Assignment2

10878862

26/04/2022

Prepare for R markdown

Here, let's turn off the warning and message for later chunks, because during model fitting there are lots of messages about fitting errors, so to make it clear and readable, I set them both as false to ignore this information. It is worth noticing that I have handled all the warnings to make sure all the processing is correct, and I would mention some warning information when necessary.

Load all packages

```
library(ISLR2)
library(tidyverse)
library(GGally)
library(leaps)
library(car) # This is for load VIF function to
library(effects)
library(caret)
library(splines)
library(gam)
library(broom)
```

Regression question

Read in dataset

Load, go through and check our data (to make sure there is no observation containing NaN which might cause unexepcted problems)

```
RegressionData <- Boston[,c(5,7,8,12,13)]
str(RegressionData)
```

```
## 'data.frame':    506 obs. of  5 variables:
##  $ nox  : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ age  : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis  : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ lstat: num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

```
missingobservation <- RegressionData[apply(RegressionData, 1, function(x) sum(is.na(x))) == 1
]

sum(missingobservation)
```

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

Luckily, there is no missing data.

Construct Baseline model

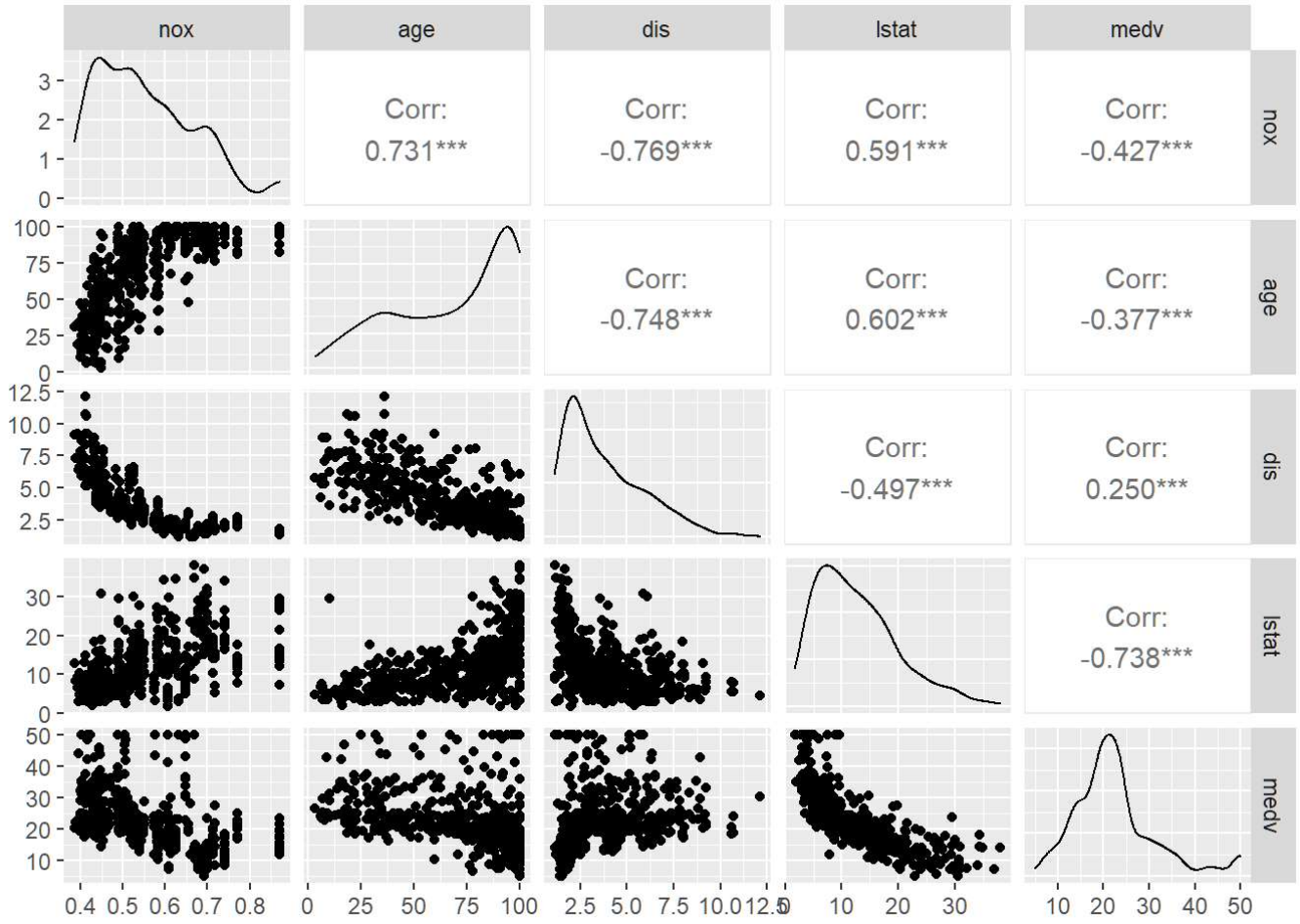
It is suggested we started from linear regression model because of the accessibility to interpretation and the rule of parsimony.

First, we can use correlation test to capture the linear relationship between predictors and dependent (response) variables. Although we won't make any decision based on this result due to the lack of reliable excluding criterion, at least we can get a intuition of our data, which is beneficial for us to make subjective choices later when necessary.

Choice of Baseline Model Variables

Here, we can use `ggpairs` function from `GGally` packages, which present the scatter plot, density plot and correlation coefficient directly and clearly for all pairs of variables.

```
ggpairs(RegressionData)
```



Through this plot, we could find that almost all the predictive variables are correlated with predicted variable - *nox*, however, the predicted variables are also correlated with each other and they are not normal-distributed, which might cause multilinearity and affect the accuracy of linear model.

Keep this information in mind and continue to construct our linear regression model.

Subset Selection

Now, let's conduct **subset selection** with function `regsubsets` , which return the best variables suggested to be used for n-varaible (n from 1 to 4 in this case) model, notably, "best" here is quantified using RSS. We can calculate the various fitting indices with `summary` function, and store them into *reg_summary*.

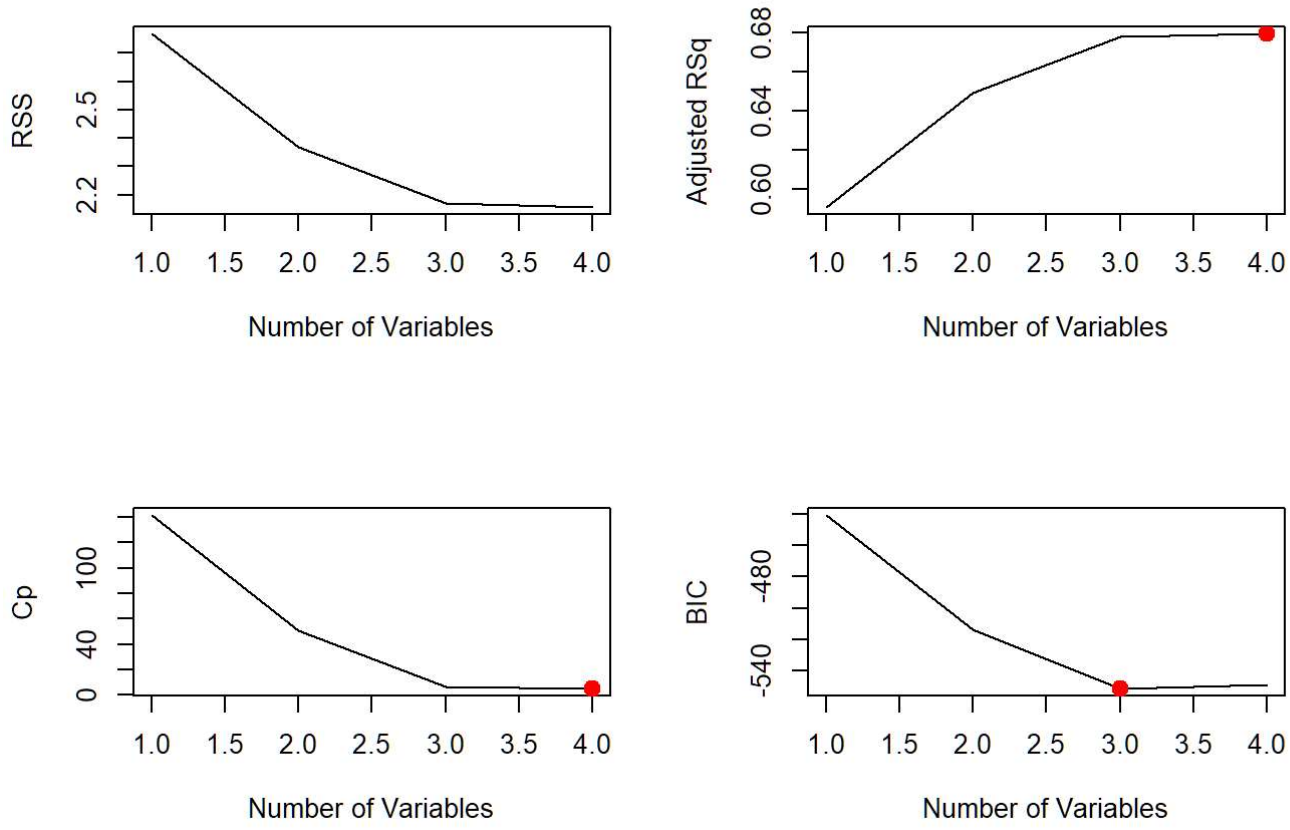
```
regfit_full = regsubsets(nox~., data = RegressionData, nvmax = 4)
reg_summary <- summary(regfit_full)
```

Besides using RSS, we could also check other parameters such as adjusted R2 , Cp, and BIC to compare these models.

```
reg_summary$rsq
```

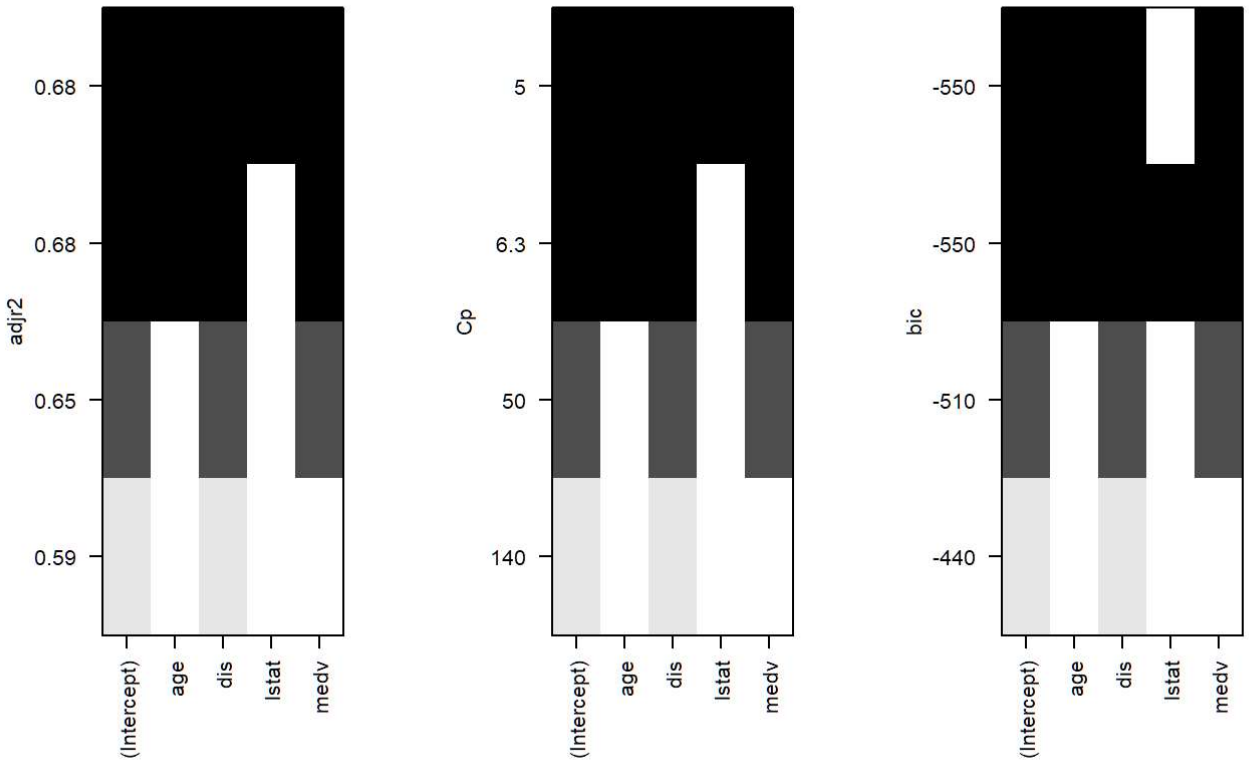
```
## [1] 0.5917150 0.6506535 0.6799426 0.6820447
```

