ISLR CH7 Exercises

```
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------------
```

```
## v ggplot2 3.3.0     v purrr   0.3.3
## v tibble  3.0.1     v dplyr   0.8.5
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
# apply wors like reduce() (sum over columns (or rows) of a matrix or a tibble)
(m1 <- matrix(C<-(1:10),nrow=5, ncol=6))
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    6    1    6    1    6
## [2,]    2    7    2    7    2    7
## [3,]    3    8    3    8    3    8
## [4,]    4    9    4    9    4    9
## [5,]    5   10    5   10    5   10
```

```
(a_m1 <- apply(m1, 2, sum))
```

```
## [1] 15 40 15 40 15 40
```

```
# apply on tibble (reduce on columns)
(a_tibble <- as_tibble(m1))
```

```
## Warning: The `x` argument of `as_tibble.matrix()` must have column names if `.name_repair` is omitte
## Using compatibility `.name_repair`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
## # A tibble: 5 x 6
##      V1    V2    V3    V4    V5    V6
##   <int> <int> <int> <int> <int> <int>
## 1     1     6     1     6     1     6
## 2     2     7     2     7     2     7
## 3     3     8     3     8     3     8
## 4     4     9     4     9     4     9
## 5     5    10     5    10     5    10
```

```
(a_m2 <- apply(a_tibble, 2, sum))
```

```
## V1 V2 V3 V4 V5 V6
## 15 40 15 40 15 40
```

```
# lApply (apply a function over a list (or tibble as a list of columns) and return a new list)
movies <- c("SPYDERMAN","BATMAN","VERTIGO","CHINATOWN")
```

```
movies_lower <-lapply(movies, tolower)
str(movies_lower)
```

```
## List of 4
##  $ : chr "spyderman"
##  $ : chr "batman"
##  $ : chr "vertigo"
##  $ : chr "chinatown"
```

```
# lapply on tible returns a list and then
(a_list <- lapply(a_tibble, function(x) {x*2}))
```

```
## $V1
## [1]  2  4  6  8 10
##
## $V2
## [1] 12 14 16 18 20
##
## $V3
## [1]  2  4  6  8 10
##
## $V4
## [1] 12 14 16 18 20
##
## $V5
## [1]  2  4  6  8 10
##
## $V6
## [1] 12 14 16 18 20
```

```
# sapply() function takes list, vector or data frame as input and gives output
# in vector or matrix
(matrix <-sapply(a_tibble, function(x) {x*2}))
```

```
##      V1 V2 V3 V4 V5 V6
## [1,]  2 12  2 12  2 12
## [2,]  4 14  4 14  4 14
## [3,]  6 16  6 16  6 16
## [4,]  8 18  8 18  8 18
## [5,] 10 20 10 20 10 20
```

```
# We can use lapply() or sapply() interchangeable to slice a data frame
below_ave <- function(x) {
    ave <- mean(x)
    return(x[x > ave])
}

(dt_s<- as_tibble(sapply(a_tibble, below_ave)))
```

```
## # A tibble: 2 x 6
##      V1    V2    V3    V4    V5    V6
##   <int> <int> <int> <int> <int> <int>
## 1     4     9     4     9     4     9
## 2     5    10     5    10     5    10
```

```
(dt_l<- as_tibble(lapply(a_tibble, below_ave)))
```

```
## # A tibble: 2 x 6
##      V1    V2    V3    V4    V5    V6
##   <int> <int> <int> <int> <int> <int>
## 1     4     9     4     9     4     9
## 2     5    10     5    10     5    10
```

```
identical(dt_s, dt_l)
```

```
## [1] TRUE
```

```
# tapply() computes a measure (mean, median, min, max, etc..) or a function for
# each factor variable in a vector. It is a very useful function that lets you
# create a subset of a vector and then apply some functions to each of the subset.

# -X: An object, usually a vector
# -INDEX: A list containing factor
# -FUN: Function applied to each element of x

#As a prior work, we can compute the median of the length for each species.
# tapply() is a quick way to perform this computation.

tapply(matrix(1:6, c(6,1)), c(1,1,1,2,2,1),sum)
```

```
##  1  2
## 12  9
```

```
data(iris)
tapply(iris$Sepal.Width, iris$Species, median)
```

```
##     setosa versicolor  virginica
##        3.4        2.8        3.0
```

```
str(iris$Sepal.Width)
```

```
##  num [1:150] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
```

```
# remove any row that has at least one empty string value
(df1 <- tibble(x=c("" , " ", "  abc  " ,"", "  de","f  "),
               y=c("12", "  54", "  ", "  c12  ", "  ", "  No  "))
)
```

```
## # A tibble: 6 x 2
##   x         y
##   <chr>     <chr>
## 1 ""        "12"
## 2 " "       "  54"
## 3 "  abc  " "  "
## 4 ""        "  c12  "
## 5 "  de"    "  "
## 6 "f  "     "  No  "
```

```
(df2 <- as_tibble(sapply(df1, function(the_col) gsub("\\s+", "", the_col))))
```

```
## # A tibble: 6 x 2
##   x     y
##   <chr> <chr>
```

```
## 1 ""    "12"
## 2 ""    "54"
## 3 "abc" ""
## 4 ""    "c12"
## 5 "de"  ""
## 6 "f"   "No"
```

```
trim.f <- function(x) trimws(x, which = c("both"))
space.f <- function(x) gsub("\\s+", NA, x)
empty.f <- function(x) gsub("^$", NA, x)
library(tidyverse)
df1 %>%
  mutate_each(funs(trim.f)) %>%
  mutate_each(funs(space.f)) %>%
  mutate_each(funs(empty.f))  %>%
  na.omit
```

```
## Warning: funs() is soft deprecated as of dplyr 0.8.0
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
## This warning is displayed once per session.
```

```
## # A tibble: 1 x 2
##   x     y
##   <chr> <chr>
## 1 f     No
```

```
library(tidyverse)
set.seed(1)
wage.df = read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Wage.csv",
                     header=T, stringsAsFactors = F, na.strings = "?")
(wage.df.original = tibble(wage.df))
```

```
## # A tibble: 3,000 x 12
##     year   age sex   maritl race  education region jobclass health health_ins
##    <int> <int> <chr> <chr>  <chr> <chr>     <chr>  <chr>    <chr>  <chr>
##  1  2006    18 1. M~ 1. Ne~ 1. W~ 1. < HS ~ 2. Mi~ 1. Indu~ 1. <=~ 2. No
##  2  2004    24 1. M~ 1. Ne~ 1. W~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 2. No
##  3  2003    45 1. M~ 2. Ma~ 1. W~ 3. Some ~ 2. Mi~ 1. Indu~ 1. <=~ 1. Yes
##  4  2003    43 1. M~ 2. Ma~ 3. A~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
##  5  2005    50 1. M~ 4. Di~ 1. W~ 2. HS Gr~ 2. Mi~ 2. Info~ 1. <=~ 1. Yes
##  6  2008    54 1. M~ 2. Ma~ 1. W~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
##  7  2009    44 1. M~ 2. Ma~ 4. O~ 3. Some ~ 2. Mi~ 1. Indu~ 2. >=~ 1. Yes
##  8  2008    30 1. M~ 1. Ne~ 3. A~ 3. Some ~ 2. Mi~ 2. Info~ 1. <=~ 1. Yes
##  9  2006    41 1. M~ 1. Ne~ 2. B~ 3. Some ~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
## 10  2004    52 1. M~ 2. Ma~ 1. W~ 2. HS Gr~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
## # ... with 2,990 more rows, and 2 more variables: logwage <dbl>, wage <dbl>
```

```
(wage.df = tibble(wage.df))
```

```
## # A tibble: 3,000 x 12
##     year   age sex   maritl race  education region jobclass health health_ins
##    <int> <int> <chr> <chr>  <chr> <chr>     <chr>  <chr>    <chr>  <chr>
##  1  2006    18 1. M~ 1. Ne~ 1. W~ 1. < HS ~ 2. Mi~ 1. Indu~ 1. <=~ 2. No
##  2  2004    24 1. M~ 1. Ne~ 1. W~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 2. No
##  3  2003    45 1. M~ 2. Ma~ 1. W~ 3. Some ~ 2. Mi~ 1. Indu~ 1. <=~ 1. Yes
##  4  2003    43 1. M~ 2. Ma~ 3. A~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
##  5  2005    50 1. M~ 4. Di~ 1. W~ 2. HS Gr~ 2. Mi~ 2. Info~ 1. <=~ 1. Yes
##  6  2008    54 1. M~ 2. Ma~ 1. W~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
##  7  2009    44 1. M~ 2. Ma~ 4. O~ 3. Some ~ 2. Mi~ 1. Indu~ 2. >=~ 1. Yes
##  8  2008    30 1. M~ 1. Ne~ 3. A~ 3. Some ~ 2. Mi~ 2. Info~ 1. <=~ 1. Yes
##  9  2006    41 1. M~ 1. Ne~ 2. B~ 3. Some ~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
## 10  2004    52 1. M~ 2. Ma~ 1. W~ 2. HS Gr~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
## # ... with 2,990 more rows, and 2 more variables: logwage <dbl>, wage <dbl>
```

```
# first clean up NA and particularly character columns
# 1) remove leading, trainling and empty characters from character columns

trim.f <- function(x) trimws(x, which = c("both")) # leading and trailing spaces
empty.f <- function(x) gsub("^$", NA, x) # empty strings

wage.df[sapply(wage.df, is.character)] <-
  wage.df %>%
    select(which(sapply(.,is.character))) %>%
    mutate_each(funs(trim.f)) %>%
    mutate_each(funs(empty.f))  %>%
    na.omit


sprintf("Any NA or empty string in character columns recognized: %s",
        !identical(wage.df,wage.df.original) )
```

```
## [1] "Any NA or empty string in character columns recognized: FALSE"
```

```
# 2) It is safe now to convert all character fields into factors
wage.df[sapply(wage.df, is.character)] <-
  wage.df %>%
  select(which(sapply(., is.character))) %>%
  mutate_each(funs(factor))


# Now fit a 4 degree polynomial
fit.poly <- lm(wage ~ poly(age, 4), data = wage.df)
coef(summary(fit.poly))
```

```
##                 Estimate Std. Error    t value     Pr(>|t|)
## (Intercept)    111.70361  0.7287409 153.283015 0.000000e+00
## poly(age, 4)1  447.06785 39.9147851  11.200558 1.484604e-28
## poly(age, 4)2 -478.31581 39.9147851 -11.983424 2.355831e-32
## poly(age, 4)3  125.52169 39.9147851   3.144742 1.678622e-03
## poly(age, 4)4  -77.91118 39.9147851  -1.951938 5.103865e-02
```

```
# draw standard error
# first get the range of the values of age
(age.limits  <- range(wage.df$age))
```

```
## [1] 18 80
```

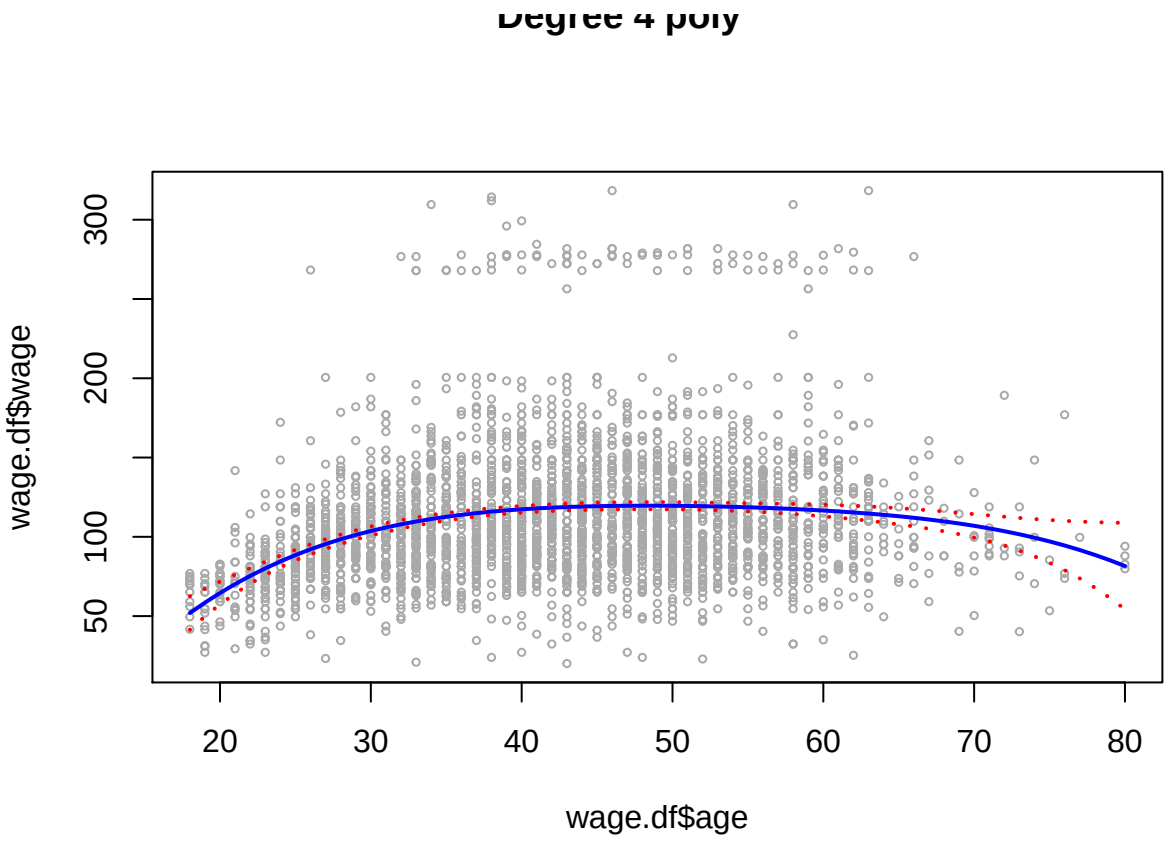
```
# Now create an interaval from these range values
(age.grid <- seq(from=age.limits[1], to=age.limits[2]))
```

```
##  [1] 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
## [26] 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67
## [51] 68 69 70 71 72 73 74 75 76 77 78 79 80
```

```
# predict value for this interval using fitted model
predicts <- predict(fit.poly, newdata = tibble(age=age.grid), se=T)
names(predicts)
```

```
## [1] "fit"            "se.fit"         "df"             "residual.scale"
```

```
se.bands <- cbind(predicts$fit + 2*predicts$se.fit,
                  predicts$fit - 2*predicts$se.fit)

# now plot the predicts
#par(mfrow=c(1,2), mar=c(4.5,4.5,1,1), oma=c(0,0,4,0))
plot(wage.df$age, wage.df$wage, xlim=age.limits, cex=0.5, col="darkgrey")
title("Degree 4 poly", outer = T)
lines(age.grid, predicts$fit, lwd=2, col = "blue")
matlines(age.grid, se.bands, lwd=2, col="red", lty=3)
```



```r
# use anova to find the best model between multiple nested models
fit.1 <- lm(wage ~ age, data = wage.df)
fit.2 <- lm(wage ~ poly(age, 2), data = wage.df)
fit.3 <- lm(wage ~ poly(age, 3), data = wage.df)
fit.4 <- lm(wage ~ poly(age, 4), data = wage.df)
fit.5 <- lm(wage ~ poly(age, 5), data = wage.df)

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

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

```r
# ------------------------------------------------------------------------------
# However the best approach is using CV to choose best degree for polynomial
#-------------------------------------------------------------------------------
# create k-fold
k <- 10
degrees <- 1:10
set.seed(1)

# create k folds
folds <- sample(1:k, nrow(wage.df), replace = T)

# For folds with same size do:
# sameSizefolds <- sample(rep(1:k, length.out = nrow(weekly.df)),
#                                       size = nrow(weekly.df), replace = F)
# table(sameSizefolds)

# perform a cross validation on a for loop
mse.per.fold <- tibble(fold.id = NULL, degree = NULL, mse=NULL)
for (dgr in degrees){
  for (j in 1:k){

    # fit the polynomial model for given degree on training fold

    fit.poly <- lm(wage ~ poly(age, dgr), data = wage.df[folds != j, ])
    # now predict on test fold
    predicts <- predict(fit.poly,
                        newdata = list(age=wage.df[folds == j, ]$age), se=T)

    # calculte the MSE
```

```
    mse.per.fold <-
      rbind(mse.per.fold,
            tibble(fold.id = j, degree = dgr,
                   mse=mean((predicts$fit - wage.df[folds == j, ]$wage)^2)))
  }
}

# We have to find average of mse rate for each degree cross all test folds


(summary <- mse.per.fold %>%
  group_by(degree) %>%
  summarise(mse.mean = mean(mse))
)
```

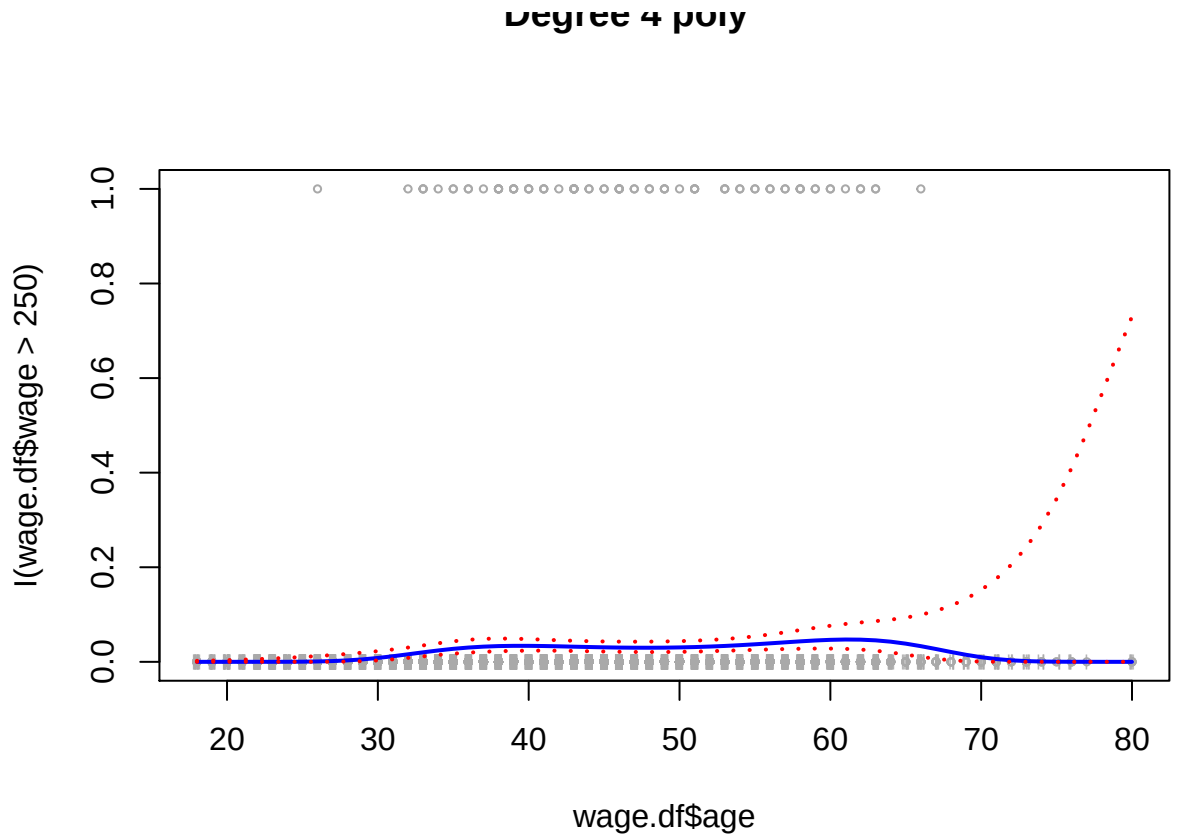
```
## # A tibble: 10 x 2
##    degree mse.mean
##     <int>    <dbl>
##  1      1    1676.
##  2      2    1600.
##  3      3    1594.
##  4      4    1593.
##  5      5    1593.
##  6      6    1593.
##  7      7    1593.
##  8      8    1594.
##  9      9    1593.
## 10     10    1594.
```

```
# and then find the degree that has minimum mean.mse
summary %>%
  slice(which.min(mse.mean))
```

```
## # A tibble: 1 x 2
##   degree mse.mean
##    <int>    <dbl>
## 1      6    1593.
```

```
# Now let's predict if an individual earns more than 20k per year

fit <- glm(I(wage > 250) ~ poly(age,4), data = wage.df, family = binomial)

# do the prediction using the model
preds <- predict(fit, newdata=list(age=age.grid), se = T)

# Note that predict function for glm model gets the prediction for logit i.e:
# log(Pr(Y=1 | X) / (1 - Pr(Y=1 | X))) = X*beta and also the standard error is
# of the same form, clealry to get prediction for Pr(Y=1|X) we need
# to calculate exp(X*beta)/(1+exp(X*beta))

prob.predict <- exp(preds$fit)/(1+exp(preds$fit))
se.bound.logit <- cbind(preds$fit + 2*preds$se, preds$fit - 2*preds$se)
se.bound <- apply(se.bound.logit, 2, function(x) exp(x)/(1+exp(x)))

# now plot the predicts
```

```
#par(mfrow=c(1,2), mar=c(4.5,4.5,1,1), oma=c(0,0,4,0))
plot(wage.df$age, I(wage.df$wage > 250),
     xlim=age.limits, cex=0.5, col="darkgrey")
# we use jitter so that observation with the same value do not cover each other
points(jitter(wage.df$age), I(wage.df$age>250)/5,
       cex=0.5, pch="|", col="darkgrey")
title("Degree 4 poly", outer = T)
lines(age.grid, prob.predict, lwd=2, col = "blue")
matlines(age.grid, se.bound, lwd=2, col="red", lty=3)
```




```
print("----------- let's fit a step function --------------")
```

```
## [1] "----------- let's fit a step function --------------"
```

```
# we use cut() function to create knots
table(cut(wage.df$age, 4))
```

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

```
print("fit a step function using cut() or giving our cut points using break(): ")
```

```
## [1] "fit a step function using cut() or giving our cut points using break(): "
```

```
# let's fit a step function
fit.step <- lm(wage ~ cut(age, 4), data=wage.df)
coef(summary(fit))
```

```
##                 Estimate Std. Error    z value     Pr(>|z|)
```

```
## (Intercept)    -4.301228  0.3450738 -12.464663 1.163621e-35
## poly(age, 4)1  71.964174 26.1175633   2.755394 5.862151e-03
## poly(age, 4)2 -85.772899 35.9043225  -2.388930 1.689754e-02
## poly(age, 4)3  34.162564 19.6889645   1.735112 8.272092e-02
## poly(age, 4)4 -47.400800 24.0909406  -1.967578 4.911664e-02
```

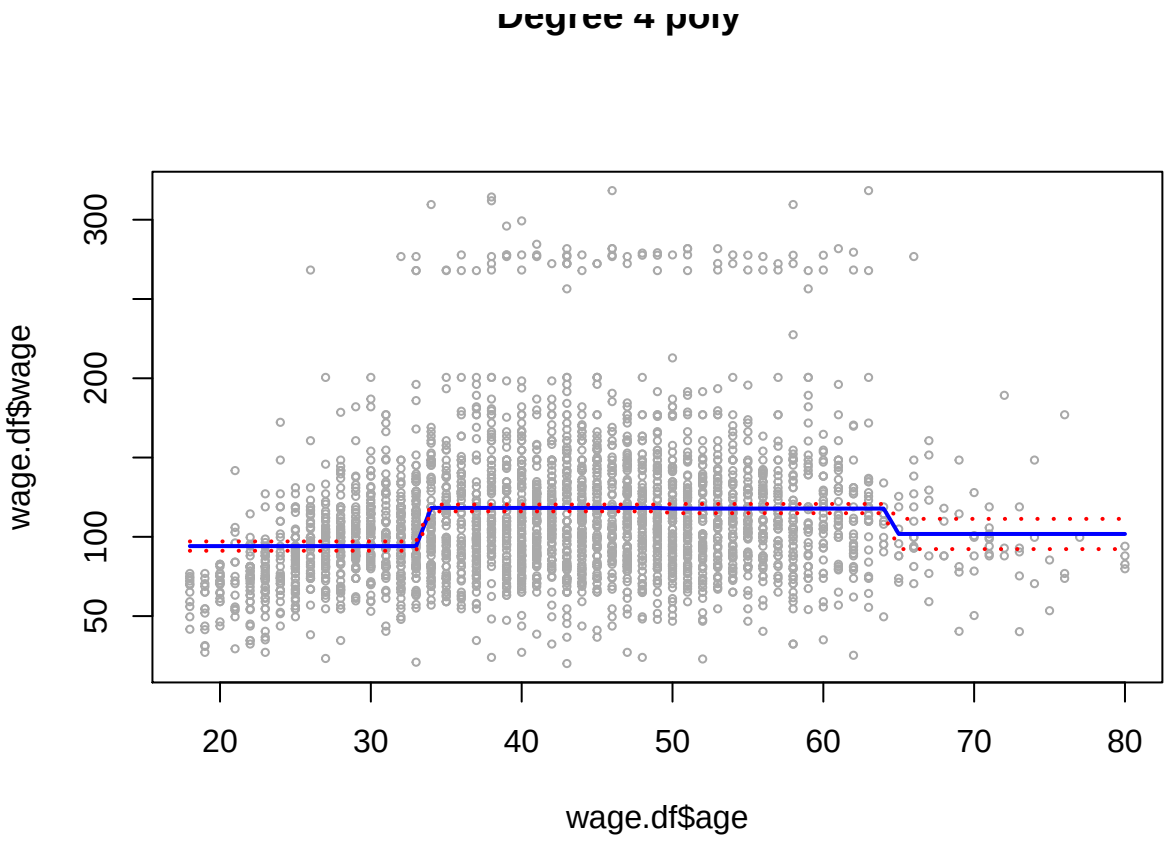
```
(age.limits  <- range(wage.df$age))
```

```
## [1] 18 80
```

```
# Now create an interaval from these range values
(age.grid <- seq(from=age.limits[1], to=age.limits[2]))
```

```
##  [1] 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
## [26] 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67
## [51] 68 69 70 71 72 73 74 75 76 77 78 79 80
```

```
# predict value for this interval using fitted model
predicts <- predict(fit.step, newdata = tibble(age=age.grid), se=T)

se.bands <- cbind(predicts$fit + 2*predicts$se.fit,
                  predicts$fit - 2*predicts$se.fit)

# now plot the predicts
#par(mfrow=c(1,2), mar=c(4.5,4.5,1,1), oma=c(0,0,4,0))
plot(wage.df$age, wage.df$wage, xlim=age.limits, cex=0.5, col="darkgrey")
title("Degree 4 poly", outer = T)
lines(age.grid, predicts$fit, lwd=2, col = "blue")
matlines(age.grid, se.bands, lwd=2, col="red", lty=3)
```



```
library(tidyverse)
library(splines)
wage.df = read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Wage.csv",
                     header=T, stringsAsFactors = F, na.strings = "?")
(wage.df.original = tibble(wage.df))
```

```
## # A tibble: 3,000 x 12
##     year   age sex   maritl race  education region jobclass health health_ins
##    <int> <int> <chr> <chr>  <chr> <chr>     <chr>  <chr>    <chr>  <chr>
##  1  2006    18 1. M~ 1. Ne~ 1. W~ 1. < HS ~ 2. Mi~ 1. Indu~ 1. <=~ 2. No
##  2  2004    24 1. M~ 1. Ne~ 1. W~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 2. No
##  3  2003    45 1. M~ 2. Ma~ 1. W~ 3. Some ~ 2. Mi~ 1. Indu~ 1. <=~ 1. Yes
##  4  2003    43 1. M~ 2. Ma~ 3. A~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
##  5  2005    50 1. M~ 4. Di~ 1. W~ 2. HS Gr~ 2. Mi~ 2. Info~ 1. <=~ 1. Yes
##  6  2008    54 1. M~ 2. Ma~ 1. W~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
##  7  2009    44 1. M~ 2. Ma~ 4. O~ 3. Some ~ 2. Mi~ 1. Indu~ 2. >=~ 1. Yes
##  8  2008    30 1. M~ 1. Ne~ 3. A~ 3. Some ~ 2. Mi~ 2. Info~ 1. <=~ 1. Yes
##  9  2006    41 1. M~ 1. Ne~ 2. B~ 3. Some ~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
## 10  2004    52 1. M~ 2. Ma~ 1. W~ 2. HS Gr~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
## # ... with 2,990 more rows, and 2 more variables: logwage <dbl>, wage <dbl>
```

```
(wage.df = tibble(wage.df))
```

```
## # A tibble: 3,000 x 12
##     year   age sex   maritl race  education region jobclass health health_ins
##    <int> <int> <chr> <chr>  <chr> <chr>     <chr>  <chr>    <chr>  <chr>
##  1  2006    18 1. M~ 1. Ne~ 1. W~ 1. < HS ~ 2. Mi~ 1. Indu~ 1. <=~ 2. No
##  2  2004    24 1. M~ 1. Ne~ 1. W~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 2. No
##  3  2003    45 1. M~ 2. Ma~ 1. W~ 3. Some ~ 2. Mi~ 1. Indu~ 1. <=~ 1. Yes
##  4  2003    43 1. M~ 2. Ma~ 3. A~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
##  5  2005    50 1. M~ 4. Di~ 1. W~ 2. HS Gr~ 2. Mi~ 2. Info~ 1. <=~ 1. Yes
##  6  2008    54 1. M~ 2. Ma~ 1. W~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
##  7  2009    44 1. M~ 2. Ma~ 4. O~ 3. Some ~ 2. Mi~ 1. Indu~ 2. >=~ 1. Yes
##  8  2008    30 1. M~ 1. Ne~ 3. A~ 3. Some ~ 2. Mi~ 2. Info~ 1. <=~ 1. Yes
##  9  2006    41 1. M~ 1. Ne~ 2. B~ 3. Some ~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
## 10  2004    52 1. M~ 2. Ma~ 1. W~ 2. HS Gr~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
## # ... with 2,990 more rows, and 2 more variables: logwage <dbl>, wage <dbl>
```

```
# first clean up NA and particularly character columns
# 1) remove leading, trainling and empty characters from character columns

trim.f <- function(x) trimws(x, which = c("both")) # leading and trailing spaces
empty.f <- function(x) gsub("^$", NA, x) # empty strings

wage.df[sapply(wage.df, is.character)] <-
  wage.df %>%
    select(which(sapply(.,is.character))) %>%
    mutate_each(funs(trim.f)) %>%
    mutate_each(funs(empty.f))  %>%
    na.omit


sprintf("Any NA or empty string in character columns recognized: %s",
        !identical(wage.df,wage.df.original) )
```

```
## [1] "Any NA or empty string in character columns recognized: FALSE"
```

```
# 2) It is safe now to convert all character fields into factors
wage.df[sapply(wage.df, is.character)] <-
  wage.df %>%
  select(which(sapply(., is.character))) %>%
  mutate_each(funs(factor))

# use df() to generate spline with knots at uniform quantiles
dim(bs(wage.df$age, knots=c(25,40,60)))
```

```
## [1] 3000    6
```

```
dim(bs(wage.df$age, df=6))
```

```
## [1] 3000    6
```

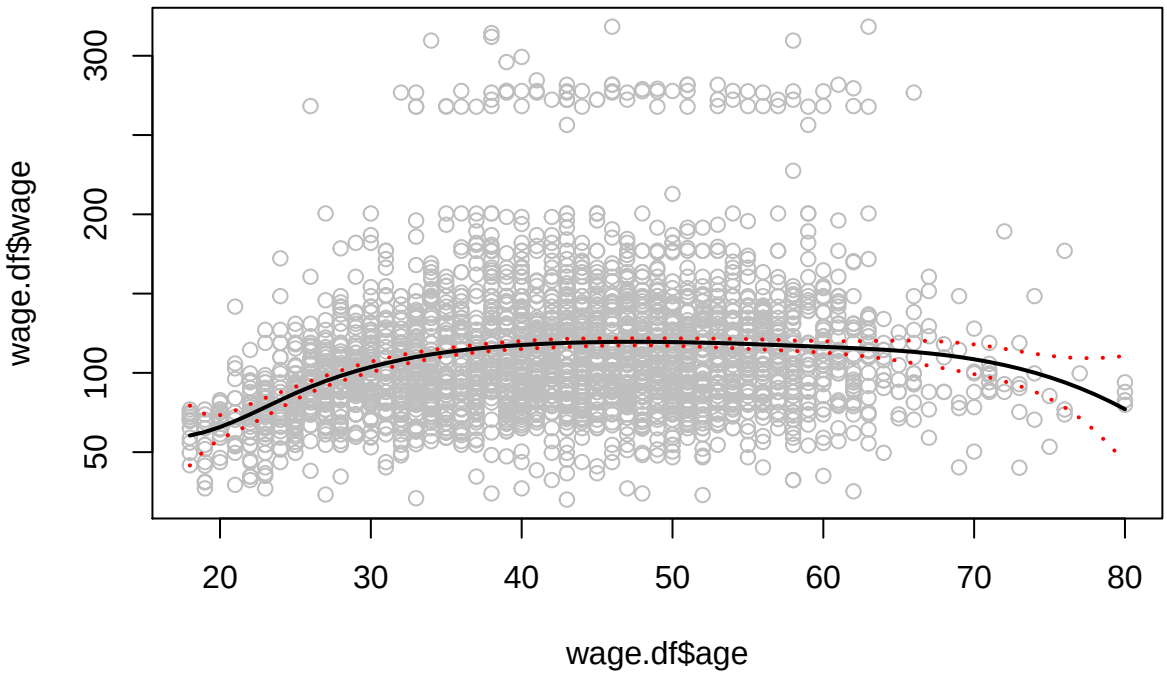
```
# Fit wage to age using regression splines
fit <- lm(wage~bs(age, knots = c(25,40,60)) ,data=wage.df)

# first get the range of the values of age
(age.limits  <- range(wage.df$age))
```

```
## [1] 18 80
```

```
# Now create an interaval from these range values
(age.grid <- seq(from=age.limits[1], to=age.limits[2]))
```

```
##  [1] 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
## [26] 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67
## [51] 68 69 70 71 72 73 74 75 76 77 78 79 80
```

```
# predict the age values
pred <- predict(fit, newdata = list(age=age.grid), se=T)
plot(wage.df$age, wage.df$wage, col="gray")
lines(age.grid, pred$fit, lwd=2)
se_bonds <- cbind(pred$fit+2*pred$se, pred$fit-2*pred$se)
matlines(age.grid, se_bonds, lwd=2, col="red", lty=3)
```



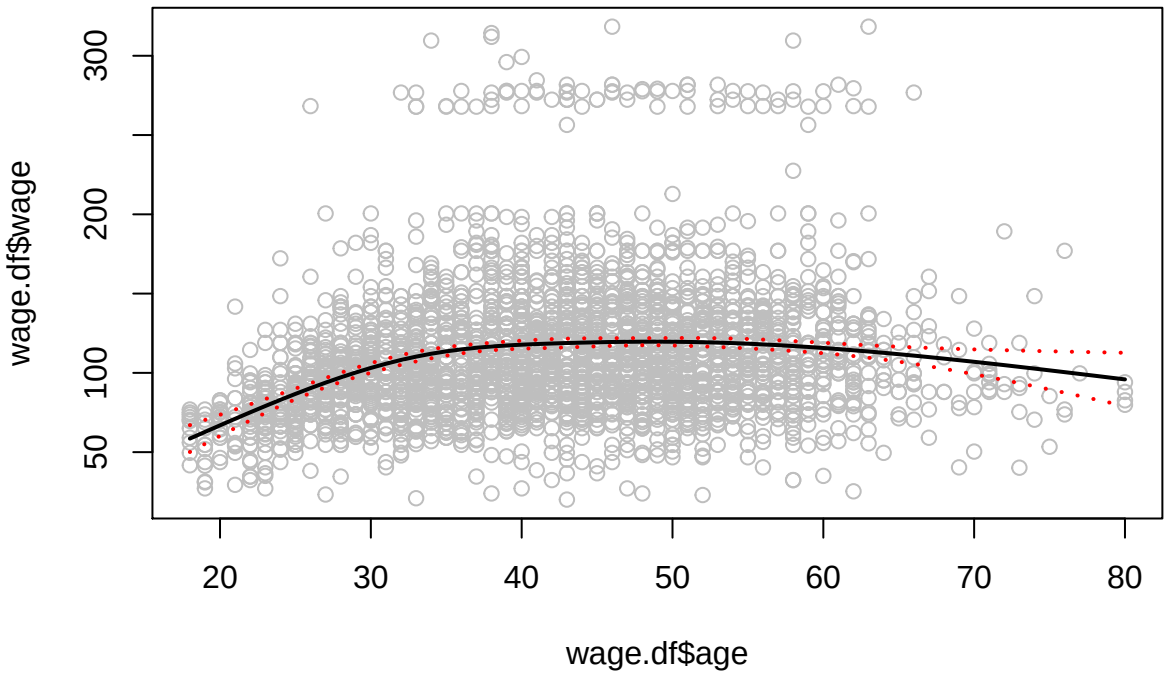

```
# use ns() to create a basis function for natural spline
dim(ns(wage.df$age, knots=c(25,40,60))) # 3 + 3 - 2
```

```
## [1] 3000    4
```

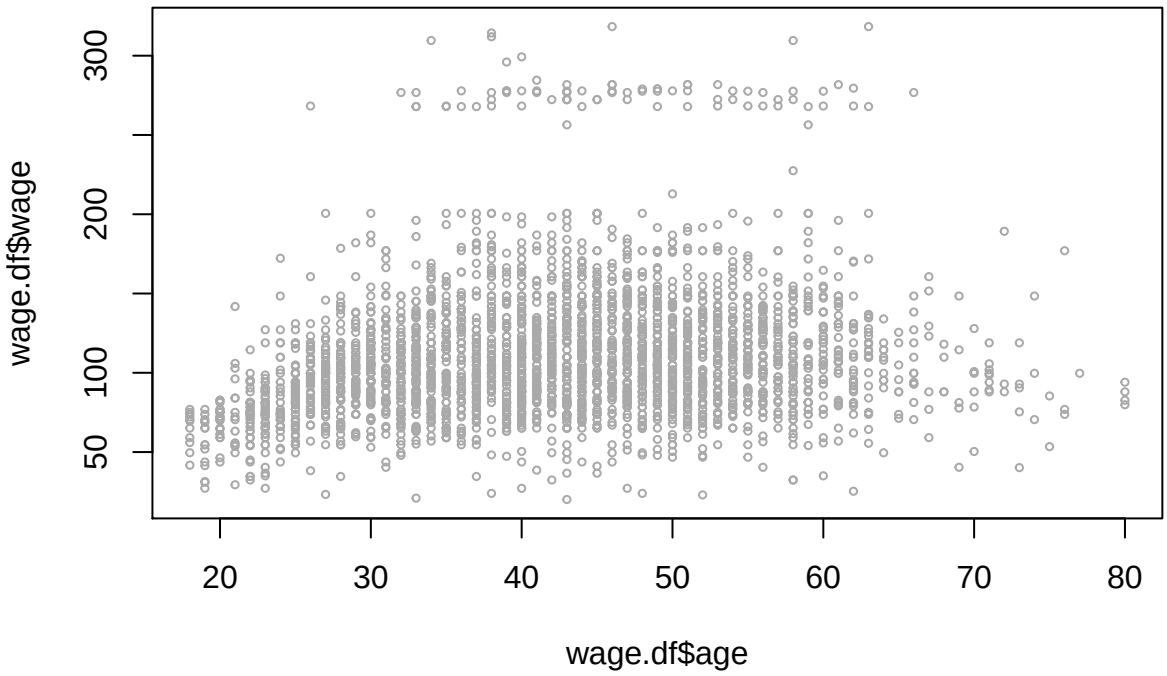
```
dim(ns(wage.df$age, df = 4))
```

```
## [1] 3000    4
```

```
# Fit wage to age using natural spline
fit2 <- lm(wage~ns(age, df = 4) ,data=wage.df)
pred2 <- predict (fit2, newdata = list(age = age.grid), se=T)
plot(wage.df$age, wage.df$wage, col="gray")
lines(age.grid, pred2$fit, col="black", lwd=2)
se_bonds <- cbind(pred2$fit+2*pred2$se, pred2$fit-2*pred2$se)
matlines(age.grid, se_bonds, lwd=2, col="red", lty=3)
```




