t value
## (Intercept)                          -1.841542e+02 6.004038e+01 -3.067172
## cbind(age, age^2, age^3, age^4)age  2.124552e+01 5.886748e+00  3.609042
## cbind(age, age^2, age^3, age^4)    -5.638593e-01 2.061083e-01 -2.735743
## cbind(age, age^2, age^3, age^4)     6.810688e-03 3.065931e-03  2.221409
## cbind(age, age^2, age^3, age^4)    -3.203830e-05 1.641359e-05 -1.951938
##                                        Pr(>|t|)
## (Intercept)                        0.0021802539
## cbind(age, age^2, age^3, age^4)age 0.0003123618
## cbind(age, age^2, age^3, age^4)    0.0062606446
## cbind(age, age^2, age^3, age^4)    0.0263977518
## cbind(age, age^2, age^3, age^4)    0.0510386498
```

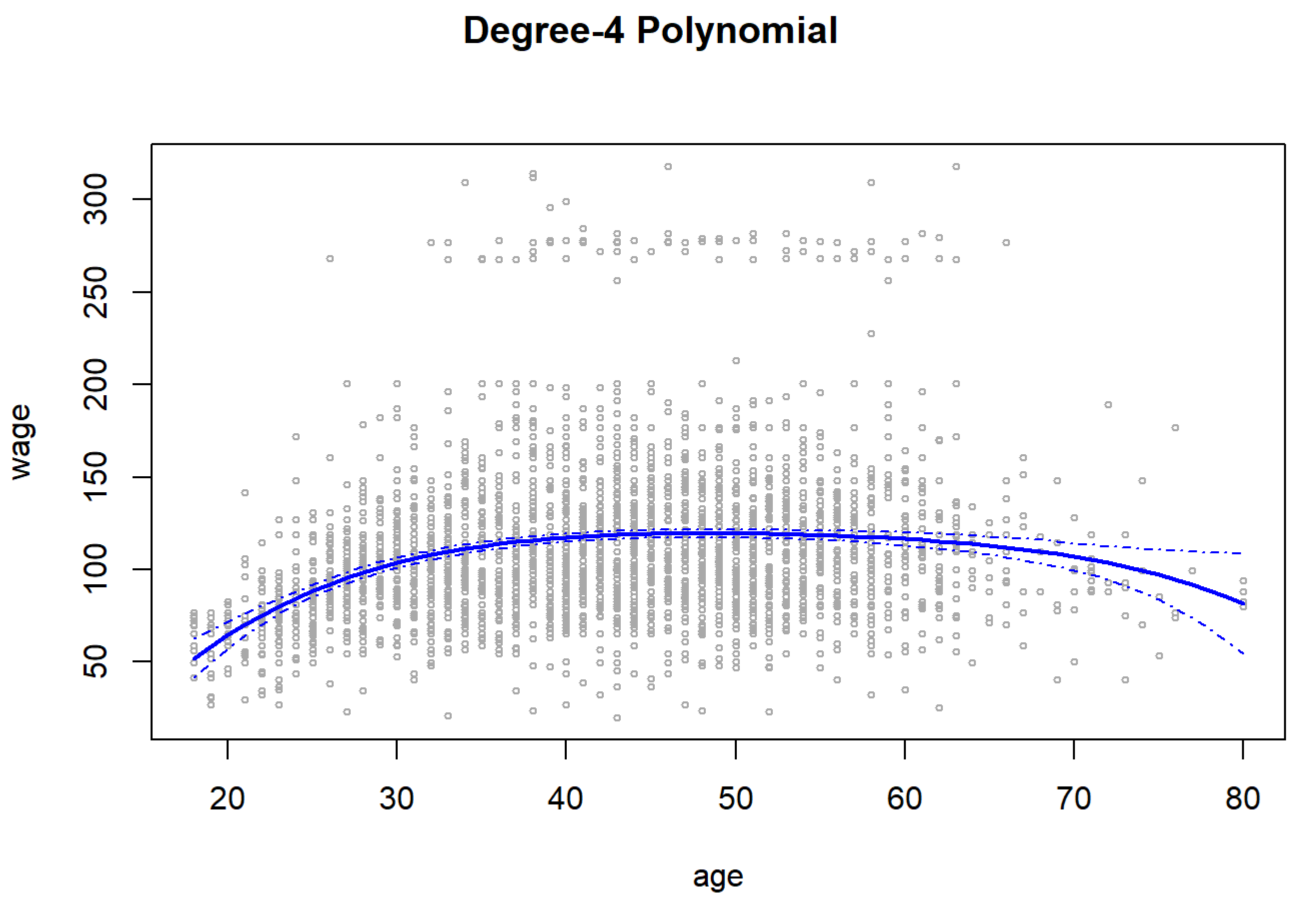
Note the difference to the following code:

```
fit4=lm(wage~age+age^2+age^3+age^4,data=Wage)
coef(summary(fit4))
```

```
##               Estimate Std. Error  t value      Pr(>|t|)
## (Intercept) 81.7047354 2.84624224 28.70618 2.543375e-160
## age          0.7072759 0.06475113 10.92299  2.900778e-27
```

Let us now fit the degree-4 polynomial to `wage` as a function of `age` in the `Wage` dataset.

```
agelims=range(age)
age.grid=seq(from=agelims[1], to=agelims[2])
preds=predict(fit, newdata=list(age=age.grid),se=TRUE)
se.bands=cbind(preds$fit+2*preds$se.fit, preds$fit-2*preds$se.fit)
par(mfrow=c(1,1),mar=c(4.5,4.5,1,1),oma=c(0,0,4,0))
plot(age,wage,xlim=agelims, cex=.5,col="darkgrey")
title("Degree-4 Polynomial", outer=T)
lines(age.grid, preds$fit, lwd=2, col="blue")
# matlines Plot the columns of one matrix against the columns of another
matlines(age.grid, se.bands, lwd=1,col="blue",lty=4)
```



Below we also illustrate one way of choosing the degree of the polynomial to use. We now fit models ranging from linear to a degree-5 polynomial and use hypothesis tests to determine the simplest model which is sufficient to explain the relationship between `wage` and `age`.

```
fit.1=lm(wage~age,data=Wage)
fit.2=lm(wage~poly(age,2),data=Wage)
fit.3=lm(wage~poly(age,3),data=Wage)
fit.4=lm(wage~poly(age,4),data=Wage)
fit.5=lm(wage~poly(age,5),data=Wage)
anova(fit.1,fit.2,fit.3,fit.4,fit.5)
```

	Res.Df <dbl>	RSS <dbl>	Df <dbl>	Sum of Sq <dbl>	F <dbl>	Pr(>F) <dbl>
1	2998	5022216	NA	NA	NA	NA
2	2997	4793430	1	228786.010	143.5931074	2.367734e-32
3	2996	4777674	1	15755.694	9.8887559	1.679213e-03
4	2995	4771604	1	6070.152	3.8098134	5.104623e-02
5	2994	4770322	1	1282.563	0.8049758	3.696820e-01

5 rows

The p-value corresponding to comparing linear $\mathtt{Model1}$ to the quadratic $\mathtt{Model2}$ is small indicating that a linear fit is not sufficient. The p-value corresponding to comparing $\mathtt{Model2}$ to $\mathtt{Model3}$ is also low indicating that $\mathtt{Model2}$ is not sufficient. The p-value corresponding to comparing $\mathtt{Model3}$ and $\mathtt{Model4}$ is greater than 0.05 indicating that there is no sufficient improvement in choosing degree-4 polynomial over the cubic polynomial. Hence the cubic polynomial appears to provide a reasonable fit to the data.

Note that as an alternative to using ANOVA, we could choose the polynomial degree using cross-validation.

Example: polynomial logistic regression

We consider a procedure for predicting whether an individual earns more than $250,000 per year.

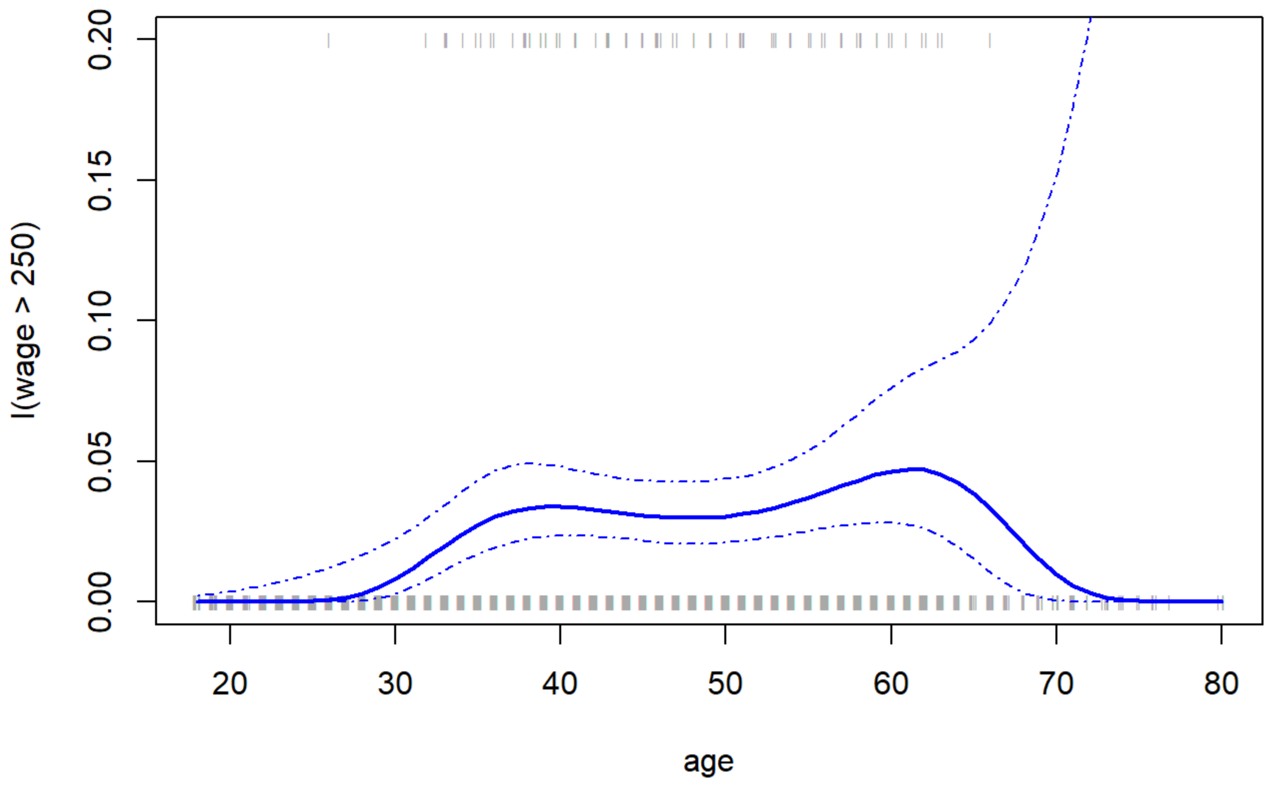
```
fit=glm(I(wage>250)~poly(age,4),data=Wage,family=binomial)

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

preds=predict(fit, newdata=list(age=age.grid),se=T)
pfit=exp(preds$fit)/(1+exp(preds$fit))
se.bands.logit=cbind(preds$fit+2*preds$se.fit,preds$fit-2*preds$se.fit)
se.bands=exp(se.bands.logit)/(1+exp(se.bands.logit))
```

In order to directly compute the probabilities we select $\mathtt{type} = "\ \mathtt{response}\ "$ in $\mathtt{predict}()$ as follows:

```
preds=predict(fit,newdata=list(age=age.grid),type="response",se=T)
```

For visualization we execute the following code:

```
plot(age, I(wage>250),xlim=agelims, type="n", ylim=c(0,0.2))
points(jitter(age), I((wage>250)/5), cex=.5,pch="|",col="darkgrey")
lines(age.grid,pfit, lwd=2, col="blue")
matlines(age.grid, se.bands, lwd=1, col="blue", lty=4)
```



Note the wide confidence intervals on the right-hand side. Although the sample size for this data is large $(3,000)$, there are only 79 high earners, which results in high variance in the estimated coefficients.