```
# Set up a 2x2 grid so we can look at 4 plots at once
par(mfrow = c(2,2))
plot(reg_summary$rss, xlab = "Number of Variables", ylab = "RSS", type = "l")
plot(reg_summary$adjr2, xlab = "Number of Variables", ylab = "Adjusted RSq", type = "l")

# We will now plot a red dot to indicate the model with the largest adjusted R^2 statistic.

adj_r2_max = which.max(reg_summary$adjr2) # 11

points(adj_r2_max, reg_summary$adjr2[adj_r2_max], col ="red", cex = 2, pch = 20)

# We'll do the same for C_p and BIC.
plot(reg_summary$cp, xlab = "Number of Variables", ylab = "Cp", type = "l")
cp_min = which.min(reg_summary$cp) # 10
points(cp_min, reg_summary$cp[cp_min], col = "red", cex = 2, pch = 20)

plot(reg_summary$bic, xlab = "Number of Variables", ylab = "BIC", type = "l")
bic_min = which.min(reg_summary$bic) # 6
points(bic_min, reg_summary$bic[bic_min], col = "red", cex = 2, pch = 20)
```



Also, we could draw pictures to show which variable should be included in each model direcly.

```
par(mfrow = c(1,3))
plot(regfit_full, scale = "adjr2")

plot(regfit_full, scale = "Cp")

plot(regfit_full, scale = "bic")
```



Although **BIC** value suggested that the 3-variable model (*lstat* was excluded) was the best, the **Cp** and **RSq** suggested 4-variable model is still slightly better than 3-variable model. Because there are only 4 variables, it's not tricky to compare them. Therefore, we included everything in our model here and use a full 4-variable model as a starting point for our later analysis.

Assumption test 1

Now, we started to test the assumption of this model to see if there are some problems existing.

```
lm.fit = lm(nox~., data = RegressionData)
vif(lm.fit)
```

```
##      age      dis    lstat     medv
## 2.702579 2.344189 3.091143 2.284361
```

According to the VIF, we should have an intuitive feeling that multicollinearity is not a noticeable issue for this model.

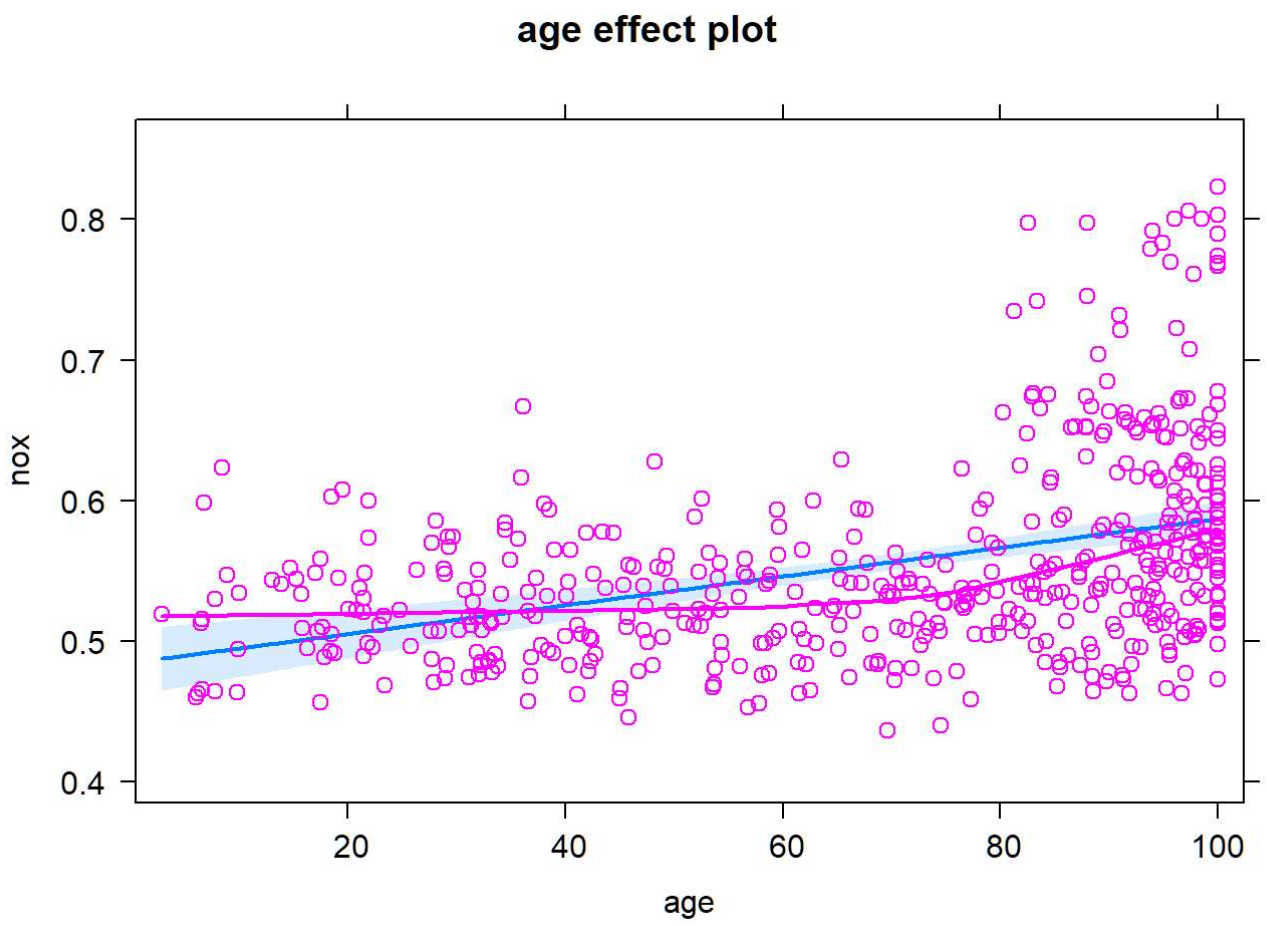
Model Improvement with Non-linear Approach

As previous scatter plot suggested, some predictor variables might has non-linear relationship with *nox*, here I work on each input variable one by one to see if using a non-linear method will improve the model.

Fitting Age

First, let's plot the residual of age in this multilinear model.

```
plot(effect("age", mod = lm.fit, partial.residuals = TRUE))
```



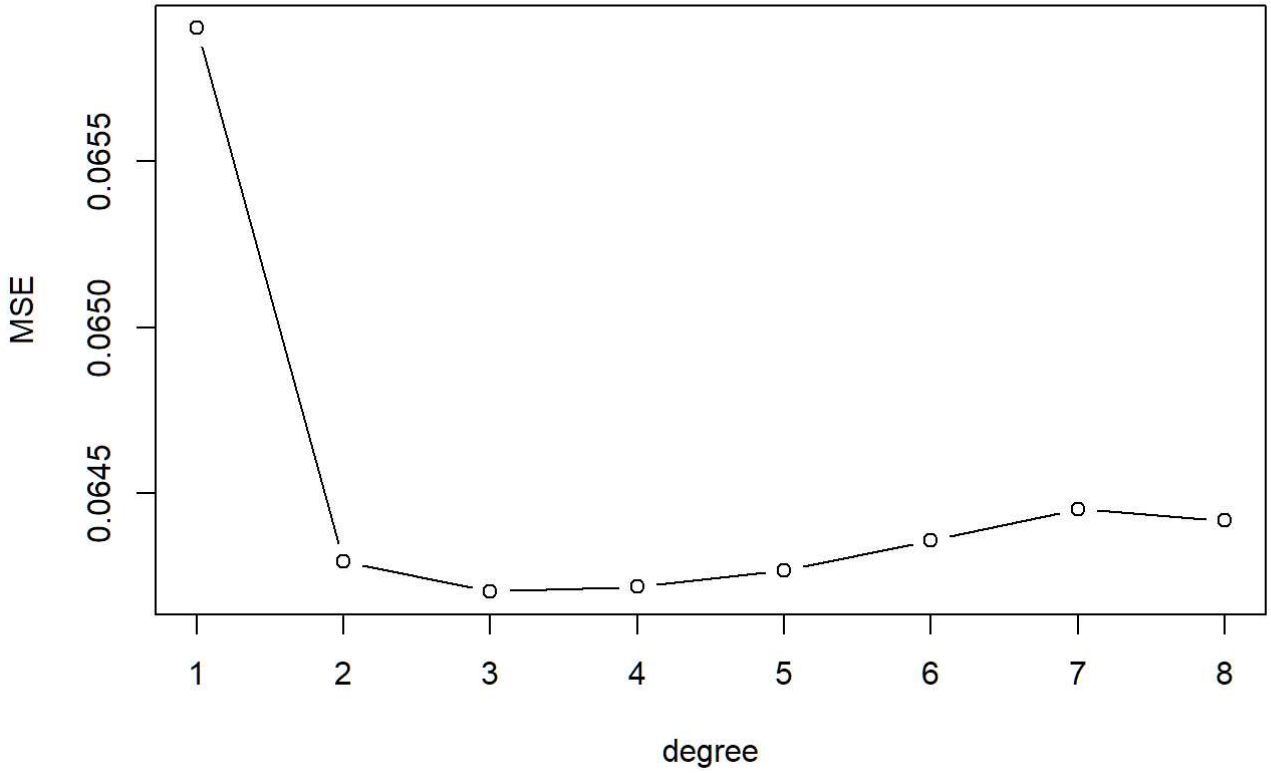
The blue line apparently differs from the pink line, which suggests that our model failed to capture the correct shape. Thus, we need to consider non-linear relationship between *age* and *nox*.

1.1.1 Refit *age* with Polynomial

Here, we use `train()` function within the loop to calculate the MSE value for each degree of polynomial model.

Noticeably, for all the cross-validation below, I always use LOOCV cross-validation because it is more stable and with lower variance, and it shows a replicable value for every run.

