ISLR Chapter 7 Applied Exercises

Yoosef Ghahreman

Contents

This is an extended solution to the applied exercises in chapter 7 of ISLR. Frequently, the solutions go beyond what is demanded in the book.

Exercise 6

We clear and load the data.

```
rm(list = ls())
search()  # detach anything that is not needed
```

```
## [1] ".GlobalEnv"        "package:stats"     "package:graphics"
## [4] "package:grDevices" "package:utils"     "package:datasets"
## [7] "package:methods"   "Autoloads"         "package:base"
```

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

Part 6.a)

Baseline C.V.

We perform C.V. for polynomial fit to find the optimal degree of the polynomial.

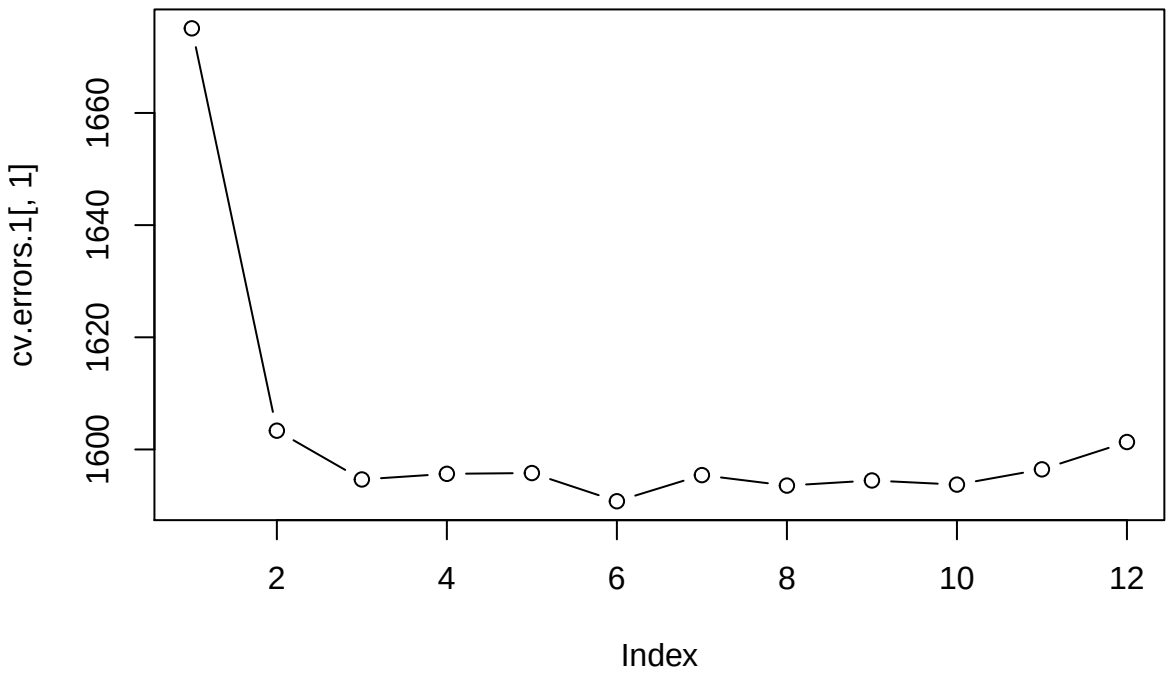
```
library(boot)
n.deg = 12
cv.errors.1 = matrix(NA, nrow = n.deg, ncol = 2, dimnames =
                     list(paste0(rep("Degree ", length = n.deg), 1:n.deg), NULL))
for (i.deg in 1:n.deg) {
  glm.fit = glm(wage ~ poly(age, i.deg, raw = T), data = Wage)
  glm.cv = cv.glm(data = Wage, glm.fit, K = 5)
  cv.errors.1[i.deg,] = glm.cv$delta
}
cv.errors.1
```

```
##                [,1]     [,2]
## Degree 1  1675.085 1674.972
## Degree 2  1603.348 1602.731
## Degree 3  1594.651 1594.417
## Degree 4  1595.657 1595.084
## Degree 5  1595.795 1595.158
## Degree 6  1590.769 1590.549
## Degree 7  1595.429 1594.594
## Degree 8  1593.559 1592.923
## Degree 9  1594.472 1593.463
## Degree 10 1593.734 1592.827
## Degree 11 1596.454 1595.232
## Degree 12 1601.327 1599.538
```

```
apply(cv.errors.1, FUN = which.min, MARGIN = 2)
```

```
## [1] 6 6
```

```
plot(cv.errors.1[, 1], type = "b")
```



When we re-run the code above, i.e. using different seeds, we see a great deal of variation in the result due to the randomness in CV splits. But in all cases we examine, adjusted and unadjusted MSE (`cv.errors.1$delta[1]` and `cv.errors.1$delta[2]`) lead to the same conclusion. The randomness does not affect the qualitative features of the plot by much. In all cases, we have a considerable drop in the error estimate from degree 1 to 2, and a small drop from 2 to 3. As a result, degree 3 seems a reasonable choice.

We could verify how good of a choice degree 3 is by finding error bands for the C.V. estimates. In the rest of analysis, we make two adjustents:

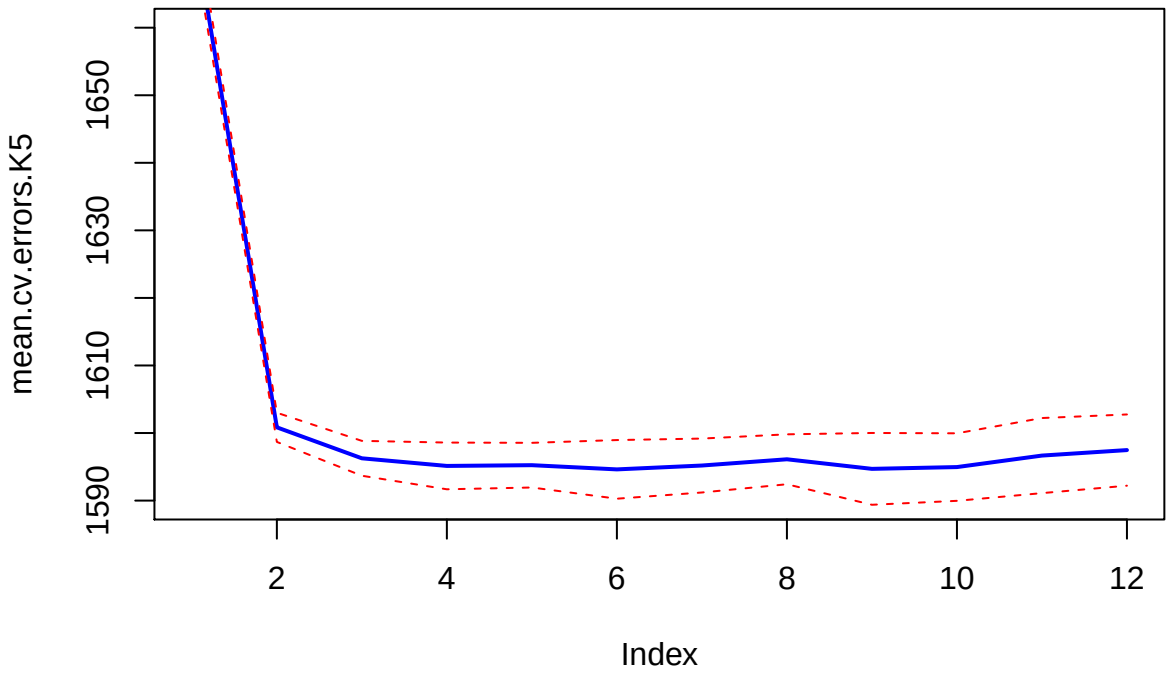
- We restrict attention to the unadjusted MSE and make the second dimension represent the randomness in choosing training set.
- We set the seed to 1 to guarantee reproducibility.

5-Fold C.V.

```
n.deg = 12
set.seed(1)
n.cv = 30
dim1.name = paste0(rep("Degree ", length = n.deg), 1:n.deg)
dim2.name = paste0(rep("CV Iter ", length = n.cv), 1:n.cv)
cv.errors.K5 = array(NA, dim = c(n.deg, n.cv), dimnames =
                     list(dim1.name, dim2.name))
for (i.deg in 1:n.deg) {
  for (j.cv in 1:n.cv) {
    glm.fit = glm(wage ~ poly(age, i.deg, raw = T), data = Wage)
    glm.cv = cv.glm(data = Wage, glm.fit, K = 5)
```

```
    cv.errors.K5[i.deg, j.cv] = glm.cv$delta[1]
  }
}
```

```
se.cv.errors.K5 = apply(cv.errors.K5, FUN = sd, MARGIN = 1)   # we want the cv dimension to be omitted
mean.cv.errors.K5 = apply(cv.errors.K5, FUN = mean, MARGIN = 1)
plot(mean.cv.errors.K5, type = "l", lwd = 2, col = "blue", ylim = c(1590, 1660))
se.bands.K5 = cbind(mean.cv.errors.K5 - 2*se.cv.errors.K5, mean.cv.errors.K5 + 2*se.cv.errors.K5)
matlines(se.bands.K5, lty = 2, col = "red")
```



ote that the plot above captures the variance of the MSE estimate for our sample, but could not say anything about the variation in the MSE due to randomness in training sets. The plot reinforces our choice of degree 3. The decrease in the MSE estimate from 2 to 3 is reflected not only in the MSE estimate, but is also confirmed by the one-standard-error rule, since the MSE of degree 2 is *not* within one standard error of the of the MSE for the degree three polynomials. We see that the difference between degrees higher than 2 can be associated with the randomness of splits in our training sample (note again that there is another source of variation: due to different training errors which we cannot say anything about it here; we just know it increases as we increase the number of folds).
Below, we show the risk involved with blindly choosing the degree with the lowest MSE estimate, regardless of the error in computing MSE, e.g. one-standard-error-rule.

```
cv.errors.K5[, 1]
```

```
##  Degree 1  Degree 2  Degree 3  Degree 4  Degree 5  Degree 6  Degree 7
##  1676.634  1599.854  1595.564  1598.107  1600.381  1593.097  1593.607
##  Degree 8  Degree 9 Degree 10 Degree 11 Degree 12
##  1595.020  1594.420  1591.311  1596.317  1596.495
```

```r
min.cv.K5 = apply(cv.errors.K5, FUN = which.min, MARGIN = 2)  # we want the degree dimension to be omit
median(min.cv.K5)
```

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

```r
mean(min.cv.K5)
```

```
## [1] 7.633333
```

The average choice of degree will be larger than 7 if we automate the choice without taking into account of the error is MSE estimation.

The results above highlight a few things, among others:

- Irrelevance of deciding based on adjusted or unadjusted error, as they lead to the same conclusion.
- Large variability in C.V. results

They also show the importance of naming array dimensions in readability of numbers.

What if we increase the number of folds? *We know it would increase the variance resulting from the choice of training set, although we have no way to quantify that, and we will see it decreases bias.*

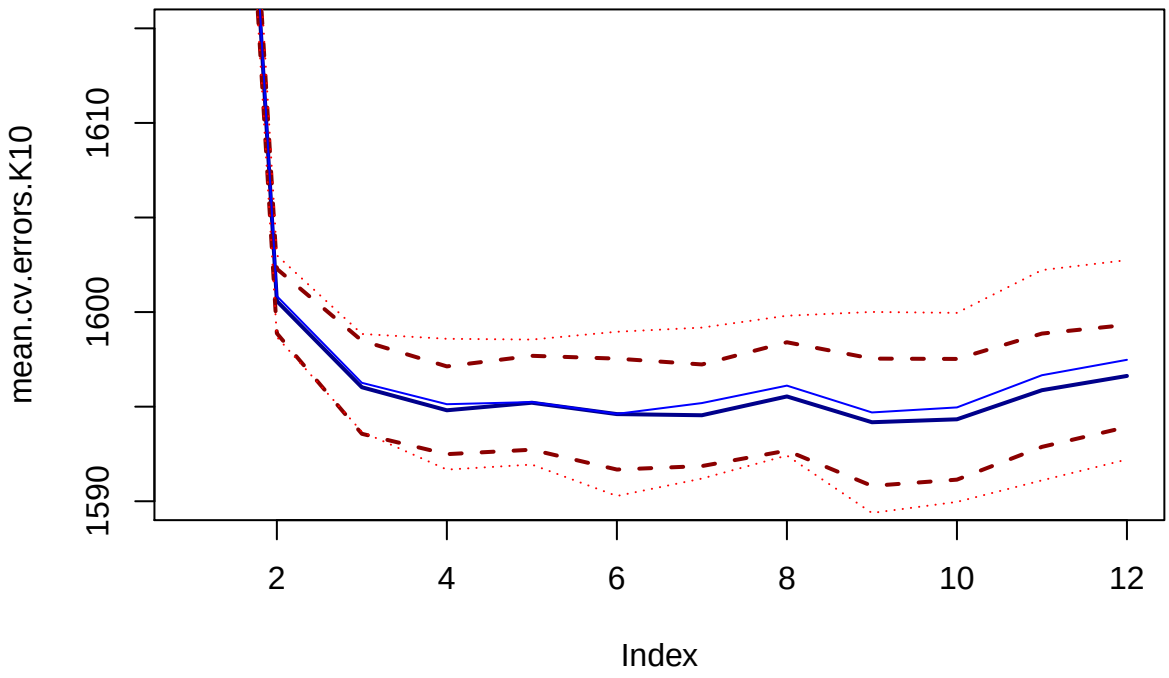
10-Fold C.V.

In the example below, we increase the number of folds to 10.

```r
n.folds = 10
n.deg = 12
set.seed(1)
n.cv = 30
dim1.name = paste0(rep("Degree ", length = n.deg), 1:n.deg)
dim2.name = paste0(rep("CV Iter ", length = n.cv), 1:n.cv)
cv.errors.K10 = array(NA, dim = c(n.deg, n.cv), dimnames =
                      list(dim1.name, dim2.name))
for (i.deg in 1:n.deg) {
  for (j.cv in 1:n.cv) {
    glm.fit = glm(wage ~ poly(age, i.deg, raw = T), data = Wage)
    glm.cv = cv.glm(data = Wage, glm.fit, K = n.folds)
    cv.errors.K10[i.deg, j.cv] = glm.cv$delta[1]
  }
}
```

Now we produce the output with 30 cross-validations:

```r
se.cv.errors.K10 = apply(cv.errors.K10, FUN = sd, MARGIN = 1)
mean.cv.errors.K10 = apply(cv.errors.K10, FUN = mean, MARGIN = 1)
plot(mean.cv.errors.K10, type = "l", lwd = 2, col = "darkblue", ylim = c(1590, 1615))
se.bands.K10 = cbind(mean.cv.errors.K10 - 2*se.cv.errors.K10, mean.cv.errors.K10 + 2*se.cv.errors.K10)
matlines(se.bands.K10, lty = 2, col = "darkred", lwd = 2)
lines(mean.cv.errors.K5, type = "l", lty = 1, col = "blue")
matlines(se.bands.K5, lty = 3, col = "red")
```



The similarity between 5 and 10 folds is reassuring. But with 10 folds, degree 4 looks a bit more reasonable, although degree 3 would still be the choice according the one-standard-error-rule.

As we know theoretically, larger number of folds leads to larger variance across training sets, and lower bias. The lower bias can be observed above, but note that the lower variance above for 10 folds is across different random splits. The variance across training sets can not be evaluated unless we have the population's data. Below, we see that the blind choice of degree leads to same conclusion for 5 and 10 folds:

```
cv.errors.K10[, 1]
```

```
##  Degree 1  Degree 2  Degree 3  Degree 4  Degree 5  Degree 6  Degree 7
##  1675.837  1600.078  1595.490  1595.037  1596.418  1593.055  1595.722
##  Degree 8  Degree 9 Degree 10 Degree 11 Degree 12
##  1594.894  1592.538  1591.164  1595.571  1595.951
```

```
min.cv.K10 = apply(cv.errors.K5, FUN = which.min, MARGIN = 2)  # we want the degree dimension to be omi
median(min.cv.K10)
```

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

```
mean(min.cv.K10)
```

```
## [1] 7.633333
```

LOOCV

To completely avoid randomness due to splits, we could use LOOCV. The LOOCV is too slow in `cv.glm` since it does not use the LOOCV formula for the linear regression. So we use `CV()` in the library `forecast`:

```
library("forecast")
cv.errors.loocv = matrix(NA, nrow = n.deg, ncol = 5, dimnames =
                           list(NULL, c("CV", "AIC", "AICc", "BIC", "AdjR2")))
for (i.deg in 1:n.deg) {
  glm.fit = lm(wage ~ poly(age, i.deg, raw = T), data = Wage)
  glm.cv = CV(glm.fit)
  cv.errors.loocv[i.deg, ] = glm.cv
}
cv.errors.loocv
```

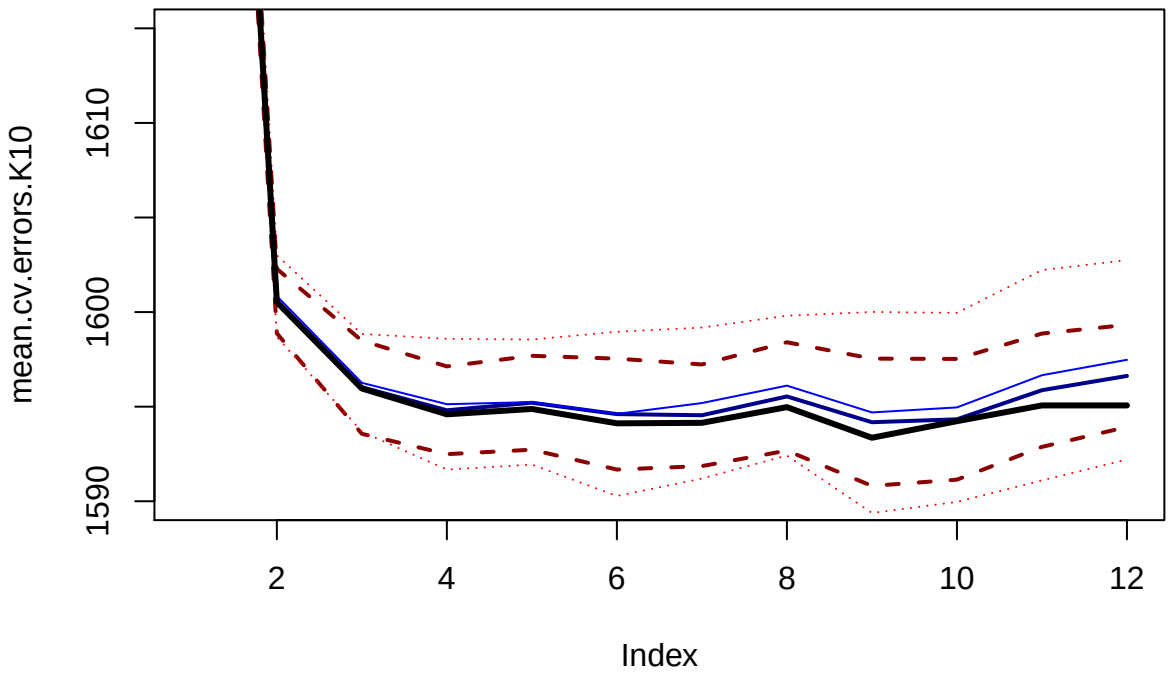
```
##             CV      AIC     AICc      BIC      AdjR2
##  [1,] 1676.235 22275.04 22275.05 22293.06 0.03795313
##  [2,] 1600.529 22137.17 22137.18 22161.19 0.08147259
##  [3,] 1595.960 22129.29 22129.31 22159.32 0.08418615
##  [4,] 1594.596 22127.48 22127.50 22163.51 0.08504433
##  [5,] 1594.879 22128.67 22128.71 22170.71 0.08498474
##  [6,] 1594.119 22128.20 22128.24 22176.25 0.08543354
##  [7,] 1594.145 22128.59 22128.65 22182.64 0.08561834
##  [8,] 1594.975 22130.51 22130.58 22190.57 0.08533695
##  [9,] 1593.356 22128.09 22128.18 22194.16 0.08637636
## [10,] 1594.232 22130.09 22130.20 22202.17 0.08607121
## [11,] 1595.064 22132.03 22132.15 22210.11 0.08578518
## [12,] 1595.064 22132.03 22132.15 22210.11 0.08578518
```

```
which.min(cv.errors.loocv[, 1])
```

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

To enable the comparison of different results, we see the 5-fold, 10-fold and n-fold C.V. in the next plot:

```
se.cv.errors.K10 = apply(cv.errors.K10, FUN = sd, MARGIN = 1)
mean.cv.errors.K10 = apply(cv.errors.K10, FUN = mean, MARGIN = 1)
plot(mean.cv.errors.K10, type = "l", lwd = 2, col = "darkblue", ylim = c(1590, 1615))
se.bands.K10 = cbind(mean.cv.errors.K10 - 2*se.cv.errors.K10, mean.cv.errors.K10 + 2*se.cv.errors.K10)
matlines(se.bands.K10, lty = 2, col = "darkred", lwd = 2)
lines(mean.cv.errors.K5, type = "l", lty = 1, col = "blue")
matlines(se.bands.K5, lty = 3, col = "red")
matlines(cv.errors.loocv[, 1], col = "black", lty = 1, lwd = 3)
```



The thick black line represents the LOOCV estimate, which is lower as expected. We know that the variance of LOOCV estimate in terms of split randomness is zero. For now, forget about the error bands. Although the full lines look very similar, note that we do not know how accurate each of them is. The brighter the line, the lower the number of folds, and the lower the variance (with respect to training set), but the larger the bias. The bias is minimal when we use LOOCV, but variance is minimal when we use 5 folds.

Note that so far we assume that if degree d is among the features, all degrees below d would also be. But this needs not be the case. For example, in the figure above we see that adding the degree 9 leads to some improvement while lower degrees like 7 and 8 lead to no improvement (although we note that in this data this may be only due to noise and not carry over to an indepenent data). One way to proceed is to use variable selection, via subset selection or lasso. We follow an alternative route which is to use orthogonal polyomials.

Orthogonal Polynomials

Let's compare the results so far with the result of orthogonal polynomials (which is closely related to ANOVA):

```
glm.fit = glm(wage ~ poly(age, n.deg), data = Wage)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = wage ~ poly(age, n.deg), data = Wage)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -100.546   -24.336    -5.093    15.576   199.857
##
```

```
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)         111.7036     0.7286 153.320  < 2e-16 ***
## poly(age, n.deg)1   447.0679    39.9053  11.203  < 2e-16 ***
## poly(age, n.deg)2  -478.3158    39.9053 -11.986  < 2e-16 ***
## poly(age, n.deg)3   125.5217    39.9053   3.145  0.00167 **
## poly(age, n.deg)4   -77.9112    39.9053  -1.952  0.05098 .
## poly(age, n.deg)5   -35.8129    39.9053  -0.897  0.36955
## poly(age, n.deg)6    62.7077    39.9053   1.571  0.11619
## poly(age, n.deg)7    50.5498    39.9053   1.267  0.20535
## poly(age, n.deg)8   -11.2547    39.9053  -0.282  0.77794
## poly(age, n.deg)9   -83.6918    39.9053  -2.097  0.03605 *
## poly(age, n.deg)10    1.6240    39.9053   0.041  0.96754
## poly(age, n.deg)11   10.1588    39.9053   0.255  0.79907
## poly(age, n.deg)12   -2.6076    39.9053  -0.065  0.94790
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 1592.431)
##
##     Null deviance: 5222086  on 2999  degrees of freedom
## Residual deviance: 4756591  on 2987  degrees of freedom
## AIC: 30648
##
## Number of Fisher Scoring iterations: 2
```

There is compelling evidence for degrees 1 to 3, and weaker evidence for degree 9 and 4. If we use raw polynomials, we would see how problems wih multicollinearity dramatically increasethe standard error of the estimates, and make them unreliable:

```
glm.fit = glm(wage ~ poly(age, n.deg, raw = T), data = Wage)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = wage ~ poly(age, n.deg, raw = T), data = Wage)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -100.546   -24.336    -5.093    15.575   199.857
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 -1.341e+04  2.135e+05  -0.063    0.950
## poly(age, n.deg, raw = T)1   5.104e+03  6.679e+04   0.076    0.939
## poly(age, n.deg, raw = T)2  -8.138e+02  9.357e+03  -0.087    0.931
## poly(age, n.deg, raw = T)3   7.333e+01  7.766e+02   0.094    0.925
## poly(age, n.deg, raw = T)4  -4.202e+00  4.256e+01  -0.099    0.921
## poly(age, n.deg, raw = T)5   1.625e-01  1.623e+00   0.100    0.920
## poly(age, n.deg, raw = T)6  -4.378e-03  4.420e-02  -0.099    0.921
## poly(age, n.deg, raw = T)7   8.307e-05  8.666e-04   0.096    0.924
## poly(age, n.deg, raw = T)8  -1.107e-06  1.215e-05  -0.091    0.927
## poly(age, n.deg, raw = T)9   1.013e-08  1.189e-07   0.085    0.932
## poly(age, n.deg, raw = T)10 -6.069e-11  7.713e-10  -0.079    0.937
## poly(age, n.deg, raw = T)11  2.145e-13  2.981e-12   0.072    0.943
```

```
## poly(age, n.deg, raw = T)12 -3.394e-16  5.193e-15  -0.065    0.948
##
## (Dispersion parameter for gaussian family taken to be 1592.431)
##
##     Null deviance: 5222086  on 2999  degrees of freedom
## Residual deviance: 4756591  on 2987  degrees of freedom
## AIC: 30648
##
## Number of Fisher Scoring iterations: 2
```

ANOVA

Now we use ANOVA to compare models with number of degrees equal to 3, 4, and 9.

```
lm.fit.1 = lm(wage ~ poly(age, 3, raw = T), data = Wage)
lm.fit.2 = lm(wage ~ poly(age, 4, raw = T), data = Wage)
lm.fit.3 = lm(wage ~ poly(age, 9, raw = T), data = Wage)
anova(lm.fit.1, lm.fit.2, lm.fit.3)
```

```
## Analysis of Variance Table
##
## Model 1: wage ~ poly(age, 3, raw = T)
## Model 2: wage ~ poly(age, 4, raw = T)
## Model 3: wage ~ poly(age, 9, raw = T)
##   Res.Df     RSS Df Sum of Sq      F  Pr(>F)
## 1   2996 4777674
## 2   2995 4771604  1    6070.2 3.8156 0.05087 .
## 3   2990 4756703  5   14901.1 1.8733 0.09566 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(lm.fit.1, lm.fit.3)
```

```
## Analysis of Variance Table
##
## Model 1: wage ~ poly(age, 3, raw = T)
## Model 2: wage ~ poly(age, 9, raw = T)
##   Res.Df     RSS Df Sum of Sq     F  Pr(>F)
## 1   2996 4777674
## 2   2990 4756703  6     20971 2.197 0.04053 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(lm.fit.2, lm.fit.3)
```

```
## Analysis of Variance Table
##
## Model 1: wage ~ poly(age, 4, raw = T)
## Model 2: wage ~ poly(age, 9, raw = T)
##   Res.Df     RSS Df Sum of Sq      F  Pr(>F)
## 1   2995 4771604
## 2   2990 4756703  5     14901 1.8733 0.09566 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We see that 4 is better than 3 at about 5% level, 9 is better than 3 at 5% level and 9 is better than 4 at 10%

level.
We used raw polynomials in ANOVA analysis above. The results below show that the result is the same when we use raw polynomials in `anova`.

```
lm.fit.1 = lm(wage ~ poly(age, 3), data = Wage)
lm.fit.2 = lm(wage ~ poly(age, 4), data = Wage)
lm.fit.3 = lm(wage ~ poly(age, 9), data = Wage)
anova(lm.fit.1, lm.fit.2, lm.fit.3)
```

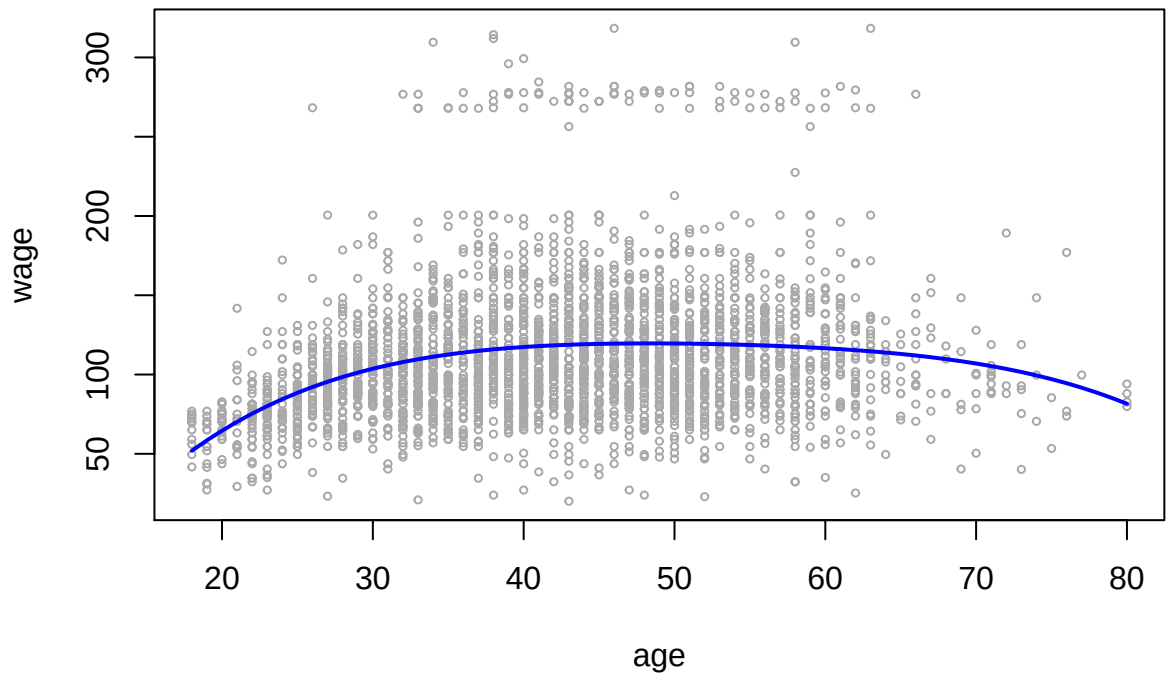
```
## Analysis of Variance Table
##
## Model 1: wage ~ poly(age, 3)
## Model 2: wage ~ poly(age, 4)
## Model 3: wage ~ poly(age, 9)
##   Res.Df     RSS Df Sum of Sq      F  Pr(>F)
## 1   2996 4777674
## 2   2995 4771604  1    6070.2 3.8156 0.05087 .
## 3   2990 4756703  5   14901.1 1.8733 0.09566 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(lm.fit.1, lm.fit.3)
```

```
## Analysis of Variance Table
##
## Model 1: wage ~ poly(age, 3)
## Model 2: wage ~ poly(age, 9)
##   Res.Df     RSS Df Sum of Sq     F  Pr(>F)
## 1   2996 4777674
## 2   2990 4756703  6     20971 2.197 0.04053 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We will choose degree 4, especially since it is unusual to use degrees above 3 or 4 with polynomial regression (which is likely to lead to unusual behavior of the fit at the boundaries. The fit is shown in the figure below:

```
lm.fit = lm(wage ~ poly(age, degree = 4, raw = T), data = Wage)
plot(age, wage, col = "darkgrey", cex = 0.5)
age.grid = seq(from = min(age), to = max(age))
preds = predict(lm.fit, newdata = data.frame(age = age.grid))
lines(age.grid, preds, lwd = 2, col = "blue")
```



Another model to investigate is a models with having degrees up to 4, plus degree 9, excluding the intermediate degrees. We will not pursue it here.