```
fit.1=glm(I(wage>250)~poly(age,1),data=Wage,family=binomial)
fit.2=glm(I(wage>250)~poly(age,2),data=Wage,family=binomial)
fit.3=glm(I(wage>250)~poly(age,3),data=Wage,family=binomial)
fit.4=glm(I(wage>250)~poly(age,4),data=Wage,family=binomial)
fit.5=glm(I(wage>250)~poly(age,5),data=Wage,family=binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
#AIC=c(fit.1$aic,fit.2$aic,fit.3$aic,fit.4$aic,fit.5$aic )
anova(fit.1,fit.2,fit.3,fit.4,fit.5, test="Chisq")
```

	Resid. Df <dbl>	Resid. Dev <dbl>	Df <dbl>	Deviance <dbl>	Pr(>Chi) <dbl>
1	2998	719.2285	NA	NA	NA
2	2997	709.0237	1	10.204766	0.001400782
3	2996	707.9215	1	1.102191	0.293785708
4	2995	701.2198	1	6.701683	0.009632195

	Resid. Df	Resid. Dev	Df	Deviance	Pr(>Chi)
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
5	2994	698.8699	1	2.349930	0.125288656

5 rows

```
anova(fit.1,fit.2,fit.4,fit.5, test="Chisq")
```

	Resid. Df	Resid. Dev	Df	Deviance	Pr(>Chi)
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	2998	719.2285	NA	NA	NA
2	2997	709.0237	1	10.204766	0.001400782
3	2995	701.2198	2	7.803874	0.020202739
4	2994	698.8699	1	2.349930	0.125288656

4 rows

```
#  https://stackoverflow.com/questions/67360883/how-to-fix-fitted-probabilities-numer
ically-0-or-1-occurred-warning-in-r
```

Example: step functions

This example illustrates how to fit a step function in R using `cut()`:

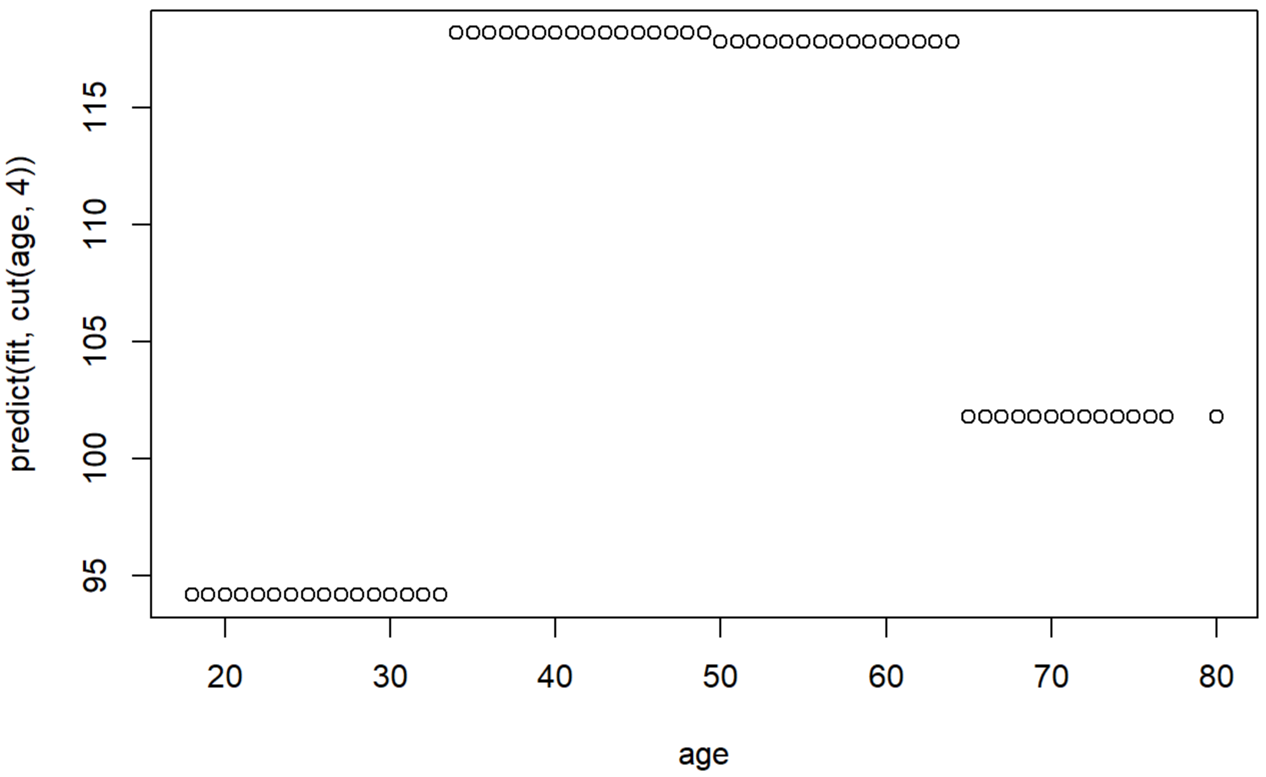
```
# cut: cut divides the range of x into intervals and codes the values in x according
to which interval they fall.
table(cut(age,breaks=4))
```

```
##
## (17.9,33.5]   (33.5,49]   (49,64.5] (64.5,80.1]
##         750        1399         779          72
```

```
fit=lm(wage~cut(age,4),data=Wage)
coef(summary(fit))
```

```
##                          Estimate Std. Error   t value     Pr(>|t|)
## (Intercept)             94.158392   1.476069 63.789970 0.000000e+00
## cut(age, 4)(33.5,49]    24.053491   1.829431 13.148074 1.982315e-38
## cut(age, 4)(49,64.5]    23.664559   2.067958 11.443444 1.040750e-29
## cut(age, 4)(64.5,80.1]   7.640592   4.987424  1.531972 1.256350e-01
```

```
plot(age,predict(fit,cut(age,4)))
```



Note that the `cut()` function returns an ordered categorical variable and `lm()` function creates a set of dummy variables for use in the regression.

Activity in R: Polynomial regression

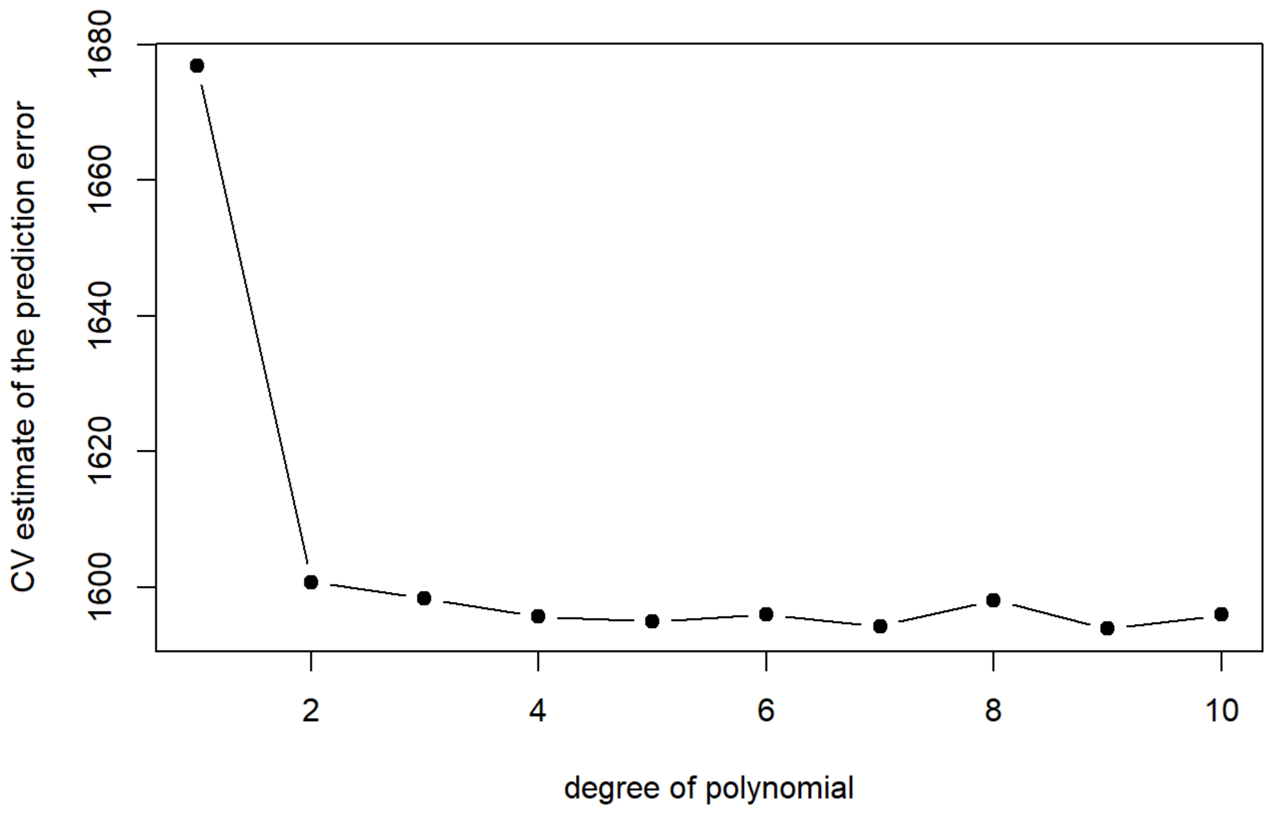
In this activity, you will further analyse the `Wage` data set considered in this section. Perform polynomial regression to predict `wage` using `age`. Use cross-validation to select the optimal degree d for the polynomial. What degree was chosen, and how does this compare to the results of hypothesis testing using ANOVA? Make a plot of the resulting polynomial fit to the data.

First we will perform polynomial regression for various polynomial degrees

```
library(boot) # Bootstrap Functions
set.seed(1)
cv.error = rep(0,10)
for( i in 1:10 ){
  glm.fit = glm( wage ~ poly(age,i), data=Wage )
  # cv.glm: This function calculates the estimated K-fold cross-validation prediction
error for generalized linear models.
  cv.error[i] = cv.glm( Wage, glm.fit, K=10 )$delta[1]
}
#delta[1]: raw cross-validation estimate of prediction error
#delta[2]: the adjusted cross-validation estimate of prediction error
```

And plot the cross-validation errors:

```
plot( 1:10, cv.error, pch=19, type='b', xlab='degree of polynomial', ylab='CV estimat
e of the prediction error' )
```



The minimal CV-error corresponds to the degree of the polynomial equal to

```
me = which.min( cv.error )
me
```

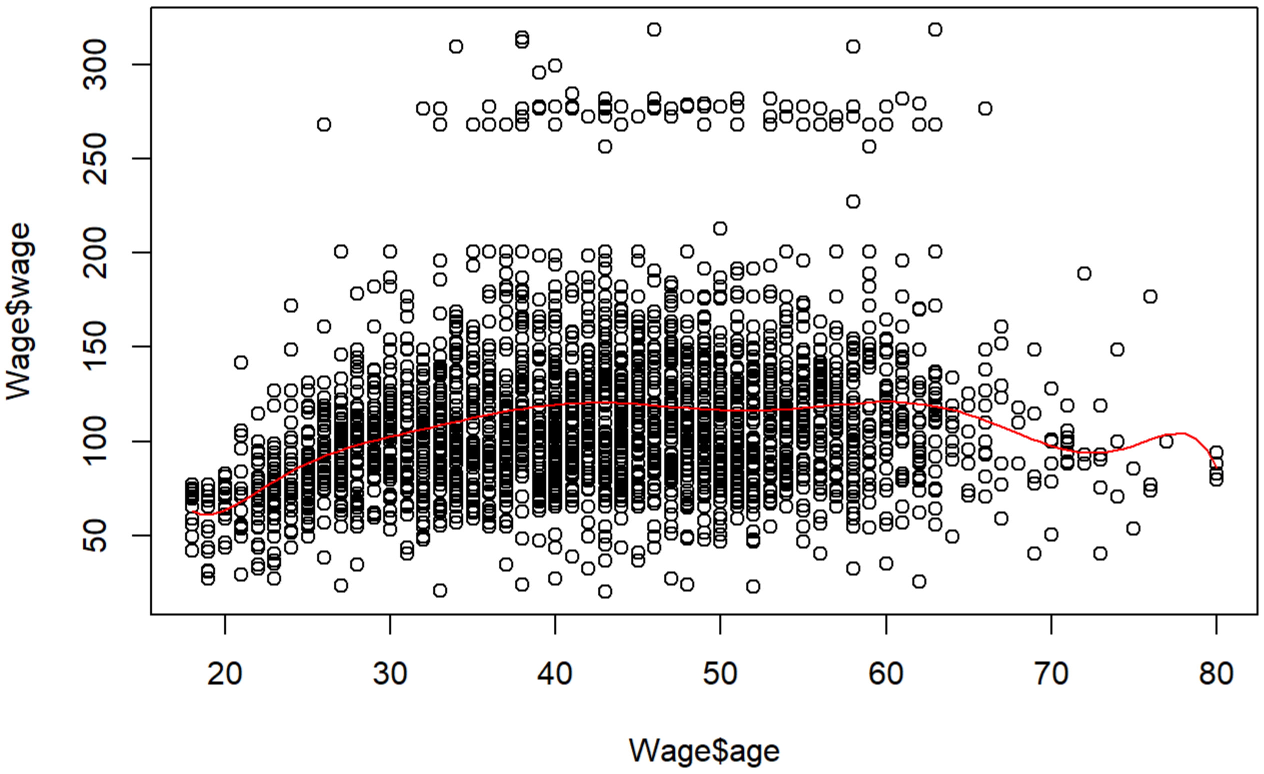
```
## [1] 9
```

Hence:

```
m = glm( wage ~ poly(age,me), data=Wage )
```

The polynomial fit plot can be obtained as follows:

```
plot( Wage$age, Wage$wage )
aRng = range(Wage$age)
a_predict = seq( from=aRng[1], to=aRng[2], length.out=100 )
w_predict = predict( m, newdata=list( age=a_predict ) )
lines( a_predict, w_predict, col='red' )
```



Section 5.3 Regression splines

{} To fit a spline we need to select the order of the spline, the number of knots and their placement.

$\mathtt{bs}(\mathtt{x}, \mathtt{df} = 7)$ in R generates a basis matrix of cubic spline functions evaluated at the N observations in $\mathtt{x}$, with the $7 - 3 = 4$ interior knots at the respective percentiles of $\mathtt{x}$.

$\mathtt{bs}(\mathtt{x}, \mathtt{degree} = 1, \mathtt{knots} = \mathtt{c}(0.2, 0.4, 0.6))$ generates a basis for linear splines, with the three interior knots.

Warning: by default the bs() function does not take the intercept into account. Therefore df = length(knots) + degree . The intercept can be included by specifying bs(…, intercept=TRUE) .

Example: A linear spline with two knots (red) and 50 knots (green):