```
set.seed(1234)


train_control <- trainControl(method="LOOCV")

fit.age.1 <- train(nox ~ poly(age,1) + dis + lstat + medv,
                       data = RegressionData,
                       method = "lm", trControl = train_control)
fit.age.2 <- train(nox ~ poly(age,2) + dis + lstat + medv,
                       data = RegressionData,
                       method = "lm", trControl = train_control)
fit.age.3 <- train(nox ~ poly(age,3) + dis + lstat + medv,
                       data = RegressionData,
                       method = "lm", trControl = train_control)
fit.age.4 <- train(nox ~ poly(age,4) + dis + lstat + medv,
                       data = RegressionData,
                       method = "lm", trControl = train_control)
fit.age.5 <- train(nox ~ poly(age,5) + dis + lstat + medv,
                       data = RegressionData,
                       method = "lm", trControl = train_control)
fit.age.6 <- train(nox ~ poly(age,6) + dis + lstat + medv,
                       data = RegressionData,
                       method = "lm", trControl = train_control)
fit.age.7 <- train(nox ~ poly(age,7) + dis + lstat + medv,
                       data = RegressionData,
                       method = "lm", trControl = train_control)
fit.age.8 <- train(nox ~ poly(age,8) + dis + lstat + medv,
                       data = RegressionData,
                       method = "lm", trControl = train_control)
MSE <- rep(0,8)
MSE[1] <- fit.age.1$results$RMSE
MSE[2] <- fit.age.2$results$RMSE
MSE[3] <- fit.age.3$results$RMSE
MSE[4] <- fit.age.4$results$RMSE
MSE[5] <- fit.age.5$results$RMSE
MSE[6] <- fit.age.6$results$RMSE
MSE[7] <- fit.age.7$results$RMSE
MSE[8] <- fit.age.8$results$RMSE

plot(1:8, MSE, type = "b", xlab = "degree")
```




```
which.min(MSE)
```

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

```
MSE[which.min(MSE)]
```

```
## [1] 0.06420508
```

The result suggested that the third-degree polynomial with the smallest MSE of 0.0642 is the best.

Here, we refit the model with a third-degree polynomial and plot the effect again.

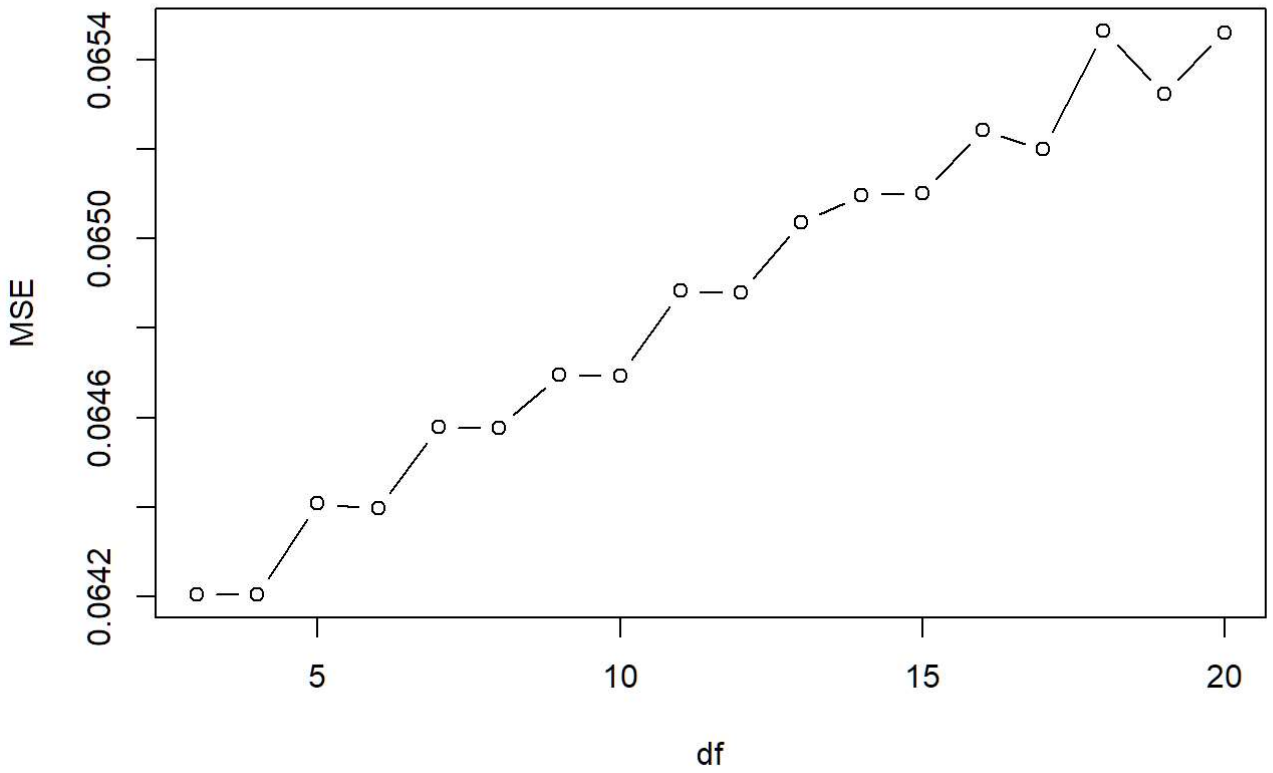
```
lm.fit.age3 = lm(nox ~ poly(age,3) + dis + lstat + medv,
                     data = RegressionData)

plot(effect("age", mod = lm.fit.age3, partial.residuals = TRUE))
```

Now the blue line is much closer to the pink line.

Except the polynomial model, we could also try to use spline to fit this model.

1.1.2 Refit *age* with Basis Spline

```
MSE <- rep(0,18)
for (i in 3:20){
  Data <- RegressionData
  Data$agespline <- bs(RegressionData$age, df = i)
  fit <- train(nox ~ agespline + dis + lstat + medv,
               data = Data,
               method = 'lm',
               trControl=train_control)
  MSE[i-2] <- fit$results$RMSE
}
plot(3:20, MSE, type='b', xlab = 'df')
```




```
dfn <- which.min(MSE) + 2
dfn
```

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

```
MSE[which.min(MSE)]
```

```
## [1] 0.06420433
```

The resultant plot showed that the 4th degrees of freedom produces the lowest MSE, which is 0.0642. Here, we refit age with 4th degree of basis spline and plot the residual.

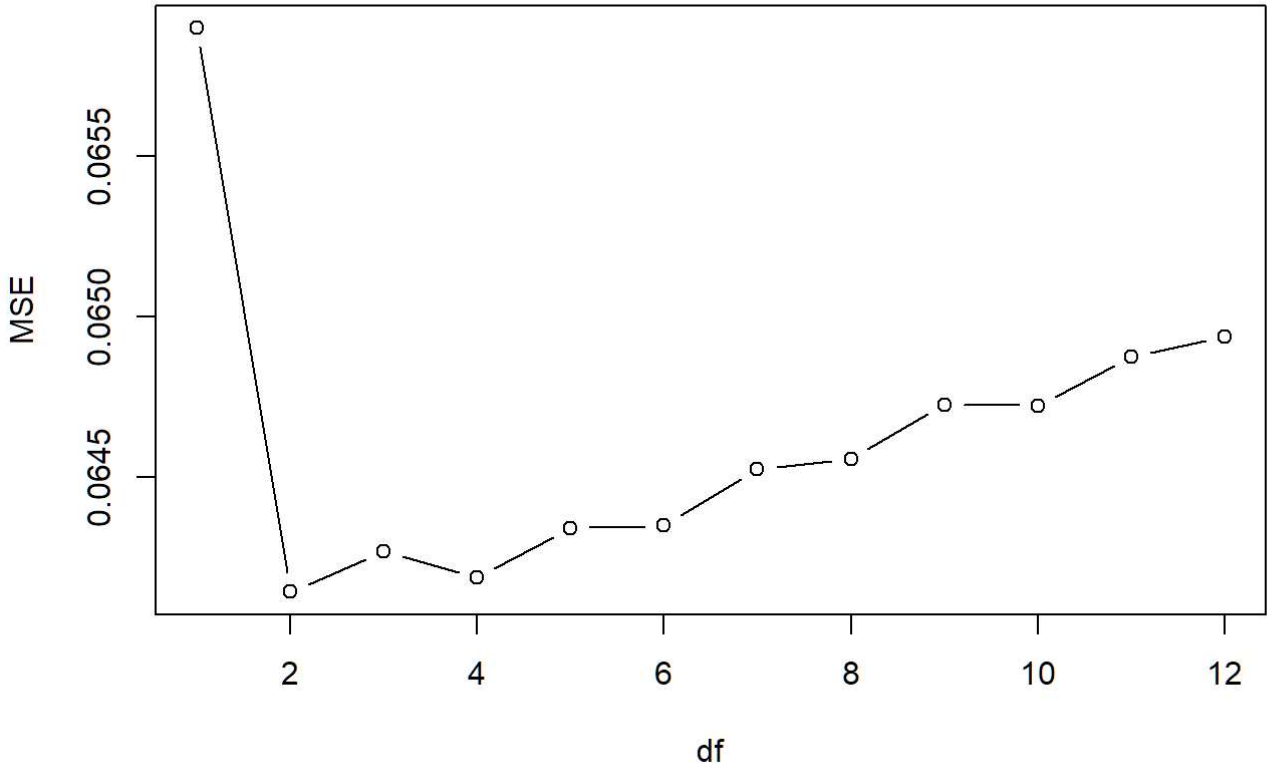
```
  lm.bs.age3 <- lm(nox ~ bs(age,df = 4) + dis + lstat + medv,
                   data = RegressionData)
  plot(effect("age", mod = lm.bs.age3, partial.residuals = TRUE))
```

The MSE for this is not different from third polynomial fitting. Let's continue to try the natural spline.

1.1.3 Refit *age* with Natural Spline

Test age with natural splines.

```
MSE <- rep(0,12)
for (i in 1:12){
  Data <- RegressionData
  Data$agens <- ns(RegressionData$age, df = i)
  fit <- train(nox ~ agens + dis + lstat + medv,
               data = Data,
               method = 'lm',
               trControl=train_control)
  MSE[i] <- fit$results$RMSE
}

plot(1:12, MSE, type='b', xlab = 'df')
```




```
which.min(MSE)
```

```
## [1] 2
```

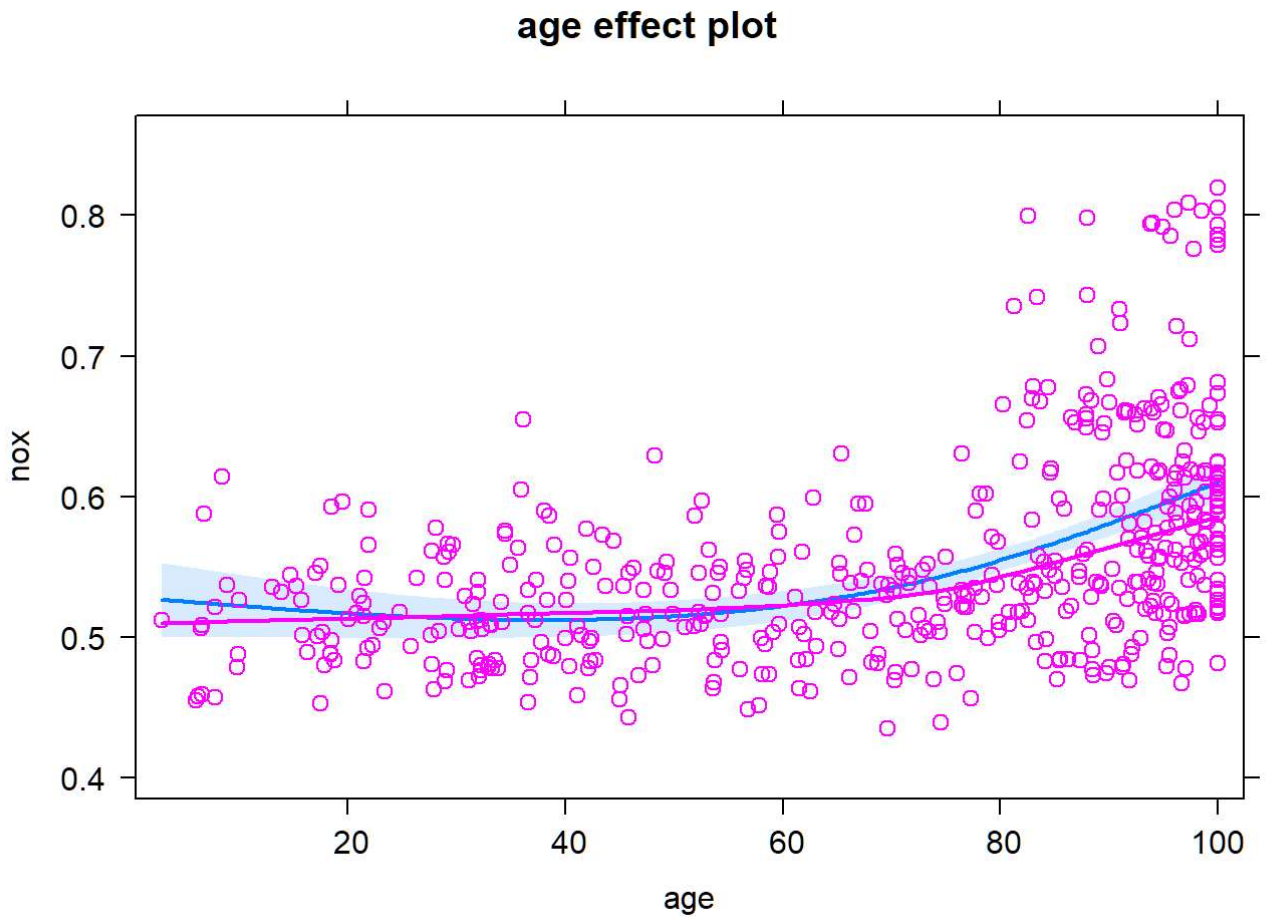
```
MSE[which.min(MSE)]
```

```
## [1] 0.06414247
```

The smallest MSE = 0.0641, which is from when df = 2.

Here, we refit age with 2 degree natural spline to inspect the pattern of residual.