```
# fit smoothing spline
plot(wage.df$age, wage.df$wage, xlim=age.limits, cex=0.5, col="darkgrey")
title("Smoothing spline")
# first no cv, just hard code effective degree of freedom as 16
fit.smooth=smooth.spline(wage.df$age, wage.df$wage, df=16)
pred.smooth <- predict(fit.smooth, newdata = list(age=age.grid), se=T)
title("Smoothing Spline with hard coded degree of freedom")
```



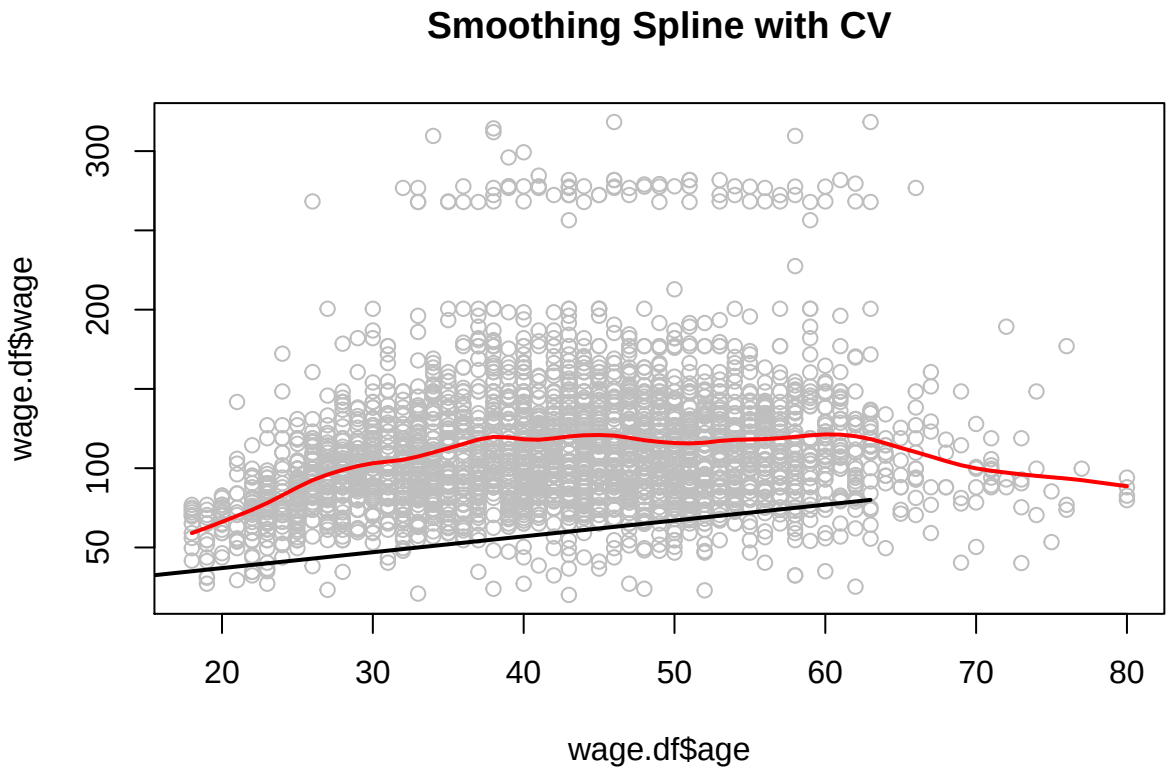
```
plot(wage.df$age, wage.df$wage, col="gray")
lines(age.grid, pred.smooth$fit, col="black", lwd=2)
lines(fit.smooth, col="red", lwd=2)

# second use cv to get the best effective degree of freedom
fit.smooth.cv=smooth.spline(wage.df$age, wage.df$wage, cv=T)
```

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

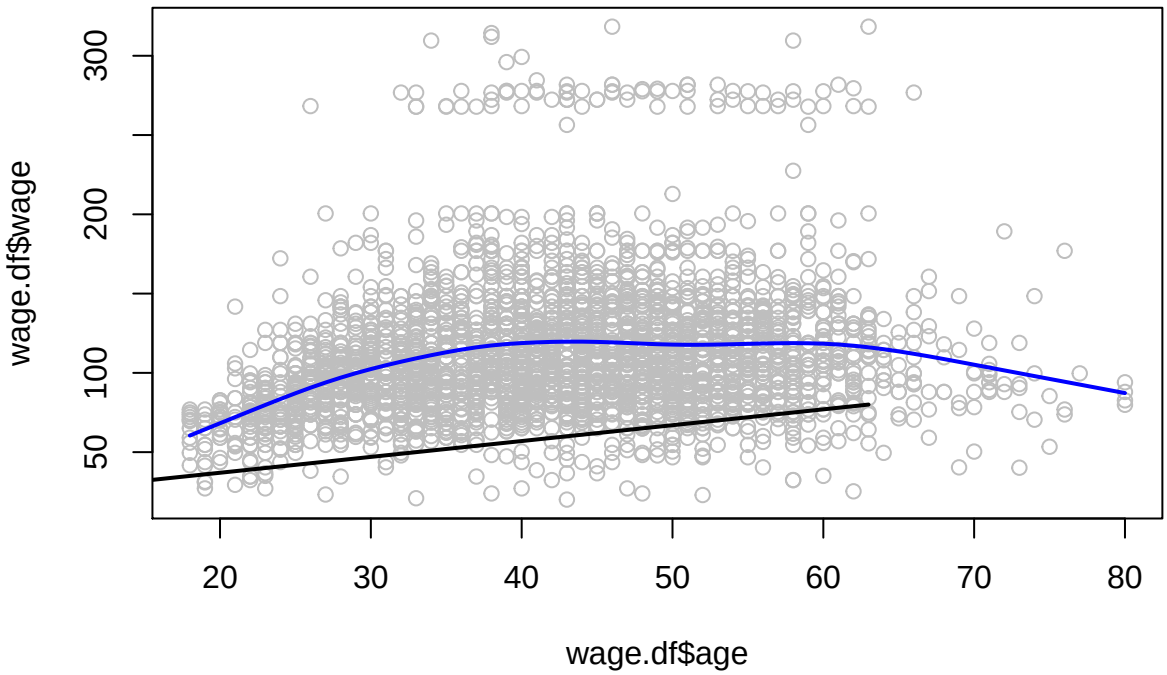
```
pred.smooth.cv <- predict(fit.smooth.cv, newdata = list(age=age.grid), se=T)

sprintf("effective degree of freedom found by cv is %s",fit.smooth.cv$df)
```

```
## [1] "effective degree of freedom found by cv is 6.79459570277247"
```

```
title("Smoothing Spline with CV")
```

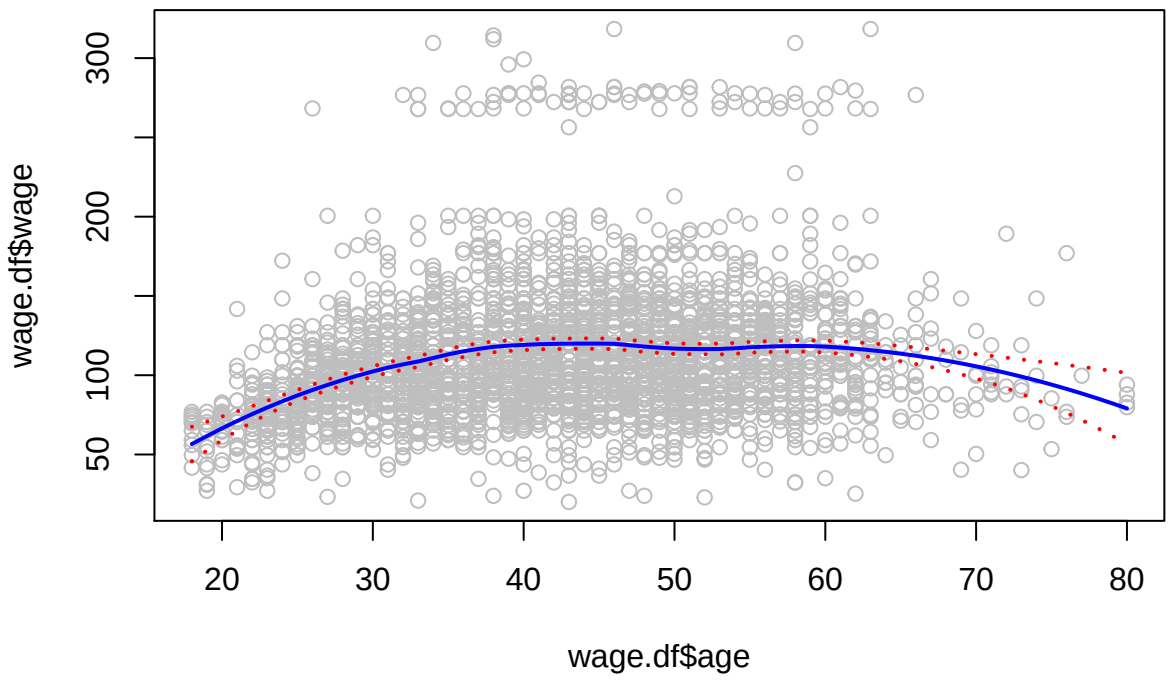



```
plot(wage.df$age, wage.df$wage, col="gray")
lines(age.grid, pred.smooth.cv$fit.cv, col="black", lwd=2)
lines(fit.smooth.cv, col="blue", lwd=2)
```




```
# Fit local regression model
plot(wage.df$age, wage.df$wage, col="gray")
title("Local regression span = 0.2")
fit.loess.1 <- loess(wage~age, span = 0.2, data = wage.df)
fit.loess.2 <- loess(wage~age, span = 0.5, data = wage.df)
pred.loess.1 <- predict(fit.loess.1, newdata = tibble(age=age.grid), se=T)
pred.loess.2 <- predict(fit.loess.2, newdata = tibble(age=age.grid), se=T)

lines(age.grid, pred.loess.1$fit, col="blue", lwd=2)
se_bonds.loess.1 <- cbind(pred.loess.1$fit+2*pred.loess.1$se, pred.loess.1$fit-2*pred.loess.1$se)
matlines(age.grid, se_bonds.loess.1, lwd=2, col="red", lty=3)
```


```
plot(wage.df$age, wage.df$wage, col="gray")
title("Local regression span = 0.4")
lines(age.grid, pred.loess.2$fit, col="blue", lwd=2)
se_bonds.loess.2 <- cbind(pred.loess.2$fit+2*pred.loess.2$se, pred.loess.2$fit-2*pred.loess.2$se)
matlines(age.grid, se_bonds.loess.2, lwd=2, col="red", lty=3)
```



```
wage.df = read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Wage.csv",
                   header=T, stringsAsFactors = F, na.strings = "?")
(wage.df.original = tibble(wage.df))
```

```
## # A tibble: 3,000 x 12
##     year   age sex   maritl race  education region jobclass health health_ins
##    <int> <int> <chr> <chr>  <chr> <chr>     <chr>  <chr>    <chr>  <chr>
##  1  2006    18 1. M~ 1. Ne~ 1. W~ 1. < HS ~ 2. Mi~ 1. Indu~ 1. <=~ 2. No
##  2  2004    24 1. M~ 1. Ne~ 1. W~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 2. No
##  3  2003    45 1. M~ 2. Ma~ 1. W~ 3. Some ~ 2. Mi~ 1. Indu~ 1. <=~ 1. Yes
##  4  2003    43 1. M~ 2. Ma~ 3. A~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
##  5  2005    50 1. M~ 4. Di~ 1. W~ 2. HS Gr~ 2. Mi~ 2. Info~ 1. <=~ 1. Yes
##  6  2008    54 1. M~ 2. Ma~ 1. W~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
##  7  2009    44 1. M~ 2. Ma~ 4. O~ 3. Some ~ 2. Mi~ 1. Indu~ 2. >=~ 1. Yes
##  8  2008    30 1. M~ 1. Ne~ 3. A~ 3. Some ~ 2. Mi~ 2. Info~ 1. <=~ 1. Yes
##  9  2006    41 1. M~ 1. Ne~ 2. B~ 3. Some ~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
## 10  2004    52 1. M~ 2. Ma~ 1. W~ 2. HS Gr~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
## # ... with 2,990 more rows, and 2 more variables: logwage <dbl>, wage <dbl>
```

```
(wage.df = tibble(wage.df))
```

```
## # A tibble: 3,000 x 12
##     year   age sex   maritl race  education region jobclass health health_ins
##    <int> <int> <chr> <chr>  <chr> <chr>     <chr>  <chr>    <chr>  <chr>
##  1  2006    18 1. M~ 1. Ne~ 1. W~ 1. < HS ~ 2. Mi~ 1. Indu~ 1. <=~ 2. No
##  2  2004    24 1. M~ 1. Ne~ 1. W~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 2. No
##  3  2003    45 1. M~ 2. Ma~ 1. W~ 3. Some ~ 2. Mi~ 1. Indu~ 1. <=~ 1. Yes
##  4  2003    43 1. M~ 2. Ma~ 3. A~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
##  5  2005    50 1. M~ 4. Di~ 1. W~ 2. HS Gr~ 2. Mi~ 2. Info~ 1. <=~ 1. Yes
##  6  2008    54 1. M~ 2. Ma~ 1. W~ 4. Colle~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
##  7  2009    44 1. M~ 2. Ma~ 4. O~ 3. Some ~ 2. Mi~ 1. Indu~ 2. >=~ 1. Yes
##  8  2008    30 1. M~ 1. Ne~ 3. A~ 3. Some ~ 2. Mi~ 2. Info~ 1. <=~ 1. Yes
##  9  2006    41 1. M~ 1. Ne~ 2. B~ 3. Some ~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
## 10  2004    52 1. M~ 2. Ma~ 1. W~ 2. HS Gr~ 2. Mi~ 2. Info~ 2. >=~ 1. Yes
## # ... with 2,990 more rows, and 2 more variables: logwage <dbl>, wage <dbl>
```

```
# first clean up NA and particularly character columns
# 1) remove leading, trainling and empty characters from character columns

trim.f <- function(x) trimws(x, which = c("both")) # leading and trailing spaces
empty.f <- function(x) gsub("^$", NA, x) # empty strings

wage.df[sapply(wage.df, is.character)] <-
  wage.df %>%
    select(which(sapply(.,is.character))) %>%
    mutate_each(funs(trim.f)) %>%
    mutate_each(funs(empty.f))  %>%
    na.omit


sprintf("Any NA or empty string in character columns recognized: %s",
        !identical(wage.df,wage.df.original) )
```

```
## [1] "Any NA or empty string in character columns recognized: FALSE"
```

```r
# 2) It is safe now to convert all character fields into factors
wage.df[sapply(wage.df, is.character)] <-
  wage.df %>%
  select(which(sapply(., is.character))) %>%
  mutate_each(funs(factor))

print("---------------- GAM with smoothing spline and CV :---------------- ")
```

```
## [1] "---------------- GAM with smoothing spline and CV :---------------- "
```

```r
library(mgcv)
```

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

```
##
## Attaching package: 'nlme'
```

```
## The following object is masked from 'package:dplyr':
##
##     collapse
```

```
## This is mgcv 1.8-31. For overview type 'help("mgcv-package")'.
```

```r
cv.gam <- gam(logwage~s(age, bs="cr") + s(year, bs="cr", k=5) + education,
              method="GACV.Cp", scale=-1, data=wage.df,family=Gamma(link=log),
              gamma=1.4)
```

```
## Warning in newton(lsp = lsp, X = G$X, y = G$y, Eb = G$Eb, UrS = G$UrS, L =
## G$L, : Fitting terminated with step failure - check results carefully
```

```r
print("----------------- cv.gam model ----------------------")
```

```
## [1] "----------------- cv.gam model ----------------------"
```

```r
print(cv.gam)
```

```
##
## Family: Gamma
## Link function: log
##
## Formula:
## logwage ~ s(age, bs = "cr") + s(year, bs = "cr", k = 5) + education
##
## Estimated degrees of freedom:
## 5.95 1.99  total = 12.93
##
## GACV score: 0.004089754
```

```r
print("------ anova for approximate significance of terms of cv.gam ---------")
```

```
## [1] "------ anova for approximate significance of terms of cv.gam ---------"
```

```r
anova(cv.gam)
```

```
##
## Family: Gamma
## Link function: log
##
## Formula:
## logwage ~ s(age, bs = "cr") + s(year, bs = "cr", k = 5) + education
```

```
##
## Parametric Terms:
##           df     F p-value
## education  4 199.3  <2e-16
##
## Approximate significance of smooth terms:
##           edf Ref.df     F  p-value
## s(age)  5.946  7.075 51.56  < 2e-16
## s(year) 1.988  2.430 10.09 1.78e-05
```

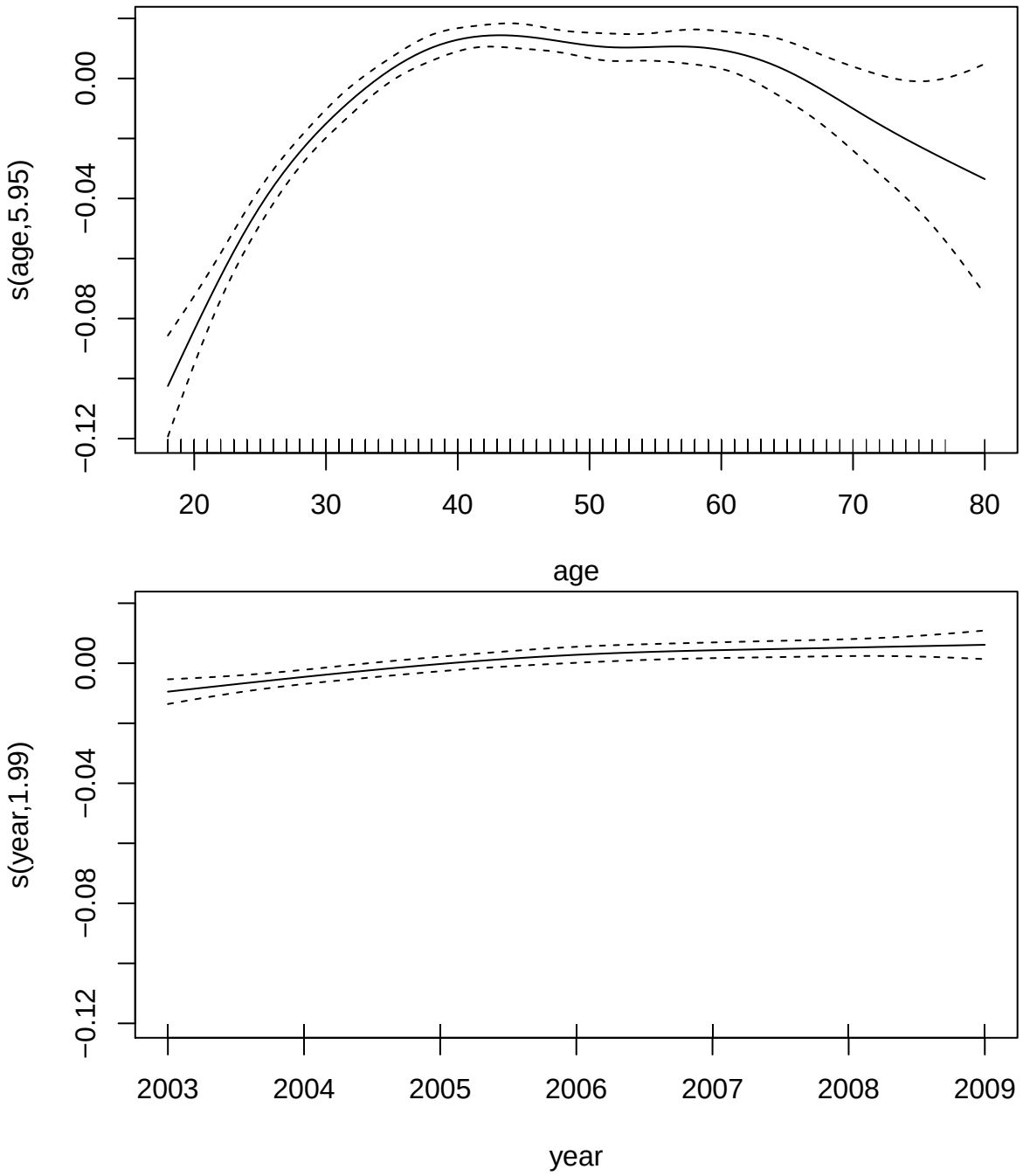
```
print("----------------- summary cv.gam1 ----------------------")
```

```
## [1] "----------------- summary cv.gam1 ----------------------"
```

```
summary(cv.gam)
```

```
##
## Family: Gamma
## Link function: log
##
## Formula:
## logwage ~ s(age, bs = "cr") + s(year, bs = "cr", k = 5) + education
##
## Parametric coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 1.484528   0.003838 386.800  < 2e-16 ***
## education2. HS Grad         0.025678   0.004329   5.932 3.34e-09 ***
## education3. Some College    0.051248   0.004562  11.234  < 2e-16 ***
## education4. College Grad    0.075793   0.004534  16.717  < 2e-16 ***
## education5. Advanced Degree 0.109607   0.004918  22.286  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##           edf Ref.df     F  p-value
## s(age)  5.946  7.075 51.56  < 2e-16 ***
## s(year) 1.988  2.430 10.09 1.78e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.309   Deviance explained = 30.7%
## GACV = 0.0040898  Scale est. = 0.0039239  n = 3000
```

```
plot(cv.gam, too.far=0.15)
```




```
#residuals(cv.gam)

#print("----------------- check ----------------------")
#gam.check(cv.gam)

# plot (cv.gam, residuals=T, select=1)
# title("age vs s(age)")
#
# plot (cv.gam, residuals=T, select=2)
# title("year vs s(year)")
```

```r
cv.gam1 <- gam(logwage~s(age, bs="cr") + s(year, bs="cr", k=5) +te(age, year, k=5) + education,
               method="GACV.Cp", scale=-1, data=wage.df,family=Gamma(link=log),
               gamma=1.4)

print("----------------- cv.gam1 model ----------------------")
```

```
## [1] "----------------- cv.gam1 model ----------------------"
```

```r
print(cv.gam1)
```

```
##
## Family: Gamma
## Link function: log
##
## Formula:
## logwage ~ s(age, bs = "cr") + s(year, bs = "cr", k = 5) + te(age,
##     year, k = 5) + education
##
## Estimated degrees of freedom:
## 5.9719 1.9611 0.0008  total = 12.93
##
## GACV score: 0.004090171
```

```r
print("------ anova for approximate signifiocance of terms of cv.gam1 ---------")
```

```
## [1] "------ anova for approximate signifiocance of terms of cv.gam1 ---------"
```

```r
anova(cv.gam1)
```

```
##
## Family: Gamma
## Link function: log
##
## Formula:
## logwage ~ s(age, bs = "cr") + s(year, bs = "cr", k = 5) + te(age,
##     year, k = 5) + education
##
## Parametric Terms:
##           df     F p-value
## education  4 199.2  <2e-16
##
## Approximate significance of smooth terms:
##                   edf    Ref.df     F  p-value
## s(age)        5.971896  7.100560 51.39  < 2e-16
## s(year)       1.961141  2.398733 10.18 1.74e-05
## te(age,year)  0.000787 19.000000  0.00    0.631
```

```r
print("----------------- summary cv.gam1 ----------------------")
```

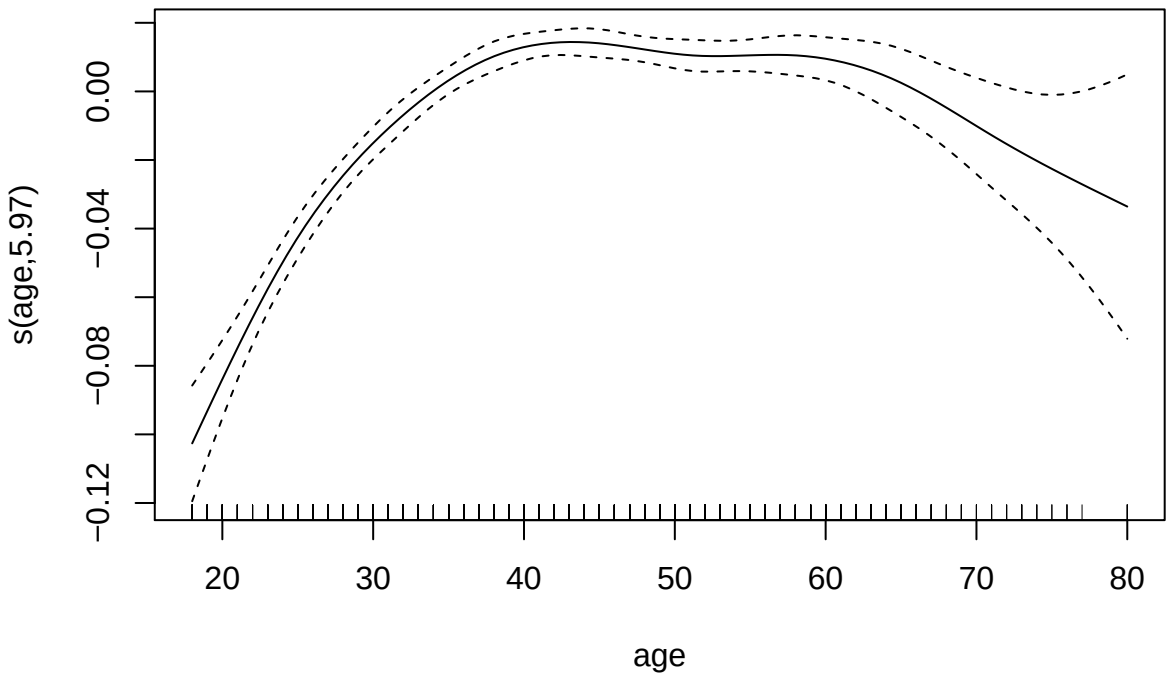
```
## [1] "----------------- summary cv.gam1 ----------------------"
```

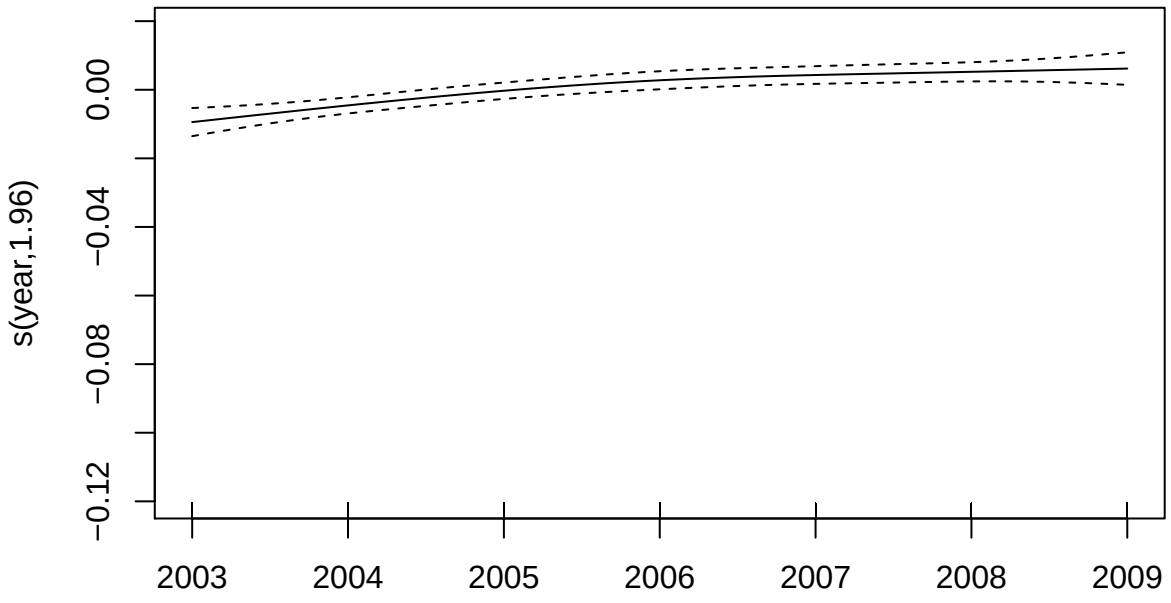
```r
summary(cv.gam1)
```

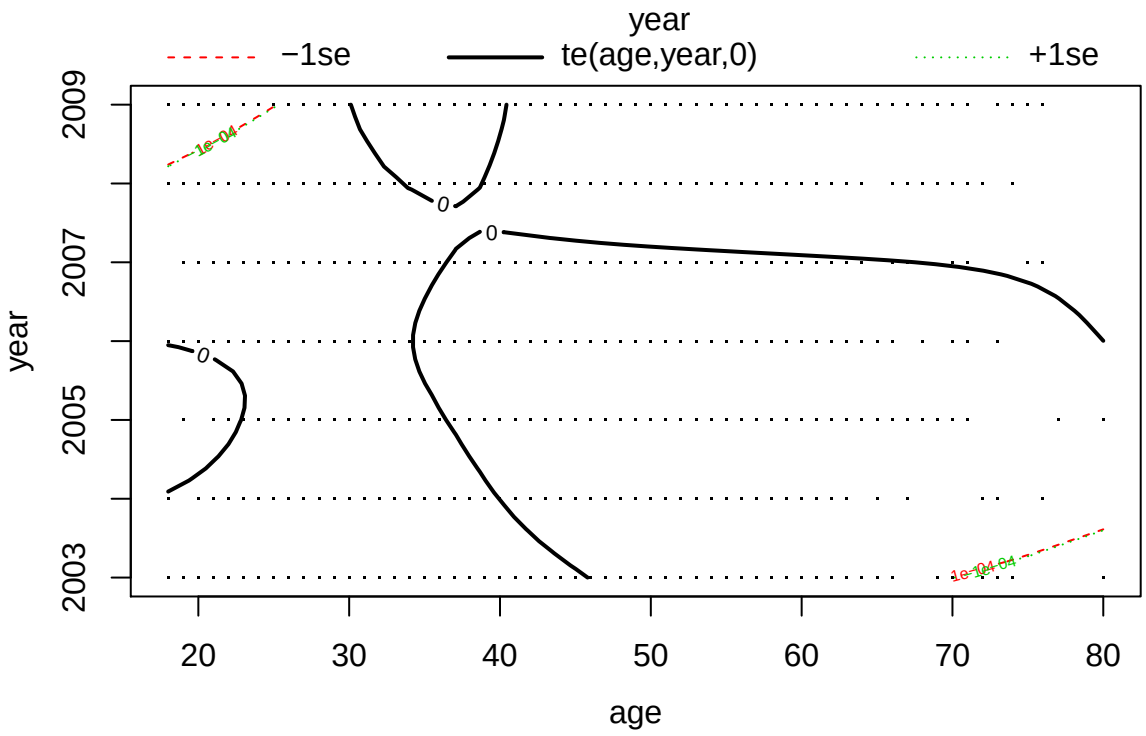
```
##
## Family: Gamma
## Link function: log
##
```

```
## Formula:
## logwage ~ s(age, bs = "cr") + s(year, bs = "cr", k = 5) + te(age,
##     year, k = 5) + education
##
## Parametric coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 1.484528   0.003838 386.800  < 2e-16 ***
## education2. HS Grad         0.025680   0.004329   5.932 3.33e-09 ***
## education3. Some College    0.051250   0.004562  11.234  < 2e-16 ***
## education4. College Grad    0.075790   0.004534  16.716  < 2e-16 ***
## education5. Advanced Degree 0.109605   0.004918  22.285  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                   edf Ref.df     F  p-value
## s(age)       5.971896  7.101 51.39  < 2e-16 ***
## s(year)      1.961141  2.399 10.18 1.74e-05 ***
## te(age,year) 0.000787 19.000  0.00    0.631
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.309   Deviance explained = 30.7%
## GACV = 0.0040902  Scale est. = 0.0039239  n = 3000
```

```
plot(cv.gam1, too.far=0.15)
```








```
# predict
pred <- predict(cv.gam, newdata = wage.df, se=TRUE,type="terms")
```

```
library(tidyverse)
library(mgcv)
library(purrr)

smarket.df =
  read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/Smarket.csv",
           header=T, stringsAsFactors = F, na.strings = "?")
```

```
smarket.df = as_tibble(smarket.df)
smarket.df.original = as_tibble(smarket.df)

# first clean up NA and particularly character columns
# 1) remove leading, trainling and empty characters from character columns

trim.f <- function(x) trimws(x, which = c("both")) # leading and trailing spaces
empty.f <- function(x) gsub("^$", NA, x) # empty strings

smarket.df[sapply(smarket.df, is.character)] <-
  smarket.df %>%
    select(which(sapply(.,is.character))) %>%
    mutate_each(funs(trim.f)) %>%
    mutate_each(funs(empty.f))  %>%
    na.omit


sprintf("Any NA or empty string in character columns recognized: %s",
        !identical(smarket.df,smarket.df.original) )
```

```
## [1] "Any NA or empty string in character columns recognized: FALSE"
```

```
# 2) It is safe now to convert all character fields into factors
smarket.df[sapply(smarket.df, is.character)] <-
  smarket.df %>%
  select(which(sapply(., is.character))) %>%
  mutate_each(funs(factor))

# split to test train
train <- smarket.df$Year < 2005
test <- !train


# Use Gam
# it returns logit{E(yi)}
cv.gam1 <- gam(Direction~ s(Year, bs="cr", k=4)+
                 s(Volume, k=20)+s(Lag1, k=20),
               method="GACV.Cp", scale=-1, data=smarket.df,subset = train,
               family=binomial(link=logit))
str(smarket.df)
```

```
## tibble [1,250 x 10] (S3: tbl_df/tbl/data.frame)
##  $ X        : int [1:1250] 1 2 3 4 5 6 7 8 9 10 ...
##  $ Year     : int [1:1250] 2001 2001 2001 2001 2001 2001 2001 2001 2001 2001 ...
##  $ Lag1     : num [1:1250] 0.381 0.959 1.032 -0.623 0.614 ...
##  $ Lag2     : num [1:1250] -0.192 0.381 0.959 1.032 -0.623 ...
##  $ Lag3     : num [1:1250] -2.624 -0.192 0.381 0.959 1.032 ...
##  $ Lag4     : num [1:1250] -1.055 -2.624 -0.192 0.381 0.959 ...
##  $ Lag5     : num [1:1250] 5.01 -1.055 -2.624 -0.192 0.381 ...
##  $ Volume   : num [1:1250] 1.19 1.3 1.41 1.28 1.21 ...
##  $ Today    : num [1:1250] 0.959 1.032 -0.623 0.614 0.213 ...
##  $ Direction: Factor w/ 2 levels "Down","Up": 2 2 1 2 2 2 1 2 2 2 ...
```

```r
print("----------------- cv.gam model ----------------------")
```

```
## [1] "----------------- cv.gam model ----------------------"
```

```r
print(cv.gam1)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## Direction ~ s(Year, bs = "cr", k = 4) + s(Volume, k = 20) + s(Lag1,
##     k = 20)
##
## Estimated degrees of freedom:
## 2.51 6.53 2.41  total = 12.45
##
## GACV score: 1.386241
```

```r
print("------ anova for approximate signifiocance of terms of cv.gam ---------")
```

```
## [1] "------ anova for approximate signifiocance of terms of cv.gam ---------"
```

```r
anova(cv.gam1)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## Direction ~ s(Year, bs = "cr", k = 4) + s(Volume, k = 20) + s(Lag1,
##     k = 20)
##
## Approximate significance of smooth terms:
##             edf Ref.df     F p-value
## s(Year)   2.512  2.825 3.075  0.0441
## s(Volume) 6.525  8.171 0.961  0.4695
## s(Lag1)   2.410  3.146 1.248  0.2755
```

```r
print("----------------- summary cv.gam1 ----------------------")
```

```
## [1] "----------------- summary cv.gam1 ----------------------"
```

```r
summary(cv.gam1)
```

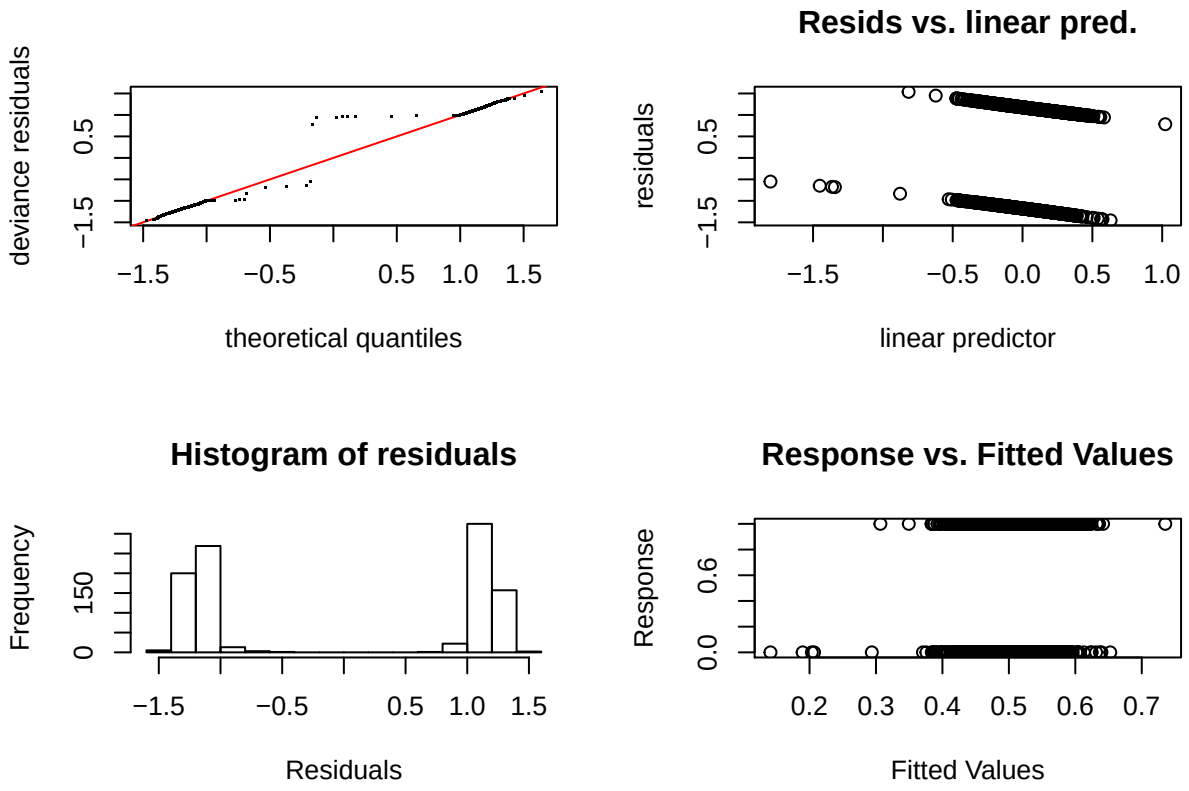
```
##
## Family: binomial
## Link function: logit
##
## Formula:
## Direction ~ s(Year, bs = "cr", k = 4) + s(Volume, k = 20) + s(Lag1,
##     k = 20)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.03151    0.06422   0.491    0.624
##
```

```
## Approximate significance of smooth terms:
##             edf Ref.df     F p-value
## s(Year)   2.512  2.825 3.075  0.0441 *
## s(Volume) 6.525  8.171 0.961  0.4695
## s(Lag1)   2.410  3.146 1.248  0.2755
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.0128   Deviance explained = 1.79%
## GACV = 1.3862  Scale est. = 1.0126    n = 998
```

```
print("------------------ check cv.gam1 ----------------------")
```

```
## [1] "------------------ check cv.gam1 ----------------------"
```

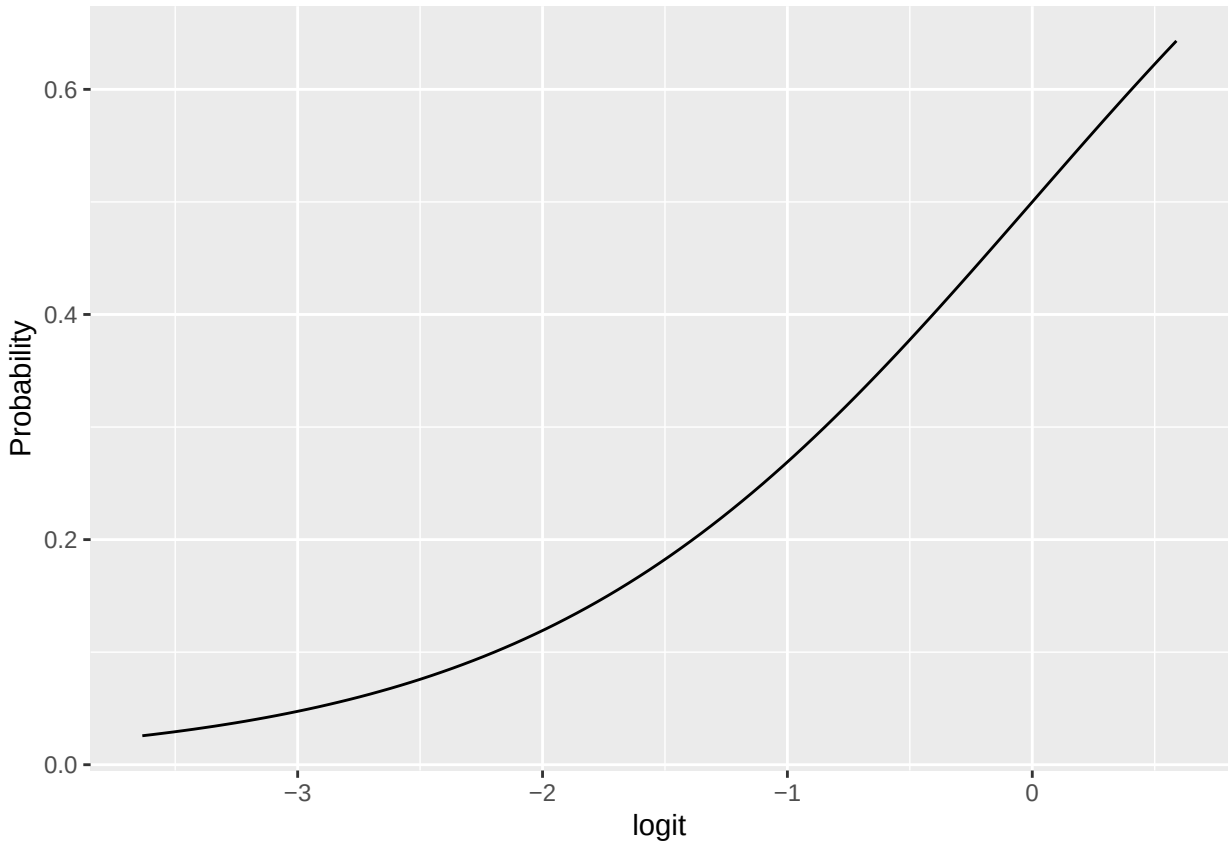
```
gam.check(cv.gam1)
```




```
##
## Method: GACV   Optimizer: outer newton
## full convergence after 6 iterations.
## Gradient range [-7.482598e-09,2.287114e-08]
## (score 1.386241 & scale 1.01263).
## Hessian positive definite, eigenvalue range [0.0002820594,0.0008857096].
## Model rank =  42 / 42
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##              k'   edf k-index p-value
## s(Year)    3.00  2.51    1.06    0.94
```