Part 6.b)

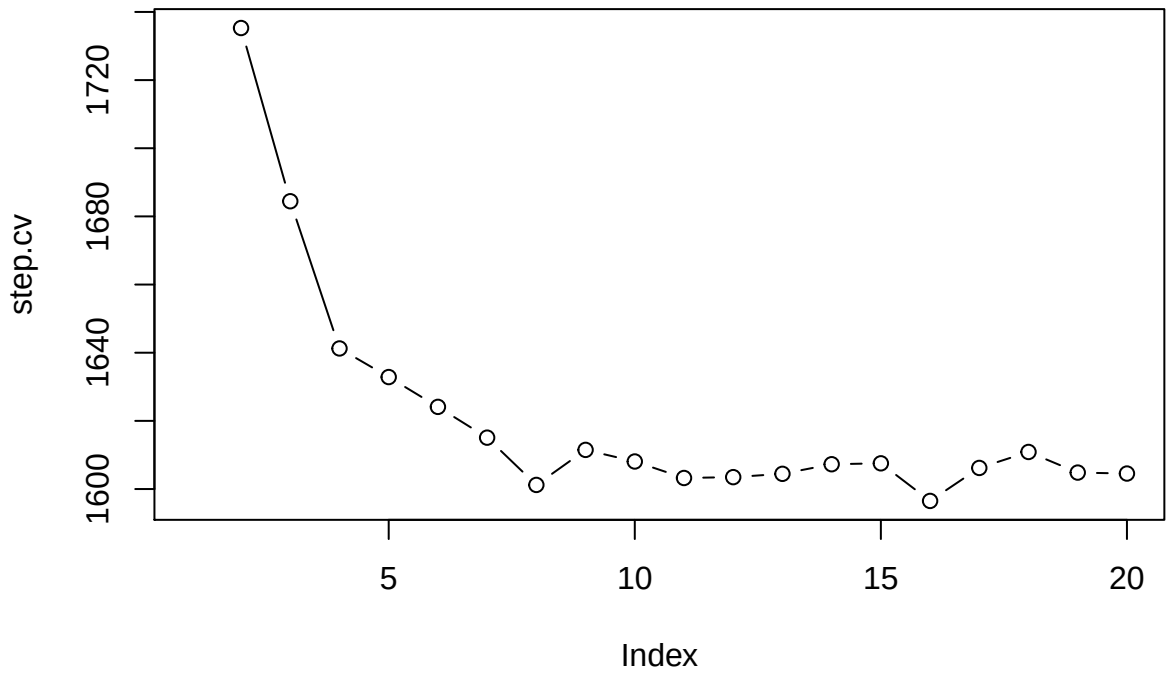
We use step function to predict wage using age, using cross-validation to choose the optimal number of pieces. We first write a function for cross-validataion:

```
library(boot)
library(ISLR)
set.seed(5082)
cv.error <- rep (0,12)
for (i in 2:13){
  Wage$tmp <- cut(Wage$age,i)
  step.fit = glm(wage~tmp, data = Wage)
  cv.error[i] <- cv.glm(Wage ,step.fit, K= 10)$delta [1]
}
cv.error
```

```
##  [1]    0.000 1733.815 1682.731 1637.200 1631.049 1623.069 1613.099
##  [8] 1600.413 1613.127 1603.581 1603.601 1604.730 1602.462
```

```
# rm(step.cv)
max.cuts = 20
# step.fits = list(mode = "list", length = max.cuts)
# names(step.fits) = paste0(1:max.cuts, rep(" Cuts", length = max.cuts))
set.seed(1)
```

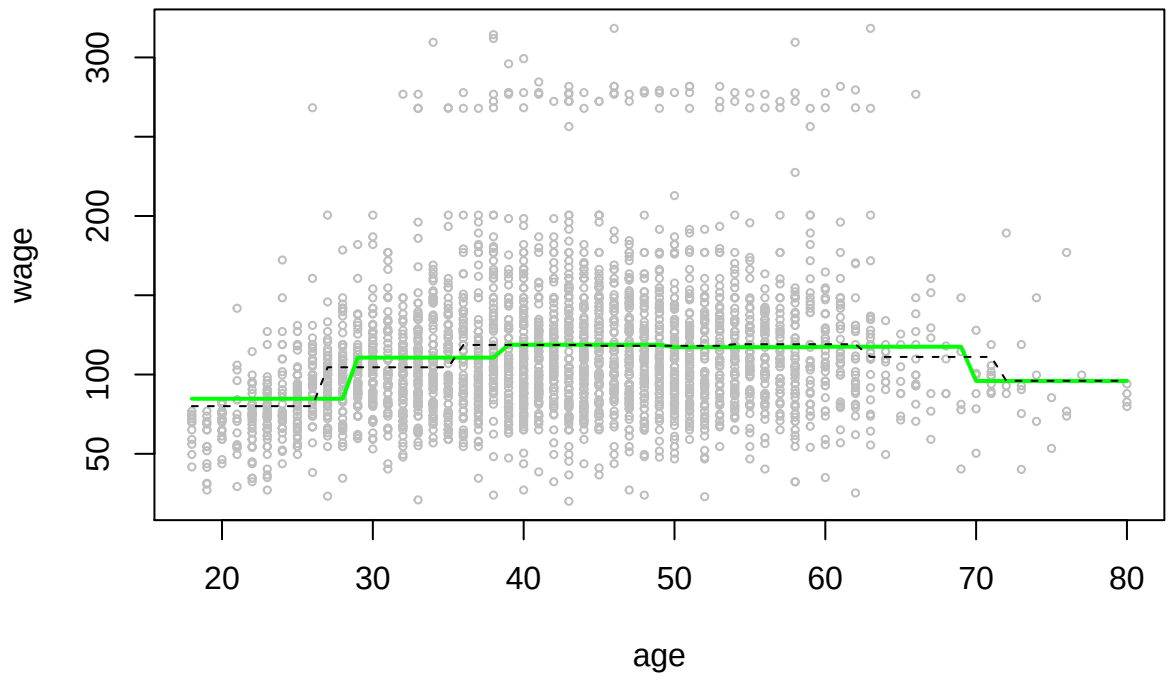
```
step.cv = rep(NA, length = max.cuts)
names(step.cv) = paste0(1:max.cuts, rep(" Cuts", length = max.cuts))
for (i in 2:max.cuts) {
  Wage$tmp = cut(age, breaks = i)
  glm.fit = glm(wage ~ tmp, data = Wage)
  step.cv[i] = cv.glm(data = Wage, glmfit = glm.fit, K = 5)$delta[1]
}
plot(step.cv, type = "b")
```



Up to 20 cuts are used. A general lesson from the code is that to be careful with the formula inside `cv.glm`. We should define it as clear as possible choosing all features from variables available in the dataset. Otherwise, for example, in the case of inline `cut()` in `glm`, when the resulting glm model is fed to the `cv.glm` it results in an error, since the variable is not present in the dataset. That is why we have defined the variable `tmp = cut(age, breaks = i)` instead of using `cut(age, breaks = i)` inline in `glm()`. Another type of error could happen from `cv.glm()` when we use a variable from another dataset in `glm()` and feed the `glm()` object to `cv.glm()`, e.g. here.

Using 6 steps results in the best fit. Here is a plot of the best model in green. For comparison, I draw the model with 7 steps that performs worse according to CV, in dashed black:

```
plot(age, wage, cex = 0.5, col = "grey")
fit.6 = lm(wage ~ cut(age, breaks = 6), data = Wage)
preds.6 = predict(fit.6, newdata = data.frame(age = age.grid))
lines(age.grid, preds.6, lwd = 2, col = "green")
fit.7 = lm(wage ~ cut(age, breaks = 7), data = Wage)
preds.7 = predict(fit.7, newdata = data.frame(age = age.grid))
lines(age.grid, preds.7, lwd = 1, lty = 2)
```



Now we remove the variable `tmp` from the data, since we will not need it anymore:

```
Wage$tmp = NULL
```

Exercise 7

We want to estimate a predictive model for `wage` (we will not focus on `logwage`). We start by learning about the data

```
names(Wage)
```

```
##  [1] "year"       "age"        "sex"        "maritl"     "race"
##  [6] "education"  "region"     "jobclass"   "health"     "health_ins"
## [11] "logwage"    "wage"
```

```
summary(Wage)
```

```
##       year           age               sex                    maritl
##  Min.   :2003   Min.   :18.00   1. Male  :3000   1. Never Married: 648
##  1st Qu.:2004   1st Qu.:33.75   2. Female:   0   2. Married      :2074
##  Median :2006   Median :42.00                    3. Widowed      :  19
##  Mean   :2006   Mean   :42.41                    4. Divorced     : 204
##  3rd Qu.:2008   3rd Qu.:51.00                    5. Separated    :  55
##  Max.   :2009   Max.   :80.00
##
##        race                     education                     region
```

```
##  1. White:2480   1. < HS Grad      :268   2. Middle Atlantic   :3000
##  2. Black: 293   2. HS Grad        :971   1. New England       :   0
##  3. Asian: 190   3. Some College   :650   3. East North Central:   0
##  4. Other:  37   4. College Grad   :685   4. West North Central:   0
##                  5. Advanced Degree:426   5. South Atlantic    :   0
##                                           6. East South Central:   0
##                                           (Other)              :   0
##            jobclass               health      health_ins      logwage
##  1. Industrial :1544   1. <=Good     : 858   1. Yes:2083   Min.   :3.000
##  2. Information:1456   2. >=Very Good:2142   2. No : 917   1st Qu.:4.447
##                                                            Median :4.653
##                                                            Mean   :4.654
##                                                            3rd Qu.:4.857
##                                                            Max.   :5.763
##
##       wage
##  Min.   : 20.09
##  1st Qu.: 85.38
##  Median :104.92
##  Mean   :111.70
##  3rd Qu.:128.68
##  Max.   :318.34
##
```

The variables `region` and `sex` only take one value, so having them in regression would result in the error: **Error in 'contrasts<-'('*tmp*', value = contr.funs[1 + isOF[nn]]) : contrasts can be applied only to factors with 2 or more levels**.
On the other hand, using `. - region - sex` would not work; we would not be able to have these zero variance variables in any formula. To address the issue, we need to create a list of features that excludes these two (the response is also added to this list, since we will create the formula for `lm()`:

```
exclude.feats = c("region", "sex", "logwage", "wage")  # features, hence should exlude wage
include.logic = !(names(Wage) %in% exclude.feats)  # it basically does names(Wage) !%in% exclude.vars
## Define the formula
include.plus = paste(names(Wage[, include.logic]), collapse = " + ")
(form = as.formula(paste("wage ~ ", include.plus)))
```

```
## wage ~ year + age + maritl + race + education + jobclass + health +
##     health_ins
```

```
lm.fit = lm(form, data = Wage)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = form, data = Wage)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -100.33  -18.70   -3.26   13.29  212.79
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 -2.423e+03  6.165e+02  -3.931 8.67e-05 ***
## year                         1.241e+00  3.074e-01   4.037 5.54e-05 ***
## age                          2.707e-01  6.223e-02   4.350 1.41e-05 ***
```

```
## maritl2. Married             1.718e+01  1.720e+00   9.985  < 2e-16 ***
## maritl3. Widowed             2.052e+00  8.005e+00   0.256  0.79774
## maritl4. Divorced            3.967e+00  2.887e+00   1.374  0.16951
## maritl5. Separated           1.153e+01  4.844e+00   2.380  0.01736 *
## race2. Black                -5.096e+00  2.146e+00  -2.375  0.01760 *
## race3. Asian                -2.814e+00  2.603e+00  -1.081  0.27978
## race4. Other                -6.059e+00  5.666e+00  -1.069  0.28505
## education2. HS Grad          7.759e+00  2.369e+00   3.275  0.00107 **
## education3. Some College     1.834e+01  2.520e+00   7.278 4.32e-13 ***
## education4. College Grad     3.124e+01  2.548e+00  12.259  < 2e-16 ***
## education5. Advanced Degree  5.395e+01  2.811e+00  19.190  < 2e-16 ***
## jobclass2. Information       3.571e+00  1.324e+00   2.697  0.00704 **
## health2. >=Very Good         6.515e+00  1.421e+00   4.585 4.72e-06 ***
## health_ins2. No             -1.751e+01  1.403e+00 -12.479  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 34 on 2983 degrees of freedom
## Multiple R-squared:  0.3396, Adjusted R-squared:  0.3361
## F-statistic: 95.89 on 16 and 2983 DF,  p-value: < 2.2e-16
```

Some values of race and marital status are statistically insignificant due to small number of observations:

```
summary(maritl)
```

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

```
summary(race)
```

```
## 1. White 2. Black 3. Asian 4. Other
##     2480      293      190       37
```

Combining some of these categories might help the fit, but we do not pursue that here. Now we fit a nonlinear model, including the additional variables:

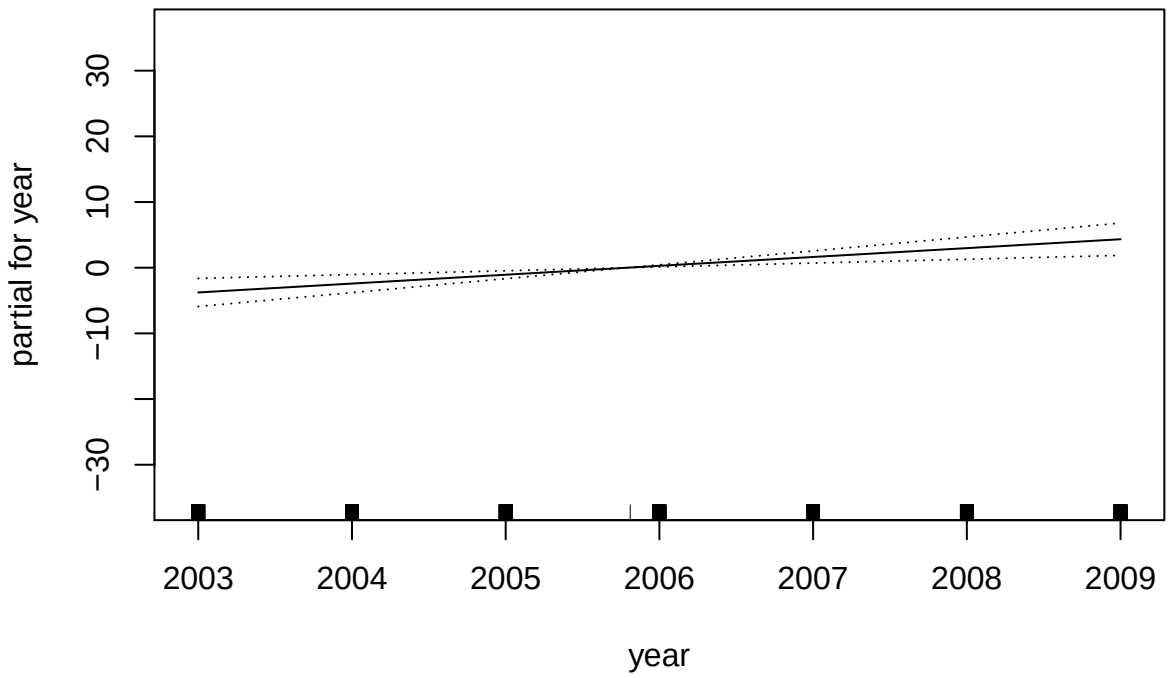
```
library(gam)
```

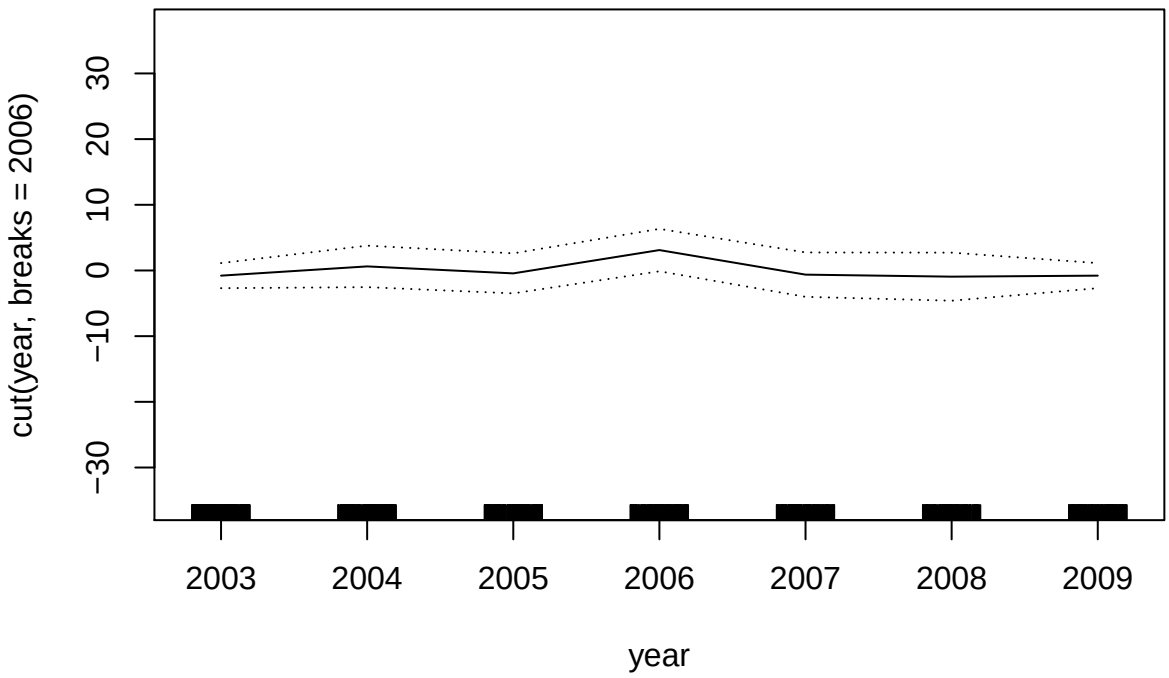
```
## Loading required package: splines
```

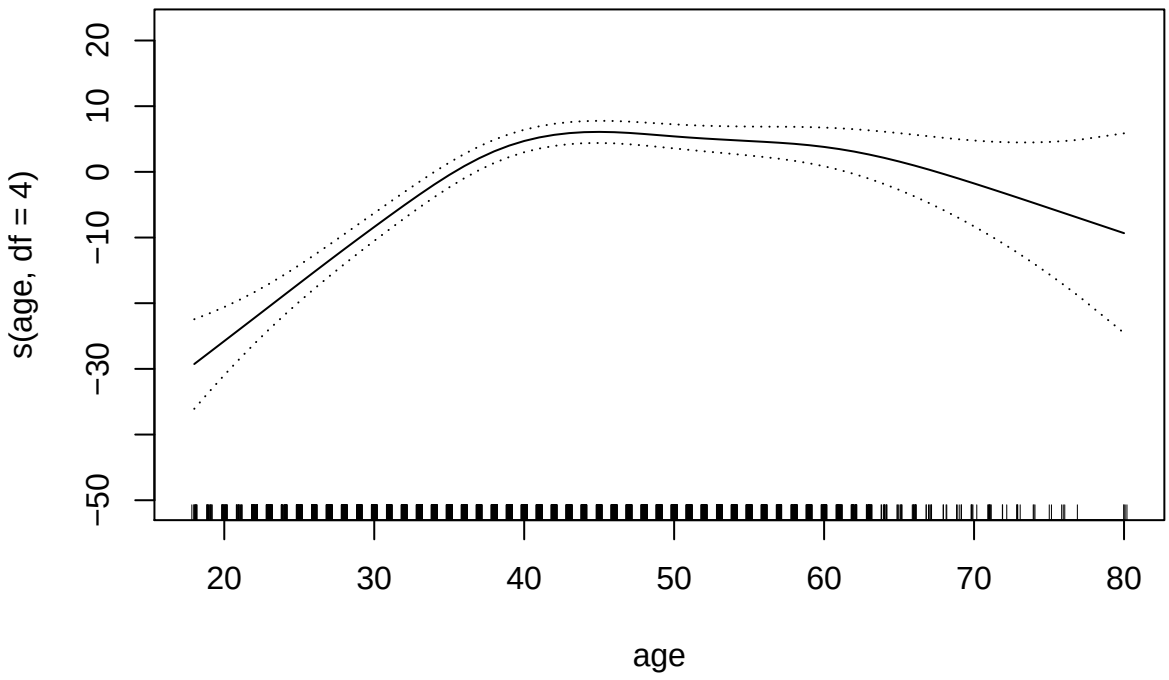
```
## Loading required package: foreach
```

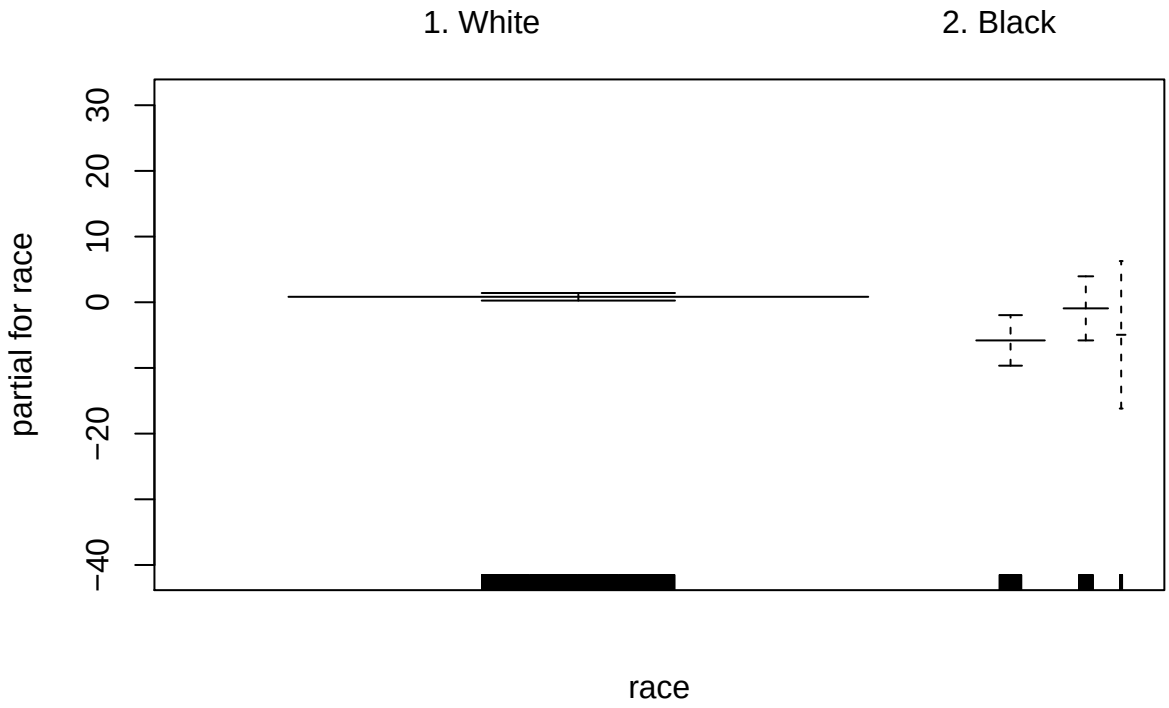
```
## Loaded gam 1.14
```

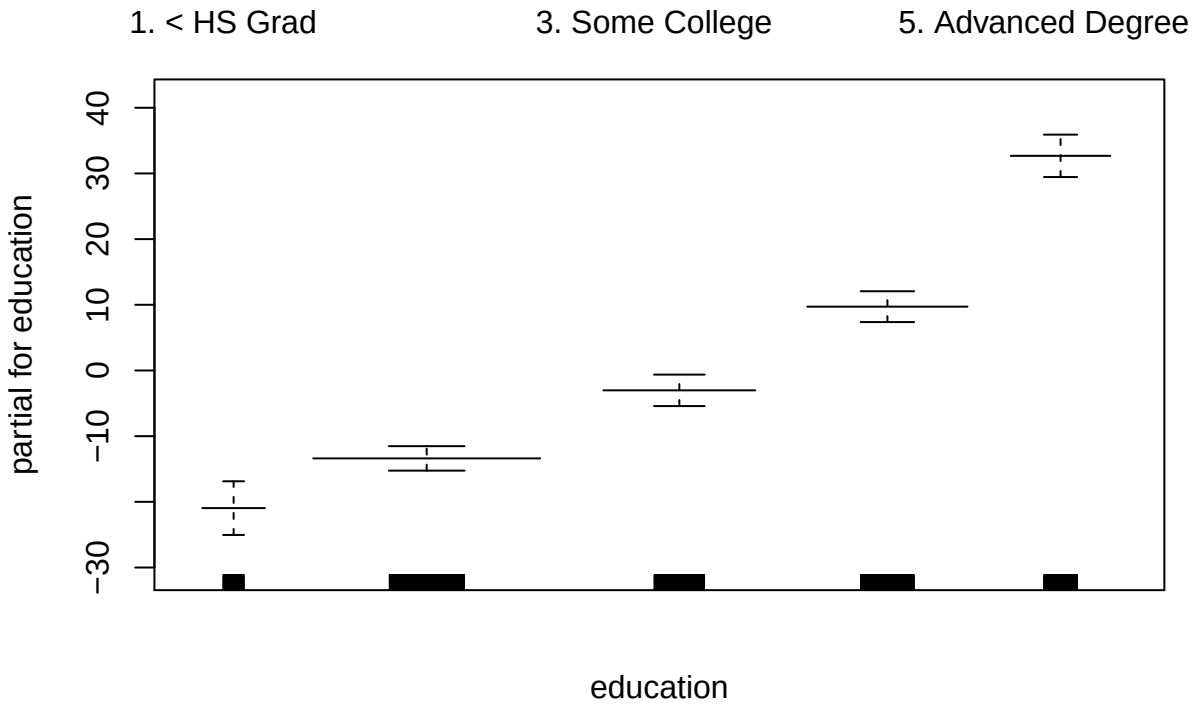
```
gam.1 = gam(wage ~ year + cut(year, breaks = 2006) + s(age, df = 4) + race + education + jobclass +
              health + health_ins, data = Wage)
plot(gam.1, se = T, scale = 72)
```

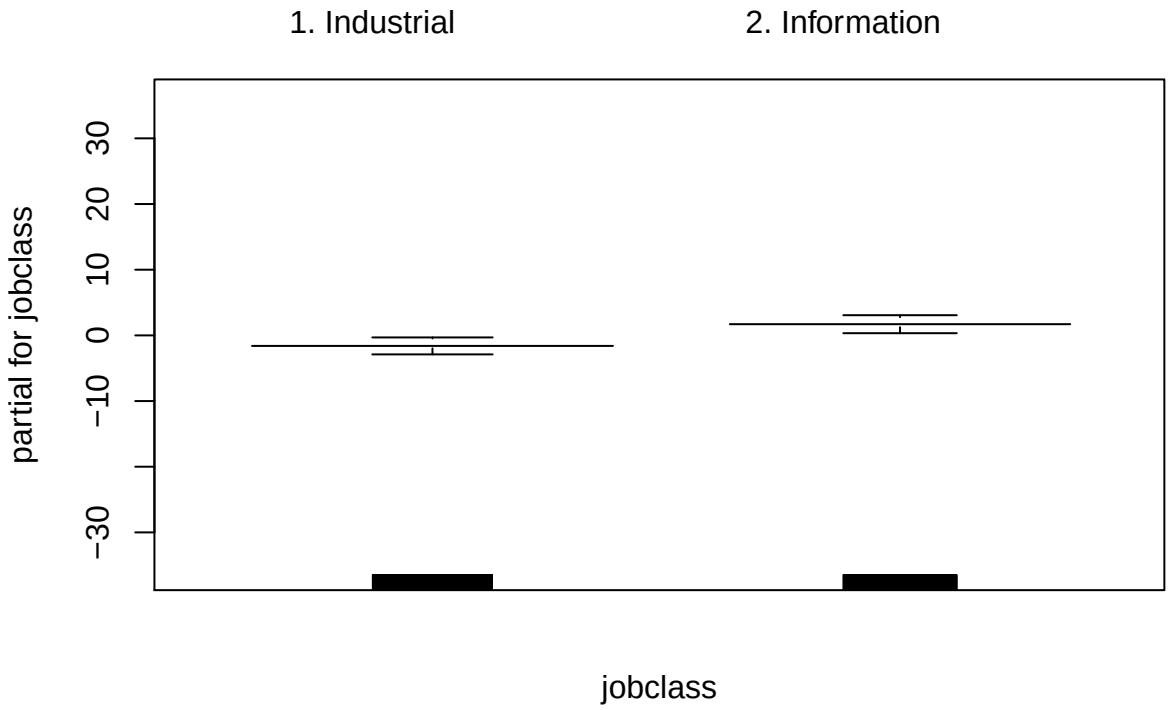

partial for year
30
20
10
0
−10
−30
2003
2004
2005
2006
2007
2008
2009
year

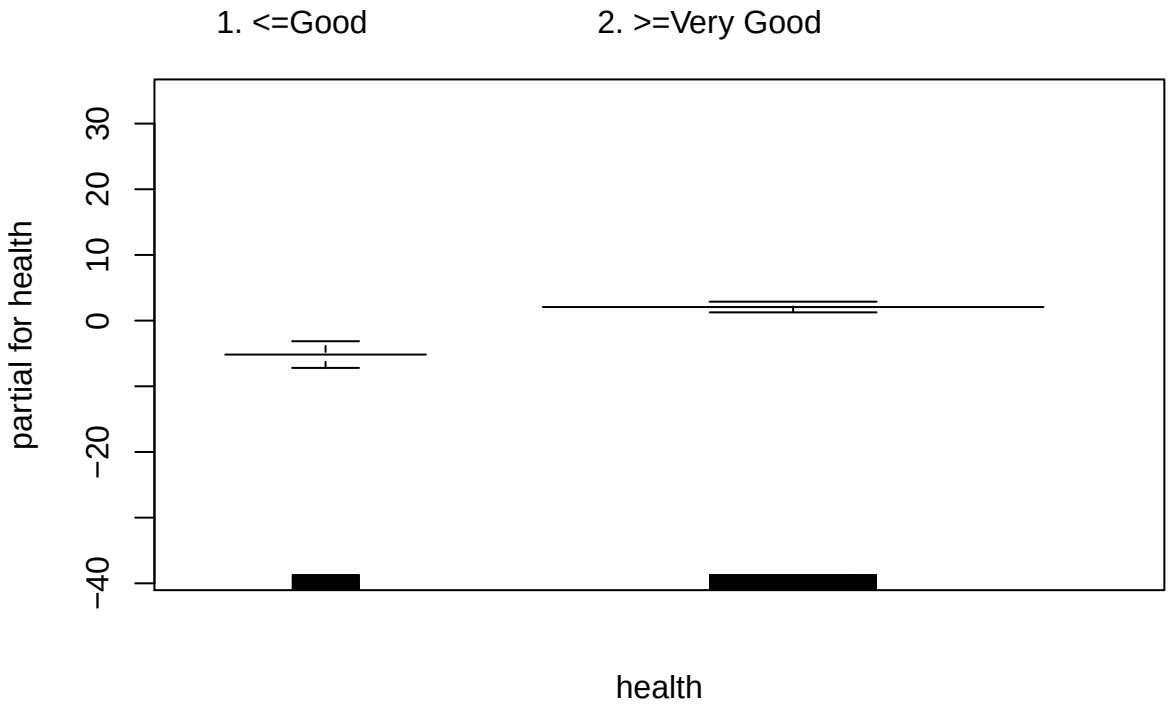

cut(year, breaks = 2006)
30
20
10
0
−10
−30
2003
2004
2005
2006
2007
2008
2009
year

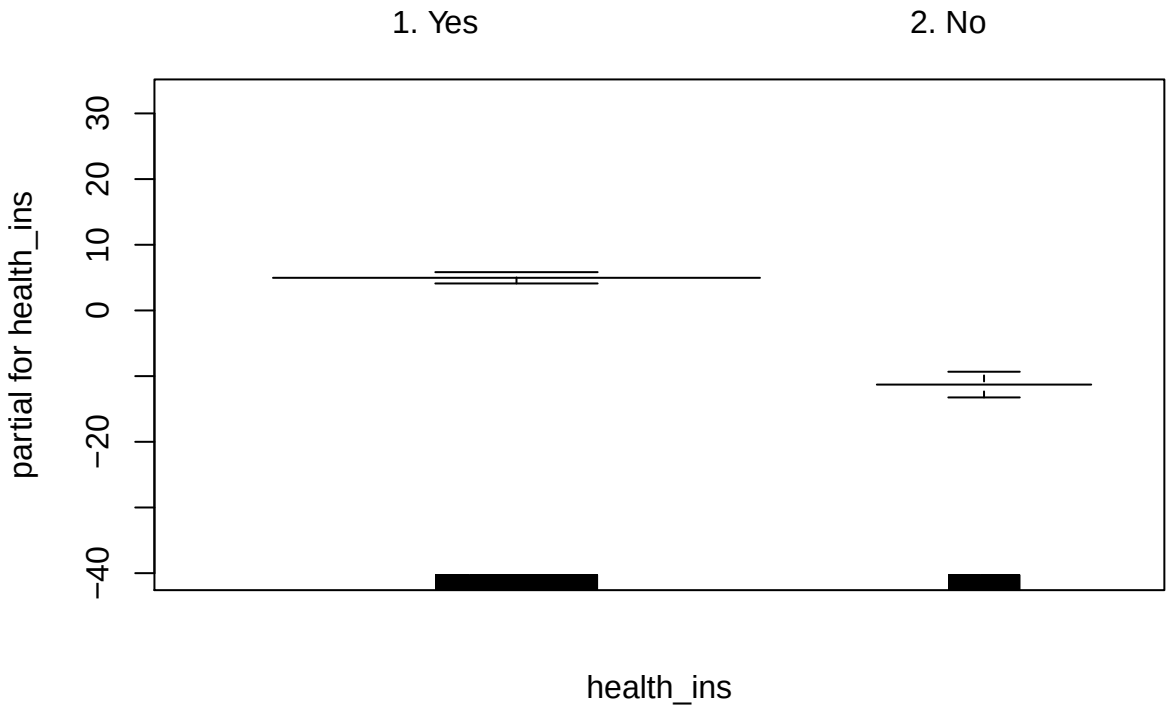

s(age, df = 4)
−50
−30
−10
0
10
20
20
30
40
50
60
70
80
age


1. White
2. Black
partial for race
30
20
10
0
−20
−40
race


1. < HS Grad
3. Some College
5. Advanced Degree
partial for education
−30
−10
0
10
20
30
40
education


1. Industrial
2. Information
partial for jobclass
30
20
10
0
−10
−30
jobclass


1. <=Good
2. >=Very Good
partial for health
30
20
10
0
−20
−40
health



`jobclass` seems to be the least important variable, while `black` might be the only race that makes a difference.

