Lab07

Yixuan Li, UPI:yil845

26 September 2021

The Data: The data is from UCI Machine Learning Repository about the Automobile data set.

Modelling Nonlinear Relationship

Consider polynomial regression models using highway.mpg to predict the value of price.

1,Find all polynomial regression models for degrees 1 to 5. An example of the third order polynomial regression coefficient, the result shows P values of the first 2 orders are very significiant, the third order is insignificiant.

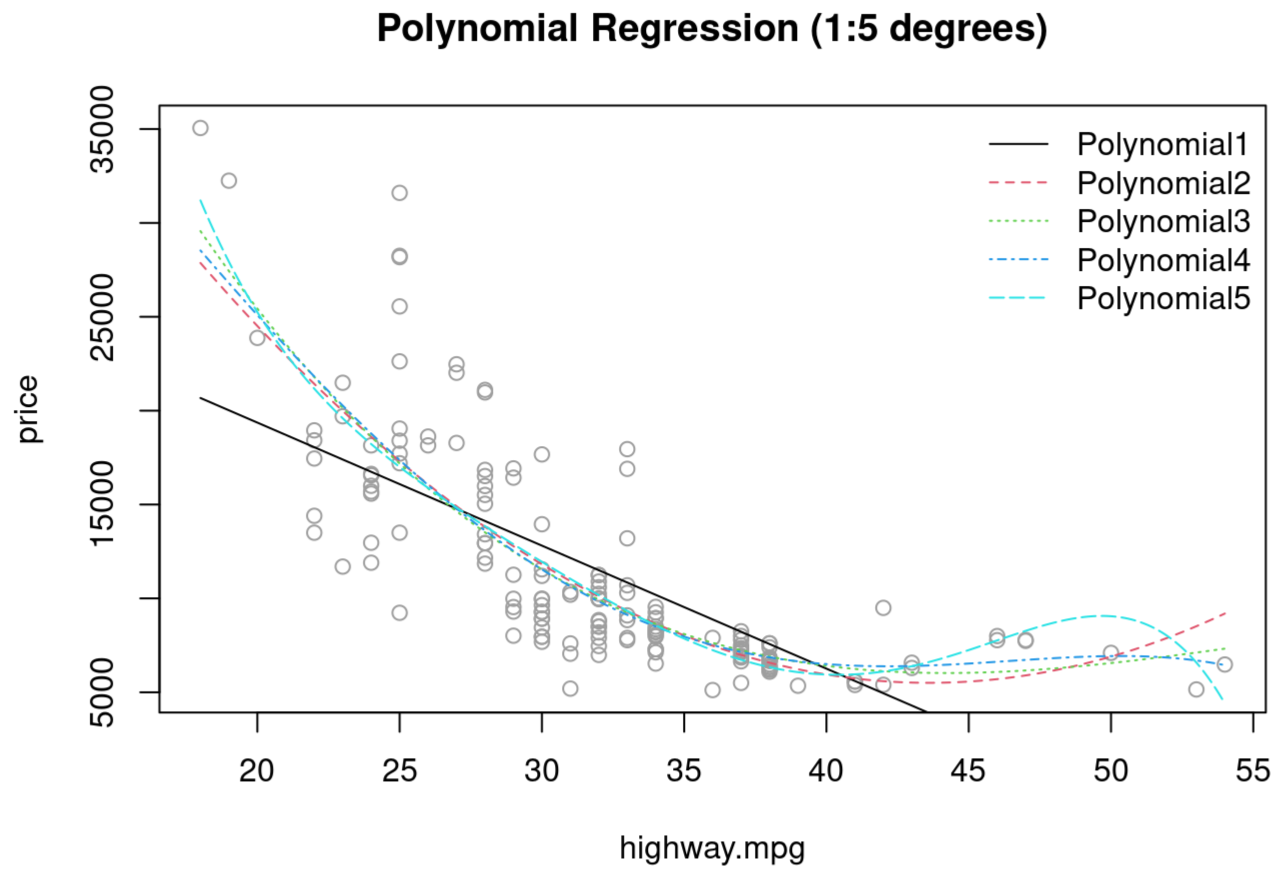
```
polyn1<- lm(price ~ poly(highway.mpg,1,raw = TRUE),data=auto)
polyn2<-lm(price ~ poly(highway.mpg,2,raw = TRUE),data=auto)
polyn3<-lm(price ~ poly(highway.mpg,3,raw = TRUE),data=auto)
polyn4<-lm(price ~ poly(highway.mpg,4,raw = TRUE),data=auto)
polyn5<-lm(price ~ poly(highway.mpg,5,raw = TRUE),data=auto)
summary(polyn3)
```

```
##
## Call:
## lm(formula = price ~ poly(highway.mpg, 3, raw = TRUE), data = auto)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8451.8 -1629.1  -354.6   938.1 14416.3
##
## Coefficients:
##                                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)                       93247.4598 18002.8700   5.180 6.82e-07 ***
## poly(highway.mpg, 3, raw = TRUE)1 -5070.7914  1612.7900  -3.144   0.0020 **
## poly(highway.mpg, 3, raw = TRUE)2    95.5198    46.6719   2.047   0.0424 *
## poly(highway.mpg, 3, raw = TRUE)3    -0.5756     0.4357  -1.321   0.1884
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3446 on 155 degrees of freedom
## Multiple R-squared:  0.6629, Adjusted R-squared:  0.6564
## F-statistic: 101.6 on 3 and 155 DF,  p-value: < 2.2e-16
```

2, Superimpose them all in one scatter plot of the data. What do you generally observe in the fitted curve when the polynomial degree increases?

Visually speaking, the 2nd and the 3rd degree polynomial fitting are the best. Increasing degree could only cause over-intepretation of results because of the high noise level and very bumpy curves.

```
### generate a random seq as sample data and fit them with polynormial coefficient 1:10
newx<- data.frame(highway.mpg = seq(min(auto$highway.mpg),max(auto$highway.mpg),by=0.1))
with(auto,plot(highway.mpg,price,pch=21,col=8, main="Polynomial Regression (1:5 degrees)"))
for (i in 1:5) {
  r.poly = lm(price ~ poly(highway.mpg,i,raw = TRUE),data=auto)
  yhat.poly = predict(r.poly, newx)
  lines(newx$highway.mpg,yhat.poly,lty = i, col = i)
}
legend("topright",lty=1:5, col = 1:5,paste0("Polynomial",1:5),bty = "n")
```



3, Find the BIC-selected polynomial regression model. Do you think this is a reasonable fit?

The result seems reasonable. The second BIC (degree of 2) is the smallest, thus the best fitting. But it only based on the log-likelihood fitting of 159 observations and k parameters in the dataset. Noted this calculated BIC is a relative value because the constant C is not included. According to in-build BIC() function, the constant is -344.5955.

```
n<-nrow(auto)
BIC1<- log(sum(residuals(polyn1)^2))*n+log(n)*(length(polyn1$coefficients)-1)
BIC2<- log(sum(residuals(polyn2)^2))*n+log(n)*(length(polyn2$coefficients)-1)
BIC3<- log(sum(residuals(polyn3)^2))*n+log(n)*(length(polyn3$coefficients)-1)
BIC4<- log(sum(residuals(polyn4)^2))*n+log(n)*(length(polyn4$coefficients)-1)
BIC5<- log(sum(residuals(polyn5)^2))*n+log(n)*(length(polyn5$coefficients)-1)
BIC1;BIC2;BIC3;BIC4;BIC5
```

```
## [1] 3453.712
```

```
## [1] 3403.894
```

```
## [1] 3407.183
```

```
## [1] 3411.729
```

```
## [1] 3412.667
```

```
min(BIC1,BIC2,BIC3,BIC4,BIC5)
```

```
## [1] 3403.894
```

```
## Comparison to results from in-build BIC()
BIC(polyn1); BIC(polyn2);BIC(polyn3);BIC(polyn4); BIC(polyn5)
```

```
## [1] 3109.116
```

```
## [1] 3059.298
```

```
## [1] 3062.587
```

```
## [1] 3067.134
```

```
## [1] 3068.072
```

```
## Constant
BIC(polyn1)-BIC1; BIC(polyn5)-BIC5
```

```
## [1] -344.5955
```

```
## [1] -344.5955
```