```r
library(SemiPar)
require(splines) #Regression Spline Functions and Classes
```

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

data(lidar)
attach(lidar)
range(lidar$range)
```

```
## [1] 390 720
```

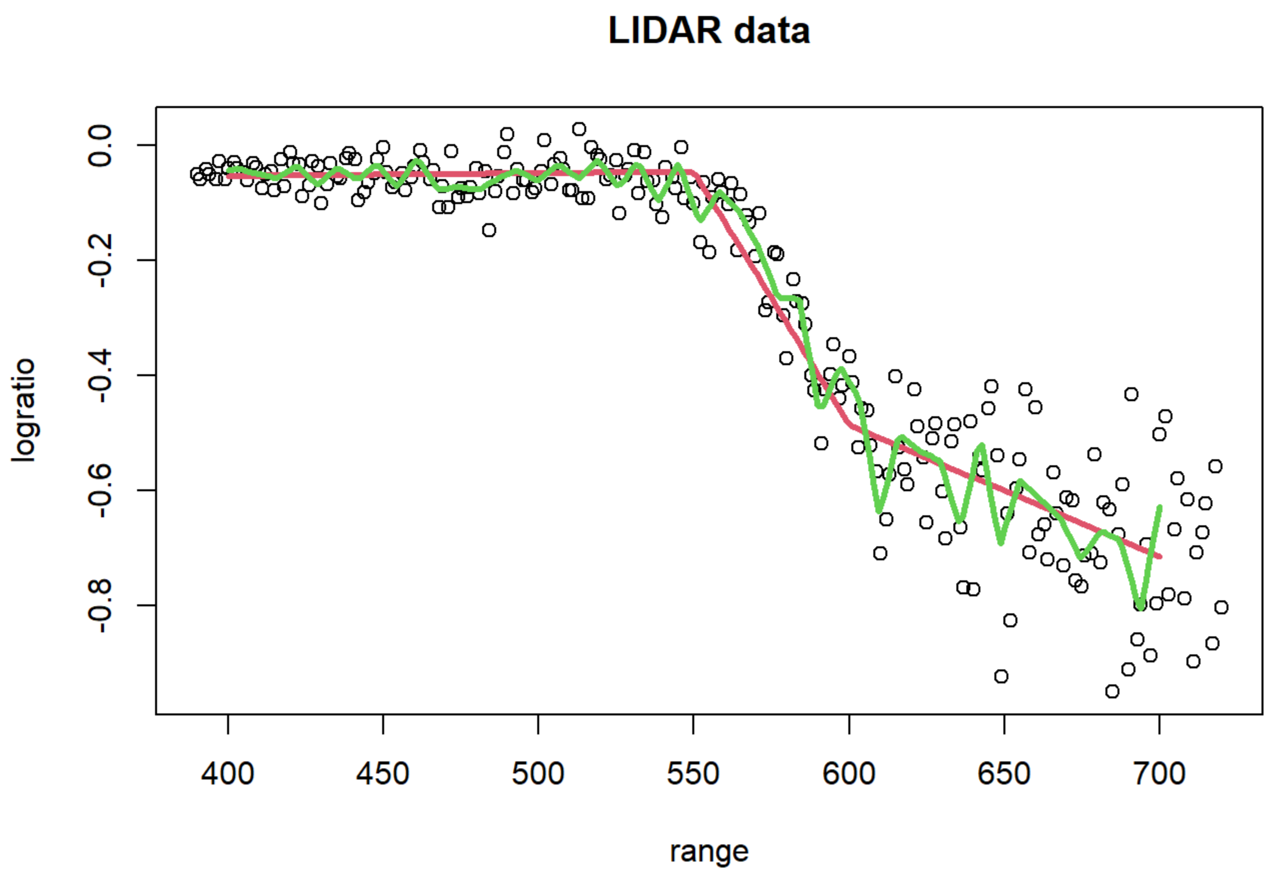
```
# bs: Generate the B-spline basis matrix for a polynomial spline. The default value f
or degree is 3.

fit <- lm(logratio ~ bs(x = range, knots = c(550,600), degree = 1))
attr(terms(fit), "predvars")
```

```
## list(logratio, bs(x = range, degree = 1L, knots = c(550, 600),
##     Boundary.knots = c(390L, 720L), intercept = FALSE))
```

```
fit2<- lm(logratio ~ bs(x = range,df = 51, degree = 1))
attr(terms(fit2), "predvars")
```

```
## list(logratio, bs(x = range, degree = 1L, knots = c(396.313725490196,
## 402.627450980392, 408.941176470588, 415.509803921569, 422.137254901961,
## 428.764705882353, 435.196078431373, 441.509803921569, 447.823529411765,
## 454.274509803922, 460.901960784314, 467.529411764706, 474.078431372549,
## 480.392156862745, 486.705882352941, 493.039215686275, 499.666666666667,
## 506.294117647059, 512.921568627451, 519.274509803922, 525.588235294118,
## 531.901960784314, 538.43137254902, 545.058823529412, 551.686274509804,
## 558.156862745098, 564.470588235294, 570.78431372549, 577.196078431372,
## 583.823529411765, 590.450980392157, 597.039215686274, 603.352941176471,
## 609.666666666667, 615.980392156863, 622.588235294118, 629.21568627451,
## 635.843137254902, 642.235294117647, 648.549019607843, 654.862745098039,
## 661.352941176471, 667.980392156863, 674.607843137255, 681.117647058824,
## 687.43137254902, 693.745098039216, 700.117647058824, 706.745098039216,
## 713.372549019608), Boundary.knots = c(390L, 720L), intercept = FALSE))
```

```
plot(lidar, main = "LIDAR data")
xpred <- seq(400,700, length.out = 200)
lines(xpred, predict(fit, data.frame(range = xpred)),col=2,lwd=3)
lines(xpred, predict(fit2, data.frame(range = xpred)),col=3,lwd=3)
```



For the LIDAR data set, the basis $1, \quad x, \quad (x-550)+, \quad (x-600)+$ was chosen, where we have defined $x_+ = \max\{x, 0\}$. The design matrix is

$$X = \begin{bmatrix} 1 & x_1 & (x_1-550)_+ & (x_1-600)_+ \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & (x_n-550)_+ & (x_n-600)_+ \end{bmatrix}.$$

Here 550 and 600 are called knots, and $(x_1 - 500)_+$ are linear spline functions. Linear combinations of linear spline functions are all piecewise linear.

```
# fit3 <- lm(logratio ~ ns(x = range, knots = c(550,600)))
# fit4<- lm(logratio ~ ns(x = range,df = 11))
# plot(lidar, main = "LIDAR data")
# xpred <- seq(400,700, length.out = 200)
# lines(xpred, predict(fit3, data.frame(range = xpred)),col=2,lwd=3)
# lines(xpred, predict(fit4, data.frame(range = xpred)),col=3,lty=2,lwd=3)
```

Example: natural spline (red) and ordinary spline (blue)

In order to fit a natural spline, we use the ns() function as follows:

```
library(ISLR)
attach(Wage)
```

```
## The following objects are masked from Wage (pos = 6):
##
##     age, education, health, health_ins, jobclass, logwage, maritl,
##     race, region, wage, year
```

```
## The following objects are masked from trade.union:
##
##     age, race, wage
```

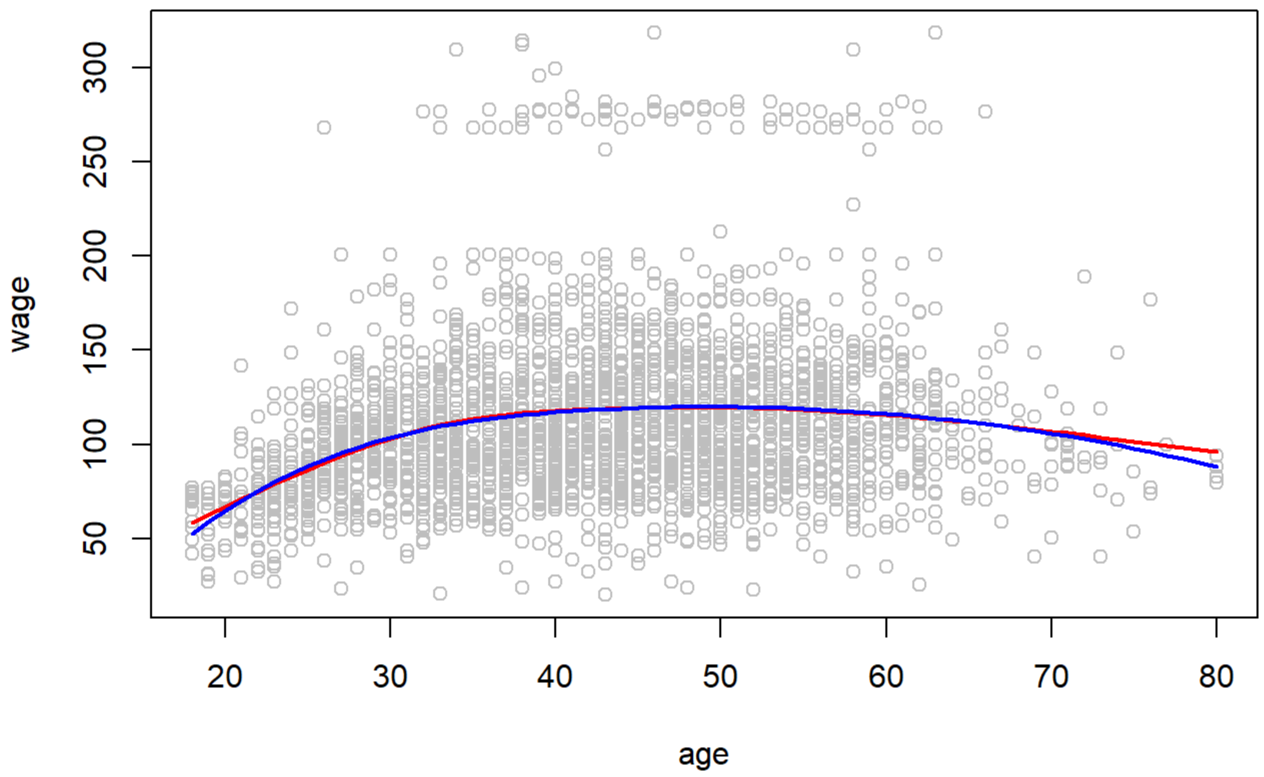
```
library(splines)
agelims=range(age)
age.grid=seq(from=agelims[1], to=agelims[2])
# ns: Generate the B-spline basis matrix for a natural cubic spline
# https://stats.stackexchange.com/questions/360993/number-of-basis-functions-in-natur
al-cubic-spline
fit=lm(wage~ns(age, df=4),data=Wage)
## To see what knots were selected
attr(terms(fit), "predvars") # attr in base library
```

```
## list(wage, ns(age, knots = c(33.75, 42, 51), Boundary.knots = c(18L,
## 80L), intercept = FALSE))
```

```
# In ns(), df - 1 - intercept=knots ---> 4-1-0=3
fit2=lm(wage~bs(age,df=4), data=Wage)
attr(terms(fit2), "predvars")
```

```
## list(wage, bs(age, degree = 3L, knots = 42, Boundary.knots = c(18L,
## 80L), intercept = FALSE))
```

```
#In bs, df - degree - intercept=knots ---> 4-3-0=1
# Boundry.knots: max and min
pred=predict(fit,newdata=list(age=age.grid),se=T)
pred2=predict(fit2,newdata=list(age=age.grid),se=T)
plot(age,wage,col="grey")
lines(age.grid,pred$fit,col="red",lwd=2)
lines(age.grid,pred2$fit,col="blue", lwd=2)
```



```
# Other examples in  https://datascienceplus.com/cubic-and-smoothing-splines-in-r/
```

Activity: Spline Regression

This question uses the variables `dis` (the weighted mean of distances to five Boston employment centers) and `nox` (nitrogen oxides concentration in parts per 10 million) from the `Boston` data. We will treat `dis` as the predictor and `nox` as the response.