Exercise 8

This question focuse on the `Auto` dataset.

```
library(ISLR)
library(gam)
names(Auto)
```

```
## [1] "mpg"          "cylinders"    "displacement" "horsepower"
## [5] "weight"       "acceleration" "year"         "origin"
## [9] "name"
```

```
lm.fit = lm(mpg ~ . - name, data = Auto)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = mpg ~ . - name, data = Auto)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.5903 -2.1565 -0.1169  1.8690 13.0604
##
## Coefficients:
```

```
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -17.218435   4.644294  -3.707  0.00024 ***
## cylinders     -0.493376   0.323282  -1.526  0.12780
## displacement   0.019896   0.007515   2.647  0.00844 **
## horsepower    -0.016951   0.013787  -1.230  0.21963
## weight        -0.006474   0.000652  -9.929  < 2e-16 ***
## acceleration   0.080576   0.098845   0.815  0.41548
## year           0.750773   0.050973  14.729  < 2e-16 ***
## origin         1.426141   0.278136   5.127 4.67e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.328 on 384 degrees of freedom
## Multiple R-squared:  0.8215, Adjusted R-squared:  0.8182
## F-statistic: 252.4 on 7 and 384 DF,  p-value: < 2.2e-16
```

```
gam.fit = gam(mpg ~ . - name, data = Auto)
summary(gam.fit)
```

```
##
## Call: gam(formula = mpg ~ . - name, data = Auto)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -9.5903 -2.1565 -0.1169  1.8690 13.0604
##
## (Dispersion Parameter for gaussian family taken to be 11.0735)
##
##     Null Deviance: 23818.99 on 391 degrees of freedom
## Residual Deviance: 4252.212 on 384 degrees of freedom
## AIC: 2064.95
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##               Df  Sum Sq Mean Sq   F value    Pr(>F)
## cylinders      1 14403.1 14403.1 1300.6838 < 2.2e-16 ***
## displacement   1  1073.3  1073.3   96.9293 < 2.2e-16 ***
## horsepower     1   403.4   403.4   36.4301 3.731e-09 ***
## weight         1   975.7   975.7   88.1137 < 2.2e-16 ***
## acceleration   1     1.0     1.0    0.0872    0.7679
## year           1  2419.1  2419.1  218.4609 < 2.2e-16 ***
## origin         1   291.1   291.1   26.2912 4.666e-07 ***
## Residuals    384  4252.2    11.1
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The results of `lm()` and `anova()` are drastically different for some variables. The reason for the such a difference is that the t-test is not order-dependent, but the F-test is. Note the difference when we change the order of variables in the tabs below:

ANOVA for Different Orders

The following two tabs show how changing the order of variables in regressions changes the ANOVA results:

ANOVA for mpg ~ weight + displacement

```
summary(gam(mpg ~ weight + displacement, data = Auto))
```

```
##
## Call: gam(formula = mpg ~ weight + displacement, data = Auto)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -12.407  -2.927  -0.357   2.320  16.376
##
## (Dispersion Parameter for gaussian family taken to be 18.4327)
##
##     Null Deviance: 23818.99 on 391 degrees of freedom
## Residual Deviance: 7170.308 on 389 degrees of freedom
## AIC: 2259.773
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##               Df  Sum Sq Mean Sq  F value    Pr(>F)
## weight         1 16497.8 16497.8 895.0283 < 2.2e-16 ***
## displacement   1   150.9   150.9   8.1879  0.004445 **
## Residuals    389  7170.3    18.4
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

ANOVA for mpg ~ displacement + weight

```
summary(gam(mpg ~ displacement + weight, data = Auto))
```

```
##
## Call: gam(formula = mpg ~ displacement + weight, data = Auto)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -12.407  -2.927  -0.357   2.320  16.376
##
## (Dispersion Parameter for gaussian family taken to be 18.4327)
##
##     Null Deviance: 23818.99 on 391 degrees of freedom
## Residual Deviance: 7170.308 on 389 degrees of freedom
## AIC: 2259.773
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##               Df  Sum Sq Mean Sq F value    Pr(>F)
## displacement   1 15440.2 15440.2 837.653 < 2.2e-16 ***
## weight         1  1208.5  1208.5  65.564 7.308e-15 ***
## Residuals    389  7170.3    18.4
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Degrees of Freedom: 4 vs. 3

The following two tabs compare ANOVA with four degrees of freedom with one with three degrees of freedom:

4 Degrees of Freedom

```
gam.fit.4 = gam(mpg ~ s(weight, 4) + s(year, 4) +  s(displacement, 4) +
                s(horsepower, 4) + acceleration + s(cylinders, 4), data = Auto)
summary(gam.fit.4)
```

```
##
## Call: gam(formula = mpg ~ s(weight, 4) + s(year, 4) + s(displacement,
##     4) + s(horsepower, 4) + acceleration + s(cylinders, 4), data = Auto)
## Deviance Residuals:
##        Min         1Q     Median         3Q        Max
## -7.4464460 -1.5459474  0.0005261  1.2532751 11.4326218
##
## (Dispersion Parameter for gaussian family taken to be 6.9577)
##
##     Null Deviance: 23818.99 on 391 degrees of freedom
## Residual Deviance: 2574.354 on 370 degrees of freedom
## AIC: 1896.228
##
## Number of Local Scoring Iterations: 4
##
## Anova for Parametric Effects
##                     Df  Sum Sq Mean Sq   F value    Pr(>F)
## s(weight, 4)         1 14803.7 14803.7 2127.6738 < 2.2e-16 ***
## s(year, 4)           1  3531.2  3531.2  507.5254 < 2.2e-16 ***
## s(displacement, 4)   1   197.6   197.6   28.3949 1.723e-07 ***
## s(horsepower, 4)     1   187.2   187.2   26.9118 3.516e-07 ***
## acceleration         1    55.9    55.9    8.0335  0.004844 **
## s(cylinders, 4)      1     2.7     2.7    0.3861  0.534747
## Residuals          370  2574.4     7.0
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                     Npar Df  Npar F     Pr(F)
## (Intercept)
## s(weight, 4)              3  7.2121 0.0001025 ***
## s(year, 4)                3 20.5716 2.386e-12 ***
## s(displacement, 4)        3 17.9373 6.870e-11 ***
## s(horsepower, 4)          3  6.4626 0.0002836 ***
## acceleration
## s(cylinders, 4)           3 10.8951 7.127e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

3 Degrees of Freedom

```
gam.fit.3 = gam(mpg ~ s(weight, 3) + s(year, 3) +  s(displacement, 3) +
                s(horsepower, 3) + acceleration + s(cylinders, 3), data = Auto)
summary(gam.fit.3)
```

```
##
## Call: gam(formula = mpg ~ s(weight, 3) + s(year, 3) + s(displacement,
##     3) + s(horsepower, 3) + acceleration + s(cylinders, 3), data = Auto)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -7.29381 -1.53949 -0.03217  1.28125 11.70254
##
## (Dispersion Parameter for gaussian family taken to be 7.1297)
##
##     Null Deviance: 23818.99 on 391 degrees of freedom
## Residual Deviance: 2673.646 on 375.0002 degrees of freedom
## AIC: 1901.063
##
## Number of Local Scoring Iterations: 3
##
## Anova for Parametric Effects
##                     Df  Sum Sq Mean Sq   F value    Pr(>F)
## s(weight, 3)         1 15007.5 15007.5 2104.9158 < 2.2e-16 ***
## s(year, 3)           1  3458.7  3458.7  485.1085 < 2.2e-16 ***
## s(displacement, 3)   1   155.7   155.7   21.8329 4.153e-06 ***
## s(horsepower, 3)     1   206.6   206.6   28.9777 1.294e-07 ***
## acceleration         1    39.3    39.3    5.5190   0.01933 *
## s(cylinders, 3)      1     8.8     8.8    1.2329   0.26755
## Residuals          375  2673.6     7.1
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                     Npar Df Npar F     Pr(F)
## (Intercept)
## s(weight, 3)             2 14.030 1.330e-06 ***
## s(year, 3)               2 25.885 2.953e-11 ***
## s(displacement, 3)       2 18.055 3.265e-08 ***
## s(horsepower, 3)         2 11.189 1.908e-05 ***
## acceleration
## s(cylinders, 3)          2 12.380 6.217e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Let's compare the results of the two models estimated above. First we look at the linear parts:

```
names(summary(gam.fit.3))
```

```
##  [1] "call"             "terms"            "anova"
##  [4] "parametric.anova" "dispersion"       "df"
##  [7] "deviance.resid"   "deviance"         "null.deviance"
## [10] "aic"              "iter"             "na.action"
```

```
summary(gam.fit.3)$parametric.anova
```

```
## Anova for Parametric Effects
##                     Df  Sum Sq Mean Sq   F value    Pr(>F)
## s(weight, 3)         1 15007.5 15007.5 2104.9158 < 2.2e-16 ***
```

```
## s(year, 3)           1  3458.7  3458.7  485.1085 < 2.2e-16 ***
## s(displacement, 3)   1   155.7   155.7   21.8329 4.153e-06 ***
## s(horsepower, 3)     1   206.6   206.6   28.9777 1.294e-07 ***
## acceleration         1    39.3    39.3    5.5190   0.01933 *
## s(cylinders, 3)      1     8.8     8.8    1.2329   0.26755
## Residuals          375  2673.6     7.1
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(gam.fit.4)$parametric.anova
```

```
## Anova for Parametric Effects
##                     Df  Sum Sq Mean Sq   F value    Pr(>F)
## s(weight, 4)         1 14803.7 14803.7 2127.6738 < 2.2e-16 ***
## s(year, 4)           1  3531.2  3531.2  507.5254 < 2.2e-16 ***
## s(displacement, 4)   1   197.6   197.6   28.3949 1.723e-07 ***
## s(horsepower, 4)     1   187.2   187.2   26.9118 3.516e-07 ***
## acceleration         1    55.9    55.9    8.0335  0.004844 **
## s(cylinders, 4)      1     2.7     2.7    0.3861  0.534747
## Residuals          370  2574.4     7.0
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The non-linear parts can be compared below:

```
summary(gam.fit.3)$anova
```

```
## Anova for Nonparametric Effects
##                     Npar Df Npar F     Pr(F)
## (Intercept)
## s(weight, 3)             2 14.030 1.330e-06 ***
## s(year, 3)               2 25.885 2.953e-11 ***
## s(displacement, 3)       2 18.055 3.265e-08 ***
## s(horsepower, 3)         2 11.189 1.908e-05 ***
## acceleration
## s(cylinders, 3)          2 12.380 6.217e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(gam.fit.4)$anova
```

```
## Anova for Nonparametric Effects
##                     Npar Df  Npar F     Pr(F)
## (Intercept)
## s(weight, 4)             3  7.2121 0.0001025 ***
## s(year, 4)               3 20.5716 2.386e-12 ***
## s(displacement, 4)       3 17.9373 6.870e-11 ***
## s(horsepower, 4)         3  6.4626 0.0002836 ***
## acceleration
## s(cylinders, 4)          3 10.8951 7.127e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

There is no clear evidence that one models is better than the other, although the non-linear parts seem to be more statistically significant for degree three. I would choose the model with 3 degrees of freedom, which is the simpler model.
Here is another model using local regression:

```r
gam.fit.lo = gam(mpg ~ lo(weight, span = 0.7) + s(year, df = 3) +  s(displacement, df = 3) +
                 s(horsepower, df = 3) + acceleration + s(cylinders, df = 3), data = Auto)
summary(gam.fit.lo)
```

```
##
## Call: gam(formula = mpg ~ lo(weight, span = 0.7) + s(year, df = 3) +
##     s(displacement, df = 3) + s(horsepower, df = 3) + acceleration +
##     s(cylinders, df = 3), data = Auto)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -7.37725 -1.51158 -0.01075  1.25310 11.70831
##
## (Dispersion Parameter for gaussian family taken to be 7.1255)
##
##     Null Deviance: 23818.99 on 391 degrees of freedom
## Residual Deviance: 2675.908 on 375.5416 degrees of freedom
## AIC: 1900.311
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##                              Df  Sum Sq Mean Sq   F value    Pr(>F)
## lo(weight, span = 0.7)    1.00 14966.7 14966.7 2100.4506 < 2.2e-16 ***
## s(year, df = 3)           1.00  3458.0  3458.0  485.3079 < 2.2e-16 ***
## s(displacement, df = 3)   1.00   168.0   168.0   23.5774 1.764e-06 ***
## s(horsepower, df = 3)     1.00   204.9   204.9   28.7578 1.436e-07 ***
## acceleration              1.00    42.8    42.8    6.0067   0.01471 *
## s(cylinders, df = 3)      1.00     8.6     8.6    1.2125   0.27155
## Residuals               375.54  2675.9     7.1
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                          Npar Df Npar F     Pr(F)
## (Intercept)
## lo(weight, span = 0.7)       1.5 17.005 2.443e-06 ***
## s(year, df = 3)              2.0 25.891 2.942e-11 ***
## s(displacement, df = 3)      2.0 19.519 8.585e-09 ***
## s(horsepower, df = 3)        2.0 11.962 9.221e-06 ***
## acceleration
## s(cylinders, df = 3)         2.0 12.600 5.050e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# ?gam.s
```

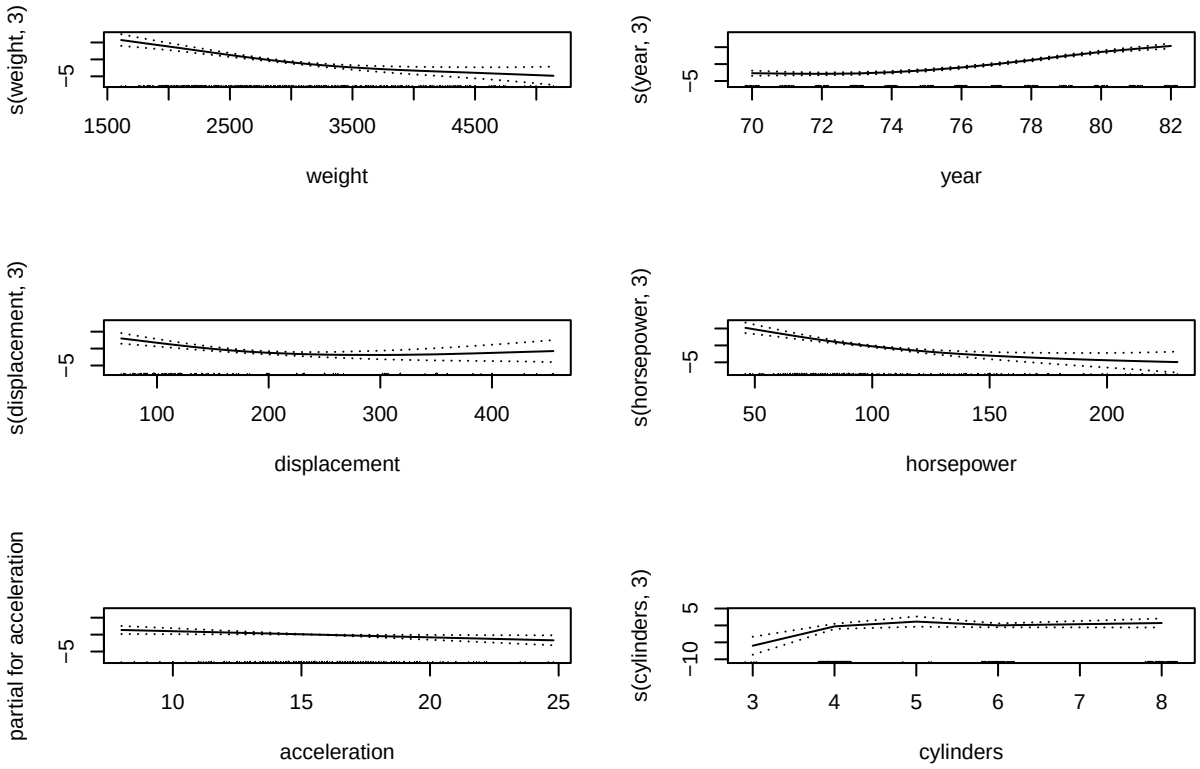
Note that we could not change the specification for the local regression by much. Having other variables in `lo()` results in warnings, and having lower `span` for `weight` results in another warning for convergence not being achieved.
The fitted plots for the three models estimated above can be seen below.

Comparison of Models

The three tabs below, labeled “3 Degrees”, “4 Degrees” and “Local”, compare the corresponding model fits:

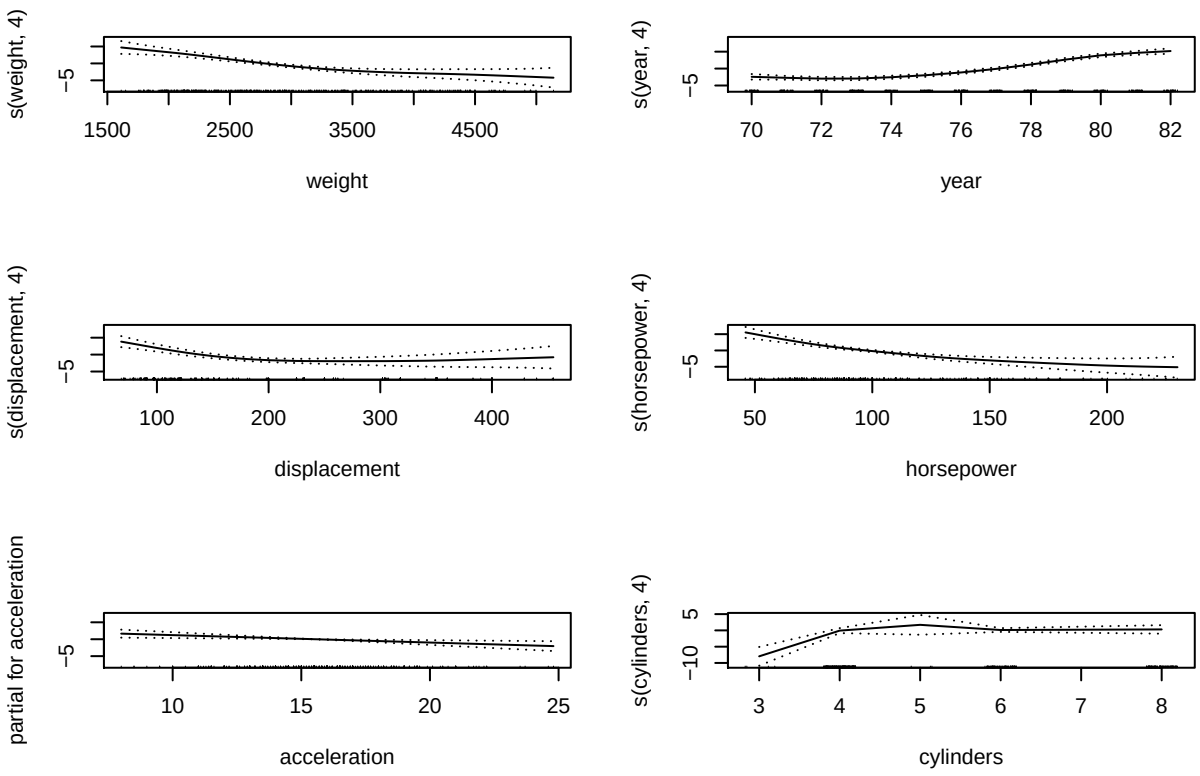
3 Degrees

```
par(mfrow = c(3,2))
plot(gam.fit.3, se = T, scale = 15)
```



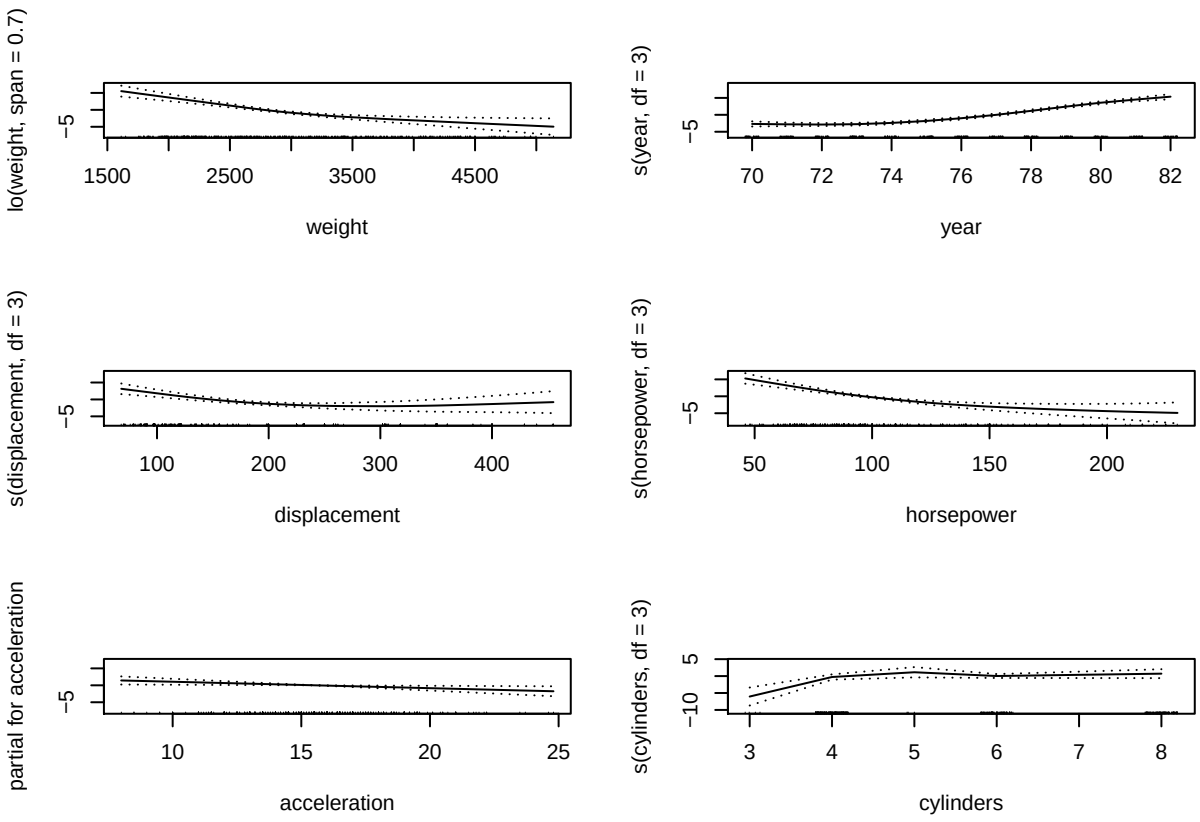
4 Degrees

```
par(mfrow = c(3,2))
plot(gam.fit.4, se = T, scale = 15)
```



Local

```
par(mfrow = c(3,2))
plot(gam.fit.lo, se = T, scale = 15)  # leads to error
```



Exercise 9

In this exercise, we explore the relationship between the distance from five Boston employment centers and nitrogen oxides concentration.

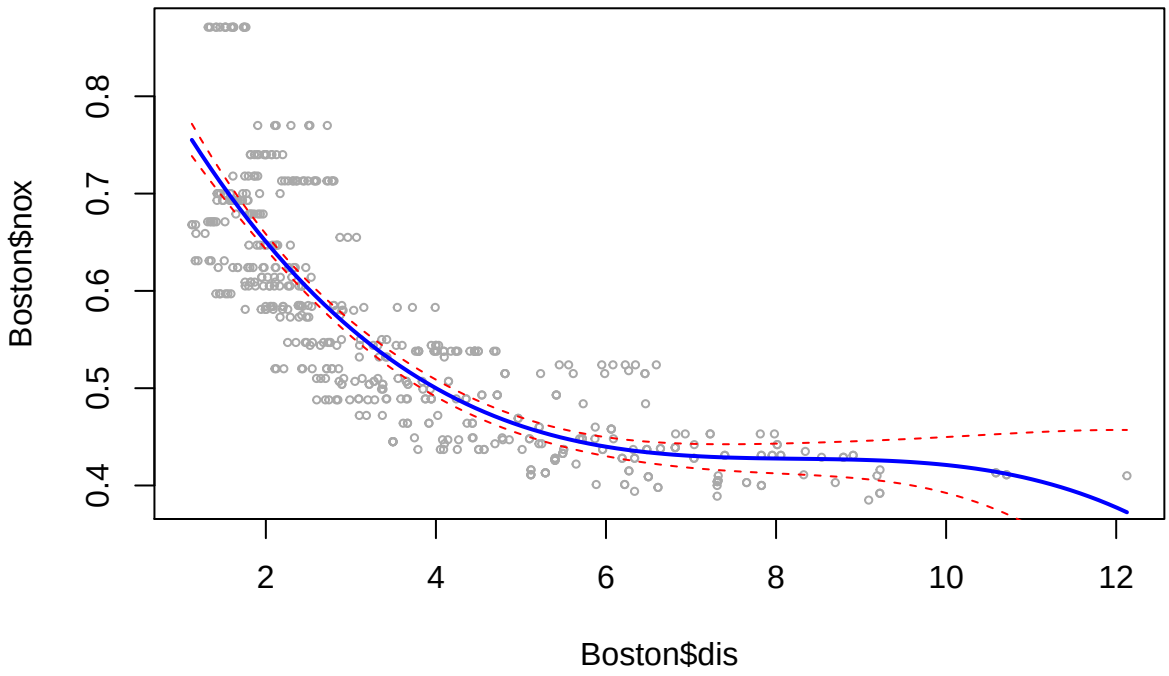
Part 9.a)

```
library(MASS)
cubic.fit = lm(nox ~ poly(dis, degree = 3, raw = T), data = Boston)
summary(cubic.fit)
```

```
##
## Call:
## lm(formula = nox ~ poly(dis, degree = 3, raw = T), 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)                   0.9341281  0.0207076  45.110  < 2e-16 ***
## poly(dis, degree = 3, raw = T)1 -0.1820817  0.0146973 -12.389  < 2e-16 ***
```

```
## poly(dis, degree = 3, raw = T)2  0.0219277  0.0029329   7.476 3.43e-13 ***
## poly(dis, degree = 3, raw = T)3 -0.0008850  0.0001727  -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
```

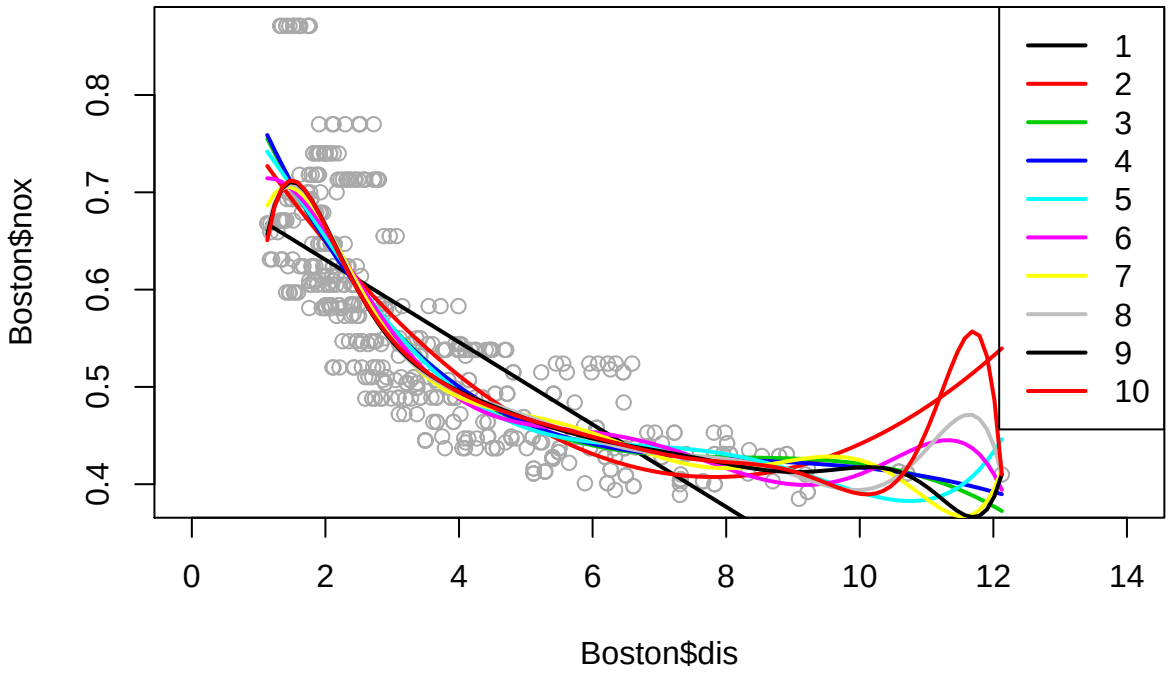
```r
dis.range = range(Boston$dis)
dis.grid = seq(from = dis.range[1], to = dis.range[2], length = 100)
nox.preds = predict(cubic.fit, newdata = data.frame(dis = dis.grid), se.fit = T)
se.bands = cbind(nox.preds$fit - 2*nox.preds$se.fit, nox.preds$fit + 2*nox.preds$se.fit)
plot(Boston$dis, Boston$nox, col = "darkgrey", cex = 0.5)
lines(dis.grid, nox.preds$fit, col = "blue", lwd = 2)
matlines(dis.grid, se.bands, lty = 2, lwd = 1, col = "red")
```



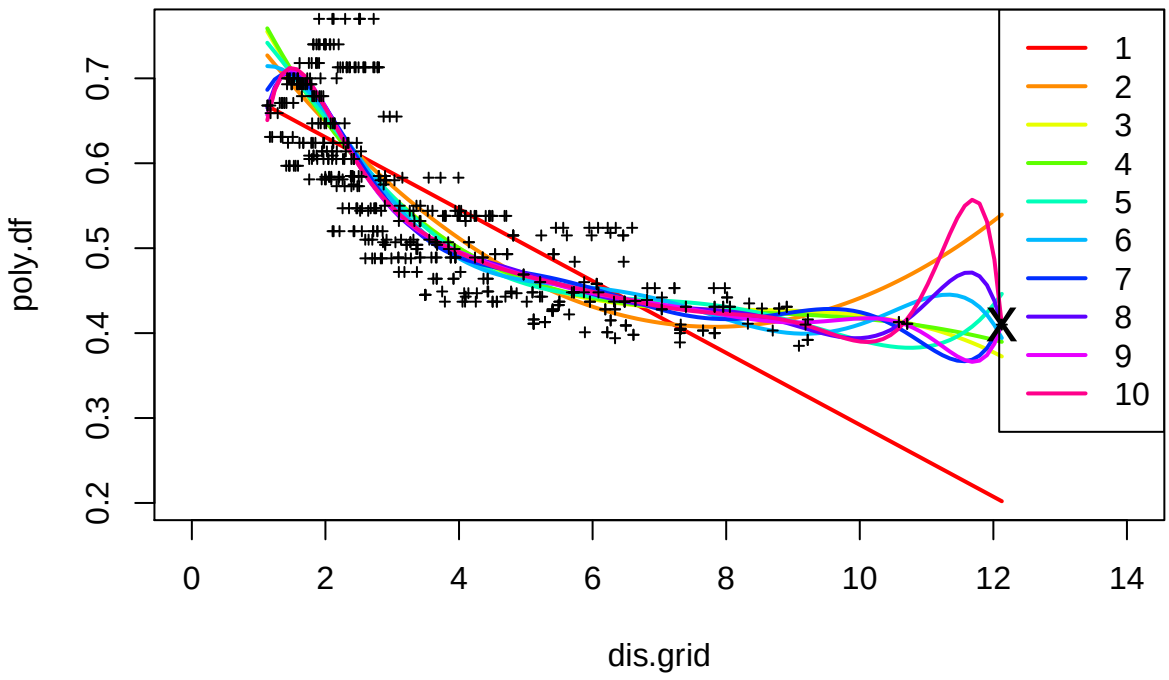
Part 9.b)