```
lm.ns.age2 <- lm(nox ~ ns(age,df = 2) + dis + lstat + medv,
                 data = RegressionData)
plot(effect("age", mod = lm.ns.age2, partial.residuals = TRUE))
```



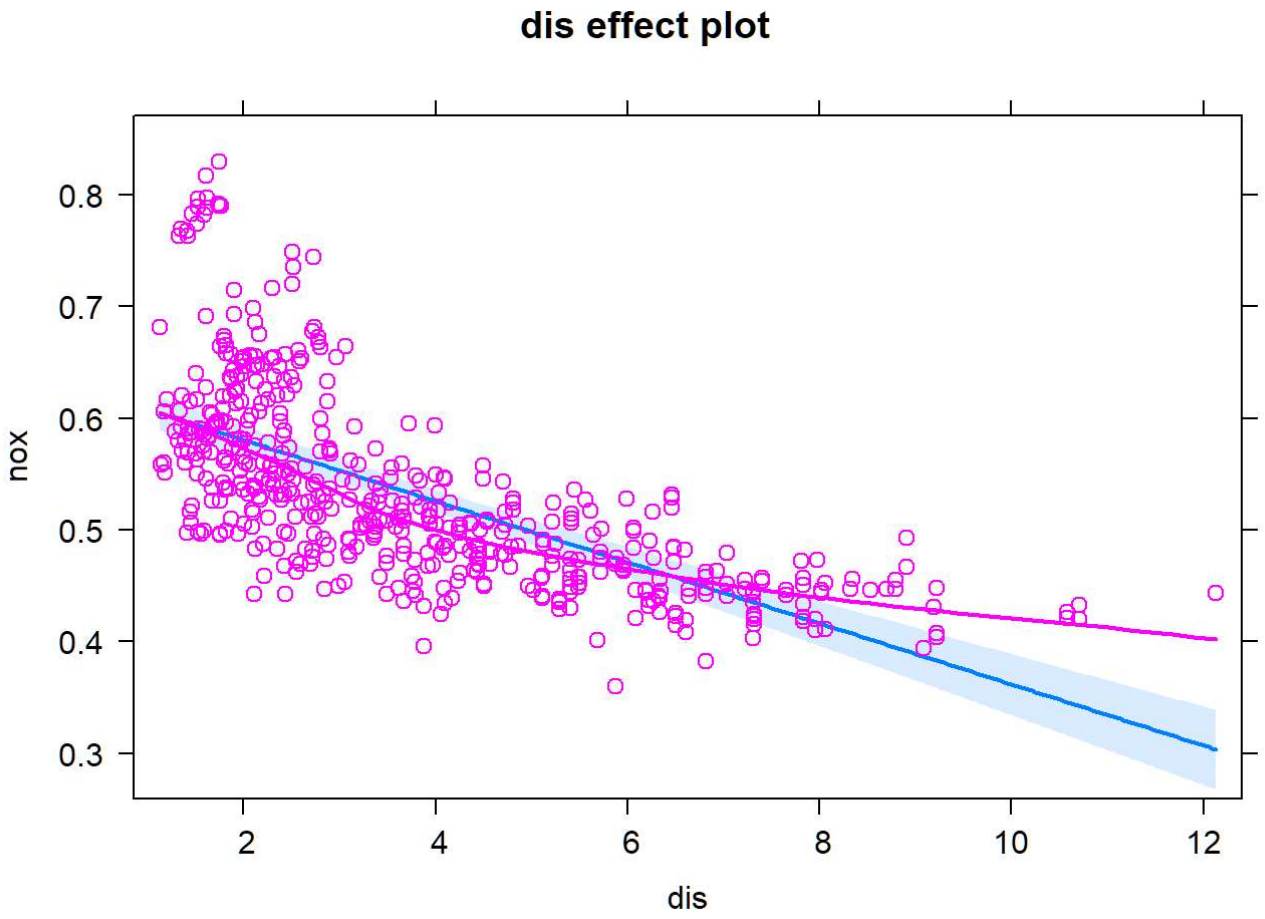
According to these four plots, we could see that the later three are all better than the first plot.

Although the natural spline has the smallest MSE, it's not that distinguishable and the residual of the cubic model almost overlaps with the fitted pink line (especially for the left-bottom part). I decided to put the third polynomial fitted age into the model.

As I have determined *age*, now, let's continue to do the same thing to *dis*, *lstat*, and *medv*.

Plot redisual for *dis*

```
fit.dis <- lm(nox ~ poly(age,3) + dis + lstat + medv,
                      data = RegressionData,
                      method = "lm")
plot(effect("dis", mod = fit.dis, partial.residuals = TRUE))
```



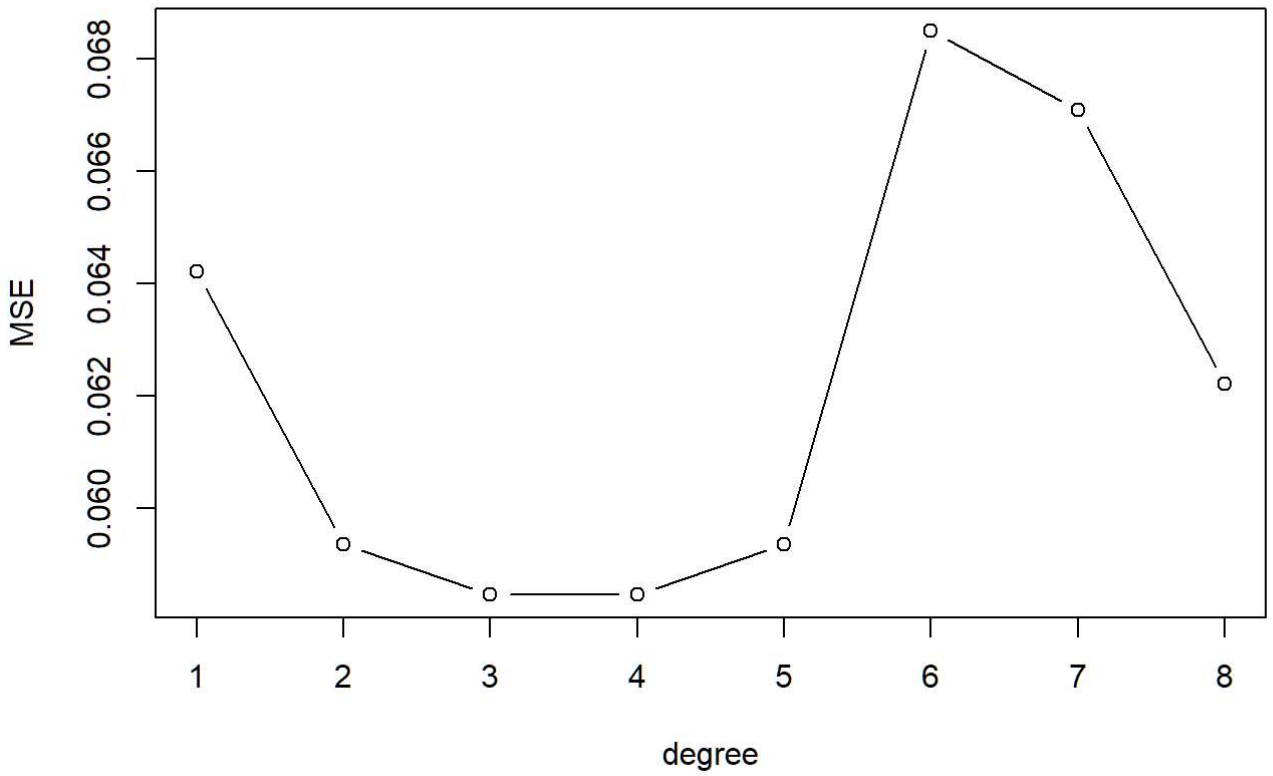
This plot shows terrible nonlinear problems.

Fitting *dis*

Because all the process is almost the same as the above processing, here I omit the narrative and just show the script and results.

1.2.1 Refit *dis* with Polynomial

```
set.seed(1234)

fit.dis.1 <- train(nox ~ poly(age,3) + poly(dis,1) + lstat + medv,
                      data = RegressionData,
                      method = "lm", trControl = train_control)
fit.dis.2 <- train(nox ~ poly(age,3) + poly(dis,2) + lstat + medv,
                      data = RegressionData,
                      method = "lm", trControl = train_control)
fit.dis.3 <- train(nox ~ poly(age,3) + poly(dis,3) + lstat + medv,
                      data = RegressionData,
                      method = "lm", trControl = train_control)
fit.dis.4 <- train(nox ~ poly(age,3) + poly(dis,4) + lstat + medv,
                      data = RegressionData,
                      method = "lm", trControl = train_control)
fit.dis.5 <- train(nox ~ poly(age,3) + poly(dis,5) + lstat + medv,
                      data = RegressionData,
                      method = "lm", trControl = train_control)
fit.dis.6 <- train(nox ~ poly(age,3) + poly(dis,6) + lstat + medv,
                      data = RegressionData,
                      method = "lm", trControl = train_control)
fit.dis.7 <- train(nox ~ poly(age,3) + poly(dis,7) + lstat + medv,
                      data = RegressionData,
                      method = "lm", trControl = train_control)
fit.dis.8 <- train(nox ~ poly(age,3) + poly(dis,8) + lstat + medv,
                      data = RegressionData,
                      method = "lm", trControl = train_control)
MSE <- rep(0,8)
MSE[1] <- fit.dis.1$results$RMSE
MSE[2] <- fit.dis.2$results$RMSE
MSE[3] <- fit.dis.3$results$RMSE
MSE[4] <- fit.dis.4$results$RMSE
MSE[5] <- fit.dis.5$results$RMSE
MSE[6] <- fit.dis.6$results$RMSE
MSE[7] <- fit.dis.7$results$RMSE
MSE[8] <- fit.dis.8$results$RMSE