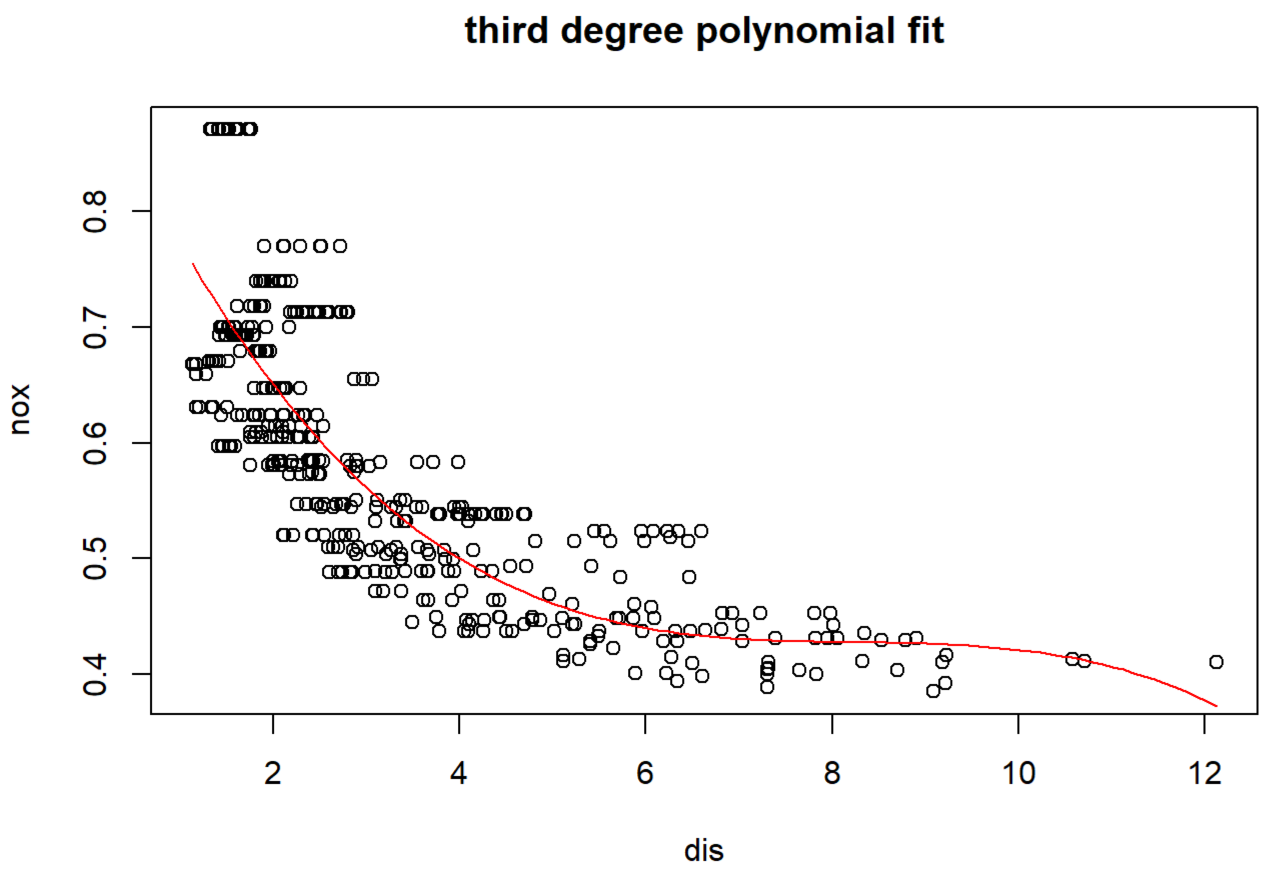
Part 1

Use the `poly()` function to fit a cubic polynomial regression to predict `nox` using `dis`. Report the regression output, and plot the resulting data and polynomial fits. Note that for this question and the following, we will consider an orthogonal polynomial basis, leaving the default argument `raw = FALSE`.

```
library(MASS)
m = lm( nox ~ poly(dis,3), data=Boston )
summary(m)
```

```
##
## Call:
## lm(formula = nox ~ poly(dis, 3), data = Boston)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.121130 -0.040619 -0.009738  0.023385  0.194904
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.554695   0.002759 201.021  < 2e-16 ***
## poly(dis, 3)1 -2.003096   0.062071 -32.271  < 2e-16 ***
## poly(dis, 3)2  0.856330   0.062071  13.796  < 2e-16 ***
## poly(dis, 3)3 -0.318049   0.062071  -5.124 4.27e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06207 on 502 degrees of freedom
## Multiple R-squared:  0.7148, Adjusted R-squared:  0.7131
## F-statistic: 419.3 on 3 and 502 DF,  p-value: < 2.2e-16
```

```
plot( Boston$dis, Boston$nox, xlab='dis', ylab='nox', main='third degree polynomial f
it' )
dis_range = range( Boston$dis )
dis_samples = seq( from=dis_range[1], to=dis_range[2], length.out=100 )
y_hat = predict( m, newdata=list( dis=dis_samples ) )

lines( dis_samples, y_hat, col='red' )
```



The data is fit quite well with this third order polynomial as the plot and the R output shows.

Part 2

Plot the polynomial fits for a range of different polynomial degrees (say, from 1 to 10), and report the associated residual sum of squares.

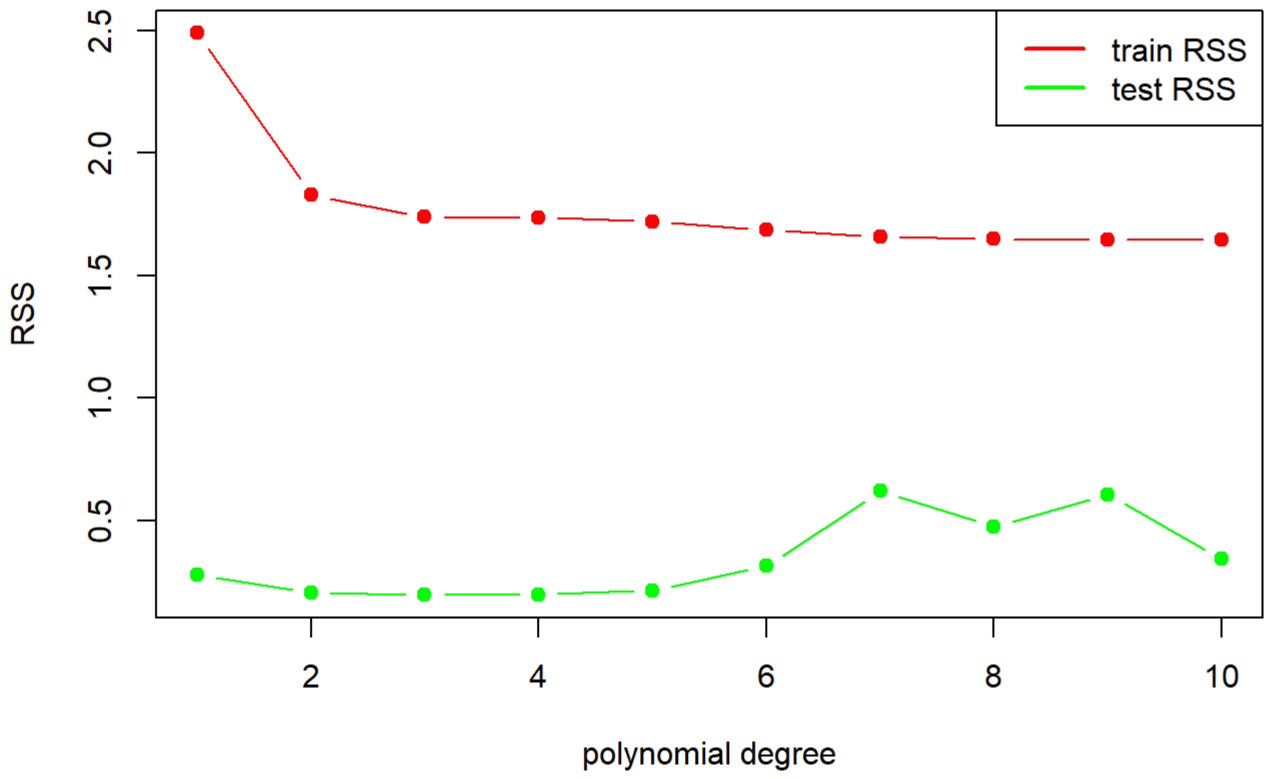
```
d_max = 10
# The training RSS:
training_rss = rep(NA,d_max)
for( d in 1:d_max ){
  m = lm( nox ~ poly(dis,d), data=Boston )
  training_rss[d] = sum( ( m$residuals )^2 )
}
training_rss
```

```
##  [1] 2.768563 2.035262 1.934107 1.932981 1.915290 1.878257 1.849484 1.835630
##  [9] 1.833331 1.832171
```

Part 3

Perform cross-validation or another approach to select the optimal degree for the polynomial, and explain your results.

```
set.seed(1)
k = 10
d_max = 10
folds = sample( 1:k, nrow(Boston), replace=TRUE )
#takes a sample from 1:k and with replacement (value 1 to k repeated sevaral times)
cv.rss.test = matrix( NA, k, d_max )
cv.rss.train = matrix( NA, k, d_max )

for( d in 1:d_max ){
  for( fi in 1:k ){ # for each fold
    fit = lm( nox ~ poly(dis,d), data=Boston[folds!=fi,] )

    y_hat = predict( fit, newdata=Boston[folds!=fi,] )
    cv.rss.train[fi,d] = sum( ( Boston[folds!=fi,]$nox - y_hat )^2 )

    y_hat = predict( fit, newdata=Boston[folds==fi,] )
    cv.rss.test[fi,d] = sum( ( Boston[folds==fi,]$nox - y_hat )^2 )
  }
}

cv.rss.train.mean = apply(cv.rss.train,2,mean)
cv.rss.train.stderr = apply(cv.rss.train,2,sd)/sqrt(k)

cv.rss.test.mean = apply(cv.rss.test,2,mean)
cv.rss.test.stderr = apply(cv.rss.test,2,sd)/sqrt(k)

min_value = min( c(cv.rss.test.mean,cv.rss.train.mean) )
max_value = max( c(cv.rss.test.mean,cv.rss.train.mean) )

plot( 1:d_max, cv.rss.train.mean, xlab='polynomial degree', ylab='RSS', col='red', pc
h=19, type='b', ylim=c(min_value,max_value) )
lines( 1:d_max, cv.rss.test.mean, col='green', pch=19, type='b' )
legend( "topright", legend=c("train RSS","test RSS"), col=c("red","green"), lty=1, lw
d=2 )
```



The training RSS steadily decreases as we add more degrees of freedom. The testing RSS decreases initially (to a low at a degree of around three) and then begins to increase again as the degree of the polynomial gets larger. Form this plot a degree three polynomial seems to be optimal.

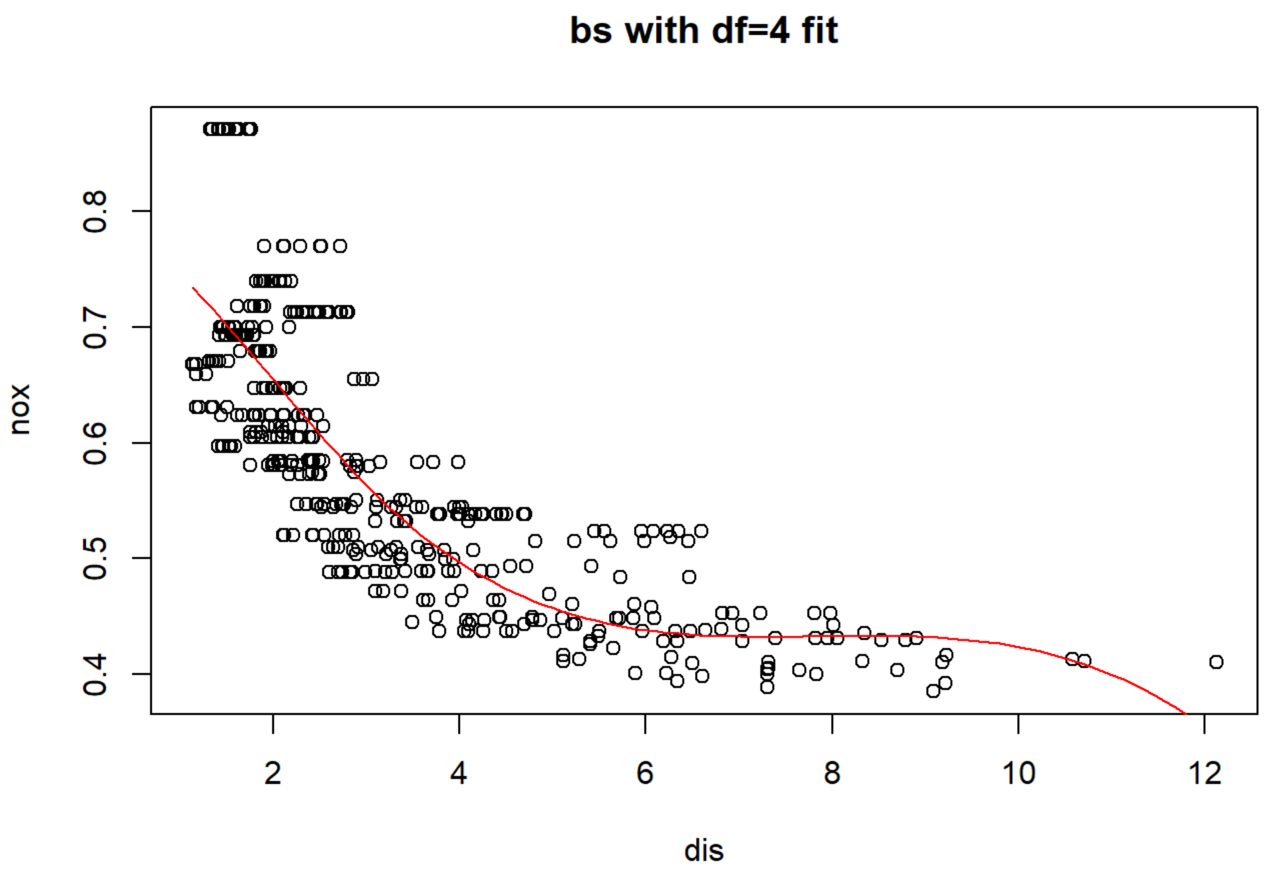
Part 4

Use the `bs()` function to fit a regression spline to predict `nox` using `dis`. How would you choose the knots? Plot the resulting fit. Report the output for the fit using four degrees of freedom.

```
library(MASS)
library(splines)
m = lm( nox ~ bs(dis,df=4), data=Boston ) # defualt degree=3
plot( Boston$dis, Boston$nox, xlab='dis', ylab='nox', main='bs with df=4 fit' )
attr(terms(m), "predvars")
```

```
## list(nox, bs(dis, degree = 3L, knots = 3.20745, Boundary.knots = c(1.1296,
## 12.1265), intercept = FALSE))
```

```
# knots=median(Boston$dis)

dis_range = range( Boston$dis )
dis_samples = seq( from=dis_range[1], to=dis_range[2], length.out=100 )
y_hat = predict( m, newdata=list( dis=dis_samples ) )
lines( dis_samples, y_hat, col='red' )
```



Part 5

Now fit a regression spline for a range of degrees of freedom, and plot the resulting fits and report the resulting RSS. Describe the results obtained.