```r
max.deg = 10
plot(Boston$dis, Boston$nox, col = "darkgrey", xlim = c(0, 14))
poly.fits = vector(mode = "list", length = max.deg)
poly.fits[[1]] = glm(nox ~ dis, data = Boston)
preds = predict(poly.fits[[1]], newdata = data.frame(dis = dis.grid), se.fit = T)
poly.df = data.frame(preds$fit)
```

```
for (i in 2:max.deg) {
  poly.fits[[i]] = glm(nox ~ poly(dis, degree = i), data = Boston)
  preds = predict(poly.fits[[i]], newdata = data.frame(dis = dis.grid), se.fit = T)
  se.lband = preds$fit - 2*preds$se.fit
  se.rband = preds$fit + 2*preds$se.fit
  poly.df = cbind(poly.df, preds$fit)
}
matlines(dis.grid, poly.df, col = 1:max.deg, type = "l", lwd = 2, lty = 1)
legend("topright", legend = 1:10, col = 1:10, lwd = 2)
```




```
col.rb = rainbow(max.deg + 1)[-5]  # since colors 4 and 5 are indistinguishable.
matplot(dis.grid, poly.df, col = col.rb[1:max.deg], type = "l", lwd = 2, lty = 1, xlim = c(0, 14))
points(Boston$dis, Boston$nox, pch = 3 , col = "black", cex = 0.5)
points(max(Boston$dis), Boston$nox[which.max(Boston$dis)], pch = "x", cex = 2)
legend("topright", legend = 1:10, col = col.rb[1:max.deg], lwd = 2)
```




```
### Errrors:
## cbind(poly.df, preds)
## matplot(poly.df, col = 1:max.deg, type = c("b"), pch = 1, lty = 1): type= "b" weired look, what abou
## set ylim = c(0,14)
## legend("topright", legend = 1:10, col = 1:10): no lines
```

We use `glm()` instead of `lm()` to be able to use `cv.glm` later on. We also compute RSS, but we wouldn't access to `residuals` as in `lm()` to compute the RSS, so we estimate `lm()`:

```
# rss
poly.rss = matrix(NA, nrow = max.deg, ncol = 2)
for (i in 1:max.deg) {
  lm.fit = lm(nox ~ poly(dis, degree = i), Boston)
  # poly.sum = summary(poly.fits[[i]])
  poly.rss[i, 1] = sum(summary(lm.fit)$residuals^2)
  poly.rss[i, 2] = summary(lm.fit)$sigma^2*summary(lm.fit)$df[2]  # an equivalent way of computing rss
}
poly.rss
```

```
##           [,1]     [,2]
##  [1,] 2.768563 2.768563
##  [2,] 2.035262 2.035262
##  [3,] 1.934107 1.934107
##  [4,] 1.932981 1.932981
##  [5,] 1.915290 1.915290
##  [6,] 1.878257 1.878257
##  [7,] 1.849484 1.849484
```

```
##  [8,] 1.835630 1.835630
##  [9,] 1.833331 1.833331
## [10,] 1.832171 1.832171
```

There a few noteworthy points:

- The plot argument `col` can be put equal to `1:10` or `rainbow(10)[1:10]`, where the color combination can be see above (the `rainbow()` generated colors in the code are a bit different, since fifth color is omitted).
- The combo of `matplot()` and `points()` is more effective in showing the data than `plot()` and `matlines()`.
  - For `matplot()`, we need to set the `type` argument (and not `lty`), e.g. to `l`; otherwise, the default arguments would make it barely recognizable.
- The odd behavior of the plots at the right boundary can be partly associated with the point with larget `dis`, shown by "X" mark: the fit is free to move on the left side of the "X" mark, since there are no data points to match, say when `dis.grid` is between 11 and 12, but then the fit comes back to match the "X"-marked point.
- The RSS decreases as we increase the degree, although after degree 8 the change is trivial.

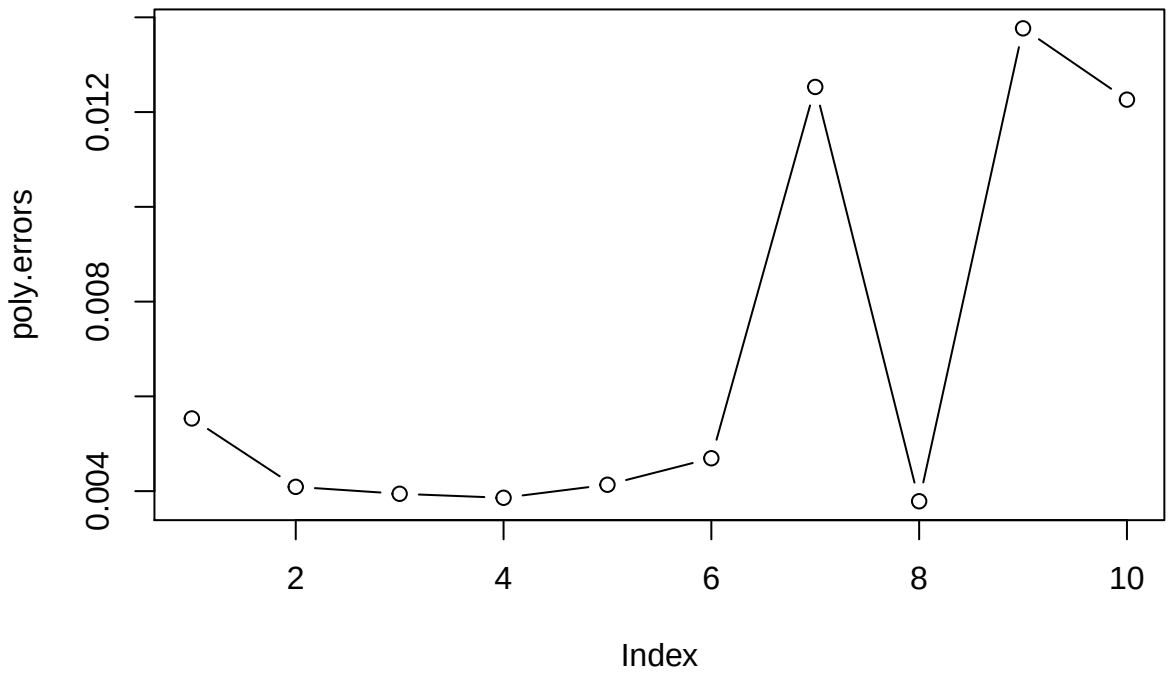
Part 9.c)

Now we use cross-validation to find the optimal degree.

```
library(boot)
poly.errors = rep(NA, length = max.deg)
for (i in 1:max.deg) {
  poly.errors[i] = cv.glm(data = Boston, glmfit = poly.fits[[i]], K = 5)$delta[1]
}
poly.errors
```

```
##  [1] 0.005534207 0.004090414 0.003944247 0.003859796 0.004134875
##  [6] 0.004694080 0.012530515 0.003786942 0.013767111 0.012262580
```

```
plot(poly.errors, type = "b")
```



Degree 4 has the lowest C.V. error, but it is too close to 3. So we choose degree 3.

Part 9.d)

We use `bs()` below.

```
bs.fit = lm(nox ~ bs(dis, df = 4), data = Boston)
summary(bs.fit)
```

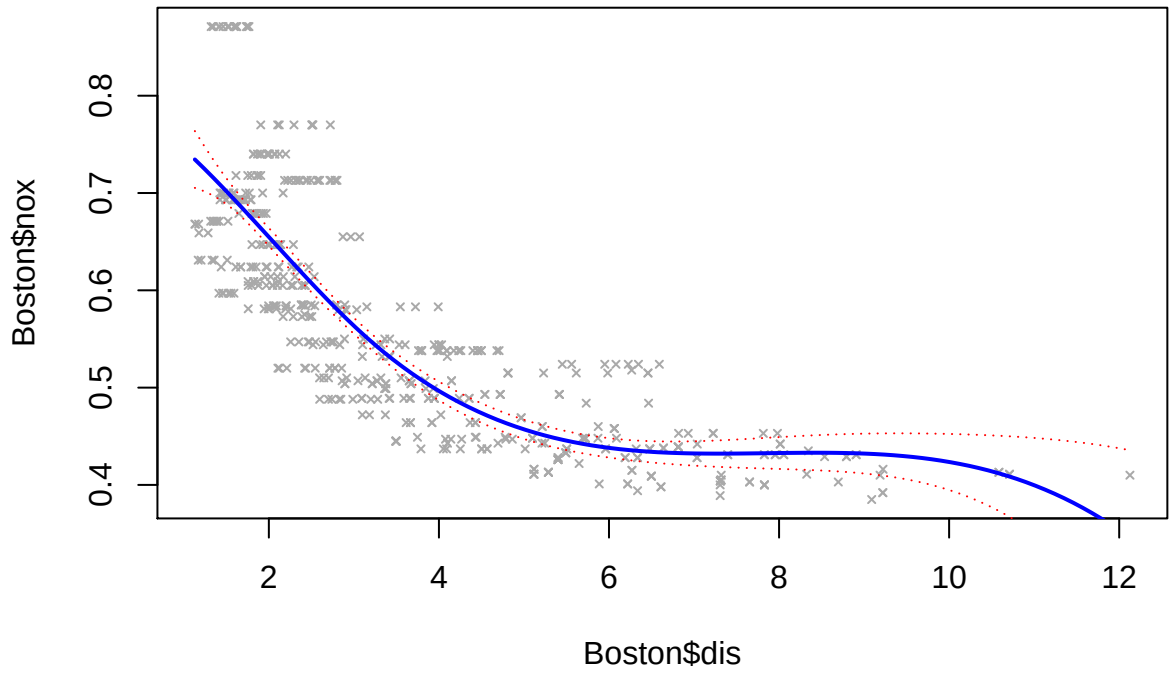
```
##
## Call:
## lm(formula = nox ~ bs(dis, df = 4), data = Boston)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.124622 -0.039259 -0.008514  0.020850  0.193891
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)       0.73447    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
```

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

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

```
# predict fit and its error
preds = predict(bs.fit, newdata = data.frame(dis = dis.grid),
                se.fit = T)
# compute bands
se.bands = cbind(preds$fit - 2*preds$se.fit, preds$fit + 2*preds$se.fit)
# plot the data
plot(Boston$dis, Boston$nox, cex = 0.5, col = "darkgrey", pch = 4)
# draw fit and its bands
lines(dis.grid, preds$fit, col = "blue", lwd = 2)
matlines(dis.grid, se.bands, col = "red", lty = 3)
```



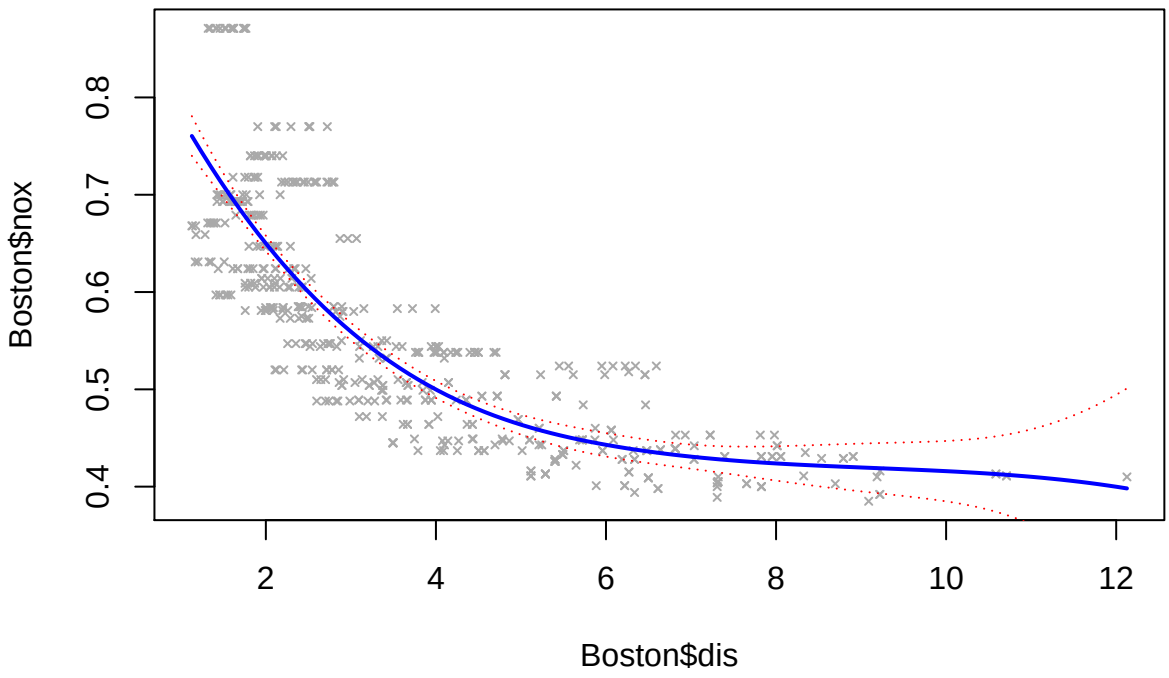
The right boundary does not look all right. We can change the knot to make it a bit better.

```
bs.fit = lm(nox ~ bs(dis, knots = 6), data = Boston)
summary(bs.fit)
```

```
##
## Call:
## lm(formula = nox ~ bs(dis, knots = 6), data = Boston)
##
```

```
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.12387 -0.04012 -0.01033  0.02308  0.19446
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)          0.76037    0.01018  74.667  < 2e-16 ***
## bs(dis, knots = 6)1 -0.23672    0.02321 -10.200  < 2e-16 ***
## bs(dis, knots = 6)2 -0.36177    0.02548 -14.200  < 2e-16 ***
## bs(dis, knots = 6)3 -0.33337    0.04044  -8.244 1.47e-15 ***
## bs(dis, knots = 6)4 -0.36220    0.05105  -7.095 4.45e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06208 on 501 degrees of freedom
## Multiple R-squared:  0.7152, Adjusted R-squared:  0.7129
## F-statistic: 314.6 on 4 and 501 DF,  p-value: < 2.2e-16
```

```r
# predict fit and its error
preds = predict(bs.fit, newdata = data.frame(dis = dis.grid),
                se.fit = T)
# compute bands
se.bands = cbind(preds$fit - 2*preds$se.fit, preds$fit + 2*preds$se.fit)
# plot the data
plot(Boston$dis, Boston$nox, cex = 0.5, col = "darkgrey", pch = 4)
# draw fit and its bands
lines(dis.grid, preds$fit, col = "blue", lwd = 2)
matlines(dis.grid, se.bands, col = "red", lty = 3)
```

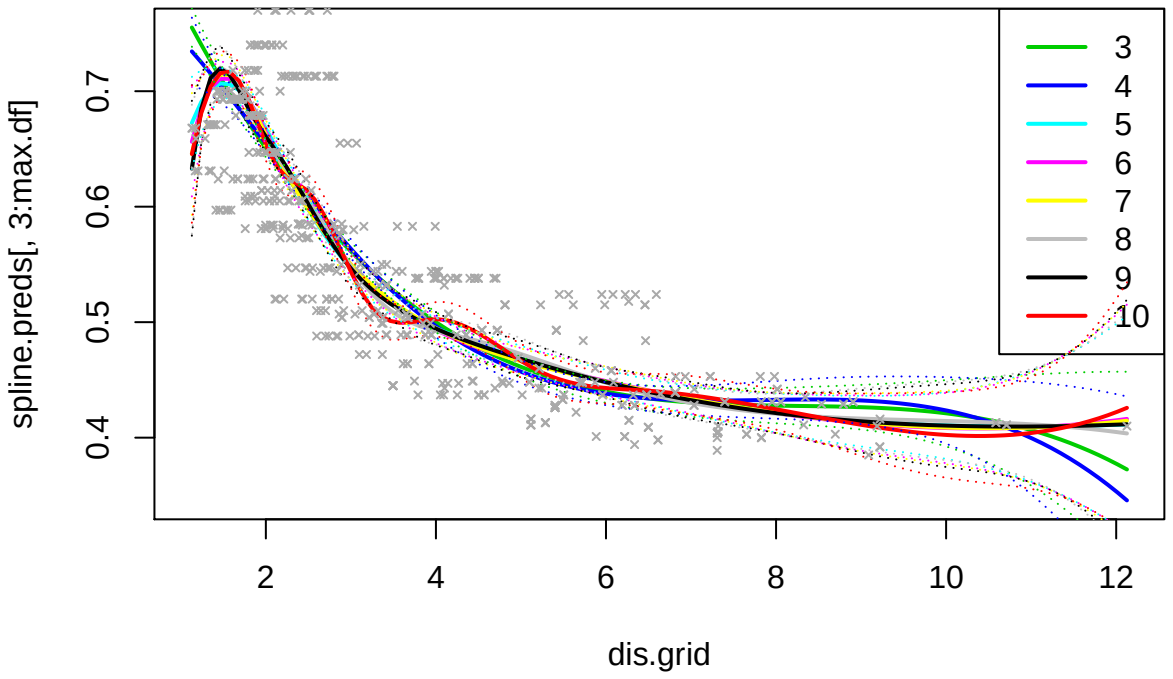


Part 9.e)

We first fit for a range of splines.

```
## fit for range
max.df = 10
# initialize
spline.fits = vector(mode = "list", length = max.df)
spline.preds = matrix(NA, length(dis.grid), max.df)
spline.lbands = matrix(NA, length(dis.grid), max.df)
spline.rbands = matrix(NA, length(dis.grid), max.df)
# estimate
for (i in 3:max.df) {
  spline.fits[[i]] = glm(nox ~ bs(dis, df = i), data = Boston)  # glm for CV
  # predict fit and se bands
  preds = predict(spline.fits[[i]], newdata = data.frame(dis = dis.grid),
                  se.fit = T)
  spline.preds[, i] = preds$fit
  spline.lbands[, i] = preds$fit - 2*preds$se.fit
  spline.rbands[, i] = preds$fit + 2*preds$se.fit
}

## plot the fits (starting df = 3)
matplot(dis.grid, spline.preds[, 3:max.df], type = "l", lty = 1, col = 3:max.df, lwd = 2)
legend("topright", legend = 3:10, col = 3:max.df, lwd = 2)
```

```
matlines(dis.grid, spline.lbands[, 3:max.df], col = 3:max.df, lwd = 1, lty = 3)
matlines(dis.grid, spline.rbands[, 3:max.df], col = 3:max.df, lwd = 1, lty = 3)
# data plot
points(Boston$dis, Boston$nox, pch = 4, col = "darkgrey", cex = 0.5)
```



Next, we report the corresponding RSS's.

```
## RSS: it is easiest to use lm(). But here we want to try glm(); but it is safest to produce all bs() v
# initialize
spline.rss = rep(NA, length = 10)
# compute rss
for (i in 3:10) {
  mat = model.matrix(nox ~ bs(dis, df = i), data = Boston)
  mat.names = names(coef(spline.fits[[i]]))

  summary(spline.fits[[i]])$coefficients  [, 1]
  coef.names = dimnames(summary(spline.fits[[i]])$coefficients)[[1]]
  preds = mat[, coef.names] %*% summary(spline.fits[[i]])$coefficients[, 1]

  resids = Boston$nox - preds
  spline.rss[i] = sum(resids^2)
}
# print rss
spline.rss
```

```
##  [1]       NA       NA 1.934107 1.922775 1.840173 1.833966 1.829884
##  [8] 1.816995 1.825653 1.792535
```

```
poly.rss[, 1]
```

```
##  [1] 2.768563 2.035262 1.934107 1.932981 1.915290 1.878257 1.849484
##  [8] 1.835630 1.833331 1.832171
```

Part 9.f)

The training MSE is larger than splines with the same degrees of freedom.

```
## CV
set.seed(1)
library(boot)
#initialize
spline.errors = rep(NA, length = max.df)
# compute cv
for (i in 3:max.df) {
  spline.errors[i] = cv.glm(data = Boston, glmfit = spline.fits[[i]], K = 5)$delta[1]
}
```

```
## Warning in bs(dis, degree = 3L, knots = numeric(0), Boundary.knots =
## c(1.1296, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = numeric(0), Boundary.knots =
## c(1.1296, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## 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 = 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 = structure(3.3603, .Names =
## "50%"), : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = structure(3.3603, .Names =
## "50%"), : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = structure(3.20745, .Names =
## "50%"), : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = structure(3.20745, .Names =
## "50%"), : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = structure(c(2.29316666666667,
## 4.1139: some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = structure(c(2.29316666666667,
```

```
## 4.1139: some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = structure(c(2.34846666666667,
## 4.14543333333333: some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = structure(c(2.34846666666667,
## 4.14543333333333: some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = structure(c(2.1069, 3.1323, 5.1167:
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = structure(c(2.1069, 3.1323, 5.1167:
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = structure(c(2.1036, 3.2797, 5.2447:
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = structure(c(2.1036, 3.2797, 5.2447:
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = structure(c(1.96062, 2.65254,
## 3.8303, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = structure(c(1.96062, 2.65254,
## 3.8303, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = structure(c(1.9879, 2.7175,
## 3.945, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = structure(c(1.9879, 2.7175,
## 3.945, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = structure(c(1.82461666666667,
## 2.35953333333333, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = structure(c(1.82461666666667,
## 2.35953333333333, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = structure(c(1.864, 2.4214,
## 3.2157, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = structure(c(1.864, 2.4214,
## 3.2157, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = structure(c(1.78564285714286,
## 2.18685714285714, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = structure(c(1.78564285714286,
## 2.18685714285714, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = structure(c(1.78741428571429,
## 2.15565714285714, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = structure(c(1.78741428571429,
## 2.15565714285714, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = structure(c(1.7266, 2.087875,
## 2.505025, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases
```

```
## Warning in bs(dis, degree = 3L, knots = structure(c(1.7266, 2.087875,
## 2.505025, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases
```

```
## Warning in bs(dis, degree = 3L, knots = structure(c(1.7572375, 2.087875, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = structure(c(1.7572375, 2.087875, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

The warnings above arise because some of the test data lie outside the range of the training set. This could be a problem when ranges are really different. The cross-validation test MSE for degrees of freedom between 3 and 10 correspond to the 3rd to 10th elements of the vector below:

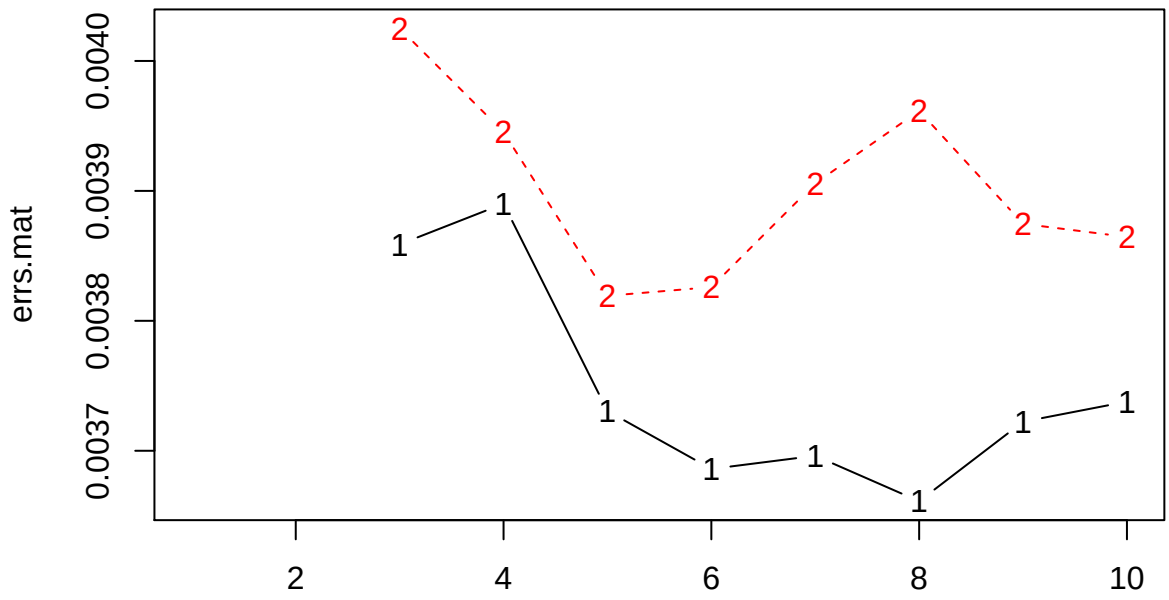
```
# print cv
spline.errors
```

```
##  [1]          NA          NA 0.003858828 0.003890122 0.003730621
##  [6] 0.003686023 0.003696153 0.003661157 0.003722342 0.003737797
```

Since the largest value of `dis` is far off from other points, we see if this might affect the cross-validation result. Hence, we compute the cross-validation errors again after excluding this point (we suppress the warnings this time though):

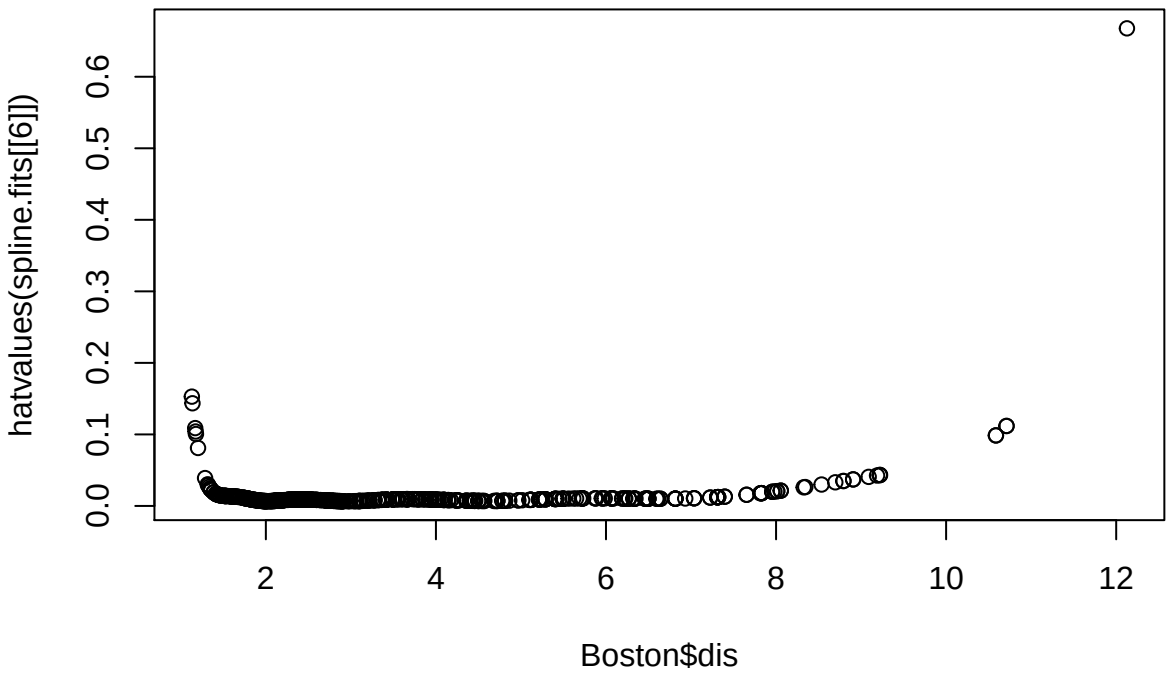
```
## CV
set.seed(1)
#initialize
spline.errors.trunc = rep(NA, length = max.df)
# the max dis point exluded
Boston.trunc = Boston[-which.max(Boston$dis), ]
# compute cv
for (i in 3:max.df) {
  spline.errors.trunc[i] = cv.glm(data = Boston.trunc, glmfit = spline.fits[[i]], K = 5)$delta[1]
}
# print cv

cat("\n")
```

```
errs.mat = cbind(spline.errors, spline.errors.trunc)
matplot(errs.mat, type = "b")
```

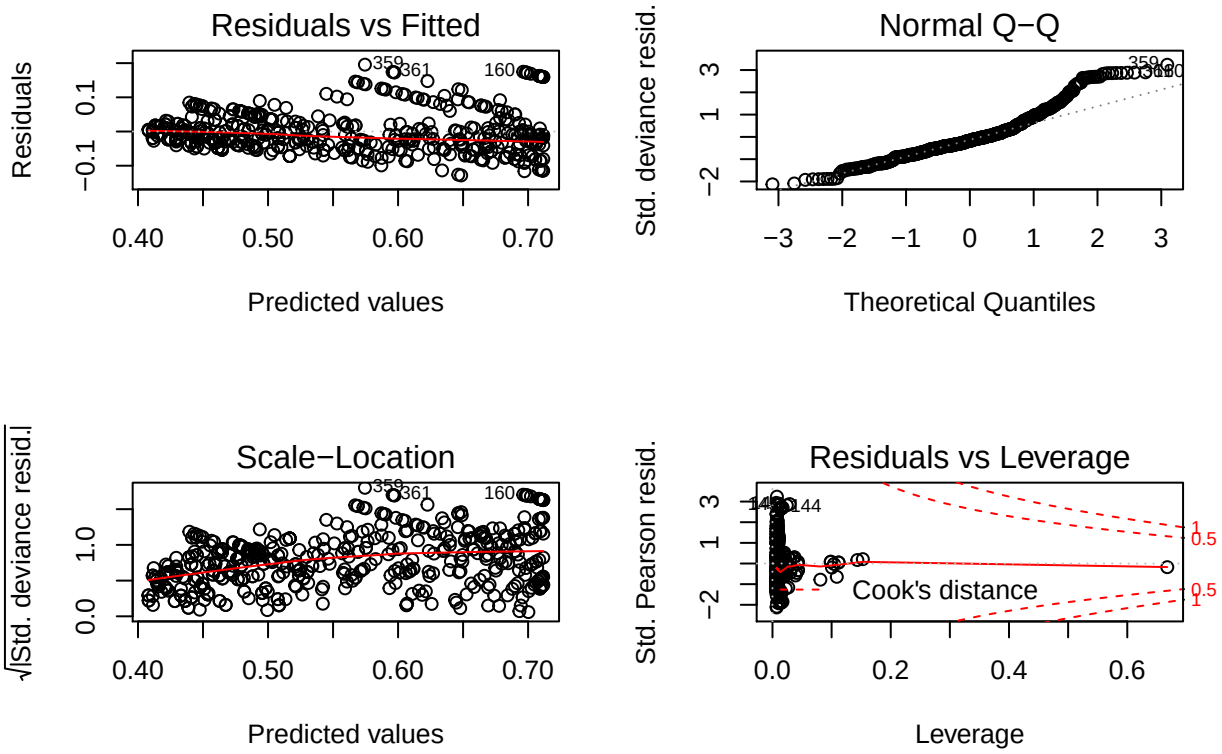


The first line above depicts the CV test error for the full sample, while the second depicts it when the single rightmost point is omitted. The first line might lead to the choice of 6, where the second leads to the choice of 5. Omitting the point results in worse fit for all models. This seems to corroborate that this point might be a leverage:

```
plot(Boston$dis, hatvalues(spline.fits[[6]]))
```




```
par(mfrow = c(2, 2))
plot(spline.fits[[6]])
```



The point is not much of an outlier, but is very far off in the bottom-right figure above, indicating it is a leverage. Whether it is included in the test or training set, hence, could potentially make a large difference. Hence, we rely on the results of the trucated dataset, where this single observation is omitted, and choose 5 degrees of freedom.