plot(1:8, MSE, type = "b", xlab = "degree")
```




```
which.min(MSE)
```

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

```
MSE[which.min(MSE)]
```

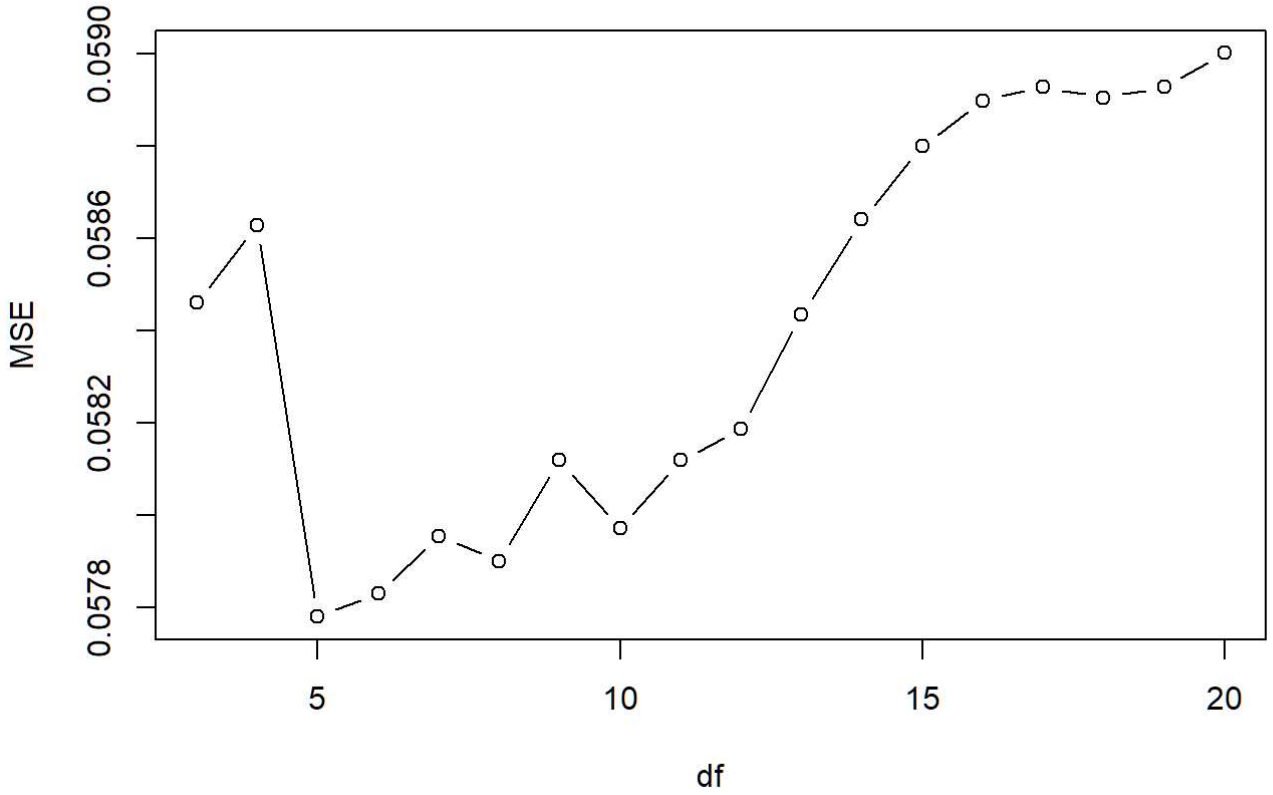
```
## [1] 0.05845656
```

The 4th polynomial fitting is the best for *dis* with MSE = 0.0585, third polynomial fitting is also good.

1.2.2 Refit *dis* with Basis Spline

The degree should beyond 3, so we set i from 3.

```
MSE <- rep(0,18)
for (i in 3:20){
  Data <- RegressionData
  Data$disspline <- bs(RegressionData$dis, df = i)
  fit <- train(nox ~ poly(age,3) + disspline + lstat + medv,
               data = Data,
               method = 'lm',
               trControl=train_control)
  MSE[i-2] <- fit$results$RMSE
}
plot(3:20, MSE, type='b', xlab = 'df')
```




```
dfn <- which.min(MSE) + 2
dfn
```

```
## [1] 5
```

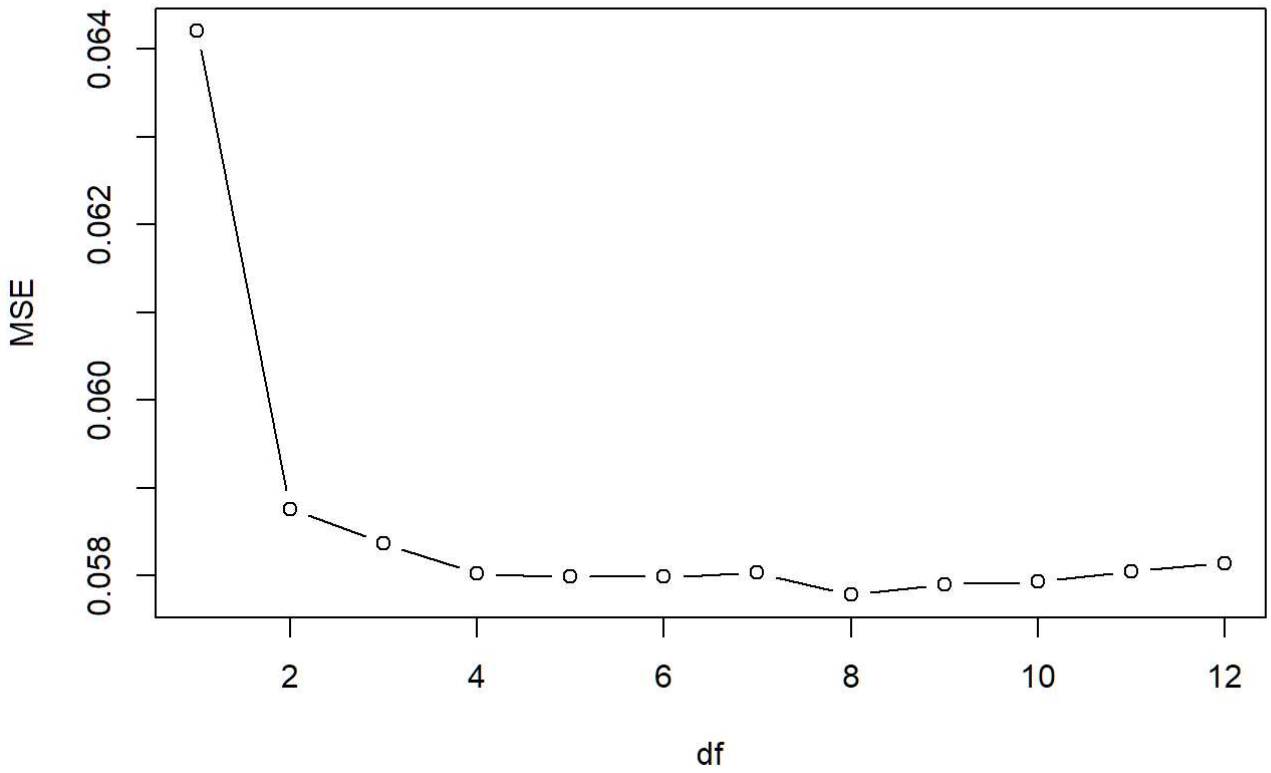
```
MSE[which.min(MSE)]
```

```
## [1] 0.05777915
```

Basis spine with 5 df is the best and the MSE is 0.05778.

1.2.3 Refit *dis* with Natural Spine model

```
MSE <- rep(0,12)
for (i in 1:12){
  Data <- RegressionData
  Data$disns <- ns(RegressionData$dis, df = i)
  fit <- train(nox ~ poly(age,3) + disns + lstat + medv,
               data = Data,
               method = 'lm',
               trControl=train_control)
  MSE[i] <- fit$results$RMSE
}

plot(1:12, MSE, type='b', xlab = 'df')
```




```
which.min(MSE)
```

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

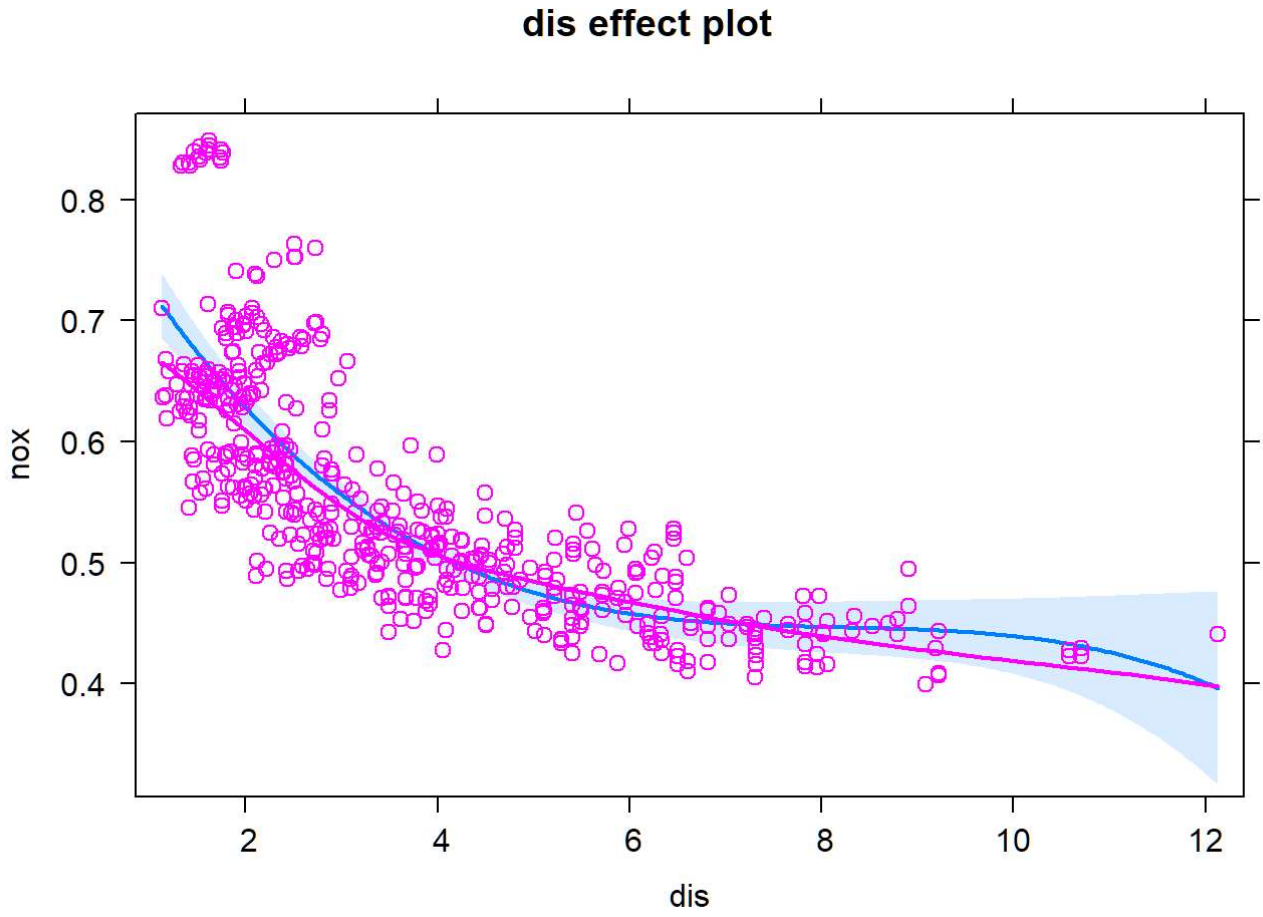
```
MSE[which.min(MSE)]
```