Next, perform cross-validation or another approach to select the best degrees of freedom for a regression spline on this data. Describe your results.

```r
set.seed(1)
dof_choices = c(3:15)
n_dof_choices = length(dof_choices)

# The RSS estimated using cross-validation:

k = 5
folds = sample( 1:k, nrow(Boston), replace=TRUE )
cv.rss.test = matrix( NA, k, n_dof_choices )
cv.rss.train = matrix( NA, k, n_dof_choices )

for( di in 1:n_dof_choices ){
  for( fi in 1:k ){ # for each fold
    fit = lm( nox ~ bs(dis,df=dof_choices[di]), data=Boston[folds!=fi,] )

    y_hat = predict( fit, newdata=Boston[folds!=fi,] )
    cv.rss.train[fi,di] = sum( ( Boston[folds!=fi,]$nox - y_hat )^2 )

    y_hat = predict( fit, newdata=Boston[folds==fi,] )
    cv.rss.test[fi,di] = sum( ( Boston[folds==fi,]$nox - y_hat )^2 )
  }
}
```

```
## Warning in bs(dis, degree = 3L, knots = numeric(0), Boundary.knots = c(1.137, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = 3.2721, Boundary.knots = c(1.137, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(2.39623333333333, 4.37753333333333:
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(2.1103, 3.2721, 5.25095),
## Boundary.knots = c(1.137, : some 'x' values beyond boundary knots may cause
## ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(1.9784, 2.7006, 3.8473, 5.885), :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(1.8755, 2.39623333333333, 3.2721, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(1.8008, 2.2271, 2.78198571428571, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
## Warning in bs(dis, degree = 3L, knots = c(1.750125, 2.1103, 2.516825, 3.2721, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(1.66397777777778, 2.04665555555556, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

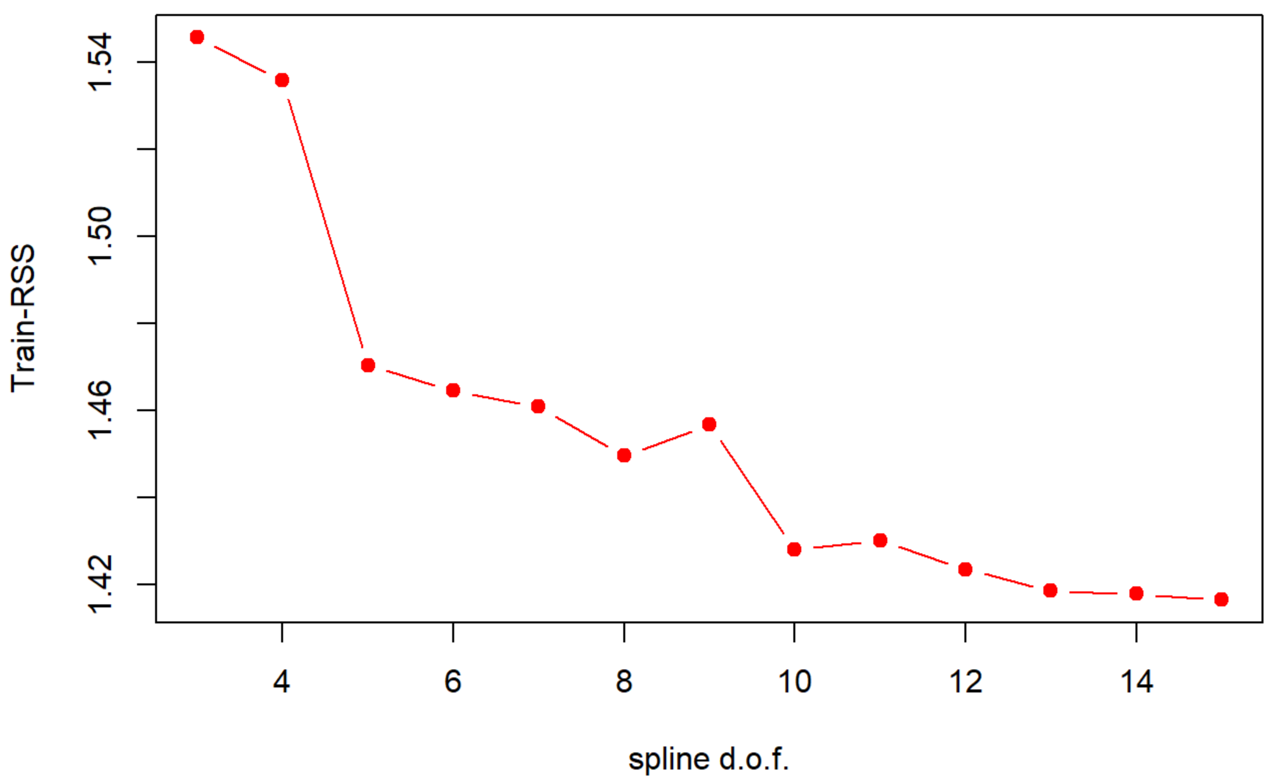
```
## Warning in bs(dis, degree = 3L, knots = c(1.6232, 1.9784, 2.2875, 2.7006, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(1.60862727272727, 1.92090909090909, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

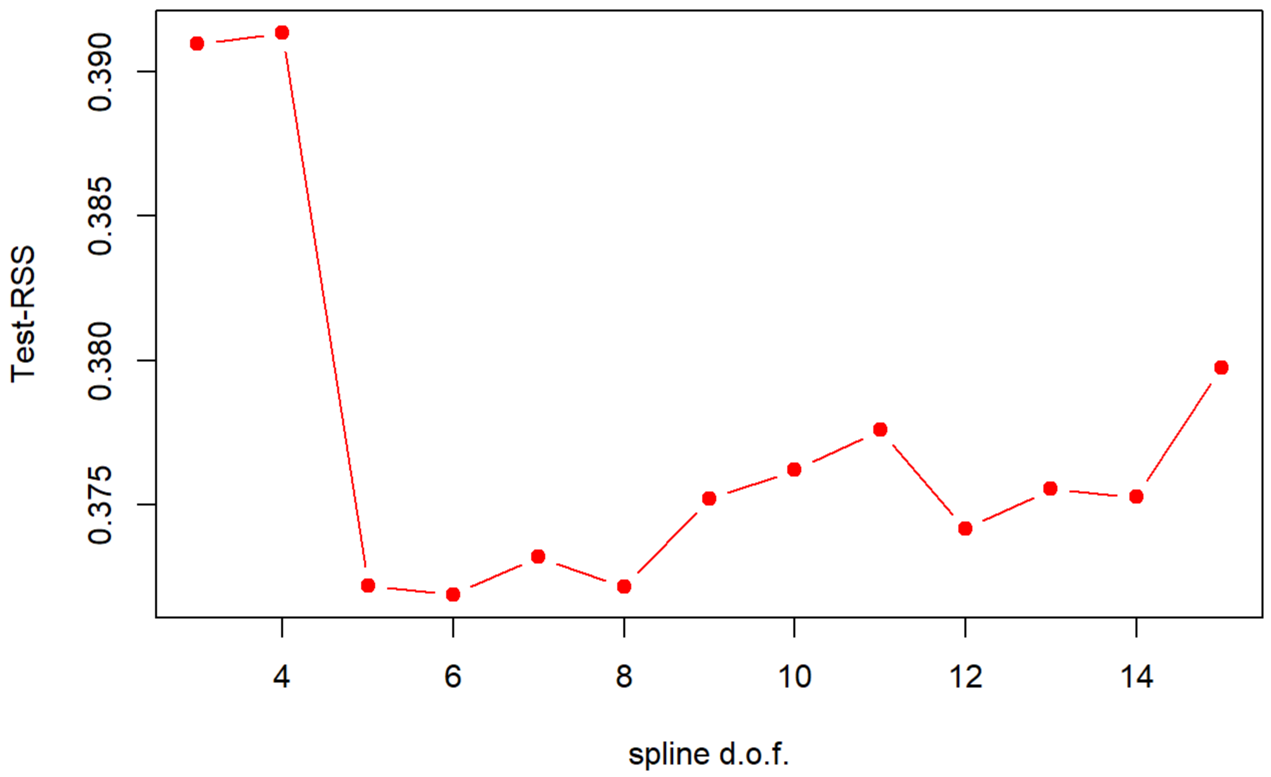
```
## Warning in bs(dis, degree = 3L, knots = c(1.58941666666667, 1.8755, 2.1103, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(1.57749230769231, 1.82353076923077, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
cv.rss.train.mean = apply(cv.rss.train,2,mean)
cv.rss.train.stderr = apply(cv.rss.train,2,sd)/sqrt(k)

cv.rss.test.mean = apply(cv.rss.test,2,mean)
cv.rss.test.stderr = apply(cv.rss.test,2,sd)/sqrt(k)

min_value = min( c(cv.rss.test.mean,cv.rss.train.mean) )
max_value = max( c(cv.rss.test.mean,cv.rss.train.mean) )

plot( dof_choices, cv.rss.train.mean, xlab='spline d.o.f.', ylab='Train-RSS', col='re
d', pch=19, type='b' )
```



```
plot( dof_choices, cv.rss.test.mean, xlab='spline d.o.f.', ylab='Test-RSS', col='re
d', pch=19, type='b' )
```



In using the`bs()` function we choose to specify the splines degree of freedom via cross-validation. This will place the knots at the percentile points of the data with a large degrees of freedom corresponding to more knots in the domain. The test RSS seems to be minimal for the degrees of freedom equal to six.

5.4 Smoothing splines

Example: Generalized Cross-Validation

```
fossil <- read.table("fossil.dat",header=T)
plot(fossil,pch=1, main="smoothed fossil data")
# smooth.spline: Fits a cubic smoothing spline to the supplied data
ss1=smooth.spline(fossil$age,fossil$strontium.ratio)
# since both spar and df are not specified, cv=FALSE
# cv (the default is FALSE): ordinary leave-one-out (TRUE) or ‘generalized’ cross-val
idation (GCV) when FALSE; is used for smoothing parameter computation only when both
spar and df are not specified
ss1$df
```

```
## [1] 13.10385
```