Exercise 10

This question asks for predicting (out-of-state) college tuition using the `College` dataset. The strategy in this question is to use subset selection on a linear model, then use `gam()` to fit a nonlinear model using the variables selected.

```
# Set-Up
library(ISLR)
names(College)
```

```
##  [1] "Private"     "Apps"        "Accept"      "Enroll"      "Top10perc"
##  [6] "Top25perc"   "F.Undergrad" "P.Undergrad" "Outstate"    "Room.Board"
## [11] "Books"       "Personal"    "PhD"         "Terminal"    "S.F.Ratio"
## [16] "perc.alumni" "Expend"      "Grad.Rate"
```

```
str(College)
```

```
## 'data.frame':    777 obs. of  18 variables:
##  $ Private    : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Apps       : num  1660 2186 1428 417 193 ...
##  $ Accept     : num  1232 1924 1097 349 146 ...
##  $ Enroll     : num  721 512 336 137 55 158 103 489 227 172 ...
```

```
##  $ Top10perc  : num  23 16 22 60 16 38 17 37 30 21 ...
##  $ Top25perc  : num  52 29 50 89 44 62 45 68 63 44 ...
##  $ F.Undergrad: num  2885 2683 1036 510 249 ...
##  $ P.Undergrad: num  537 1227 99 63 869 ...
##  $ Outstate   : num  7440 12280 11250 12960 7560 ...
##  $ Room.Board : num  3300 6450 3750 5450 4120 ...
##  $ Books      : num  450 750 400 450 800 500 500 450 300 660 ...
##  $ Personal   : num  2200 1500 1165 875 1500 ...
##  $ PhD        : num  70 29 53 92 76 67 90 89 79 40 ...
##  $ Terminal   : num  78 30 66 97 72 73 93 100 84 41 ...
##  $ S.F.Ratio  : num  18.1 12.2 12.9 7.7 11.9 9.4 11.5 13.7 11.3 11.5 ...
##  $ perc.alumni: num  12 16 30 37 2 11 26 37 23 15 ...
##  $ Expend     : num  7041 10527 8735 19016 10922 ...
##  $ Grad.Rate  : num  60 56 54 59 15 55 63 73 80 52 ...
```

```
cat("\n")
```

```
summary(College$Outstate)  # in dollars
```

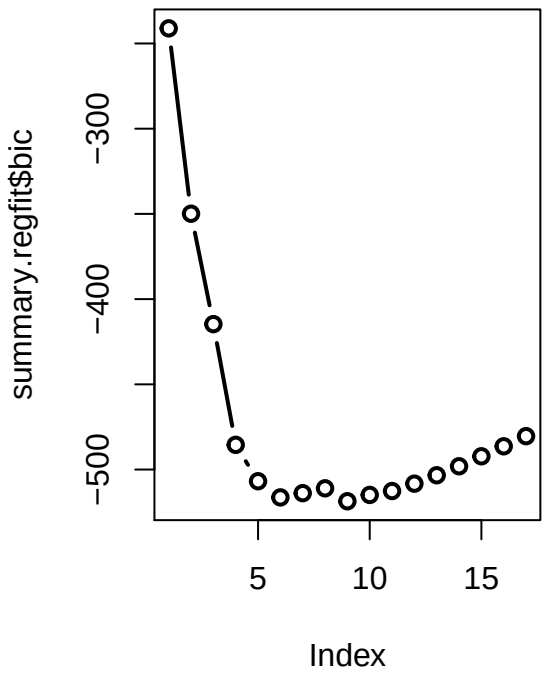
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2340    7320    9990   10440   12920   21700
```

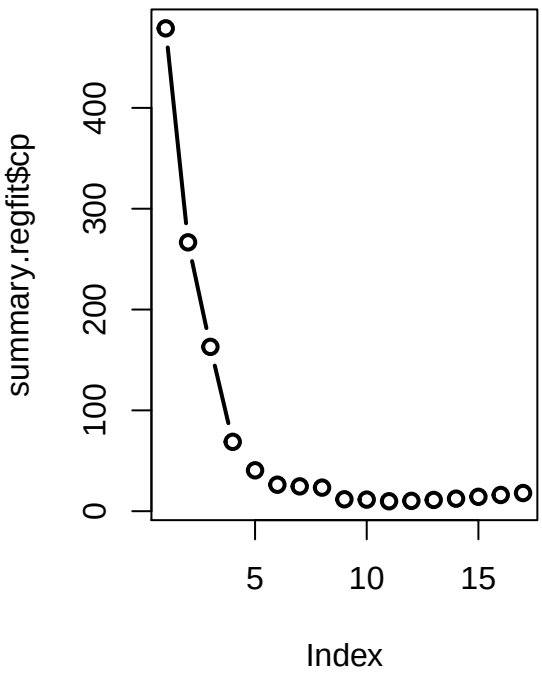
```
# initilize
library(leaps)
```

Part 10.a)

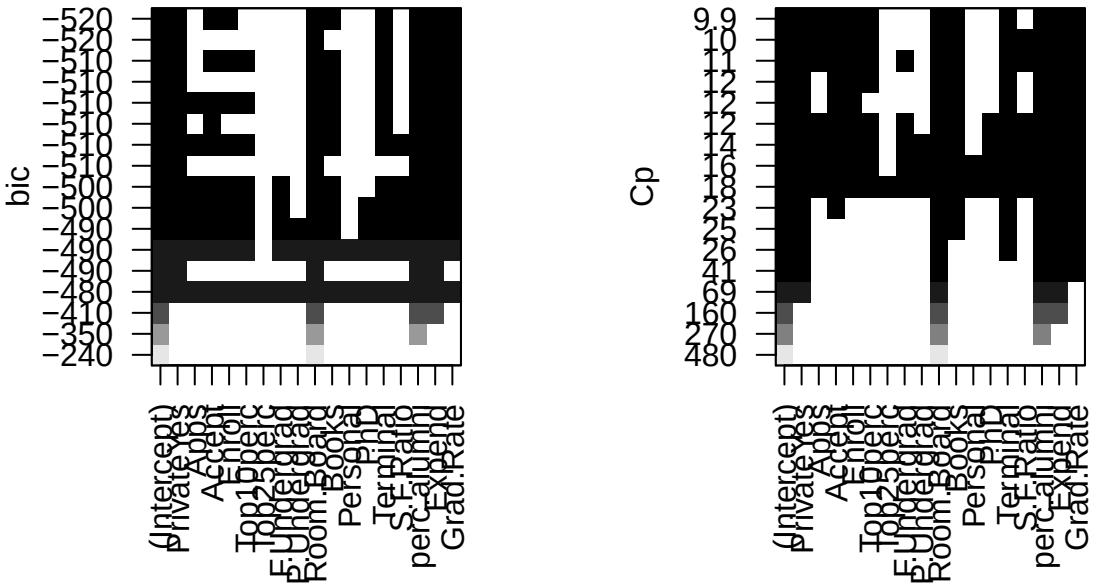
Forward stepwise selection on the training set:

```
# split hte data
set.seed(1)
train = sample(c(T, F), size = nrow(College)/2, replace = T)
test = !train

# forward stepwise on train data
regfit.full = regsubsets(Outstate ~ ., data = College, method = "forward", nvmax = 18,
                         subset = train)
# simple plot
summary.regfit = summary(regfit.full)
par(mfrow = c(1, 2))
plot(summary.regfit$bic, lwd = 2, type = "b")  #  6 > 9;  13 with full data
plot(summary.regfit$cp, lwd = 2, type = "b")  # 9 > 11; 13 with full data
```






```
# plot: another way to see the simple plot and have an idea of what variable chosen or not
plot(regfit.full, scale = "bic")
plot(regfit.full, scale = "Cp")
```



Although using 9 variables minimizes BIC, the model with 6 variables seems to be good enough, since it is a simpler model, with a similar BIC. C_p implies choosing 11 variables, while 9 might be close enough. We look more closely at the values of the BIC and C_p for models with similar size to the minimizers:

```
# zoom in for cp, since scale is too large above
summary.regfit$cp[8:13]  # to be able to see it more clearly
```

```
## [1] 23.373729 11.803954 11.516306  9.874794 10.234206 11.169251
```

```
summary.regfit$bic[5:10]
```

```
## [1] -506.8430 -516.3396 -513.8565 -511.0123 -518.5913 -514.8932
```

6 is a reasonable choice according to the BIC, while according to C_p 11 might be the best choice (since the C_p for 11 is almost 15% smaller than the one for 9). As a compromise, we choose 9 which works good enough for both BIC and C_p criteria. The estimated model's coefficeints can be seen below:

```
# coefficients: what are the chosen variables?
coef(regfit.full, id = 9)
```

```
##   (Intercept)    PrivateYes        Accept        Enroll    Room.Board
## -2512.4543309  2672.6847370     0.5369411    -1.2820126     0.8899203
##         Books      Terminal   perc.alumni        Expend     Grad.Rate
##    -1.1656204    34.5267894    36.5335795     0.2086952    31.7706626
```

```
names(coef(regfit.full, id = 9))
```

```
##  [1] "(Intercept)" "PrivateYes"  "Accept"      "Enroll"      "Room.Board"
##  [6] "Books"       "Terminal"    "perc.alumni" "Expend"      "Grad.Rate"
```

The variables that are excluded could (roughly) be found using

```
fwd.names = c("Outstate", names(coef(regfit.full, id = 9)))
fwd.ind = !(names(College) %in% fwd.names)
names(College)[fwd.ind]
```

```
## [1] "Private"     "Apps"        "Top10perc"   "Top25perc"   "F.Undergrad"
## [6] "P.Undergrad" "Personal"    "PhD"         "S.F.Ratio"
```

Note that the first variable above, `Private`, should not be there. It is a factor variable that is in fact included in the regression with id equal to 9, but its name is changed to `PrivateYes` to indicate how the dummy variable is defined.
It is in general worthwhile to keep track of what variable are factor variables, e.g. for possible changes that R commands make to their names.
Omission of some variables is easy to understand, for example due to similar definition and resulting high correlations:

```
cor(College$Terminal, College$PhD)
```

```
## [1] 0.849587
```

Part 10. b)

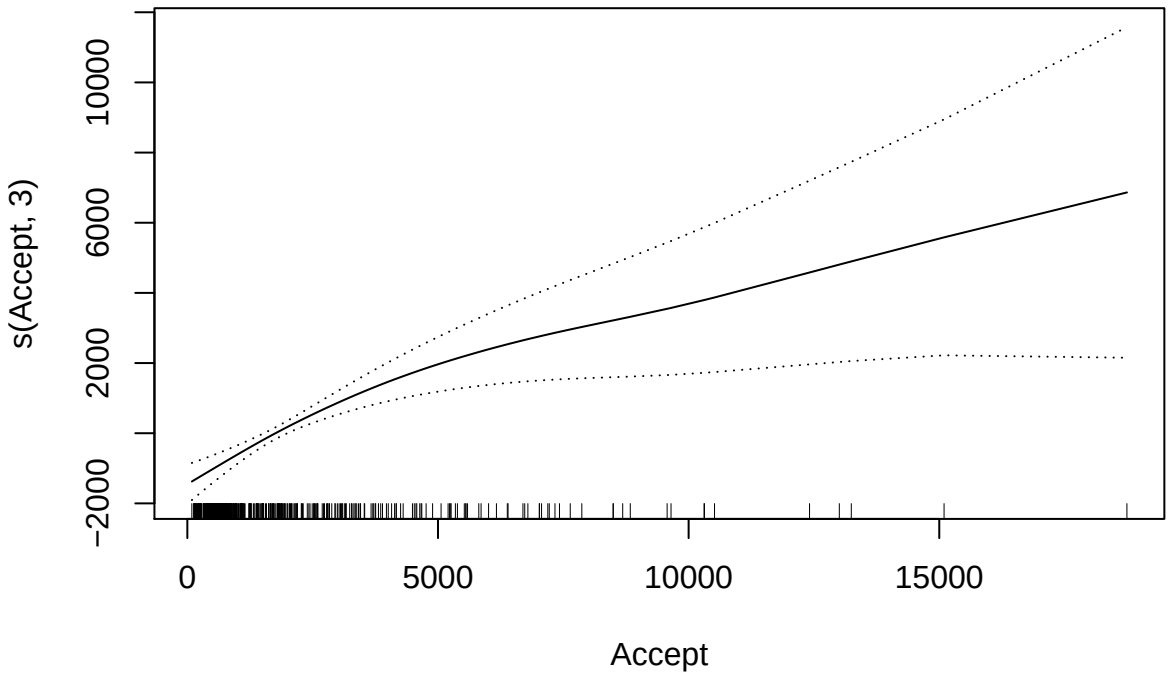
We estimate GAM on the training set. The formula can be written manually, especially since we have a few variables here; but that may be difficult when we have many variables. For expositional purposes, we see how the GAM formula can be retrieved dynamically from `regsubsets` output:

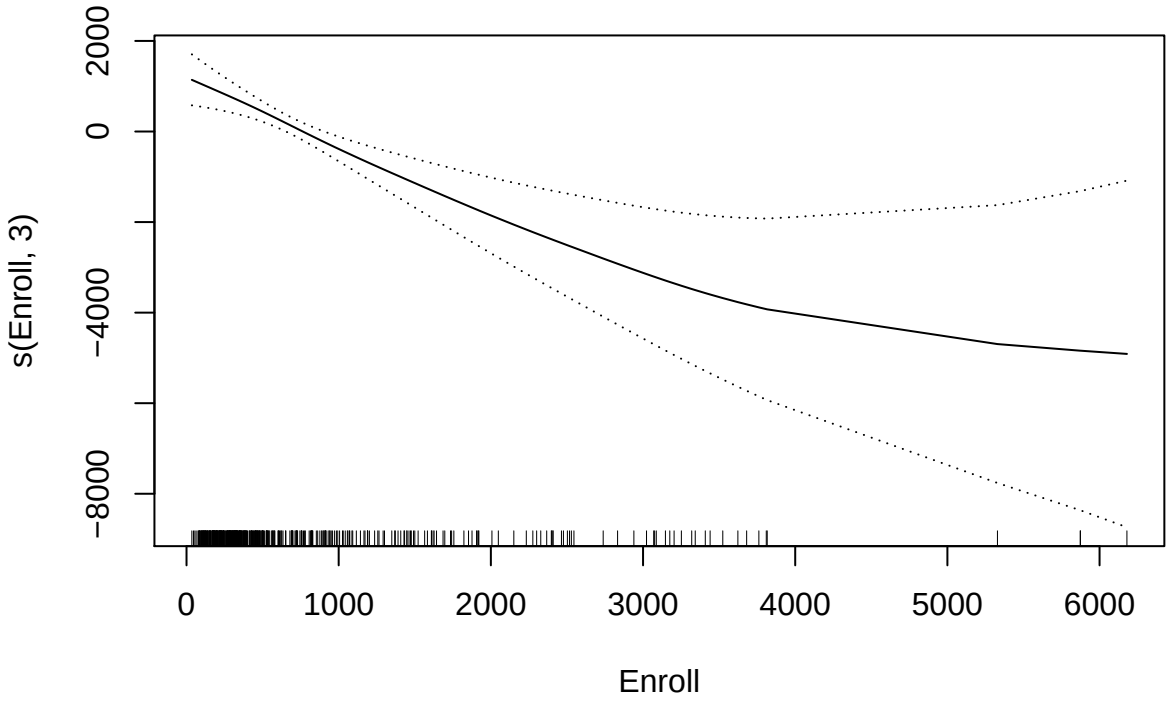
```
library(gam)
# leave out factor variable and the response
gam.varnames = names(coef(regfit.full, id = 9))[-(1:2)]
gam.form = as.formula(paste("Outstate ~ s(",                                  # response and start
                            paste0(gam.varnames, collapse = ", 3) + s(" ),    # quant. and middle
                            ", 3) + Private"))                                # qual. and end

gam.fit.df3 = gam(gam.form, data = College, subset = train)
summary(gam.fit.df3)
```

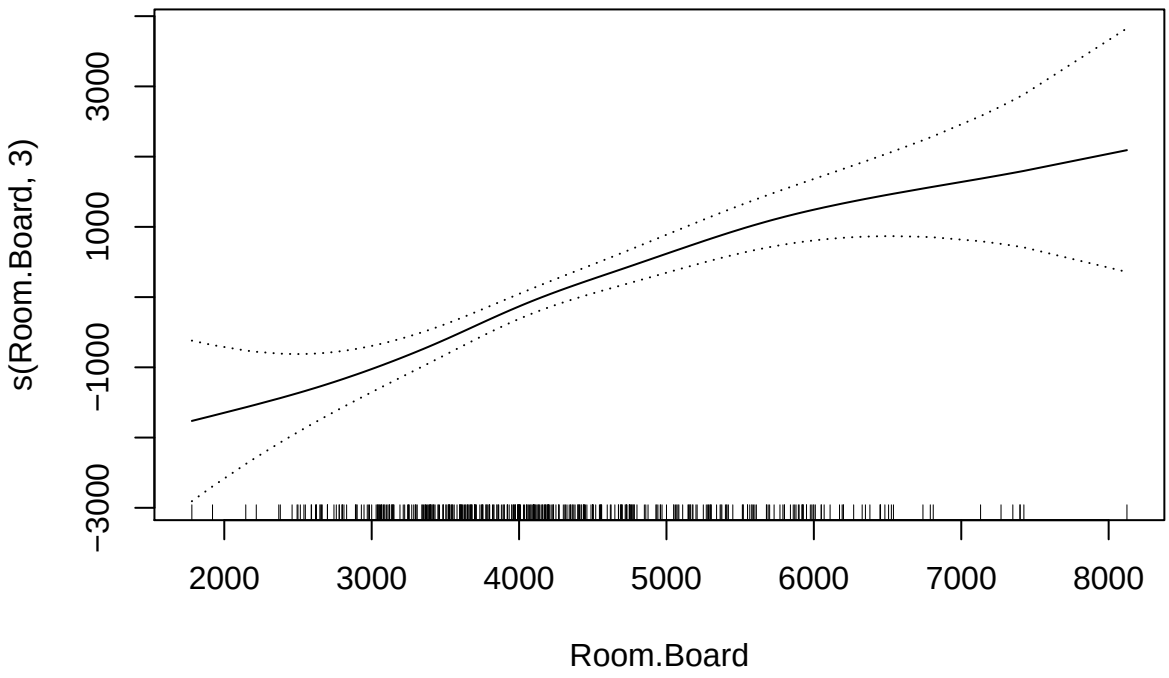
```
##
## Call: gam(formula = gam.form, data = College, subset = train)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -5850.55 -1066.60    46.38  1113.35  8791.73
##
## (Dispersion Parameter for gaussian family taken to be 3249258)
##
##     Null Deviance: 6306847377 on 418 degrees of freedom
## Residual Deviance: 1276956892 on 392.9995 degrees of freedom
## AIC: 7498.689
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##                    Df     Sum Sq    Mean Sq  F value    Pr(>F)
## s(Accept, 3)        1    1367905    1367905   0.4210  0.516822
## s(Enroll, 3)        1 1005219703 1005219703 309.3690 < 2.2e-16 ***
## s(Room.Board, 3)    1 1614835344 1614835344 496.9859 < 2.2e-16 ***
## s(Books, 3)         1   26237450   26237450   8.0749  0.004722 **
```

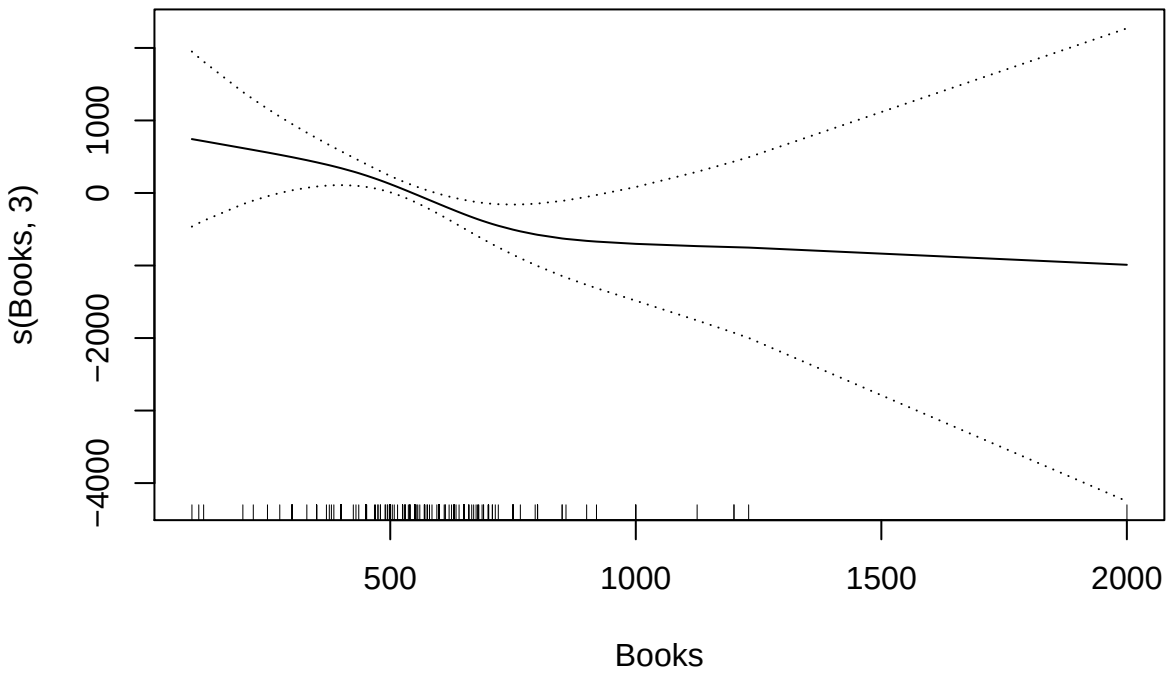
```
## s(Terminal, 3)      1  137783923  137783923  42.4047 2.271e-10 ***
## s(perc.alumni, 3)   1  511022728  511022728 157.2737 < 2.2e-16 ***
## s(Expend, 3)        1  561425965  561425965 172.7859 < 2.2e-16 ***
## s(Grad.Rate, 3)     1  120958415  120958415  37.2265 2.521e-09 ***
## Private             1  231486837  231486837  71.2430 6.142e-16 ***
## Residuals         393 1276956892    3249258
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                   Npar Df Npar F     Pr(F)
## (Intercept)
## s(Accept, 3)            2  5.915  0.002944 **
## s(Enroll, 3)            2  3.873  0.021599 *
## s(Room.Board, 3)        2  1.655  0.192337
## s(Books, 3)             2  1.739  0.177082
## s(Terminal, 3)          2  1.273  0.281158
## s(perc.alumni, 3)       2  1.218  0.296842
## s(Expend, 3)            2 34.999 1.021e-14 ***
## s(Grad.Rate, 3)         2  1.345  0.261768
## Private
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

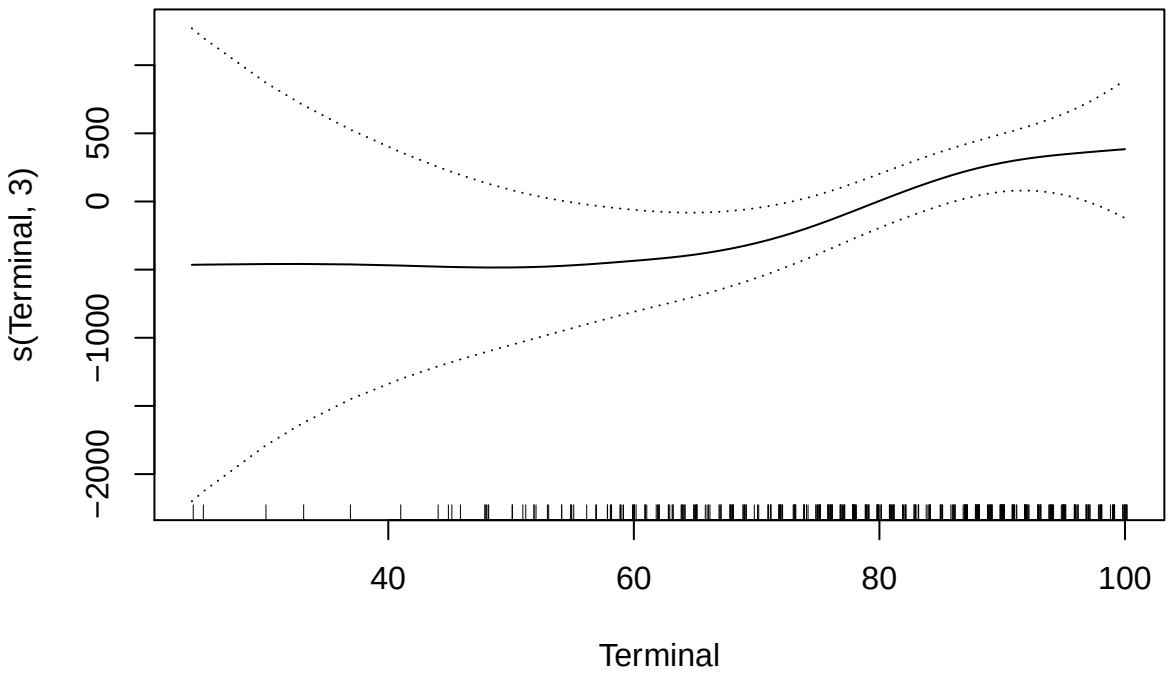
```
plot.gam.df3 = plot(gam.fit.df3, se = T)
```

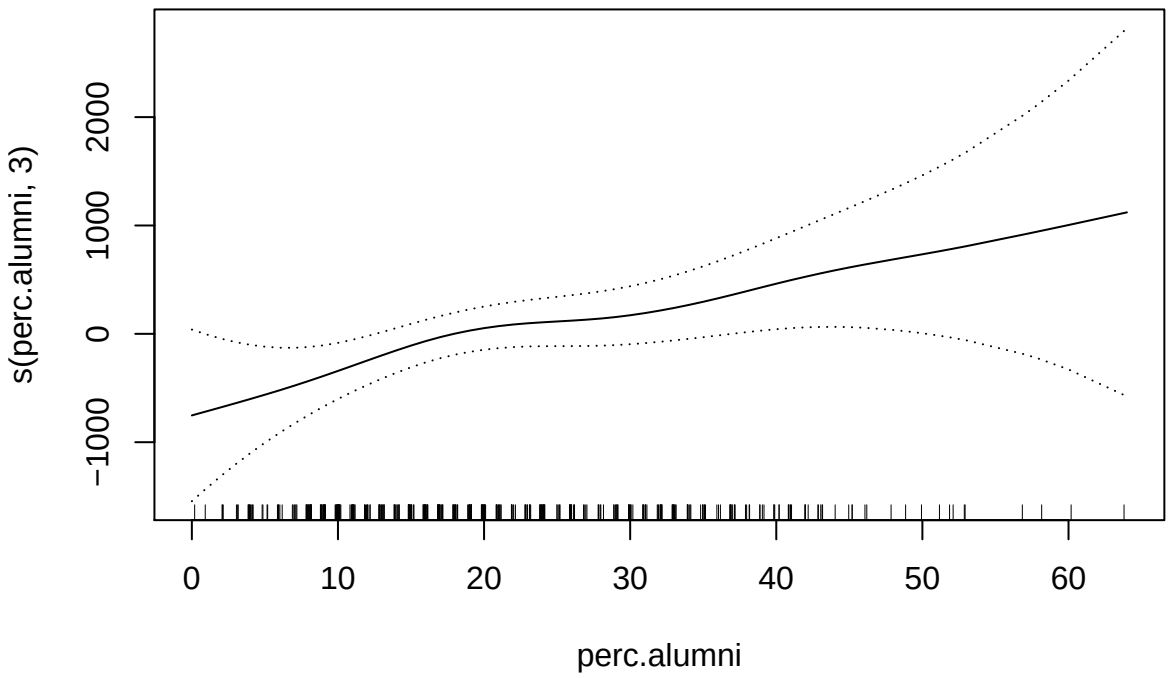


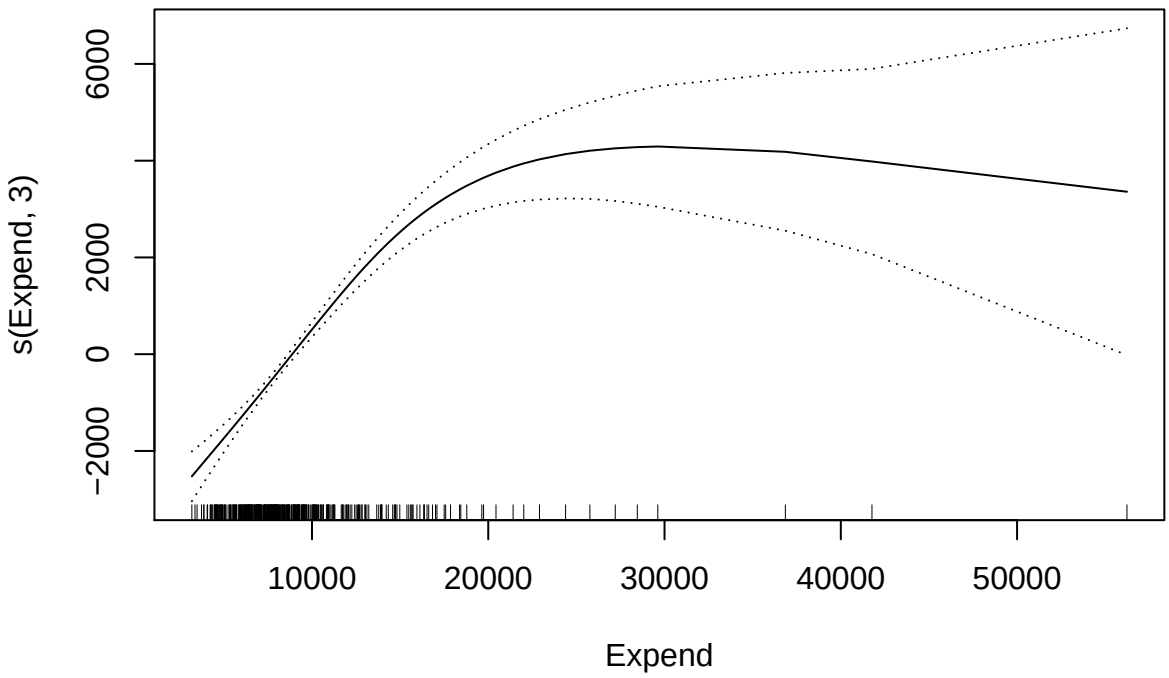

s(Enroll, 3)
2000
0
−4000
−8000
0
1000
2000
3000
4000
5000
6000
Enroll

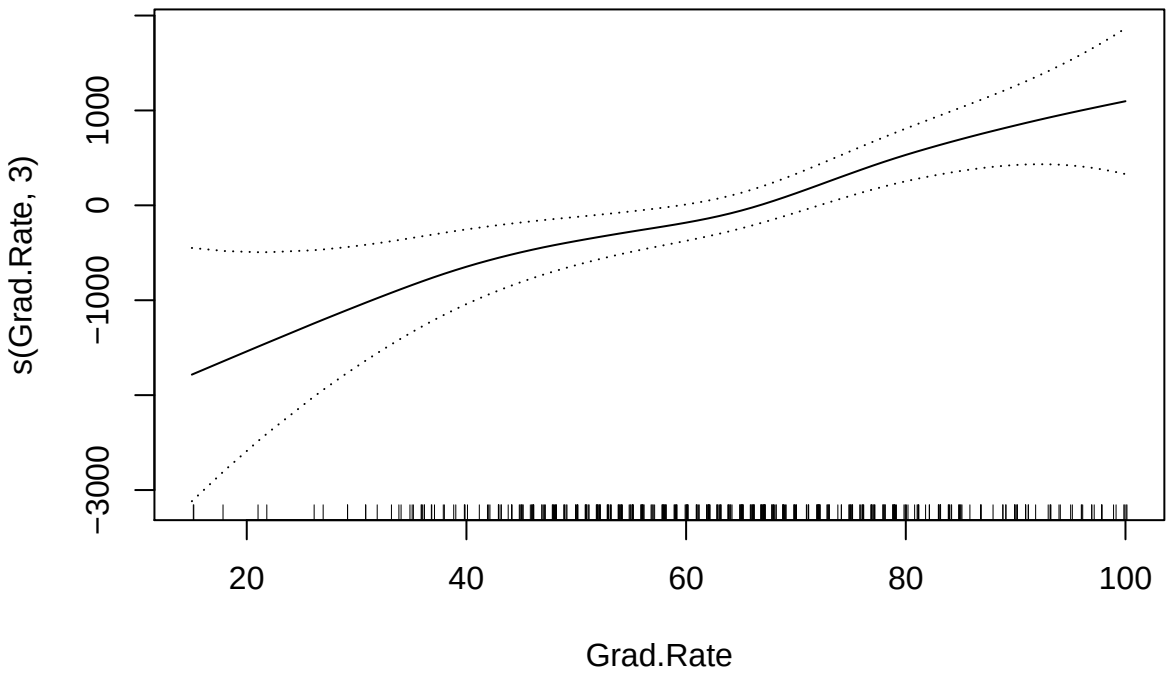

s(Room.Board, 3)
−3000
−1000
1000
3000
2000
3000
4000
5000
6000
7000
8000
Room.Board