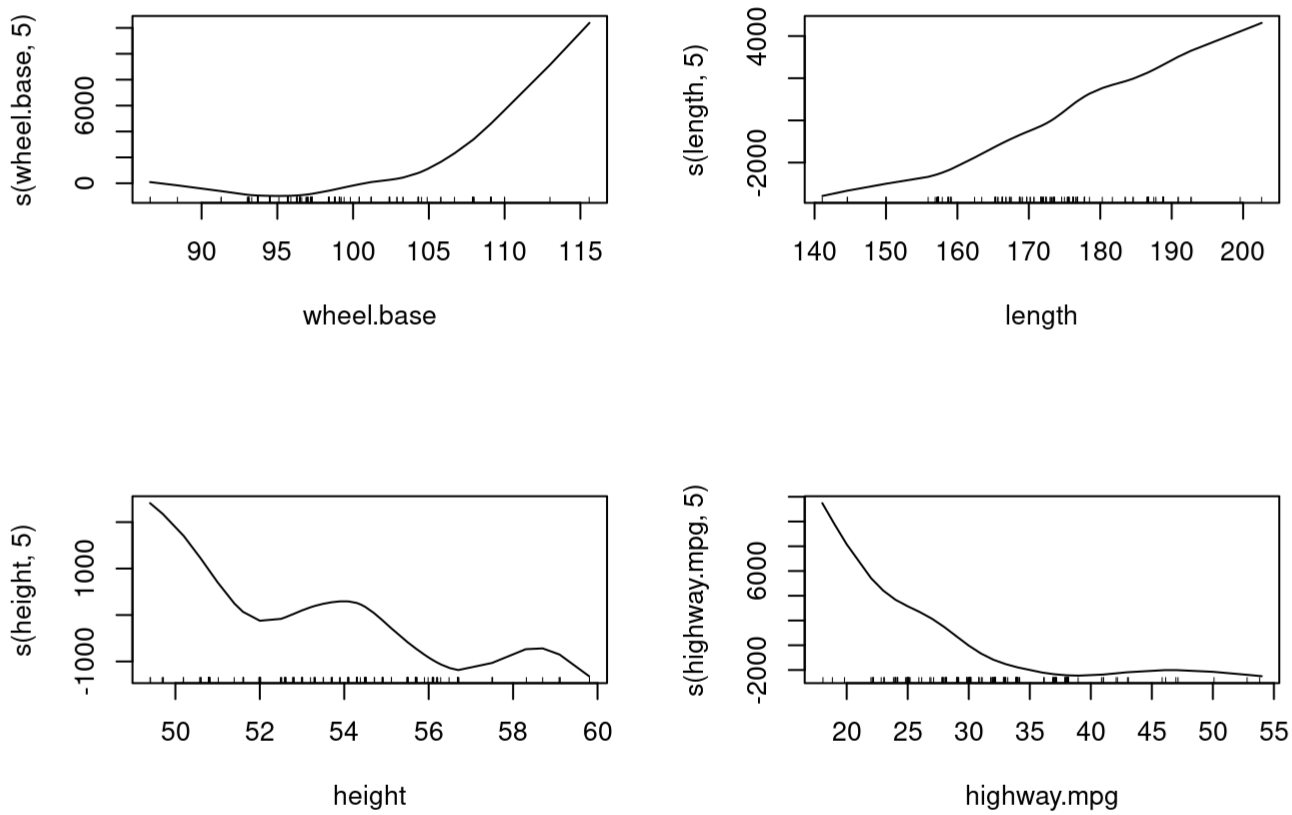
Generalised Additive Models

4, Fit a GAM to the data set, using a smoothing spline with 5 degrees of freedom for each predictor variable: wheel.base, length, height, highway.mpg.

```
r.gam = gam(price ~ s(wheel.base, 5) + s(length,5) + s(height, 5) + s(highway.mpg,5), data=auto)
summary((r.gam))
```

```
##
## Call: gam(formula = price ~ s(wheel.base, 5) + s(length, 5) + s(height,
##     5) + s(highway.mpg, 5), data = auto)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -7655.4 -1132.0  -111.5   726.5 10546.1
##
## (Dispersion Parameter for gaussian family taken to be 6419159)
##
##     Null Deviance: 5458772565 on 158 degrees of freedom
## Residual Deviance: 885845691 on 138.0003 degrees of freedom
## AIC: 2964.993
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##                    Df     Sum Sq    Mean Sq F value    Pr(>F)
## s(wheel.base, 5)    1 2796569269 2796569269 435.660 < 2.2e-16 ***
## s(length, 5)        1  483447711  483447711  75.313 1.003e-14 ***
## s(height, 5)        1  125420980  125420980  19.538 1.980e-05 ***
## s(highway.mpg, 5)   1  361611027  361611027  56.333 6.835e-12 ***
## Residuals         138  885845691    6419159
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                   Npar Df  Npar F     Pr(F)
## (Intercept)
## s(wheel.base, 5)        4  6.7856 5.137e-05 ***
## s(length, 5)            4  0.9532   0.43542
## s(height, 5)            4  2.2284   0.06906 .
## s(highway.mpg, 5)       4 15.2155 2.491e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
par(mfrow=c(2,2))
plot(r.gam)
```



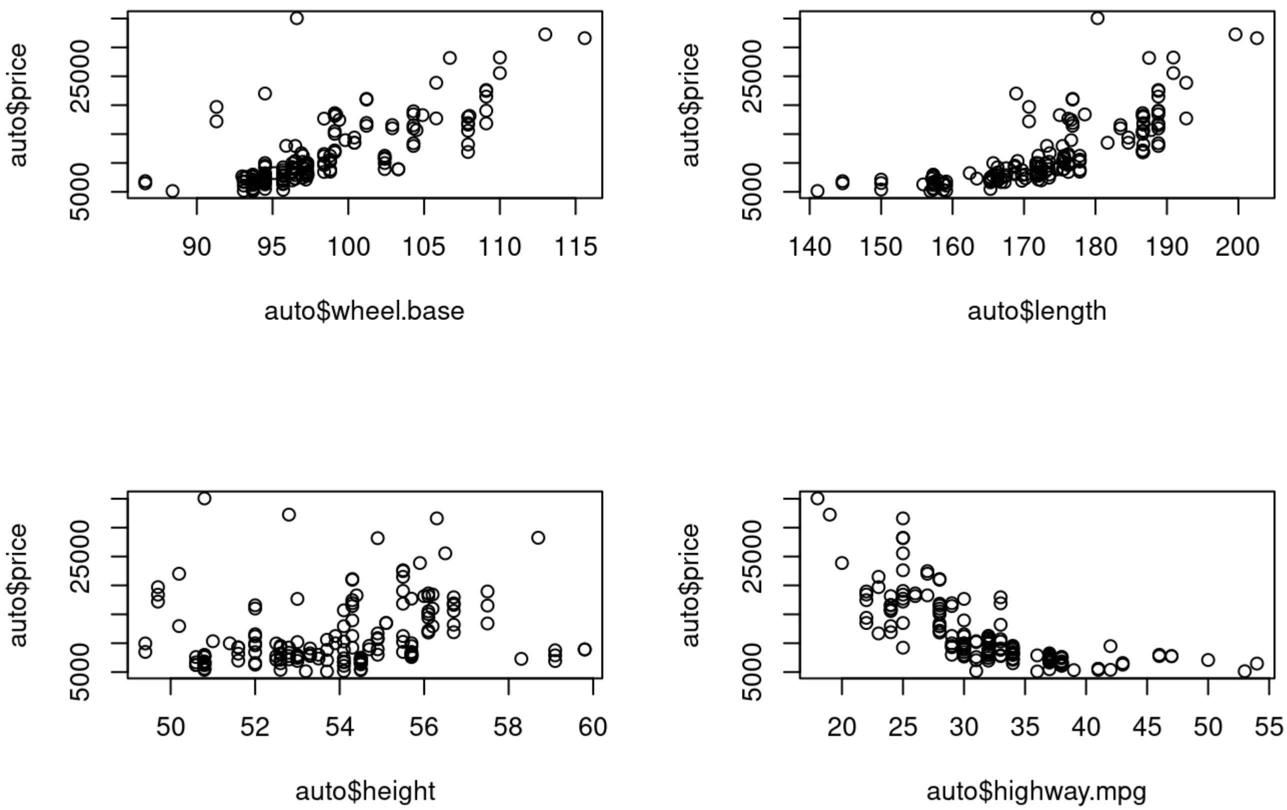
5, Use GAM to predict the price value for a new observation: wheel.base = 110, length = 190, height = 55, highway.mpg = 25.

```
new<- data.frame(wheel.base = 110,length = 190, height = 55, highway.mpg = 25)
predict(r.gam,new)
```

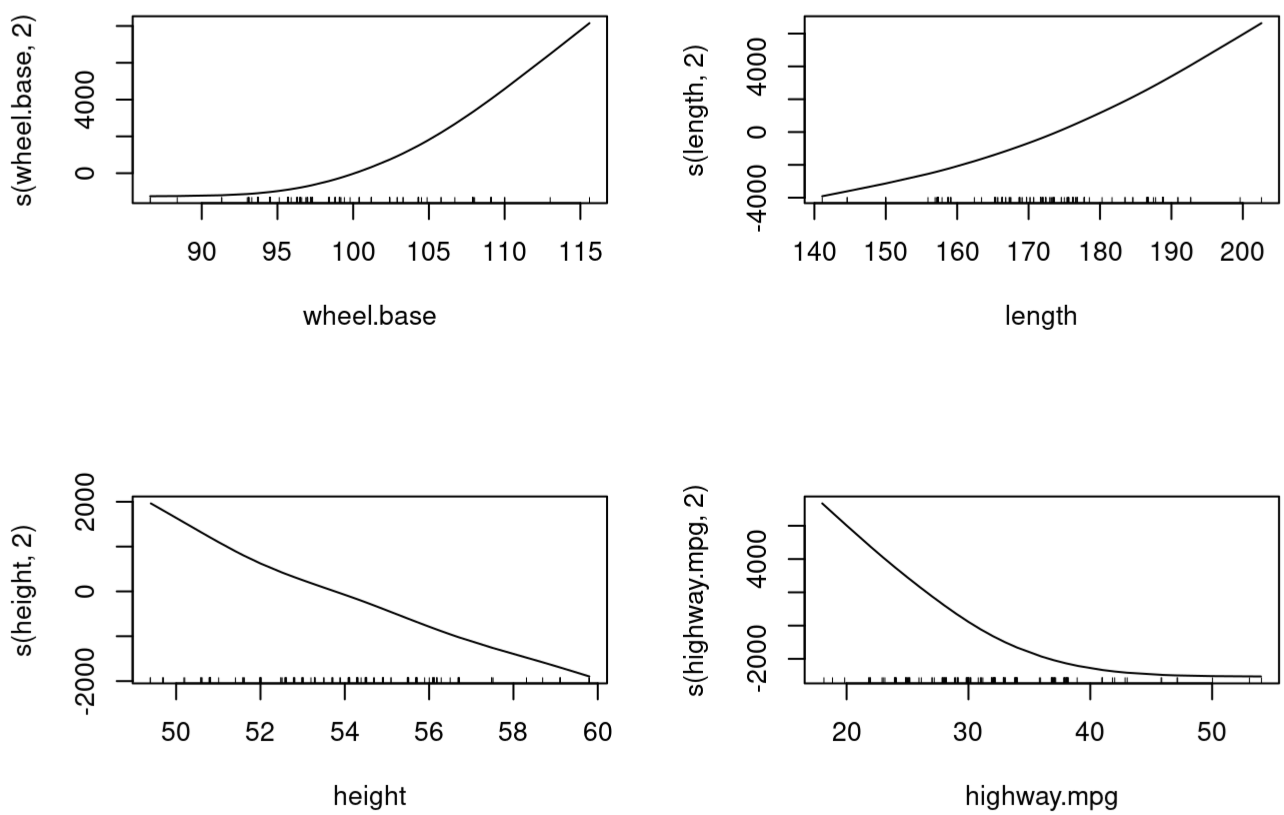
```
##        1
## 22908.86
```

6, Re-fit GAM by adjusting manually the degrees of freedom of smoothing splines. Here I manually adjusted gam smooth spline from degree 2-5. Visually speaking, the good fit is between 2-3 degrees of freedom for all variables.