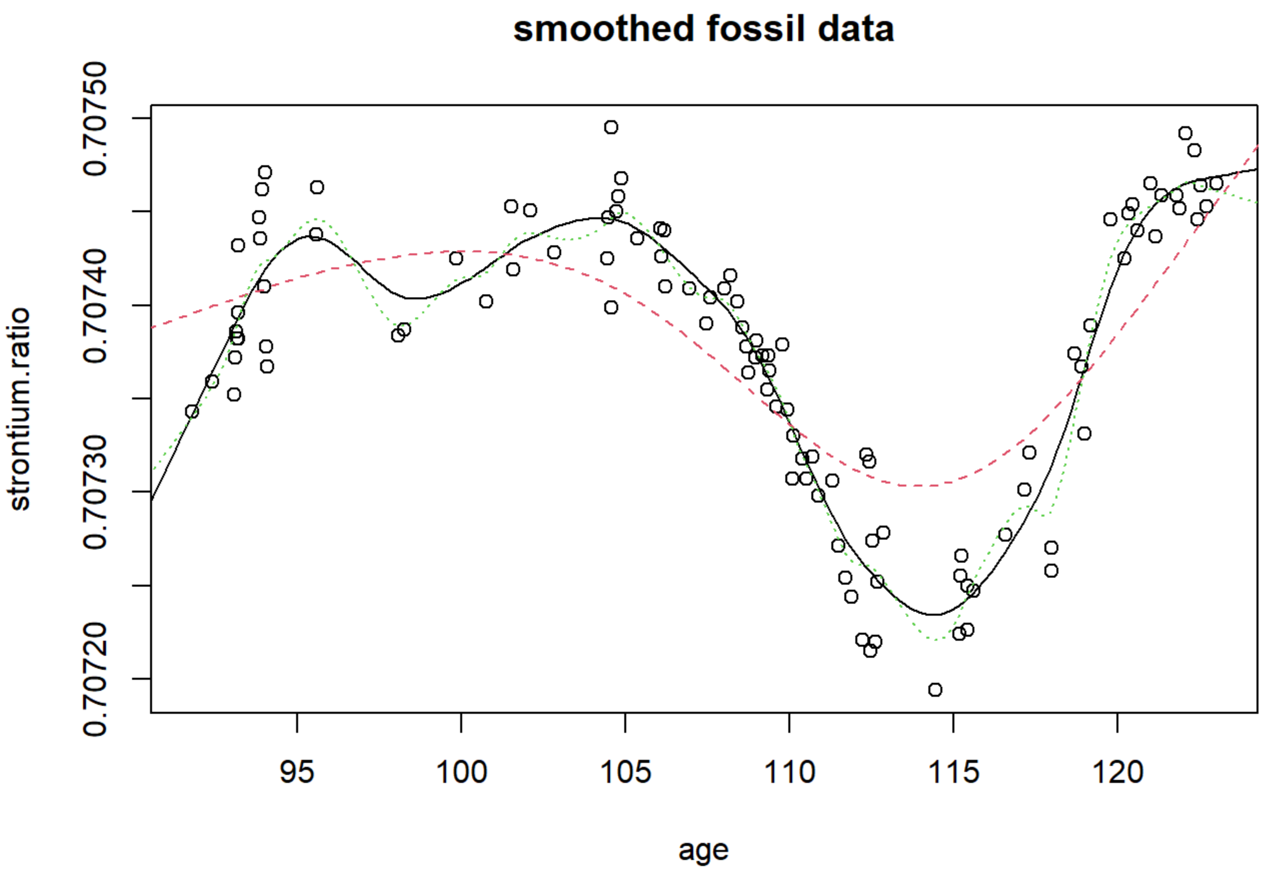
```
ss2=smooth.spline(fossil$age,fossil$strontium.ratio,df = 4)
ss2$df
```

```
## [1] 3.999547
```

```
ss3=smooth.spline(fossil$age,fossil$strontium.ratio,df = 25)
ss3$df
```

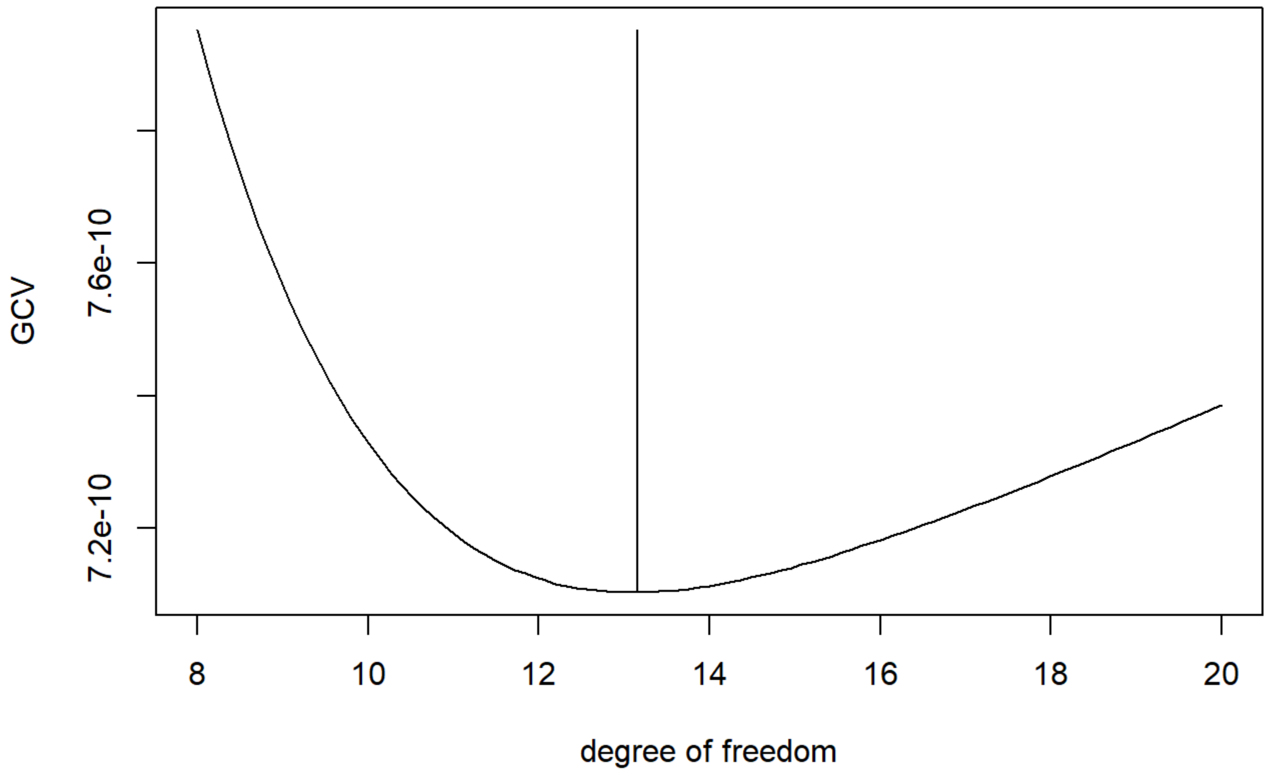
```
## [1] 24.99731
```

```
xx=seq(90,130,length.out = 200)
lines(predict(ss1,xx), col=1)
lines(predict(ss2,xx), col=2,lty=2)
lines(predict(ss3,xx), col=3,lty=3)
```



The following code shows the dependence of the GCV on the d.o.f.

```
fossil <- read.table("fossil.dat",header=T)
num.df <- 101
df.grid<-seq(8,20,length=num.df)
GCV <- rep(NA,num.df)
n <- length(fossil$age)
for (i in 1:num.df)
{
 fit    <- smooth.spline(fossil$strontium.ratio~fossil$age,df=df.grid[i])
 RSS    <- sum((fossil$strontium.ratio-predict(fit,fossil$age)$y)^2)
 GCV[i] <- (1/n)*RSS/(1-df.grid[i]/n)^2
}

plot(df.grid,GCV,type="l",xlab="degree of freedom", ylab="GCV")

ind.min <- order(GCV)[1] #44
lines(rep(df.grid[ind.min],2),c(min(GCV),max(GCV)))
```



The same result can be obtained by simply setting $\texttt{cv} = \texttt{FALSE}$ in $\texttt{smooth.spline}$ function:

```
fossil <- read.table("fossil.dat",header=T)
sp<-smooth.spline(fossil$age,fossil$strontium.ratio,cv=FALSE)
sp$df
```

```
## [1] 13.10385
```

Activity in R: Smoothing spline

Recall the Wage data set in the library ISLR. Visualise age vs wage using a scatterplot. Fit a smoothing spline using the smooth.spline() function in R. Compare two methods of fitting the smoothing spline: one of them is by providing $\mathtt{df} = 16$ and the second one by using cross-validation $\mathtt{cv} = \mathtt{TRUE}$. Add the visualisation of both fitted splines onto the scatterplot. Explain how the fit is obtained in both cases.

```
library(ISLR)
attach(Wage)
```

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

```
## The following objects are masked from Wage (pos = 8):
##
##     age, education, health, health_ins, jobclass, logwage, maritl,
##     race, region, wage, year
```

```
## The following objects are masked from trade.union:
##
##     age, race, wage
```

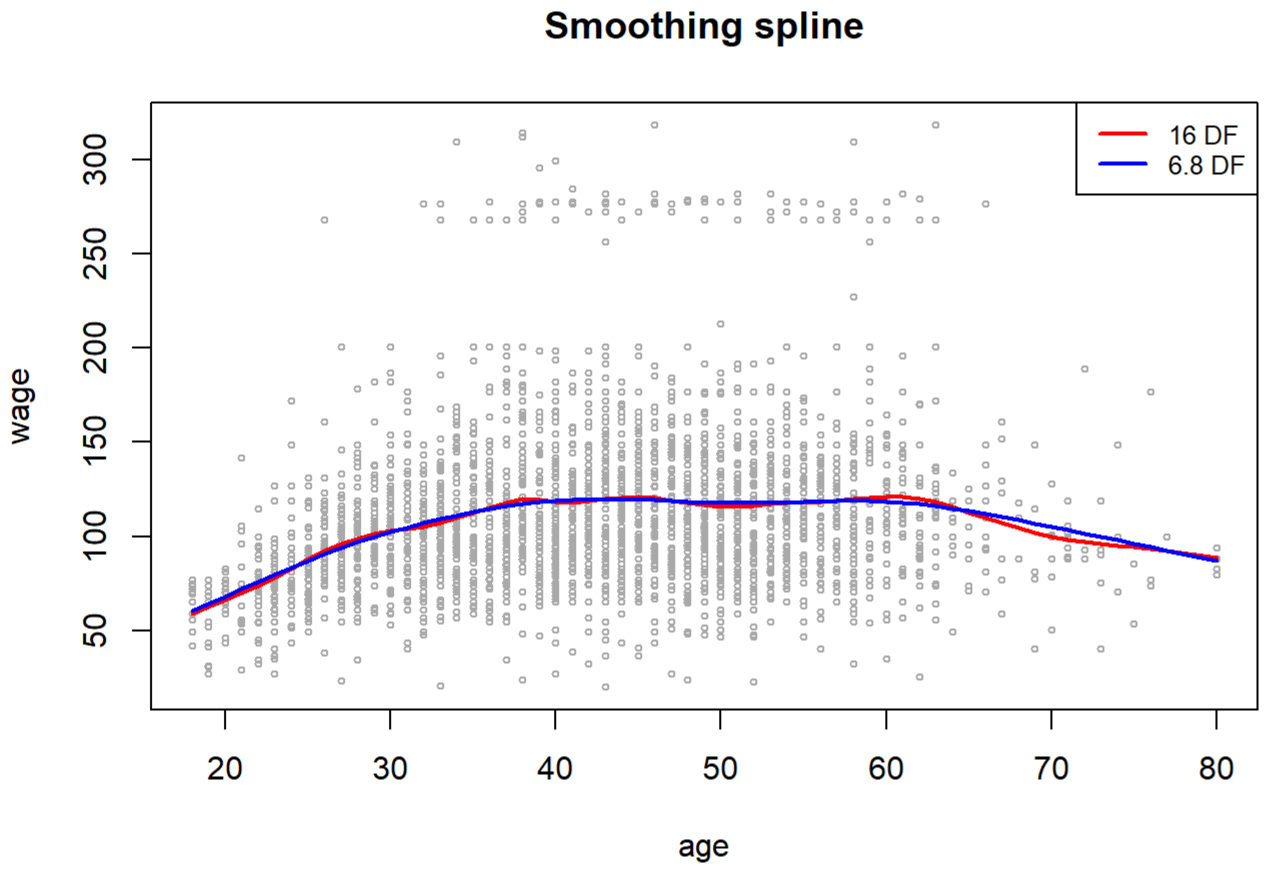
```
agelims=range(age)
plot(age,wage,xlim=agelims, cex=.5,col="darkgrey")
title("Smoothing spline")
fit=smooth.spline(age,wage,df=16)
fit2=smooth.spline(age,wage,cv=TRUE)
```

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

```
fit2$df
```

```
## [1] 6.794596
```

```
lines(fit,col="red",lwd=2)
lines(fit2,col="blue",lwd=2)
legend("topright",legend=c("16 DF","6.8 DF"),col=c("red","blue"),lty=1,lwd=2,cex=.8)
```



```
fit$lambda
```

```
## [1] 0.0006537868
```

```
fit2$lambda
```

```
## [1] 0.02792303
```

In the first call to $\texttt{smooth.spline}()$, we specified $\texttt{df} = 16$. The function then determines which λ leads to 16 degrees of freedom. In the second case, we select the smoothness level by cross-validation, and this results in a value of λ that yields $\mathrm{df} = 6.8$.