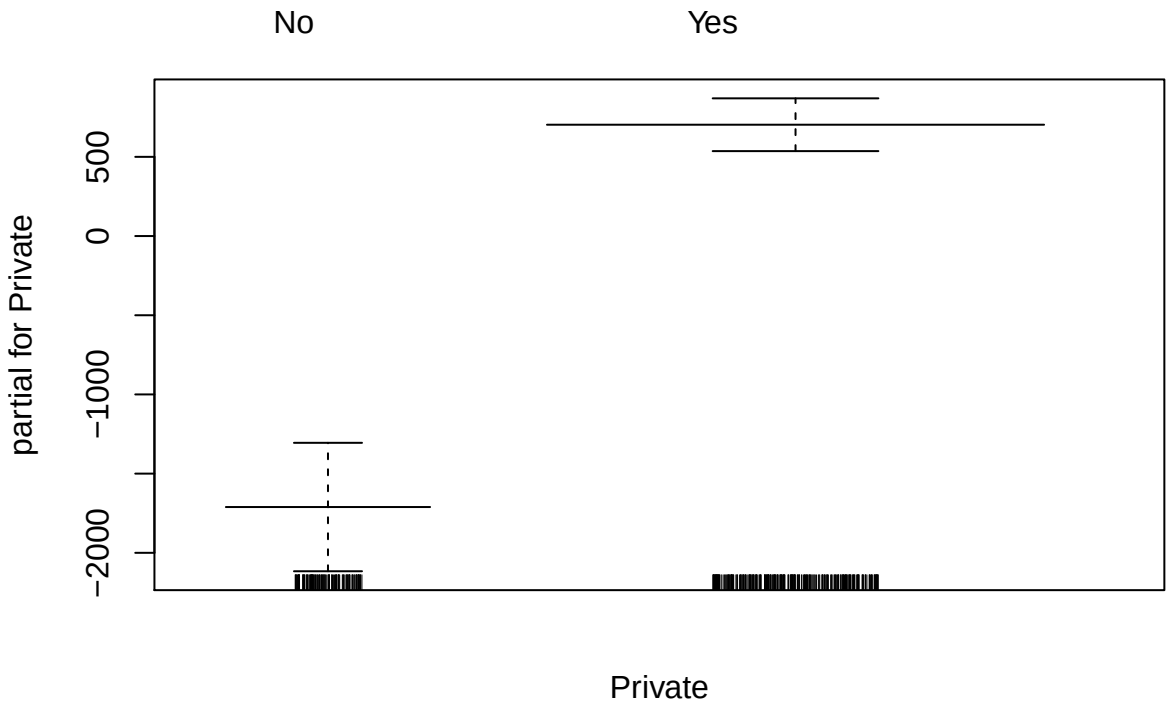

s(Books, 3)
1000
0
−2000
−4000
500
1000
1500
2000
Books


s(Terminal, 3)
−2000
−1000
0
500
40
60
80
100
Terminal


s(perc.alumni, 3)
−1000
0
1000
2000
0
10
20
30
40
50
60
perc.alumni

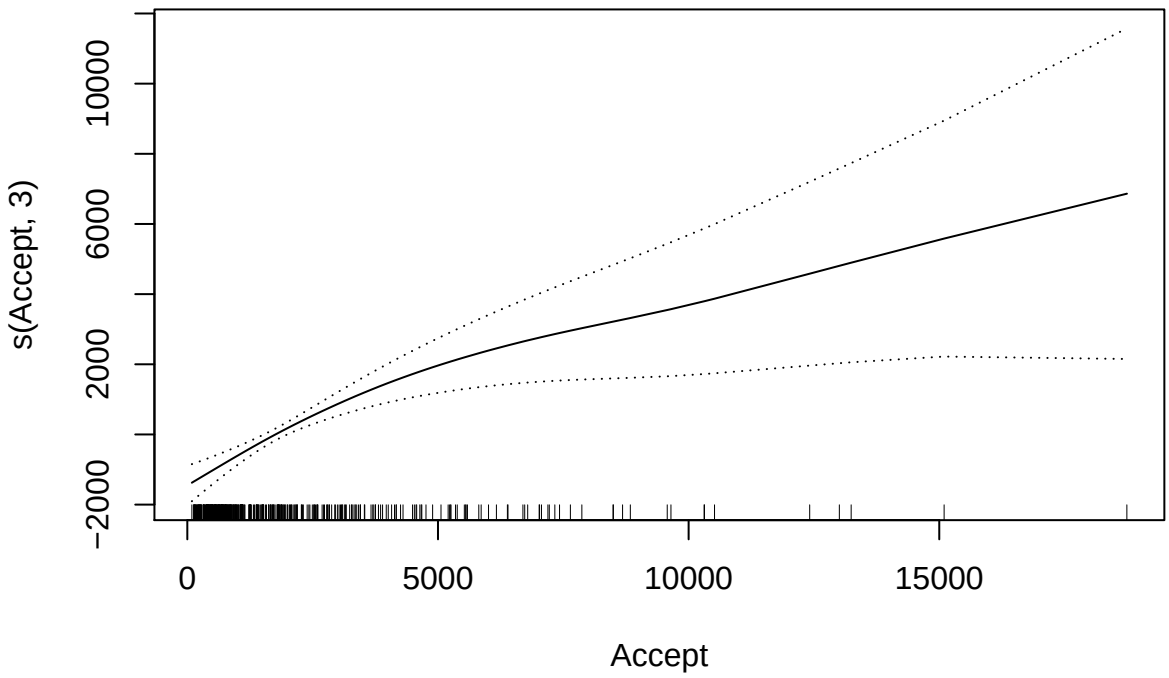

s(Expend, 3)
6000
2000
0
−2000
10000
20000
30000
40000
50000
Expend

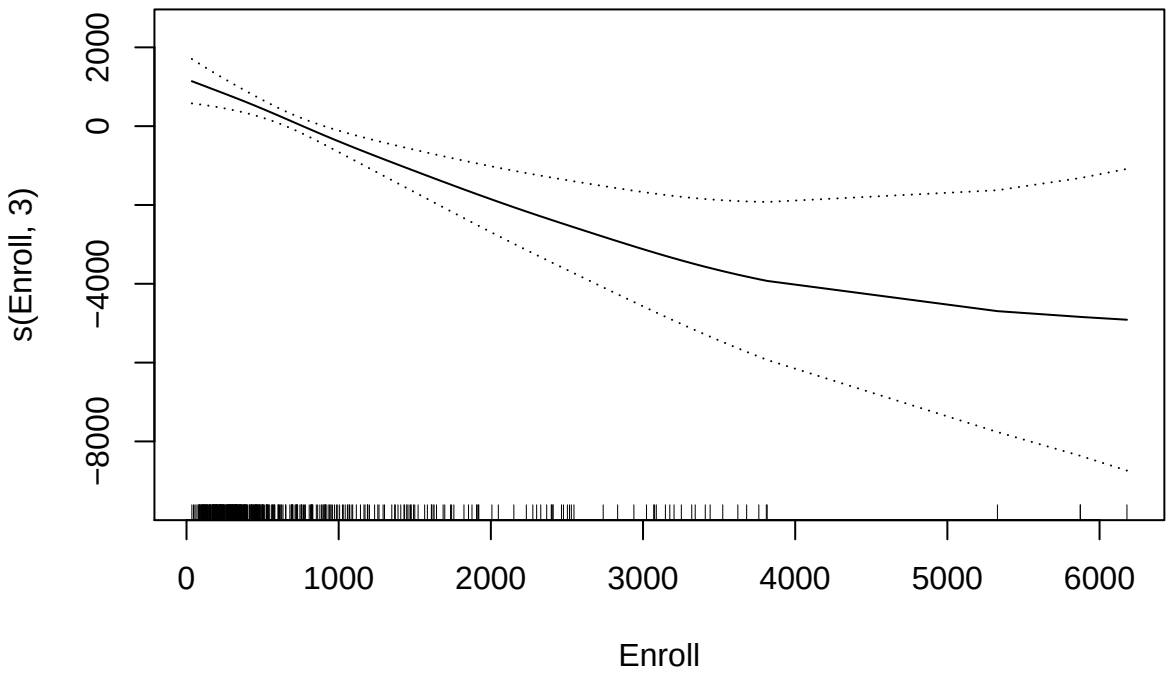

s(Grad.Rate, 3)
−3000
−1000
0
1000
20
40
60
80
100
Grad.Rate

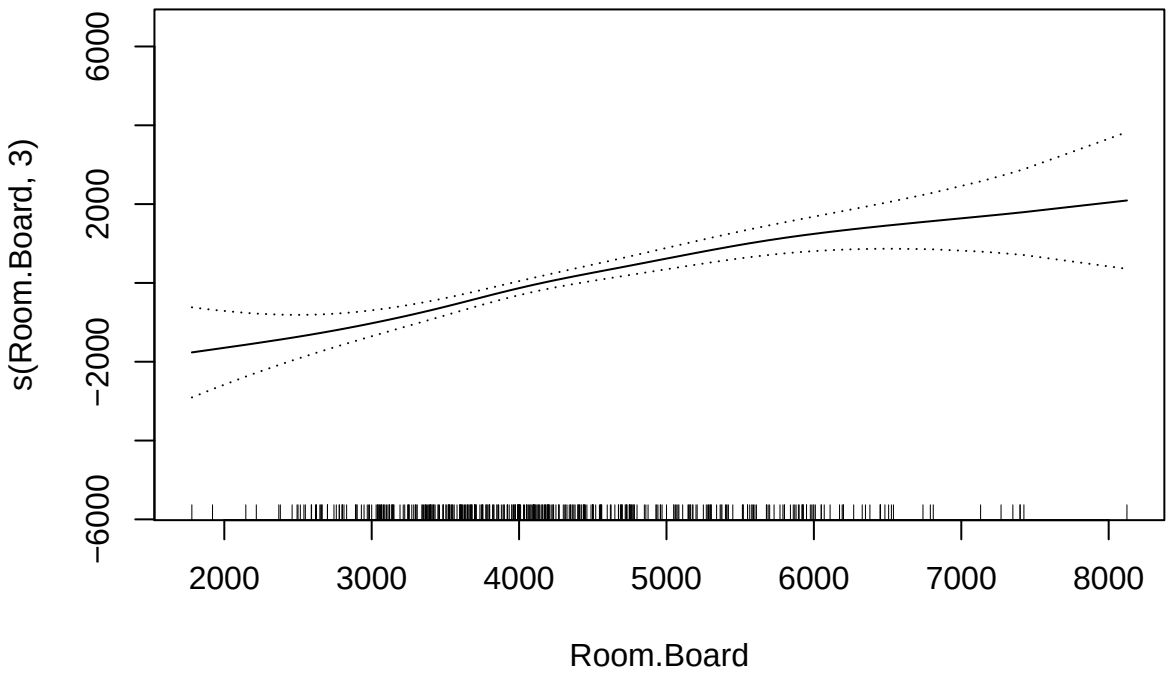


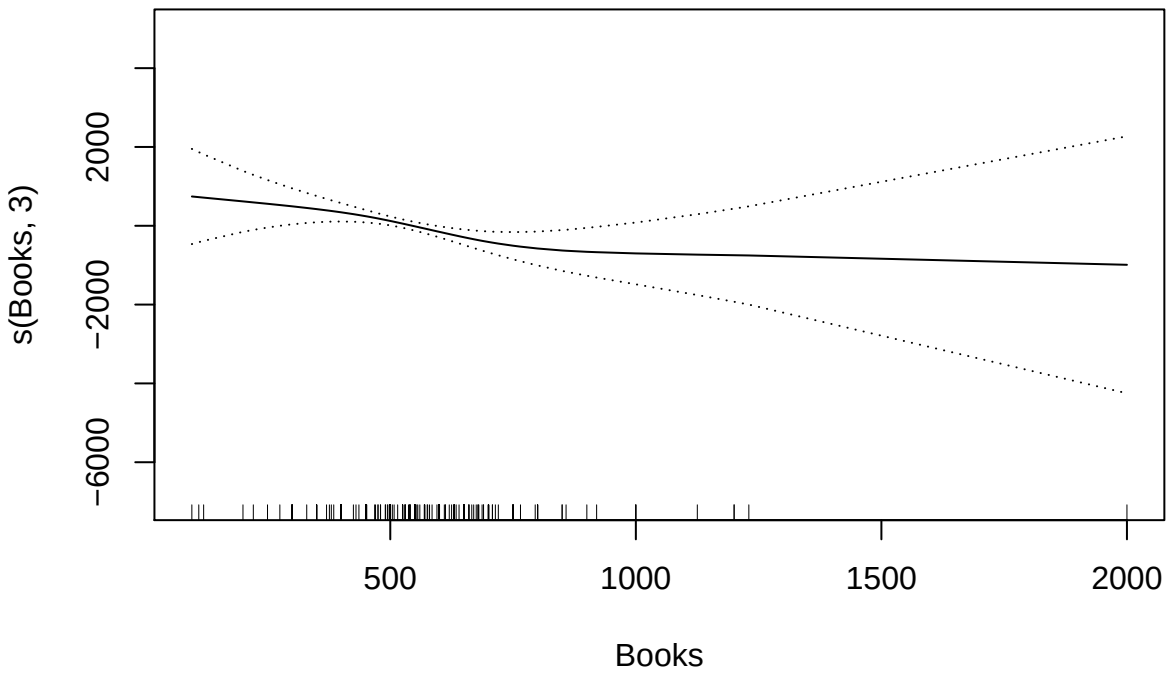
Next, we use the same scale for all variables to see which variables are more important.

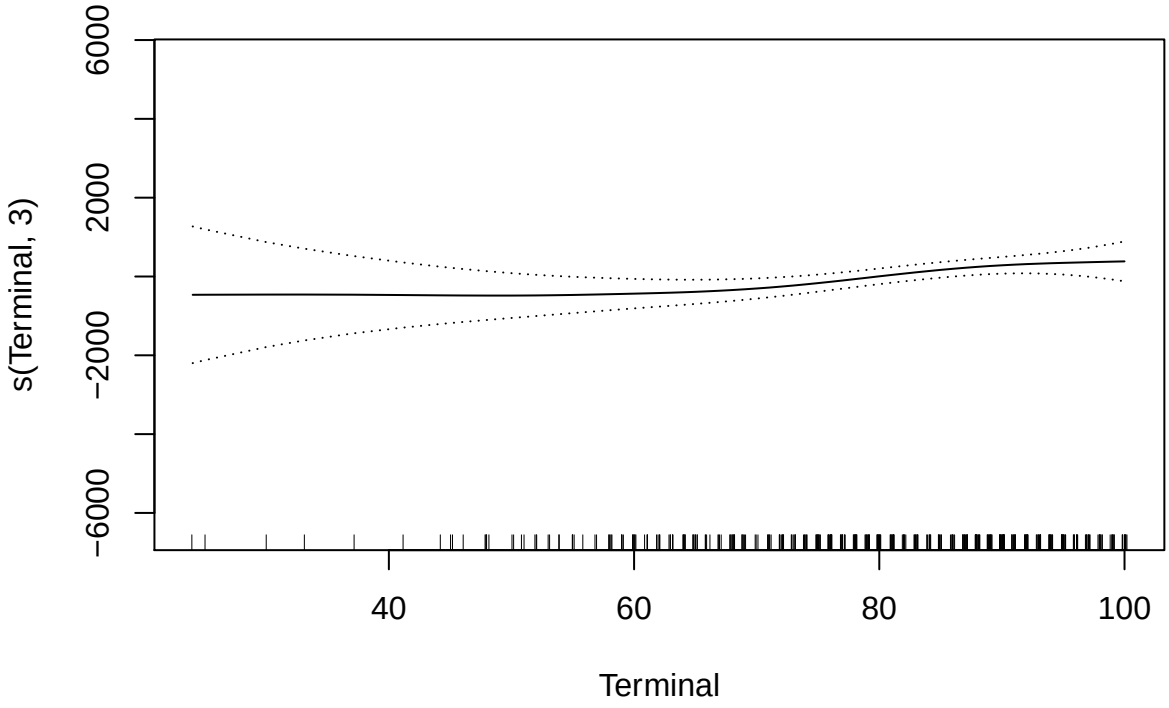
```
plot(gam.fit.df3, se = T, scale = 12000)
```

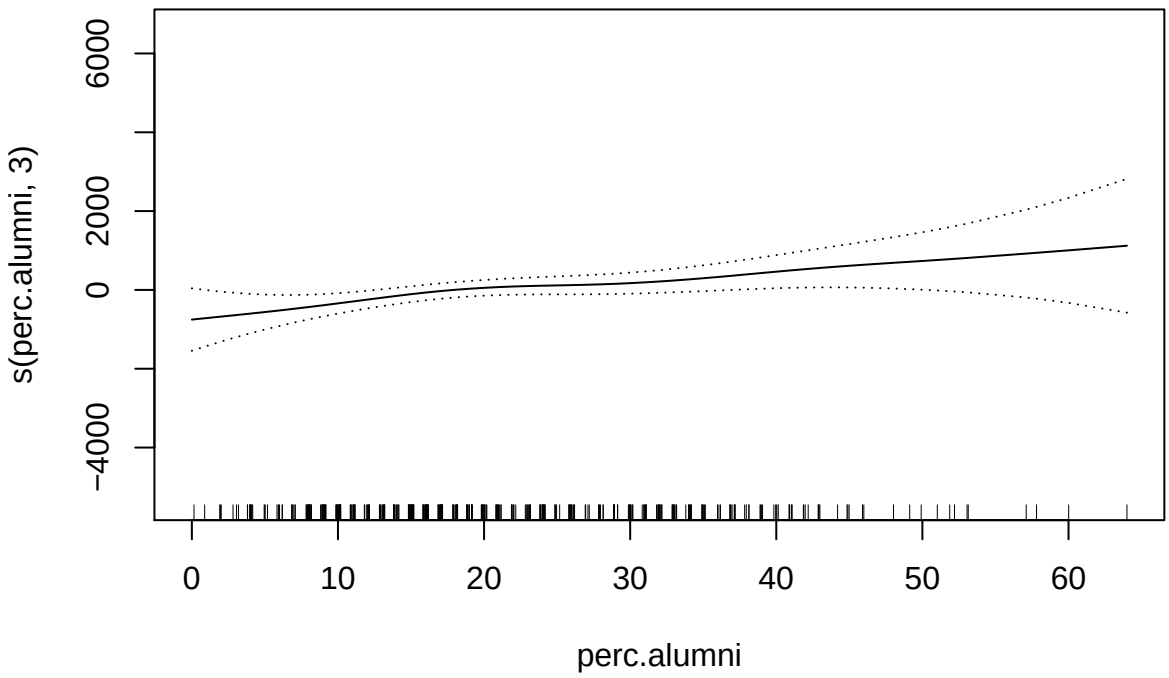

s(Accept, 3)
−2000
2000
6000
10000
0
5000
10000
15000
Accept

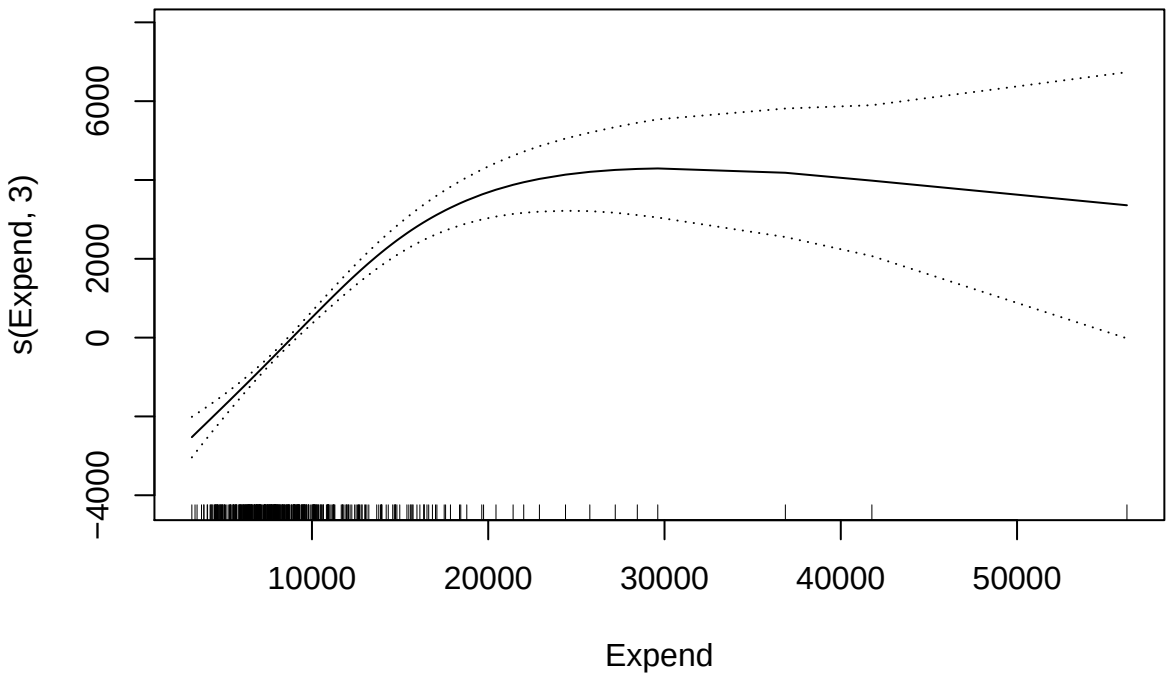

s(Enroll, 3)
2000
0
−4000
−8000
0
1000
2000
3000
4000
5000
6000
Enroll

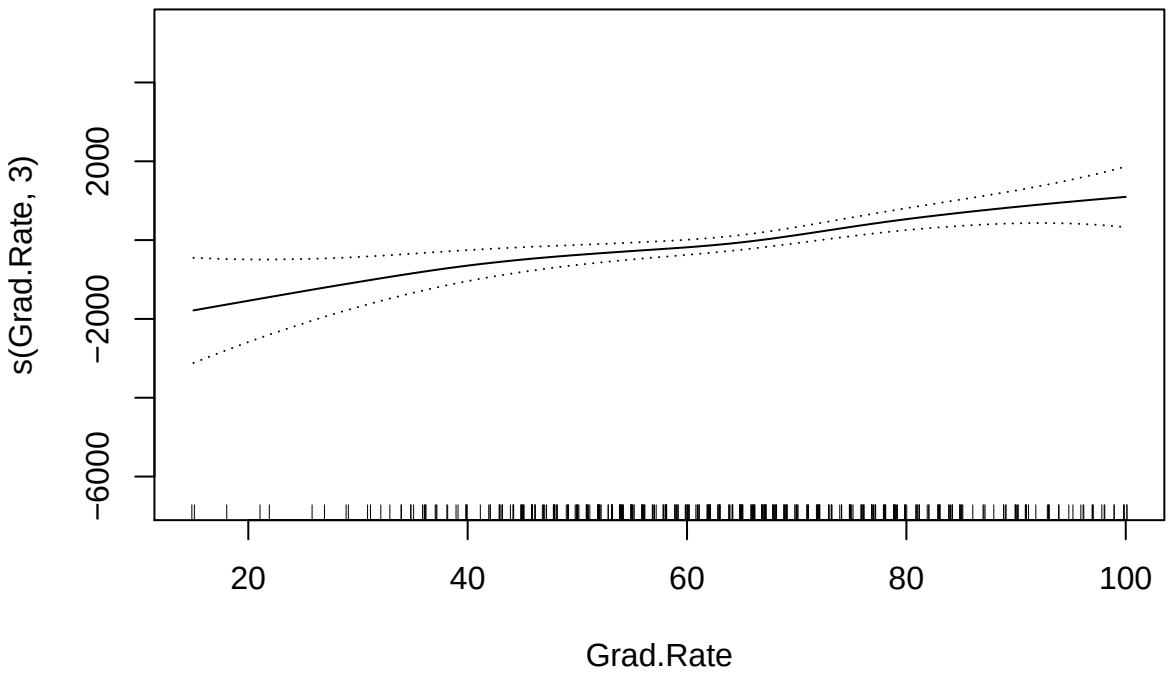

s(Room.Board, 3)
6000
2000
−2000
−6000
2000
3000
4000
5000
6000
7000
8000
Room.Board

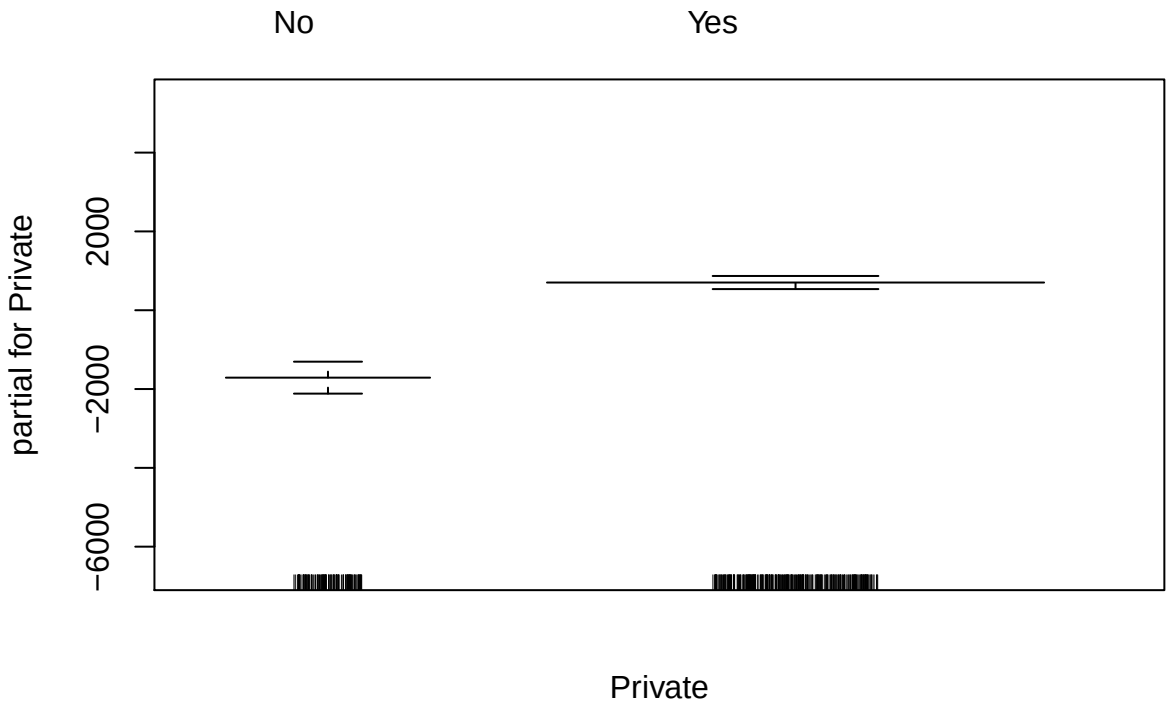

s(Books, 3)
2000
−2000
−6000
500
1000
1500
2000
Books


s(Terminal, 3)
6000
2000
−2000
−6000
40
60
80
100
Terminal


s(perc.alumni, 3)
6000
2000
0
−4000
0
10
20
30
40
50
60
perc.alumni


s(Expend, 3)
−4000
0
2000
6000
10000
20000
30000
40000
50000
Expend


s(Grad.Rate, 3)
−6000
−2000
2000
20
40
60
80
100
Grad.Rate



Two important things from the plots:

- Possibility of outliers and leverage points, e.g. for `Accept` and `Enroll`
- Interaction and normalized variables
  - interaction of private and other variables: private schools tuitions are expected to be more sensitive to changes
  - rates of enrollment and acceptance
- Small linear effects of some variables.
- There are both demand and supply side effects influencing how the tuition is determined
  - Apps ~~normalized by Enroll (to account for school size)~~ could proxy demand,
  - Supply would be the current capacity, which would equal `Enroll` if admission committee knows the percentage of students who accept the admssion offer
  - The resulting endogeneity makes causal inference impossible

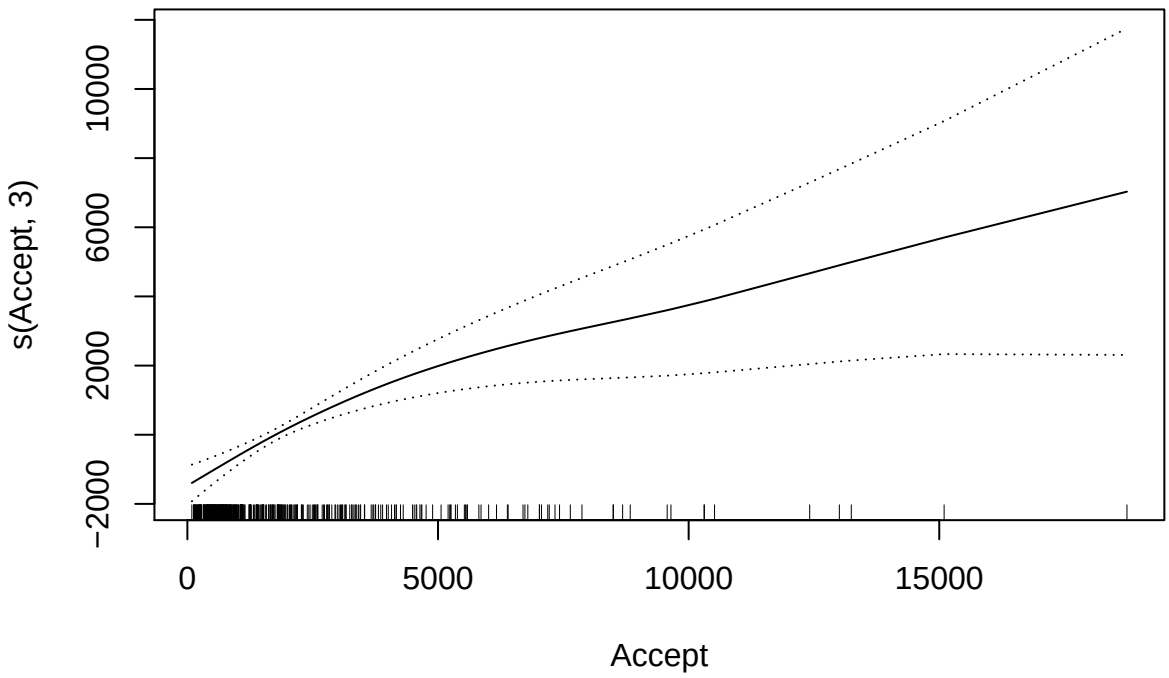
The model estimated above used three degrees of freedom for all variables. After looking at the resulting plots, we try two other combinations of degrees of freedom. Below, we lower the degrees for some variables:

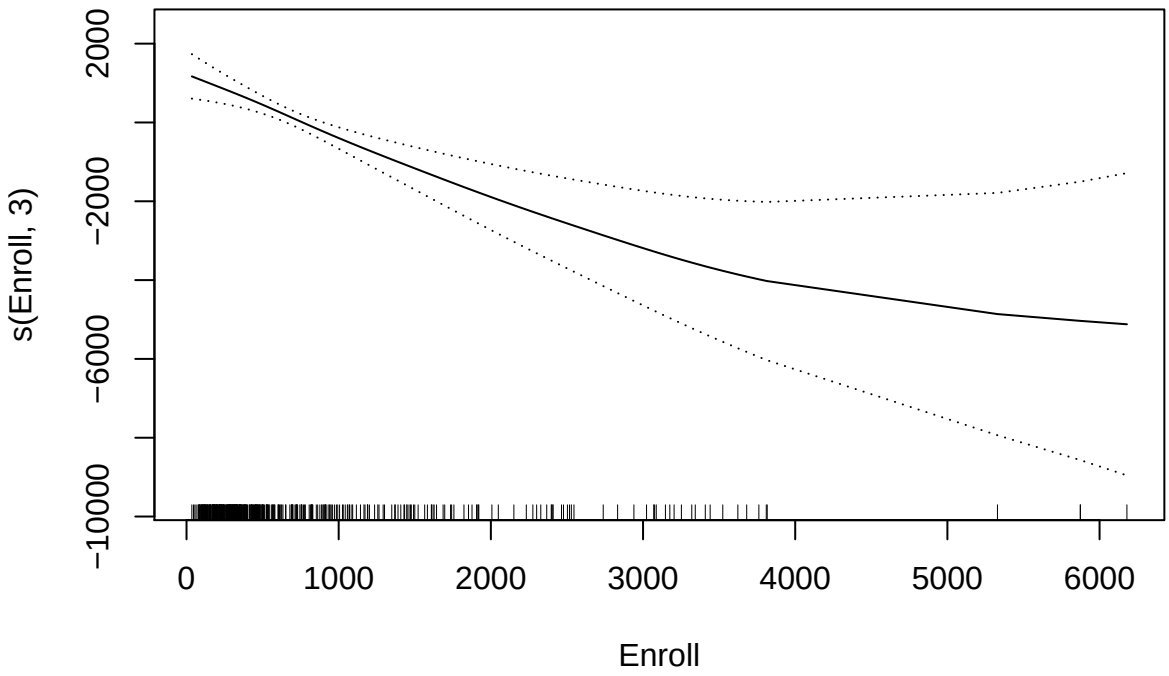
```
gam.fit.df321 = gam(Outstate ~ s(Accept, 3) + s(Enroll, 3) + s(Books, 3) + s(Expend, 3) +
                            s(Terminal, 2) + s(perc.alumni, 2) + s(Room.Board, 2) +
                            Grad.Rate + Private,
                  data = College, subset = train)
summary(gam.fit.df321)
```

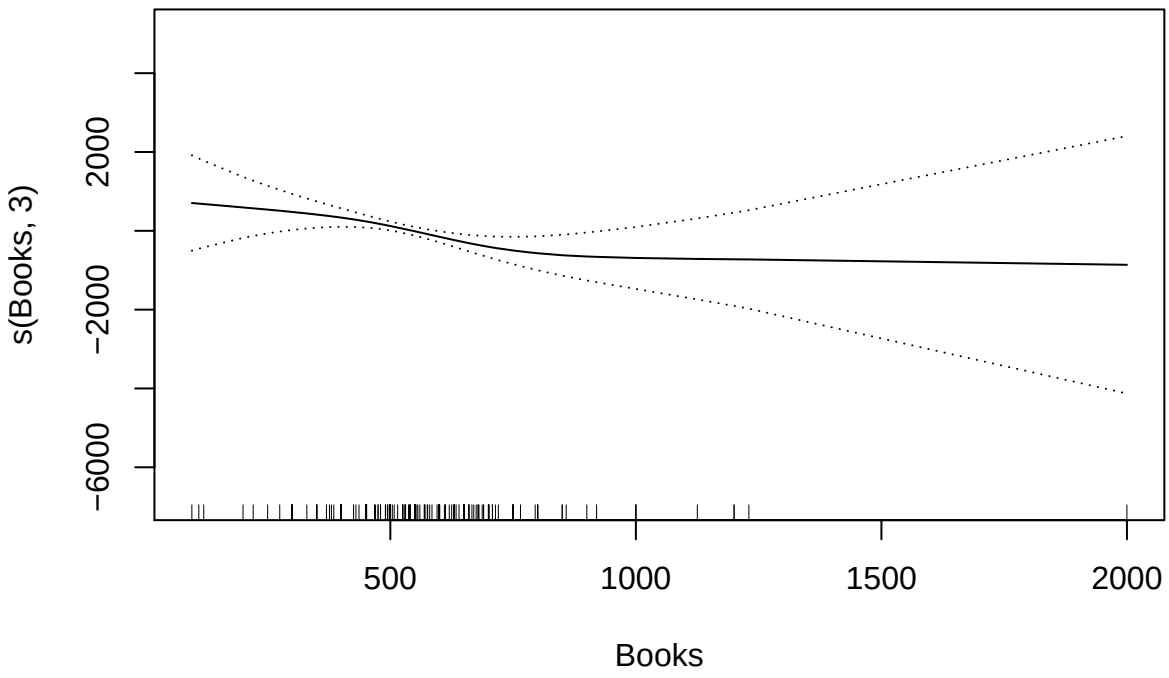
```
##
## Call: gam(formula = Outstate ~ s(Accept, 3) + s(Enroll, 3) + s(Books,
##     3) + s(Expend, 3) + s(Terminal, 2) + s(perc.alumni, 2) +
##     s(Room.Board, 2) + Grad.Rate + Private, data = College, subset = train)
## Deviance Residuals:
```

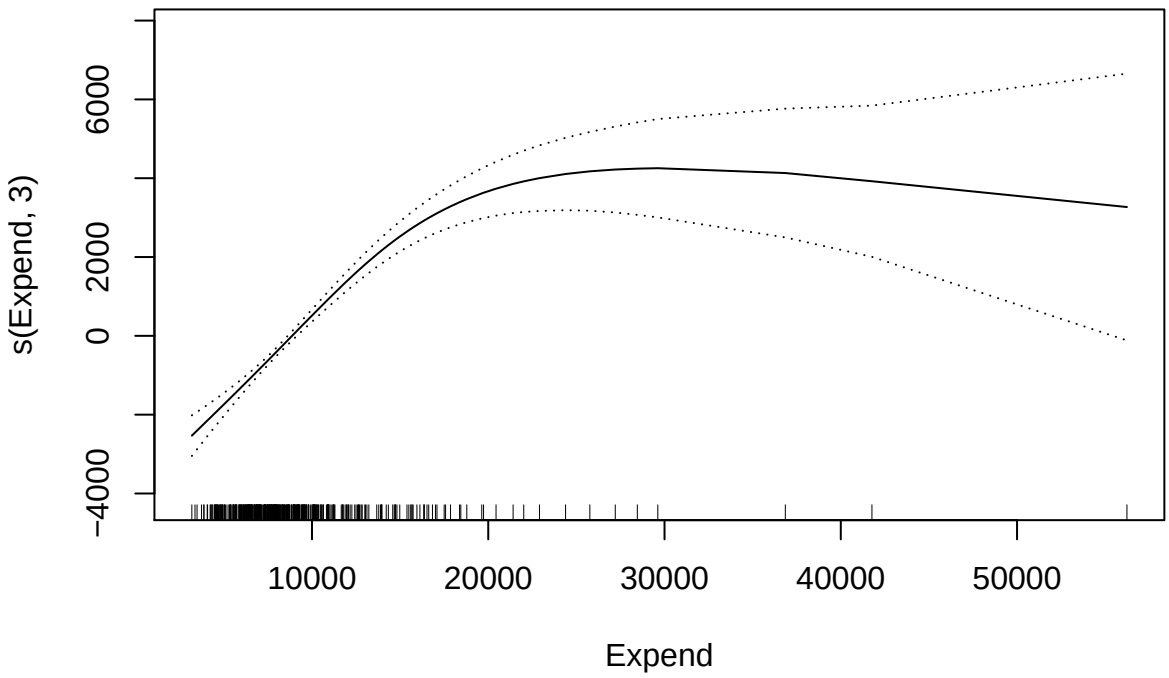
```
##       Min        1Q    Median        3Q       Max
## -5754.752 -1048.003    -6.608  1126.070  8771.808
##
## (Dispersion Parameter for gaussian family taken to be 3263286)
##
##     Null Deviance: 6306847377 on 418 degrees of freedom
## Residual Deviance: 1298788183 on 398.0001 degrees of freedom
## AIC: 7495.791
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##                     Df     Sum Sq    Mean Sq  F value    Pr(>F)
## s(Accept, 3)         1    1306237    1306237   0.4003    0.5273
## s(Enroll, 3)         1 1012152254 1012152254 310.1635 < 2.2e-16 ***
## s(Books, 3)          1     489243     489243   0.1499    0.6988
## s(Expend, 3)         1 2171370769 2171370769 665.3939 < 2.2e-16 ***
## s(Terminal, 2)       1   64235946   64235946  19.6844 1.184e-05 ***
## s(perc.alumni, 2)    1  240055906  240055906  73.5626 < 2.2e-16 ***
## s(Room.Board, 2)     1  368196068  368196068 112.8298 < 2.2e-16 ***
## Grad.Rate            1  123155121  123155121  37.7396 1.962e-09 ***
## Private              1  223175957  223175957  68.3899 2.033e-15 ***
## Residuals          398 1298788183    3263286
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                    Npar Df Npar F     Pr(F)
## (Intercept)
## s(Accept, 3)             2  5.930   0.00290 **
## s(Enroll, 3)             2  3.797   0.02326 *
## s(Books, 3)              2  1.786   0.16889
## s(Expend, 3)             2 35.243 8.105e-15 ***
## s(Terminal, 2)           1  1.240   0.26616
## s(perc.alumni, 2)        1  0.779   0.37791
## s(Room.Board, 2)         1  1.653   0.19929
## Grad.Rate
## Private
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

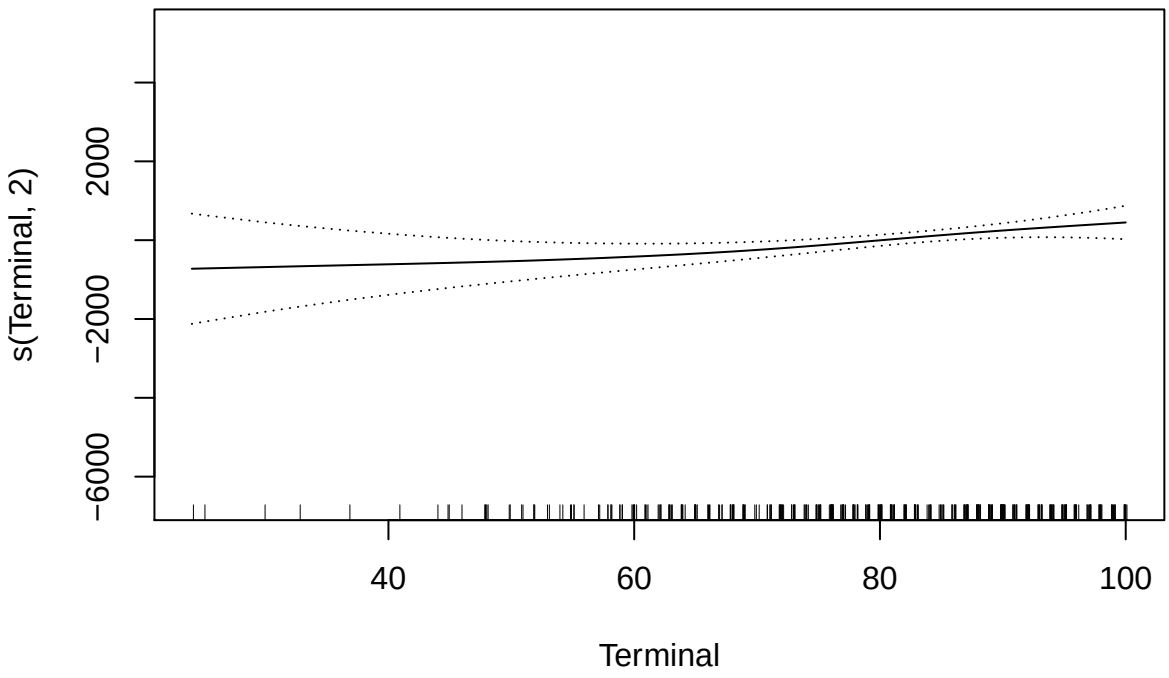
```
plot(gam.fit.df321, se = T, scale = 12000)
```

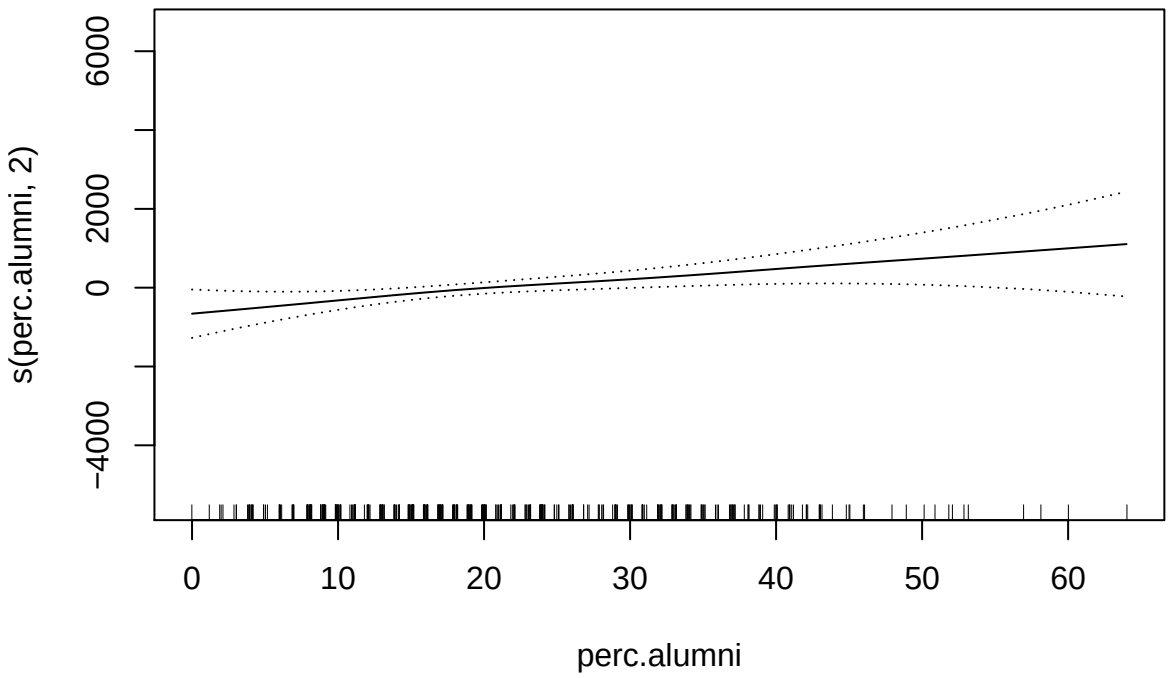

s(Accept, 3)
−2000
2000
6000
10000
0
5000
10000
15000
Accept

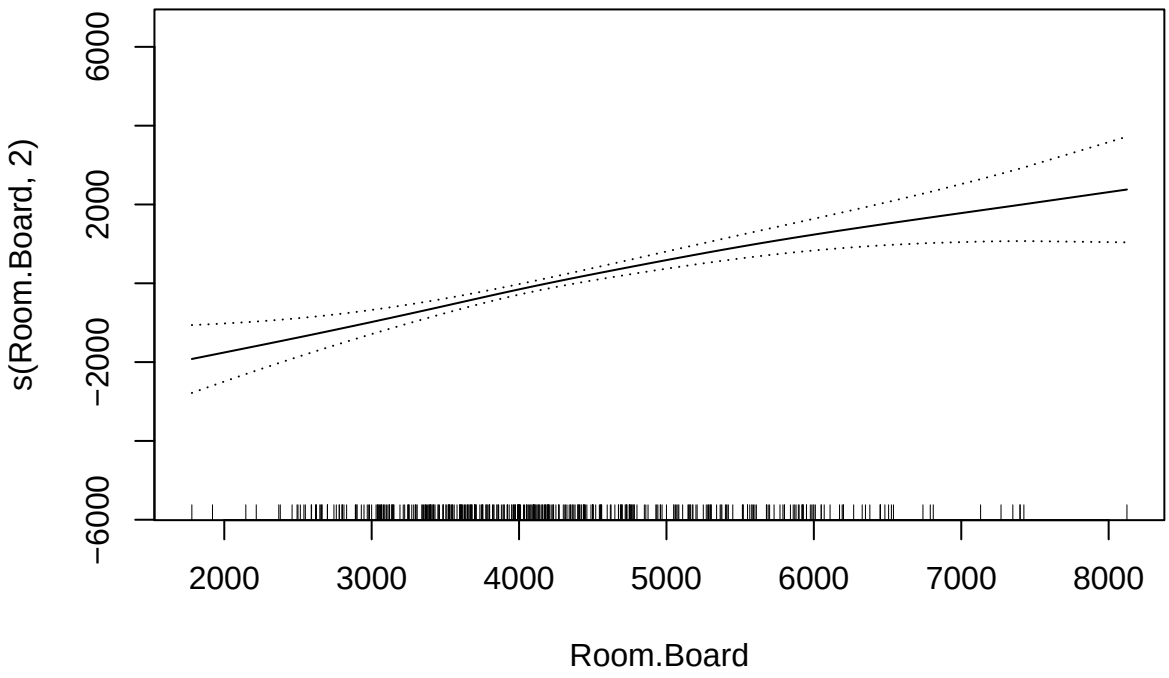

s(Enroll, 3)
2000
−2000
−6000
−10000
0
1000
2000
3000
4000
5000
6000
Enroll

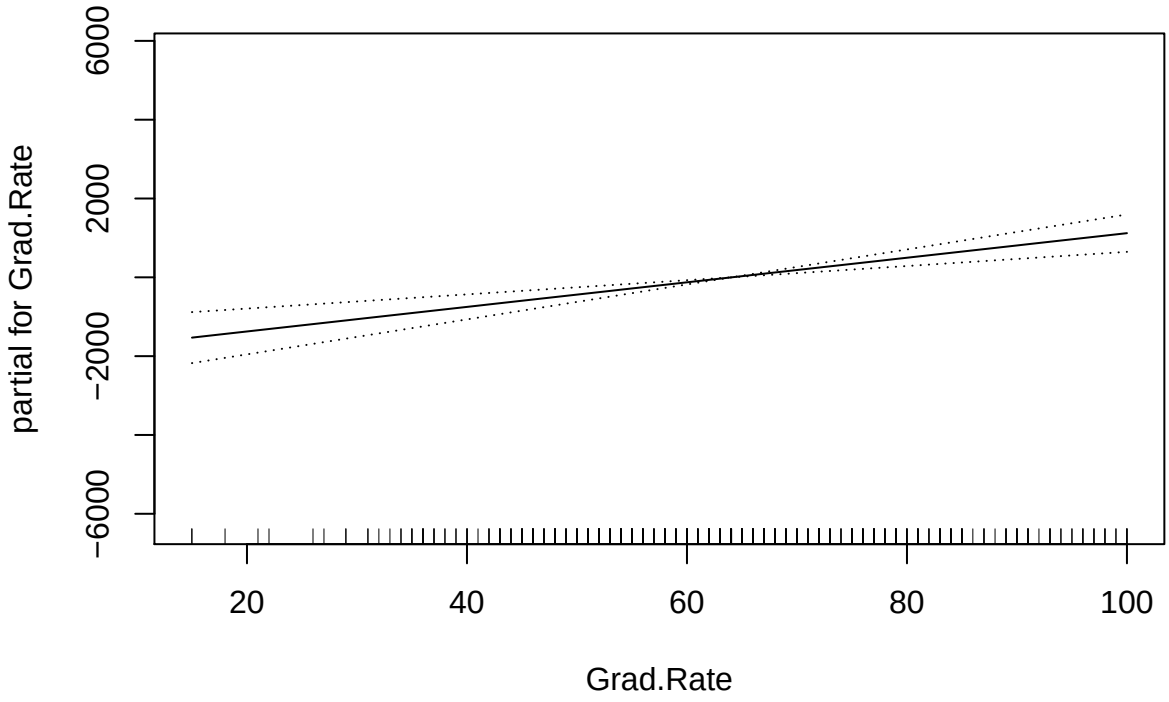

s(Books, 3)
2000
−2000
−6000
500
1000
1500
2000
Books

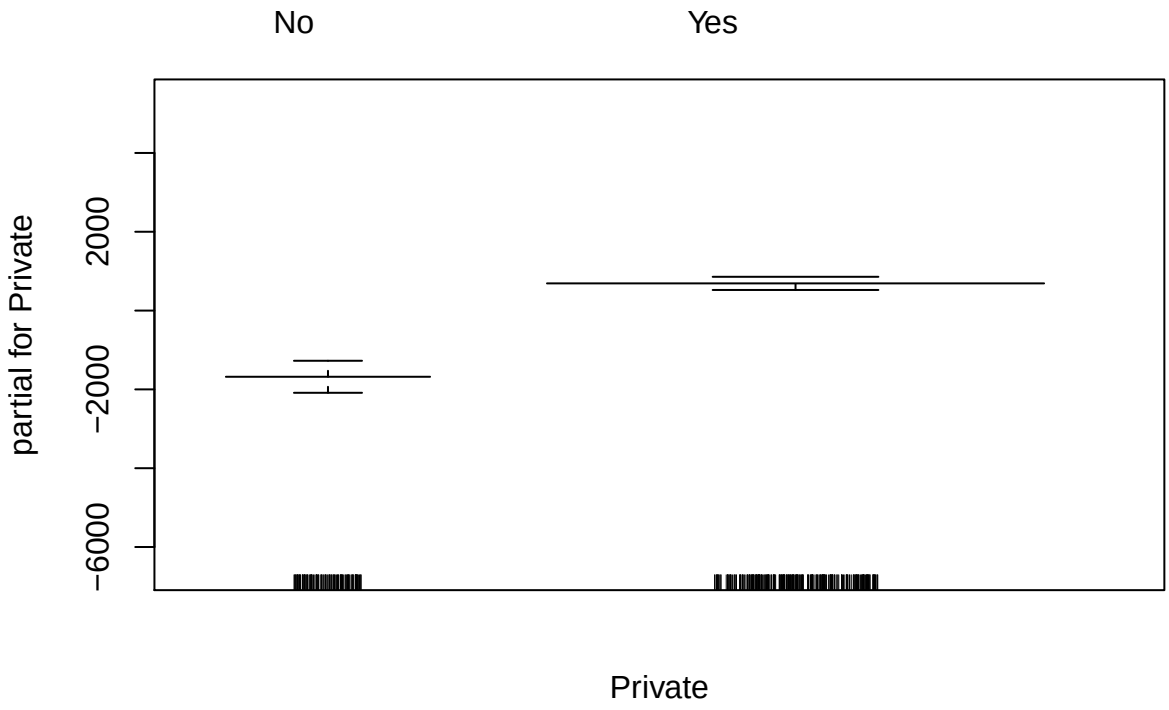

s(Expend, 3)
6000
2000
0
−4000
10000
20000
30000
40000
50000
Expend


s(Terminal, 2)
2000
−2000
−6000
40
60
80
100
Terminal


s(perc.alumni, 2)
6000
2000
0
−4000
0
10
20
30
40
50
60
perc.alumni


s(Room.Board, 2)
6000
2000
−2000
−6000
2000
3000
4000
5000
6000
7000
8000
Room.Board


partial for Grad.Rate
6000
2000
−2000
−6000
20
40
60
80
100
Grad.Rate



We further lower degrees, by making all quadratic terms linear now. We also introduce some interactions. It makes sense intuitively that the pricing behavior of private schools be more or less sensitive to different predictors, for example because profit-making is expected to play less of a role in their decision making. Hence, we introduced interactions of being a private school with the linear terms. Then we picked those that seemed more promising in the linear regression to be included in GAM. Since this is done on the training set, we do not worry about overfitting. We will finally be able to evaluate whether the model works on the test data.

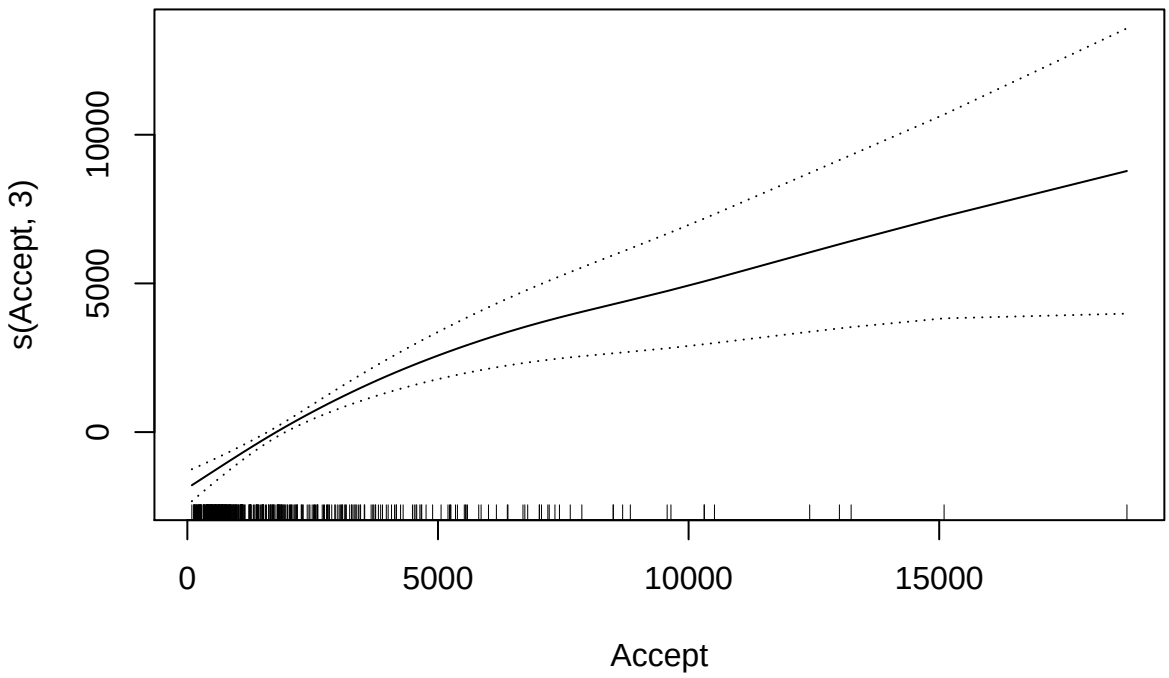
```
gam.fit.df31 = gam(Outstate ~ s(Accept, 3) + s(Enroll, 3) + s(Expend, 3) +
                           Books*Private + Terminal + perc.alumni + Room.Board*Private +
                           Private + Private + Top10perc,
                 data = College, subset = train)
summary(gam.fit.df31)
```

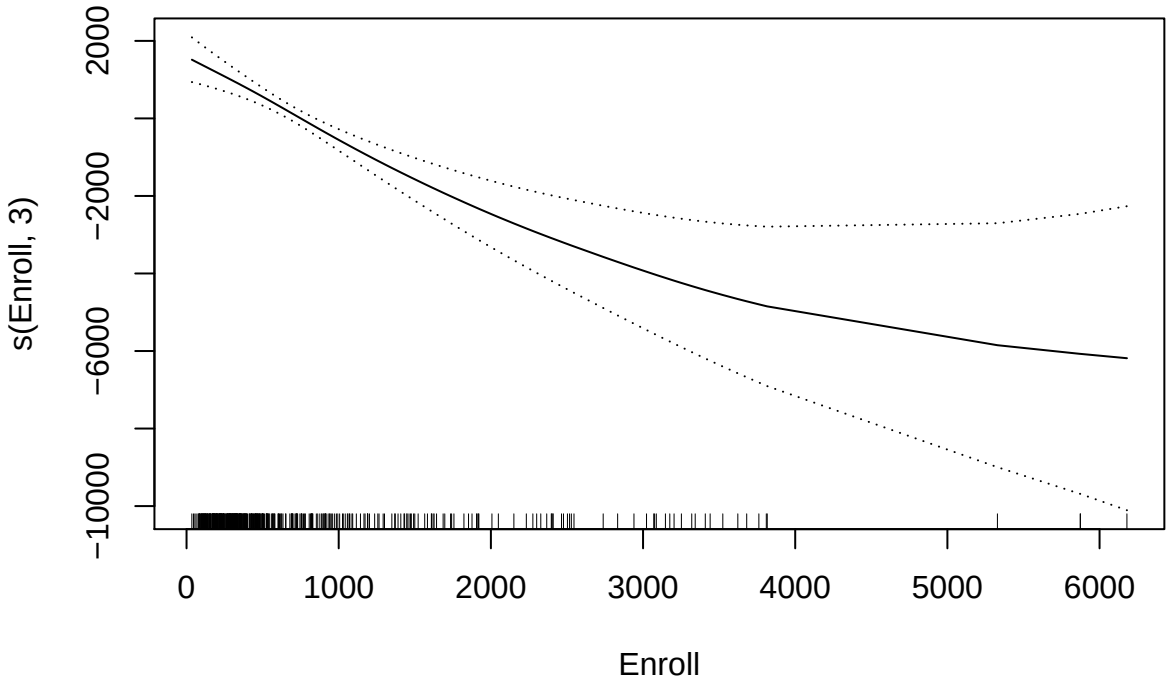
```
##
## Call: gam(formula = Outstate ~ s(Accept, 3) + s(Enroll, 3) + s(Expend,
##     3) + Books * Private + Terminal + perc.alumni + Room.Board *
##     Private + Private + Private + Top10perc, data = College,
##     subset = train)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -4900.85 -1165.99    58.61  1155.77  8676.32
##
## (Dispersion Parameter for gaussian family taken to be 3340420)
##
##     Null Deviance: 6306847377 on 418 degrees of freedom
## Residual Deviance: 1339508072 on 400.9999 degrees of freedom
```

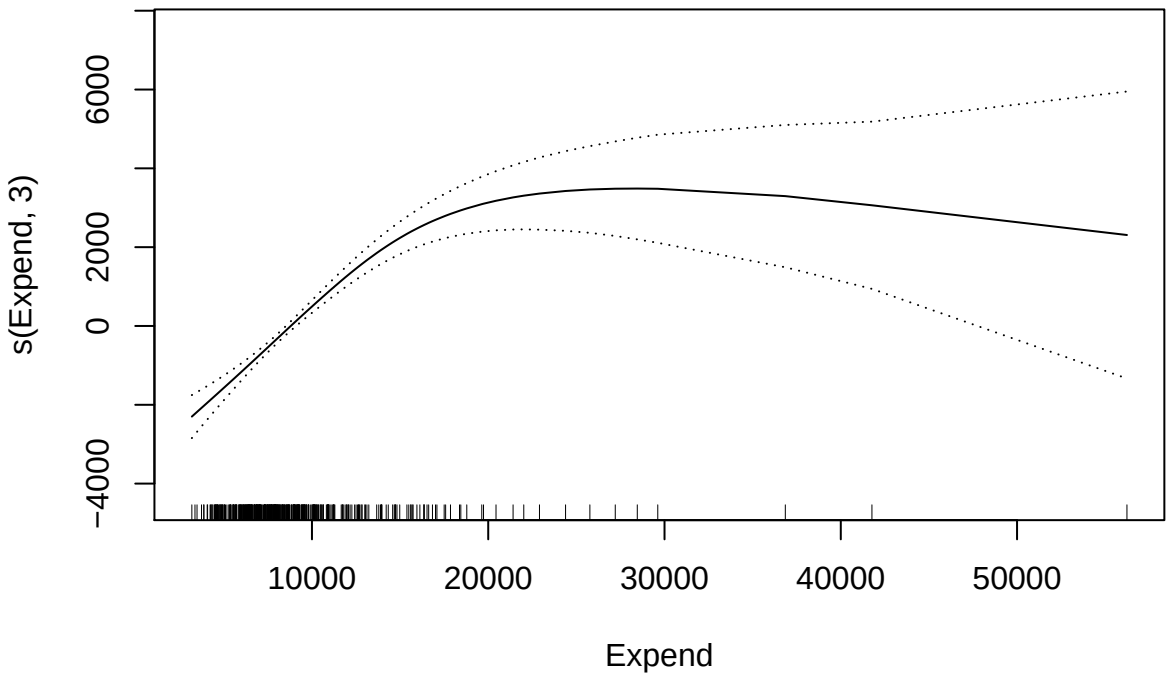
```
## AIC: 7502.726
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##                     Df     Sum Sq    Mean Sq  F value    Pr(>F)
## s(Accept, 3)         1    1498199    1498199   0.4485  0.503431
## s(Enroll, 3)         1 1063485880 1063485880 318.3689 < 2.2e-16 ***
## s(Expend, 3)         1 2137789077 2137789077 639.9762 < 2.2e-16 ***
## Books                1    5740427    5740427   1.7185  0.190640
## Private              1  560538061  560538061 167.8047 < 2.2e-16 ***
## Terminal             1  145161297  145161297  43.4560 1.369e-10 ***
## perc.alumni          1   66904527   66904527  20.0288 9.953e-06 ***
## Room.Board           1  182291512  182291512  54.5714 8.774e-13 ***
## Top10perc            1   13006086   13006086   3.8935  0.049158 *
## Books:Private        1   15946280   15946280   4.7737  0.029475 *
## Private:Room.Board   1   30931939   30931939   9.2599  0.002496 **
## Residuals          401 1339508072    3340420
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                     Npar Df  Npar F     Pr(F)
## (Intercept)
## s(Accept, 3)              2  9.0352 0.0001452 ***
## s(Enroll, 3)              2  6.8055 0.0012401 **
## s(Expend, 3)              2 29.6548 9.743e-13 ***
## Books
## Private
## Terminal
## perc.alumni
## Room.Board
## Top10perc
## Books:Private
## Private:Room.Board
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

