Ch.7 Exercises: Moving Beyond Linearity

Conceptual

1. (a)

$$f_1(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$$

- Where, $a_1 = \beta_0$, $b_1 = \beta_1$, $c_1 = \beta_2$, $d_1 = \beta_3$

(b)

$$\begin{aligned} f_2(x) &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x-\xi)^3 \\ &= \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x^3 - 3x^2\xi + 3x\xi^2 - \xi^3) \\ &= (\beta_0 - \beta_4\xi^3) + x(\beta_1 + 3\beta_4\xi^2) + x^2(\beta_2 - 3\beta_4\xi) + x^3(\beta_3 + \beta_4) \end{aligned}$$

- Where, $a_2 = \beta_0 - \beta_4\xi^3$, $b_2 = \beta_1 + 3\beta_4\xi^2$, $c_2 = \beta_2 - 3\beta_4\xi$, $d_2 = \beta_3 + \beta_4$

(c)

$$\begin{aligned} f_1(\xi) &= \beta_0 + \beta_1\xi + \beta_2\xi^2 + \beta_3\xi^3 \\ f_2(\xi) &= (\beta_0 - \beta_4\xi^3) + \xi(\beta_1 + 3\beta_4\xi^2) + \xi^2(\beta_2 - 3\beta_4\xi) + \xi^3(\beta_3 + \beta_4) \\ &= \beta_0 - \beta_4\xi^3 + \beta_1\xi + 3\beta_4\xi^3 + \beta_2\xi^2 - 3\beta_4\xi^3 + \beta_3\xi^3 + \beta_4\xi^3 \\ &= \beta_0 + \beta_1\xi + \beta_2\xi^2 + \beta_3\xi^3 = f_1(\xi) \end{aligned}$$

(d)

$$f_1'(\xi) = \beta_1 + 2\beta_2\xi + 3\beta_3\xi^2$$

$$\begin{aligned} f_2'(\xi) &= (\beta_1 + 3\beta_4\xi^2) + 2\xi(\beta_2 - 3\beta_4\xi) + 3\xi^2(\beta_3 + \beta_4) \\ &= \beta_1 + 3\beta_4\xi^2 + 2\beta_2\xi - 6\beta_4\xi^2 + 3\beta_3\xi^2 + 3\beta_4\xi^2 \\ &= \beta_1 + 2\beta_2\xi + 3\beta_3\xi^2 = f_1'(\xi) \end{aligned}$$

(e)

$$f_1''(\xi) = 2\beta_2 + 6\beta_3\xi$$

$$\begin{aligned} f_2''(\xi) &= 2(\beta_2 - 3\beta_4\xi) + 6\xi(\beta_3 + \beta_4) \\ &= 2\beta_2 - 6\beta_4\xi + 6\beta_3\xi + 6\beta_4\xi \\ &= 2\beta_2 + 3\beta_3\xi^2 = f_1''(\xi) \end{aligned}$$

2.

- In general, when $\lambda = \infty$, the penalty term is so large that it forces a function g chosen to minimize the RSS into being perfectly smooth. This is because the penalty term reduces the variability in g.

(a)

- When $\lambda = \infty$, $g = 0$. So, $\hat{g} = 0$.

(b)

- When $\lambda = \infty$, $g' = 0$ (slope=0). So, $\hat{g} = constant$(say a horizontal line).

(c)

- When $\lambda = \infty$, $g'' = 0$ (the change in slope=0).So, $\hat{g}$ must be a straight line with a slope, say g_hat= cx + d.

(d)

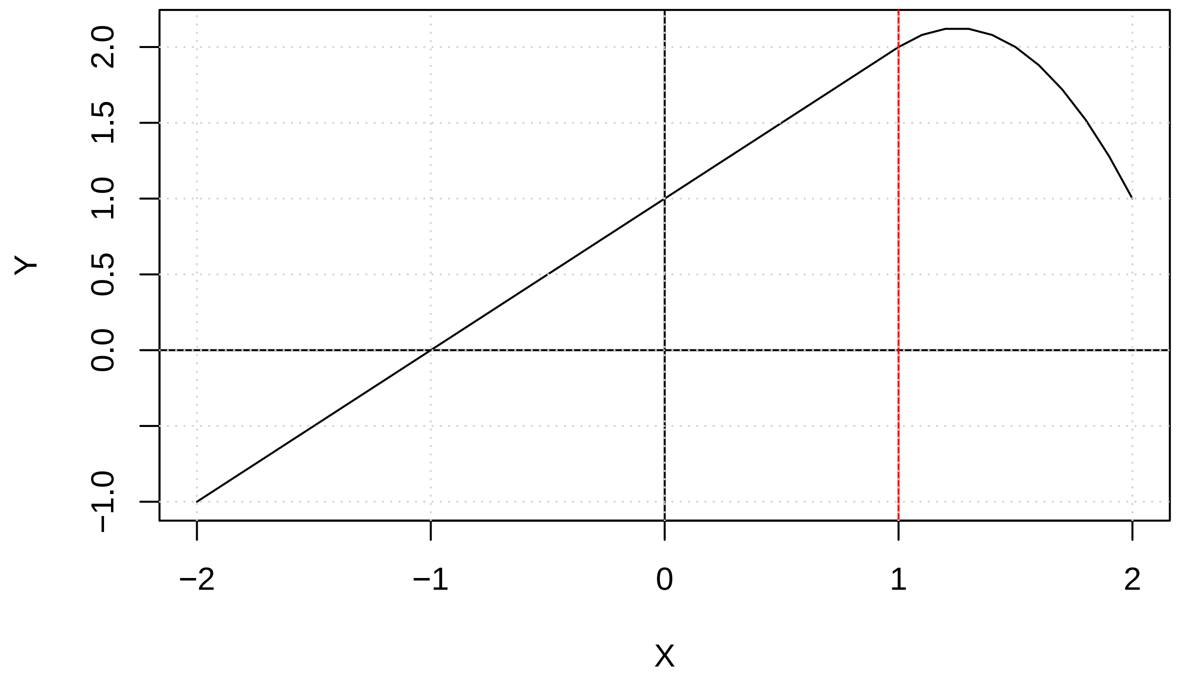
- When $\lambda = \infty$, $g''' = 0$(change in second derivative=0). So, $\hat{g}$ must be a quadratic curve, say g_hat= cx^2 + dx + e.

(e)

- When $\lambda = 0$ the penalty term has no effect, so we get a curve that interpolates all the n points perfectly (RSS Train = 0).

3.

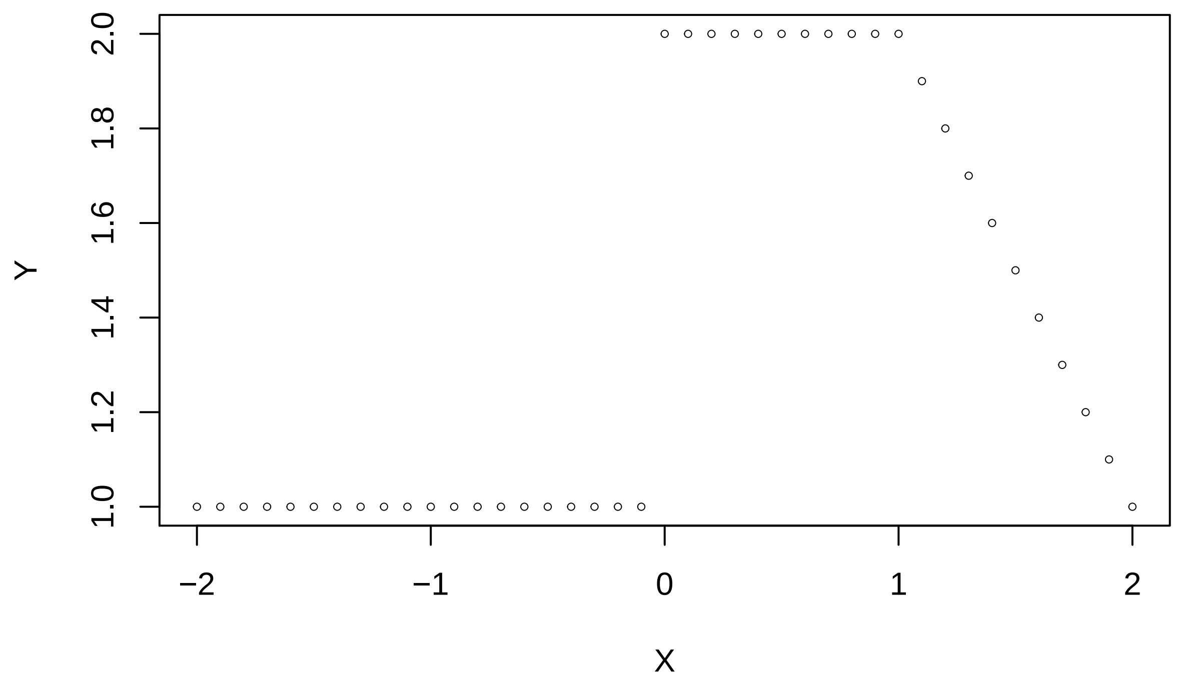
```
X = seq(-2,2,0.1)
Y = rep(NA,length(X))

for (i in 1:length(X)){
  if (X[i]<1){
    Y[i] = 1 + 1*X[i]
  }
  else{
    Y[i] = 1 + 1*X[i] - 2*(X[i]-1)^2
  }
}

plot(X,Y,type='l')
abline(h=0);abline(v=0);abline(v = 1, col = "red")
grid()
```



- The curve is linear when $-2 < X \leqslant 1$, this portion has a slope and y intercept of 1. The curve then takes a quadratic shape when $1 < X \leqslant 2$.

4.

```
for (i in 1:length(X)){
 a = if (X[i]>=0 & X[i]<=2) 1 else 0
 b = if (X[i]>=1 & X[i]<=2) 1 else 0

 Y[i] = 1 + (a-(X[i]-1)*b)
}

plot(X,Y,type = 'p', pch=1, lwd = 0.5, cex = 0.5)
```



- The chart consists of straight lines and a linear section with a slope of -2.

5.

(a)

- $\hat{g}_2$ is more flexible due to the higher order of the penalty term than $\hat{g}_1$, so it will likely have a lower training RSS.

(b)

- This depends on the shape of the underlying function for the dataset used. Generally, $\hat{g}_1$ will perform better on less flexible functions, and $\hat{g}_2$ will perform better on more flexible functions.

(c)

- The penalty terms will be zero for both equations, so training and test terms will be equal.