5.5 Multidimensional splines

###Example: Fitting a thin plate spline in R

A thin-plate spline fit is implemented in the package `mgcv`:

```
require(mgcv)
```

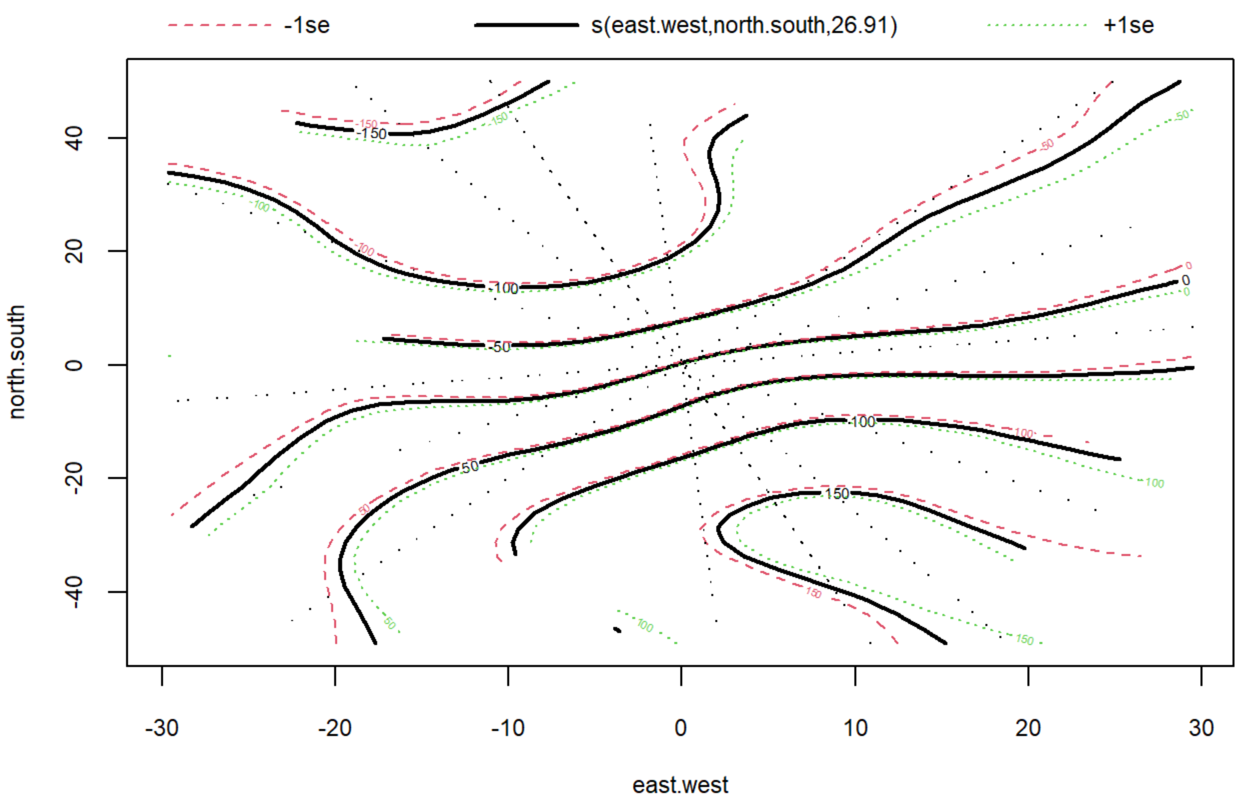
```
## Loading required package: mgcv
## Loading required package: nlme
```

```
## This is mgcv 1.9-0. For overview type 'help("mgcv-package")'.
```

```
galaxy <-read.table("galaxy.txt")

fit=gam(formula = velocity~s(east.west,north.south),data = galaxy)
fit
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## velocity ~ s(east.west, north.south)
##
## Estimated degrees of freedom:
## 26.9  total = 27.91
##
## GCV score: 191.5176
```

```
plot(fit)
```



```
## gam: Generalized additive models with integrated smoothness estimation
## s(...): Function used in definition of smooth terms within gam model formula and c
ontains a list of variables that are the covariates that this smooth is a function of

#https://stats.stackexchange.com/questions/423375/why-is-there-residual-dots-in-plot-
of-gam
```

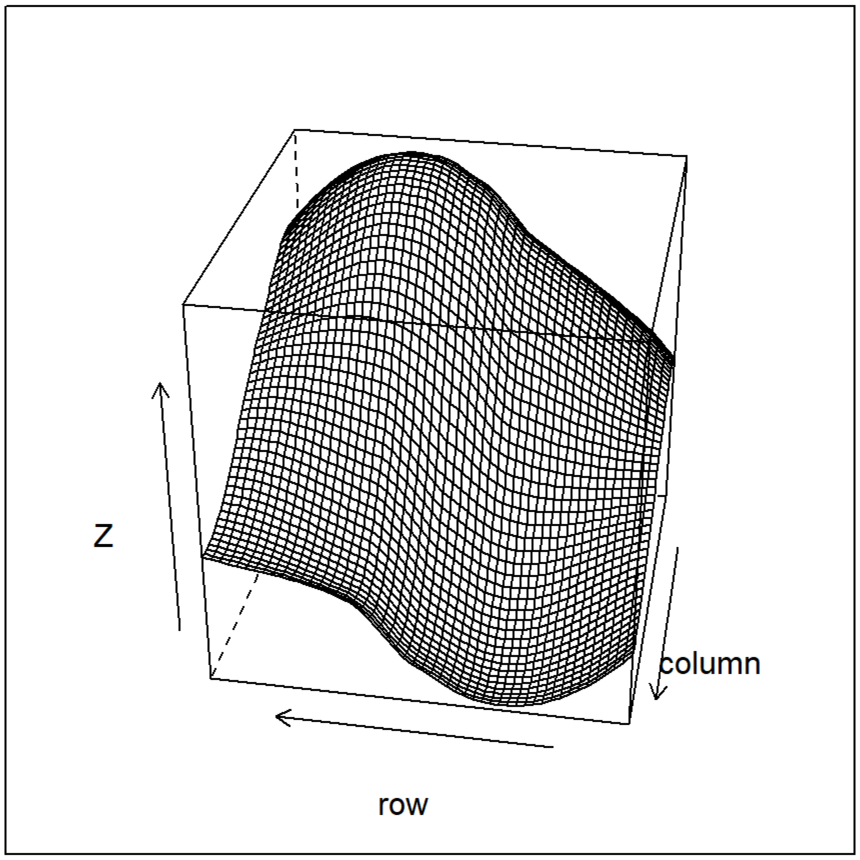
Fits in higher dimensions may be calculated as well, but are not visualised as easily. Alternatively, a loess smooth can be fitted:

```
attach(galaxy)

loc.lin.reg = loess(formula = velocity ~ north.south +
                      east.west, data = galaxy, span = 0.7, degree = 2,
                    normalize = T, family = "gaussian")

xgrid = seq(min(north.south), max(north.south), length.out = 50)
ygrid = seq(min(east.west), max(east.west), length.out = 50)

new.coords = matrix(data = NA, nrow = 2500, ncol = 2)
new.coords[, 1] = as.vector(outer(rep(1, 50), xgrid))
new.coords[, 2] = as.vector(outer(ygrid, rep(1, 50)))

new.coords = as.data.frame(new.coords)
names(new.coords) <- c("north.south", "east.west")

Z = predict(loc.lin.reg, newdata = new.coords)
Z = matrix(data = Z, ncol = 50)
library(lattice)
```

```
##
## Attaching package: 'lattice'
```

```
## The following object is masked from 'package:boot':
##
##     melanoma
```

```
wireframe(Z, screen = list(z = 170, x = -60))
```

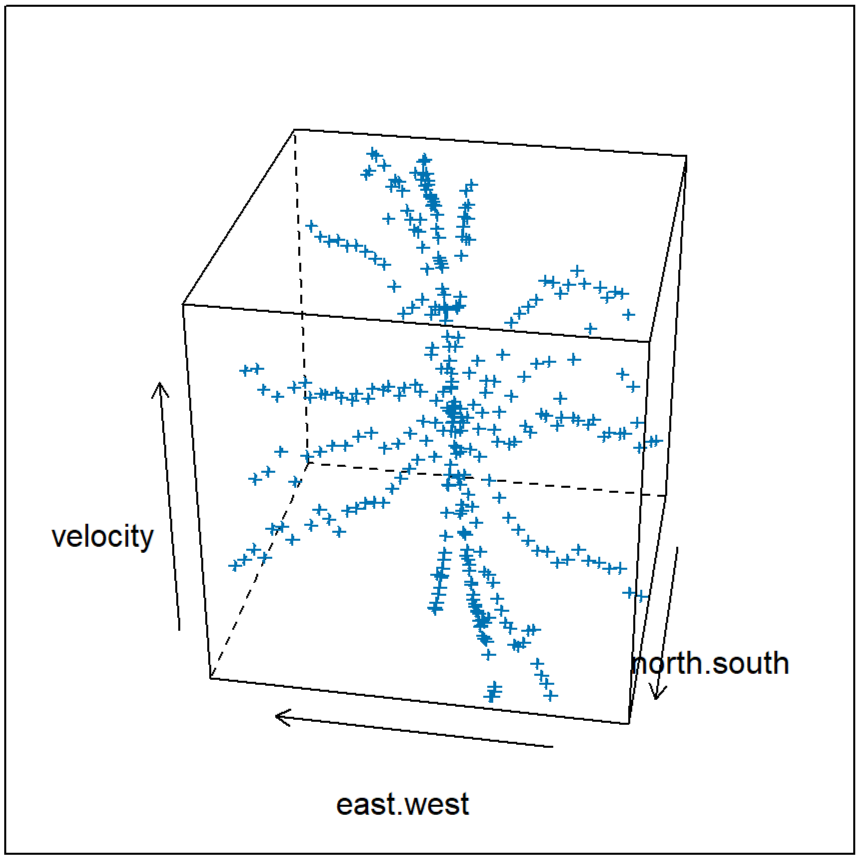


```
# wireframe: 3d Scatter Plot and Wireframe Surface Plot
```

Activity in R: 3D visualisation

Consider the `galaxy` dataset. Generate a plot of the three-dimensional point cloud `east.west`, `north.south`, `velocity` using the *cloud()* function in the `lattice` library in R.

```
require("lattice")

## Loading required package: lattice

cloud(velocity ~ east.west + north.south,data = galaxy, pretty = T,
      screen = list(z = 170, x = -60))
```



5.6 Activities

In this exercise we follow the Lab notes from Section 7.8 of James et al. (2013), i.e. the ISLR book. We will skip Section 7.8.1 on polynomial regression and step functions but it is recommended to read through this section (combined with the course notes). Instead we focus on section 7.8.2 to learn how to fit regression splines, natural splines and smoothing splines.

We will focus on the `Wage` dataset available in the `R` package `ISLR`.

- Consider a spline regression to fit `wage` to `age` considering knots at 25, 40 and 60. Represent the estimated regression line for a sequence of age values of length 100 ranging from the minimum to the maximum observed values. Add a confidence interval using ± 2 standard error.

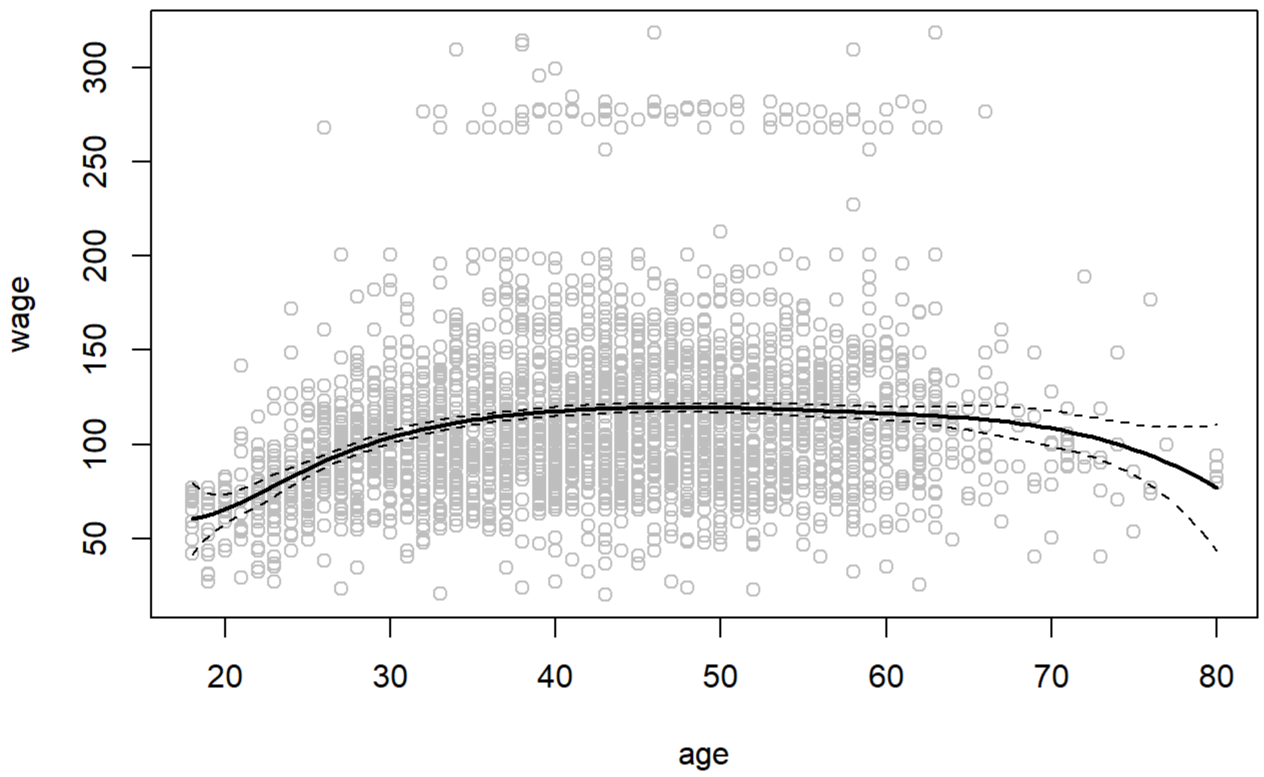
```
library(ISLR)

data("Wage")
attach(Wage)
```

```
## The following objects are masked from Wage (pos = 7):
##
##     age, education, health, health_ins, jobclass, logwage, maritl,
##     race, region, wage, year
## The following objects are masked from Wage (pos = 9):
##
##     age, education, health, health_ins, jobclass, logwage, maritl,
##     race, region, wage, year
```

```
## The following objects are masked from Wage (pos = 13):
##
##     age, education, health, health_ins, jobclass, logwage, maritl,
##     race, region, wage, year
```

```
## The following objects are masked from trade.union:
##
##     age, race, wage
```

```
fit <- lm(wage~bs(age, knots=c(25,40,60)), data=Wage)

age.grid <- seq(from=min(age), to=max(age), length=100)
pred <- predict(fit, newdata=list(age=age.grid), se=TRUE)

plot(age, wage, col="gray")
lines(age.grid, pred$fit, lwd=2)
lines(age.grid, pred$fit + 2*pred$se, lty=2)
lines(age.grid, pred$fit - 2*pred$se, lty=2)
```