```
## s(Volume) 19.00  6.52    1.01    0.54
## s(Lag1)   19.00  2.41    1.02    0.70
```

```
# plot(cv.gam1, too.far=0.15)


# predict the test model
pred <- predict(cv.gam1, newdata = smarket.df[test,], se=TRUE)
preds <- pred$fit
# Note pred$fit is logit, we need to convert it to probability

logit.inverse <- function(logit) {
  odds <- exp(logit)
  odds/(1+odds)
}

gam.probs <- map(preds, logit.inverse)

# let's draw probabilities vs logit to see if it makes sense
library(ggplot2)
ggplot(tibble(x=preds), aes(x = x)) +
        stat_function(fun = logit.inverse) +
  scale_x_continuous(name = "logit") +
        scale_y_continuous(name = "Probability")
```




```
# calculate test.mse
threshold = 0.5
gam.pred <- ifelse(gam.probs > threshold, "Up", "Down")
```

```
missclassificationRate = NULL
nullClassificationRate = NULL
FP_rates = NULL
TP_rates = NULL
precisions = NULL
specificities = NULL
confusionTables = NULL
aucs = NULL

library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

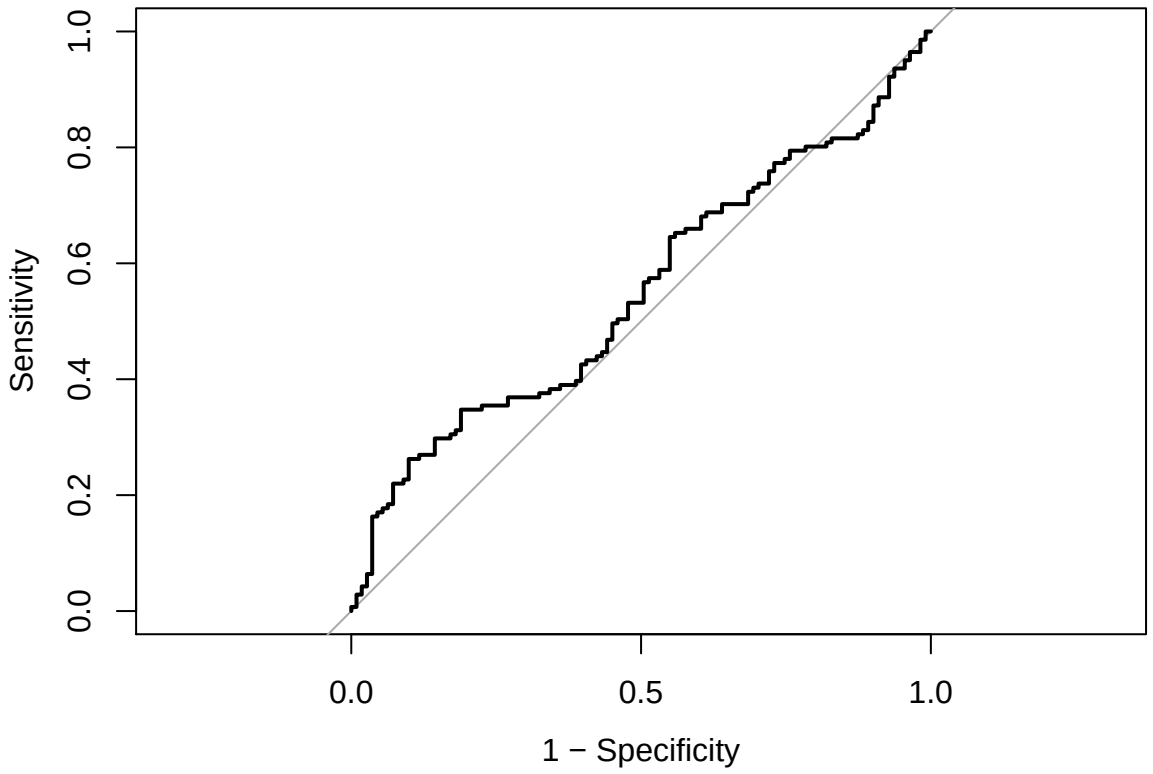
```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
df <- smarket.df[test,]

confusion_table <- table(gam.pred, smarket.df[test,]$Direction)

nullClassifier <- max(
    (confusion_table[1,1] + confusion_table[2,1])/(confusion_table[1,1] + confusion_table[2,1]+ confusi
    (confusion_table[1,2] + confusion_table[2,2])/(confusion_table[1,1] + confusion_table[2,1]+ confusi

nullClassificationRate <- c(nullClassificationRate, nullClassifier)
roc_obj <- roc(df$Direction, purrr::flatten_dbl(gam.probs))
```

```
## Setting levels: control = Down, case = Up
```

```
## Setting direction: controls < cases
```

```
# Let's draw some AUC plots
plot(roc_obj, legacy.axes = TRUE)
```




```
aucs <- c(aucs, auc(roc_obj))

missclassificationRate <- c(missclassificationRate, mean(gam.pred != smarket.df[test,]$Direction))
FP_rates <- c(FP_rates, confusion_table[2,1]/(confusion_table[2,1]+ confusion_table[1,1]))
TP_rates <- c(TP_rates, confusion_table[2,2]/(confusion_table[2,2]+ confusion_table[1,2]))
precisions <- c(precisions, confusion_table[2,2] / (confusion_table[2,2] + confusion_table[2,1]))
specificities <- c(specificities , 1 - confusion_table[2,1]/(confusion_table[2,1]+ confusion_table[1,1]))

# overall fraction of wrong predictions:
# print(confusion_table)

# average missclassification error rate
sprintf("GAM classifier : Missclassification error rate : %s", mean(missclassificationRate))
```

```
## [1] "GAM classifier : Missclassification error rate : 0.464285714285714"
```

```
sprintf("GAM classifier : Null Classifier: %s", mean(nullClassificationRate))
```

```
## [1] "GAM classifier : Null Classifier: 0.55952380952381"
```

```
sprintf("GAM classifier AUC: %s", mean (aucs))
```

```
## [1] "GAM classifier AUC: 0.548846719059485"
```

```
# FP rate:
sprintf("GAM classifier : FP rate (TypeI error, 1 - specificity) : %s", mean(FP_rates))
```

```
## [1] "GAM classifier : FP rate (TypeI error, 1 - specificity) : 0.720720720720721"
```

```
# TP rate:
sprintf("GAM classifier : TP rate (1-TypeII error, power, sensetivity, recall) : %s", mean(TP_rates))
```

```
## [1] "GAM classifier : TP rate (1-TypeII error, power, sensetivity, recall) : 0.737588652482269"
```

```
# precision:
sprintf("GAM classifier : precision: %s", mean(precisions))
```

```
## [1] "GAM classifier : precision: 0.565217391304348"
```

```
# specificity 1-FP/N:
sprintf("GAM classifier : specificity 1-FP/N: %s", mean(specificities))
```

```
## [1] "GAM classifier : specificity 1-FP/N: 0.279279279279279"
```

```
library(tidyverse)
library(mgcv)
library(purrr)

locv1 <- function(x1, y1, nd, span, ntrial)
{
  locvgcv <- function(sp, x1, y1)
  {
    nd <- length(x1)

    assign("data1", data.frame(xx1 = x1, yy1 = y1))
    fit.lo <- loess(yy1 ~ xx1, data = data1, span = sp, family = "gaussian", degree = 2, surface = "dir
    res <- residuals(fit.lo)

    dhat2 <- function(x1, sp)
    {
        nd2 <- length(x1)
        diag1 <- diag(nd2)
        dhat <- rep(0, length = nd2)

        for(jj in 1:nd2){
            y2 <- diag1[, jj]
            assign("data1", data.frame(xx1 = x1, yy1 = y2))
            fit.lo <- loess(yy1 ~ xx1, data = data1, span = sp, family = "gaussian", degree = 2, surfac
            ey <- fitted.values(fit.lo)
            dhat[jj] <- ey[jj]
            }
            return(dhat)
        }

        dhat <- dhat2(x1, sp)
        trhat <- sum(dhat)
        sse <- sum(res^2)

        cv <- sum((res/(1 - dhat))^2)/nd
        gcv <- sse/(nd * (1 - (trhat/nd))^2)

        return(gcv)
    }

    gcv <- lapply(as.list(span1), locvgcv, x1 = x1, y1 = y1)
    #cvgcv <- unlist(cvgcv)
    #cv <- cvgcv[attr(cvgcv, "names") == "cv"]
    #gcv <- cvgcv[attr(cvgcv, "names") == "gcv"]
```

```
    return(gcv)
}
```

```
library(tidyverse)
library(mgcv)


wage.df = read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Wage.csv", header=T, stri
wage.df.original = tibble(wage.df)
wage.df = tibble(wage.df)

# remove empty characters and NA helper
remove.empty.characters <- function(df)
  df %>%
    select_all %>%
    filter_if(is.character, any_vars(!is.na(.) & trimws(.) != ""))

# Next remove leading and trailing spaces from all elements in character columns
trim.f <- function(col, na.rm = F) {
  isNA <- !reduce(col, ~ (is.na(.x) & is.na(.y)))
  if (na.rm && isNA)
    unlist(map(col, ~ (if (is.na(.x)) "" else .x) ),use.names = F)
  else trimws(col, which = c("both")) # leading and trailing spaces
}

trim.spaces <- function(df)
  df %>%
    mutate_if(is.character, trim.f, na.rm = T)

# Finally convert character columns to factor

char.to.fctor <- function(df)
  df %>%
    mutate_if(is.character, ~ factor(.x, levels = (.x %>% table() %>% names())))


wage.df <-
  wage.df %>%
  na.omit() %>%
  trim.spaces() %>%
  remove.empty.characters() %>%
  char.to.fctor()


# Polynomial regression to predict wage using age and find the degree with cv
set.seed(1)
k <- 10
degrees <- 1:5

# create k folds
folds <- sample(1:k, nrow(wage.df), replace = T)

cv.result <- tibble(degree=NULL, cv.mse = NULL)
```

```
for (degree in degrees){
  test.mses <- double(length(folds))
  for (fold in folds){
    fit.poly <- lm(wage ~ poly(age, degree), data = wage.df[folds != fold, ])
    predicts <- predict(fit.poly,
                        newdata = list(age=wage.df[folds == fold, ]$age), se=T)
    testMSE <- mean((predicts$fit - wage.df[folds == fold, ]$wage)^2)
    test.mses <- c(test.mses, testMSE)
  }
  cv.result<- rbind(cv.result, tibble(degree = degree, cv.mse=mean(test.mses)))

}
cv.result
```

```
## # A tibble: 5 x 2
##   degree cv.mse
##    <int>  <dbl>
## 1      1   838.
## 2      2   800.
## 3      3   797.
## 4      4   797.
## 5      5   797.
```

```
# Among ploynimials of degrees 1 to 5 , CV shows ploynomial with degree
# 4 has smallest MSE.
# ----------- Expect to get the same result when using ANOVA ------------------
# perform an analysis of variance (ANOVA, using an F-test) in order to test the
# null hypothesis that a model M1 is sufficient to explain the data against the
# alternative hypothesis that a more complex model M2 is required.
# M1 and M2 must be nested models. Model M1 ploynomial with smaller degree is nested
# in model M2 which has polynomial with larger degree.

# Note that ANOVA is all  based on training data only
nested.models <- 1:5 %>% map(~lm(wage ~ poly(age, .), data = wage.df))
do.call("anova", nested.models)
```

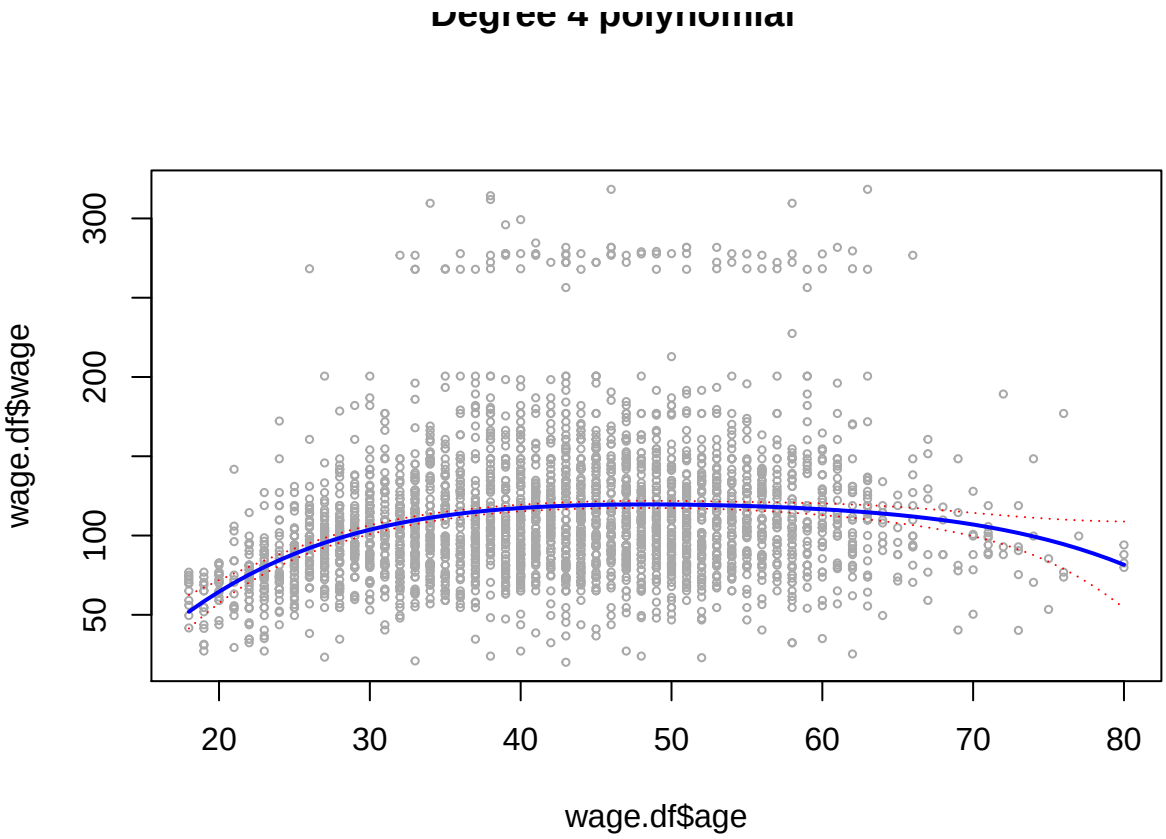
```
## Analysis of Variance Table
##
## Model 1: wage ~ poly(age, .)
## Model 2: wage ~ poly(age, .)
## Model 3: wage ~ poly(age, .)
## Model 4: wage ~ poly(age, .)
## Model 5: wage ~ poly(age, .)
##   Res.Df     RSS Df Sum of Sq        F    Pr(>F)
## 1   2998 5022216
## 2   2997 4793430  1    228786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

```
# ploynomial with degree 1 comparing with ploynomial with degree 2
# has almost zero p-value (2.2e-16) showing model is not enough.
# similarly ploynimial degree 2 comparing with degree 3 shows smaller
```

```r
# p-value. Comparing Plynomial degree 3 to degree 4 show still smaller p-value
# but ploynomial with degree 4 has the best p-value(0.051046)
# Ploynomial with degree 5 is unnecessarily too complex (p-va;ue > 0.05).
# So models with plynomial degree 3 or 4 are best fit, which is align
# with whqt we got from CV.

# --------------  Use orthogonal polynomial to find p-values -----------------

# We can use the fact that poly() creates orthogonal polynomials.'
# so we can obtain p-values for each model using summary() function:
# note that square of t-statistics here is equal to -statistics from anova()

coef(summary(nested.models[[5]]))
```

```
##                 Estimate Std. Error     t value     Pr(>|t|)
## (Intercept)    111.70361  0.7287647 153.2780243 0.000000e+00
## poly(age, .)1  447.06785 39.9160847  11.2001930 1.491111e-28
## poly(age, .)2 -478.31581 39.9160847 -11.9830341 2.367734e-32
## poly(age, .)3  125.52169 39.9160847   3.1446392 1.679213e-03
## poly(age, .)4  -77.91118 39.9160847  -1.9518743 5.104623e-02
## poly(age, .)5  -35.81289 39.9160847  -0.8972045 3.696820e-01
```

```r
# ANOVA method works whether or not we used orthogonal polynomials
# particularly when we have other terms in the model


# lets draw the prediction
age.range <- range(wage.df$age)
age.grid <- seq (age.range[[1]], age.range[[2]])
preds <- predict(lm(wage ~ poly(age, 4), data = wage.df), newdata = tibble(age = age.grid), se=T)
se.bands <- cbind(preds$fit + 2 * preds$se.fit , preds$fit - 2 * preds$se.fit )
plot(wage.df$age, wage.df$wage, xlim = age.range, cex=0.5, col="darkgray")
title("Degree 4 polynomial ", outer = T)
lines(age.grid, preds$fit, lwd=2, col="blue")
matlines(age.grid, se.bands, lwd=1, col="red", lty=3)
```




```
library(tidyverse)
library(mgcv)


wage.df = read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Wage.csv", header=T, stri
wage.df.original = tibble(wage.df)
wage.df = tibble(wage.df)

# remove empty characters and NA helper
remove.empty.characters <- function(df)
  df %>%
    select_all %>%
    filter_if(is.character, any_vars(!is.na(.) & trimws(.) != ""))

# Next remove leading and trailing spaces from all elements in character columns
trim.f <- function(col, na.rm = F) {
  isNA <- !reduce(col, ~ (is.na(.x) & is.na(.y)))
  if (na.rm && isNA)
    unlist(map(col, ~ (if (is.na(.x)) "" else .x) ),use.names = F)
  else trimws(col, which = c("both")) # leading and trailing spaces
}

trim.spaces <- function(df)
  df %>%
    mutate_if(is.character, trim.f, na.rm = T)

# Finally convert character columns to factor
```

```r
char.to.fctor <- function(df)
  df %>%
    mutate_if(is.character, ~ factor(.x, levels = (.x %>% table() %>% names())))


wage.df <-
  wage.df %>%
  na.omit() %>%
  trim.spaces() %>%
  remove.empty.characters() %>%
  char.to.fctor()

# Fit a step function to predict wage using age
# use cv to find optimal number of cuts

# we use cut function to unformly cut the quantiles
table(cut(wage.df$age, 4))
```

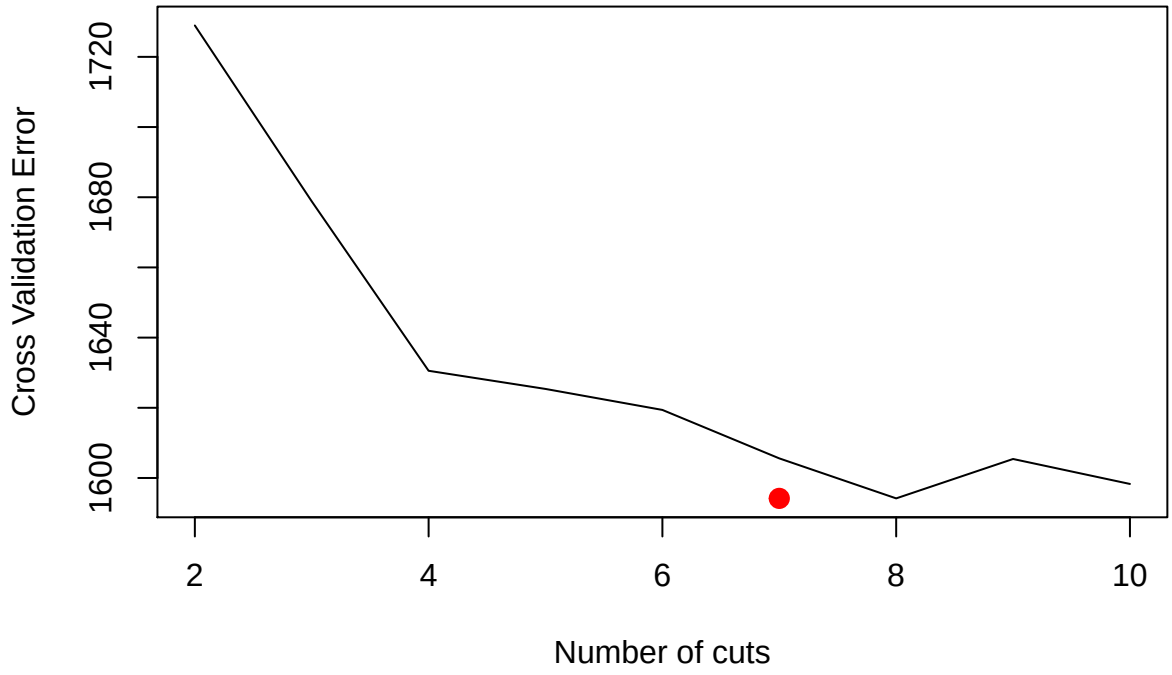
```
##
## (17.9,33.5]   (33.5,49]   (49,64.5] (64.5,80.1]
##         750        1399         779          72
```

```r
# use cv to find number of cuts
set.seed(12)
cuts <- 2:10
k = 10
row.indices <- sample(1:k, nrow(wage.df), replace = T)

cv.result <- tibble(cuts = NULL, cv.mse = NULL)
for (k in cuts){
  # cut needs to have its own column in dataframe

  wage.df$age_cut <- cut(wage.df$age, k)
  cv.mses <- double(length(cuts))
  for (row.index.test in row.indices){
    #  If we do below line we get : "factor cut(age, k) has new levels "
    # problem is cut(age, k) existed only as an inline creation within your lm()
    #  step.model <- lm(wage~cut(age, k), data=wage.df[row.indices != row.index.test, ])
    step.model <- lm(wage~age_cut, data=wage.df[row.indices != row.index.test, ])
    preds <- predict(step.model, newdata=wage.df[row.indices == row.index.test, ], se=T)
    cv.mses = c(cv.mses, mean((preds$fit - wage.df[row.indices == row.index.test, ]$wage)^2))
  }
  cv.result <- rbind(cv.result , tibble(cuts = k, cv.mse = mean(cv.mses)))
}
cv.result
```

```
## # A tibble: 9 x 2
##    cuts cv.mse
##   <int>  <dbl>
## 1     2  1729.
## 2     3  1679.
## 3     4  1631.
## 4     5  1625.
## 5     6  1619.
```

```
## 6     7  1606.
## 7     8  1594.
## 8     9  1605.
## 9    10  1598.
```

```r
plot(cv.result,xlab='Number of cuts',ylab='Cross Validation Error',type='l')
x=which.min(cv.result$cv.mse)
points(x,cv.result[x, ]$cv.mse,pch=20,cex=2,col='red')
```




```r
# cv shows 8 cuts is the best, lets make a model with 8 cuts:
wage.df$age_cut <- cut(wage.df$age, 8)
step.model <- lm(wage~age_cut, data=wage.df[row.indices != row.index.test, ])
preds <- predict(step.model, newdata=wage.df[row.indices == row.index.test, ], se=T)

# lets draw the prediction
age.range <- range(wage.df$age)
age.grid <- seq (age.range[[1]], age.range[[2]])
age.grid.cut = cut(age.grid, 8)
preds <- predict(step.model, newdata = tibble(age_cut = age.grid.cut), se=T)
se.bands <- cbind(preds$fit + 2 * preds$se.fit , preds$fit - 2 * preds$se.fit )
plot(wage.df$age, wage.df$wage, xlim = age.range, cex=0.5, col="darkgray")
title("step function with 8 cuts ", outer = T)
lines(age.grid, preds$fit, lwd=2, col="blue")
matlines(age.grid, se.bands, lwd=1, col="red", lty=3)
```


```r
# Explore the relationships between some of these other predictors and wage,
# and use non-linear fitting techniques in order to fit flexible models to the data.
# Create plots of the results obtained, and write a summary of your findings.

library(boot)
library(tidyverse)
library(mgcv)


wage.df = read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Wage.csv", header=T, stri
wage.df.original = tibble(wage.df)
wage.df = tibble(wage.df)

# remove empty characters and NA helper
remove.empty.characters <- function(df)
  df %>%
    select_all %>%
    filter_if(is.character, any_vars(!is.na(.) & trimws(.) != ""))

# Next remove leading and trailing spaces from all elements in character columns
trim.f <- function(col, na.rm = F) {
  isNA <- !reduce(col, ~ (is.na(.x) & is.na(.y)))
  if (na.rm && isNA)
    unlist(map(col, ~ (if (is.na(.x)) "" else .x) ),use.names = F)
  else trimws(col, which = c("both")) # leading and trailing spaces
}
```

```r
trim.spaces <- function(df)
  df %>%
    mutate_if(is.character, trim.f, na.rm = T)

# Finally convert character columns to factor

char.to.fctor <- function(df)
  df %>%
    mutate_if(is.character, ~ factor(.x, levels = (.x %>% table() %>% names())))


wage.df <-
  wage.df %>%
  na.omit() %>%
  trim.spaces() %>%
  remove.empty.characters() %>%
  char.to.fctor()

# "sex" and "region" are factors with single levels are constants and to be rmoved
wage.df <- wage.df %>%
  select(-c(sex, region))

# wage (DV) is factor/continous (and hoefully normal).
# To see its relation with 1 or more interval IVs and/or 1 or more categorical IVs
# we use multiple regression
#(see: https://stats.idre.ucla.edu/other/mult-pkg/whatstat/)

summary(glm(wage ~ ., data = wage.df))
```

```
##
## Call:
## glm(formula = wage ~ ., data = wage.df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -16.721   -5.359   -3.058    0.712   94.152
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 -42.18179  229.31872  -0.184   0.8541
## year                         -0.18561    0.11449  -1.621   0.1051
## age                          -0.01716    0.02317  -0.740   0.4592
## maritl2. Married             -1.46198    0.65235  -2.241   0.0251 *
## maritl3. Widowed             -3.49564    2.96917  -1.177   0.2392
## maritl4. Divorced            -1.27917    1.07126  -1.194   0.2325
## maritl5. Separated           -2.50237    1.79938  -1.391   0.1644
## race2. Black                 -0.53547    0.79644  -0.672   0.5014
## race3. Asian                 -0.41331    0.96561  -0.428   0.6687
## race4. Other                  0.68373    2.10209   0.325   0.7450
## education2. HS Grad          -1.53915    0.88136  -1.746   0.0809 .
## education3. Some College     -2.38319    0.94682  -2.517   0.0119 *
## education4. College Grad     -0.77893    0.97362  -0.800   0.4238
## education5. Advanced Degree   4.86794    1.10266   4.415 1.05e-05 ***
## jobclass2. Information        0.64614    0.49152   1.315   0.1887
```

```
## health2. >=Very Good         -0.18688    0.52922  -0.353   0.7240
## health_ins2. No               4.41498    0.54460   8.107 7.51e-16 ***
## logwage                     113.26508    0.82815 136.769  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 159.0076)
##
##     Null deviance: 5222086  on 2999  degrees of freedom
## Residual deviance:  474161  on 2982  degrees of freedom
## AIC: 23740
##
## Number of Fisher Scoring iterations: 2
```

```
# summary shows only "maritl", "education", "health_ins"
# are sinnificantly important, in other word  "wage"
# has relationship with these 4 DV

# First note that all the IVs that has relationship with DV are factors so we
# use step function to model them. Let's model each one separately using step function:
library(grid)
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

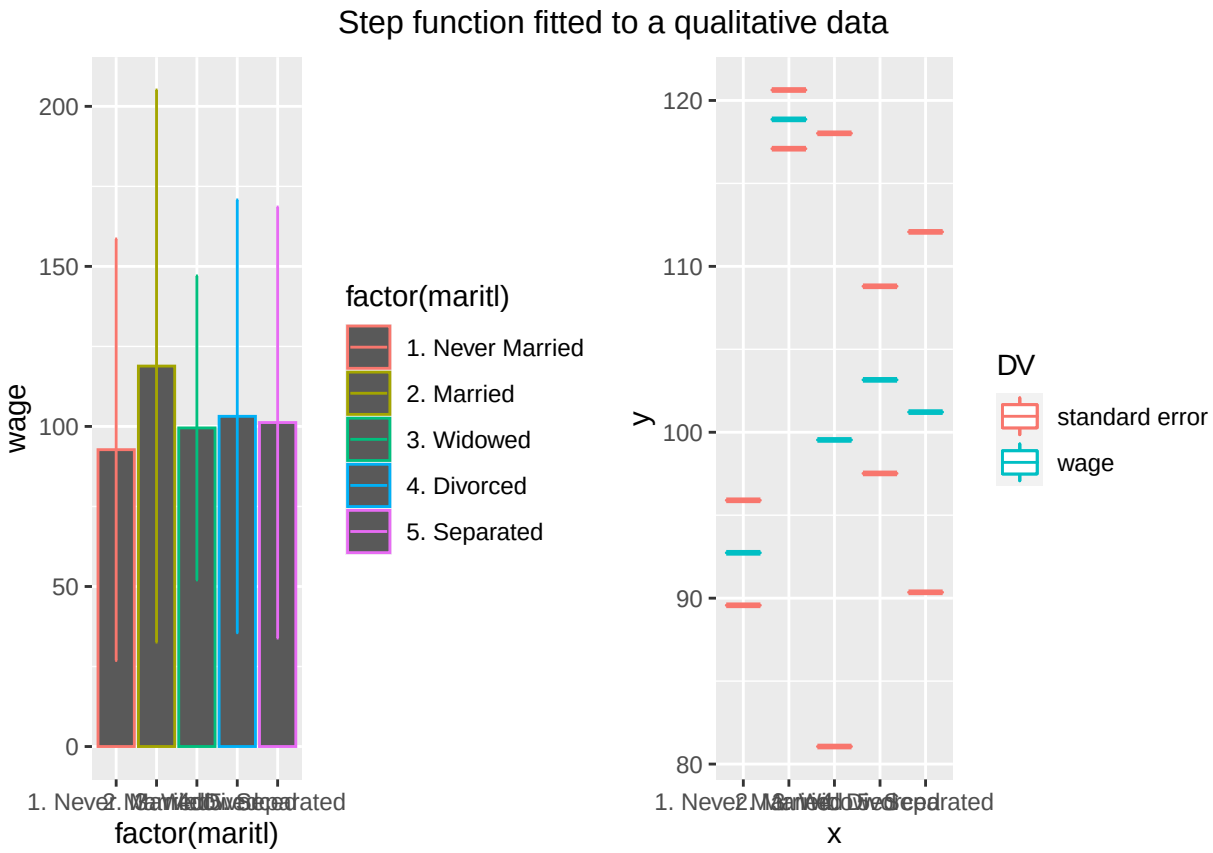
```
fit.step.model <- function(df, DV, IVs, seed=1113){
  frm <- as.formula(glue::glue(DV,"~", str_c(IVs, collapse = "+")))
  model.fit <- glm(frm, data = df)
  new.sample <- sample( levels(df [[IVs]]), 1000, replace=T)
  new.df = tibble(factor(new.sample))
  names(new.df) <- IVs
  preds <- predict(model.fit, newdata=new.df, se=T)

  se.bands <- cbind(preds$fit + 2 * preds$se.fit , preds$fit - 2 * preds$se.fit )

  data.to.draw <- tibble(x = factor(new.sample), y = preds$fit,
                         y.se.1 = preds$fit + 2 * preds$se.fit,
                         y.se.2 = preds$fit - 2 * preds$se.fit)

  # draw box plots for quality columns
  g1 <- ggplot(df, aes(factor(.data[[IVs]]), .data[[DV]], color = factor(.data[[IVs]]))) +
    geom_bar(aes(factor(.data[[IVs]]), .data[[DV]]),  stat = "summary", fun.y = mean) +
    stat_summary(fun.data = mean_sdl, width=0.05, geom = "errorbar",fun.y = mean)


  #the sample we predict on
  g2 <- ggplot(data.to.draw, aes(x=x)) +
    geom_boxplot(aes(y = y, colour = DV)) +
    geom_boxplot(aes(y = y.se.1, colour = "standard error")) +
    geom_boxplot(aes(y = y.se.2, colour = "standard error"))
```

```
  grid.arrange(g1, g2, nrow = 1,
               top = textGrob("Step function fitted to a qualitative data"))
}

fit.step.model(wage.df, "wage", "maritl")
```

```
## Warning: Ignoring unknown parameters: fun.y
```

```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```

```
## No summary function supplied, defaulting to `mean_se()`
```

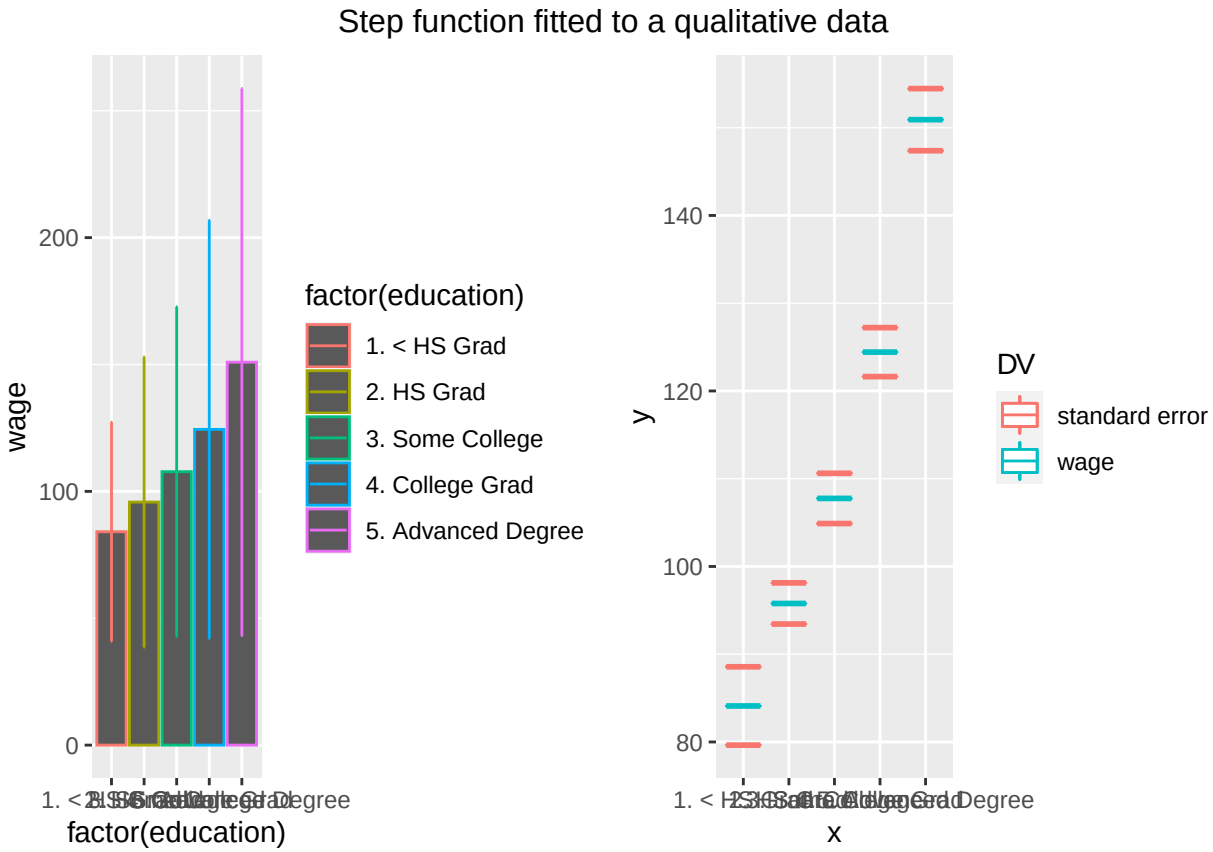



```
fit.step.model(wage.df, "wage", "education")
```

```
## Warning: Ignoring unknown parameters: fun.y
```

```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```

```
## No summary function supplied, defaulting to `mean_se()`
```

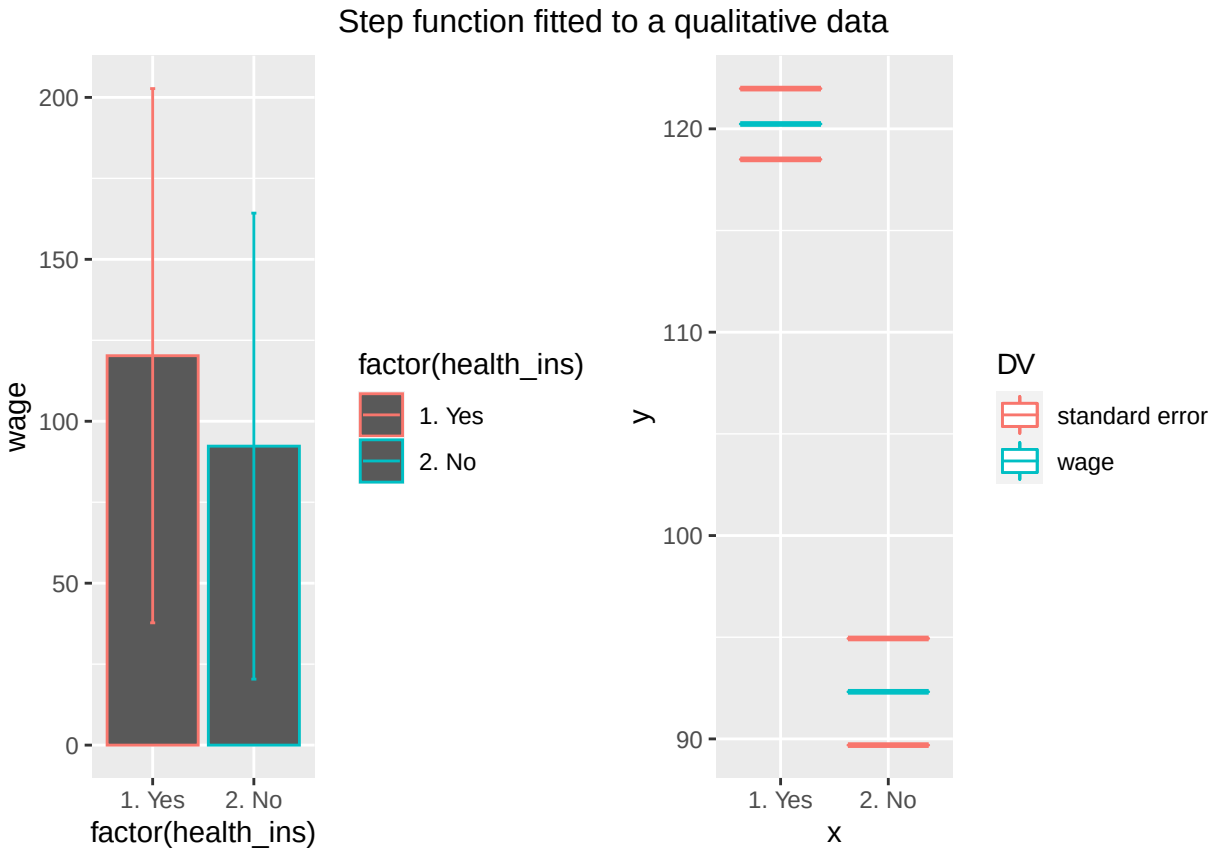



```
fit.step.model(wage.df, "wage", "health_ins")
```

```
## Warning: Ignoring unknown parameters: fun.y
```

```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```

```
## No summary function supplied, defaulting to `mean_se()`
```




```
library(boot)
library(tidyverse)
library(dataPreparation)
```

```
## Loading required package: lubridate
```

```
##
## Attaching package: 'lubridate'
```

```
## The following object is masked from 'package:base':
##
##     date
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loading required package: progress
```

```
## dataPreparation 0.4.3
```

```
## Type dataPrepNews() to see new features/changes/bug fixes.
```

```
library(mgcv)

auto.df = read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Auto.csv", header=T, stri
```

```
auto.df.original = tibble(auto.df)
auto.df = tibble(auto.df)

# remove empty characters and NA helper
remove.empty.characters <- function(df)
  df %>%
    select_all %>%
    filter_if(is.character, any_vars(!is.na(.) & trimws(.) != ""))

# Next remove leading and trailing spaces from all elements in character columns
trim.f <- function(col, na.rm = F) {
  isNA <- !reduce(col, ~ (is.na(.x) & is.na(.y)))
  if (na.rm && isNA)
    unlist(map(col, ~ (if (is.na(.x)) "" else .x) ),use.names = F)
  else trimws(col, which = c("both")) # leading and trailing spaces
}

trim.spaces <- function(df)
  df %>%
    mutate_if(is.character, trim.f, na.rm = T)

# Finally convert character columns to factor

char.to.fctor <- function(df)
  df %>%
    mutate_if(is.character, ~ factor(.x, levels = (.x %>% table() %>% names())))


auto.df <-
  auto.df %>%
  na.omit() %>%
  trim.spaces() %>%
  remove.empty.characters() %>%
  char.to.fctor()

# remove constant variables
(constant_cols <- whichAreConstant(auto.df))
```

```
## [1] "whichAreConstant: it took me 0s to identify 0 constant column(s)"
```

```
## integer(0)
```

```
# remove Variables that are in double (for example col1 == col2)
(double_cols <- whichAreInDouble(auto.df))
```

```
## [1] "whichAreInDouble: it took me 0.01s to identify 0 column(s) to drop."
```