Applied

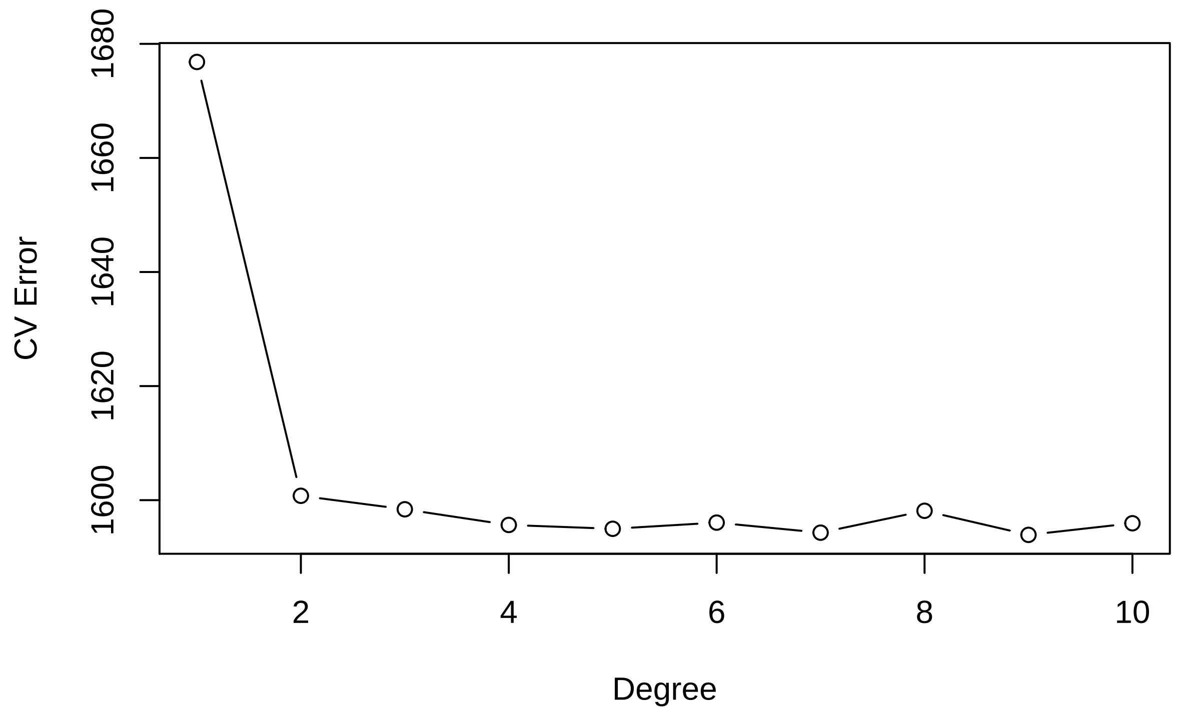
```r
library(ISLR)
library(boot)
library(splines)
library(MASS)
library(leaps)
library(gam)

require(caTools)

attach(Wage)
attach(Auto)
```

6. (a)

```r
# Cross validation to choose degree of polynomial.
set.seed(1)
cv.error.10 = rep(0,10)
for (i in 1:10) {
  glm.fit=glm(wage poly(age,i),data=Wage)
  cv.error.10[i]=cv.glm(Wage,glm.fit,K=10)$delta[1]
}
cv.error.10
```

```
##  [1] 1676.826 1600.763 1598.399 1595.651 1594.977 1596.061 1594.298 1598.134
##  [9] 1593.913 1595.950
```

```r
plot(cv.error.10, type="b", xlab="Degree", ylab="CV Error")
```



- The CV errors does not show clear improvement after degree 4 polynomial.

```r
lm.fit = glm(wage poly(age,4),data=Wage)
summary(lm.fit)
```

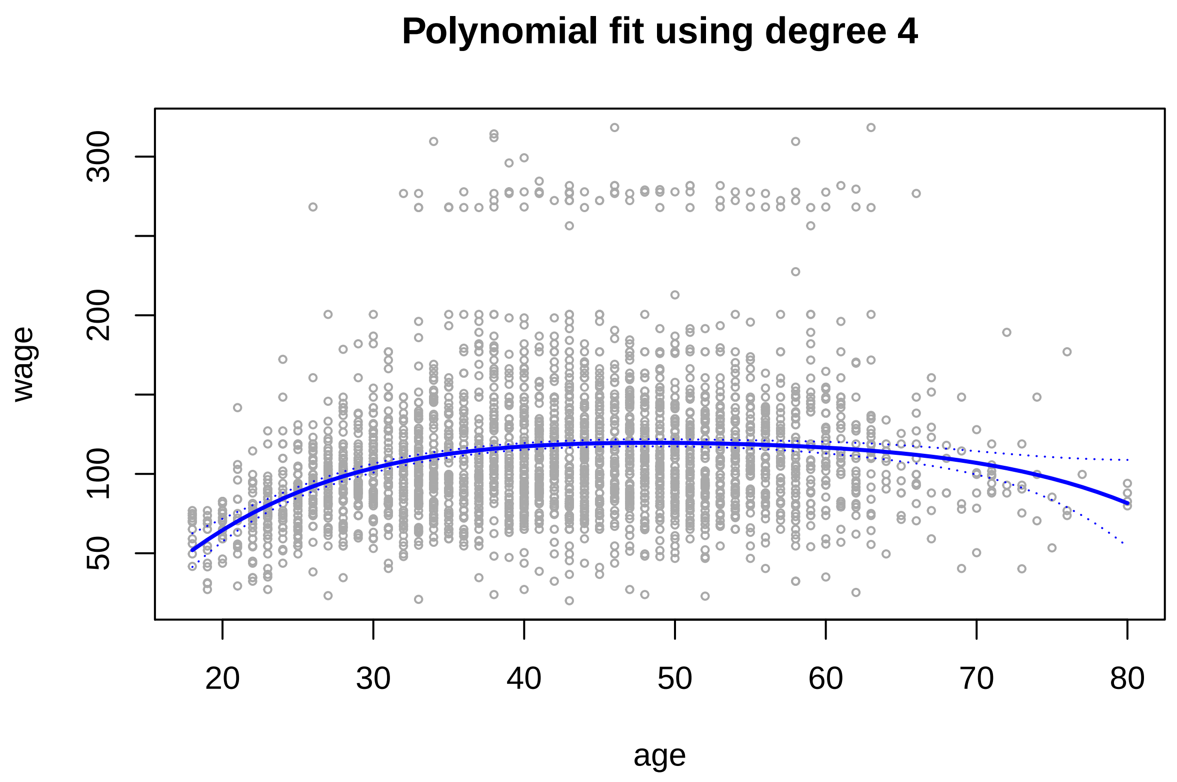
```
##
## Call:
## glm(formula = wage ~ poly(age, 4), data = Wage)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -98.707  -24.626   -4.993   15.217  203.693
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    111.7036     0.7287 153.283  < 2e-16 ***
## poly(age, 4)1  447.0679    39.9148  11.201  < 2e-16 ***
## poly(age, 4)2 -478.3158    39.9148 -11.983  < 2e-16 ***
## poly(age, 4)3  125.5217    39.9148   3.145  0.00168 **
## poly(age, 4)4  -77.9112    39.9148  -1.952  0.05104 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1593.19)
##
##     Null deviance: 5222086  on 2999  degrees of freedom
## Residual deviance: 4771604  on 2995  degrees of freedom
## AIC: 30641
##
## Number of Fisher Scoring iterations: 2
```

```r
# Using Anova() to compare degree 4 model with others.
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)
```

```
## Analysis of Variance Table
##
## Model 1: wage ~ age
## Model 2: wage ~ poly(age, 2)
## Model 3: wage ~ poly(age, 3)
## Model 4: wage ~ poly(age, 4)
## Model 5: wage ~ poly(age, 5)
##   Res.Df     RSS Df Sum of Sq        F    Pr(>F)
## 1   2998 5022216
## 2   2997 4793430  1    228786 143.5931 < 2.2e-16 ***
## 3   2996 4777674  1     15756   9.8888  0.001679 **
## 4   2995 4771604  1      6070   3.8098  0.051046 .
## 5   2994 4770322  1      1283   0.8050  0.369682
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- The p-values comparing lower order models up to degree 3 are statistically significant.The p-value comparing model 3 to 4 is slightly above 5%, whereas that comparing 4 to 5 is not statistically significant. The results show a cubic or quartic model as providing the best fit, with higher or lower order polynomials being unjustified.
- The results match that of polynomial regression using 4 degrees.

```r
# Grid of values for age at which we want predictions.
agelims=range(age)
age.grid=seq(from=agelims[1],to=agelims[2])

# Predictions.
preds=predict(lm.fit,newdata=list(age=age.grid),se=TRUE)
se.bands=cbind(preds$fit+2*preds$se.fit,preds$fit-2*preds$se.fit)

# Plot of polynomial fit onto data including SE bands.
plot(age,wage,xlim=agelims,cex=.5,col="darkgrey")
title("Polynomial fit using degree 4")
lines(age.grid,preds$fit,lwd=2,col="blue")
matlines(age.grid,se.bands,lwd =1,col="blue",lty =3)
```

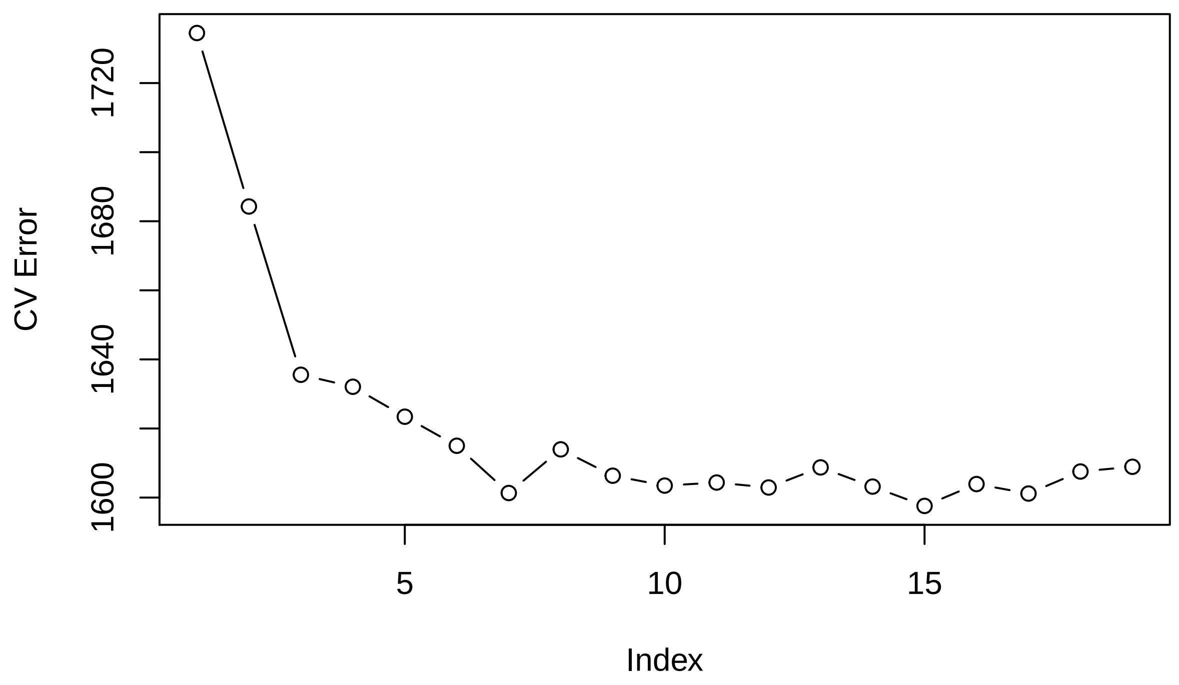


(b)

```r
# Cross validation to choose optimal number of cuts.
set.seed(1)
cv.error.20 = rep(NA,19)

for (i in 2:20) {
  Wage$age.cut = cut(Wage$age,i)
  step.fit=glm(wage age.cut,data=Wage)
  cv.error.20[i-1]=cv.glm(Wage,step.fit,K=10)$delta[1] # [1]: Std [2]: Bias corrected.
}
cv.error.20
```