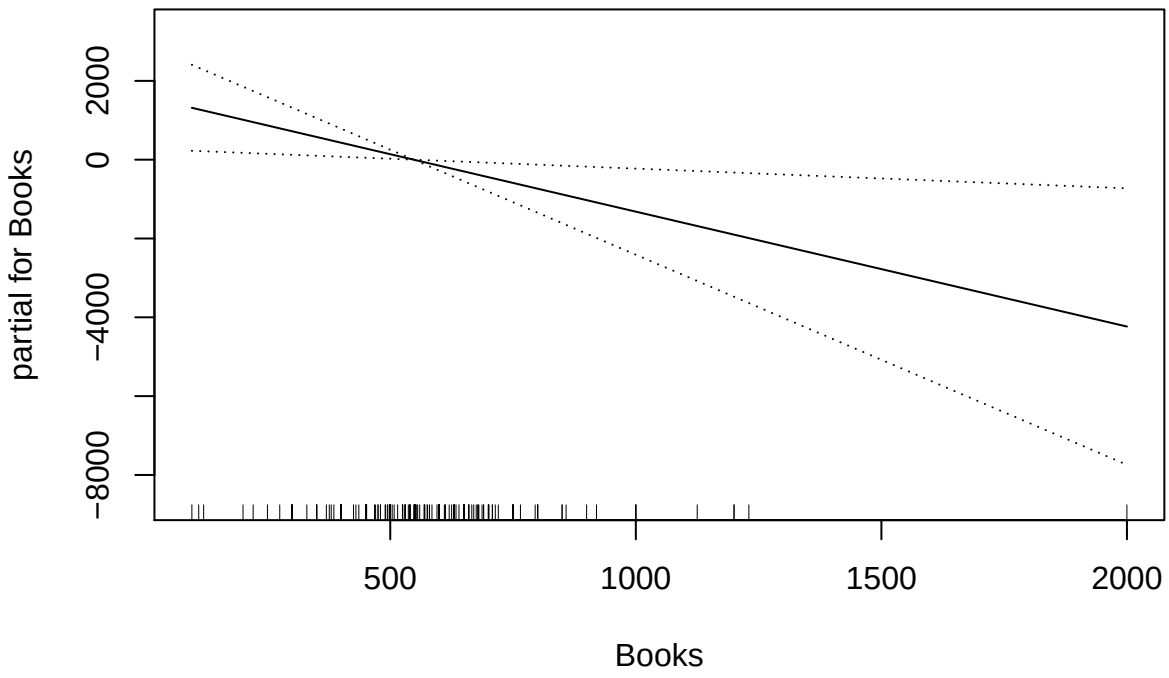
```
plot(gam.fit.df31, se = T, scale = 12000)
```

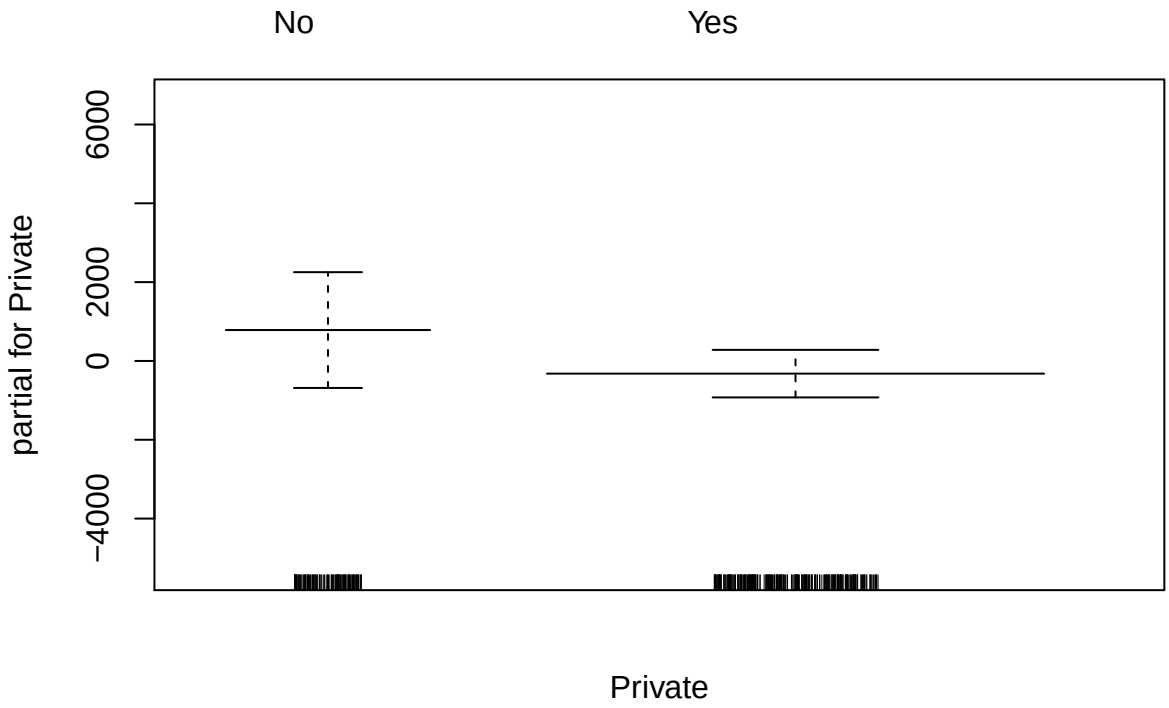
```
## Warning in preplot.gam(x, terms = terms): No terms saved for "a:b" style
## interaction terms
```

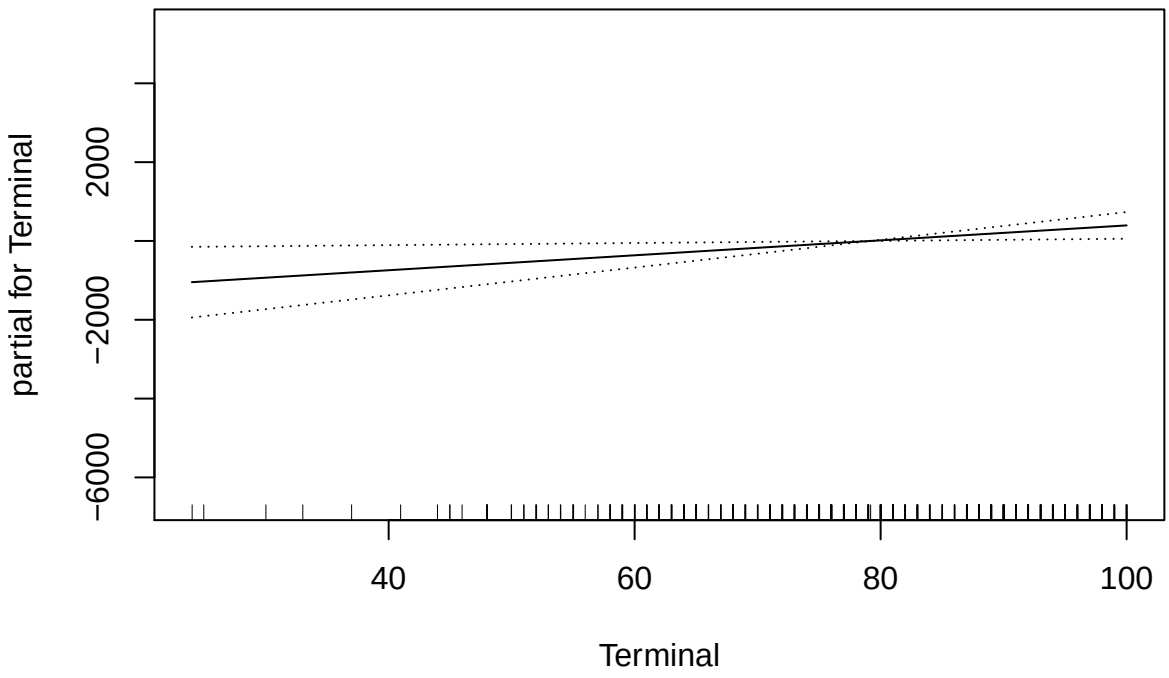

s(Accept, 3)
0
5000
10000
0
5000
10000
15000
Accept

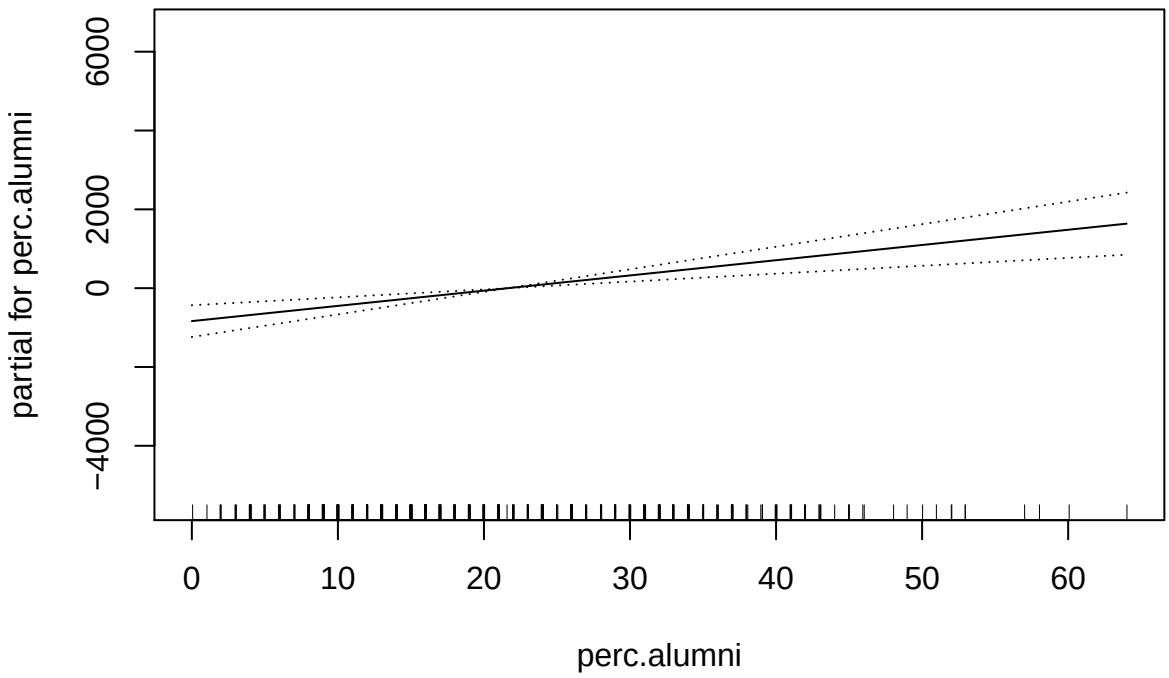

s(Enroll, 3)
2000
−2000
−6000
−10000
0
1000
2000
3000
4000
5000
6000
Enroll

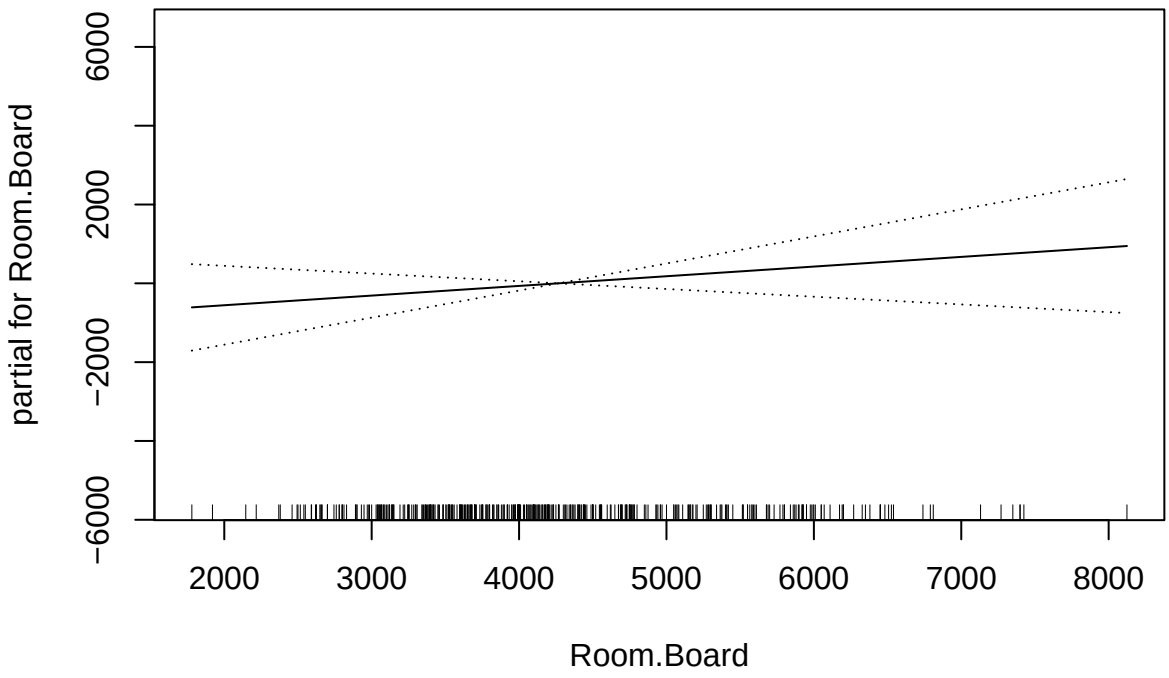

s(Expend, 3)
−4000
0
2000
6000
10000
20000
30000
40000
50000
Expend

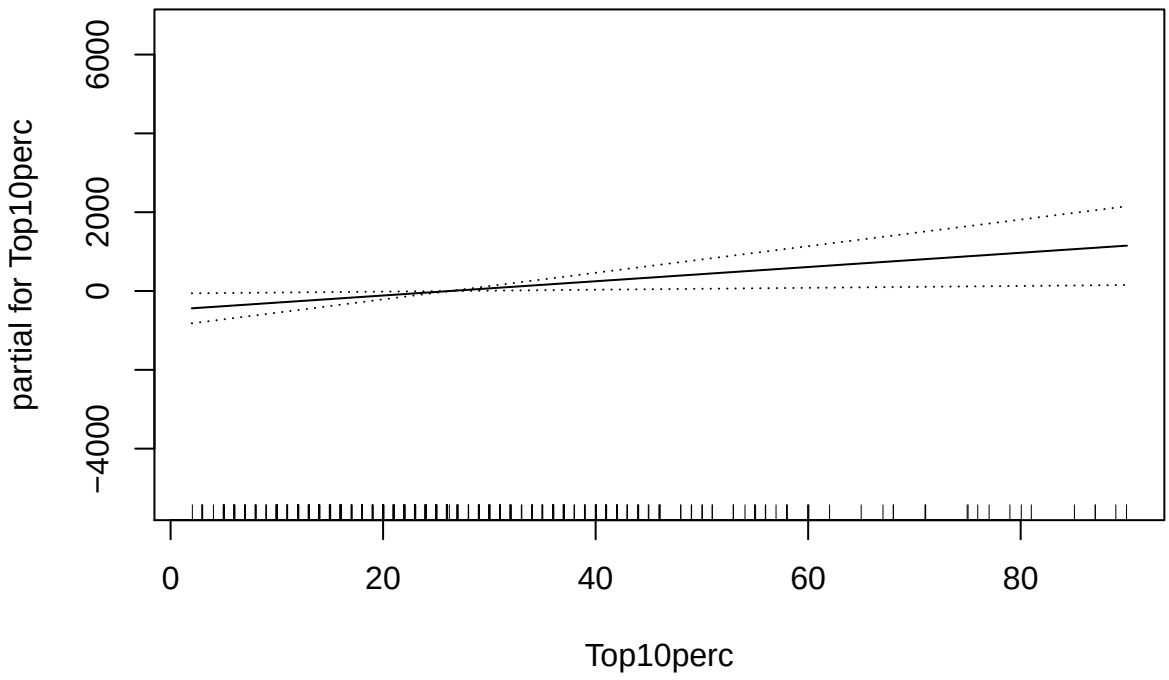

partial for Books
−8000
−4000
0
2000
500
1000
1500
2000
Books


No
Yes
partial for Private
6000
2000
0
−4000
Private


partial for Terminal
2000
−2000
−6000
40
60
80
100
Terminal


partial for perc.alumni
6000
2000
0
−4000
0
10
20
30
40
50
60
perc.alumni


partial for Room.Board
6000
2000
−2000
−6000
2000
3000
4000
5000
6000
7000
8000
Room.Board



Part 10.c)

We evaluate the models we estimated in the previous part.

```
# Setup
test.mse = rep(NA, length = 3)
gam.models = list(gam.fit.df3, gam.fit.df321, gam.fit.df31)
# predict for the test set
for (i in 1:length(gam.models)) {
  preds = predict(gam.models[[i]], newdata = College[test, ])
  test.mse[i] = mean((College$Outstate[test] - preds)^2)
}
names(test.mse) = c("df=3", "df=3,2,1", "Interact")
sqrt(test.mse)
```

```
##     df=3 df=3,2,1 Interact
## 1906.054 1905.415 1881.946
```

The last GAM model that we used (the one with interactions) has the least error, as indicated by square roots of MSE above.

```
mean(College$Outstate[test])
```

```
## [1] 10834.26
```

The model is doing a good job when comparing the average value of response to the squre root of MSE (which tends to be a bit higher than mean absolute value of differene), implying that the model has reasonable

predictive power.
In the test set, the actual tuition deviates from the predicted tuition by approximately $1880, on average. The mean value of tuition across the schools in the test set implies that the estimate for the precentage error is less than 20%. This is pretty good, when we note that it also includes the irreducible error.

Part 10.d)

There is clear evidence for non-linearity in Acceptance, Enrollment and Expenditure and evidence of interaction between `Private` and some of the variables.
Future work could consider the interaction of `Private` with the non-linear variables, and how it affects the test MSE. The drawback of such interaction, however, is that we could not see ANOVA results for interactions.

Exercise 11

Backfitting:

```
# SEtup
beta1 = -1
beta2 = 3
beta0 = 0
x1 = rnorm(100)
x2 = rnorm(100)
eps = rnorm(100)
y = beta0 + beta1*x1 + beta2*x2 + eps
# initialize
beta1hat = 3
# find partial residual for x2
r = y - beta1hat*x1
beta2hat = coef(lm(r ~ x2))[2]
```

Below, we run the codes for parts e), f) and g) of Question 11.

```
# initialize
n.iter = 1000
beta1hat = -100
param.values = matrix(NA, nrow = n.iter, ncol = 4,
                      dimnames = list(paste(1:n.iter),
                                      c("beta0_1" , "beta1hat", "beta0_2", "beta2hat")))
for (i in 1:n.iter) {
  # First regression: partial residual for x2
  r2 = y - beta1hat*x1
  coef.1 = coef(lm(r2 ~ x2))
  beta2hat = coef.1[2]
  param.values[i, 1:2] = coef.1
  # Second: partial for x1
  r1 = y - beta2hat*x2
  coef.2 = coef(lm(r1 ~ x1))
  beta1hat = coef.2[2]
  param.values[i, 3:4] = coef.2
}
c(beta1hat, beta2hat)
```

```
##         x1         x2
```

```
## -0.9533402  3.0948298
```

```
c(beta1, beta2)
```

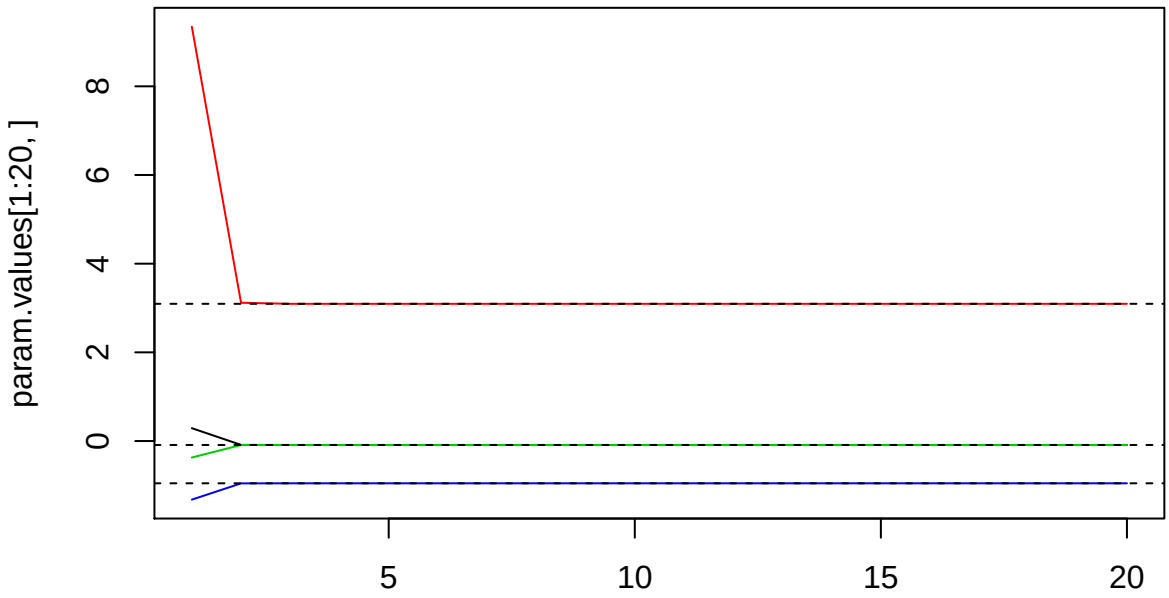
```
## [1] -1  3
```

Plot of the estimated values can be see below. The convergence happens at an exponentially fast pace, so no matter from what iteration we start, we would see a jump from it to the next iteration and a relatively flat line afterwards. To see the change, we start from the third set of fully backfitted models (otherwise, the plot would look flast starting at the second iteration) and focus on only one of the estimates to see the change (otherwise the scale would disable seeing much change):

```
matplot(param.values[1:20, ], type = "l", lty = 1)
coef.mult = coef(lm(y ~ x1 + x2))
coef.mult
```

```
## (Intercept)          x1          x2
##  -0.0887722  -0.9533402   3.0948298
```

```
abline(h = coef.mult, lty = 2)
```



The dotted lines depict the multiple regression results. Convergence happens very fast and lines are indistinguishable from the backfitting lines at the second iteration.

Exercise 12

Number of required backfitting iterations depends on n and p. If n=p, multiple regression could not fit and backfitting would not lead to findng population parameters (for example, for n=p = 9, 800 and 8000 iterations

both result in an error equal to 0.707). When n>p, backfitting finds the population parameters, but the more the dimensions are, i.e. where there is more overfitting, the more the iterations that are needed to achieve convergence (for example for p = 10 and n = 500, the error in 9 iterations is 2.606071e-11, while if we only change n to 50, the error in 9 iterations would be 0.002587096). *The code for this paragraph's numbers is not available. A variant on the code, which is convenient for drawing a plot is availble below:*

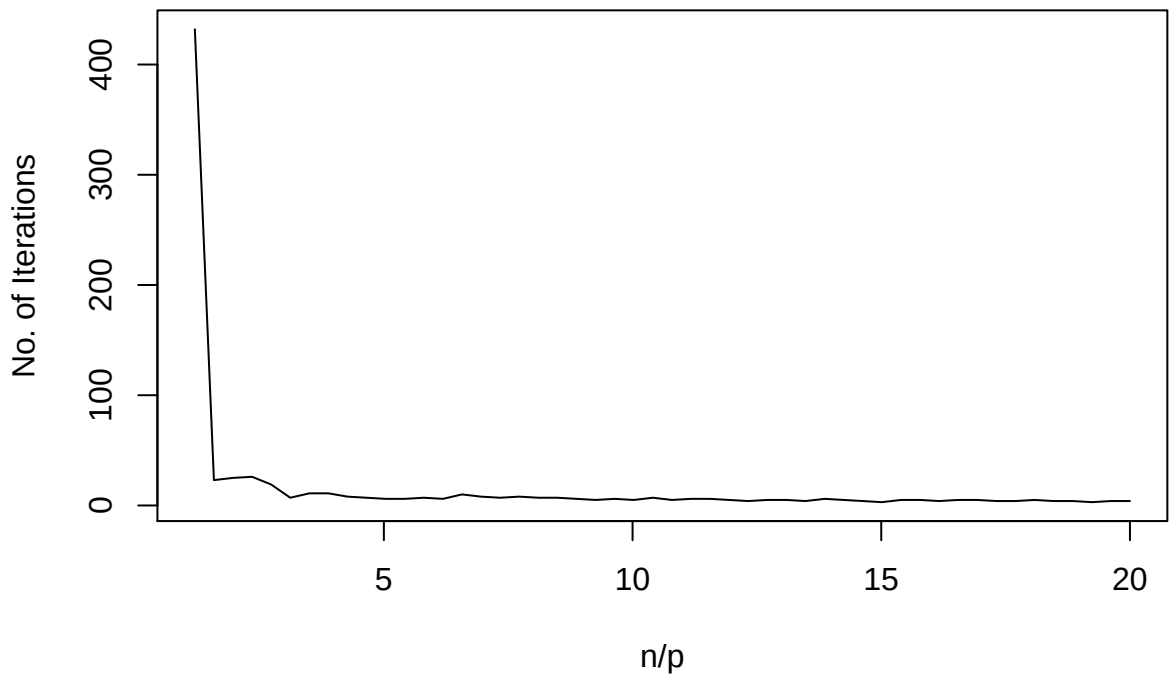
```
### Define a function that gets n, p (req, and tol as input and outputs the number of iterations needed
rm(list = ls())
backfit.n.iter = function(n, p, tol, max.n.iter) {
  ## Generate data
  # initialize
  set.seed(1)
  # generate features
  feat.mat = matrix(rnorm(n*p), nrow = n, ncol = p, dimnames = list(NULL, paste0("v", 1:p)))
  beta0 = 1
  Beta = rnorm(p)
  # generate response
  y = beta0 + feat.mat %*% Beta
  # generate dataset
  data.set = cbind(y, as.data.frame(feat.mat))

  ### compute multiple regression to compare with backfitting
  coef.mult = coef(lm(y ~ ., data = data.set))
  Beta.mult = coef.mult[-1]
  beta0.mult = coef.mult[1]

  ### backfitting
  ## intialize
  # initial values for backfitting
  Beta.back = rnorm(p)  # intial values for backfitting
  beta0.back = 1
  names(Beta.back) = paste0("v", 1:p)
  ## compute partial residuals
  # backfitting iterations
  n.iter = 0
  err = 10^10  # initialize with a large number
  while (err > tol & n.iter < max.n.iter) {
    # one iteration over all features:
    for (i in 1:p) {
      # partial residual of vi:
      r = y - feat.mat[, -i] %*% Beta.back[-i]
      # Update the Beta.back vector for i'th variable
      coef.vec = coef(lm(r ~ feat.mat[, i]))
      Beta.back[i] = coef.vec[2]  # since the first is for the intercept
      beta0.back = coef.vec[1]
    }
    err = max(max(abs(Beta.back - Beta.mult)), abs(beta0.mult - beta0.back))
    n.iter = n.iter + 1
  }
  if (n.iter >= max.n.iter) {
    n.iter = paste0("Covergence Not Achieved in", max.n.iter, " Iterations")
  }
  n.iter
}
```

A point to remember is that when we computed the partial residual, we only deducted the features and exluced the intercept terms.
A good plot would fix p, i.e. p = 100, and change n/p. Since p = 100 would make the code time-consuming, I plot the result using p = 10.

```
p = 10
n_p = seq(from = 1.2, to = 20, length = 50)
n.vec = ceiling(p*n_p)
n.iter.vec = rep(NA, length = length(n.vec))

for (i in 1:length(n.vec)) {
  n.iter.vec[i] = backfit.n.iter(n = n.vec[i], p, tol = 1e-3, max.n.iter = 500)
}
n.vec[2]
```

```
## [1] 16
```

```
plot(n_p, n.iter.vec, xlab = "n/p", ylab = "No. of Iterations", type = "l")
```



Below is a closer look at how the speed of convergence decreases as we increase n/p.

```
p = 10
n_p = seq(from = 1.2, to = 1.5, length = 20)
n.vec = ceiling(p*n_p)
n.iter.vec = rep(NA, length = length(n.vec))

for (i in 1:length(n.vec)) {
  n.iter.vec[i] = backfit.n.iter(n = n.vec[i], p, tol = 1e-3, max.n.iter = 500)
```

```
}
plot(n_p, n.iter.vec, xlab = "n/p", ylab = "No. of Iterations", type = "l")
```