```
## integer(0)
```

```
# remove Variables that are exact bijections (for example col1 = A, B, B, A and col2 = 1, 2, 2, 1)
(bijections_cols <- whichAreBijection(auto.df))
```

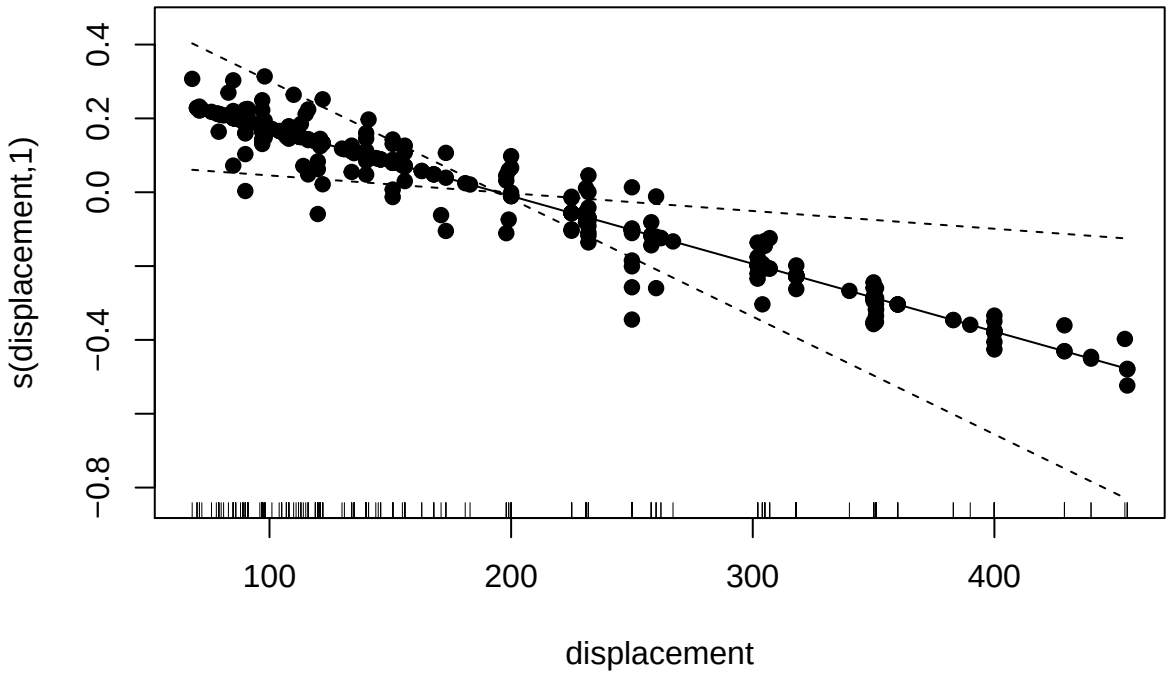
```
## [1] "whichAreBijection: it took me 0.03s to identify 0 column(s) to drop."
```

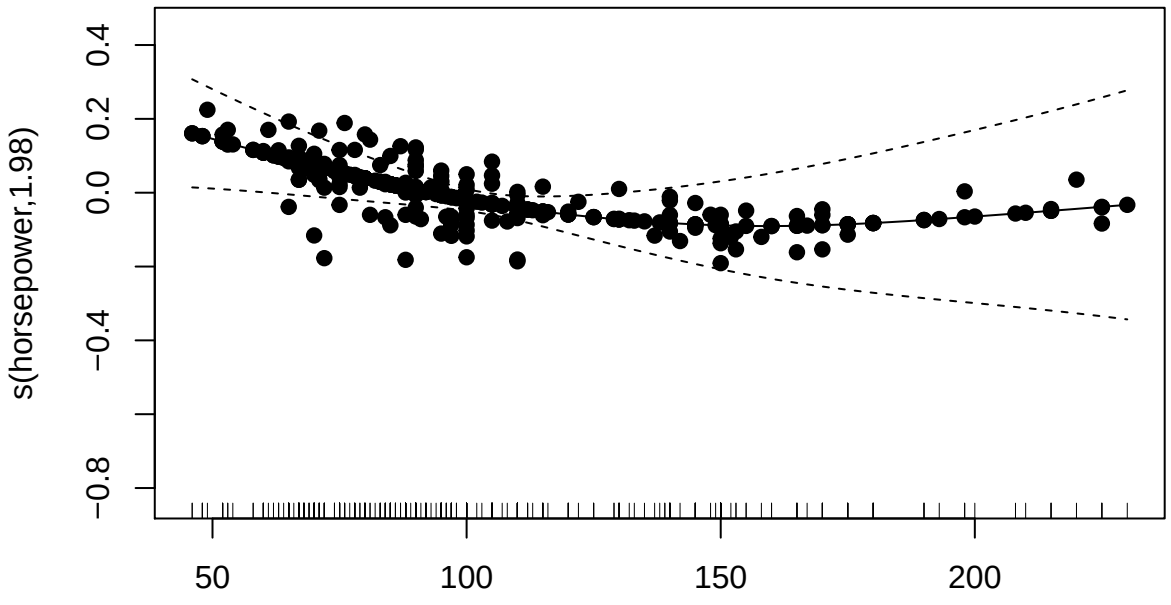
```
## integer(0)
```

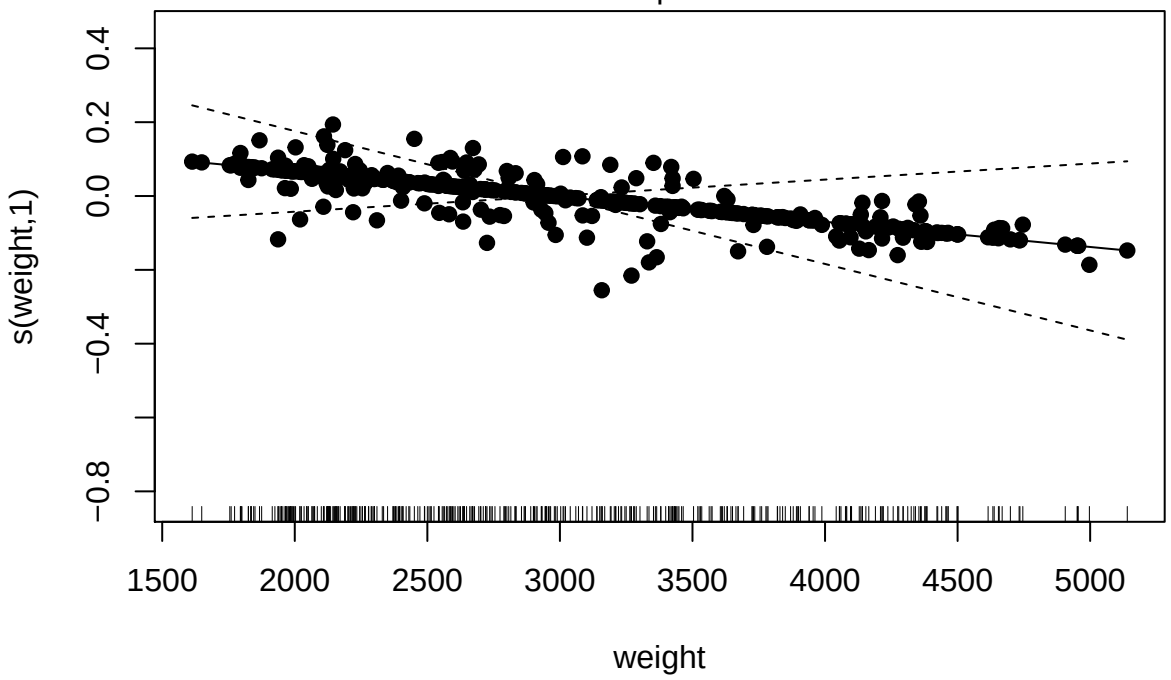
```r
# we fit smoothing spine and step function using GAM'
# bs="cr" cubic regression spline
# Gamma is usefull because DV (i.e mpg) is strictly positive real valued
# also defaukt link is used in some waiting time applications, log link is most often used
# Use GCV.CP to estimate the parameter
# k-1 is upper limit for degree of freedom using GCV
# gamma = 1.2 since GCV tends to slightly overfit
# bs="cr" is cubic regression , default is thin plate spline which is computatinally very heavy

model.fit <- gam(mpg ~ s(displacement, bs="cr", k= 13) +
             s(horsepower, bs="cr", k= 13) + s(weight, bs="cr", k= 13) +
             s(acceleration, bs="cr", k= 13) + name + cylinders + year + origin,
             method="GACV.Cp", scale=-1,family = Gamma(link=log), data=auto.df, gamma = 1.2)

model.fit
```

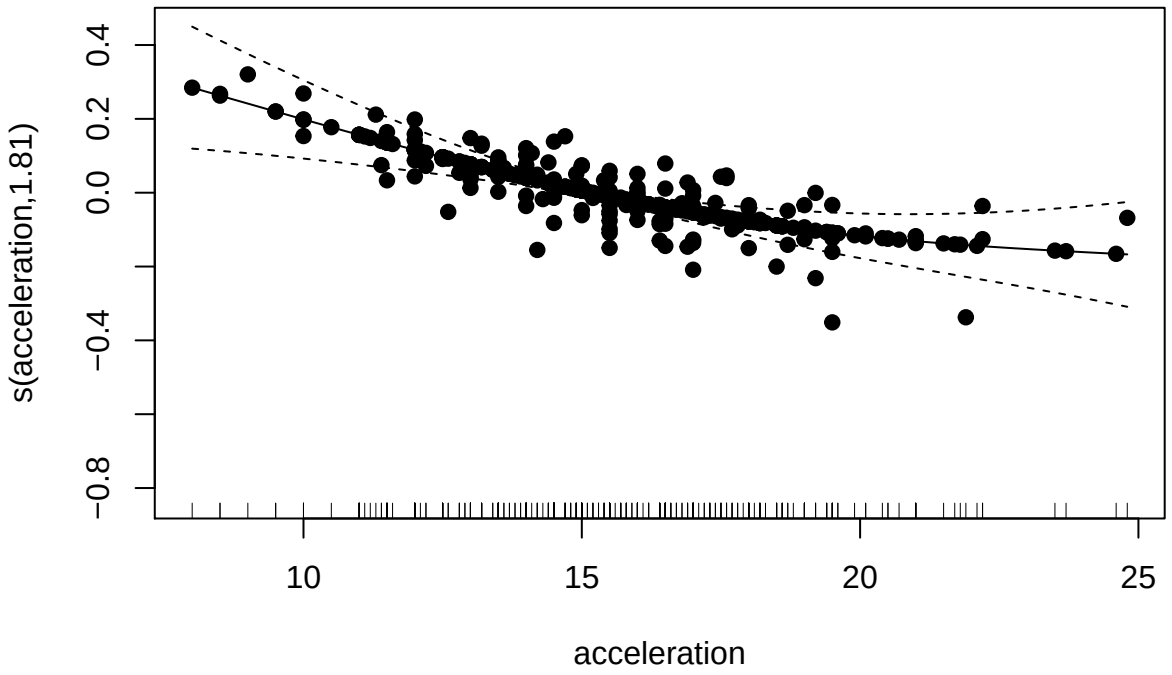
```
##
## Family: Gamma
## Link function: log
##
## Formula:
## mpg ~ s(displacement, bs = "cr", k = 13) + s(horsepower, bs = "cr",
##     k = 13) + s(weight, bs = "cr", k = 13) + s(acceleration,
##     bs = "cr", k = 13) + name + cylinders + year + origin
##
## Estimated degrees of freedom:
## 1.00 1.98 1.00 1.81  total = 308.79
##
## GACV score: 0.05375584     rank: 351/352
```

```r
plot(model.fit,residuals=TRUE,pch=19)
```

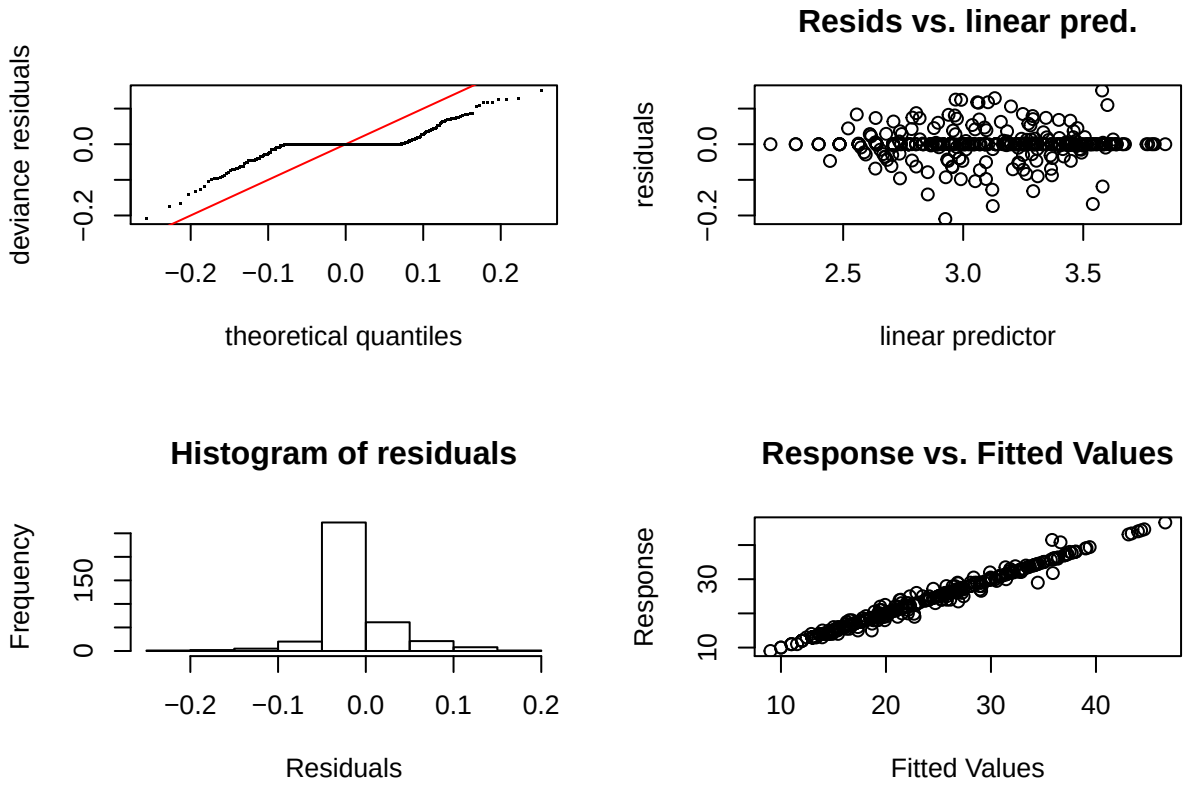



s(horsepower,1.98)
0.4
0.2
0.0
−0.4
−0.8
50
100
150
200
horsepower


s(weight,1)
0.4
0.2
0.0
−0.4
−0.8
1500
2000
2500
3000
3500
4000
4500
5000
weight



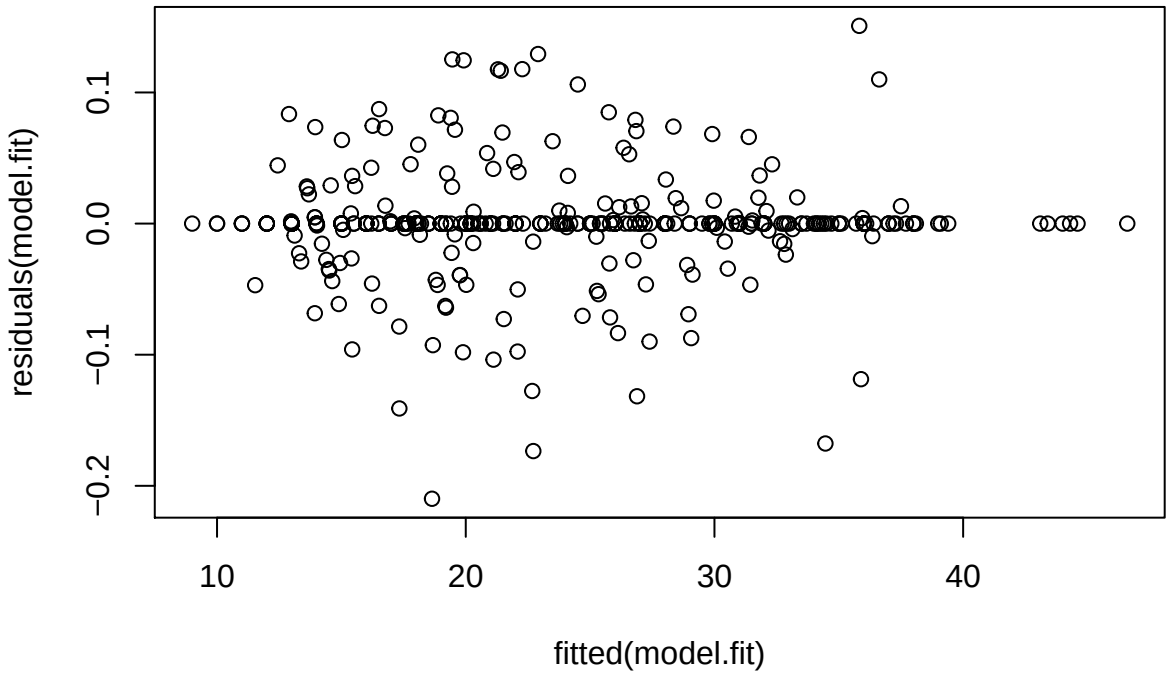

```
gam.check(model.fit)
```



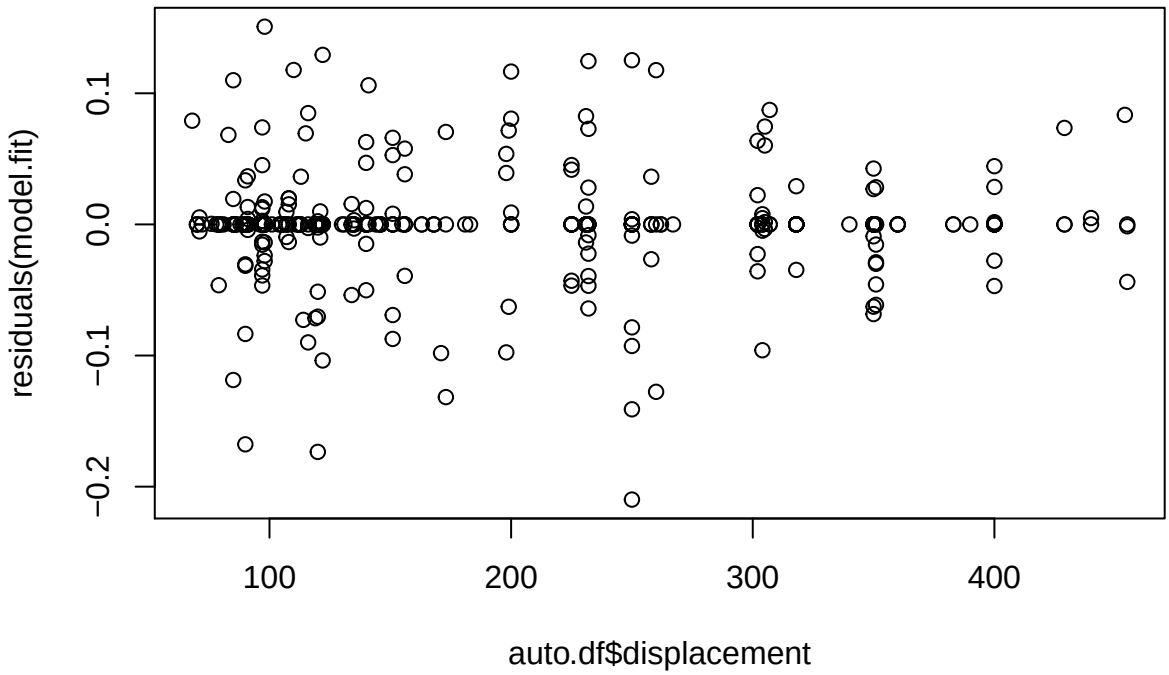

```
##
## Method: GACV   Optimizer: outer newton
## full convergence after 18 iterations.
## Gradient range [-1.028877e-08,-2.63748e-10]
## (score 0.05375584 & scale 0.007121396).
## Hessian positive definite, eigenvalue range [1.023544e-08,0.001311472].
```

```
## Model rank =  351 / 352
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                     k'   edf k-index p-value
## s(displacement) 12.00  1.00    1.00    0.49
## s(horsepower)   12.00  1.98    0.98    0.28
## s(weight)       12.00  1.00    0.94    0.10
## s(acceleration) 12.00  1.81    1.12    0.98
```

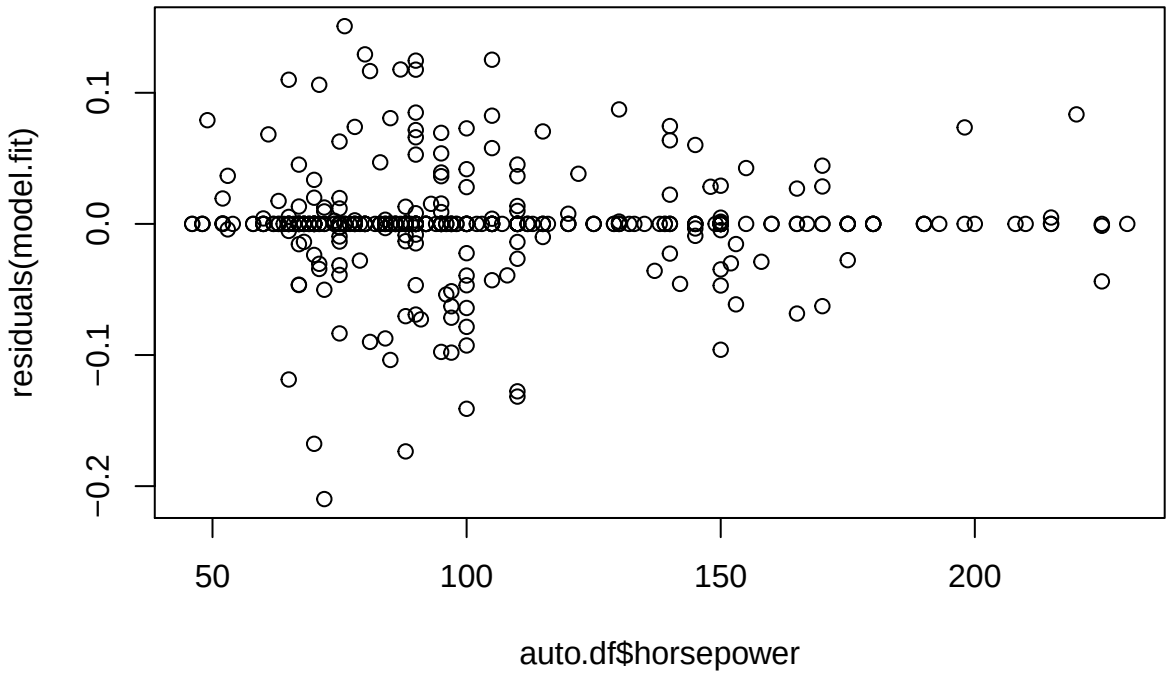
```
plot(fitted(model.fit),residuals(model.fit))
```



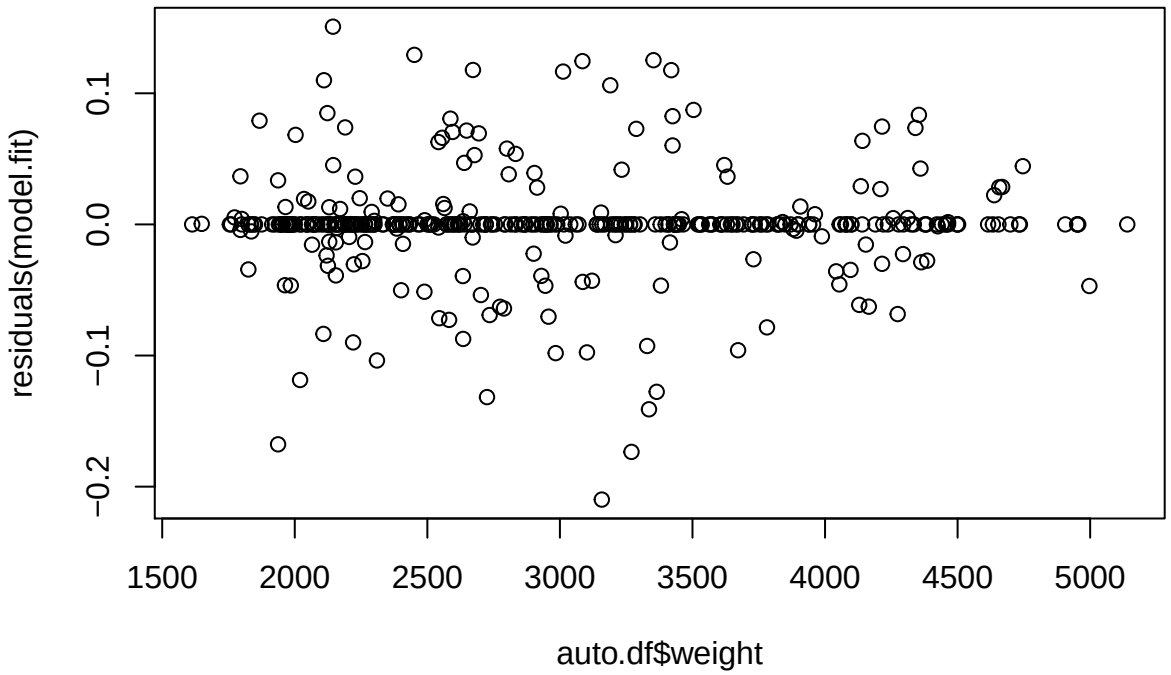

```
plot(auto.df$displacement,residuals(model.fit))
```




```
plot(auto.df$horsepower,residuals(model.fit))
```



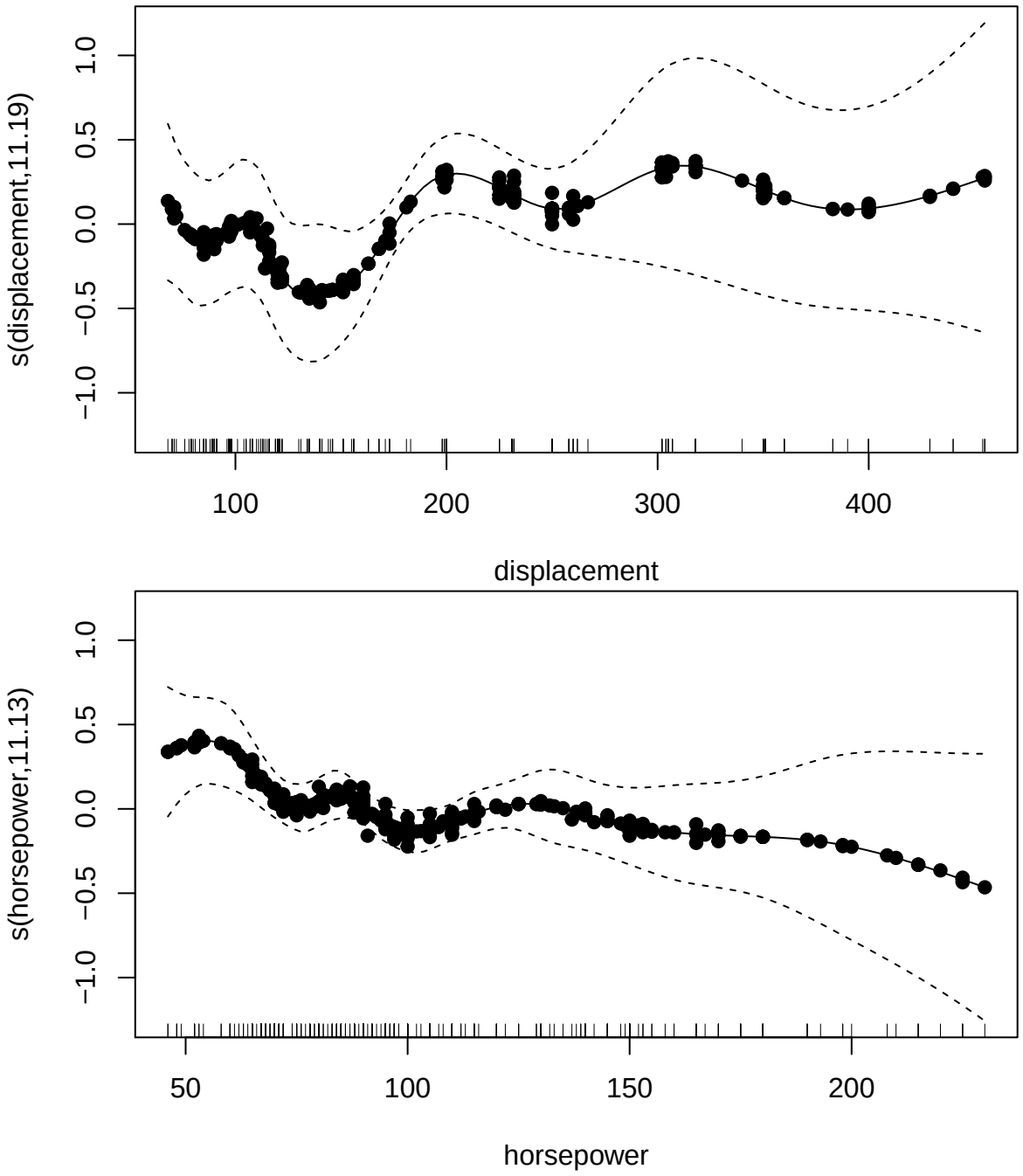

```
plot(auto.df$weight,residuals(model.fit))
```

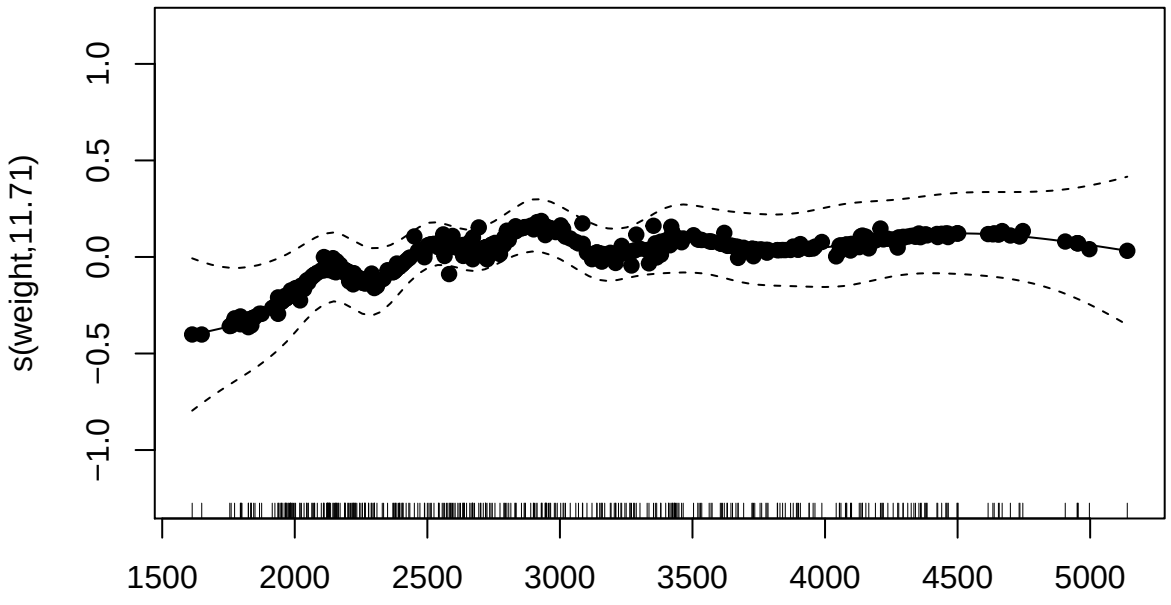


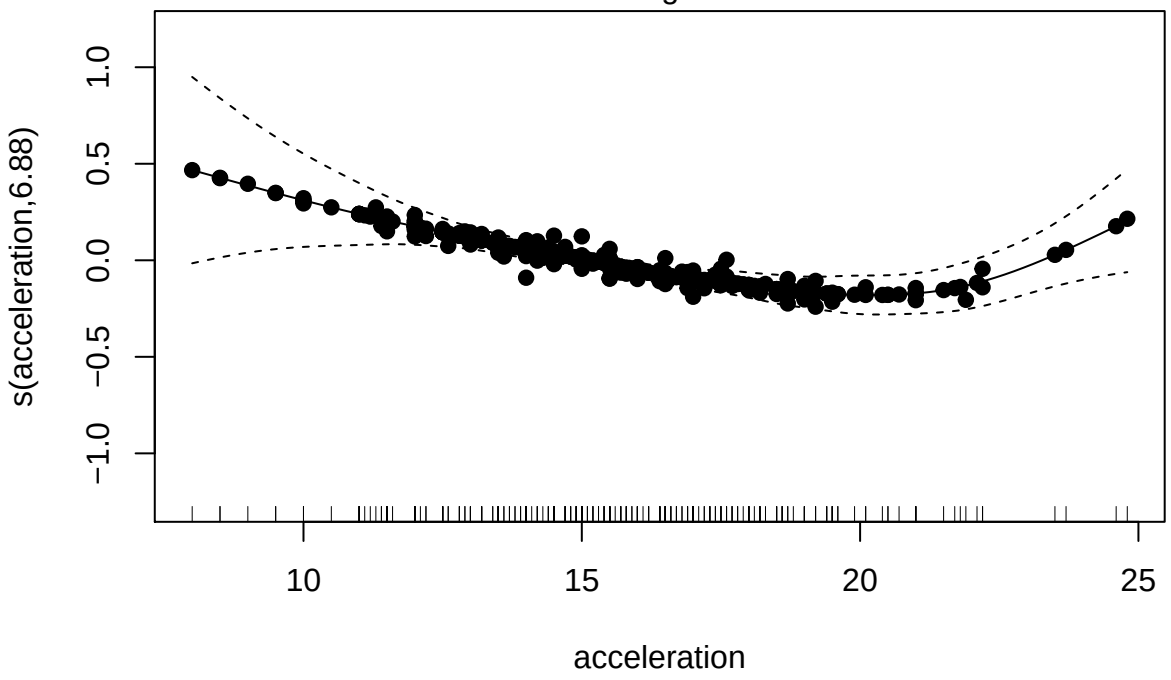

```
# vis.gam(model.fit,theta=-45,ticktype="detailed",se=2)
# vis.gam(model.fit,theta=30,ticktype="detailed")
# vis.gam(model.fit,plot.type="contour")

# Model shows there is a non linear relationship between
# mpg on one hand and  horsepower and acceleration on the other hand
# however mpgg has linear relation with displacement and


# to check the robostness model with different criterion
# (i.e. "ML" or "REML" versus "GCV.Cp" or "GACV.Cp")
# because likelihood based methods tend to be more robust

model.fit1 <- gam(mpg ~ s(displacement, bs="cr", k= 13) +
             s(horsepower, bs="cr", k= 13) + s(weight, bs="cr", k= 13) +
             s(acceleration, bs="cr", k= 13) + name + cylinders + year + origin,
             method="ML", scale=-1,family = Gamma(link=log), data=auto.df, gamma = 1)

model.fit1
```

```
##
## Family: Gamma
## Link function: log
##
## Formula:
## mpg ~ s(displacement, bs = "cr", k = 13) + s(horsepower, bs = "cr",
##     k = 13) + s(weight, bs = "cr", k = 13) + s(acceleration,
##     bs = "cr", k = 13) + name + cylinders + year + origin
##
## Estimated degrees of freedom:
## 11.19 11.13 11.71  6.88  total = 343.92
##
## ML score: 469.525     rank: 351/352
```

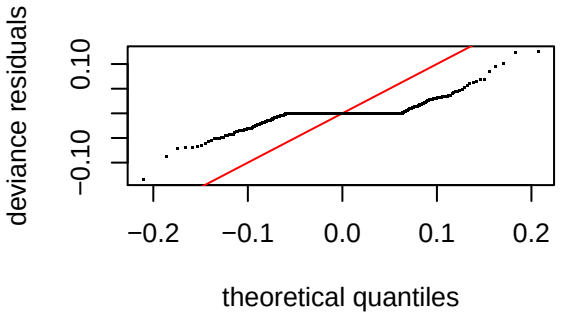
```
plot(model.fit1,residuals=TRUE,pch=19)
```

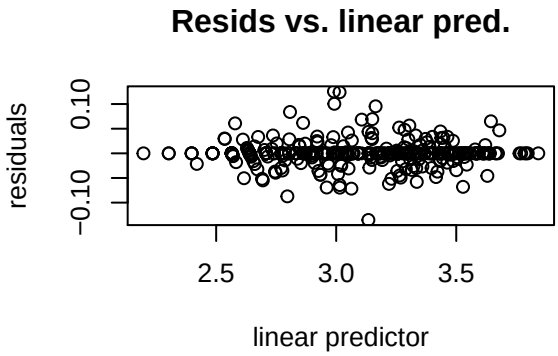


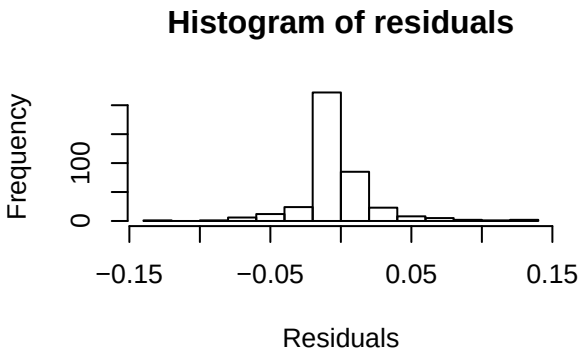


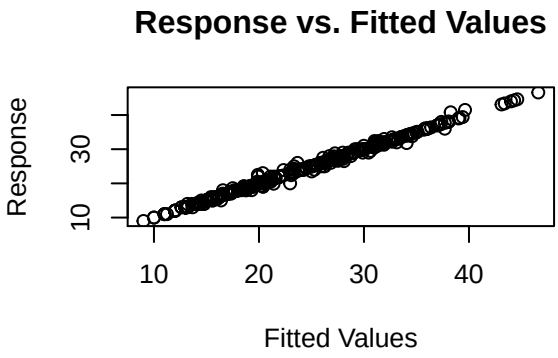



```
gam.check(model.fit1)
```



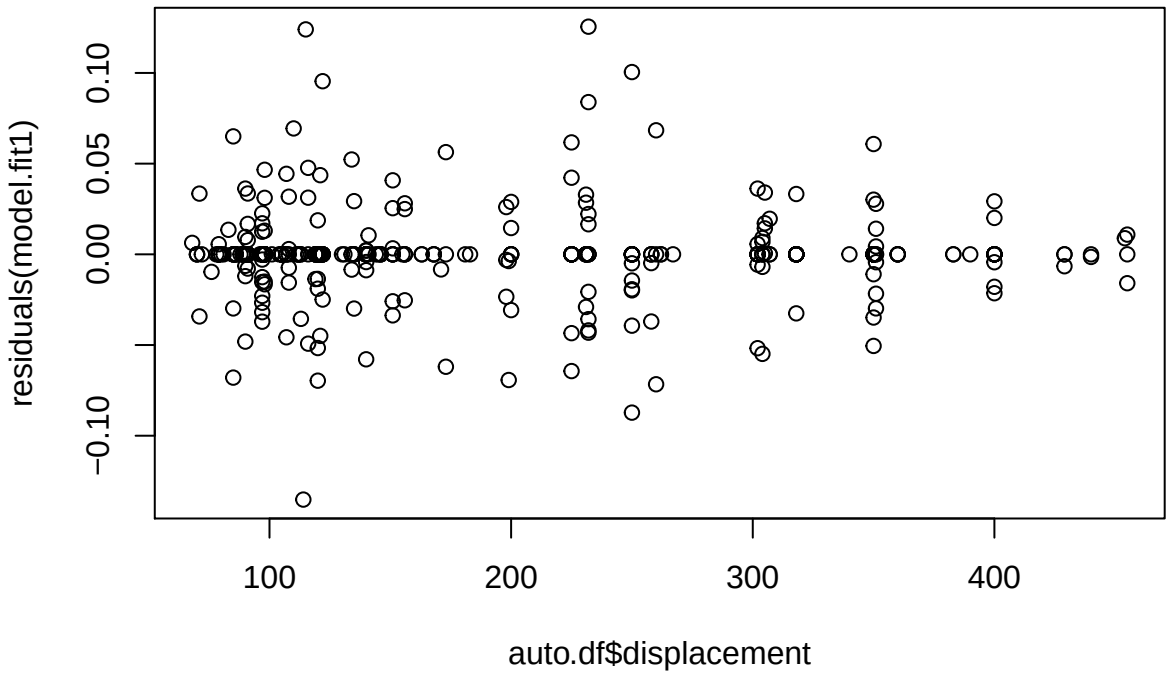




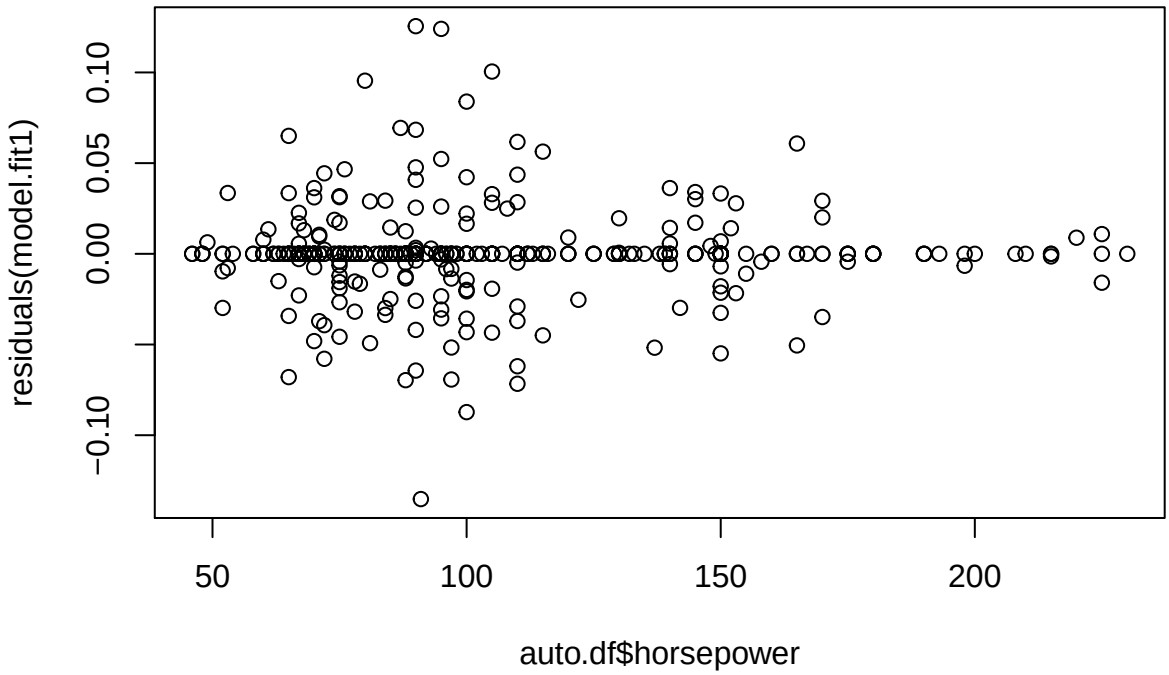