```
##  [1] 1734.489 1684.271 1635.552 1632.080 1623.415 1614.996 1601.318 1613.954
##  [9] 1606.331 1603.465 1604.349 1602.915 1608.731 1603.178 1597.583 1603.909
## [17] 1601.161 1607.540 1608.915
```

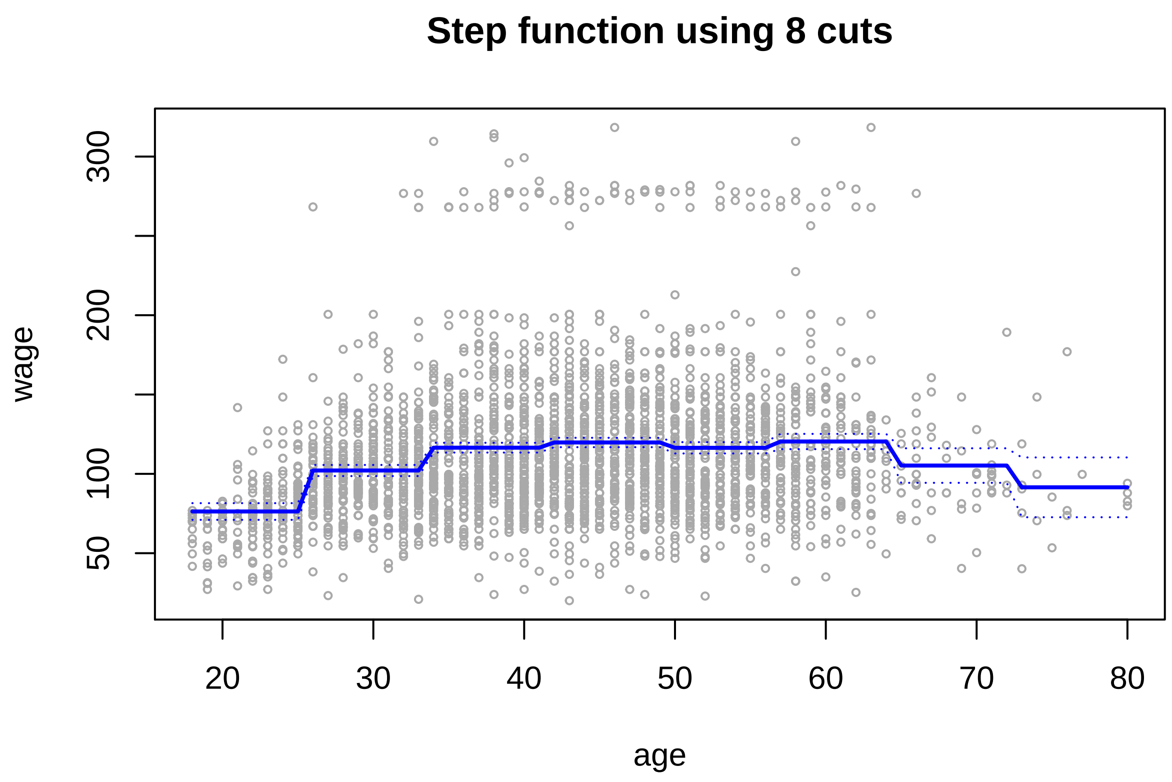
```r
plot(cv.error.20,type='b',ylab="CV Error")
```



- The data and chart shows that the CV errors do not improve substantially after 8 (Index+1) cuts, and so 8 cuts will be used to fit the step function.

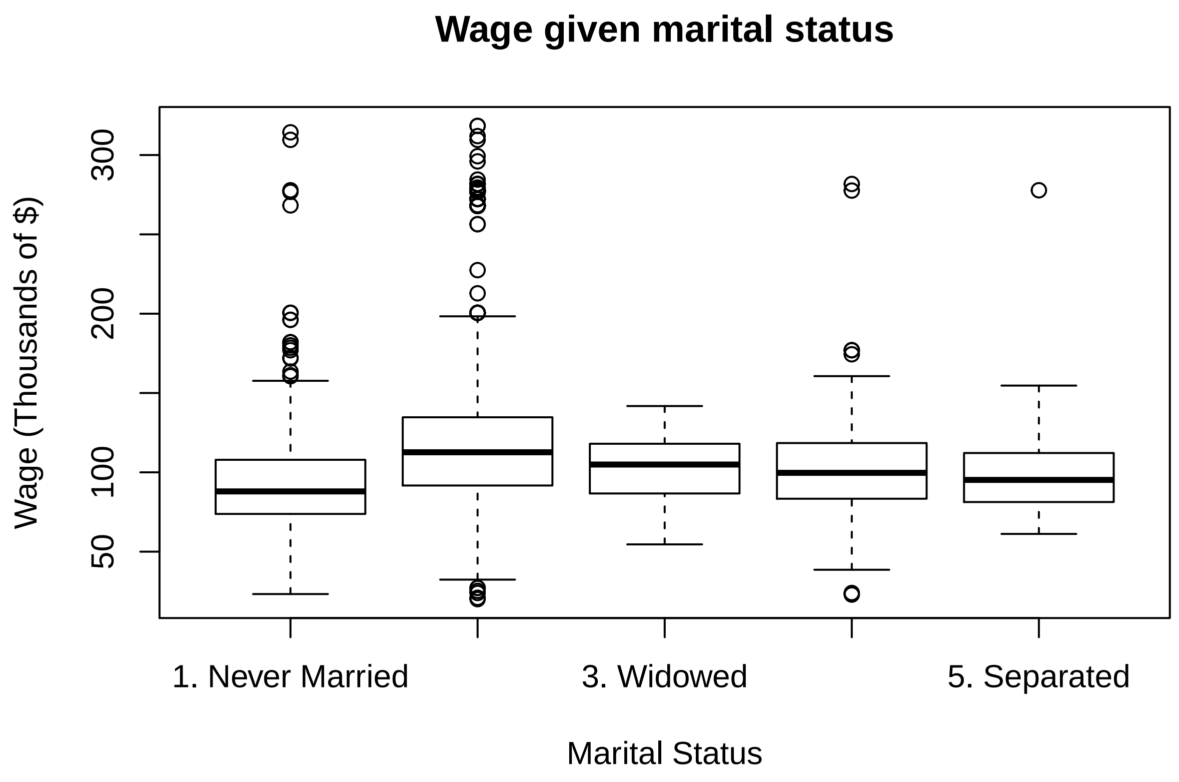
```r
step.fit = glm(wage~cut(age,8), data=Wage)

preds2=predict(step.fit,newdata=list(age=age.grid), se=T)
se.bands2=cbind(preds2$fit+2*preds2$se.fit,preds2$fit-2*preds2$se.fit)
```

```
plot(age,wage,xlim=agelims,cex=.5,col="darkgrey")
title("Step function using 8 cuts")
lines(age.grid,preds2$fit,lwd=2,col="blue")
matlines(age.grid,se.bands2,lwd =1,col="blue",lty =3)
```



7.

```
boxplot(wage~maritl, data=Wage, main="Wage given marital status",
        xlab="Marital Status",ylab="Wage (Thousands of $)")
```

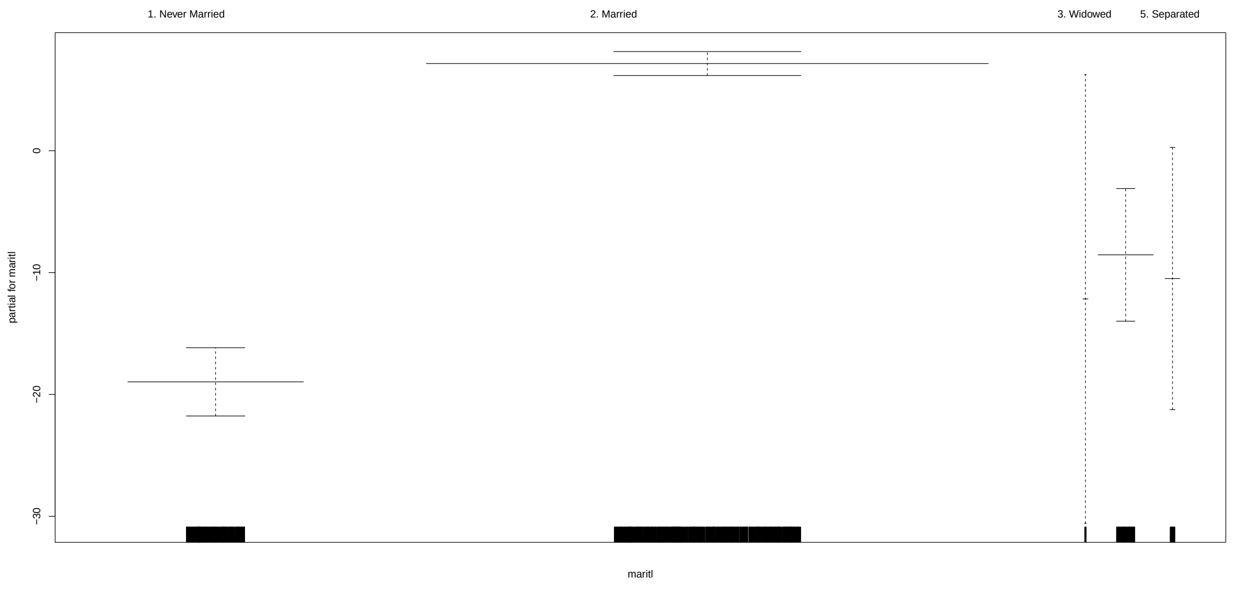


- `Married` and `Never Married` have a substantial number of outliers, and a bigger maximum (Q3 + 1.5*IQR) to minimum (Q1 - 1.5*IQR) range.
- The box plots for `Widowed` and `Separated` are similar to each other.

```
gam.model = gam(wage~maritl, data=Wage)
```

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

```
plot(gam.model, col="blue", se=T)
```



- `Married` status tends to have a higher median wage, and has a lower range for standard errors/lower confidence intervals that the other categories.
- `Widowed` has very high confidence intervals. This is likely due to this category having very few examples.