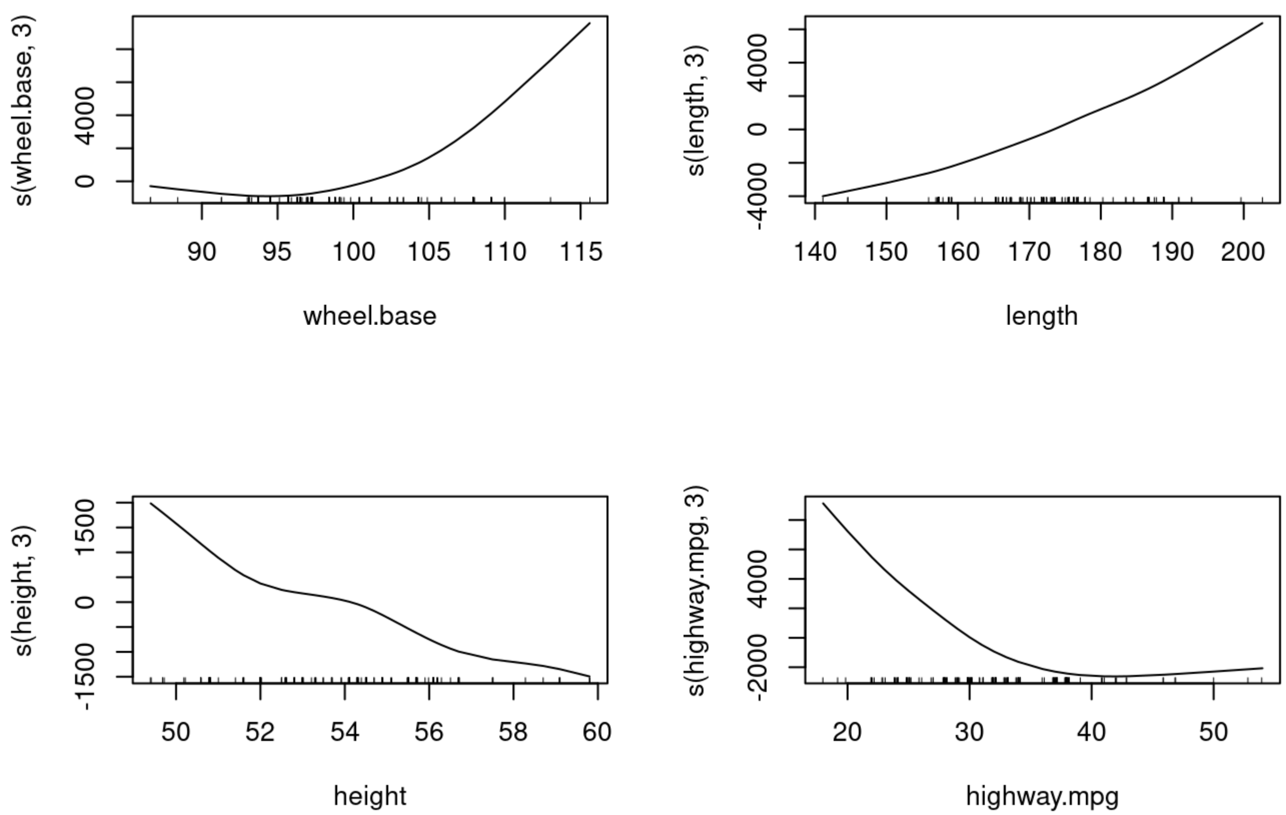
```
r.gam2 = gam(price ~ s(wheel.base, 2) + s(length,2) + s(height, 2) + s(highway.mpg,2), data=aut
o)
r.gam3 = gam(price ~ s(wheel.base, 3) + s(length,3) + s(height, 3) + s(highway.mpg,3), data=aut
o)
r.gam4 = gam(price ~ s(wheel.base, 4) + s(length,4) + s(height, 4) + s(highway.mpg,4), data=aut
o)
r.gam5 = gam(price ~ s(wheel.base, 5) + s(length,5) + s(height, 5) + s(highway.mpg,5), data=aut
o)

par(mfrow=c(2,2))
plot(auto$wheel.base,auto$price); plot(auto$length,auto$price);plot(auto$height,auto$price);plot
(auto$highway.mpg,auto$price)
```



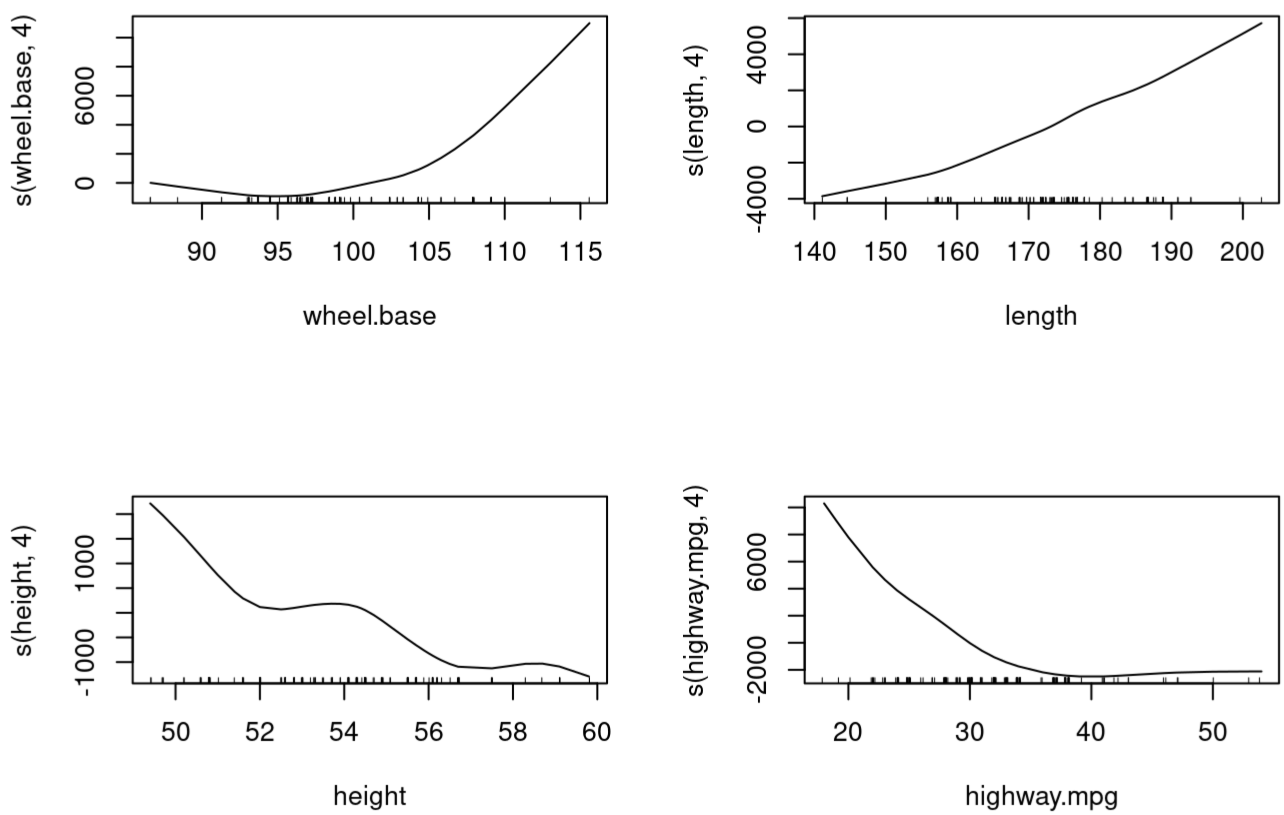

```
plot(r.gam2)
```


s(wheel.base, 2)
0
4000
90
95
100
105
110
115
wheel.base
s(length, 2)
-4000
0
4000
140
150
160
170
180
190
200
length
s(height, 2)
-2000
0
2000
50
52
54
56
58
60
height
s(highway.mpg, 2)
-2000
4000
20
30
40
50
highway.mpg