```
##
## Method: ML   Optimizer: outer newton
## full convergence after 8 iterations.
## Gradient range [-9.740937e-07,2.596516e-07]
## (score 469.525 & scale 0.004789358).
## Hessian positive definite, eigenvalue range [1.596529,196.6053].
## Model rank =  351 / 352
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                    k'   edf k-index p-value
## s(displacement) 12.00 11.19    1.14    0.99
## s(horsepower)   12.00 11.13    1.08    0.94
## s(weight)       12.00 11.71    1.01    0.54
## s(acceleration) 12.00  6.88    1.09    0.96
```

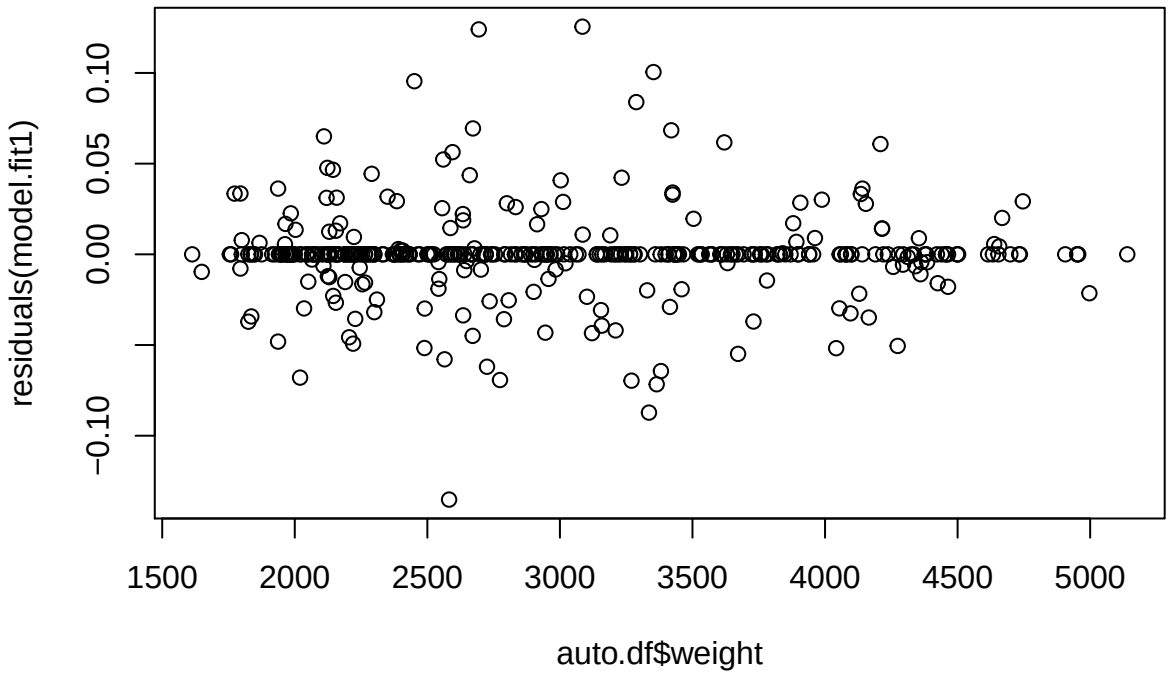
```
plot(auto.df$displacement,residuals(model.fit1))
```




```
plot(auto.df$horsepower,residuals(model.fit1))
```




```
plot(auto.df$weight,residuals(model.fit1))
```




```
# Clearly ML has a tendency to overfit, therefore we see all models have high edf
# vis.gam(model.fit1,theta=-45,ticktype="detailed",se=2)
# vis.gam(model.fit1,theta=30,ticktype="detailed")
# vis.gam(model.fit1,plot.type="contour")
```

```
library(boot)
library(tidyverse)
library(dataPreparation)
library(splines)

boston.df = read.csv(
  "/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/BostonHousing.csv",
  header=T, stringsAsFactors = F, na.strings = "?")

boston.df <- tibble(boston.df)

# see if there is any NA in any records
nrow(df[which(is.na(boston.df)),])
```

```
## [1] 0
```

```
# No NA anywhere , let's do usual clean up:

# remove constant variables
constant_cols <- whichAreConstant(boston.df)
```

```
## [1] "whichAreConstant: it took me 0s to identify 0 constant column(s)"
```

```
# remove Variables that are in double (for example col1 == col2)
(double_cols <- whichAreInDouble(boston.df))
```

```
## [1] "whichAreInDouble: it took me 0s to identify 0 column(s) to drop."
```

```
## integer(0)
```

```r
# remove Variables that are exact bijections (for example col1 = A, B, B, A and col2 = 1, 2, 2, 1)
(bijections_cols <- whichAreBijection(boston.df))
```

```
## [1] "whichAreBijection: it took me 0.04s to identify 0 column(s) to drop."
```

```
## integer(0)
```

```r
# a) use ploynomial regression to predict nox using dis

# Now fit a 4 degree polynomial
fit.poly <- glm(nox ~ poly(dis, 3), data = boston.df)

# Here is the regression output
# polynomial coefficients are all statistically significamt
summary(fit.poly)
```

```
##
## Call:
## glm(formula = nox ~ poly(dis, 3), data = boston.df)
##
## Deviance Residuals:
##       Min         1Q     Median         3Q        Max
## -0.121130  -0.040619  -0.009738   0.023385   0.194904
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.554695   0.002759 201.021  < 2e-16 ***
## poly(dis, 3)1 -2.003096   0.062071 -32.271  < 2e-16 ***
## poly(dis, 3)2  0.856330   0.062071  13.796  < 2e-16 ***
## poly(dis, 3)3 -0.318049   0.062071  -5.124 4.27e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.003852802)
##
##     Null deviance: 6.7810  on 505  degrees of freedom
## Residual deviance: 1.9341  on 502  degrees of freedom
## AIC: -1370.9
##
## Number of Fisher Scoring iterations: 2
```

```r
# draw standard error
# first get the range of the values of dis
(dis.limits  <- range(boston.df$dis))
```

```
## [1]  1.1296 12.1265
```

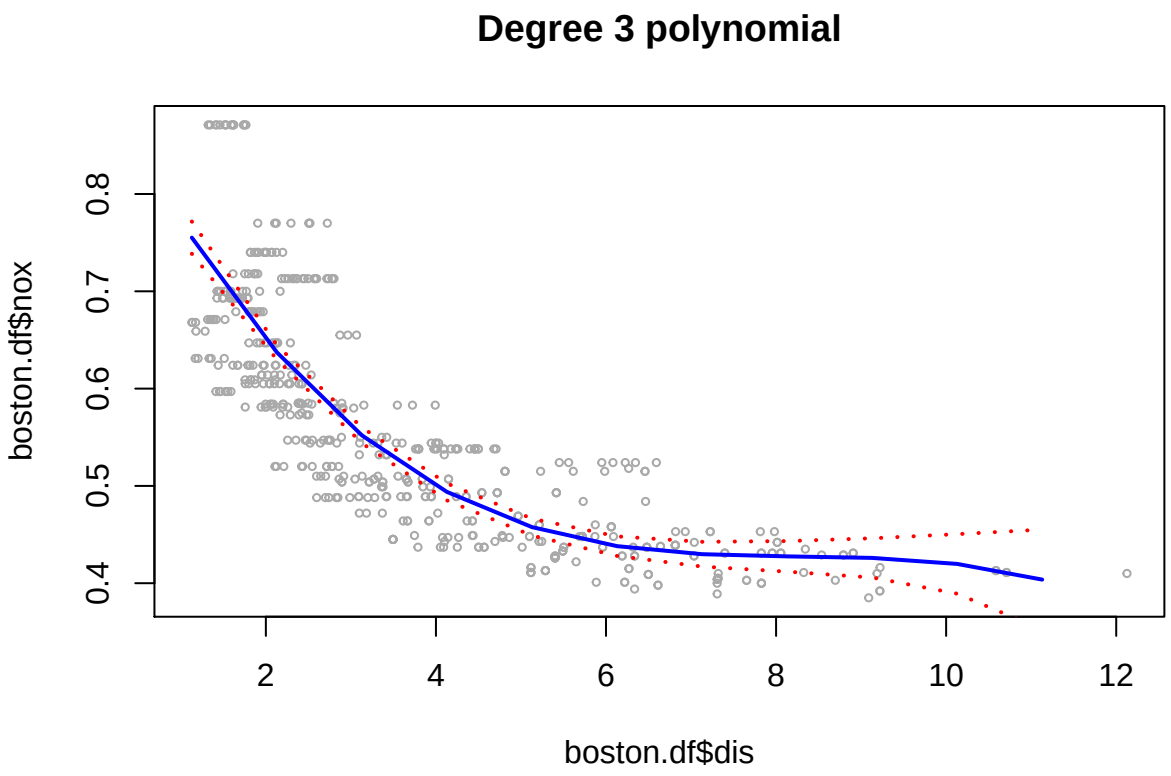
```r
# Now create an interaval from these range values
(dis.grid <- seq(from=dis.limits[1], to=dis.limits[2]))
```

```
##  [1]  1.1296  2.1296  3.1296  4.1296  5.1296  6.1296  7.1296  8.1296  9.1296
## [10] 10.1296 11.1296
```

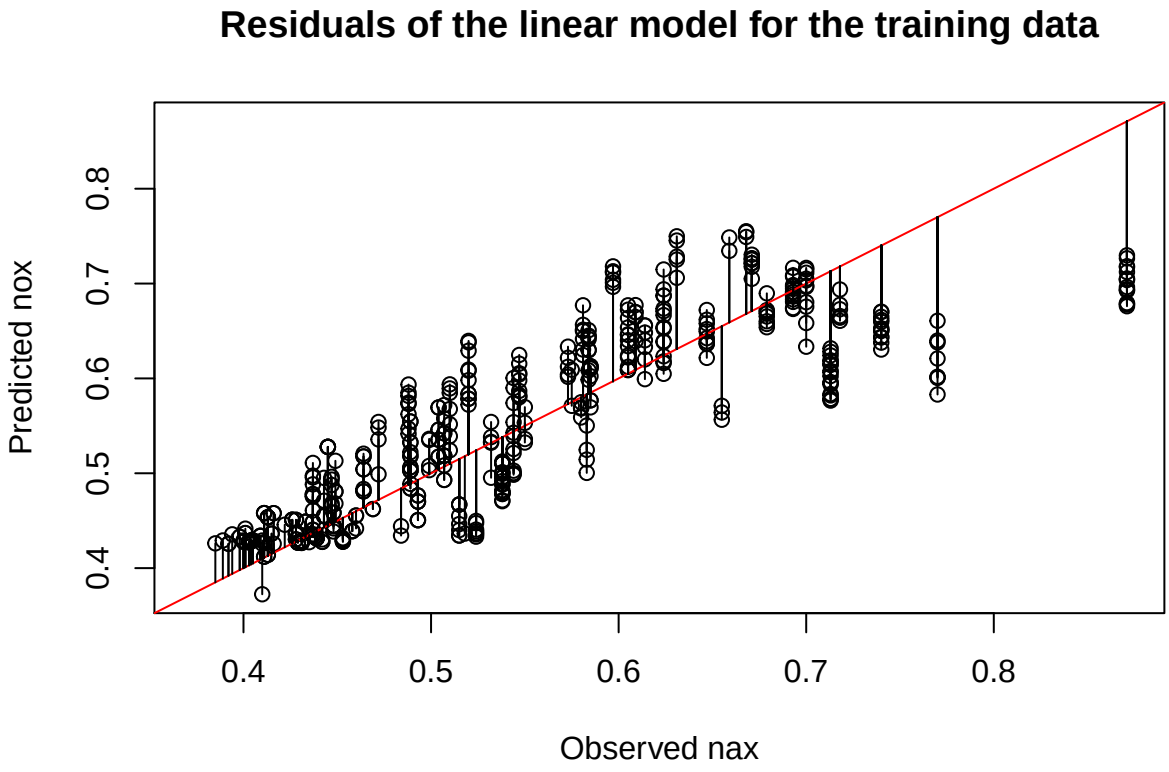
```r
# predict value for this interval using fitted model
predicts <- predict(fit.poly, newdata = tibble(dis=dis.grid), se=T)
names(predicts)
```

```
## [1] "fit"            "se.fit"         "residual.scale"
```

```
se.bands <- cbind(predicts$fit + 2*predicts$se.fit,
                  predicts$fit - 2*predicts$se.fit)

# now plot the predicts
plot(boston.df$dis, boston.df$nox, xlim=dis.limits, cex=0.5, col="darkgrey")
title("Degree 3 polynomial", outer = F)
lines(dis.grid, predicts$fit, lwd=2, col = "blue")
matlines(dis.grid, se.bands, lwd=2, col="red", lty=3)
```

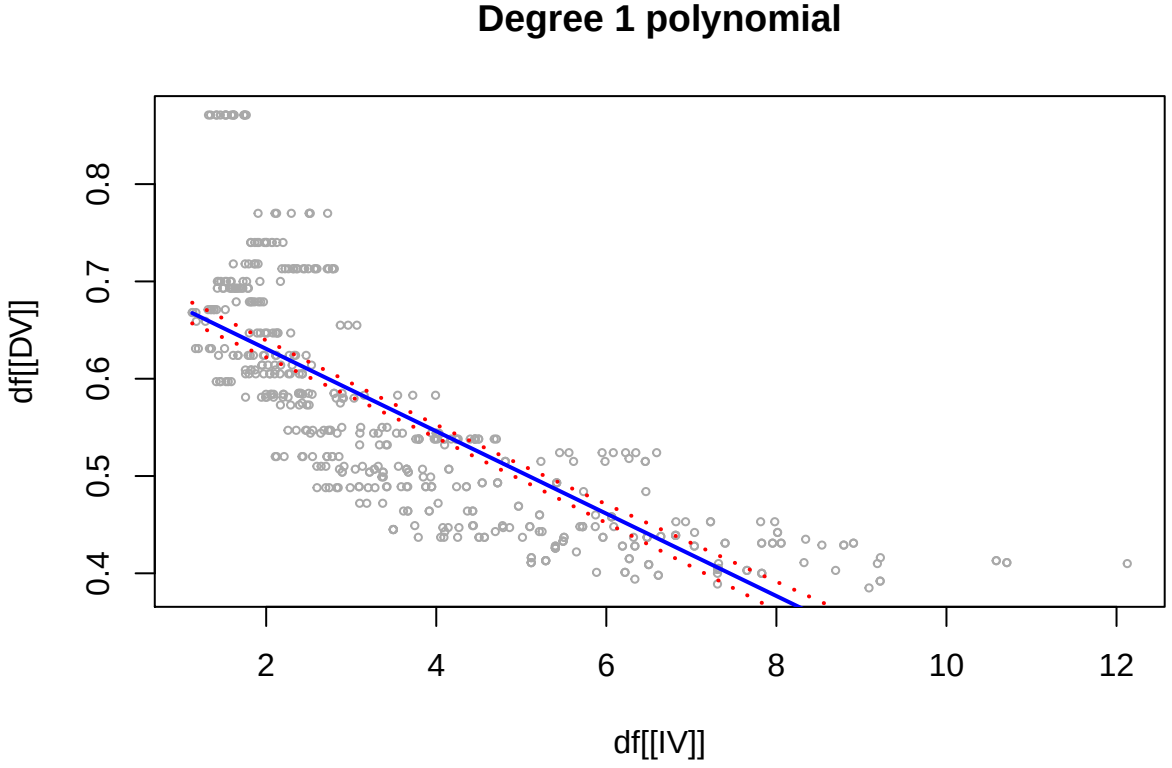


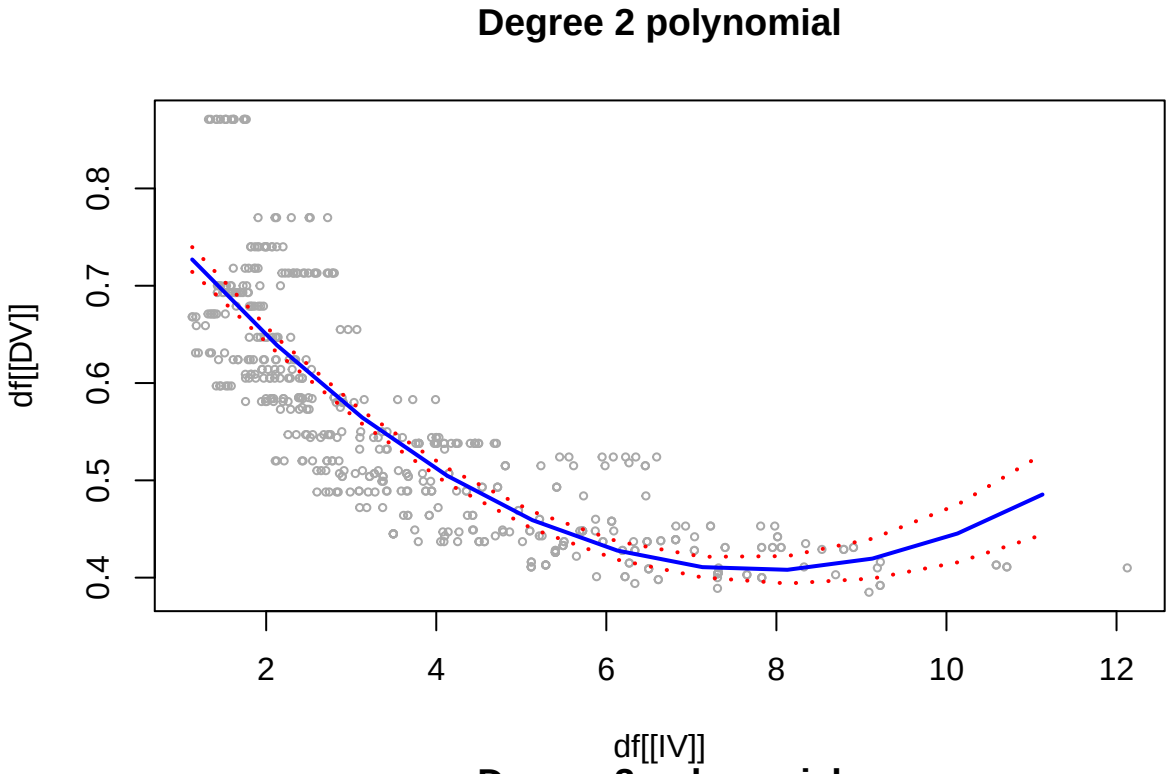

```
res <- residuals(fit.poly, type = "response")
# ensure that x- and y-axis have the same range
pred <- fit.poly$fitted.values

observed <- boston.df$nox
# determine maximal range
val.range <- range(pred, observed)
plot(observed, pred,
    xlim = val.range, ylim = val.range,
    xlab = "Observed nax",
    ylab = "Predicted nox",
    main = "Residuals of the linear model for the training data")
# show ideal prediction as a diagonal
abline(0,1, col = "red")
# add residuals to the plot
segments(observed, pred, observed, pred + res)
```

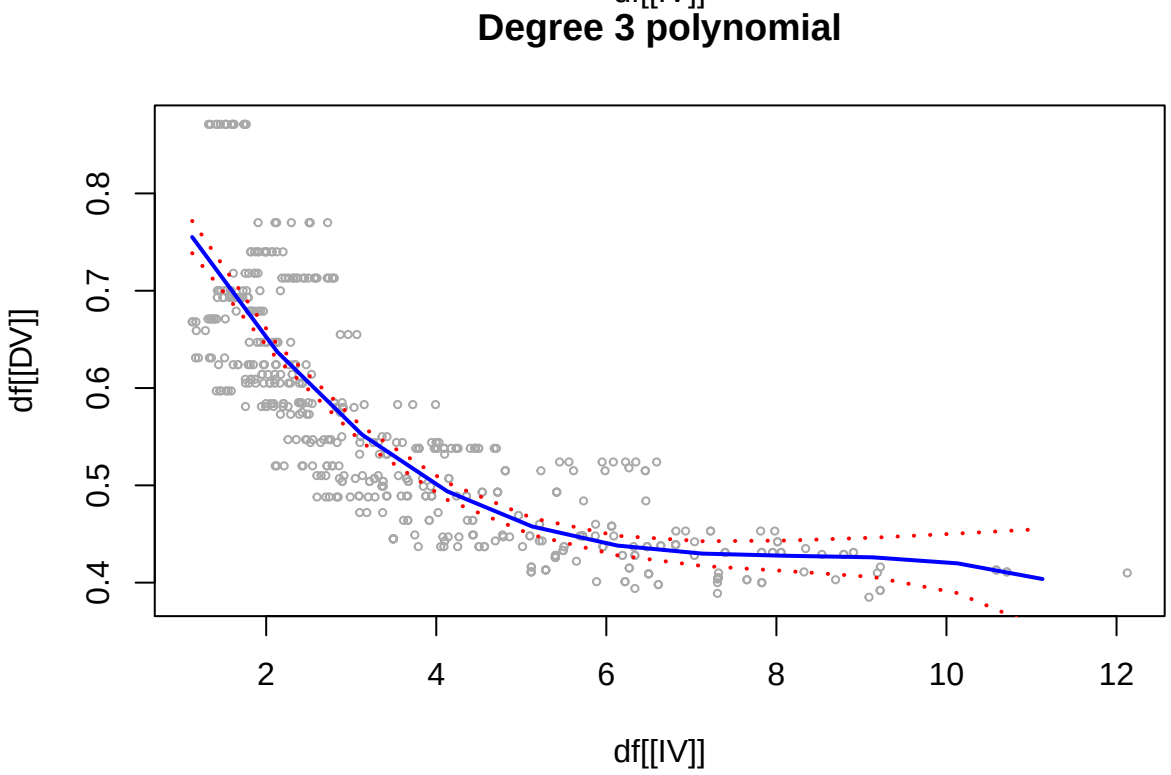


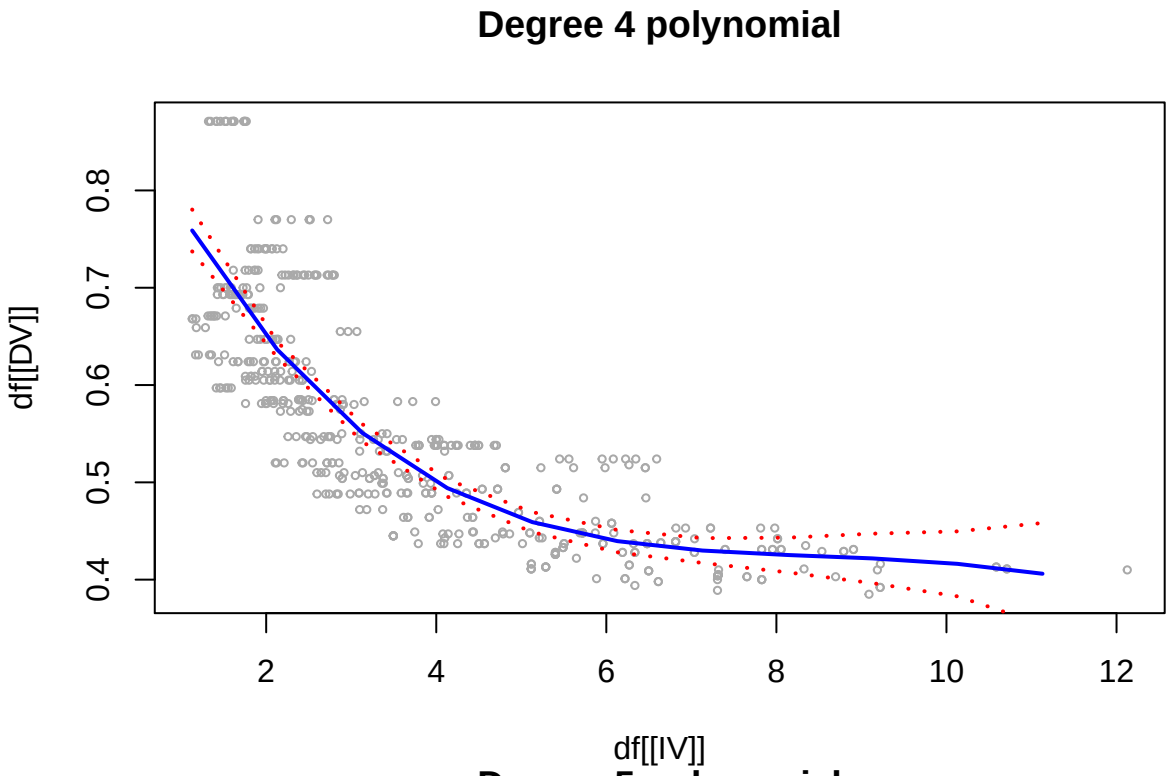

```
# calculate RSS
rss <- res %>% reduce( ~.x + .y^2 , .init = 0) %>% unlist


# ------------------------------------------------------------------
# b) plot plynomial fit for a range of degrees and report each rss
# ------------------------------------------------------------------
fit.ploys.fun <- function (df, IV, DV, i) {

  # Now fit a polynomial with degree i
  frm <- as.formula(glue::glue(DV," ~ poly(",IV,",",i,")"))
  fit.poly <- glm(frm, data = df)

  # Here is the regression output
  # polynomial coefficients are all statistically significamt
  summary(fit.poly)

  # draw standard error
  # first get the range of the values of dis
  IV.limits  <- range(df[[IV]])

  # Now create an interaval from these range values
  IV.grid <- seq(from=IV.limits[1], to=IV.limits[2])

  # predict value for this interval using fitted model
  new.data <- tibble(IV.grid)
  names(new.data)  <- IV
  predicts <- predict(fit.poly, newdata = new.data, se=T)
```

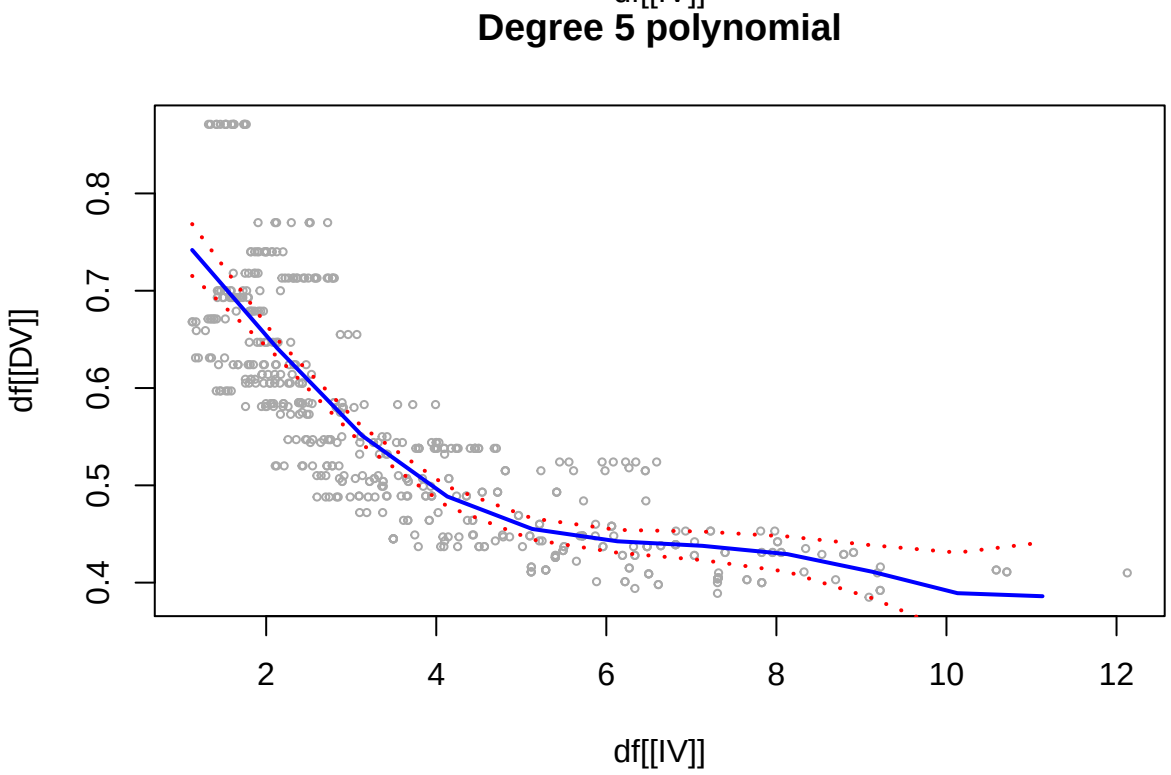
```
  se.bands <- cbind(predicts$fit + 2*predicts$se.fit,
                    predicts$fit - 2*predicts$se.fit)

  # now plot the predicts
  plot(df[[IV]], df[[DV]], xlim=IV.limits, cex=0.5, col="darkgrey")
  title(glue::glue("Degree ",i, " polynomial"), outer = F)
  lines(IV.grid, predicts$fit, lwd=2, col = "blue")
  matlines(IV.grid, se.bands, lwd=2, col="red", lty=3)

  res <- residuals(fit.poly, type = "response")

  # calculate RSS
  res %>% reduce( ~.x + .y^2 , .init = 0) %>% unlist

}

# RSS for ploynomilas degree 1 to 10
1:10 %>% reduce (~append(.x,fit.ploys.fun(boston.df, "dis", "nox", .y)),
                 .init = NULL)
```

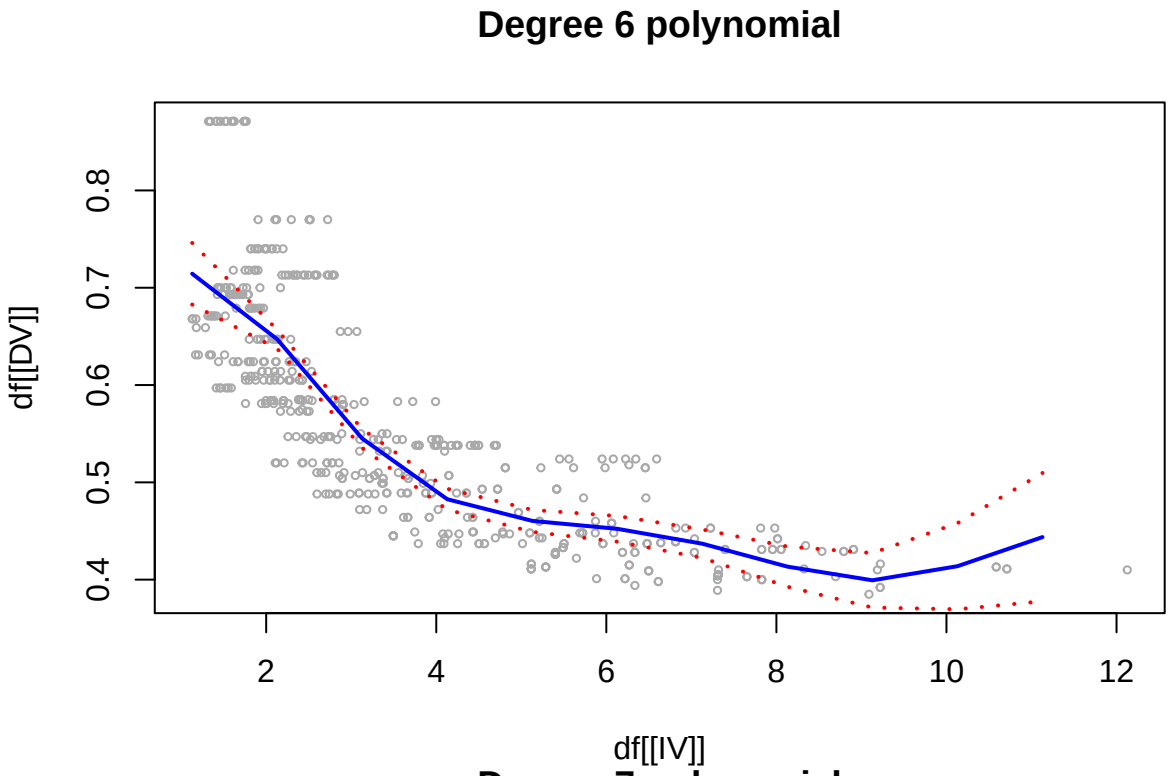


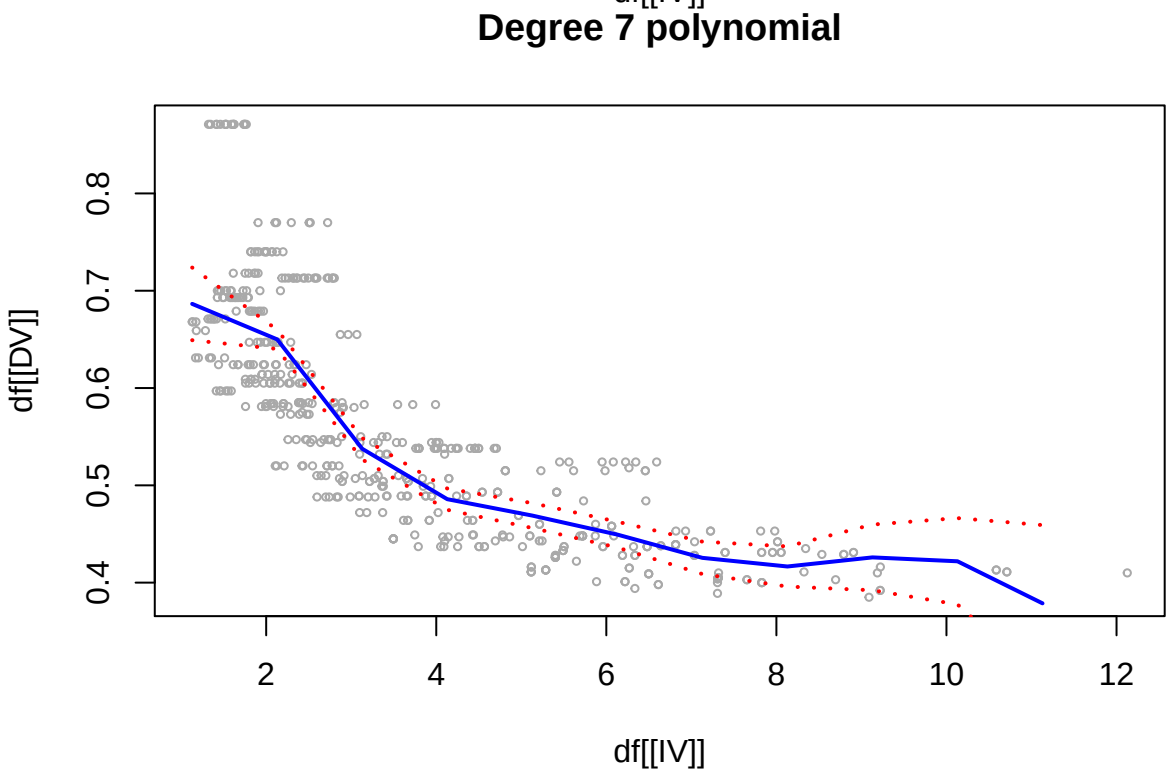

Degree 2 polynomial
df[[DV]]
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
df[[IV]]

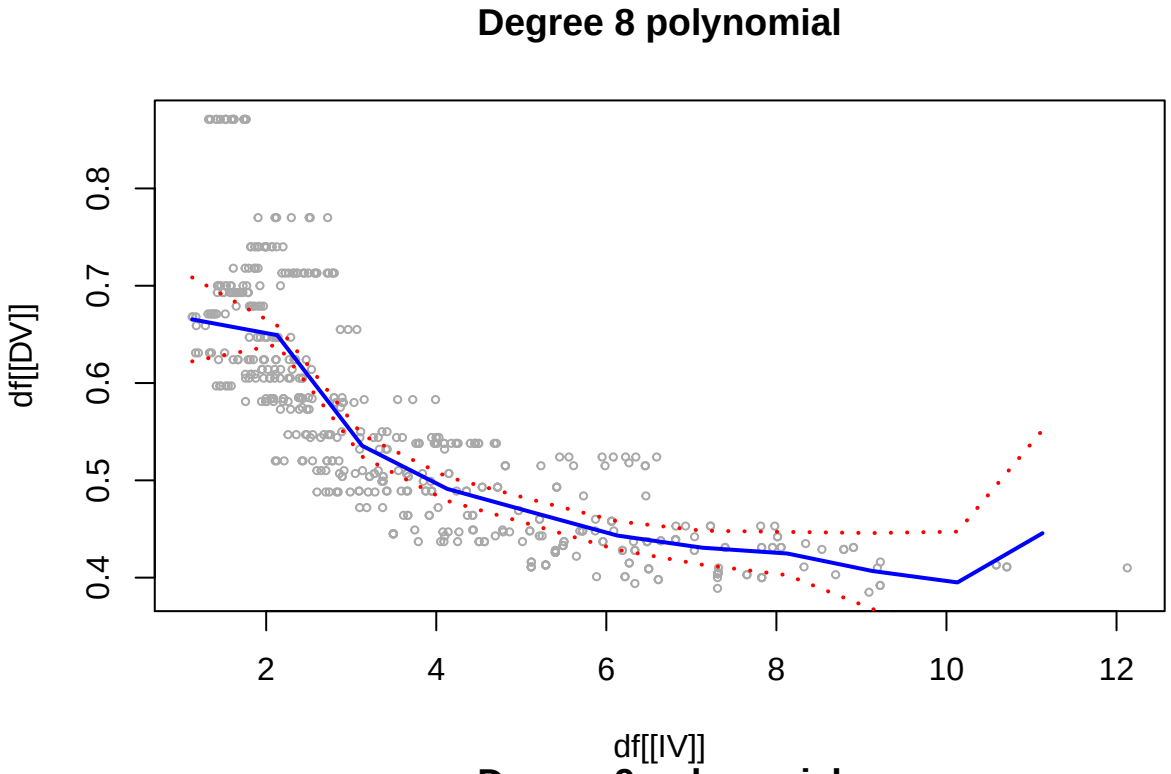

Degree 3 polynomial
df[[DV]]
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
df[[IV]]

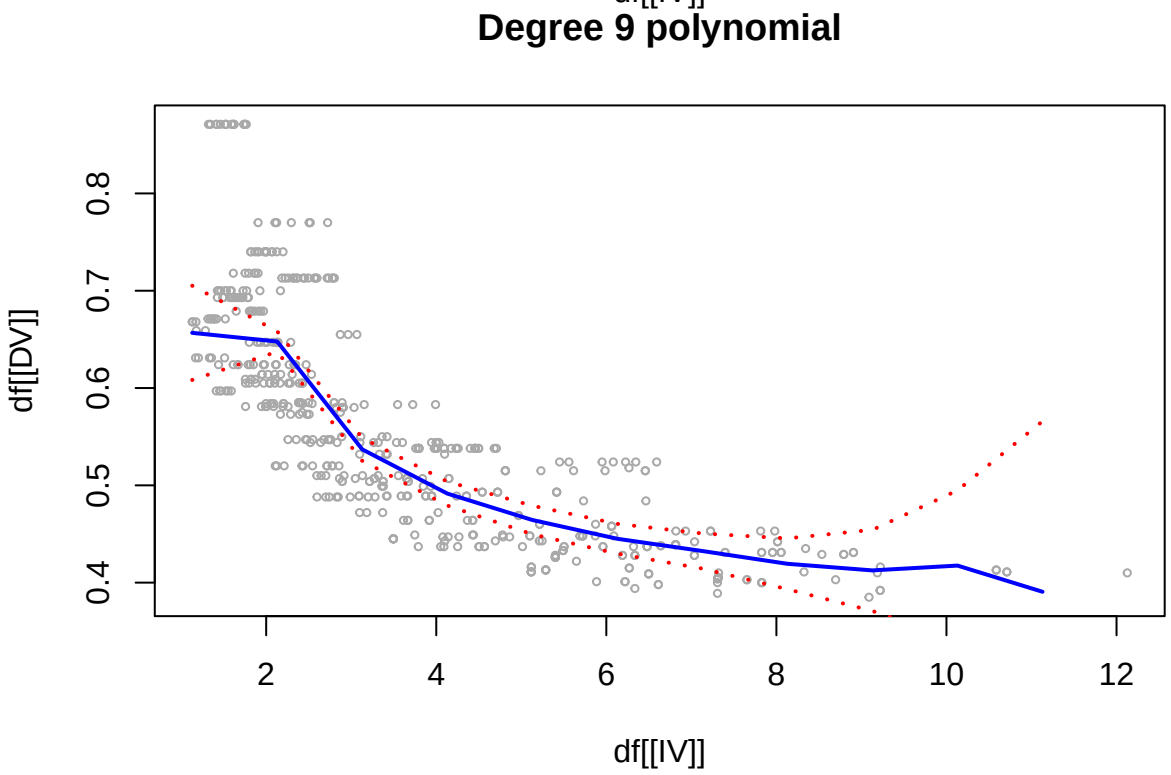

Degree 4 polynomial
df[[DV]]
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
df[[IV]]

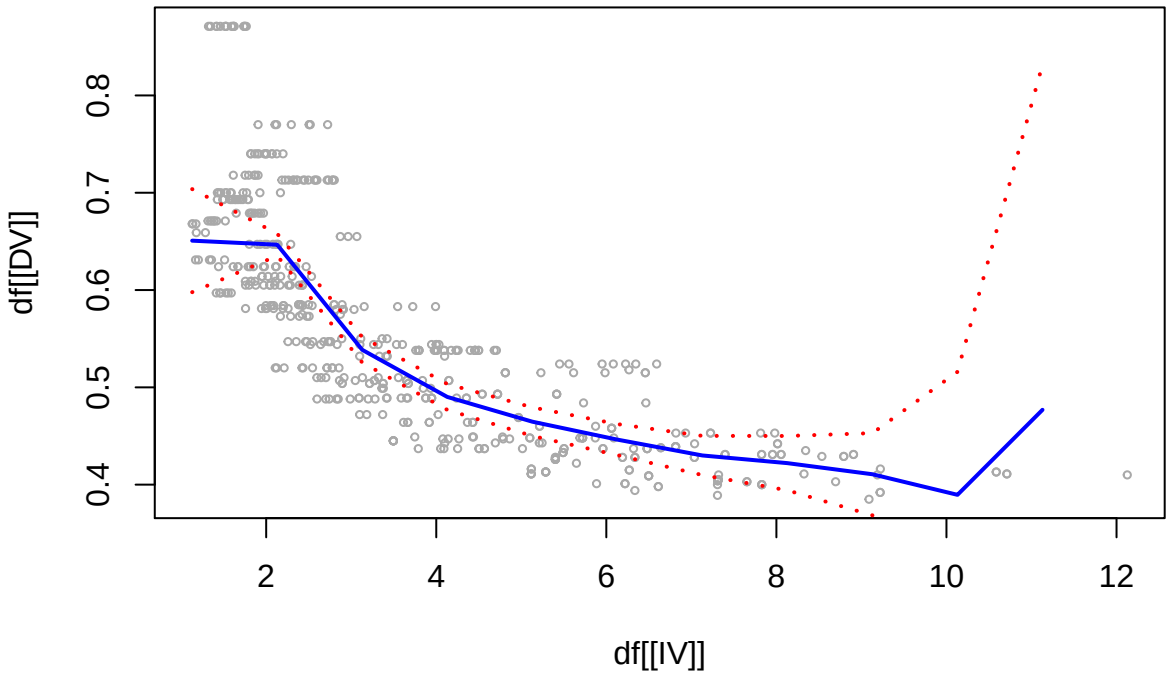

Degree 5 polynomial
df[[DV]]
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
df[[IV]]


Degree 6 polynomial
df[[DV]]
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
df[[IV]]


Degree 7 polynomial
df[[DV]]
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
df[[IV]]


Degree 8 polynomial
df[[DV]]
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
df[[IV]]

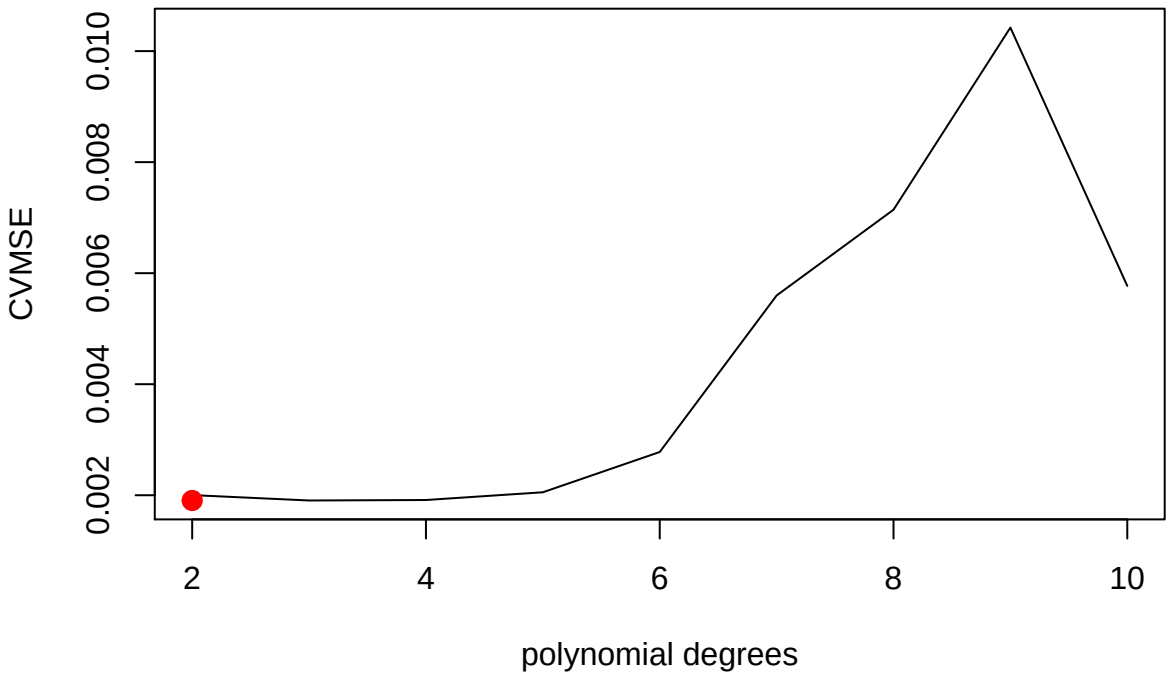

Degree 9 polynomial
df[[DV]]
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
df[[IV]]




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