```
table(maritl)
```

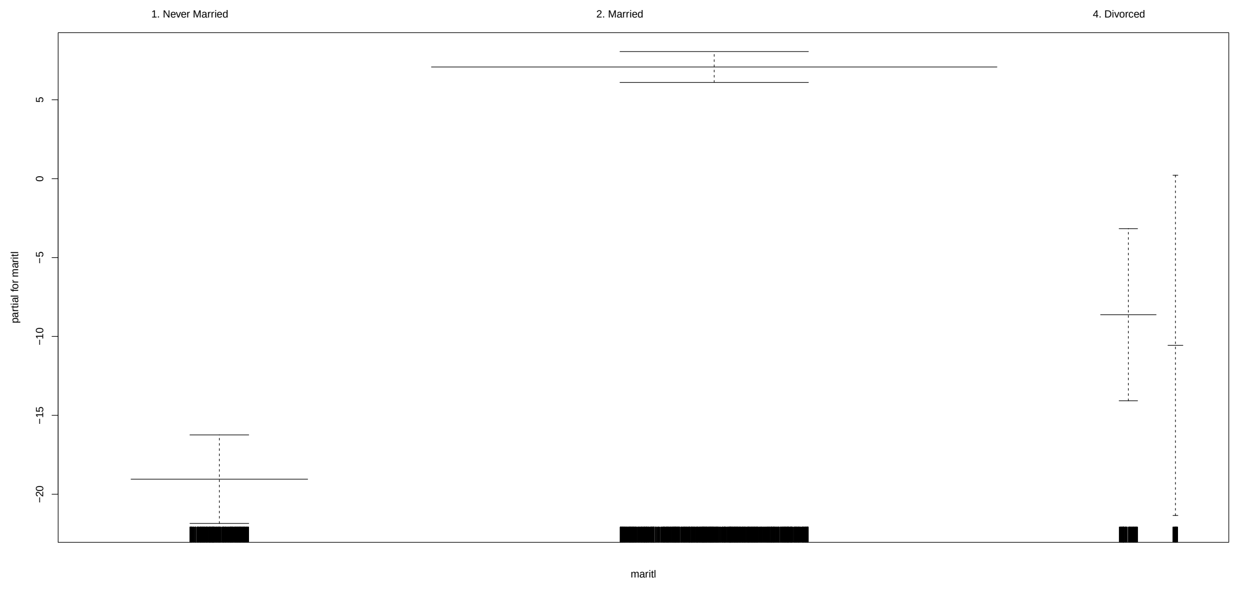
```
## maritl
## 1. Never Married        2. Married        3. Widowed       4. Divorced 
##               648              2074                19               204 
##      5. Separated 
##                55
```

Chart after removing `Widowed`:

```
# Removing rows with Widowed data.
Wage2 = Wage[(Wage$maritl!="3. Widowed"),]
gam.model = gam(wage~maritl, data=Wage2)
```

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

```
plot(gam.model, col="blue", se=T)
```

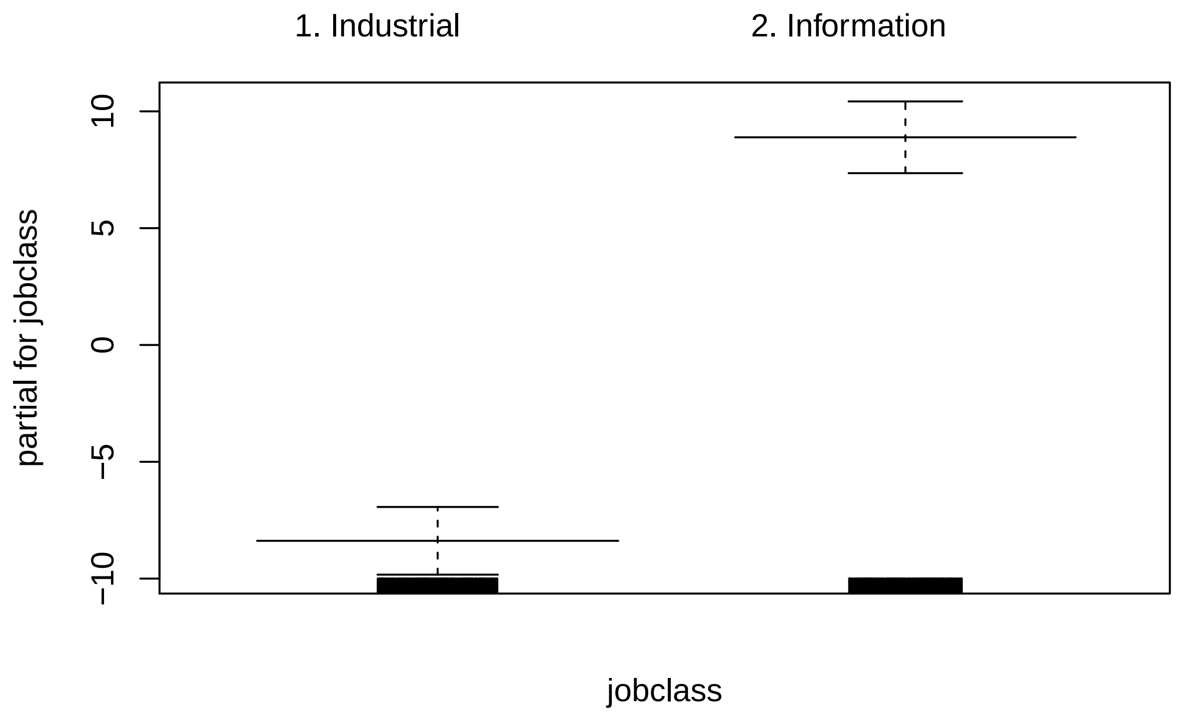


- The chart again confirms that those with a `Married` status have higher wages.

```
gam.model2 = gam(wage~jobclass, data=Wage)
```

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

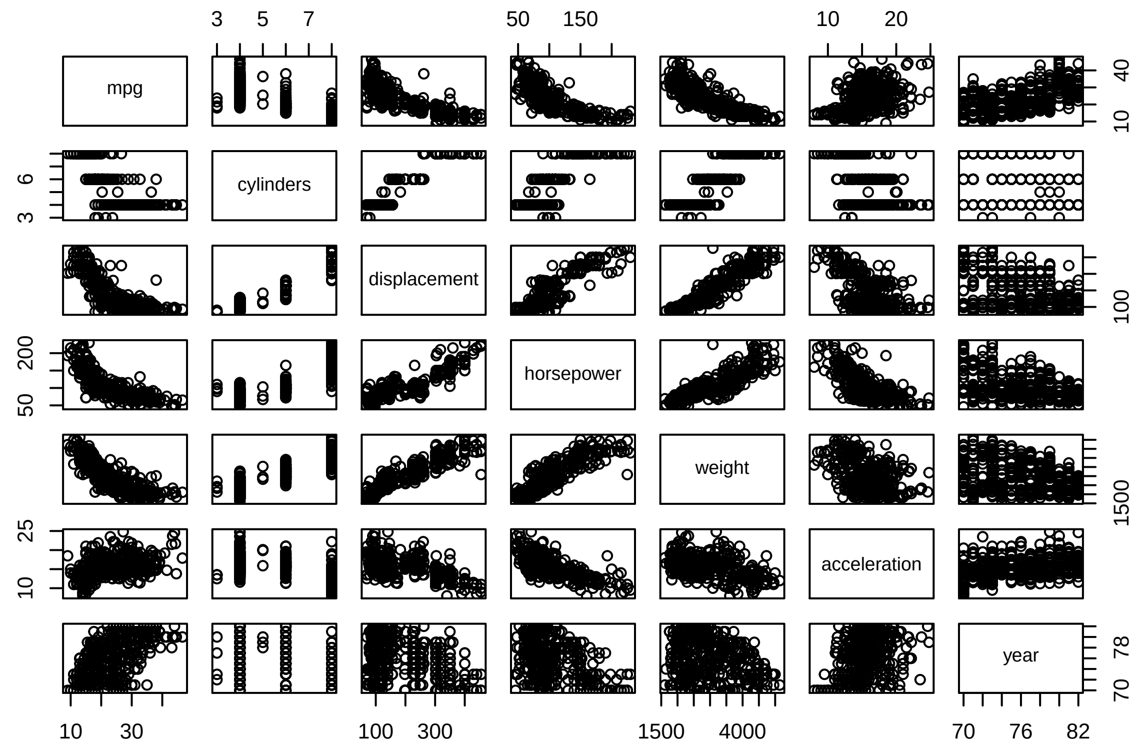
```
plot(gam.model2, col="blue", se=T)
```



- We can see that jobs in the `information` category have a higher wage.

8.

```r
#fix(Auto)
pairs(Auto[1:7])
```



- `mpg` appears to have a non-linear relationship with `horsepower`, `displacement` and `weight`.
- The relationship between `mpg` and `horsepower` will be explored in more detail.

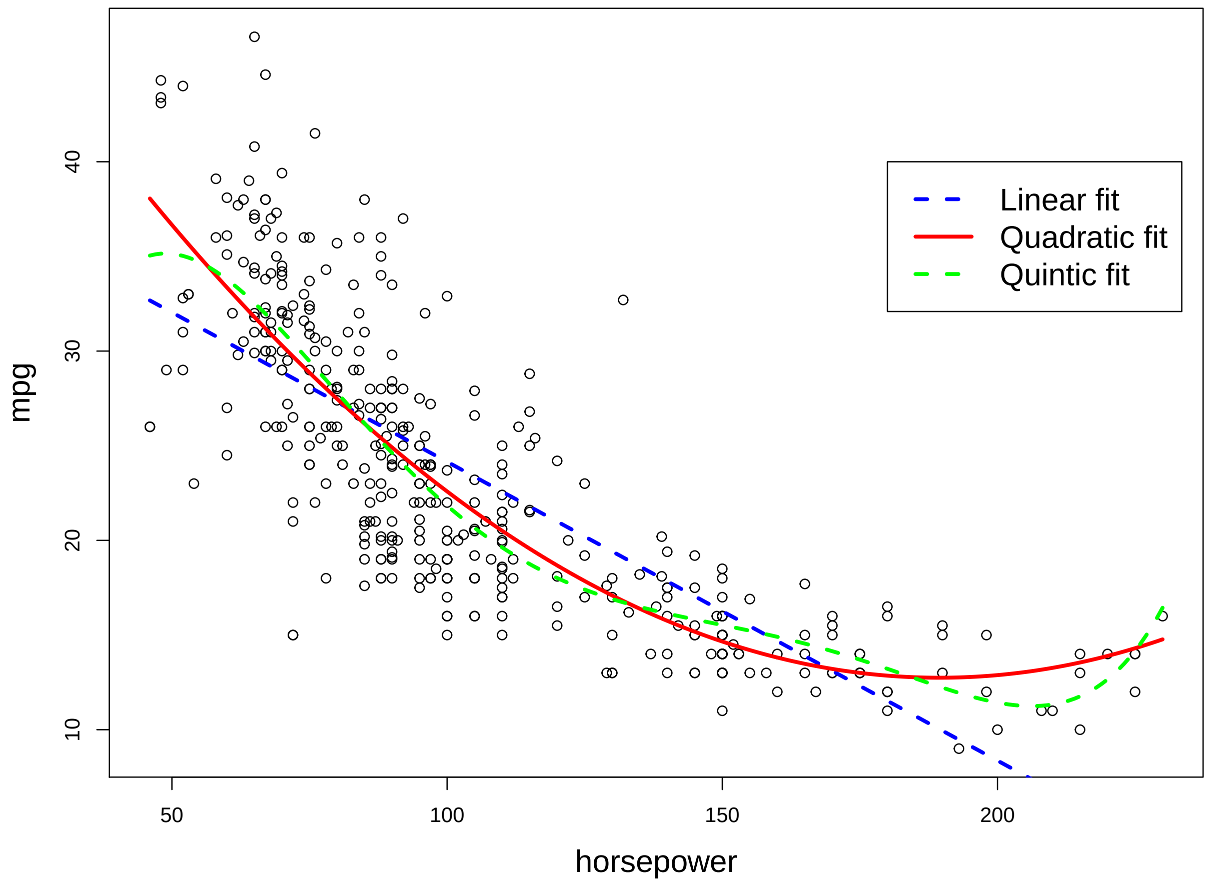
```
fit.1 = lm(mpg horsepower ,data=Auto)
fit.2 = lm(mpg poly(horsepower ,2) ,data=Auto)
fit.3 = lm(mpg poly(horsepower ,3) ,data=Auto)
fit.4 = lm(mpg poly(horsepower ,4) ,data=Auto)
fit.5 = lm(mpg poly(horsepower ,5) ,data=Auto)
fit.6 = lm(mpg poly(horsepower ,6) ,data=Auto)