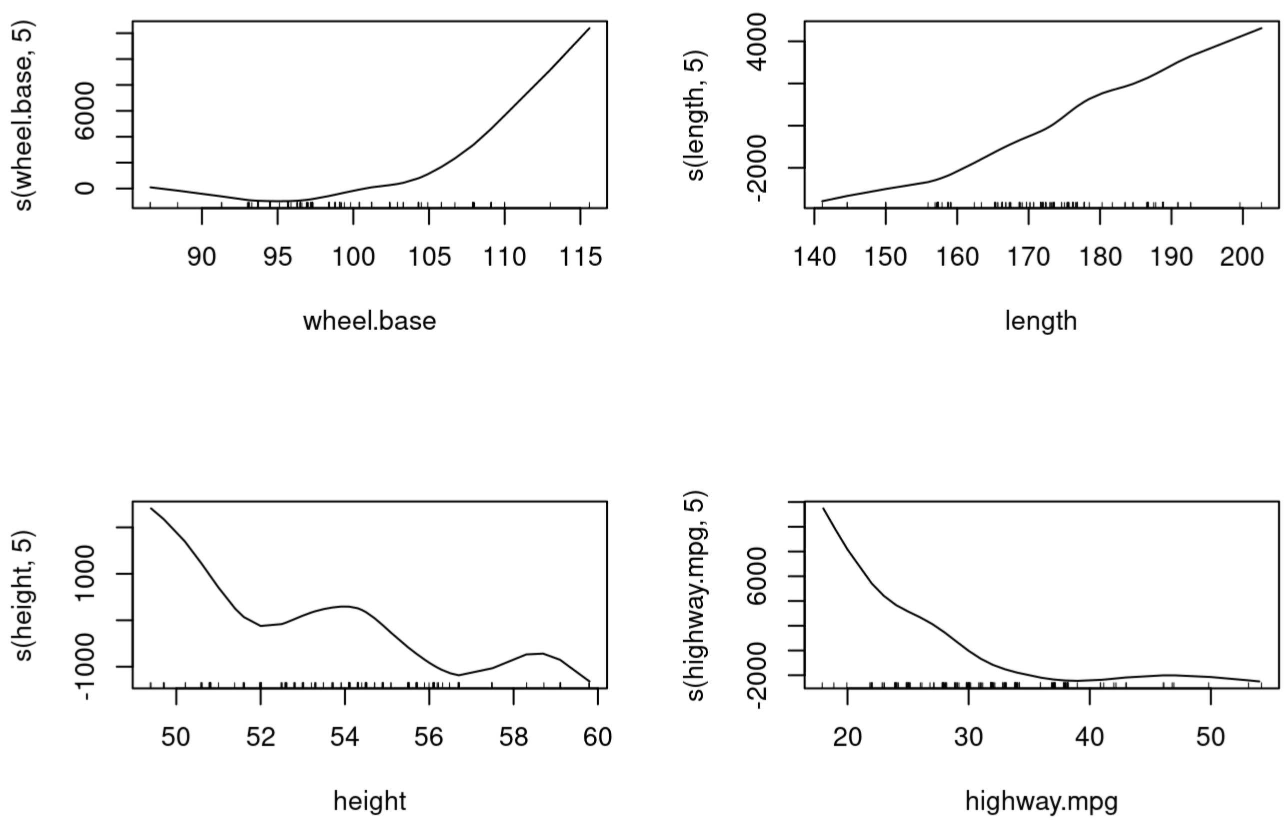
```
plot(r.gam3)
```


s(wheel.base, 3)
0
4000
90
95
100
105
110
115
wheel.base
s(length, 3)
-4000
0
4000
140
150
160
170
180
190
200
length
s(height, 3)
-1500
0
1500
50
52
54
56
58
60
height
s(highway.mpg, 3)
-2000
4000
20
30
40
50
highway.mpg

```
plot(r.gam4)
```


s(wheel.base, 4)
0
6000
90
95
100
105
110
115
wheel.base
s(length, 4)
-4000
0
4000
140
150
160
170
180
190
200
length
s(height, 4)
-1000
1000
50
52
54
56
58
60
height
s(highway.mpg, 4)
-2000
6000
20
30
40
50
highway.mpg

```
plot(r.gam5)
```



Cross-validation for Regression Splines

7, For cubic regression splines, let us consider using knots that are evenly distributed between the two extreme values (minimum and maximum) of the predictor variable, but excluding the two extreme values. That is, given the number of knots, m, we create the knots as follow:

```
m = 5             # number of knots
(rg = range(auto$highway.mpg))# range of min and max of highway.mpg
```

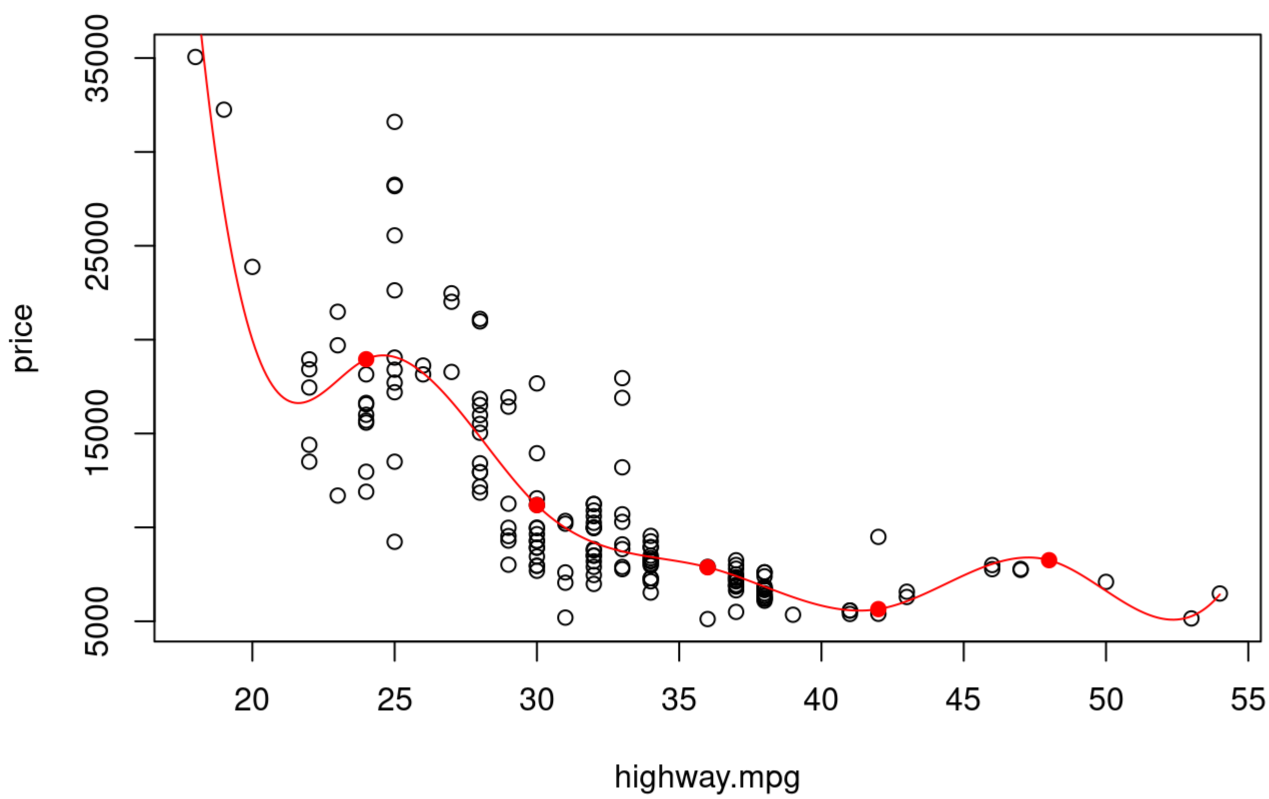
```
## [1] 18 54
```

```
(knots = seq(rg[1], rg[2], length=m+2)[-c(1,m+2)]) # create 5 knots excluding min / max values o
f highway.mpg
```

```
## [1] 24 30 36 42 48
```

The following codes fit the cubic regression spline fit (with 5 knots) to the data (price vs. highway.mpg).A scatter plot of the two variables is shown.

```
## new dataframe is generated based on min / max value of highway.mpg
newx<- data.frame(highway.mpg = seq(min(auto$highway.mpg), max(auto$highway.mpg), by=0.1))

## cubic regression spline fitting with 5 knots
rcub = lm(price ~ bs(highway.mpg, knots=knots, degree=3), data=auto)
yhatcub = predict(rcub, newx)
## plotting, black points are data from auto; red curve shows fitted values based on test data n
ewx; red dots shows 5 knots used for cubic spline fitting
with(auto, plot(highway.mpg, price))
lines(newx$highway.mpg, yhatcub, col="red")
points(knots, predict(rcub, data.frame(highway.mpg=knots)), col="red", pch=19)
```



8, Use 10-fold cross-validation to find an appropriate value for m. Consider m = 1, 2, …, 5 in your CV study. Explain why you have used the technique of same subsamples.