```
# c) perform cross validation to find optimal degree for cross validation

set.seed(1113)
k <- 10
degrees <- 2:10

# bootstrap resampling
row.indices <- sample(1:k, nrow(boston.df), replace = T)

results <- tibble(degrees=NULL, cv.mse=NULL)

poly.cv <- function(df, DV, IV, row.indices, degree){
  test.mses <- double(k)
  for(test.index in 1:k){
    frm <- formula(glue::glue(DV , "~" , "poly(",IV,",",degree,")"))
    train.model <- glm(frm, data = df[row.indices != test.index,])

    # now predict it on test rows
    new.data <- tibble(x = df[row.indices == test.index,][[IV]])
    names(new.data) <- IV
    predicts <- predict(train.model, newdata = new.data, se=T)
    test.mses <- c(test.mses,
                   mean((df[row.indices == test.index,][[DV]] - predicts$fit)^2))
  }
  tibble(degrees=degree, cv.mse=mean(test.mses))
}
```

```
results <-
  degrees %>%
  reduce(~rbind(.x,poly.cv(boston.df, "nox", "dis",row.indices, .y)),
                   .init = results)

# plot the min degree
plot(results, xlab = "polynomial degrees", ylab="CVMSE",type = "l")

index <- which.min(results$cv.mse)
points(index, results$cv.mse[index], col="red", cex=2, pch=20)
```



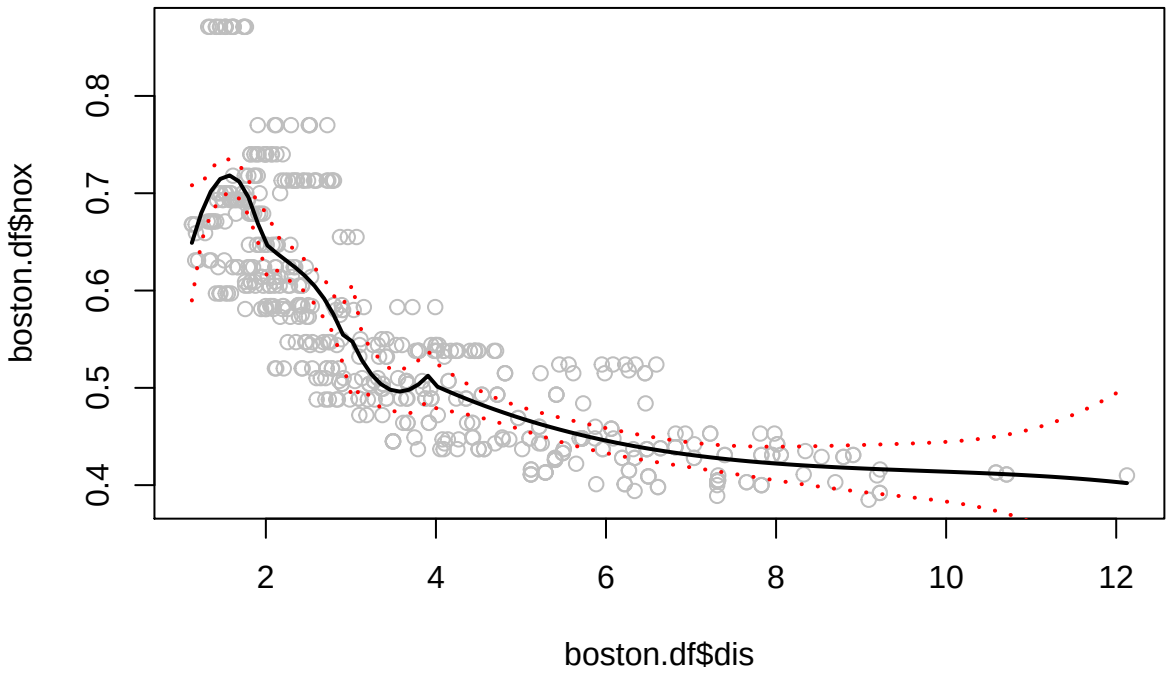

```
# so degree 2 is most optimized degree for the polynomial based on CV


#d) use bs() to fit regression spline to predict nox using dis using 4 degrees of freedom
k <- 4

fit <- glm(nox~bs(dis, knots = cut(dis, k)) ,data=boston.df)

# first get the range of the values of dis
(dis.limits  <- range(boston.df$dis))
```

```
## [1]  1.1296 12.1265
```

```
# Now create an interaval from these range values
dis.grid <- seq(from=dis.limits[1], to=dis.limits[2], length.out = 100)

# predict the nox values
pred <- predict(fit, newdata = list(dis=dis.grid), se=T)
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = residual.scale, type = if
## (type == : prediction from a rank-deficient fit may be misleading
```

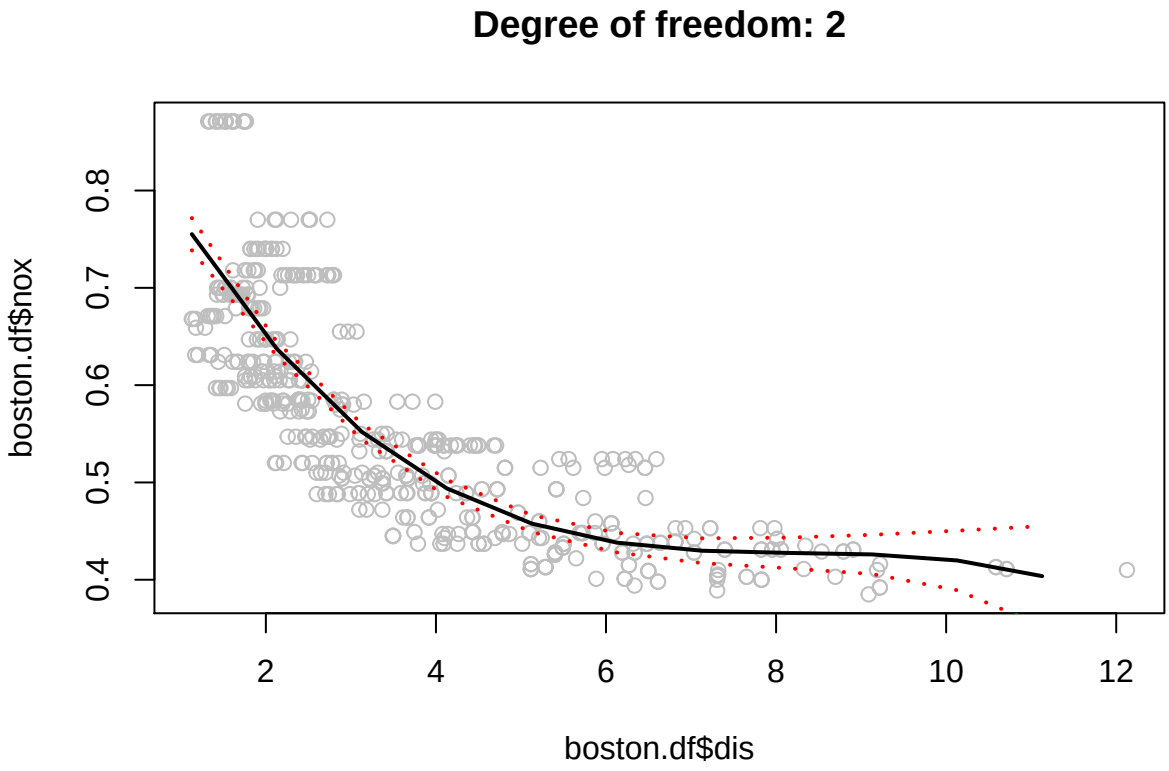
```r
plot(boston.df$dis, boston.df$nox, col="gray")
lines(dis.grid, pred$fit, lwd=2)
se_bonds <- cbind(pred$fit+2*pred$se, pred$fit-2*pred$se)
matlines(dis.grid, se_bonds, lwd=2, col="red", lty=3)
```




```r
# knots are chosen uniformly
# Since Boston DF is too small , only 3 degree os freedom is chosen by R

# e) fit a regression splines with range of degrees of freedom and reort resulting RSS
regression.spline <- function(k){
  fit <- glm(nox~bs(dis, df = k) ,data=boston.df)

  # first get the range of the values of dis
  (dis.limits  <- range(boston.df$dis))

  # Now create an interaval from these range values
  (dis.grid <- seq(from=dis.limits[1], to=dis.limits[2]))

  # predict the nox values
  pred <- predict(fit, newdata = list(dis=dis.grid), se=T)
  plot(boston.df$dis, boston.df$nox, col="gray")
  title(glue::glue("Degree of freedom: ", k), outer = F)
  lines(dis.grid, pred$fit, lwd=2)
  se_bonds <- cbind(pred$fit+2*pred$se, pred$fit-2*pred$se)
  matlines(dis.grid, se_bonds, lwd=2, col="red", lty=3)

  rse <- residuals(fit, type = "response")

  # calculate RSS
  res %>% reduce( ~.x + .y^2 , .init = 0) %>% unlist

}
```

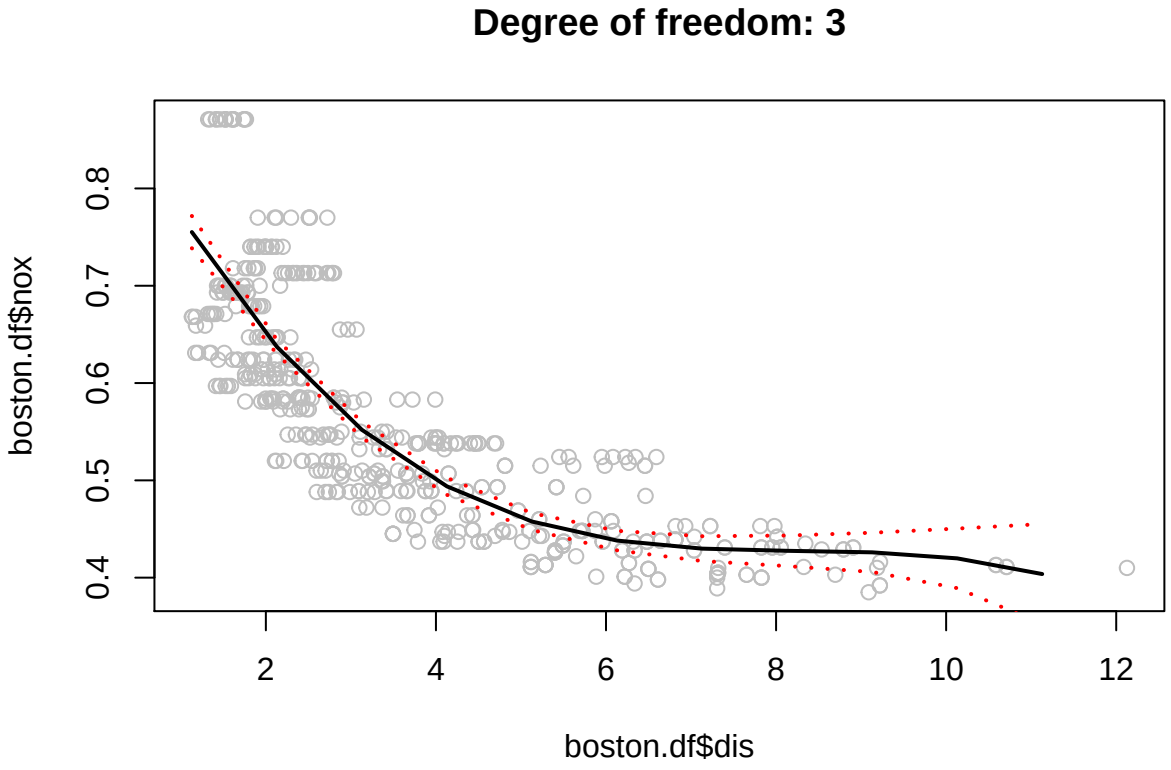
```
regression.spline(2)
```

```
## Warning in bs(dis, df = k): 'df' was too small; have used 3
```



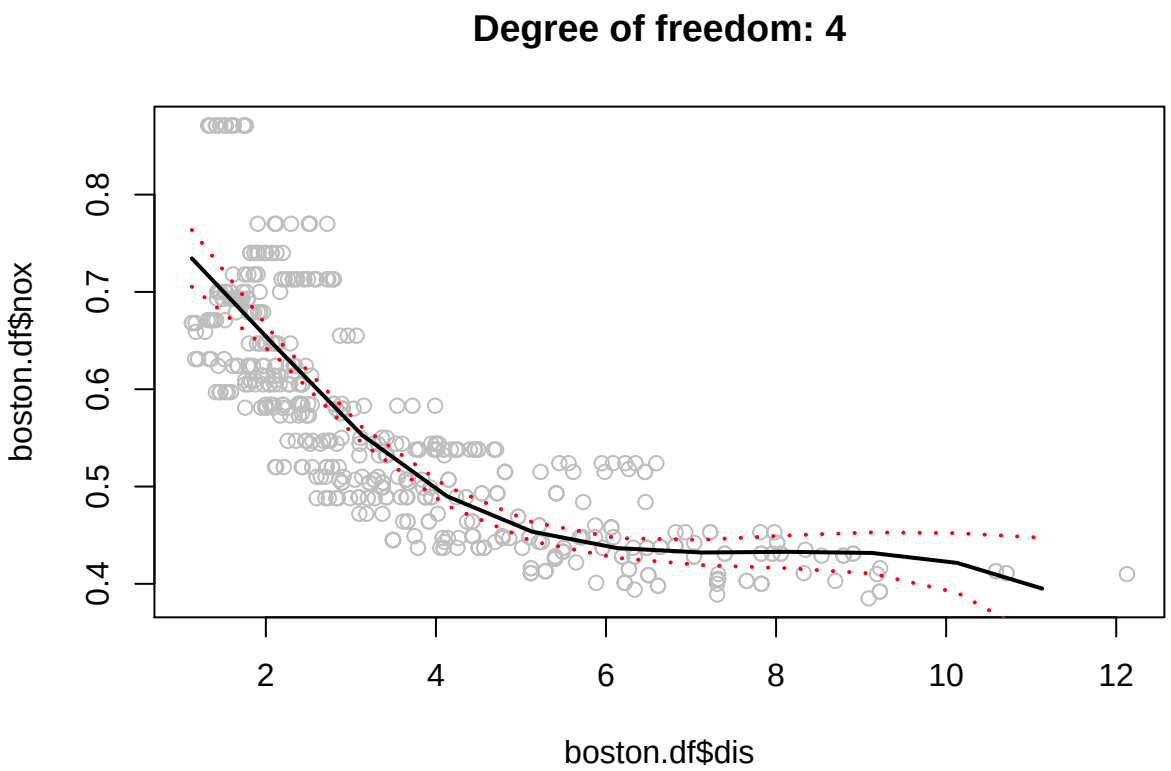

```
## [1] 1.934107
```

```
regression.spline(3)
```




```
## [1] 1.934107
```

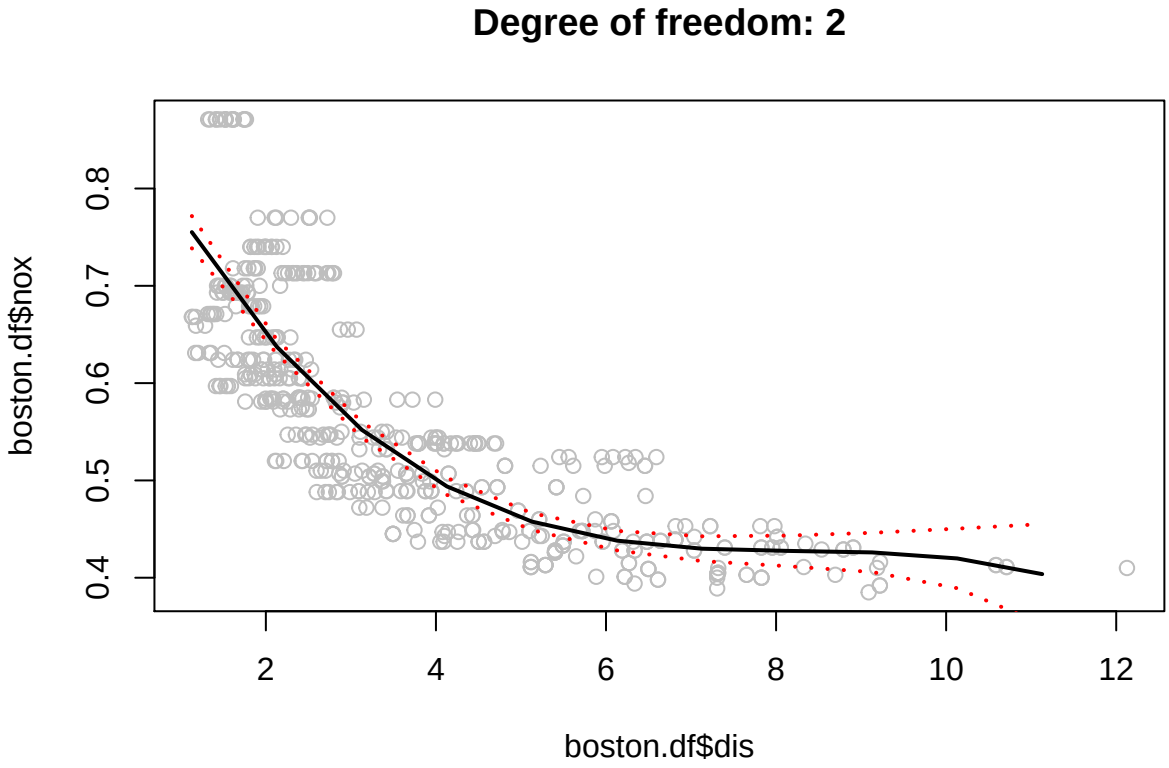
```
regression.spline(4)
```

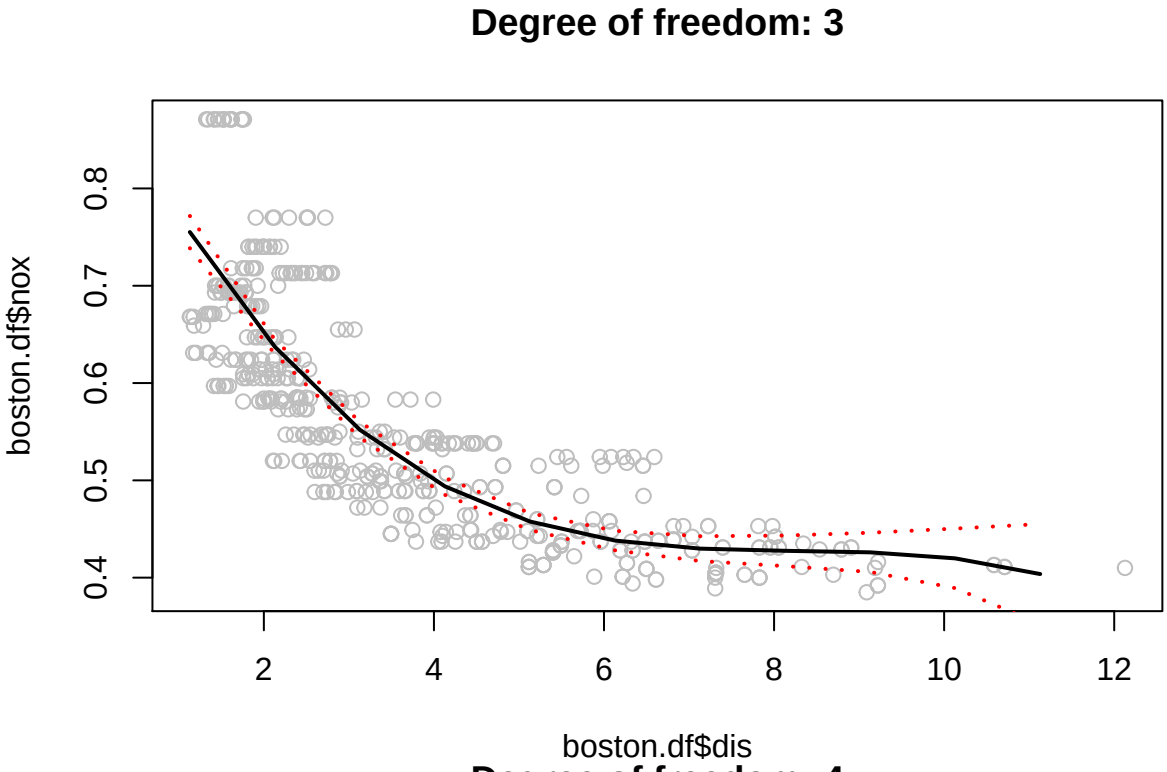


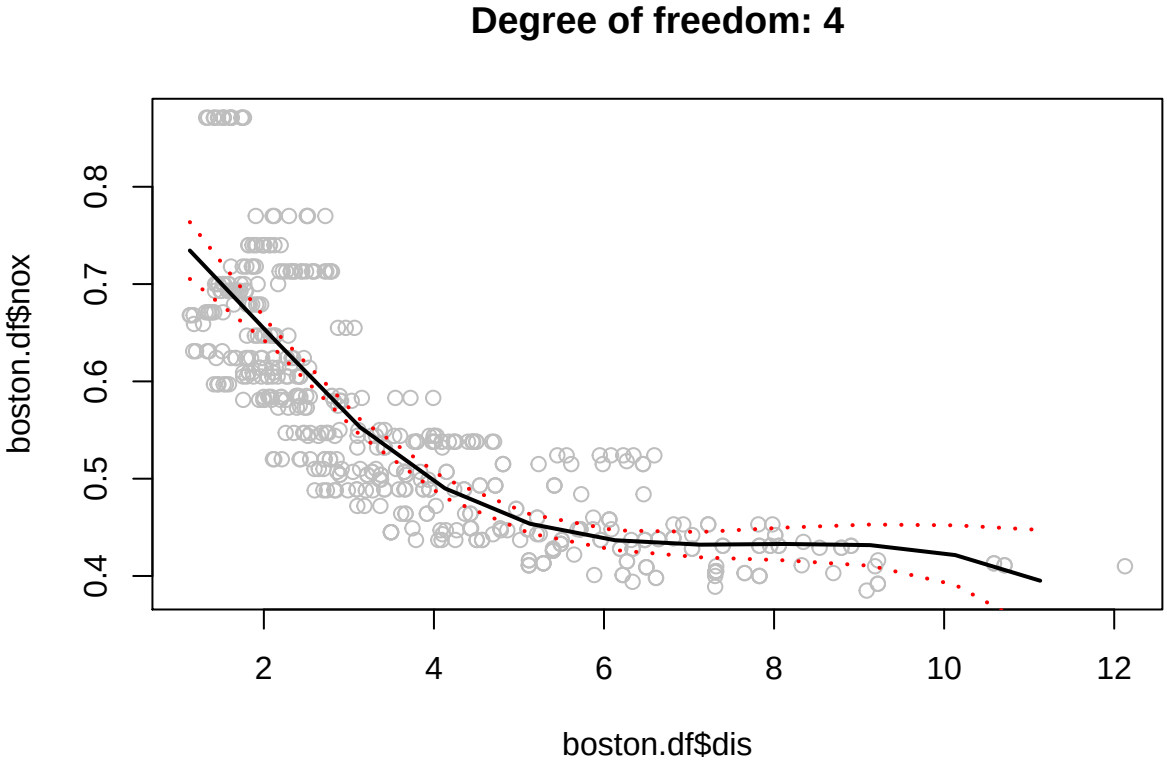

```
## [1] 1.934107
```

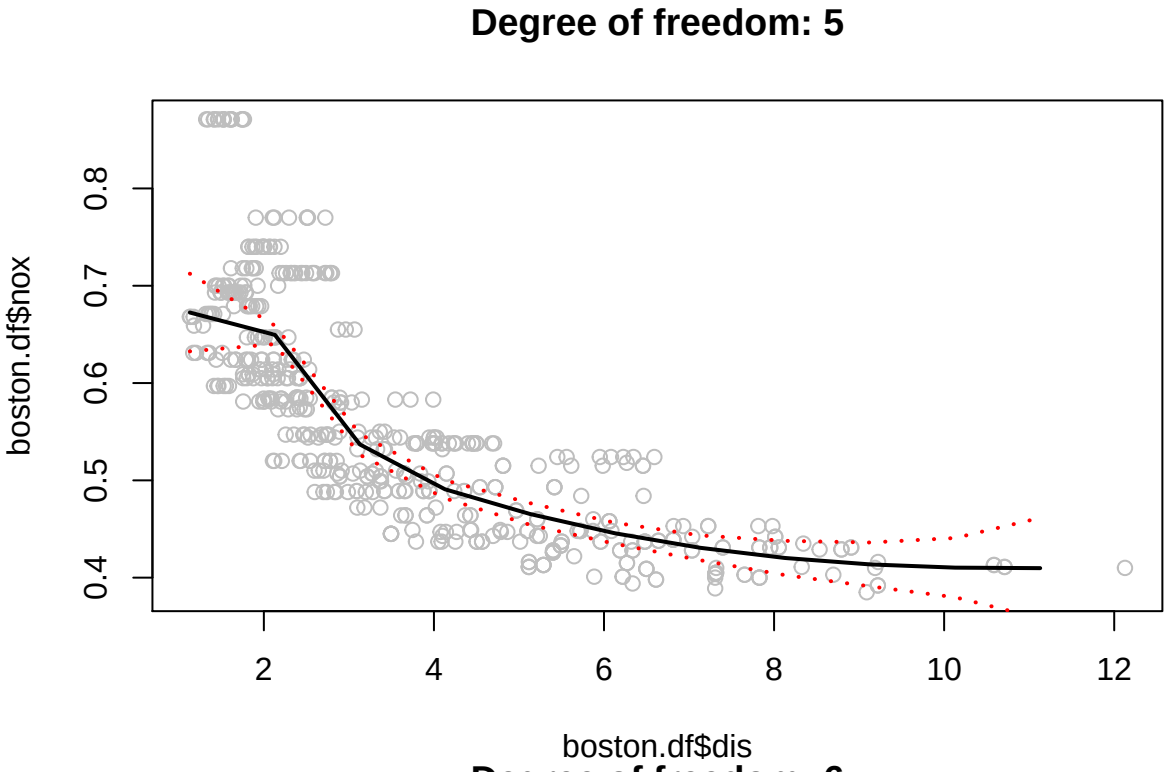
```
results <- tibble(degree.of.freedom = NULL, rse=NULL)
results <- 2:10 %>%
  reduce(~rbind(.x, tibble(degree.of.freedom = .y ,
                           rse = regression.spline(.y))),
         .init=results)
```

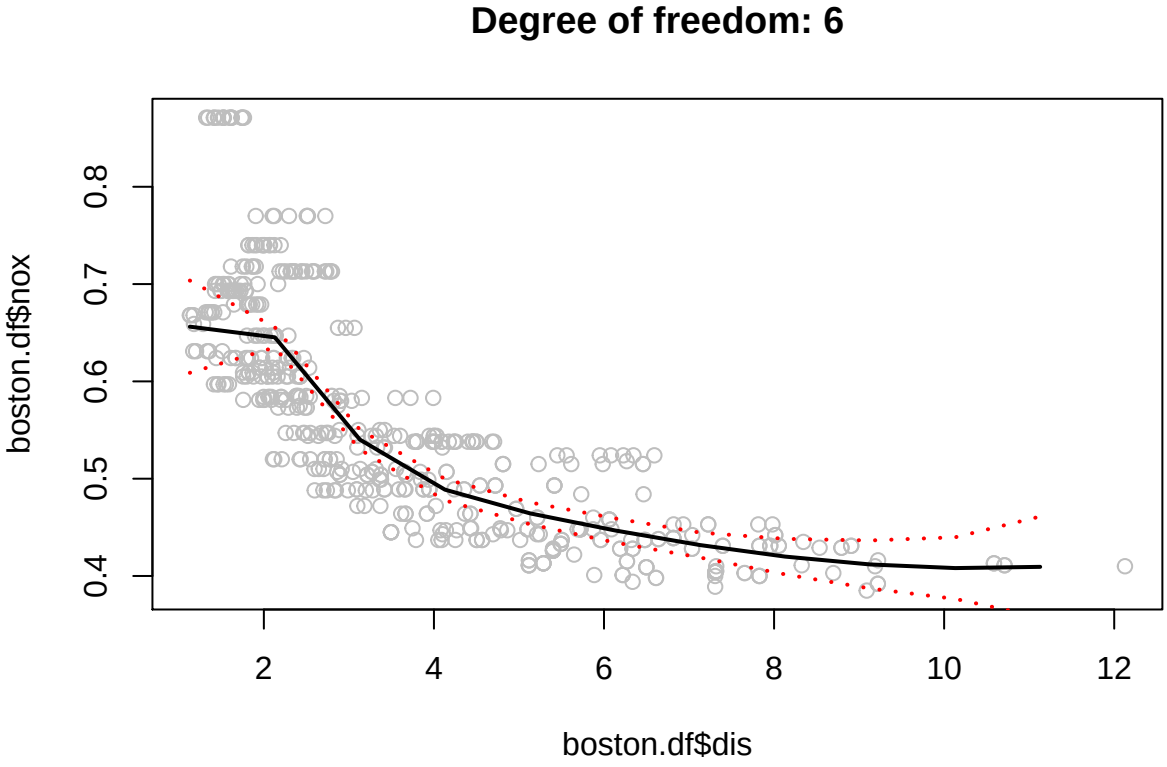
```
## Warning in bs(dis, df = k): 'df' was too small; have used 3
```



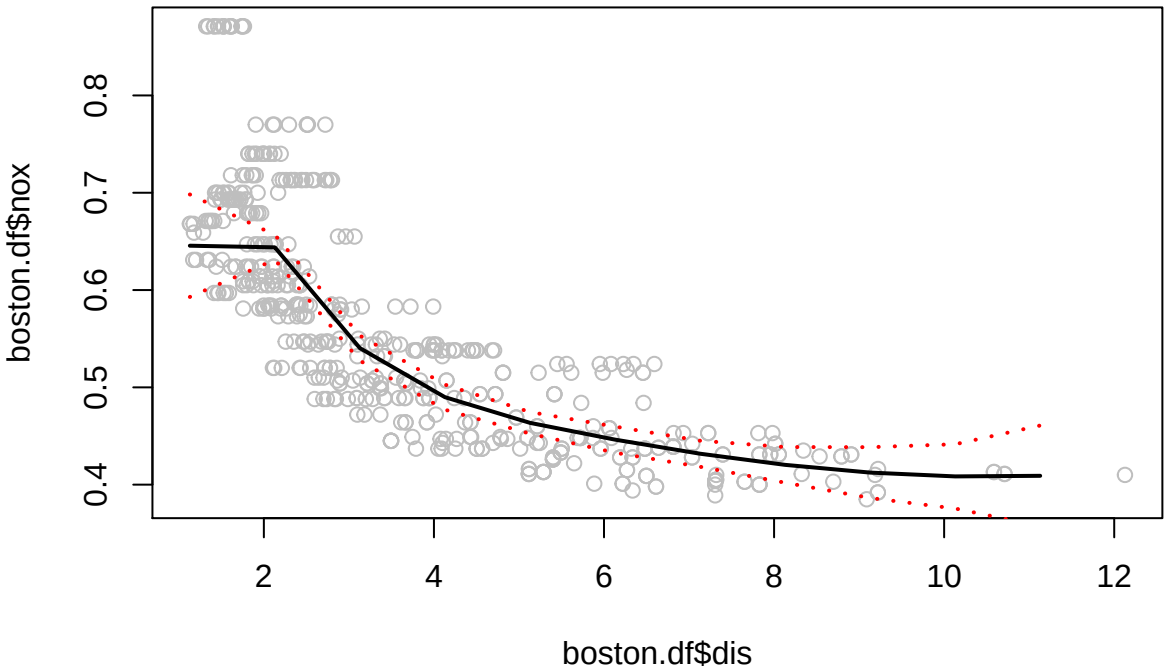

Degree of freedom: 3
boston.df$nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
boston.df$dis


Degree of freedom: 4
boston.df$nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
boston.df$dis

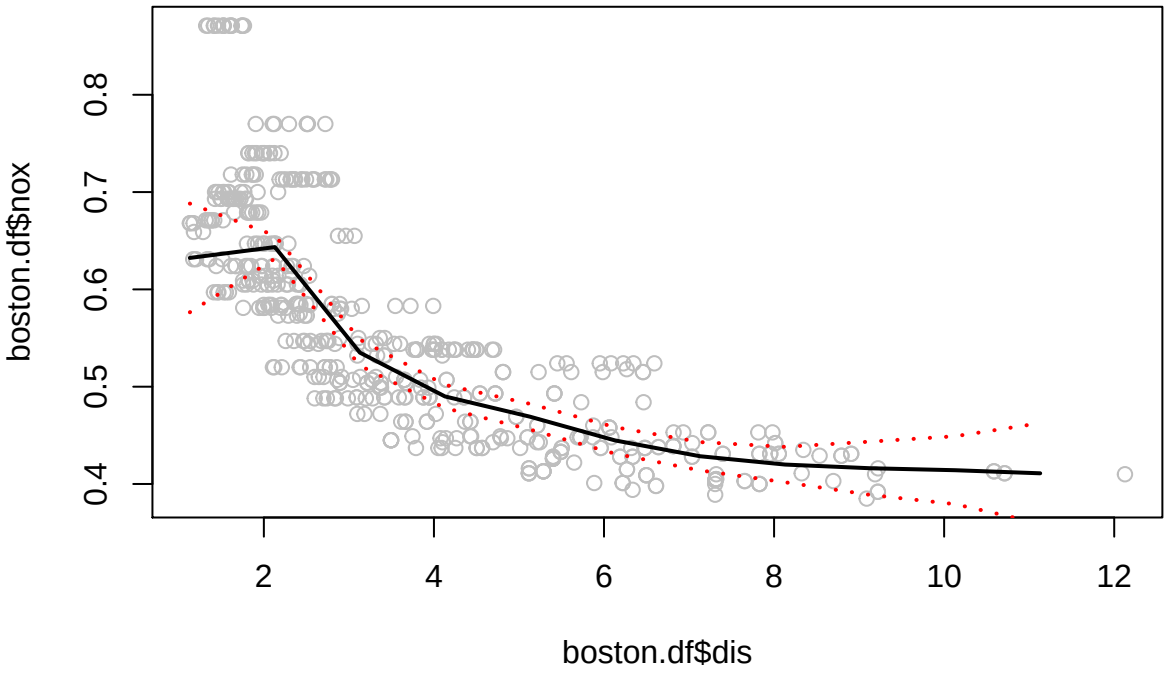

Degree of freedom: 5
boston.df$nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
boston.df$dis

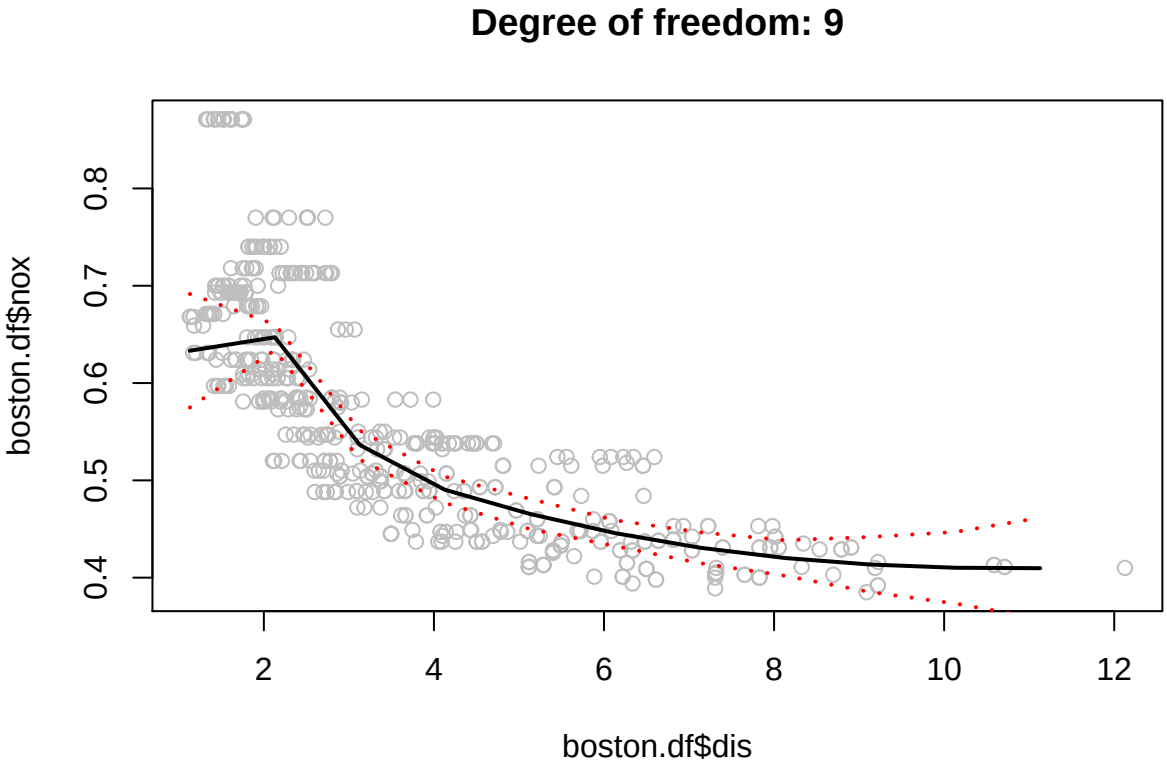

Degree of freedom: 6
boston.df$nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
boston.df$dis

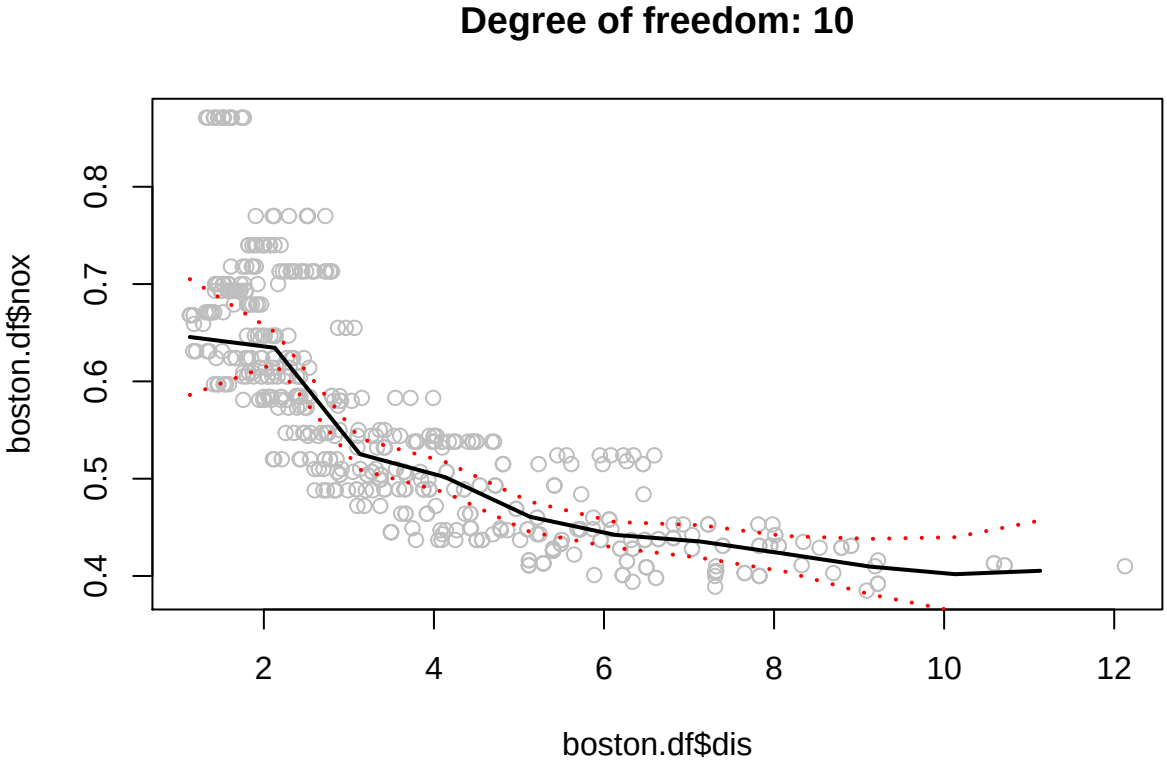
Degree of freedom: 7



Degree of freedom: 8