anova(fit.1,fit.2,fit.3,fit.4,fit.5,fit.6)
```

```
## Analysis of Variance Table
##
## Model 1: mpg ~ horsepower
## Model 2: mpg ~ poly(horsepower, 2)
## Model 3: mpg ~ poly(horsepower, 3)
## Model 4: mpg ~ poly(horsepower, 4)
## Model 5: mpg ~ poly(horsepower, 5)
## Model 6: mpg ~ poly(horsepower, 6)
##   Res.Df    RSS Df Sum of Sq        F    Pr(>F)
## 1    390 9385.9
## 2    389 7442.0  1   1943.89 104.6659 < 2.2e-16 ***
## 3    388 7426.4  1     15.59   0.8396  0.360083
## 4    387 7399.5  1     26.91   1.4491  0.229410
## 5    386 7223.4  1    176.15   9.4846  0.002221 **
## 6    385 7150.3  1     73.04   3.9326  0.048068 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- The p-value comparing fit.1(linear) to fit.2(quadratic) is statistically significant, and the p-value comparing fit.2 to fit.3(cubic) is not significant. This indicates that a linear or cubic fit is not sufficient, but a quadratic fit should suffice.
- When taking into account the plot below and the ANOVA results, there is strong evidence of a non-linear relationship between horsepower and mpg.

Comparison of various fits to the data:

```r
hplim = range(Auto$horsepower)
hp.grid = seq(from=hplim[1],to=hplim[2])

preds1=predict(fit.1,newdata=list(horsepower=hp.grid))
preds2=predict(fit.2,newdata=list(horsepower=hp.grid))
preds5=predict(fit.5,newdata=list(horsepower=hp.grid))

# Plot of linear and polynomial fits.
plot(horsepower,mpg,xlim=hplim,cex.lab=1.5)
lines(hp.grid,preds1,lwd=3,col="blue",lty=2)
lines(hp.grid,preds2,lwd=3,col="red")
lines(hp.grid,preds5,lwd=3,col="green",lty=2)

legend(180,40,legend=c("Linear fit", "Quadratic fit", "Quintic fit"),
       col=c("blue", "red", "green"),lty=c(2,1,2), lwd=c(3,3,3),cex=1.5)
```



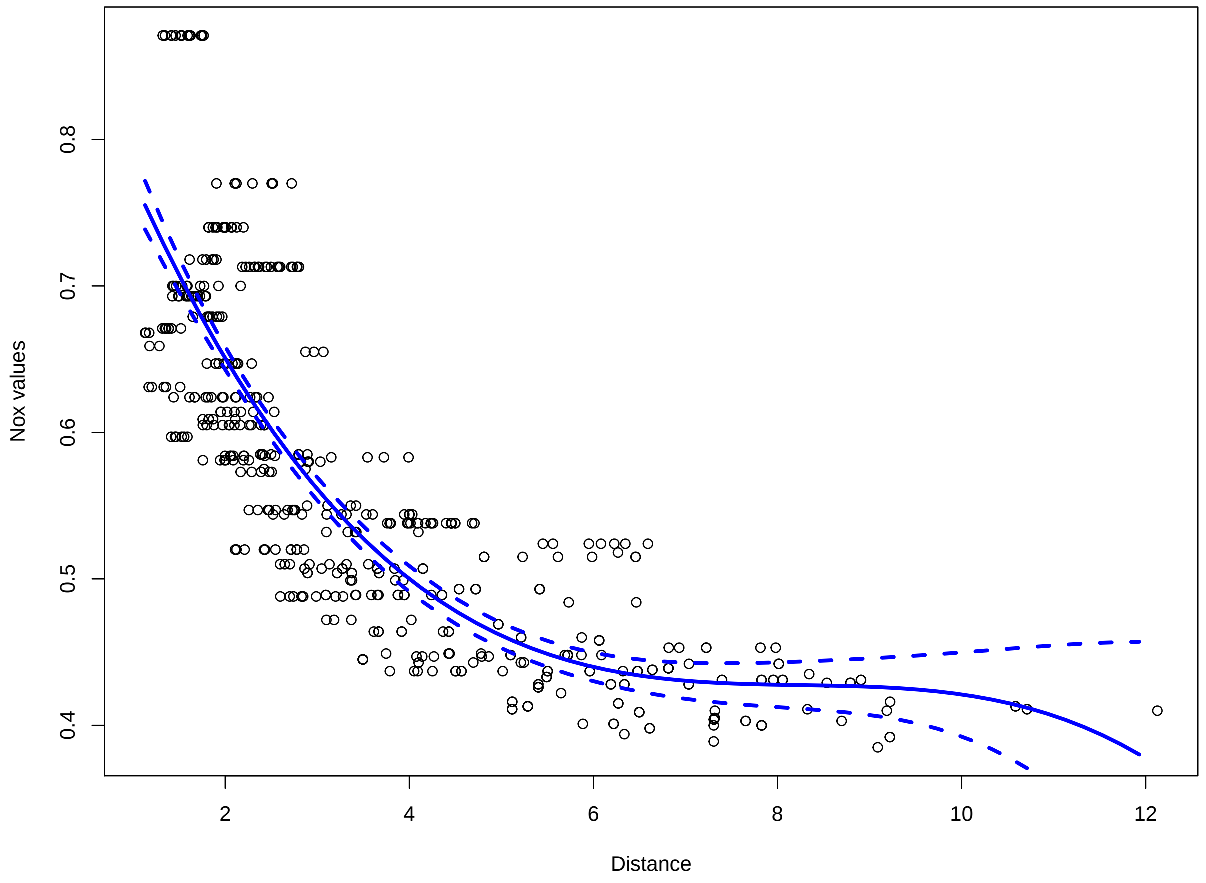
- As can be seen, a quadratic model likely provides the best fit to the underlying data.

9. (a)

```
plot(Boston$dis,Boston$nox, xlab="Distance", ylab="Nox values")

model.1 = glm(nox~poly(dis,3), data=Boston)
summary(model.1)
```

```
##
## Call:
## glm(formula = nox ~ poly(dis, 3), data = Boston)
##
## Deviance 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
##
## (Dispersion parameter for gaussian family taken to be 0.003852802)
##
##     Null deviance: 6.7810  on 505  degrees of freedom
## Residual deviance: 1.9341  on 502  degrees of freedom
## AIC: -1370.9
##
## Number of Fisher Scoring iterations: 2
```

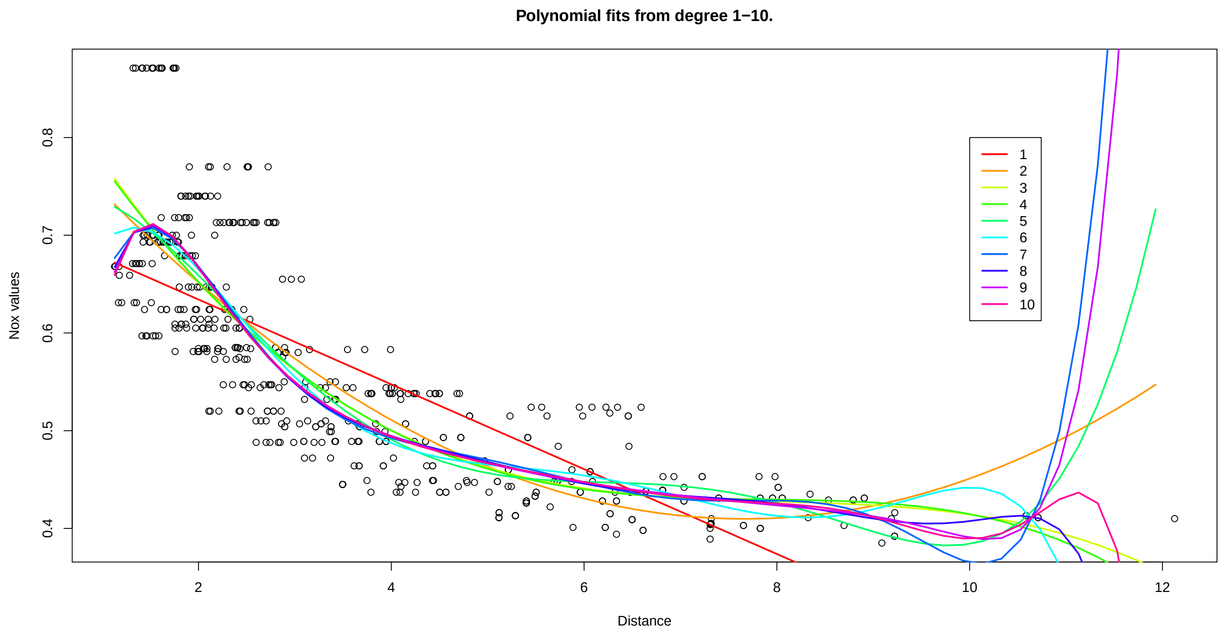
```
dis.grid = seq(from=min(Boston$dis),to=max(Boston$dis),0.2)
preds=predict(model.1,newdata=list(dis=dis.grid), se=T)
lines(dis.grid,preds$fit,col="blue",lwd=3)
lines(dis.grid,preds$fit+2*preds$se,col="blue",lwd=3,lty=2)
lines(dis.grid,preds$fit-2*preds$se,col="blue",lwd=3,lty=2)
```



- The regression summary shows a cubic fit is statistically significant.
- The cubic fit is plotted on the chart, and does appear to match the underlying shape of the data.

```r
set.seed(2)

boston_df = Boston
boston_sample = sample.split(boston_df$dis, SplitRatio = 0.80)
boston_train = subset(boston_df, boston_sample==TRUE)
boston_test = subset(boston_df, boston_sample==FALSE)
```

```r
rss = rep(0,10)
colours = rainbow(10)
plot(Boston$dis,Boston$nox,xlab="Distance", ylab="Nox values",
     main="Polynomial fits from degree 1-10.")

for (i in 1:10){
  model = glm(nox~poly(dis,i), data=boston_train)
  rss[i] = sum((boston_test$nox - predict(model,newdata=list(dis=boston_test$dis)))^2)
  preds=predict(model,newdata=list(dis=dis.grid))
  lines(dis.grid,preds,col=colours[i], lwd=2, lty=1)
}

legend(10,0.8,legend=1:10, col= colours[1:10],lty=1,lwd=2)
```