The subsampling is defined in the second outer loop comparing to the knots searching in the inner loop; therefore for a generated new dataset of train and test, all knots of cubic spline fittings are done within the inner loop. The outer most loop is for the number of repetition for subsampling and knots searching process. The knots are chosen based on the min / max of each training set, but excluding end points. Prediction errors (PE) are calculated based on mse, and the minimum PE is found with four-knots cubic spline fittings.

```
## cross validation / fitting m
R = 20                   # number of repetitions
m = 5                    # maximum knots we want to do spline regression
n = nrow(auto)           # data size
K = 10                   # K-fold CV
## function for shuffle the i-th sample from n size of data by K fold selection.
test.set = function(i, n, K=10) {
  if(i < 1 || i > K)
    stop(" i out of range (1, K)")
  start = round(1 + (i-1) * n / K)
  end = ifelse(i == K, n, round(i * n / K))
  start:end
}
##

set.seed(180)           # set a random seed
mse = matrix(nrow=R*K, ncol=m)
for(i in 1:R) {                  # for each repetition
  ind = sample(n)                # shuffle index
  for(k in 1:K) {                # for each fold
    index = ind[test.set(k, n, K)]
    test = auto[index,]
    train = auto[-index,]
    rgtrain = range(train$highway.mpg)
     # for each knots
    for(j in 1:m) {
      knots5 = seq(rgtrain[1], rgtrain[2], length=j+2)[-c(1,j+2)]
      rhw= lm(price ~ bs(highway.mpg, knots=knots5, degree=3, Boundary.knots = range(auto$highwa
y.mpg)), data=train)
      yhathw = predict(rhw, test)        # prediction for test data
      mse[K*(i-1)+k,j] = mean( (test$price - yhathw)^2 )
    }
  }
}
mse
```

```
##           [,1]     [,2]     [,3]     [,4]       [,5]
##  [1,] 13750754 13305223 10932279  8250242  9563940.2
##  [2,]  5003731  5070613  4320867  5271937  6012239.7
##  [3,]  5296729  5115554  5096766  5745323  5650969.3
##  [4,] 18700279 19985574 15392922 10931139 18848139.0
##  [5,]  9979745  9829357  9500618 10089646 10523703.0
##  [6,]  5559532  5340709  4358719  4326376  5815132.8
##  [7,]  9484844  9180117 10745650 12985594 10492436.4
##  [8,] 19112001 20341535 20761097 17805172 15998598.3
##  [9,] 23975995 25736026 26175845 22290073 18811360.4
## [10,] 14034519 15091535 12826697 10084026  7151049.3
## [11,] 32004189 31259761 28136560 27085767 32769096.4
## [12,]  3972780  4207331  5292100  5346281  5012922.4
## [13,]  5001667  4864024  5040059  5294199  5042388.0
## [14,] 15495223 15109220 13088846 12656571 14746951.1
## [15,] 17696816 18319714 17451026 14611595 13237465.5
## [16,] 11320401 10962737 10069710  8726481  7520877.4
## [17,]  8840673  8544748  7964707  7812611  8479101.3
## [18,]  7170325  7437804  4970439  4466139  2554946.3
## [19,] 16024316 16150654 15762437 14106739 13351252.6
## [20,]  9237343  8889448  7458597  6828981  7588755.3
## [21,]  7858181  8299168  6976330  7915544  6514428.9
## [22,] 16955520 19506977 12983862 10570335 21439615.3
## [23,]  5226090  5280919  5129700  4528486  4194328.3
## [24,] 10005590  9825766 10540525 11366237 11244722.5
## [25,] 12693014 14500532 15694738 13055883 10119713.3
## [26,] 13736203 13499023 13535526 13815595 12598509.1
## [27,] 22485534 23202169 21485887 17008787 14283870.4
## [28,] 17657820 18171924 17132762 14072328 12388733.6
## [29,]  8652004  8010661  7061875  7198403  9193259.5
## [30,]  9018329  8807776  7157849  5819181  6588470.9
## [31,] 14577722 14540251 14672432 18215489 14669751.6
## [32,]  6595298  6243200  4891975  4772527  6855754.2
## [33,]  7419919  6726106  5697481  5819526  7994774.3
## [34,]  6827931  7295074  8051658  6594933  5035257.4
## [35,] 10615981 10952182  9622966  7951750  6839260.0
## [36,] 28767173 30955743 30872204 26624004 23815503.1
## [37,] 13095797 13258562  8729452  6956411 16609184.8
## [38,]  7094990  7112390  6518084  4768274  3099357.5
## [39,] 19661197 20044340 18196143 15201388 14923407.5
## [40,] 13147021 12657838 12134715 12937788 13642869.9
## [41,] 10626672 10091786  7842125  6106760  8007120.8
## [42,] 12608250 12459817 11072241 10248208 10796978.2
## [43,]  3164656  3072719  4222600  5358380  5991369.3
## [44,]  8413188  8671990  9230771  7993894  7466045.9
## [45,] 13469797 14476743 14812377 12286571 10144893.7
## [46,] 32513722 32035514 28883034 28172146 34029963.8
## [47,] 21924059 22046401 20826904 18347956 17153545.3
## [48,] 12485305 12026127 10813350  9417552  7599169.8
## [49,]  1385830  2174542  1540654  3247499  1898681.7
## [50,]  7025401  7210855  6227936  5733919  6148309.3
## [51,]  8389243  8102107  7827593  8614445 11392570.5
```

```
## [52,]  4046739  4259244   4135138  4416441   4056195.3
## [53,] 18791970 19113253  19537585 17355698  14084387.4
## [54,] 13129063 12852507  10905135  7613029   6346968.8
## [55,] 31212736 31027066  31230697 29597608  28147481.1
## [56,]  3177685  4522955   4108257  4824082   2977375.3
## [57,]  5895305  6297189   6095491  5411061   4826713.2
## [58,] 17149604 17209967  15688130 13200188  11541952.6
## [59,]  7266252  7023610   7150536  7733078   8929996.8
## [60,] 12872190 13759052   9097322  5008554  16412512.0
## [61,] 10960354 10600298   9460315  8456513   8607033.4
## [62,] 14145430 13881702  12461644 10845580  11723733.7
## [63,] 18109513 17890605  17332716 15752711  15146887.8
## [64,]  4094065  5582466   6721957  9323635   6107346.9
## [65,] 25719195 27085896  26154238 21827505  19247078.9
## [66,] 10699489 10476655   9163127  6956945   6282496.2
## [67,] 15569754 17324869  11791579  8571577  22862595.4
## [68,] 11250139 11870117  11012921  9409007   9951950.3
## [69,]  2805086  2988453   3024638  2978428   2215787.2
## [70,] 13780342 13458877  13112884 12766258   9844684.8
## [71,] 27367843 27802398  27651129 26015495  23512622.8
## [72,]  3522701  3805789   4670765  5041416   4317575.4
## [73,] 10248263 10577357   9600628 12007134  12220314.2
## [74,] 20870475 21370305  17075905 13235531  19663525.2
## [75,] 17043339 17272113  16246144 13885583  12592973.9
## [76,]  8741932  9268145  10922269 10569004   8282601.7
## [77,]  7006391  6411419   5900724  5642742   7092941.1
## [78,]  4829784  4897880   5106973  6111933   5924382.3
## [79,]  1590516  1860714   1760194  1280694    635917.3
## [80,] 20263469 19931368  17407091 13412112  12238117.0
## [81,] 11436482 10888216   8889891  7978504   6904173.4
## [82,] 14485961 15356321  10561534  8535911  20268391.9
## [83,]  5584366  7495536   8669563 36501207  10632556.5
## [84,]  6813743  6721743   5864045  5310440   6162098.5
## [85,] 15418817 15099972  12250921  8885914  10826345.8
## [86,]  5307202  5413080   4968746  5128207   5403549.8
## [87,] 24377515 25831891  28735406 24877635  20654820.3
## [88,] 29812082 30084702  29299514 26258745  26673063.6
## [89,]  8606697  9235521   9877739  8903202   7393623.9
## [90,]  3214495  3157579   3070910  3567376   3949606.7
## [91,] 14098187 16223193  11257013  5706110  15786497.4
## [92,] 16333596 17413035  19860135 19342521  15214632.1
## [93,]  6030068  5718789   5144208  4588137   5347672.1
## [94,] 22092135 22907551  21129095 16256152  13170180.6
## [95,] 12986631 12410500  10668828 10733171  12859090.4
## [96,] 10108662  9800886   9268600 10070039   9045935.0
## [97,] 14802319 14365453  13438833 12447919  11569915.5
## [98,]  7235006  7248035   7242677  6521786   6447831.0
## [99,]  3050359  3874832   3007687  8185838   6071236.4
## [100,] 17723816 19281335  20207343 17679470  15762250.8
## [101,]  3503679  3339799   3629594  4223840   3072001.7
## [102,] 14715439 16388899  10691537  7191629  17292342.5
## [103,] 15270606 15490349  14442464 12599437  12434915.5
```

```
## [104,] 13540217 12941061  11003476 11759444  14940980.2
## [105,]  2090900  2334495   1981346  1503925   3717637.4
## [106,] 24291326 32874892 478470120 78087222 975722453.8
## [107,] 18686290 19112725  18200106 14795910  12547415.6
## [108,] 10348551 10609333   9644334  7858405   7080613.3
## [109,] 16476987 16716681  17033234 15169860  12743988.0
## [110,] 19213021 19586759  19376346 16654948  13700536.3
## [111,]  4261006  3997486   4654865  5076448   4942695.5
## [112,] 10752051 10969355  12602096 12606971   9802409.5
## [113,] 20259129 20668751  19601778 16704036  15197675.7
## [114,] 20251850 20397612  16305375 13893056  24501746.0
## [115,]  6366961  6271060   5017638  4192342   5739887.7
## [116,] 20081589 19697165  17070380 13657141  13430224.0
## [117,]  5944910  8489923   9055605  8881644   5520223.2
## [118,]  6280027  6070246   5982791  7016299  10234692.9
## [119,] 14055289 14587423  12692709 10608853  10516327.0
## [120,] 13481684 12799042  12229099 11686237  10654543.9
## [121,]  3176056  3635439   4287889  3357436   2176451.0
## [122,] 11623796 11228789   9085790  6927540   5687574.4
## [123,] 17158214 17504734  11485397 10994093  20516446.6
## [124,] 13790062 14195136  14140147 12813256  12277172.5
## [125,] 15999435 15590680  13103457 10027085  10719457.0
## [126,] 18405108 19523455  19813753 16888218  14467534.5
## [127,]  3438350  3656847   3401731  2881689   2276447.2
## [128,] 14447159 13880159  14915356 17314220  16062391.3
## [129,] 23421189 23904075  22015337 28339136  30859817.9
## [130,]  3601804  3612994   3593729  4074972   4422205.8
## [131,] 15883373 15625369  15015719 15332151  15154700.4
## [132,]  7115179  6957953   8262346  9087808   7325969.0
## [133,]  9869247  9873965   9326437  8250975   8150710.8
## [134,]  7391449  6911454   5779474  5048996   6663735.4
## [135,]  8246717  8012385   7783581  8050004   7569341.6
## [136,]  5513675  5368902   4569076  4605169   5859018.1
## [137,] 21798607 22580806  18039754 14348166  22810663.4
## [138,] 32551276 31775011  26152601 45416758  28527002.0
## [139,]  5504825  5649284   4586064  3353161   2617499.9
## [140,] 13216374 13750814  14018692 12292531  10190646.9
## [141,] 16442078 16930834  12204637  9263823  21005026.4
## [142,] 17493910 18382479  17563945 14033677  11393794.1
## [143,] 17139454 16583219  14829447 15680730  15867550.2
## [144,]  6066097  6432574   6361253  5576371   5832639.8
## [145,]  9593870  9291252   6592373  8940404  10678772.8
## [146,]  8582914  8559710   9090967  7971715   7361326.9
## [147,]  5023267  4894388   3897639  2782509   4160503.8
## [148,]  7360544  7223918   8317896 10384498   8654064.9
## [149,] 23677246 25588091  26414128 22823254  19305015.3
## [150,] 12568654 13015748  12093878  9335399   7258293.3
## [151,] 24030096 26291851  26712723 22884921  19432069.5
## [152,]  3374211  3110088   2315232  1898651   2727325.6
## [153,]  6858156  8539523   8843781  9333254   6808412.1
## [154,]  5972428  6006592   5451270  4629216   4426217.1
## [155,] 11050248 10431830   9435706 11201435  11930587.9
```

```
## [156,]  6535265  6606841   5623409  3829622   2698264.2
## [157,] 20186804 21622021  16811681 11781324  19069880.3
## [158,] 16432579 15913628  13653480 15564581  19357194.9
## [159,] 14032977 14486917  14300259 12773334  11250221.7
## [160,] 15405778 16469932  15618548 11818666   9286642.2
## [161,] 12597912 12329683  12881223 14613088  12933117.1
## [162,]  9051269  8946170   8257776  7014403   6922506.9
## [163,] 12391986 12164096  10555500  8667007   8610494.5
## [164,] 14040223 14541250   8865833  6885098  14541333.9
## [165,] 15940748 16064746  14141133 15921032  17041043.9
## [166,] 34962262 35806895  33400614 28288581  25046619.8
## [167,]  2861436  3230042   4922014  4819179   3417580.5
## [168,] 14484215 14536674  14792377 14417606  12258545.0
## [169,]  5982910  5416979   4691910  4193886   5509258.8
## [170,]  6453898  6343623   4950044  3885431   4974491.3
## [171,] 19746585 19998169  18694775 18415763  16478655.6
## [172,]  6361545  6144609   7447402  8917342   8493115.5
## [173,] 16947091 19004406  19062897 15368605  12739111.2
## [174,] 10314992 10422833   8694662  5804145   6041477.5
## [175,] 11353396 11182499  11489720 11747542  11153541.3
## [176,]  9691832  9189092   8365943  9550856  10274781.2
## [177,]  8825759  9124479   8235211  6875532   7073856.6
## [178,] 11512006 11951601   7746324  5628340  14368600.4
## [179,] 18274361 19028180  18106463 15168225  13767740.2
## [180,] 11454399 11186523   9538084  9407548   9513813.0
## [181,]  7899709  7419233   6808821  7288033   7724722.5
## [182,] 18387288 19233338  18081403 14211836  11545131.7
## [183,] 15131942 14666460  13654070 13996242  14685625.9
## [184,] 16887238 16558491  11602086 15833098  55227882.7
## [185,] 10090338  9890245   8120938  6519597   8326892.8
## [186,] 13972644 14676146  14946845 13726556  12814909.9
## [187,] 10896720 10631667   8343426  6067603   5845303.8
## [188,] 24478883 24429582  25746363 25206362  22592240.1
## [189,]  2921176  3113322   3366603  2737662   1770260.4
## [190,]  5572694  5175865   5134602  5728654   6081658.2
## [191,]  6110547  6935967   7352451  7146039   5407873.4
## [192,]  6598310  6560973   7202797  8330074   7361605.9
## [193,] 24969691 25589172  24074411 20327365  19033734.3
## [194,] 14112554 15464603  10717510 14069341  83777371.3
## [195,]  8208341  7673545   6574483  5928418   5645456.2
## [196,]  3474039  3480293   3565656  3033403   2940147.7
## [197,] 17717094 18134246  19266228 18005293  15950905.9
## [198,] 20585719 20436746  20536066 19369582  16728266.9
## [199,]  9216193  9130152   7457255  6299754   8566331.0
## [200,]  8368087  9296440   8546554  8773116   8306140.2
```

```
## Prediction error PE for the different knots fitting
PE2<- cbind(knots = 1:ncol(mse), mse = colMeans(mse))
PE2
```

```
##      knots      mse
## [1,]     1 12530675
## [2,]     2 12815116
## [3,]     3 14074012
## [4,]     4 11332143
## [5,]     5 16429562
```