```
## [1] 0.05778422
```

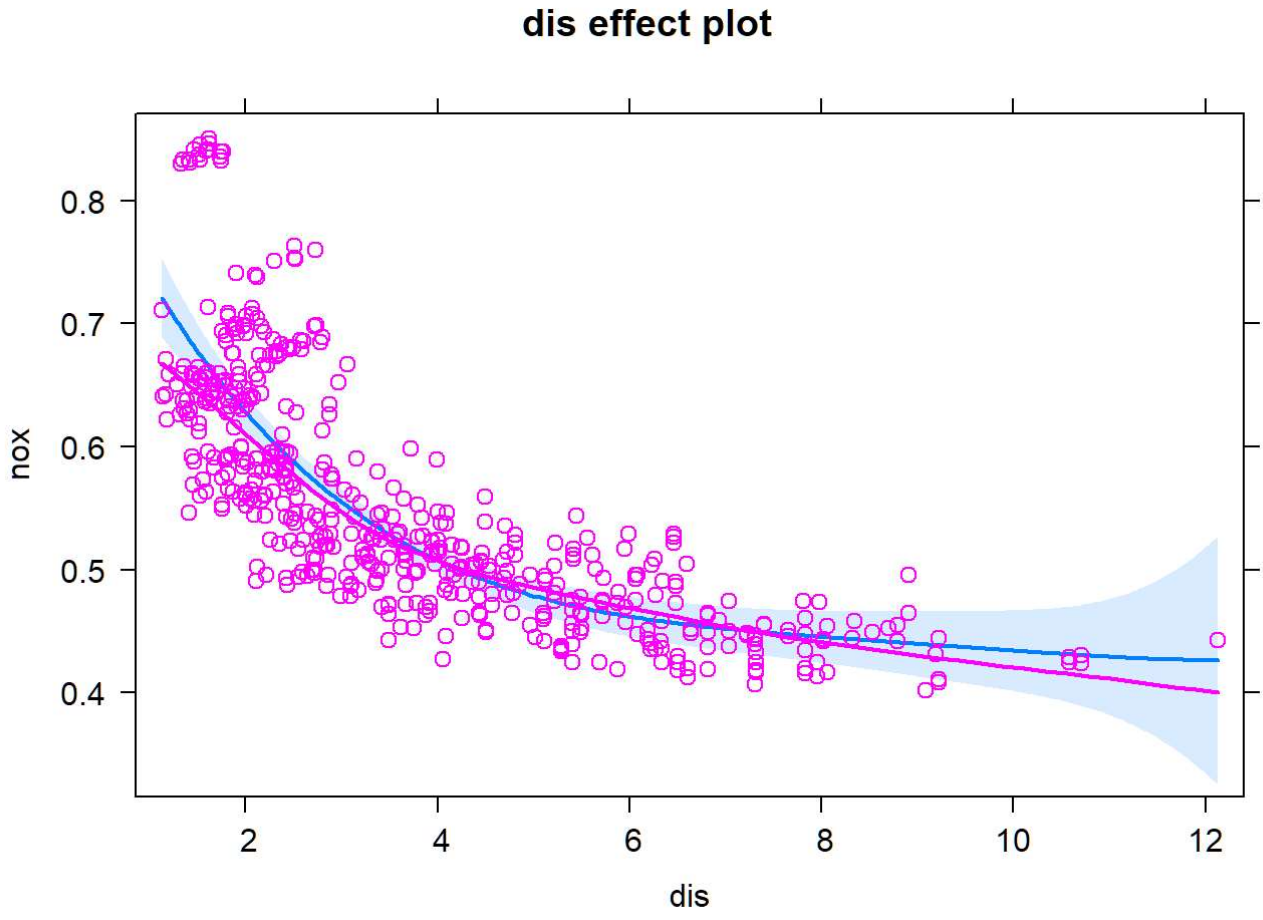
For natural spline, df=8 has the smallest MSE, which is 0.05778, and from df = 4 to df = 8 are almost the same.

Here, let's compare the fitted residual among different fitting methods with well performance.

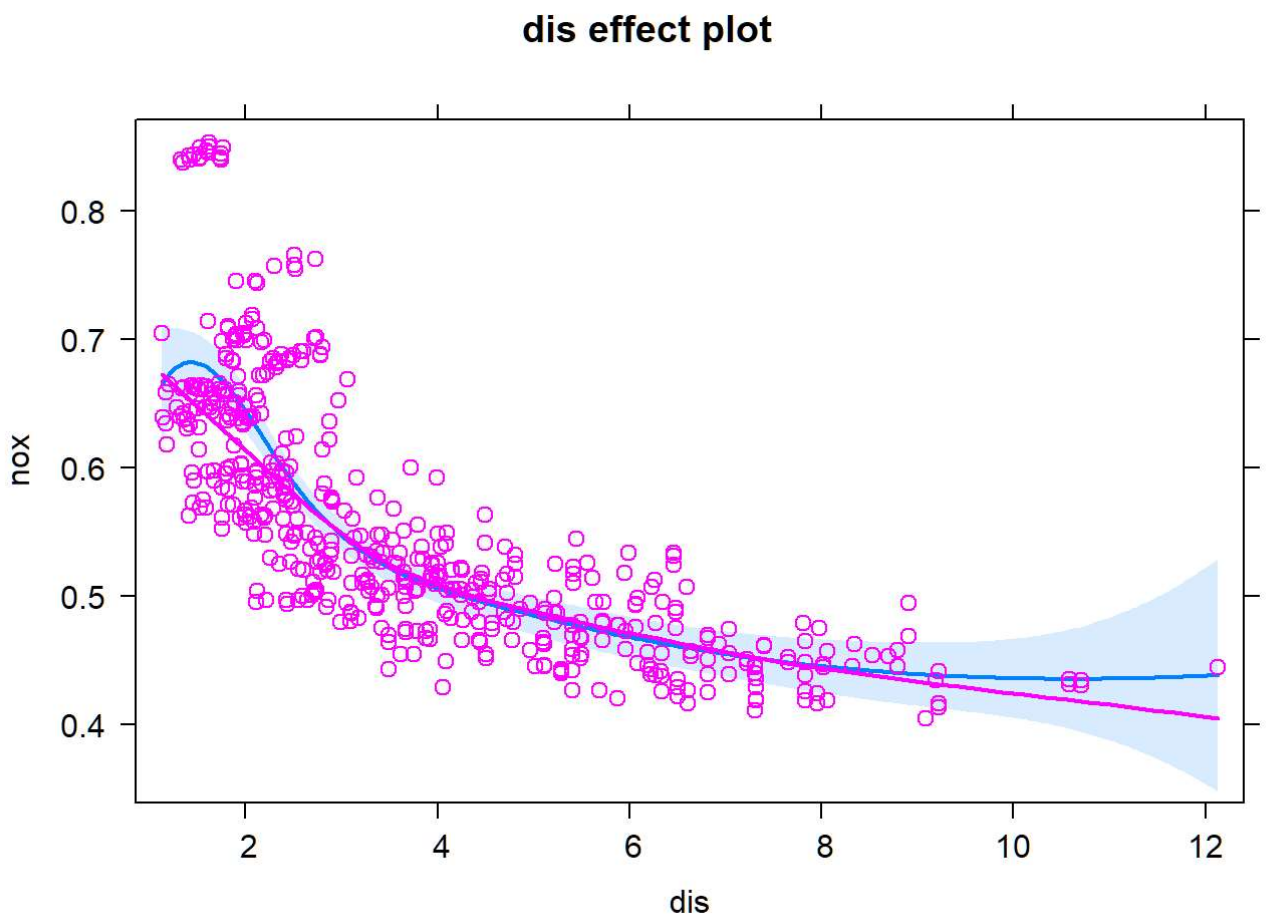
```
lm.fit.dis3 = lm(nox ~ poly(age,3) + poly(dis,3) + lstat + medv,
                       data = RegressionData)
lm.fit.dis4 = lm(nox ~ poly(age,3) + poly(dis,4) + lstat + medv,
                       data = RegressionData)
lm.bs.dis5 <- lm(nox ~ poly(age,3) + bs(dis,5) + lstat + medv,
                 data = RegressionData)
lm.ns.dis4 <- lm(nox ~ poly(age,3) + ns(dis,df=4) + lstat + medv,
                 data = RegressionData)
lm.ns.dis8 <- lm(nox ~ poly(age,3) + ns(dis,df=8) + lstat + medv,
                 data = RegressionData)

plot(effect("dis", mod = lm.fit.dis3, partial.residuals = TRUE))
```



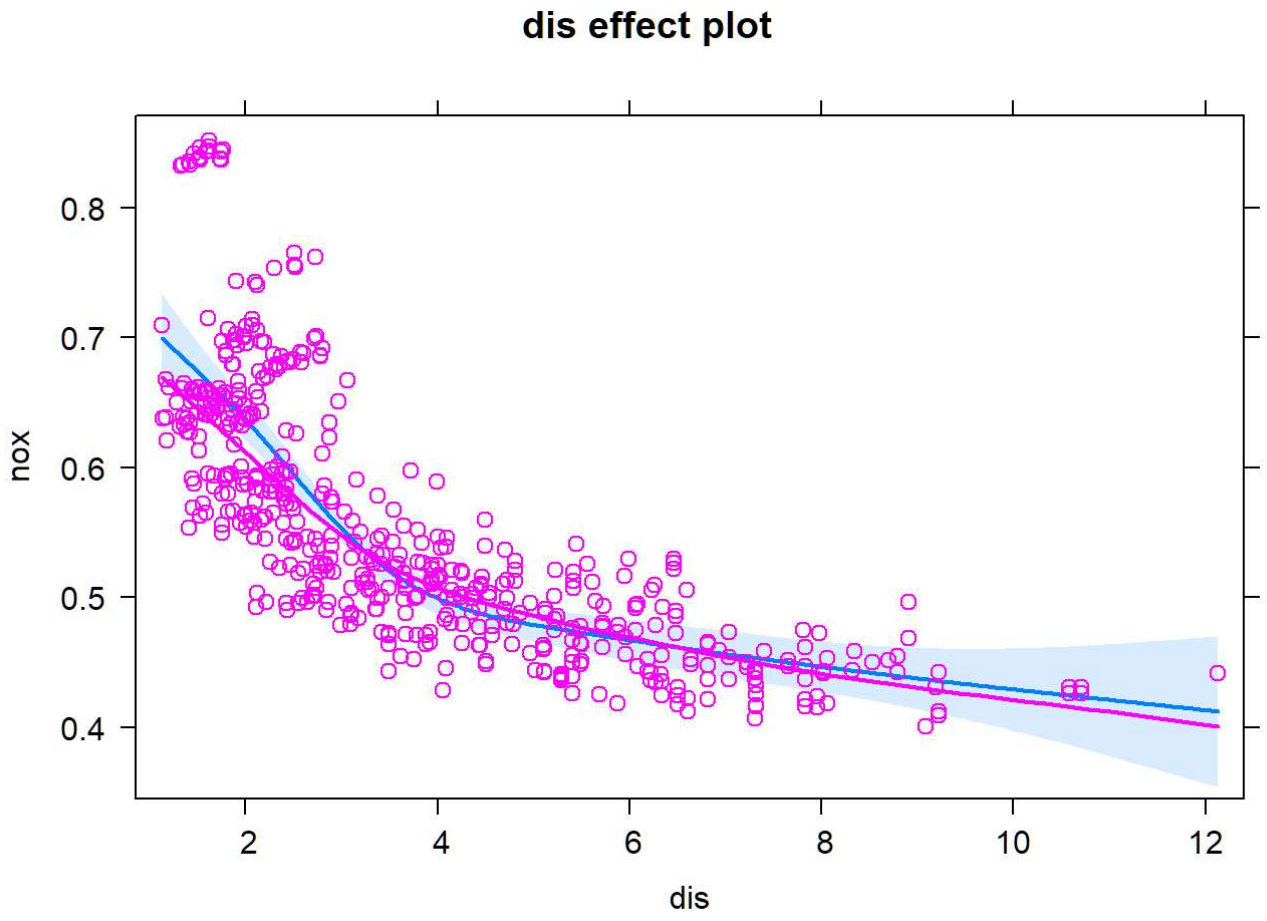