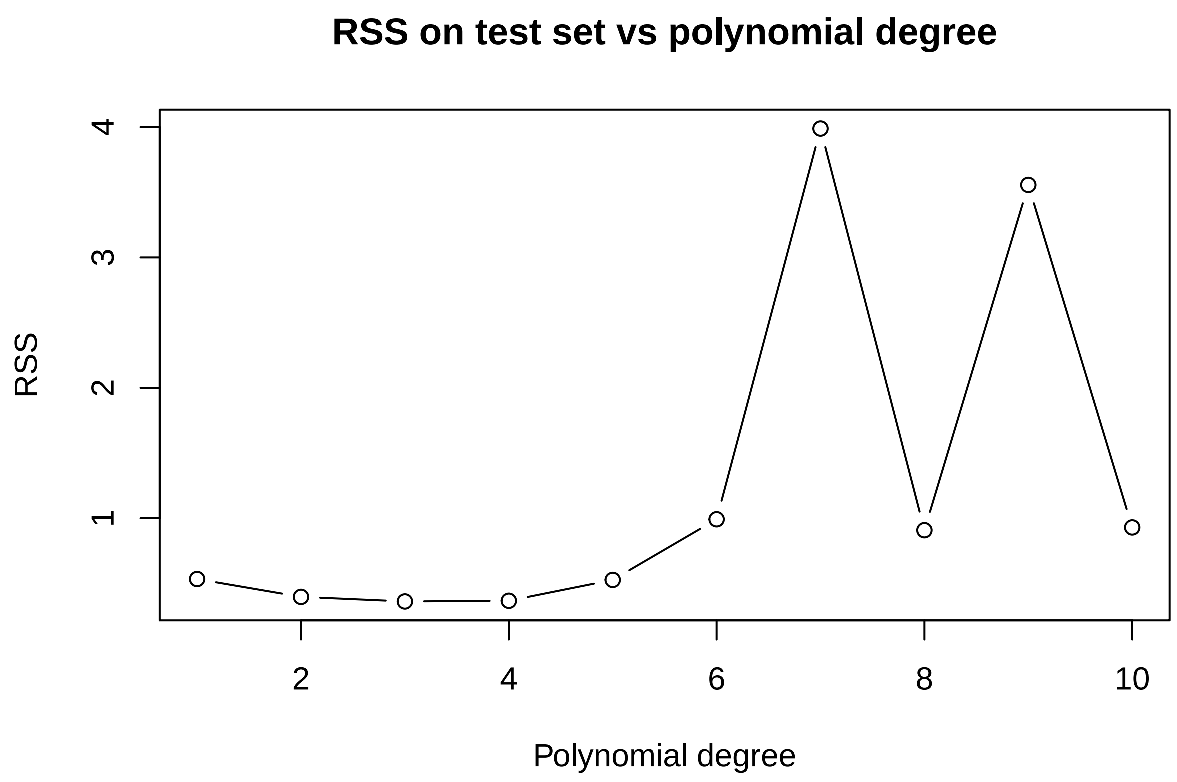
```r
rss
```

```
##  [1] 0.5335570 0.3967867 0.3616802 0.3668664 0.5269547 0.9919226 3.9884511
##  [8] 0.9079979 3.5563442 0.9293713
```

- The RSS decreases from the linear(0.533) to the cubic model(0.361), and increases thereafter. This supports the argument that the cubic model provides the best fit.

(b)

```r
# RSS on the test set.
plot(1:10,rss,xlab="Polynomial degree", ylab="RSS",
     main="RSS on test set vs polynomial degree", type='b')
```



- RSS is at a minimum for the degree 3 polynomial.

(d)

```r
# Regression spline with four degrees of freedom.
spline.fit = lm(nox~bs(dis,df=4), data=Boston)
summary(spline.fit)
```

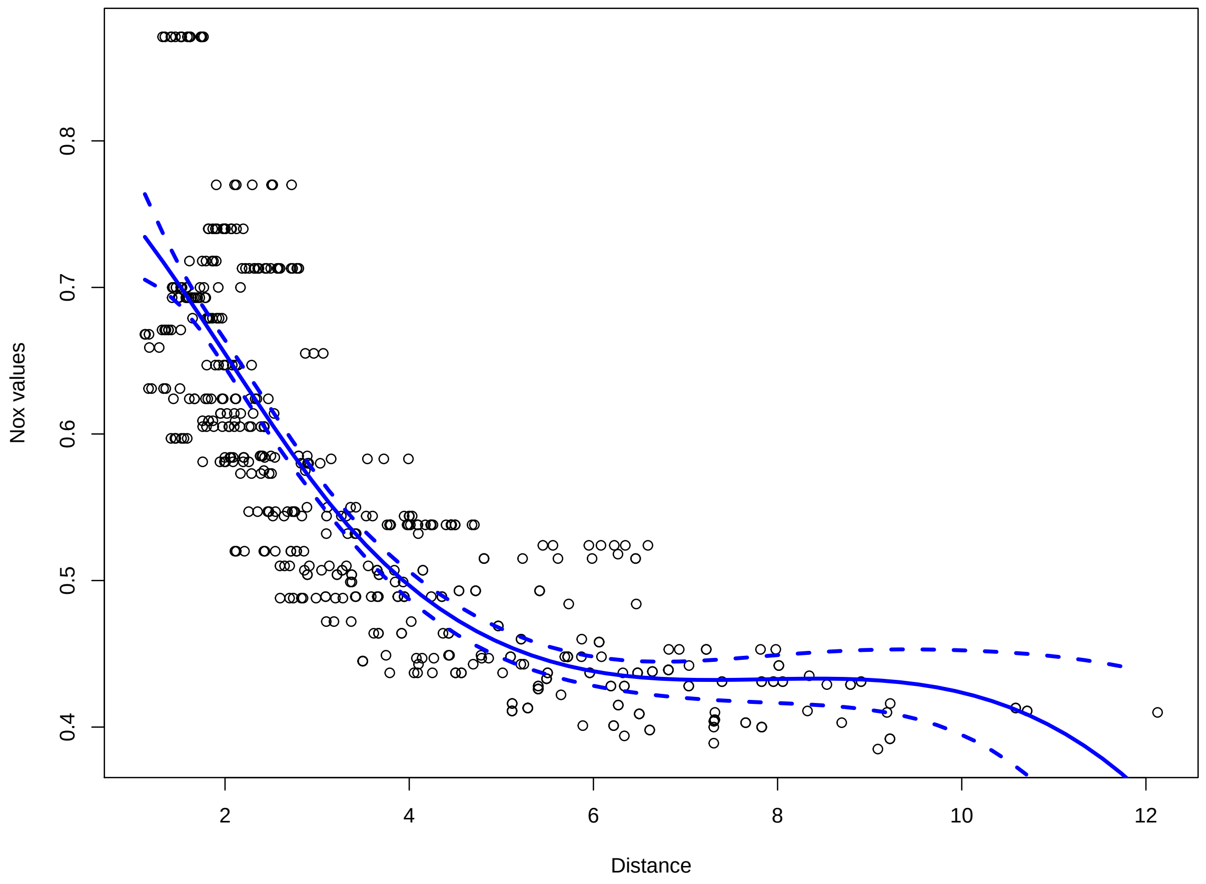
```
##
## Call:
## lm(formula = nox ~ bs(dis, df = 4), data = Boston)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.124622 -0.039259 -0.008514  0.020850  0.193891
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)        0.73447    0.01460  50.306  < 2e-16 ***
## bs(dis, df = 4)1 -0.05810    0.02186  -2.658  0.00812 **
## bs(dis, df = 4)2 -0.46356    0.02366 -19.596  < 2e-16 ***
## bs(dis, df = 4)3 -0.19979    0.04311  -4.634 4.58e-06 ***
## bs(dis, df = 4)4 -0.38881    0.04551  -8.544  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06195 on 501 degrees of freedom
## Multiple R-squared:  0.7164, Adjusted R-squared:  0.7142
## F-statistic: 316.5 on 4 and 501 DF,  p-value: < 2.2e-16
```

```r
attr(bs(Boston$dis,df=4),"knots")
```

```
##     50%
## 3.20745
```

- A regression spline with four degrees of freedom is statistically significant.
- The knots are chosen automatically when using the df() function. In this case we have single knot at the 50th percentile value.

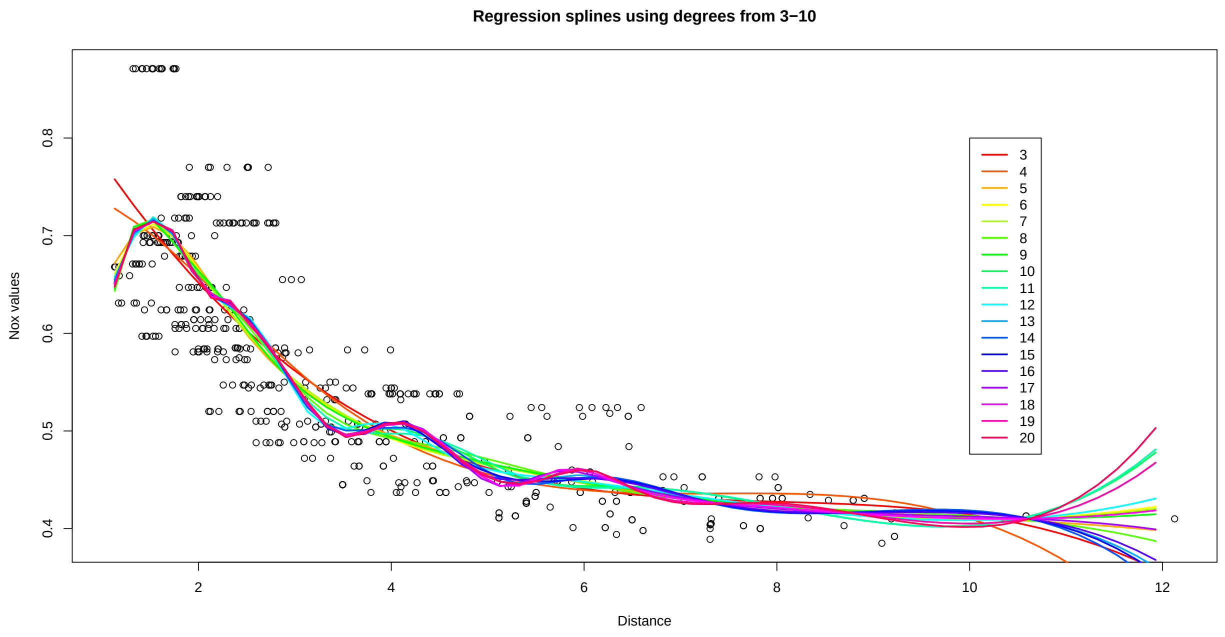
```r
plot(Boston$dis,Boston$nox,xlab="Distance", ylab="Nox values")
preds = predict(spline.fit, newdata=list(dis=dis.grid), se=T)
lines(dis.grid, preds$fit,col="blue",lwd=3)
lines(dis.grid, preds$fit+2*preds$se,col="blue",lwd=3,lty=2)
lines(dis.grid, preds$fit-2*preds$se,col="blue",lwd=3,lty=2)
```



- The resulting spline fit is very similar to that of polynomial regression using degree 3.

(e)

```
rss = rep(0,18)
colours = rainbow(18)
plot(Boston$dis,Boston$nox,xlab="Distance", ylab="Nox values",
     main="Regression splines using degrees from 3-10")

# Degree of freedom starts from 3, anything below is too small.
for (i in 3:20){
  spline.model = lm(nox~bs(dis,df=i), data=boston_train)
  rss[i-2] = sum((boston_test$nox - predict(spline.model,newdata=list(dis=boston_test$dis)))^2)
  preds=predict(spline.model,newdata=list(dis=dis.grid))
  lines(dis.grid,preds,col=colours[i-2], lwd=2, lty=1)
}
legend(10,0.8,legend=3:20, col=colours[1:18],lty=1,lwd=2)
```



```r
# Degree with minimum RSS value.
which.min(rss)+2
```

```
## [1] 12
```

- Smaller differences between spline fits than with the polynomial fits.
- RSS is lowest for the degree 12 model.