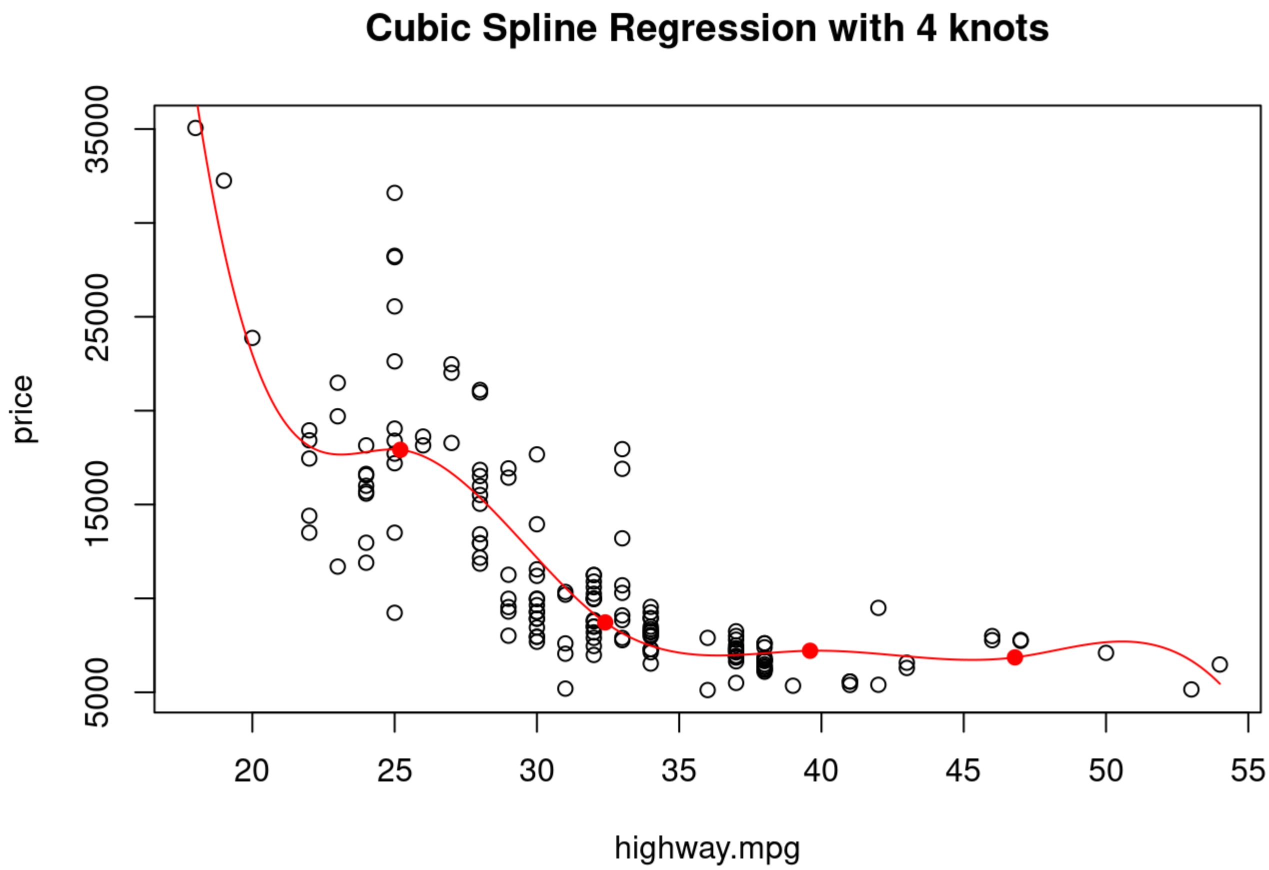
```
## min PE - the best model has 4 knots, which gives minimum of prediction error.
PE2[which.min(PE2[,2])]
```

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

9, Find your final model and show it in a scatter plot of the two variables.

Best fitting is found with 4 knots, therefore underlying graph shows data with 4 knots fitted cubic spline curve.

```
rg = range(auto$highway.mpg)
knots4 = seq(rg[1], rg[2], length= 6)[-c(1,6)]
rhw4 = lm(price ~ bs(highway.mpg, knots = knots4, degree=3), data = auto)
yhathw4 = predict(rhw4, newx)
with(auto, plot(highway.mpg, price, main = "Cubic Spline Regression with 4 knots"))
lines(newx$highway.mpg, yhathw4, col="red")
points(knots4, predict(rhw4, data.frame(highway.mpg = knots4)), col="red", pch=19)
```



Jackknifing and Parallel Computing

10, Use the Jackknifing technique (with a 90% for training and 10% for testing) to find an appropriate degree for polynomial regression as in Tasks 1-3. Use R = 200 as the number of repetitions.

results shows the third degree polynomial fitting gives the minimun PE value, thus the best fitting.