```
results
```

```
## # A tibble: 9 x 2
##   degree.of.freedom   rse
##               <int> <dbl>
## 1                 2  1.93
```

```
## 2                3  1.93
## 3                4  1.93
## 4                5  1.93
## 5                6  1.93
## 6                7  1.93
## 7                8  1.93
## 8                9  1.93
## 9               10  1.93
```

```
# Since Boston DF is too small , only 3 degree os freedom is chosen by R

# f) Perform cross validation to select the best fegree of freedom
 # since number of data is not enough , matrix of the model will have
 # rank defficiency do cross validation would give the same result.
```

```
library(leaps)
library(tidyverse)
library(dataPreparation)
library(mgcv)

# first define some helpers
# remove empty characters and NA helper
remove.empty.characters <- function(df)
  df %>%
    select_all %>%
    filter_if(is.character, any_vars(!is.na(.) & trimws(.) != ""))

# Next remove leading and trailing spaces from all elements in character columns
trim.f <- function(col, na.rm = F) {
  isNA <- !reduce(col, ~ (is.na(.x) & is.na(.y)))
  if (na.rm && isNA)
    unlist(map(col, ~ (if (is.na(.x)) "" else .x) ),use.names = F)
  else trimws(col, which = c("both")) # leading and trailing spaces
}

trim.spaces <- function(df)
  df %>%
    mutate_if(is.character, trim.f, na.rm = T)

# Finally convert character columns to factor

char.to.fctor <- function(df)
  df %>%
    mutate_if(is.character, ~ factor(.x, levels = (.x %>% table() %>% names())))

# ------------  read the data ----------------

college.df = read.csv(
  "/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/College.csv",
  header=T, stringsAsFactors = F, na.strings = "?")

college.df <- tibble(college.df)

# a) First and foremost lets split data to 80% train and the rest test
```

```r
set.seed(1113)
train.idx <- sample(1:nrow(college.df), 0.8*nrow(college.df))
test.idx <- setdiff(1:nrow(college.df), train.idx)

train.df <- college.df[train.idx, ]
test.df <- college.df[test.idx, ]

# --------------- prepare train data ----------------------
# see if there is any NA in any records
nrow(df[which(is.na(train.df)),])
```

```
## [1] 0
```

```r
# No NA anywhere , let's do usual clean up:

# remove constant variables
constant_cols <- whichAreConstant(train.df)
```

```
## [1] "whichAreConstant: it took me 0s to identify 0 constant column(s)"
```

```r
# remove Variables that are in double (for example col1 == col2)
(double_cols <- whichAreInDouble(train.df))
```

```
## [1] "whichAreInDouble: it took me 0.01s to identify 0 column(s) to drop."
```

```
## integer(0)
```

```r
# remove Variables that are exact bijections (for example col1 = A, B, B, A and col2 = 1, 2, 2, 1)
(bijections_cols <- whichAreBijection(train.df))
```

```
## [1] "whichAreBijection: it took me 0.15s to identify 0 column(s) to drop."
```

```
## integer(0)
```

```r
# It is safe now to convert all character fields into factors

train.df <-
  train.df %>%
  na.omit() %>%
  trim.spaces() %>%
  remove.empty.characters() %>%
  char.to.fctor()

# --------------- prepare test data ----------------------

# see if there is any NA in any records
nrow(df[which(is.na(test.df)),])
```

```
## [1] 0
```

```r
# No NA anywhere , let's do usual clean up:

# remove constant variables
constant_cols <- whichAreConstant(test.df)
```

```
## [1] "whichAreConstant: it took me 0s to identify 0 constant column(s)"
```

```r
# remove Variables that are in double (for example col1 == col2)
(double_cols <- whichAreInDouble(test.df))
```

```
## [1] "whichAreInDouble: it took me 0s to identify 0 column(s) to drop."
```

```
## integer(0)
```

```
# remove Variables that are exact bijections (for example col1 = A, B, B, A and col2 = 1, 2, 2, 1)
(bijections_cols <- whichAreBijection(test.df))
```

```
## [1] "whichAreBijection: it took me 0.12s to identify 0 column(s) to drop."
```

```
## integer(0)
```

```
# It is safe now to convert all character fields into factors

test.df <-
  test.df %>%
  na.omit() %>%
  trim.spaces() %>%
  remove.empty.characters() %>%
  char.to.fctor()


# ---------- stepwise forward feature selectin with CV -----------#

k.fold <- 10

set.seed(1113)

# create k folds
folds <- sample(1:k.fold, nrow(train.df), replace = T)

# number of features
noOfFeatures <- ncol(train.df) -1


cv.errors <- matrix(NA, k.fold, noOfFeatures,
                    dimnames = list(NULL, paste(1:noOfFeatures)))


# perform a cross validation on a for loop
for (j in 1:k.fold){
  # step# 2 of algorithm 6.2 page 207 is evaluated on all folds except one of
  # them each time it chooses best models with number of features
  # 1,2,..., noOfFeatures on k-1 training folds.
  best.fit <- regsubsets(Outstate ~ ., data = train.df[folds != j, ],
                         nvmax = noOfFeatures, method = "forward")

  # now compute CV test error for each of  models that have best number of
  # predictors on test fold # j
  for(i in 1:noOfFeatures){
    # extract coefficients for model # i
    coefi <- coef(best.fit, id = i)

    # For GAM we are interested in name
    # of features for model # i (not their coefficients)
    # except intercept
    col.names <- names(coefi)[-1]
```

```
    # first separate factors , then remove "Yes" from the end of factor name and
    #finally drop "(Intercept)"

    factor.post.script <- "Yes"
    intercept.name <- "(Intercept)"

    regular.features <- col.names %>%
      keep(!str_detect(., factor.post.script)) %>%
      discard(str_detect(., intercept.name))

    factor.features <- col.names %>%
        keep(str_detect(., factor.post.script)) %>%
        str_replace(factor.post.script, "")

    # now dynamically build the formula

    k = 10
    bs = "\"cr\""

    regular.part <- regular.features %>%
      map(~glue::glue("s(",.,", bs=",bs,","," k=",k,")")) %>%
      str_c(., collapse = " + ")

    factor.part <- factor.features %>% str_c(., collapse = " + ")

    formula <-
      case_when(
        factor.part != "" & regular.part != "" ~ glue::glue("Outstate ~ " ,
                                                            regular.part ,
                                                            " + ",
                                                            factor.part),

        factor.part != "" & regular.part == "" ~ glue::glue("Outstate ~ " ,
                                                            factor.part),

        factor.part == "" & regular.part != "" ~ glue::glue("Outstate ~ " ,
                                                            regular.part)
      )


    model.fit <- gam(as.formula(formula),method="GACV.Cp", scale=-1,
                                family = Gamma(link=log),
                                data=train.df[folds != j, ], gamma = 1)

#    apply model on test fold and accumulate the test errors
    preds <- predict(model.fit, newdata = train.df[folds == j, ], se=TRUE)
    cv.errors[j,i] <- mean ((preds$fit - train.df[folds == j, ]$Outstate)^2)

  }
}

# finally calculate mean of CV MSE error for each model
cv.error.means <- rep(NA, ncol(cv.errors))
```

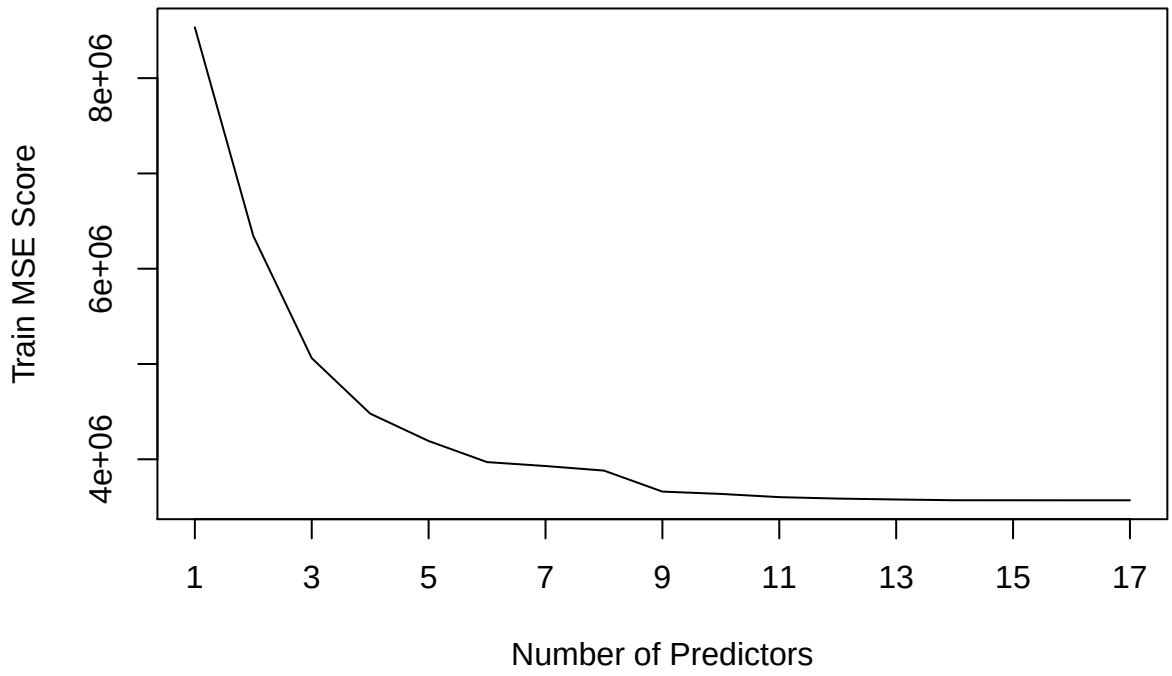
```
for (l in 1:ncol(cv.errors)){
  cv.error.means[l] <- mean(cv.errors[ ,l])
}


print("most optimal No. of selected covariates")
```

```
## [1] "most optimal No. of selected covariates"
```

```
(covariates.selected <- which.min(cv.error.means))
```

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

```
gam.best.fit <- regsubsets(Outstate ~ ., data = train.df,
                           nvmax = ncol(train.df), method = "forward")

plot(1/nrow(train.df)*summary(gam.best.fit)$rss,type='l',xlab='Number of Predictors',ylab='Train MSE Sc
axis(side=1,at=seq(1,17,2),labels = seq(1,17,2))
```




```
print("Seems like 7 is the good choice selected covariates:")
```

```
## [1] "Seems like 7 is the good choice selected covariates:"
```

```
which(summary(gam.best.fit)$which[7,-1])
```

```
##  PrivateYes  Room.Board    Personal    Terminal perc.alumni      Expend
##           1           9          11          13          15          16
##   Grad.Rate
##          17
```

```
(covariates.selected.names <- names(coef(gam.best.fit, id = 7)))
```

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

```
# "Private", "Room.Board", "Personal", "Terminal", "perc.alumni", "Expend", "Grad.Rate"


# b) ---------------- Apply GAM ----------------------

# first separate factors , then remove "Yes" ate the end of factor name and finally drop intercept
factor.post.script <- "Yes"
intercept.name <- "(Intercept)"

regular.features <- covariates.selected.names %>%
  keep(!str_detect(., factor.post.script)) %>%
  discard(str_detect(., intercept.name))

factor.features <- covariates.selected.names %>%
    keep(str_detect(., factor.post.script)) %>%
    str_replace(factor.post.script, "")

# now dynamically build the formula

k = 20
bs = "\"cr\""

# regular.part <- regular.features %>%
#   map(~glue::glue("s(",.,")")) %>%
#   str_c(., collapse = " + ")


regular.part <- regular.features %>%
  map(~glue::glue("s(",.,", bs=",bs,","," k=",k,")")) %>%
  str_c(., collapse = " + ")

factor.part <- factor.features %>% str_c(., collapse = " + ")

formula <-
  case_when(
    factor.part != "" & regular.part != "" ~ glue::glue("Outstate ~ " ,
                                                        regular.part ,
                                                        " + ",
                                                        factor.part),

    factor.part != "" & regular.part == "" ~ glue::glue("Outstate ~ " ,
                                                        factor.part),

    factor.part == "" & regular.part != "" ~ glue::glue("Outstate ~ " ,
                                                        regular.part)
  )

# "Private", "Room.Board", "Personal", "Terminal", "perc.alumni", "Expend", "Grad.Rate"


print ("-------------- GAM -----------------")
```

```
## [1] "-------------- GAM -----------------"
```

```
best.model.fit <- gam(as.formula(formula),method="GACV.Cp", scale=-1,
                            family = Gamma(link=log),
                            data=train.df, gamma = 1.5)
```

```
## Warning in newton(lsp = lsp, X = G$X, y = G$y, Eb = G$Eb, UrS = G$UrS, L =
## G$L, : Fitting terminated with step failure - check results carefully
```

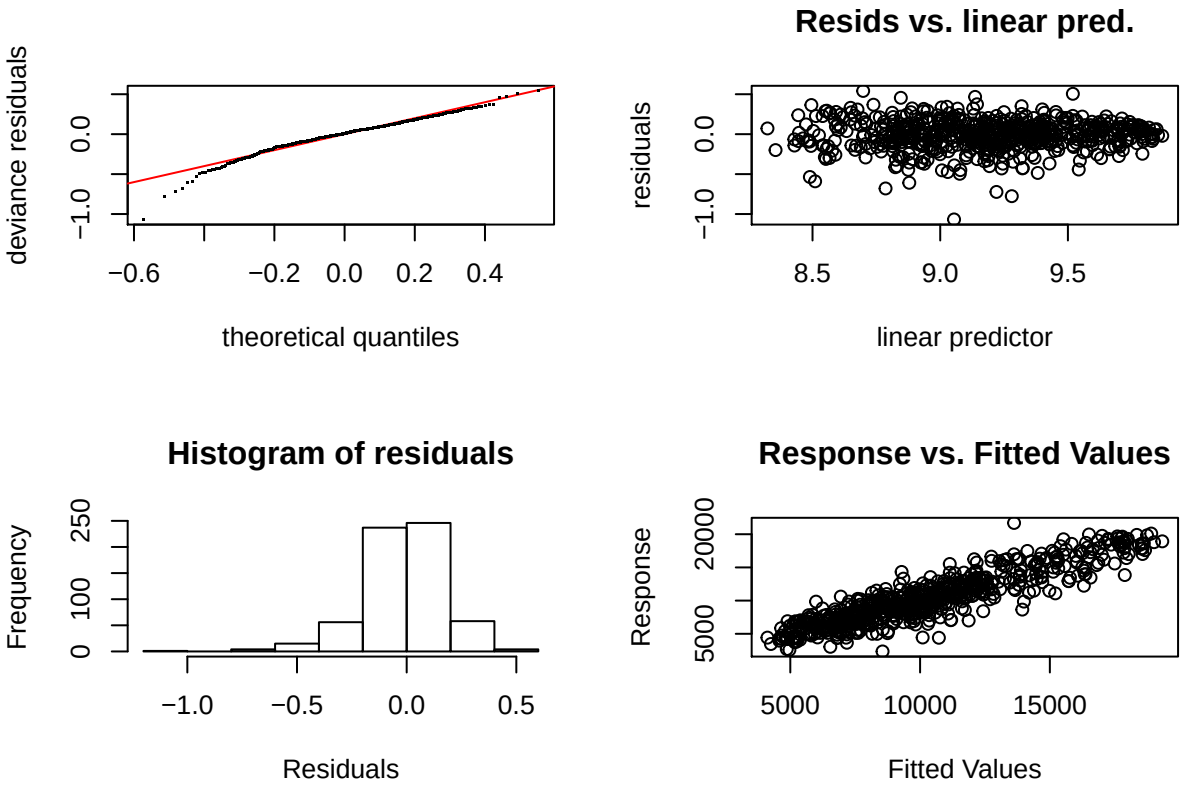
```
print.gam(best.model.fit)
```

```
##
## Family: Gamma
## Link function: log
##
## Formula:
## Outstate ~ s(Room.Board, bs = "cr", k = 20) + s(Personal, bs = "cr",
##     k = 20) + s(Terminal, bs = "cr", k = 20) + s(perc.alumni,
##     bs = "cr", k = 20) + s(Expend, bs = "cr", k = 20) + s(Grad.Rate,
##     bs = "cr", k = 20) + Private
##
## Estimated degrees of freedom:
##  7.07  2.13  3.39  2.64 10.41  4.55  total = 32.18
##
## GACV score: 0.03810709
```

```
print ("-------------- model checking -----------------")
```

```
## [1] "-------------- model checking -----------------"
```

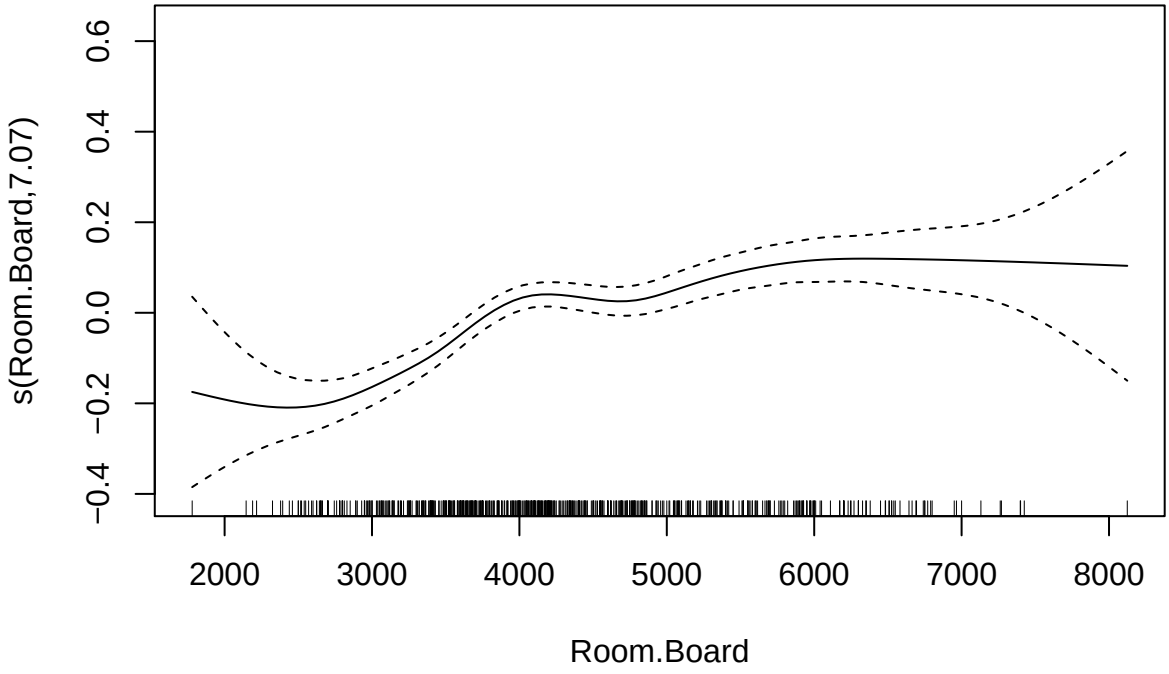
```
gam.check(best.model.fit)
```

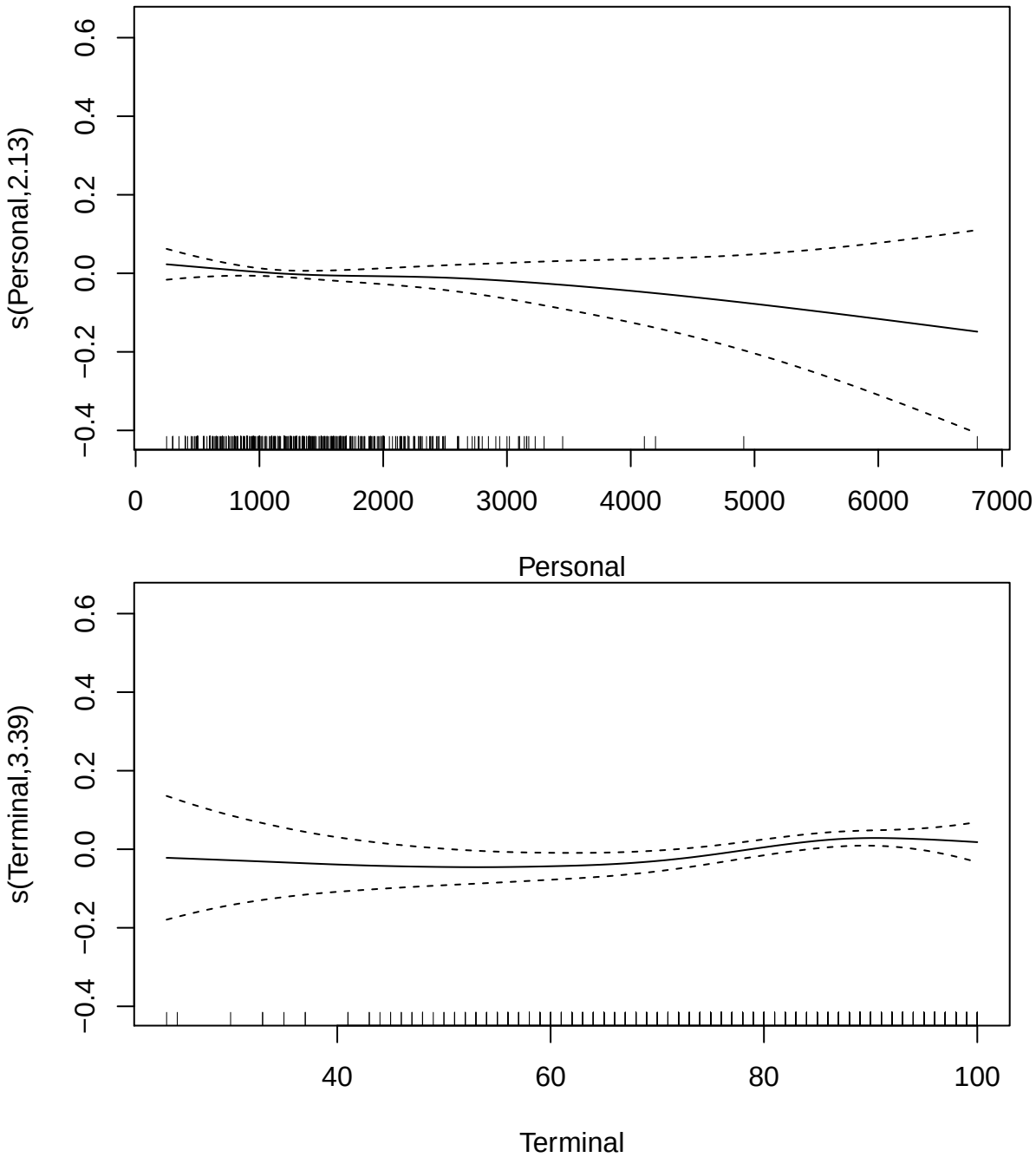


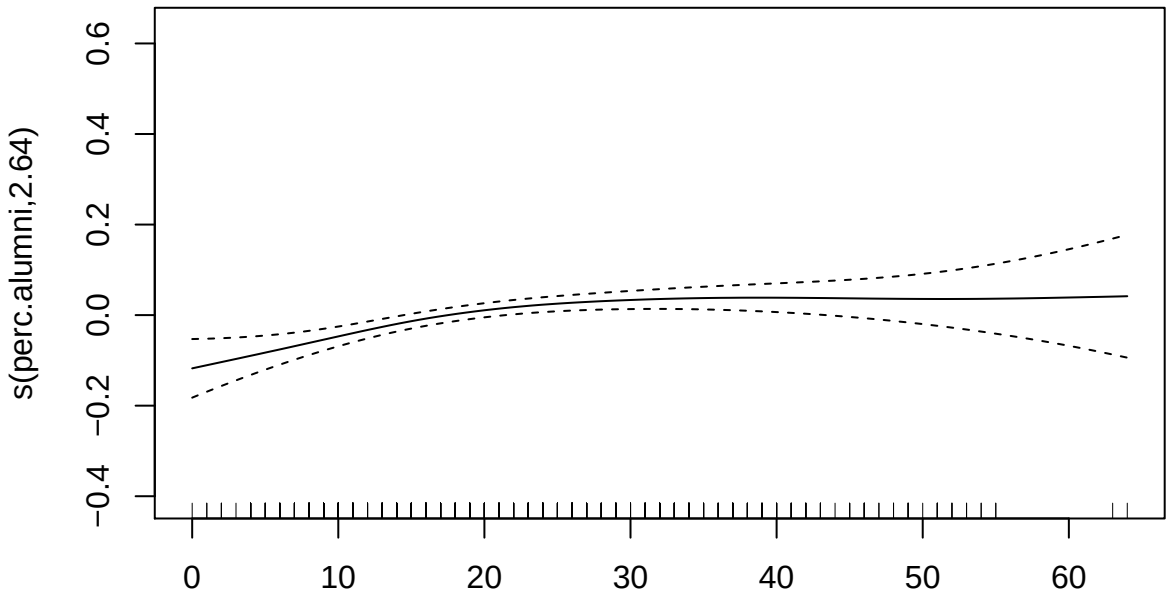

```
##
```

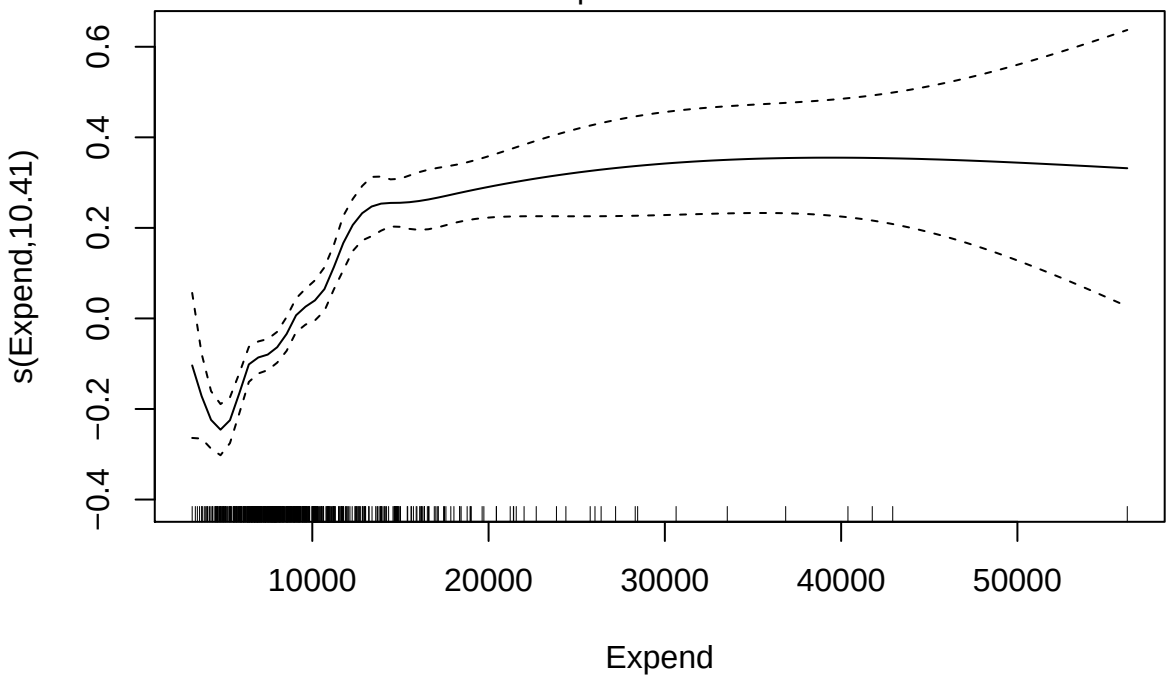
```
## Method: GACV   Optimizer: outer newton
## step failed after 8 iterations.
## Gradient range [-2.293557e-05,4.3548e-05]
## (score 0.03810709 & scale 0.03174157).
## Hessian positive definite, eigenvalue range [1.016465e-05,9.628196e-05].
## Model rank =  116 / 116
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                   k'   edf k-index p-value
## s(Room.Board)  19.00  7.07    0.95    0.10
## s(Personal)    19.00  2.13    1.00    0.53
## s(Terminal)    19.00  3.39    1.02    0.67
## s(perc.alumni) 19.00  2.64    0.99    0.42
## s(Expend)      19.00 10.41    1.01    0.65
## s(Grad.Rate)   19.00  4.55    0.99    0.39
```

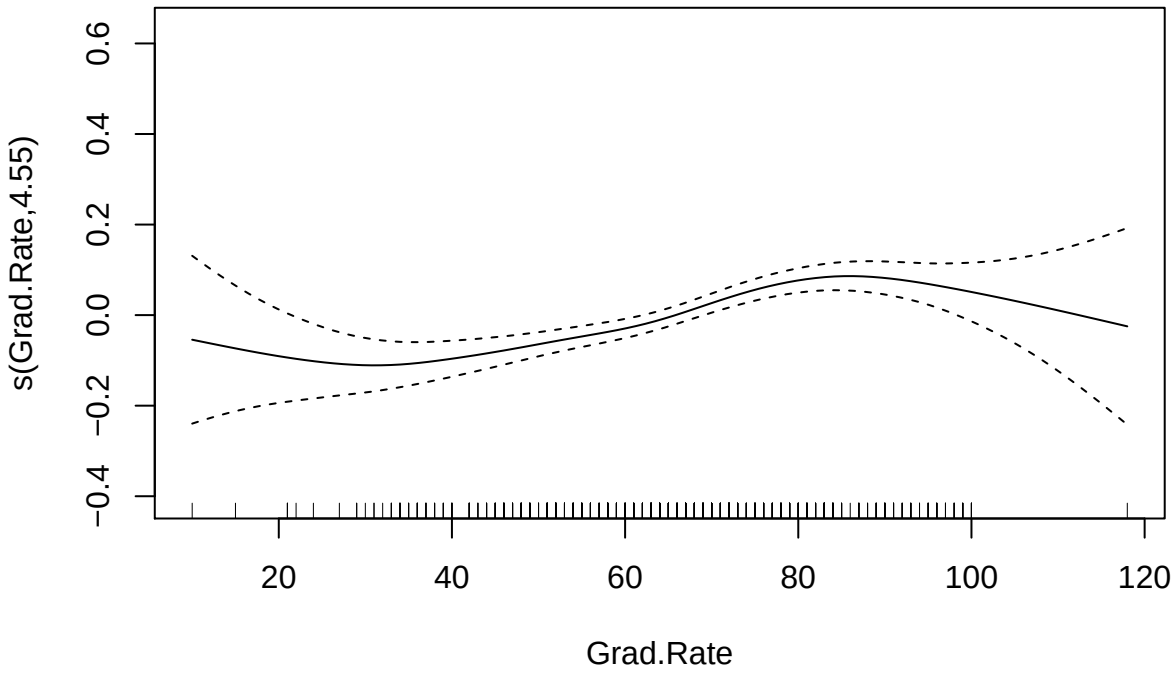
```
# finally draw the plots
plot.gam(best.model.fit)
```




s(Personal,2.13)
0.6
0.4
0.2
0.0
−0.2
−0.4
0
1000
2000
3000
4000
5000
6000
7000
Personal
s(Terminal,3.39)
0.6
0.4
0.2
0.0
−0.2
−0.4
40
60
80
100
Terminal


s(perc.alumni,2.64)
0.6
0.4
0.2
0.0
−0.2
−0.4
0
10
20
30
40
50
60
perc.alumni


s(Expend,10.41)
0.6
0.4
0.2
0.0
−0.2
−0.4
10000
20000
30000
40000
50000
Expend




```
rsd <- residuals(best.model.fit,type="deviance")
print(" ------ Check value of K is large enough for Room.Board ------------")
```

```
## [1] " ------ Check value of K is large enough for Room.Board ------------"
```

```
gam(rsd~s(Room.Board,k=20)-1,data=train.df,select=TRUE)  # -1 supresses the intercept
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## rsd ~ s(Room.Board, k = 20) - 1
##
## Estimated degrees of freedom:
## 0  total = 0
##
## GCV score: 0.03305406
```

```
print(" ------ Check value of K is large enough for Personal ------------")
```

```
## [1] " ------ Check value of K is large enough for Personal ------------"
```

```
gam(rsd~s(Personal,k=20)-1,data=train.df,select=TRUE)  # -1 supresses the intercept
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## rsd ~ s(Personal, k = 20) - 1
##
## Estimated degrees of freedom:
## 0  total = 0
##
```

```
## GCV score: 0.03305406
```

```
print(" ------ Check value of K is large enough for Terminal ------------")
```

```
## [1] " ------ Check value of K is large enough for Terminal ------------"
```

```
gam(rsd~s(Terminal,k=20)-1,data=train.df,select=TRUE)  # -1 supresses the intercept
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## rsd ~ s(Terminal, k = 20) - 1
##
## Estimated degrees of freedom:
## 0  total = 0
##
## GCV score: 0.03305406
```

```
print(" ------ Check value of K is large enough for perc.alumni ------------")
```

```
## [1] " ------ Check value of K is large enough for perc.alumni ------------"
```

```
gam(rsd~s(perc.alumni,k=20)-1,data=train.df,select=TRUE)  # -1 supresses the intercept
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## rsd ~ s(perc.alumni, k = 20) - 1
##
## Estimated degrees of freedom:
## 0  total = 0
##
## GCV score: 0.03305406
```

```
print(" ------ Check value of K is large enough for Expend ------------")
```

```
## [1] " ------ Check value of K is large enough for Expend ------------"
```

```
gam(rsd~s(Expend,k=20)-1,data=train.df,select=TRUE)  # -1 supresses the intercept
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## rsd ~ s(Expend, k = 20) - 1
##
## Estimated degrees of freedom:
## 0  total = 0
##
## GCV score: 0.03305406
```

```
print(" ------ Check value of K is large enough for Grad.Rate ------------")
```

```
## [1] " ------ Check value of K is large enough for Grad.Rate ------------"
```

```r
gam(rsd~s(Grad.Rate,k=20)-1,data=train.df,select=TRUE)  # -1 supresses the intercept
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## rsd ~ s(Grad.Rate, k = 20) - 1
##
## Estimated degrees of freedom:
## 0  total = 0
##
## GCV score: 0.03305406
```

```r
print("We can extract the lambdai estimates, and, for RE/ML smoothness
selection, the covariance matrix of the log lambdai estimates.")
```

```
## [1] "We can extract the lambdai estimates, and, for RE/ML smoothness\nselection, the covariance matri
```

```r
best.model.fit$sp
```

```
##  s(Room.Board)    s(Personal)    s(Terminal) s(perc.alumni)      s(Expend)
##      508.47050    15805.81574     7593.68122    21058.58540       24.74739
##   s(Grad.Rate)
##     1711.22428
```

```r
# Alternativly
#gam.vcomp(best.model.fit)

print(" Seems like the only statistically significant covariate is Room.Board")
```