(f)

Cross validation:

```r
k=10
set.seed(3)

folds = sample(1:k, nrow(Boston), replace=T)
cv.errors = matrix(NA,k,18, dimnames = list(NULL, paste(3:20)))  #CV errors for degrees 3 to 20.

  # Create models (total=k) for each degree using the training folds.
  # Predict on held out folds and calculate their MSE's(total=k).
  # Continue until all j degrees have been used.
  # Take the mean of MSE's for each degree.

for(j in 3:20){
  for(i in 1:k){
    spline.model=lm(nox~bs(dis,df=j), data=Boston[folds!=i,])
    pred=predict(spline.model,Boston[folds==i,],id=i)
    cv.errors[i,j-2]=mean((Boston$nox[folds==i] - pred)^2)
  }
}

mean.cv.errors = apply(cv.errors,2,mean)
mean.cv.errors[which.min(mean.cv.errors)]
```

```
##           8
## 0.003693139
```

- The minimum for the CV errors is using degree 8.
- This is different to the degree 12 model found using a validation set.

Cross validation using cv.glm() function:

```r
set.seed(3)
cv.err = rep(0,18)

for(j in 3:20){
    fit=glm(nox~bs(dis,df=j), data=Boston)
    cv.err[j-2] = cv.glm(Boston, fit, K=10)$delta[1]
}

which.min(cv.err)+2
```

```
## [1] 8
```

- The cv.glm() method finds a minimum at degree 8. This is the same degree found using the previous cross validation method, but different to using a validation set.

10. (a)

```r
set.seed(4)
#sum(is.na(College$Outstate))

college_df = College
college_sample = sample.split(college_df$Outstate, SplitRatio = 0.80)
college_train = subset(college_df, college_sample==TRUE)
college_test = subset(college_df, college_sample==FALSE)
```

Forward stepwise selection to identify a satisfactory model:

```r
# Forward stepwise on the training set using all variables.
fit.fwd = regsubsets(Outstate~., data=college_train, nvmax=17, method="forward")
fit.summary = summary(fit.fwd)

# Some Statistical metrics.
which.min(fit.summary$cp)   #Estimate of the test error, lower is better.
```

```
## [1] 13
```

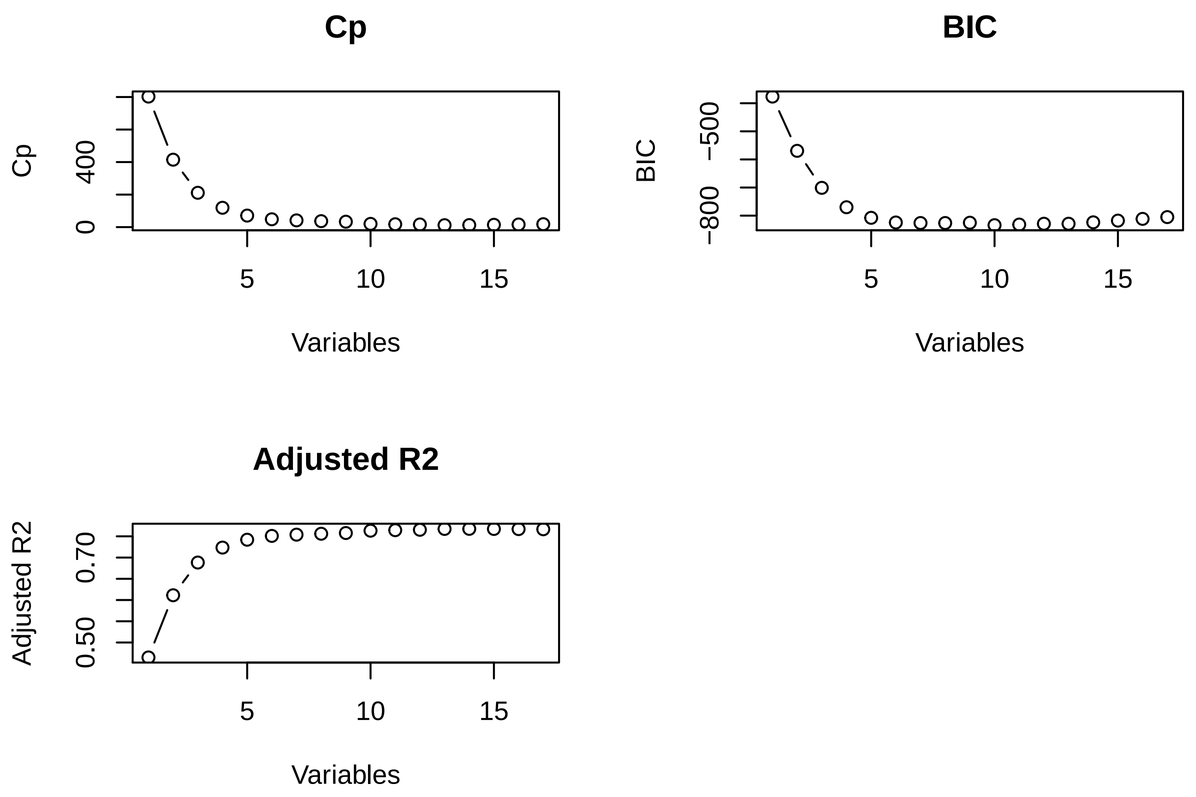
```r
which.min(fit.summary$bic)  #Takes a small value for models with low test errors,
```

```
## [1] 10
```

```r
                              #so lower is better generally.
which.max(fit.summary$adjr2) #A larger value indicates a model with low test error.
```

```
## [1] 14
```

```r
par(mfrow=c(2,2))
plot(1:17, fit.summary$cp,xlab="Variables",ylab="Cp",main="Cp", type='b')
plot(1:17, fit.summary$bic,xlab="Variables",ylab="BIC",main="BIC", type='b')
plot(1:17, fit.summary$adjr2,xlab="Variables",ylab="Adjusted R2",main="Adjusted R2", type='b')
```



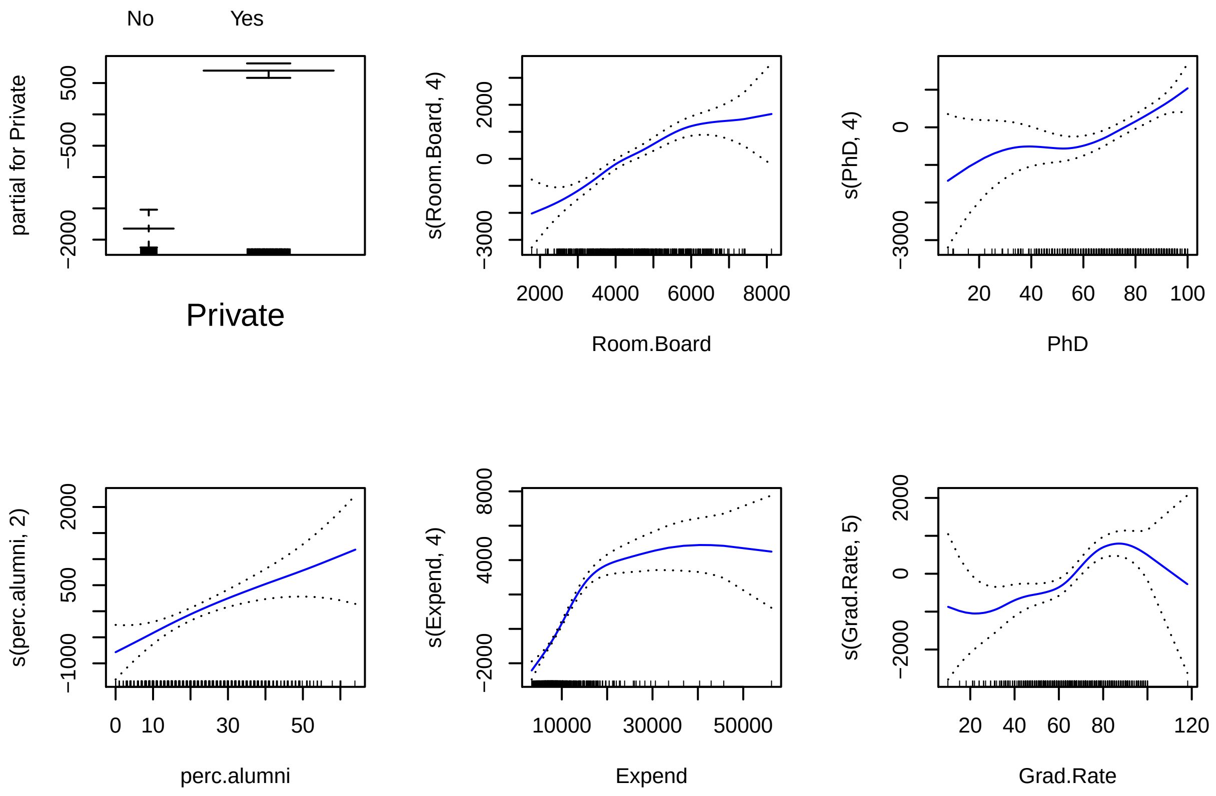
- The Cp, BIC and Adjusted R^2 all identify minimums and a maximum for models with a different number of variables. As can be seen from the charts, the metrics change rapidly as the number of variables increase, but there are only small improvements after a model with 6 variables.
- Therefore, the model with 6 variables is selected as it appears to be satisfactory in describing this relationship.

```r
coef(fit.fwd,6)
```

```
##   (Intercept)    PrivateYes    Room.Board           PhD   perc.alumni 
## -3650.4408407  2808.2152589     0.9423170    41.2025877    41.6939034 
##        Expend     Grad.Rate 
##     0.2261454    28.1575077
```

(b)

```r
gam.m1 = gam(Outstate~Private+
               s(Room.Board,4)+
               s(PhD,4)+
               s(perc.alumni,2)+
               s(Expend,4)+
               s(Grad.Rate,5), data=college_train)

par(mfrow=c(2,3))
plot(gam.m1, col="blue", se=T)
```



- Holding other variables fixed, out of state tuition increases as room and board costs get higher.
- Similarly, out of state tuition increases as the proportion of alumni who donate increase.