Show the curve of the selected model in a scatter plot of the data.

```r
newx<- data.frame(highway.mpg = seq(min(auto$highway.mpg), max(auto$highway.mpg), by=0.1))
set.seed(189)
R = 200 # shuffle times
n = nrow(auto) # sample size
k = round(n*0.1) # test size (10% of sample size)
p = 5 # degrees of polynomial regression

mse = matrix(nrow=R, ncol=p)  # mse size (repetition no * degrees of polynomial)
for(j in 1:R) {
  index = sample(n, k)
  test = auto[index,]
  train = auto[-index,]
  for(i in 1:p) {
  r.poly<- lm(price ~ poly(highway.mpg,degree = i,raw = TRUE),data=train)
  pred.poly = predict(r.poly, test)
  mse[j,i] = mean((test$price - pred.poly)^2 )
  }
}
## MSE
mse
```

```
##           [,1]     [,2]     [,3]     [,4]     [,5]
##  [1,]  4695952  5939801  6243518  6395197  5754659
##  [2,]  8951108  4314566  3892691  3830153  4809189
##  [3,] 24791254 14427535 13703810 19248169 16392869
##  [4,] 12632353 15640450 16049716 16475012 15215512
##  [5,] 32604228 21980064 21547327 21497850 25954524
##  [6,] 13097838 11047079 11086457 10985037 10270768
##  [7,] 10744979  6625246  6574272  6404616  6557046
##  [8,] 27324853 14415477 13965111 19126372 15797783
##  [9,] 30458199 11297261 12763173 41112310 63038049
## [10,] 10937341  8312322  9236103  9283986  8560293
## [11,]  9590528  5134226  5395387  5192295  5299523
## [12,] 31461313 24468793 25068810 25082559 26332272
## [13,]  6412670  8361705  8824631  9044904  7897389
## [14,] 11098350 12205299 13581219 13396729 12618657
## [15,]  9662125  7503048  7508947  7459212  7643562
## [16,]  8410617  7971230  8006377  8405348  7443291
## [17,] 19261734 11713490 11868173 11666270 13663947
## [18,] 11313907 10118172  9521146 10060762  9510111
## [19,] 11939226  3541078  2787105  2572823  5627369
## [20,] 20262437 14745722 14168515 13981757 15396866
## [21,] 12544559  6362700  5706917  5683460  7312448
## [22,] 10683670 11823537 13538297 13476143 12360993
## [23,]  9475754  3516238  2745824  2681585  5015868
## [24,]  9248035  6344938  6408045  6241699  6059938
## [25,] 18150085 16792893 16973981 16899526 17252260
## [26,] 27422803 24471217 25094147 25027330 24723233
## [27,] 10163776  7988407  8191441  8145067  7927588
## [28,] 25091211  3103825  2466599  2251365  6249438
## [29,] 33025258 25393404 26040570 26140732 25024674
## [30,] 35777181 31696274 34244514 34468308 34599937
## [31,]  6511982  1114289  1031012  1018488  1631540
## [32,] 16954484  8628005  8469706  8307805  9163275
## [33,] 13475071 10886347 10969394 10852212 10792045
## [34,]  9699868  8593930  8563261  8593356  8508207
## [35,] 28696394 15263813 14560407 19596479 16647900
## [36,] 25857919 10636416  9874951 12783608 10048265
## [37,] 17275522 13988112 14893033 14725133 14493371
## [38,] 11117688 10506825 10620309 10773250 10541117
## [39,] 12257026 12261181 12846158 12774714 12152172
## [40,] 16393648 14812277 15098180 14913258 15306657
## [41,]  7933005  7322736  7374731  7386064  7293705
## [42,] 14625621  8225545  7685099  7483391  7531774
## [43,] 13531992  9225221  8517521  8403387  8382090
## [44,] 15117806 13594014 13759416 13640469 13716453
## [45,] 10516631  5579537  4889207  4681725  4470260
## [46,] 36082919 18515520 17588006 17778803 18827286
## [47,] 19119199 12523178 11909554 13290163 12185392
## [48,] 12104249 11428415 11912590 11928018 11293611
## [49,] 10129752 15038983 15825643 16270800 14848934
## [50,]  3042959  3975315  4333112  4489927  3742973
## [51,] 24387585 15335958 14679412 17114522 14869023
```

```
## [52,]  9029385  6130495  5872361  5852483  6385931
## [53,]  4943344  3270080  3431687  3440459  3198164
## [54,] 13542946  8915638  8511597  8363597  9499508
## [55,] 20272149 17738182 18519455 18325384 17966530
## [56,] 38830272 31535757 31444758 31528415 33372347
## [57,] 22311116 20988060 21428077 21259030 21481249
## [58,] 11462942  6808724  6326770  6244624  7262398
## [59,] 26462330 11540162 10809435 14204999 11172177
## [60,] 26908189 13492224 12694933 16205375 13042923
## [61,] 20067165 14586204 14472150 14273799 16418095
## [62,]  7579357  2610741  2722750  2555888  2515954
## [63,] 12694224 11922459 11944761 11920377 11091226
## [64,] 29113445 14263590 13458569 16504260 13334977
## [65,] 37230382 29552532 30064205 30158513 33224108
## [66,] 26448630 26947110 28636699 28449139 27568190
## [67,] 12863557  6720789  6160939  5967898  6099105
## [68,] 22816774 15820098 16153066 16118303 17204899
## [69,] 13065002  6974697  6343634  6116549  6840113
## [70,]  3864583  2131998  2272393  2236502  1930233
## [71,]  9760510  9755238 10952440 10770879 10046008
## [72,] 24100204  7510659  7664264 13017717  9172497
## [73,]  8165156  6671345  6816958  6789800  6180523
## [74,] 14889200 12900553 13343702 13169596 12992826
## [75,] 19001230 14733741 15256655 15071181 15107586
## [76,] 38631723 24892865 24029883 26302908 23433101
## [77,]  9518993  9066843  8918414  9028138  8866927
## [78,] 18123043  7169869  6439252  7374727  6150824
## [79,] 10420464  8228480  8367060  8256142  7813935
## [80,] 21312371 13159031 12457054 13373619 12477562
## [81,] 14276531 11059632 10377687 10471812 11194834
## [82,] 12304846 11068583 11353336 11241221 10643468
## [83,] 18204501 20286472 21696274 21630712 20137486
## [84,] 14346308  3880687  3255295  3070380  3094999
## [85,]  8804806 11787742 12264039 12782552 11468602
## [86,]  9337924 11225383 11479165 11767505 11270246
## [87,]  7373686 10439174 11585405 11588722 10389445
## [88,] 28406128 19040264 18816139 19088208 17969888
## [89,] 16346399 16065425 16871458 16755937 16139879
## [90,]  5389328  1898530  1786892  1682223  1526189
## [91,] 20461814  7152270  6558377 10992811  7841777
## [92,] 29053423  8709753  8565808 13580940 10991148
## [93,] 20374322 11711471 11241083 11016351 13941703
## [94,] 33199853 33830211 34892329 34797009 34717059
## [95,] 11103709  6126205  5575198  5352675  5453581
## [96,] 10568172 11752025 12221860 12234794 11345371
## [97,]  8039104  6937306  7119890  7149175  6731034
## [98,] 10408955 10974953 10868745 11130558 10784448
## [99,] 17820372 14127791 14762250 14566668 14749643
## [100,] 29258396 27399374 28352599 28165509 27123012
## [101,] 16284320 17140760 17803953 17771082 17267288
## [102,] 10942972  7620542  7684735  7561908  7307070
## [103,] 22309981 14093206 13538648 14169256 12913510
```

```
## [104,] 29582201 26560269 27897811 27827021 28584636
## [105,] 11116179  6041827  5644896  5426797  5892567
## [106,] 29214725  8941889  8895774 13664243 11327911
## [107,] 30989366 16441642 15607972 19670242 16506266
## [108,] 12130269  9036707  9379324  9240568  8562900
## [109,] 16390169 15582651 15848776 15764713 16016944
## [110,] 27005091 16555989 16262662 16117862 17793155
## [111,] 22304569 15474494 14786725 16756507 15480602
## [112,] 13783869 13564342 14182450 14339451 12777748
## [113,] 21040016  5300417  4618893  5161835  6400789
## [114,] 15848289 10510012 10379024 10182471 11017641
## [115,] 11538077  3777391  3117812  2907223  3232418
## [116,]  5231535  3526927  3187559  3141501  3298072
## [117,]  9776066  6514616  6478957  6345381  6407177
## [118,] 12457399  5648688  5054179  4861448  4827737
## [119,]  9550204  7710655  7379043  7606790  7625508
## [120,] 12482609 11625734 11880234 11888667 11069454
## [121,] 10477973 13744501 15159321 15162496 13890871
## [122,]  5332102  4903945  5402999  5339297  4881395
## [123,] 39860186 27190663 26357591 29524601 26726751
## [124,] 31825791 11802394 13587902 41093412 62495727
## [125,] 13517078  9797573  9749971  9575663  9689667
## [126,] 19280985 10269269  9638531  9395779 13323720
## [127,] 18860071 10387699  9939178  9716296 12998651
## [128,] 26257136 15159203 14753410 14745467 17522055
## [129,] 12658678  8426846  8615810  8426391  8382512
## [130,] 28698101  9150629  9080561 12755812 10530897
## [131,] 11020429 11217329 11098116 11360106 10745770
## [132,]  4143431  5533506  6512392  6476601  5520064
## [133,] 27028154 18598882 17858771 23365582 20860227
## [134,] 20724102 19345975 19449329 19445291 19699254
## [135,] 14712814  6775094  6536942  6326296  7754587
## [136,]  9055956  6116968  6373523  6237498  5984640
## [137,] 20068693  8152833  7425279 11722502  9134522
## [138,] 12688209  7646070  6921345  6998122  8205821
## [139,] 11081504  8355902  8404357  8282822  8532136
## [140,]  8843556 12146580 12329683 12464448 11501224
## [141,] 26275610 17886560 17510877 17384896 19575912
## [142,] 13931476  7411418  7451317  7271193  7766431
## [143,] 14843897 13453337 13185065 13055799 13327963
## [144,] 16388085 10123393 10015007  9822764 11033855
## [145,] 33202125 27680930 28179967 28030920 29370351
## [146,] 11103165  7295724  6767013  6605626  6336905
## [147,] 53966847 39934898 39355587 39238857 40231442
## [148,] 15414348 10754955 10402632 10260357 10831197
## [149,] 34804091 26796290 27040235 26851737 28477978
## [150,] 14676571  9220535  9071444  8877710  9753921
## [151,] 24411265 13994741 13254210 16971161 14522673
## [152,] 25576800 10706882  9999723 13089300 10073220
## [153,]  9053126  5335217  4965445  4887143  5001060
## [154,] 11093846 10751252 10877774 10960384 10867973
## [155,] 11257802 12621410 12370127 12989059 12278053
```

```
## [156,]  8372537  8110635  8138113  8054064  7669000
## [157,] 14984148  9511375  9385662  9184553  9587929
## [158,] 36506523 23298668 22499390 24184526 21505812
## [159,]  8808822  5341882  4779140  4596881  4412425
## [160,] 28548767 16164010 15319615 17592088 14904685
## [161,] 29713073 15886372 15228694 17949221 14867836
## [162,] 23118850 19343522 19425231 19249533 20118273
## [163,] 20611654  7930049  7370690 11875756  9076622
## [164,] 18994929 10082135  9373013 13354558 11215274
## [165,] 11708600  7865555  7240883  7085362  7461052
## [166,]  8208230  5780963  5613671  5611430  5247115
## [167,] 24287521 16835977 16797907 16758136 18737984
## [168,] 15344609 11068600 10348542 10223582 10666541
## [169,] 11759278 12060941 12465963 12399405 11786594
## [170,]  4075983  2636353  2632366  2637009  2515038
## [171,] 18874610 12987502 12779674 13381503 12030833
## [172,] 10263140 11608396 11794772 11979155 11566896
## [173,] 12328343 16017504 16260569 16426923 15477631
## [174,] 14741829 14117860 13851805 14205371 13434677
## [175,] 29682217 23237264 23370263 23239864 25022997
## [176,] 26912316 22589956 23519209 23426596 24558732
## [177,]  8411357  9096720  9149935  9157001  9007350
## [178,] 20297660  9345888  8736259 14137833 11034209
## [179,]  8770976  4828505  4858998  4665294  4702579
## [180,] 12375143 11420140 11142707 11434243 10342972
## [181,] 26457535  8785434  8138732  7945933  9175742
## [182,]  9569189  7466560  7818953  7660751  7790605
## [183,]  9171973  7470137  7468030  7307786  7422835
## [184,] 13284540 10875615 11480992 11281159 11500407
## [185,] 10479409  9467843  9474348  9483262  9354356
## [186,]  6864170  6175238  6134966  6208100  5832044
## [187,] 22261633 15288845 14520587 14344738 17315114
## [188,] 13595132  7007278  6320277  6251921  8415108
## [189,]  8878280  2418066  2292586  2242947  3210663
## [190,] 16387069 10425921 10510254 10334980 10961200
## [191,] 30430238 15730676 15124918 18514153 15244725
## [192,] 23792208 14035381 13411779 13222443 14153361
## [193,] 22895627 23356360 24405189 24291999 23853834
## [194,] 12930939  6316996  5638919  5527782  7932767
## [195,] 31534637 16631795 16016247 18397928 15404828
## [196,] 12579179  7161568  7306966  7298844  6996709
## [197,] 19585152  8745264  8095542  9385440  8732854
## [198,] 20089089 17381731 18140334 17941402 18430765
## [199,]  8044891  5704522  5755484  5640170  5883873
## [200,] 15132738 12081626 11957874 11938665 12128246
```