```
## [1] " Seems like the only statistically significant covariate is Room.Board"
```

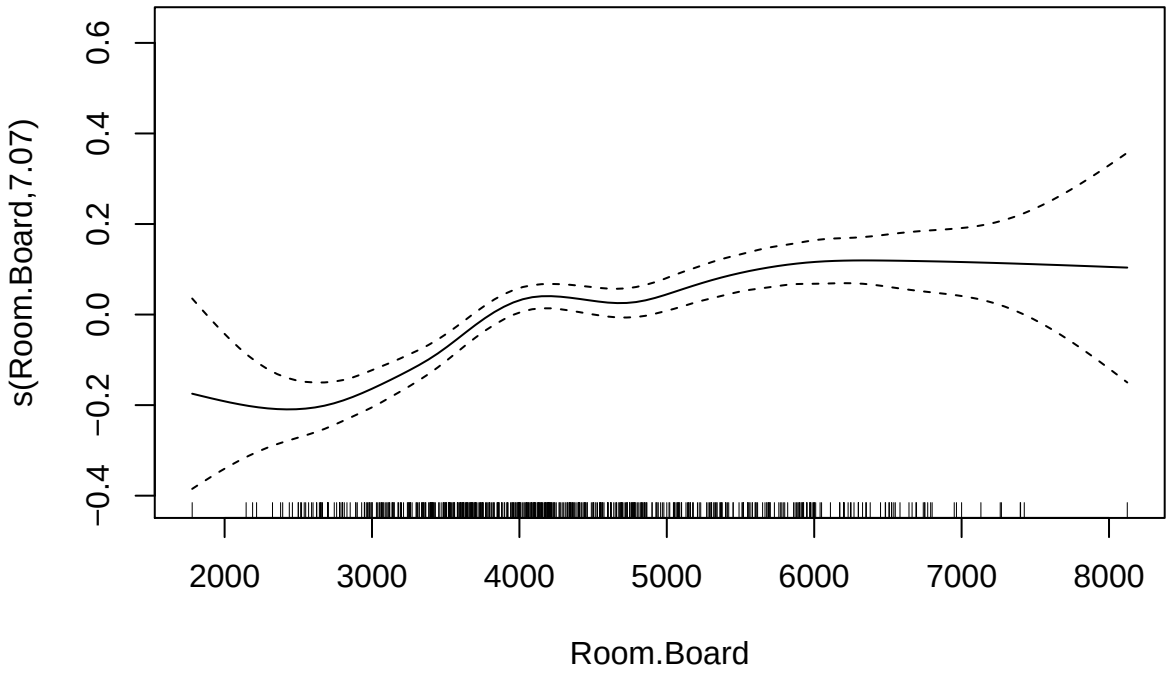
```r
anova(best.model.fit)
```

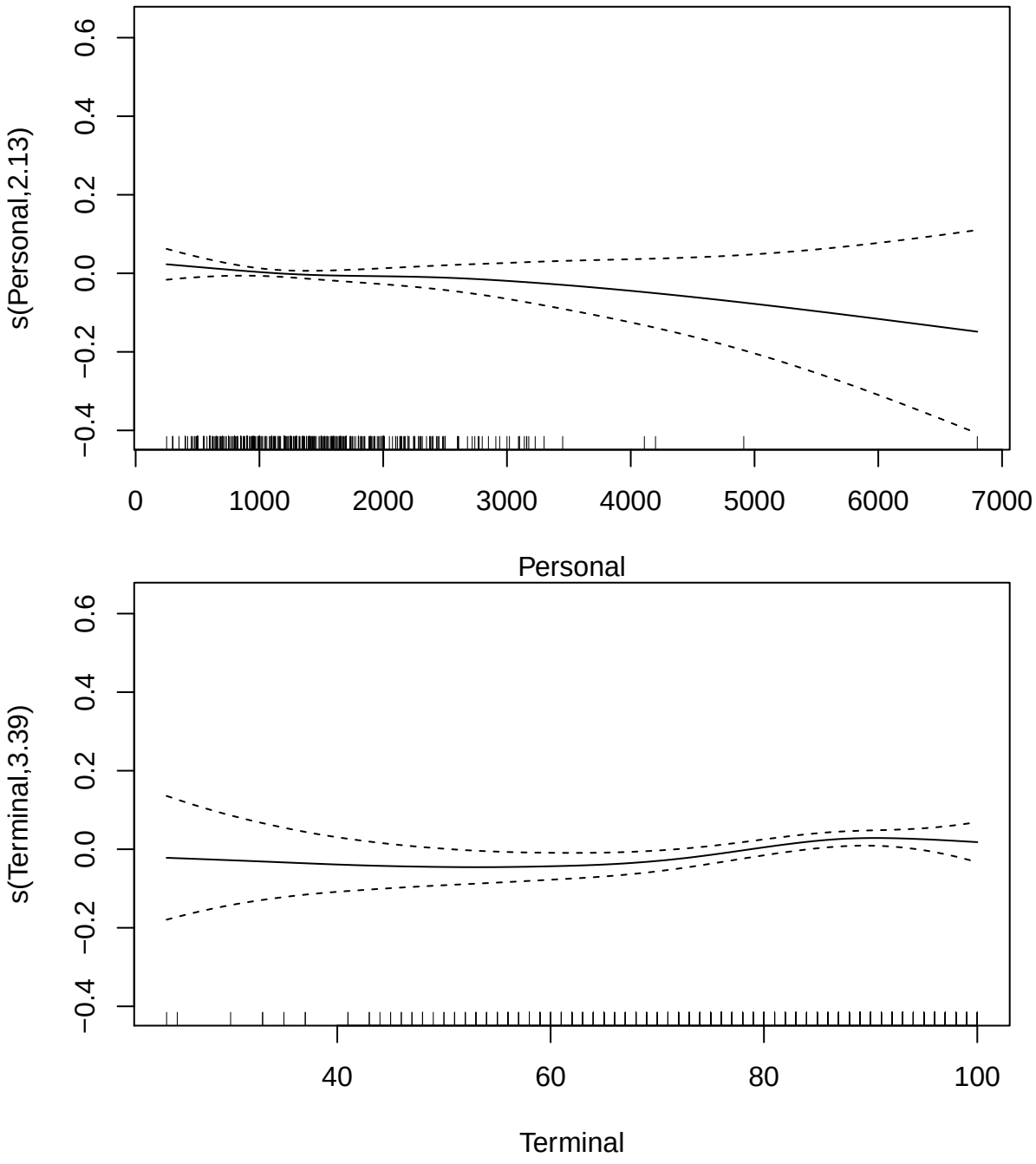
```
##
## Family: Gamma
## Link function: log
##
## Formula:
## Outstate ~ s(Room.Board, bs = "cr", k = 20) + s(Personal, bs = "cr",
##     k = 20) + s(Terminal, bs = "cr", k = 20) + s(perc.alumni,
##     bs = "cr", k = 20) + s(Expend, bs = "cr", k = 20) + s(Grad.Rate,
##     bs = "cr", k = 20) + Private
##
## Parametric Terms:
##         df     F p-value
## Private  1 144.1  <2e-16
##
## Approximate significance of smooth terms:
##                   edf Ref.df      F  p-value
## s(Room.Board)   7.074  8.685 11.699  < 2e-16
## s(Personal)     2.130  2.705  0.834 0.412423
## s(Terminal)     3.386  4.289  2.250 0.052949
## s(perc.alumni)  2.638  3.362  6.363 0.000193
## s(Expend)      10.405 12.048 15.764  < 2e-16
## s(Grad.Rate)    4.551  5.790  8.577 1.46e-08
```

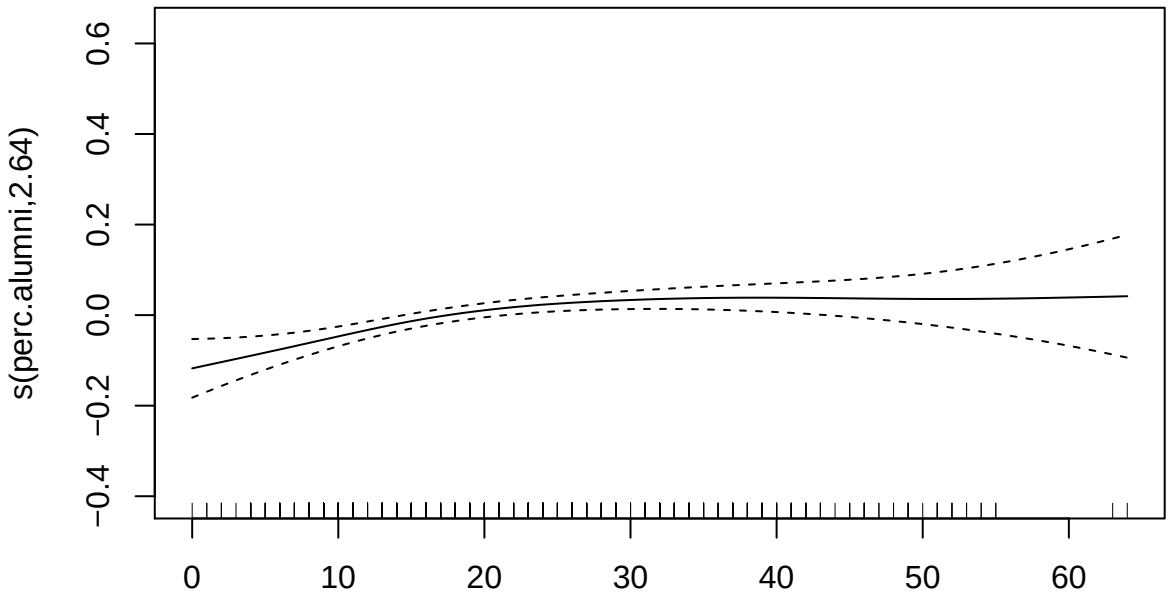
```r
best.preds <- predict(best.model.fit, newdata = test.df, se=TRUE)

# print(" matrix mapping the estimated coefficients to the linear predictor:")
# Xp <- predict(best.model.fit,newdata=test.df,type="lpmatrix")
# Xp%*%coef(best.model.fit) ## result same as predict(ct1,pd)

# let's see how much variance explained by model
tss <- mean((test.df$Outstate - mean(test.df$Outstate))^2)
rss <- mean ((test.df$Outstate - best.preds$fit)^2)
(r.squared <- 1 - rss/tss)
```

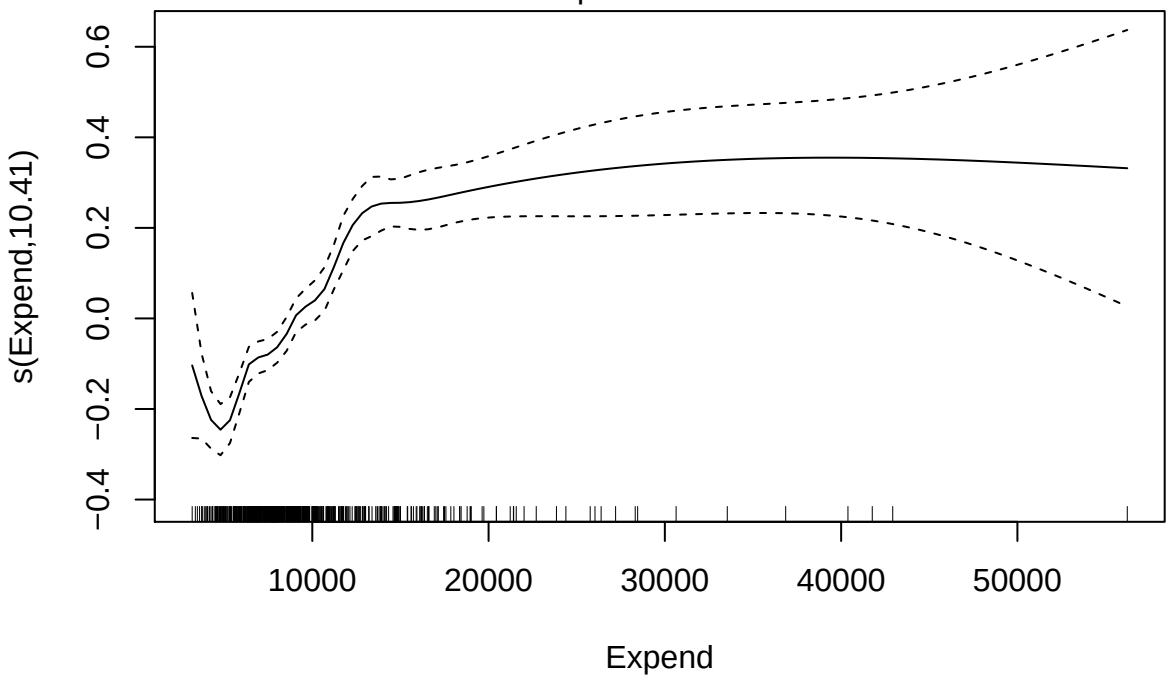
```
## [1] -6.640267
```

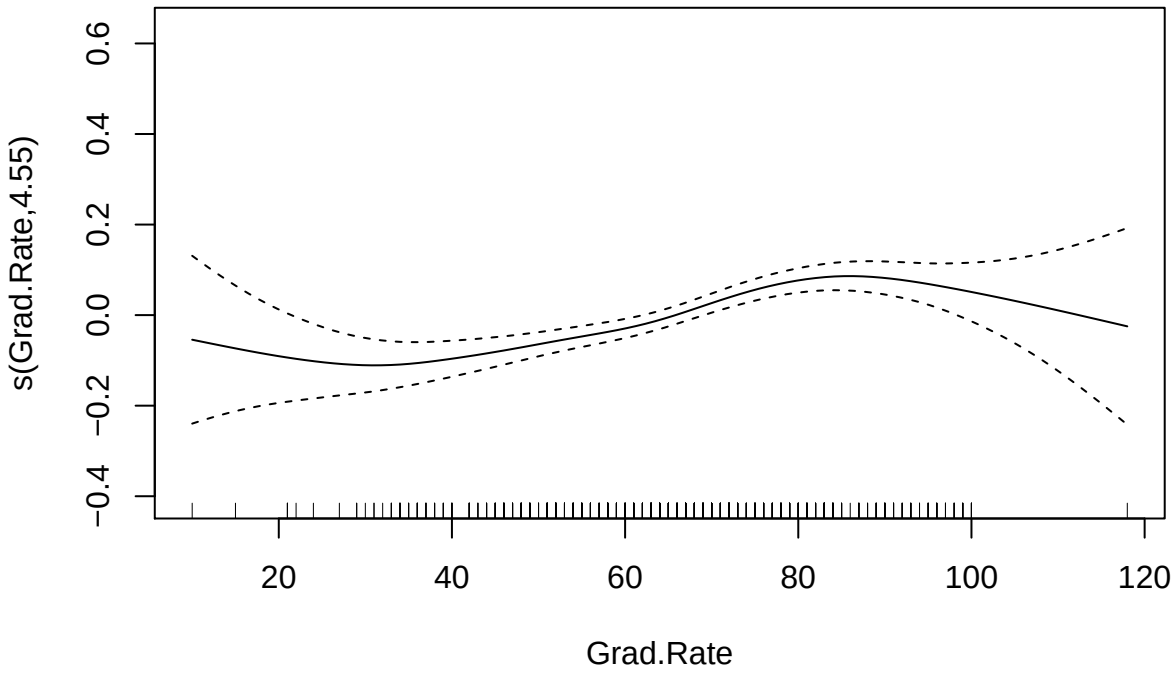
```r
# Clearly  many of the predictors are NOT statistically significant and thus
# the model does not explain any variation in response.

plot(best.model.fit, too.far=0.15)
```




s(Personal,2.13)
0.6
0.4
0.2
0.0
−0.2
−0.4
0
1000
2000
3000
4000
5000
6000
7000
Personal
s(Terminal,3.39)
0.6
0.4
0.2
0.0
−0.2
−0.4
40
60
80
100
Terminal


s(perc.alumni,2.64)
0.6
0.4
0.2
0.0
−0.2
−0.4
0
10
20
30
40
50
60
perc.alumni


s(Expend,10.41)
0.6
0.4
0.2
0.0
−0.2
−0.4
10000
20000
30000
40000
50000
Expend




```
print("-------------- use gam library to get a better result ------------------")
```

```
## [1] "-------------- use gam library to get a better result ------------------"
```

```
library(gam)
```

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

```
##
## Attaching package: 'foreach'
```

```
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
```

```
## Loaded gam 1.16.1
```

```
##
## Attaching package: 'gam'
```

```
## The following objects are masked from 'package:mgcv':
##
##     gam, gam.control, gam.fit, s
```

```
regular.part <- regular.features %>%
  map(~glue::glue("s(",.,,")")) %>%
  str_c(., collapse = " + ")

factor.part <- factor.features %>% str_c(., collapse = " + ")

formula <-
  case_when(
    factor.part != "" & regular.part != "" ~ glue::glue("Outstate ~ " ,
                                                        regular.part ,
                                                        " + ",
                                                        factor.part),
```

```
    factor.part != "" & regular.part == "" ~ glue::glue("Outstate ~ " ,
                                                        factor.part),

    factor.part == "" & regular.part != "" ~ glue::glue("Outstate ~ " ,
                                                regular.part)
  )


best.model.fit = gam(as.formula(formula), data = train.df)
```

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

```
best.preds <- predict(best.model.fit, newdata = test.df)

# let's see how much variance explained by model
tss <- mean((test.df$Outstate - mean(test.df$Outstate))^2)
rss <- mean ((test.df$Outstate - best.preds)^2)

print("seems like with 'gam' librry we get more sensible result: ")
```

```
## [1] "seems like with 'gam' librry we get more sensible result: "
```

```
(r.squared <- 1 - rss/tss)
```

```
## [1] 0.7646616
```

```
print("in this model all chosen covariates are statistically significant")
```

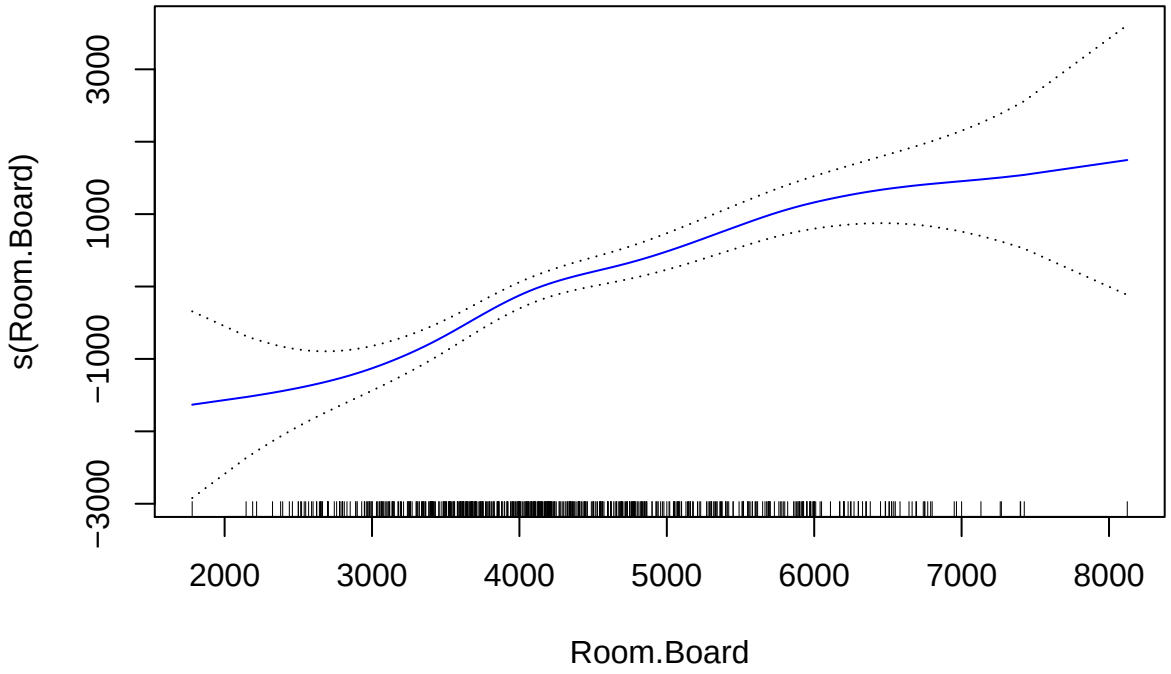
```
## [1] "in this model all chosen covariates are statistically significant"
```

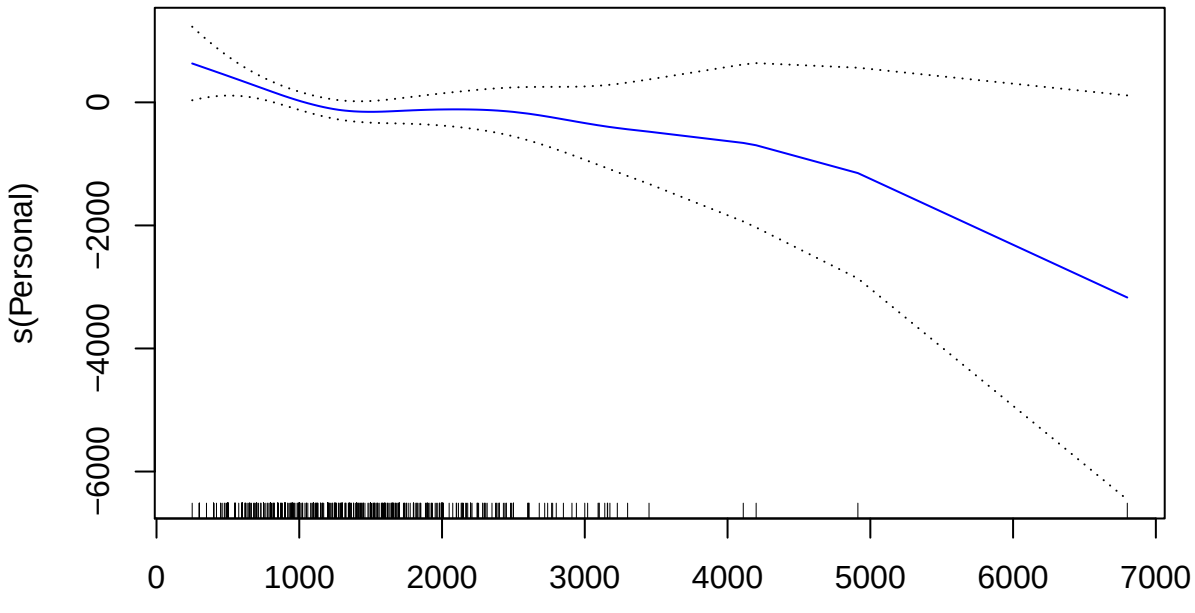
```
summary(best.model.fit)
```

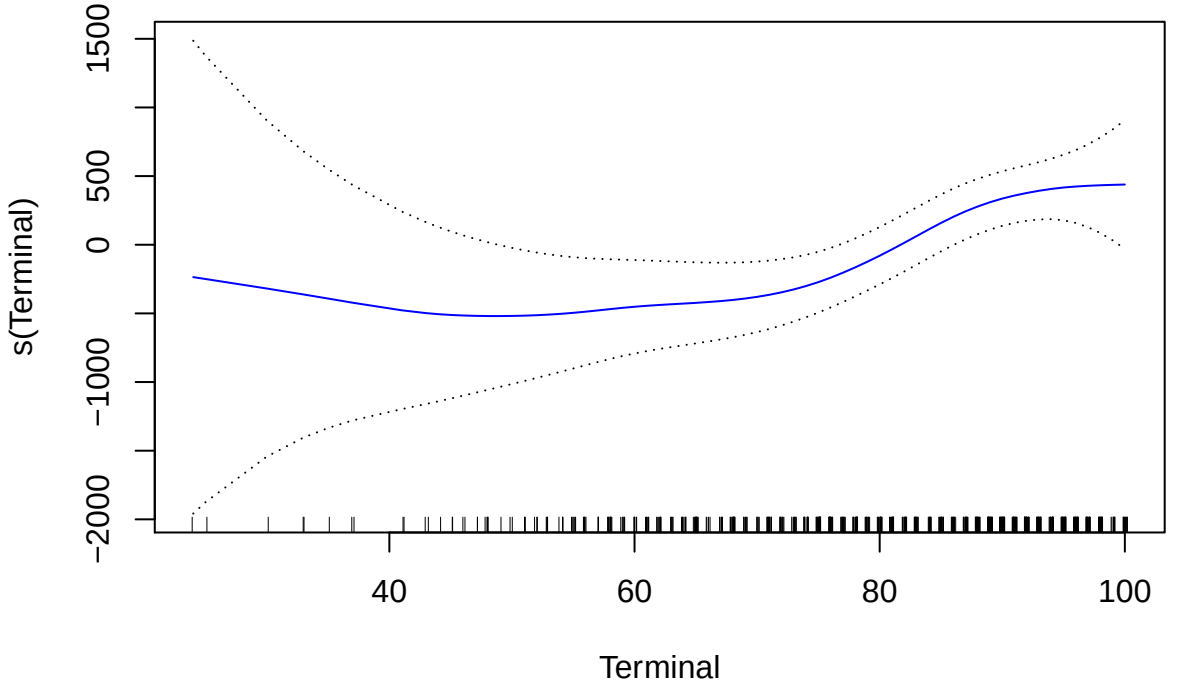
```
##
## Call: gam(formula = as.formula(formula), data = train.df)
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -7332.62 -1094.39    15.07  1293.45  7594.22
##
## (Dispersion Parameter for gaussian family taken to be 3265247)
##
##     Null Deviance: 9885716776 on 620 degrees of freedom
## Residual Deviance: 1942821465 on 594.9998 degrees of freedom
## AIC: 11104.05
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##                  Df     Sum Sq    Mean Sq  F value    Pr(>F)
## s(Room.Board)     1 3596011772 3596011772 1101.298 < 2.2e-16 ***
## s(Personal)       1  206275898  206275898   63.173 9.537e-15 ***
## s(Terminal)       1  225836517  225836517   69.164 6.189e-16 ***
## s(perc.alumni)    1  973836350  973836350  298.243 < 2.2e-16 ***
## s(Expend)         1 1087873298 1087873298  333.167 < 2.2e-16 ***
## s(Grad.Rate)      1  182079552  182079552   55.763 2.925e-13 ***
## Private           1  410027727  410027727  125.573 < 2.2e-16 ***
```

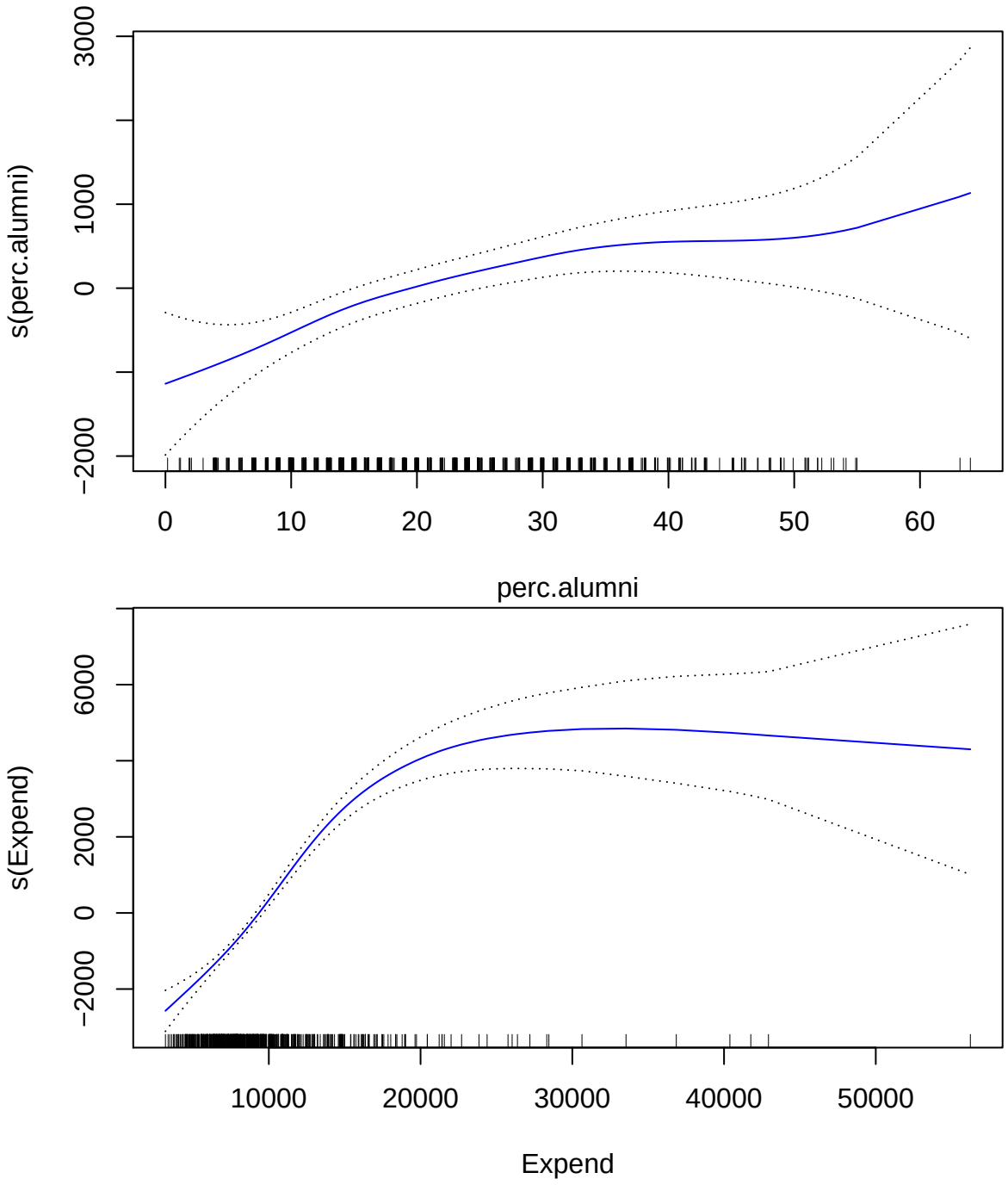
```
## Residuals      595 1942821465    3265247
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                Npar Df Npar F     Pr(F)
## (Intercept)
## s(Room.Board)        3  2.682  0.046048 *
## s(Personal)          3  2.015  0.110633
## s(Terminal)          3  1.826  0.141209
## s(perc.alumni)       3  1.660  0.174572
## s(Expend)            3 34.278 < 2.2e-16 ***
## s(Grad.Rate)         3  5.131  0.001647 **
## Private
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

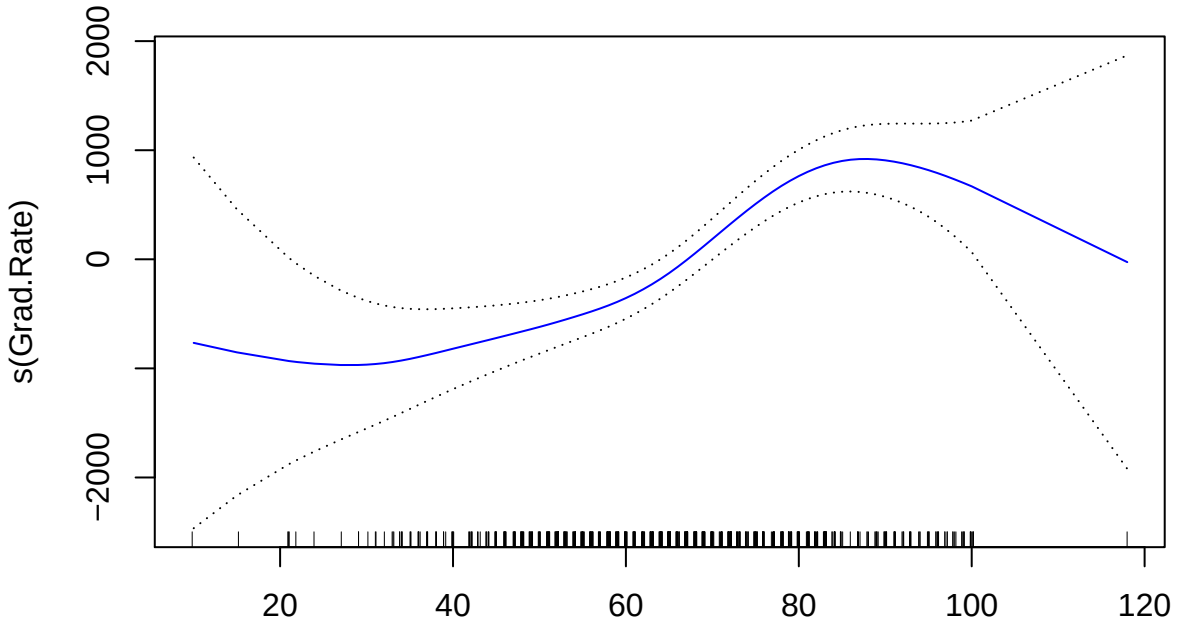
```
plot(best.model.fit, se = T, col = "blue")
```

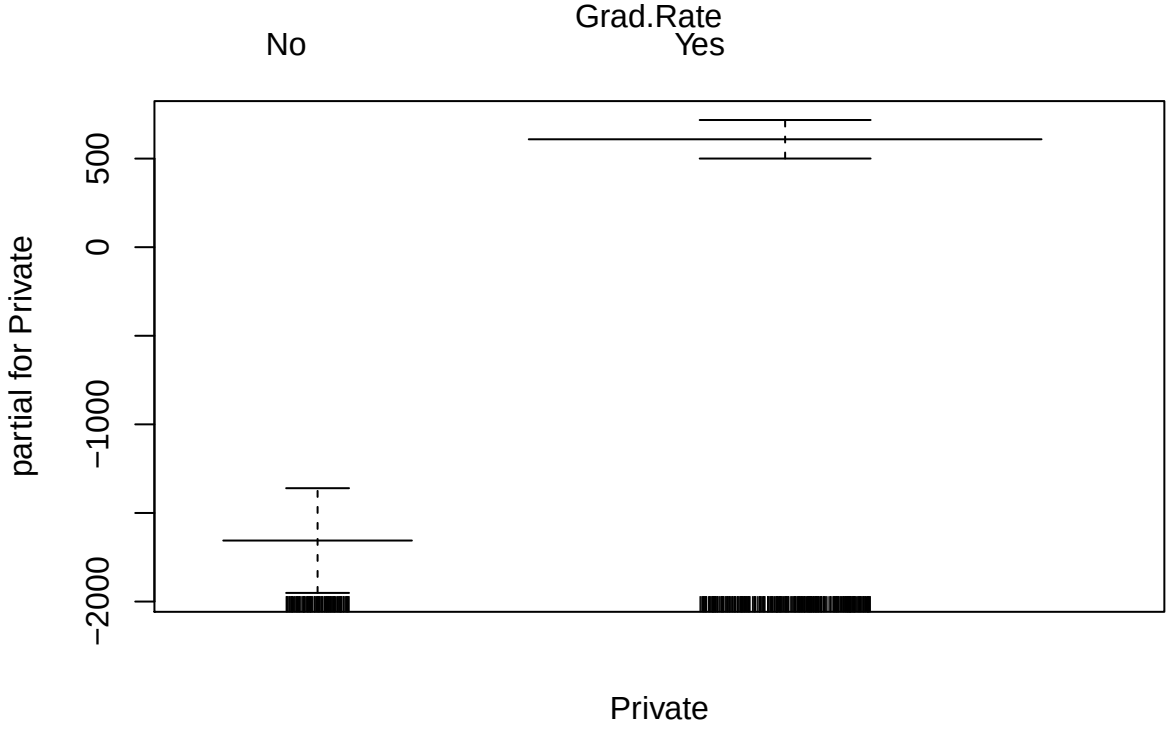



s(Personal)
0
−2000
−4000
−6000
0
1000
2000
3000
4000
5000
6000
7000
Personal


s(Terminal)
1500
500
0
−1000
−2000
40
60
80
100
Terminal


s(perc.alumni)
3000
1000
0
−2000
0
10
20
30
40
50
60
perc.alumni
s(Expend)
6000
2000
0
−2000
10000
20000
30000
40000
50000
Expend






```
print("Plot shows all covariates except 'private' are non linear")
```

```
## [1] "Plot shows all covariates except 'private' are non linear"
```

```
library(tidyverse)
# a) Generate response Y and predictor X1 and X2 using n = 100

set.seed(1)
x1 <- rnorm(100)
x2 <- rnorm(100)
y <- 1003.24 + 0.002*x1 - 0.0089*x2+rnorm(100)
```

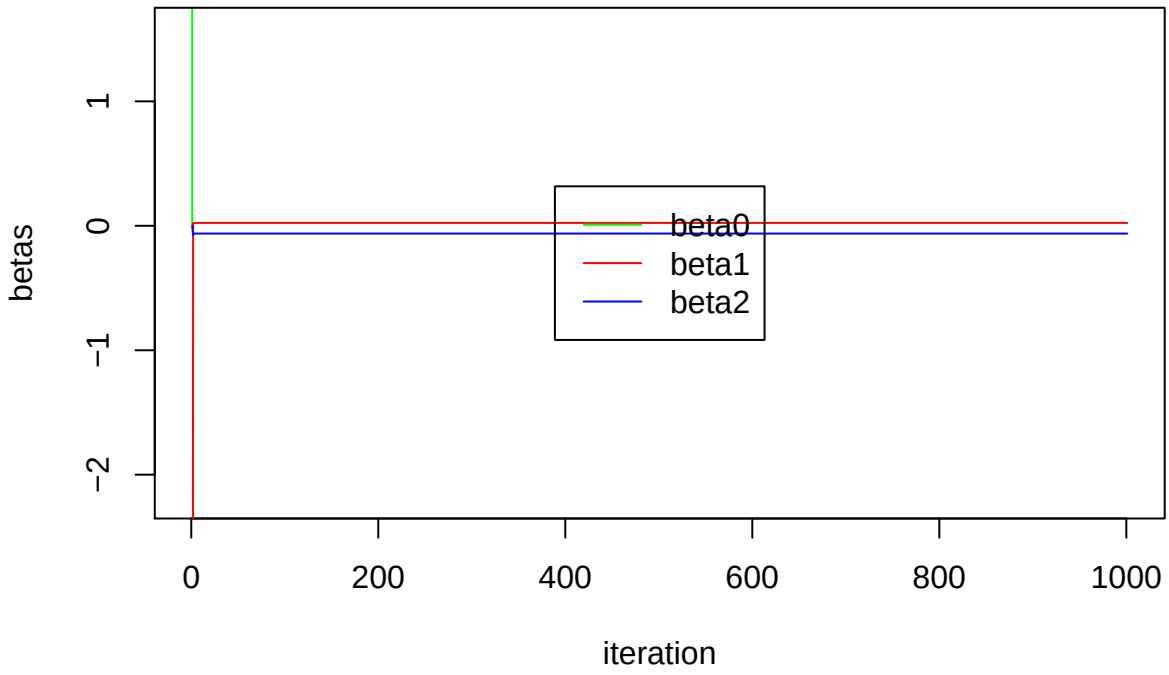
```r
# b) initialize beta1.hat
beta1.hat <- 0.2

# c) keep beta1.hat fixed and fit the model
a <- y - beta1.hat * x1
beta2.hat <- lm(a ~ x2)$coef[[2]]

# d) keep beta2.hat fixed and fit the model
a <- y - beta2.hat * x2
beta1.hat <- lm(a ~ x1)$coef[[2]]
beta1.hat
```

```
## [1] 0.02311034
```

```r
# e) repeat c and d above 100 times and plot the values

beta1.hat <- -13.3
result <- tibble(b0=0, b1=beta1.hat, b2=0)

find.betas <- function(acc){
  beta1.hat <- acc[nrow(acc), ]$b1
  a1 <- y - beta1.hat * x1
  model1 <- lm(a1 ~ x2)
  beta2.hat <- model1$coef[[2]]

  a2 <- y - beta2.hat * x2
  model2 <- lm(a2 ~ x1)
  beta1.hat <- model2$coef[[2]]
  beta0.hat <- model2$coef[[1]]

  rbind(acc ,tibble(b0 = beta0.hat, b1 = beta1.hat, b2 = beta2.hat))
}

results <- 1:1000 %>%
  reduce(~find.betas (.x), .init=result)

dim(results)
```

```
## [1] 1001    3
```

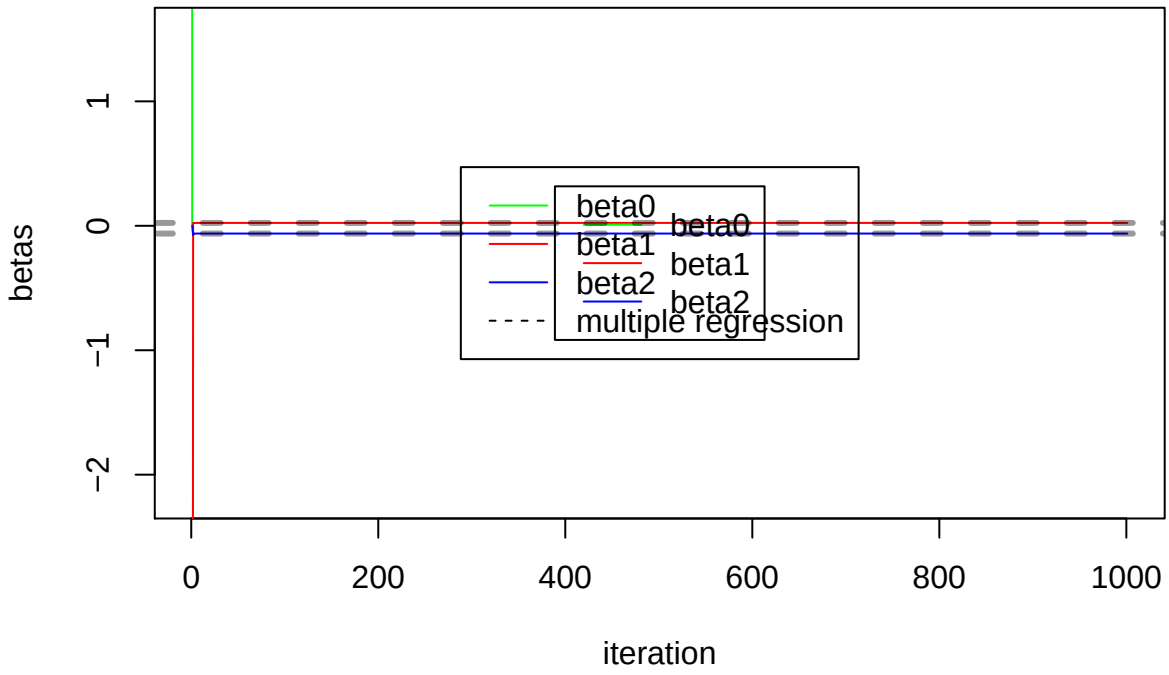
```r
plot(1:1001, results$b0, type = "l", xlab = "iteration", ylab = "betas", ylim = c(-2.2,
    1.6), col = "green")
lines(1:1001, results$b1, col = "red")
lines(1:1001, results$b2, col = "blue")
legend("center", c("beta0", "beta1", "beta2"), lty = 1, col = c("green", "red",
    "blue"))
```




```
# f) perform a linear regression and compare thr coefficients
model3 <- lm(y ~ x1 + x2)
model3$coefficients
```

```
##   (Intercept)            x1            x2
## 1003.26535343    0.02311017   -0.06236682
```

```
# Results are very close
#   (Intercept)            x1            x2
# 1003.26535343    0.02311017   -0.06236682
# 1003.265         0.02311017     -0.06236682

plot(1:1001, results$b0, type = "l", xlab = "iteration", ylab = "betas", ylim = c(-2.2,
    1.6), col = "green")
lines(1:1001, results$b1, col = "red")
lines(1:1001, results$b2, col = "blue")
legend("center", c("beta0", "beta1", "beta2"), lty = 1, col = c("green", "red",
    "blue"))
abline(h = model3$coef[1], lty = "dashed", lwd = 3, col = rgb(0, 0, 0, alpha = 0.4))
abline(h = model3$coef[2], lty = "dashed", lwd = 3, col = rgb(0, 0, 0, alpha = 0.4))
abline(h = model3$coef[3], lty = "dashed", lwd = 3, col = rgb(0, 0, 0, alpha = 0.4))
legend("center", c("beta0", "beta1", "beta2", "multiple regression"), lty = c(1,
    1, 1, 2), col = c("green", "red", "blue", "black"))
```




```
# g) Only 2 back fitting iteration was required
```