(c)

```r
# Predictions and MSE on Outstate from test set.
preds = predict(gam.m1,newdata = college_test)
mse = mean((college_test$Outstate - preds)^2)
mse
```

```
## [1] 3753134
```

```
# Graduation rate appears to have a non-linear relationship with Outstate.
# We can confirm this by performing an ANOVA test.
gam.m2 = gam(Outstate~Private+s(Room.Board,4)+s(PhD,4)+s(perc.alumni,2)
             +s(Expend,4), data=college_train) # No Grad.Rate
gam.m3 = gam(Outstate~Private+s(Room.Board,4)+s(PhD,4)+s(perc.alumni,2)
             +s(Expend,4)+Grad.Rate, data=college_train) # Linear Grad rate
gam.m4 = gam(Outstate~Private+s(Room.Board,4)+s(PhD,4)+s(perc.alumni,2)
             +s(Expend,4)+s(Grad.Rate,4), data=college_train) # Spline with 4 degrees
anova(gam.m2,gam.m3,gam.m4,gam.m1, test="F")
```

```
## Analysis of Deviance Table
##
## Model 1: Outstate ~ Private + s(Room.Board, 4) + s(PhD, 4) + s(perc.alumni,
##     2) + s(Expend, 4)
## Model 2: Outstate ~ Private + s(Room.Board, 4) + s(PhD, 4) + s(perc.alumni,
##     2) + s(Expend, 4) + Grad.Rate
## Model 3: Outstate ~ Private + s(Room.Board, 4) + s(PhD, 4) + s(perc.alumni,
##     2) + s(Expend, 4) + s(Grad.Rate, 4)
## Model 4: Outstate ~ Private + s(Room.Board, 4) + s(PhD, 4) + s(perc.alumni,
##     2) + s(Expend, 4) + s(Grad.Rate, 5)
##   Resid. Df Resid. Dev     Df Deviance       F    Pr(>F)
## 1       605 2185197963
## 2       604 2088077461 1.0000 97120502 28.3753 1.415e-07 ***
## 3       601 2059357222 3.0002 28720240  2.7968   0.03949 *
## 4       600 2053630774 0.9995  5726448  1.6739   0.19623
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

- The results provide compelling evidence that a GAM which includes `Grad.Rate` as a non-linear function(spline with degree 4) is needed(p=0.03939).

11. (a)

```
set.seed(5)

# Generated dataset
X1 = rnorm(100, sd=2)
X2 = rnorm(100, sd=sqrt(2))
eps = rnorm(100, sd = 1)
b0 = 5; b1=2.5 ; b2=11.5
Y = b0 +b1*X1 + b2*X2 + eps
```

(b) (c)

```
beta1 = 0.1

a=Y-beta1*X1
beta2=lm(a X2)$coef[2]
beta2
```

```
##       X2
## 11.90966
```

(d)

```
a=Y-beta2*X2
beta1 = lm(a X1)$coef[2]
beta1
```

```
##      X1
## 2.49134
```

(e)

```
beta.df = data.frame("beta0"=rep(0,1000),"beta1"=rep(0,1000),"beta2"=rep(0,1000))
beta1 = 0.1

for (i in 1:1000){
  a=Y-beta1*X1
  model = lm(a X2)
  beta2 = model$coef[2]
  beta.df$beta2[i]= beta2

  a=Y-beta2*X2
  model = lm(a X1)
  beta1 = model$coef[2]
  beta.df$beta1[i]=beta1

  beta.df$beta0[i]=model$coef[1]
  }

# Beta values after 5 iterations.
beta.df$beta0[5]
```

```
## [1] 4.996049
```

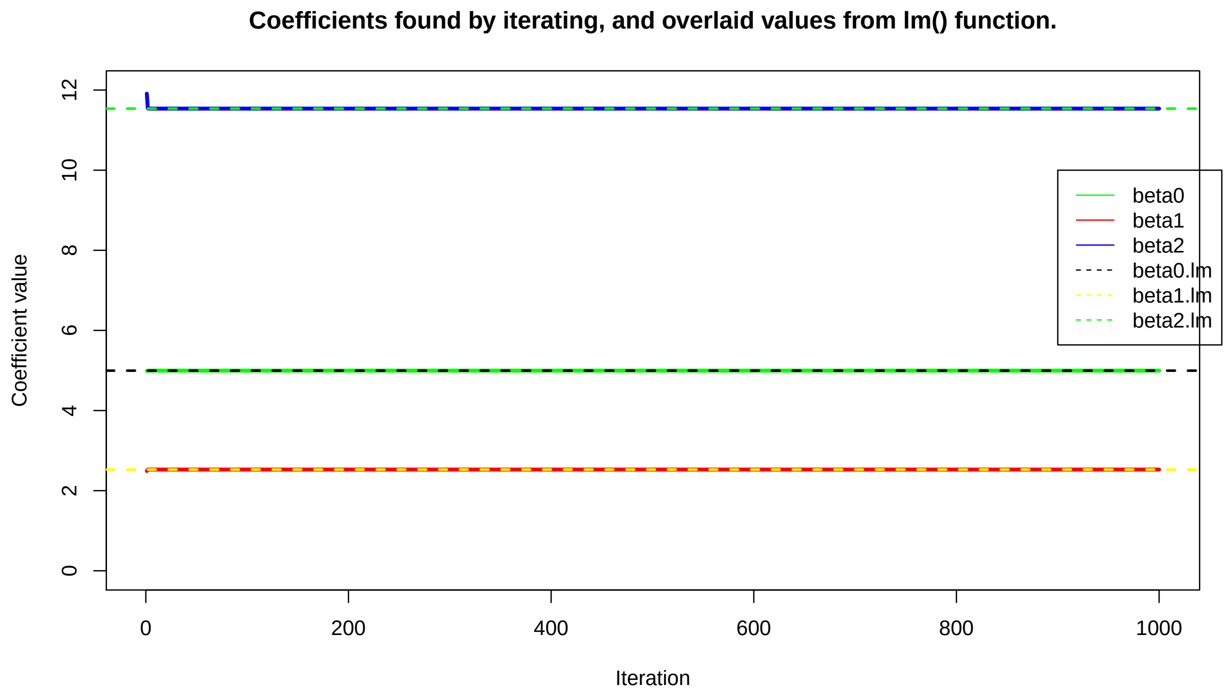
```
beta.df$beta1[5]
```

```
## [1] 2.52644
```

```
beta.df$beta2[5]
```

```
## [1] 11.53675
```

```
plot(1:1000,beta.df$beta2,ylim=range(0:12),type='l', lwd="3", col="blue",
     xlab="Iteration",ylab="Coefficient value")
title("Coefficients found by iterating, and overlaid values from lm() function.")
lines(1:1000,beta.df$beta1, col="red", lwd=3)
lines(1:1000,beta.df$beta0, col="green",lwd=3)

# Using results from multiple linear regression in part (f)
abline(h=4.996049, lty=2, lwd=2)
abline(h=2.526440, lty=2, lwd=2, col="yellow")
abline(h=11.536752, lty=2, lwd=2, col="green")

legend(900,10, legend=c("beta0", "beta1", "beta2", "beta0.lm", "beta1.lm", "beta2.lm"),
       col=c("green","red","blue","black","yellow","green"), lty = c(1,1,1,2,2,2), xpd=T)
```



(f)

```
lm.fit = lm(Y~X1+X2)
coef(lm.fit)
```

```
## (Intercept)          X1          X2
##    4.996049    2.526440   11.536752
```

- The coefficients found by multiple linear regression exactly match those found by iteration.

(g)

- Five iterations were needed to achieve a very good approximation to the multiple linear regression results.

12

Backfitting when p=100 and for 25 iterations:

```
set.seed(6)

n = 1000 # Number of examples
p = 100  # Number of predictors

# Generated dataset
X = matrix(rnorm(n*p),n,p)
beta0 = 8
betas = rnorm(100, sd = 2)
eps = rnorm(100, sd = 1)
```

```
Y = beta0 +  X%*%betas + eps
```

```
bhats = matrix(0,nrow=25,ncol=100)
intercepts = matrix(0,25,1)

# For loop to create models(M) that excludes the predictors that are 'fixed',
# and uses lm() to estimate the coefficent of the unfixed predictor.
# Estimates are stored such that the values can be used for the next set of calculations.

for (i in 1:25){
  for(j in 1:100){
    # Matrix multiplication of the fixed predictors and their respective coefficient estimates.
    M = Y - (X[,-j] %*% bhats[i,-j])
    # Linear regression with results stored in all rows from row(i):row(end).
    bhats[i:25,j] = lm(M~X[,j])$coef[2]
    intercepts[i] = lm(M~X[,j])$coef[1]
  }
}
```

```
lm.fit = lm(Y~X)
# Intercept and selected coefficient values.
coef(lm.fit)[1];coef(lm.fit)[2]; coef(lm.fit)[10]; coef(lm.fit)[20]
```

```
## (Intercept)
##    7.962958

##        X1
## -2.679775

##         X9
## -0.7035457

##      X19
## 3.413312
```

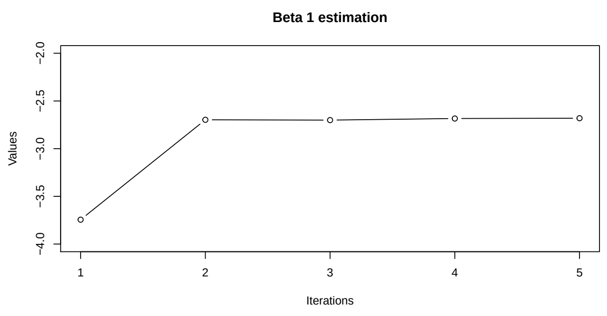
```
# Intercept, and selected coefficient values after iterating 25 times.
intercepts[25];bhats[25,1]; bhats[25,9]; bhats[25,19]
```

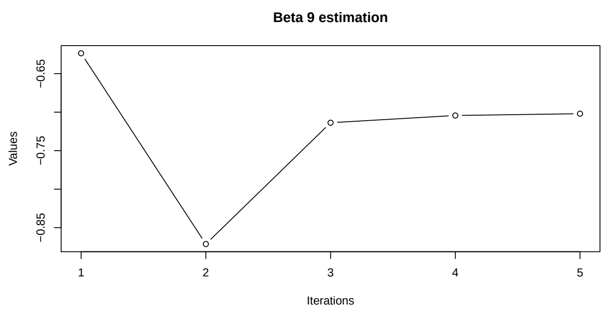
```
## [1] 7.962958

## [1] -2.679775

## [1] -0.7035457

## [1] 3.413312
```

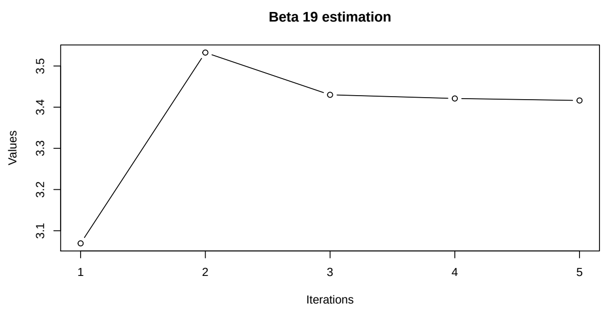
- As can be seen, the values given by iterating 25 times and using multiple linear regression match exactly. Additionally, we can see from the chart below that iterating quickly converges to these values.

Plot of some coefficient estimates after 5 iterations:

```r
par(mfrow=c(2,2))
plot(1:5,bhats[0:5,1],ylim=range(-2:-4),type='b',
     xlab='Iterations',ylab='Values',main='Beta 1 estimation')
plot(1:5,bhats[0:5,9],type='b',xlab='Iterations',ylab='Values',main='Beta 9 estimation')
plot(1:5,bhats[0:5,19],type='b',xlab='Iterations',ylab='Values',main='Beta 19 estimation')
```


Beta 1 estimation
Values
Iterations


Beta 9 estimation
Values
Iterations


Beta 19 estimation
Values
Iterations