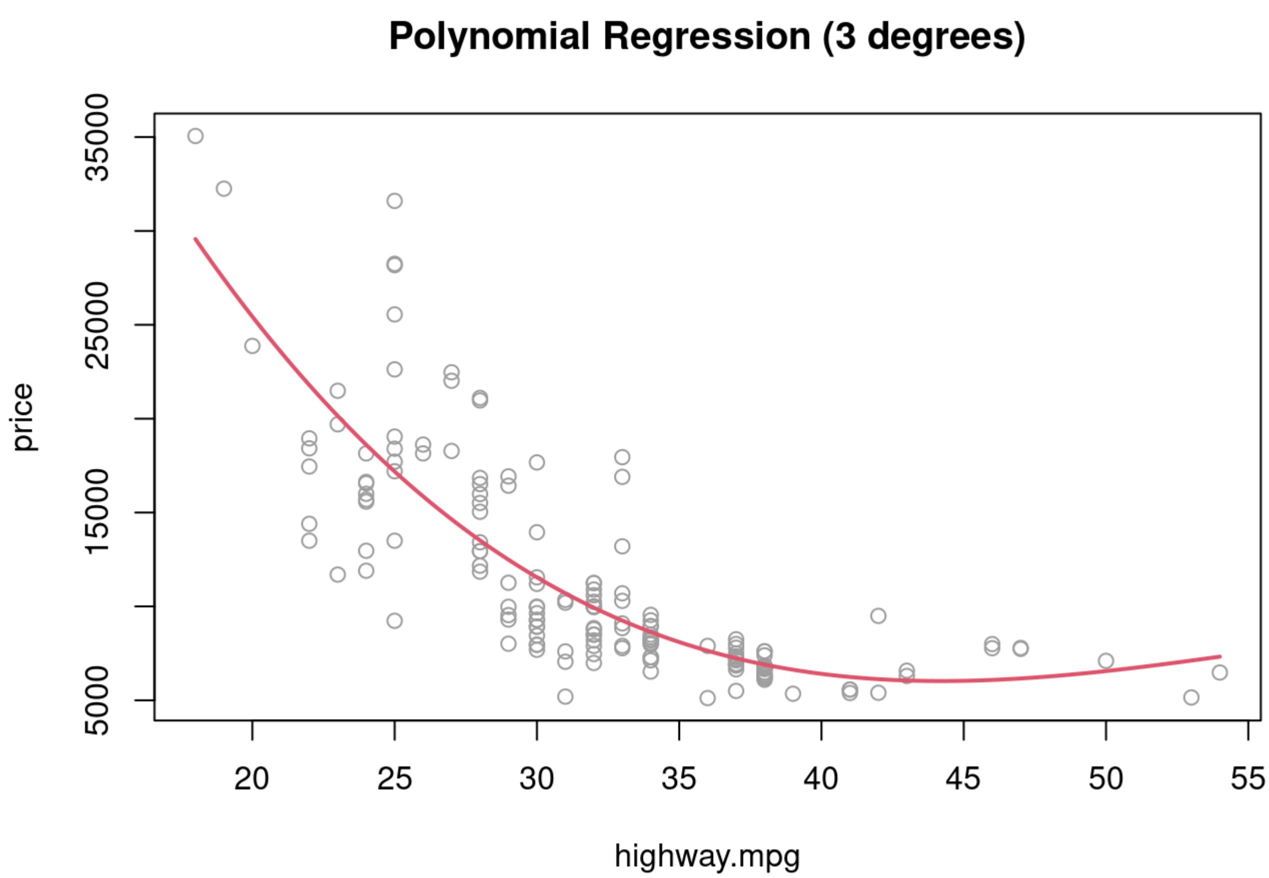
```
## PEs for 1 - 5 degrees
PE3<- cbind(degree = 1:ncol(mse), mse = colMeans(mse))
PE3
```

```
##      degree      mse
## [1,]      1 17271743
## [2,]      2 11981374
## [3,]      3 11955641
## [4,]      4 12770674
## [5,]      5 12774749
```

```
## Degree based on minimum PEs
PE3[which.min(colMeans(mse))]
```

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

```
## the 3rd degree polynomial regressions gives the minimum prediction error.
## Plotting based on 3rd degree polynomial regressions.
newx<- data.frame(highway.mpg = seq(min(auto$highway.mpg), max(auto$highway.mpg), by=0.1))
with(auto,plot(highway.mpg,price,pch=21,col=8, main="Polynomial Regression (3 degrees)"))
pred3 = predict(polyn3,newx)
lines(newx$highway.mpg,pred3,lwd = 2, col = 2)
```



11, Rewrite/reorganise your code so that each repetition can be carried out independently. Perform the Jackknifing selection of the polynomial degree using parallel computing. Compare the timings, when 1, 5, 10 or 20 cores are used.

```
library(parallel)
## function for reshuffle of data, R shuffle times; K fold of sampling (10 means 10% test, 90% t
rain); p degree of polynomial regressions
shuffle = function(R,K=10,p=5) {
   set.seed(189)
   k = round(n*(1/K))
   n = nrow(auto)
   index = sample(n, k)
   test = auto[index,]
   train = auto[-index,]
   for(i in 1:p) { ## calculate mse based on each train / test dataset
   r.poly<- lm(price ~ poly(highway.mpg,degree = i,raw = TRUE),data=train)
   pred.poly = predict(r.poly, test)
   mse[i] = mean((test$price - pred.poly)^2 )

  }
}


system.time({
    mclapply(1:200, function(R) shuffle(R ,K=10,p=5), mc.cores=1)
})
```

```
##    user  system elapsed
##   1.531   0.000   1.532
```

```
## check results
colMeans(mse)
```

```
## [1] 17271743 11981374 11955641 12770674 12774749
```

```
system.time({
    mclapply(1:200, function(R) shuffle(R ,K=10,p=5), mc.cores=5)
})
```

```
##    user  system elapsed
##   1.754   0.224   0.512
```

```
system.time({
    mclapply(1:200, function(R) shuffle(R ,K=10,p=5), mc.cores=10)
})
```

```
##    user  system elapsed
##   0.585   0.204   0.275
```

```
system.time({
    mclapply(1:200, function(R) shuffle(R ,K=10,p=5), mc.cores=20)
})
```

```
##    user  system elapsed
##   1.980   0.792   0.240
```

12, For results to be reproducible, it is better to use random seeds. Investigate and demonstrate how this can be achieved when mclapply() is used.

```
system.time({
    mclapply(1:200, function(R) shuffle(R ,K=10,p=5),mc.set.seed = TRUE, mc.cores=5)
})
```

```
##    user  system elapsed
##   1.800   0.224   0.518
```

```
## check results
colMeans(mse)
```

```
## [1] 17271743 11981374 11955641 12770674 12774749
```

Summary

In this lab we have learn different kinds of non-linear regression methods, including polynomial, smooth spline, cubic spline and GAM. In addition, we learn different sampling way, cross validation, Jackkniffting etc.. The way to find best fitting is to compute prediction errors based mse, the minimum shows the best fitting. As shown in Q8, the best polynomial fitting degree is three, as well as in Q7, the most optimum numbers of knots for cubic spline regression is four. Cross validation which has equivallent computing capacity, is a more efficient resampling technique than Jackknifing.