```
plot(effect("dis", mod = lm.fit.dis4, partial.residuals = TRUE))
```



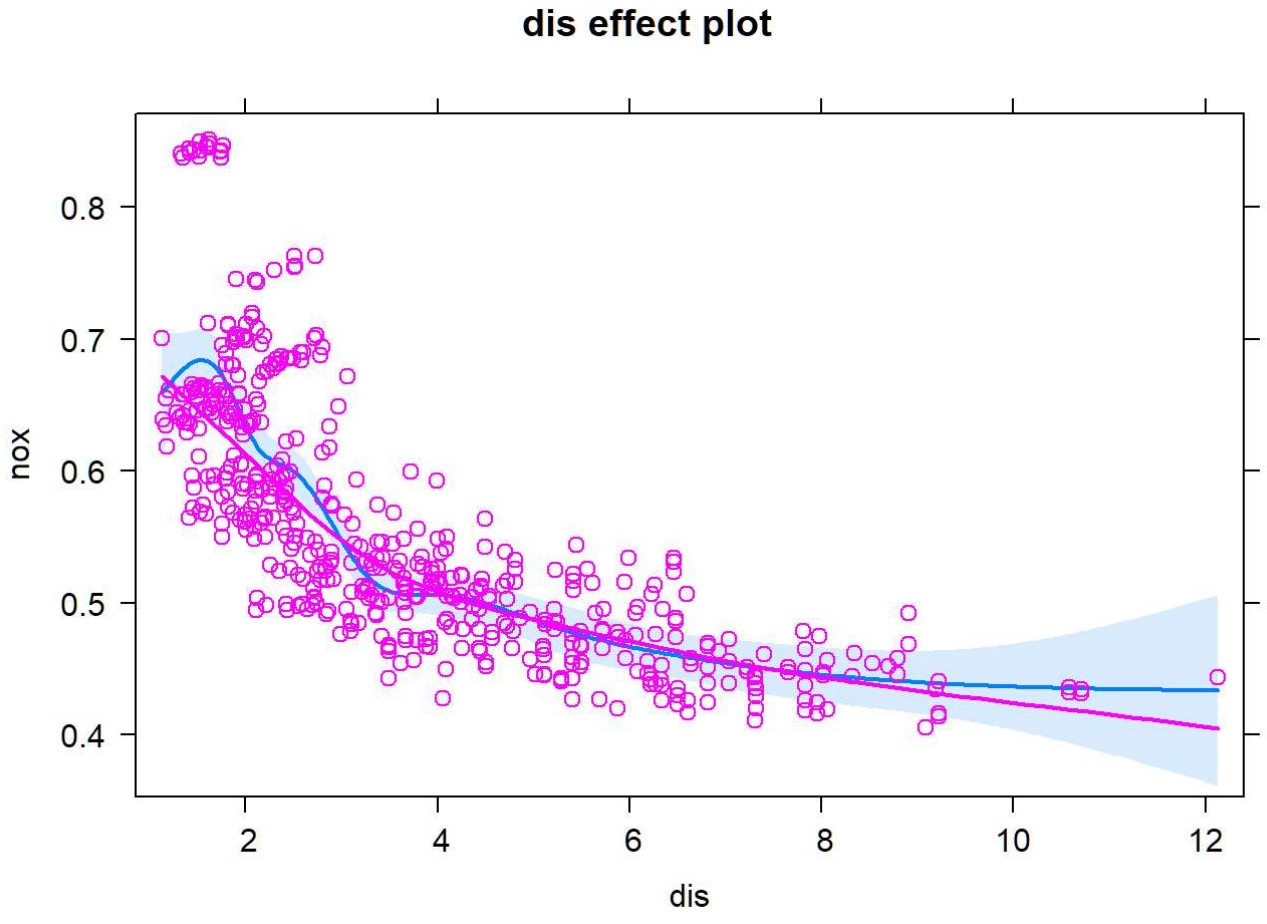
```
plot(effect("dis", mod = lm.bs.dis5, partial.residuals = TRUE))
```




```
plot(effect("dis", mod = lm.ns.dis4, partial.residuals = TRUE))
```




```
plot(effect("dis", mod = lm.ns.dis8, partial.residuals = TRUE))
```



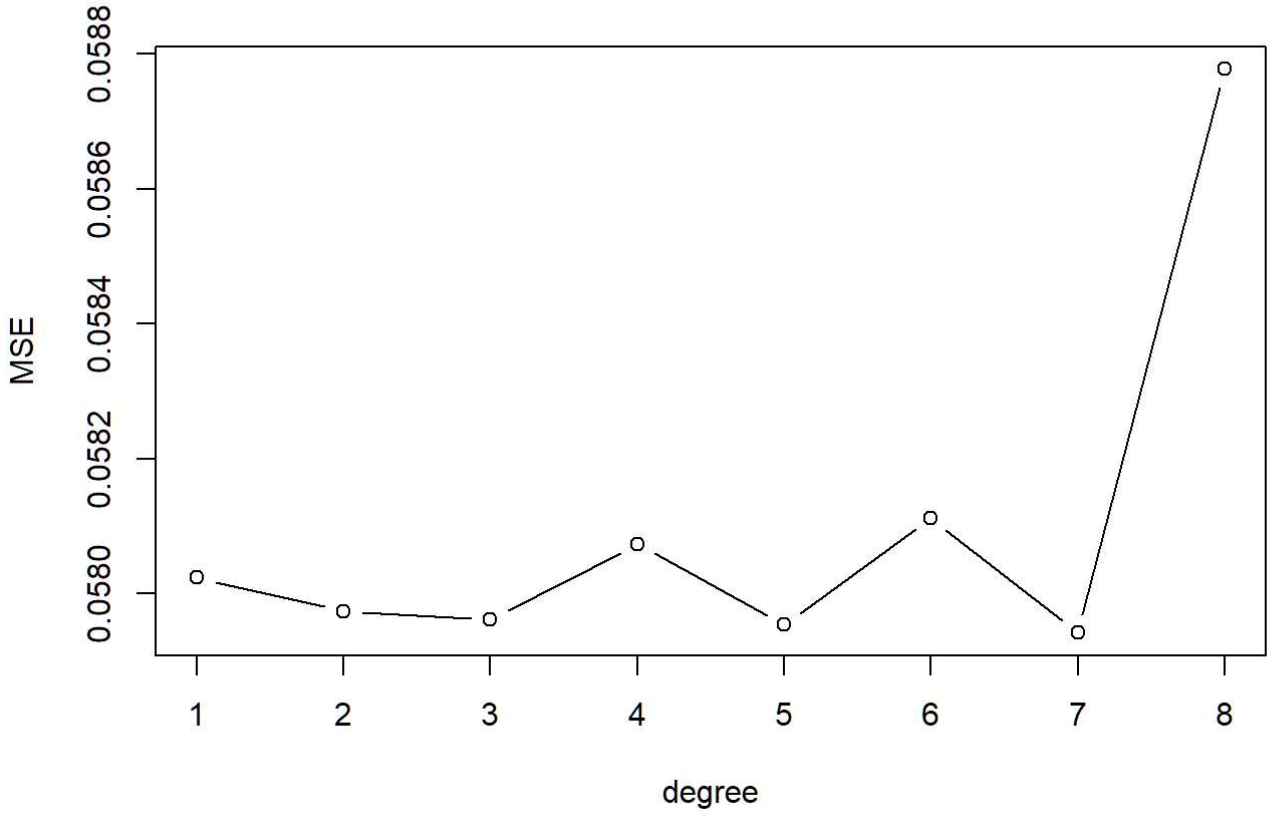
Here, it seems like the shape of residual of natural spline model with 4th-degree performance best as well as with the least wiggles.

Fitting *lstat*

1.3.1 Refit *lstat* with Polynomial

polynomial model

```
fit.lstat.1 <- train(nox ~ poly(age,3) + ns(dis,df = 4) + poly(lstat,1) + medv,
                     data = RegressionData,
                     method = "lm", trControl = train_control)
fit.lstat.2 <- train(nox ~ poly(age,3) + ns(dis,4) + poly(lstat,2) + medv,
                     data = RegressionData,
                     method = "lm", trControl = train_control)
fit.lstat.3 <- train(nox ~ poly(age,3) + ns(dis,4) + poly(lstat,3) + medv,
                     data = RegressionData,
                     method = "lm", trControl = train_control)
fit.lstat.4 <- train(nox ~ poly(age,3) + ns(dis,4) + poly(lstat,4) + medv,
                     data = RegressionData,
                     method = "lm", trControl = train_control)
fit.lstat.5 <- train(nox ~ poly(age,3) + ns(dis,4) + poly(lstat,5) + medv,
                     data = RegressionData,
                     method = "lm", trControl = train_control)
fit.lstat.6 <- train(nox ~ poly(age,3) + ns(dis,4) + poly(lstat,6) + medv,
                     data = RegressionData,
                     method = "lm", trControl = train_control)
fit.lstat.7 <- train(nox ~ poly(age,3) + ns(dis,4) + poly(lstat,7) + medv,
                     data = RegressionData,
                     method = "lm", trControl = train_control)
fit.lstat.8 <- train(nox ~ poly(age,3) + ns(dis,4) + poly(lstat,8) + medv,
                     data = RegressionData,
                     method = "lm", trControl = train_control)
MSE <- rep(0,8)
MSE[1] <- fit.lstat.1$results$RMSE
MSE[2] <- fit.lstat.2$results$RMSE
MSE[3] <- fit.lstat.3$results$RMSE
MSE[4] <- fit.lstat.4$results$RMSE
MSE[5] <- fit.lstat.5$results$RMSE
MSE[6] <- fit.lstat.6$results$RMSE
MSE[7] <- fit.lstat.7$results$RMSE
MSE[8] <- fit.lstat.8$results$RMSE

plot(1:8, MSE, type = "b", xlab = "degree")
```




```
which.min(MSE)
```

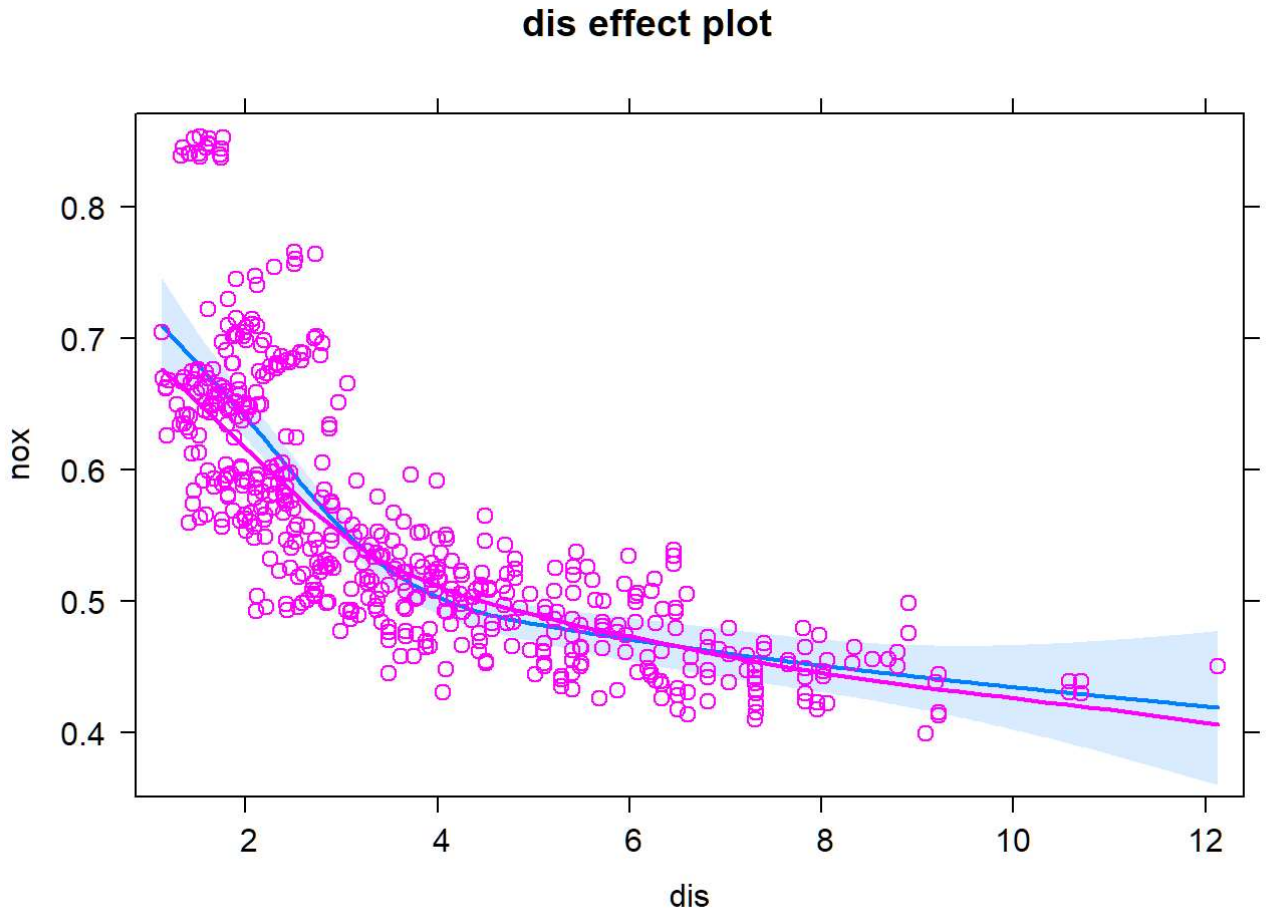
```
## [1] 7
```