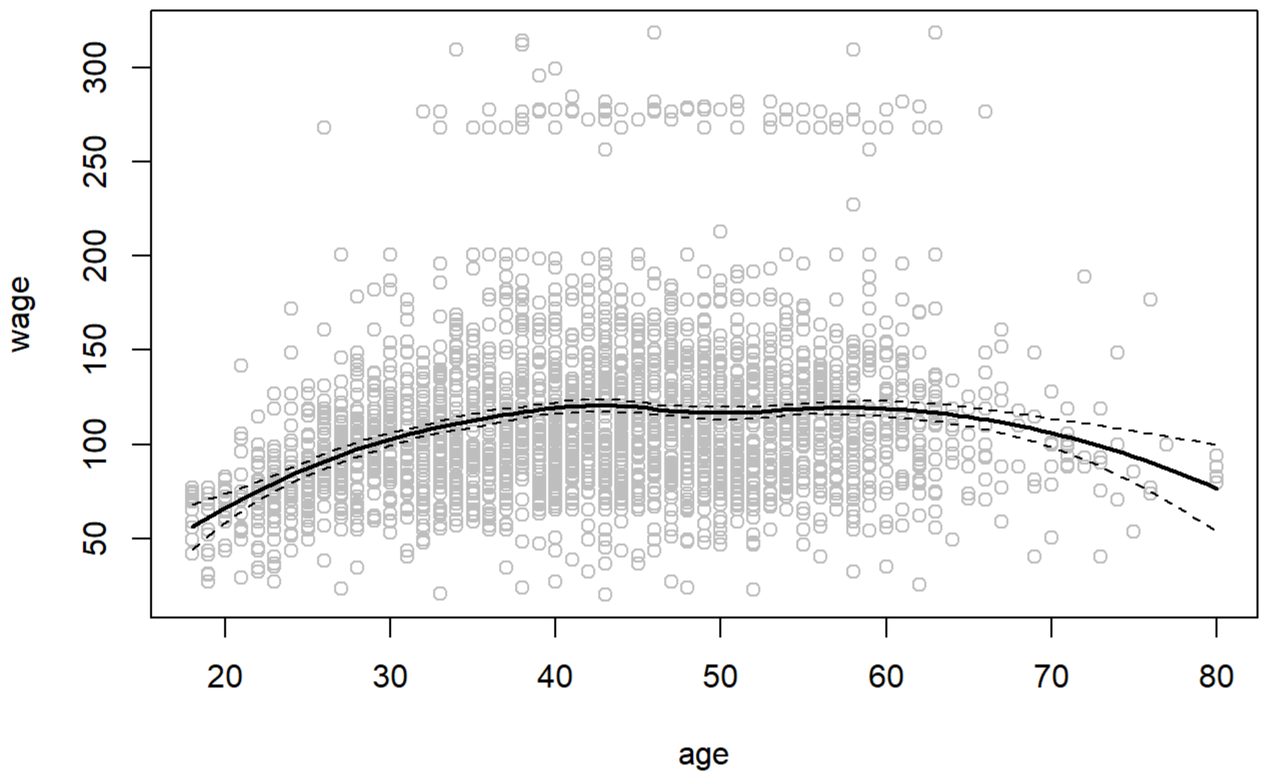


- How many basis function were used in the previous question? Now, instead of specifying the locations of the knots, use the `df` and `degree` arguments to specify a quadratic spline regression with 6 degrees of freedom. Fit the model and display it as in 1). Where were the knots located?

```
fit2 <- lm(wage~bs(age, df=6, degree=2), data=Wage)
attr(bs(age, df=6, degree=2), "knots")
```

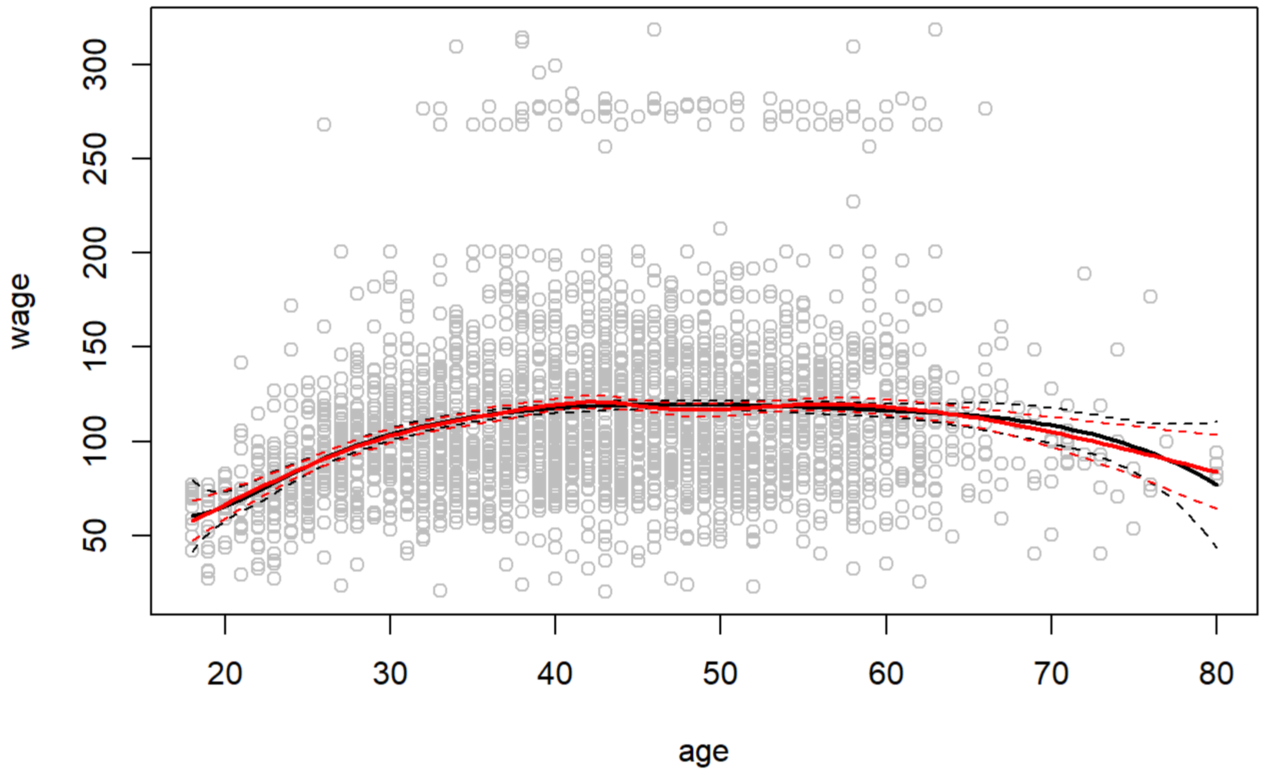
```
## [1] 32 39 46 53
```

```
# 20% 40% 60% 80%
#  32  39  46  53
pred2 <- predict(fit2, newdata=list(age=age.grid), se=TRUE)

plot(age, wage, col="gray")
lines(age.grid, pred2$fit, lwd=2)
lines(age.grid, pred2$fit + 2*pred2$se, lty=2)
lines(age.grid, pred2$fit - 2*pred2$se, lty=2)
```



- Use the ns() function to fit a natural cubic spline with 6 degree of freedom. Superimpose it with the spline regression from 1). What is the main comment that you can make when comparing both models?

```
fit3 <- lm(wage~ns(age, df=6), data=Wage)
pred3 <- predict(fit3, newdata=list(age=age.grid), se=TRUE)

plot(age, wage, col="gray")
lines(age.grid, pred$fit, lwd=2)
lines(age.grid, pred$fit + 2*pred$se, lty=2)
lines(age.grid, pred$fit - 2*pred$se, lty=2)
lines(age.grid, pred3$fit, lwd=2, col="red")
lines(age.grid, pred3$fit + 2*pred3$se, lty=2, col="red")
lines(age.grid, pred3$fit - 2*pred3$se, lty=2, col="red")
```



We observe less variability at the boundary of the support when using the natural splines.

- Use the `smooth.spline()` function to fit a smoothing spline to the data. In the first place use the argument `df` $= 16$ allowing for a 16 degrees of freedom. In a second step, consider instead the argument `cv` $=$ TRUE to use cross-validation to select the optimal degree of freedom. Display both fitted lines. What do the respective models do?

```
plot(age, wage, xlim=range(age), cex=0.5, col="darkgrey")
title("Smoothing Spline")
fit4 <- smooth.spline(age, wage, df=16)
fit5 <- smooth.spline(age, wage, cv=TRUE)
```

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

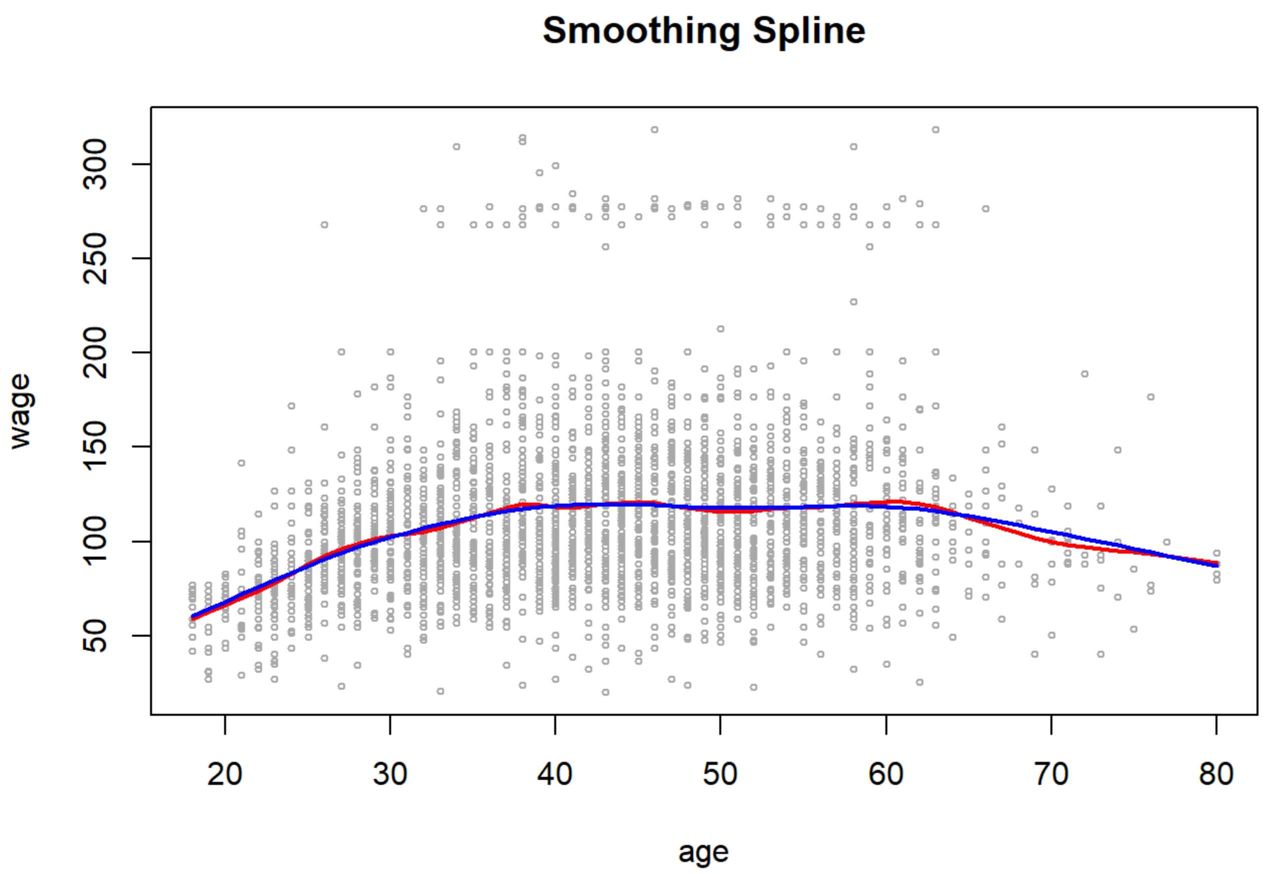
```
table(age)
```

```
## age
##  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37
##  11  14  20  15  38  45  32  56  47  53  59  58  74  63  78  87  76  75  66  77
##  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57
##  83  89 113  92  88  98  93  95  80  98  93  83  95  82  69  62  68  65  62  42
##  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77
##  57  39  37  33  30  27  11   8  13   7   4   5   6   8   3   5   3   2   3   1
##  80
##   4
```

```
fit5$df
```

```
## [1] 6.794596
```

```
lines(fit4, col="red", lwd=2)
lines(fit5, col="blue", lwd=2)
```


Smoothing Spline
wage
age