```
MSE[which.min(MSE)]
```

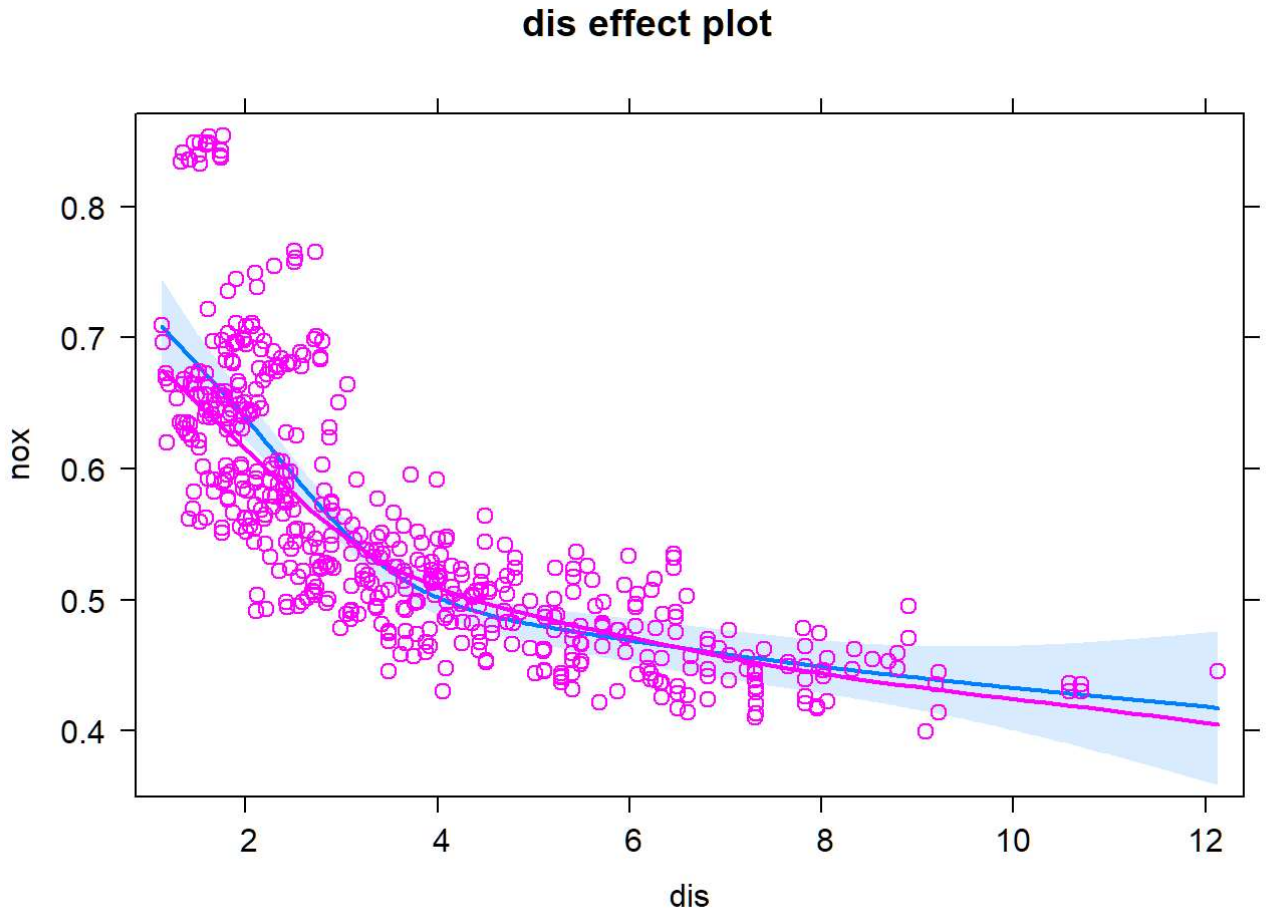
```
## [1] 0.05794206
```

Here, 2, 3, 5, 7 are almost the same, so let's plot the residual shapes of these models. On the grounds of parsimony. Let's compare th 2, 3, and 7.

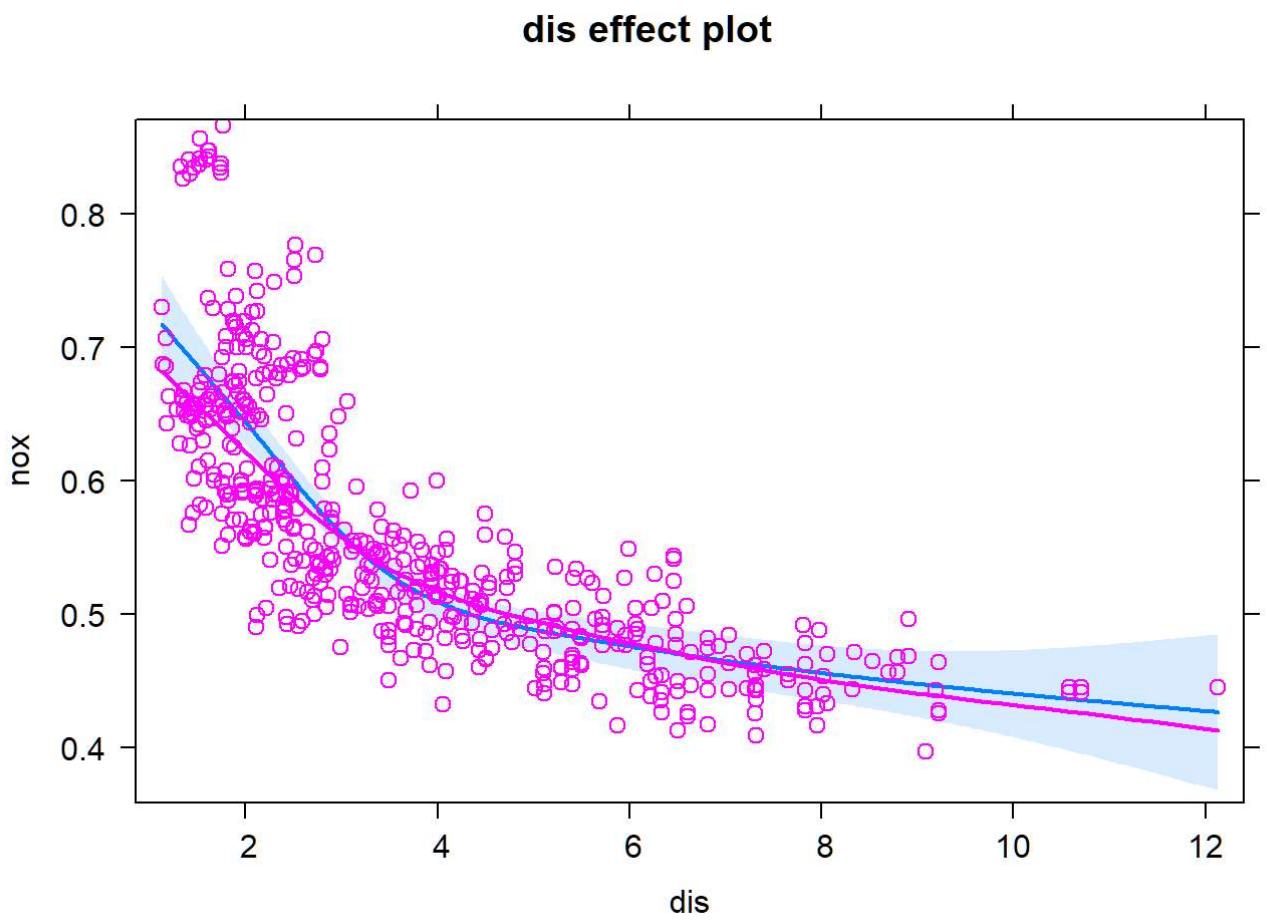
```
lm.fit.lstat2 = lm(nox ~ poly(age,3) + ns(dis,4) + poly(lstat,2) + medv,
                   data = RegressionData)
lm.fit.lstat3 = lm(nox ~ poly(age,3) + ns(dis,4) + poly(lstat,3) + medv,
                   data = RegressionData)
lm.fit.lstat7 = lm(nox ~ poly(age,3) + ns(dis,4) + poly(lstat,7) + medv,
                   data = RegressionData)

plot(effect("dis", mod = lm.fit.lstat2, partial.residuals = TRUE))
```




```
plot(effect("dis", mod = lm.fit.lstat3, partial.residuals = TRUE))
```



```
plot(effect("dis", mod = lm.fit.lstat7, partial.residuals = TRUE))
```



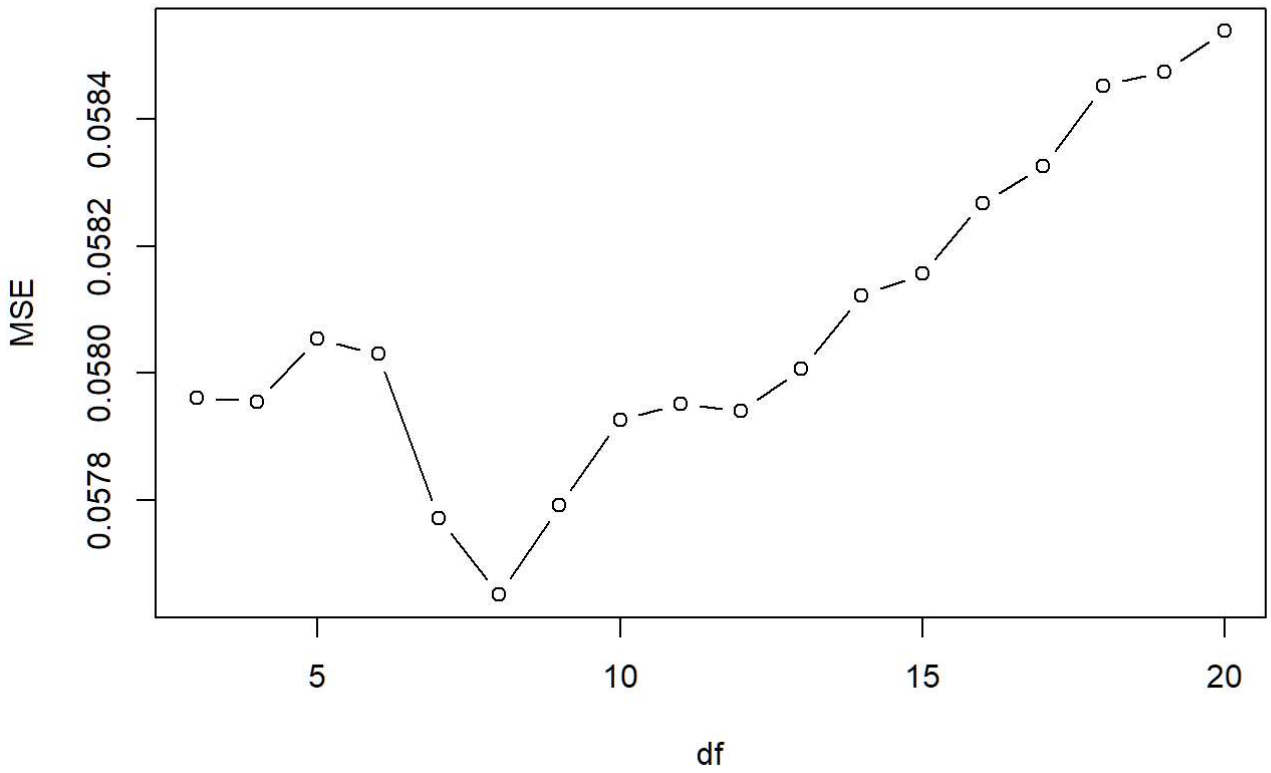

```
MSE[2]
```

```
## [1] 0.05797407
```

There's no significant difference and the second polynomial still has a relatively small MSE = 0.0580.

1.3.2 Refit *lstatbs* with Basis Spline

```
MSE <- rep(0,18)
for (i in 3:20){
  Data <- RegressionData
  Data$lstatbs <- bs(RegressionData$lstat, df = i)
  fit <- train(nox ~ poly(age,3) + ns(dis,4) + lstatbs + medv,
               data = Data,
               method = 'lm',
               trControl=train_control)
  MSE[i-2] <- fit$results$RMSE
}
plot(3:20, MSE, type='b', xlab = 'df')
```




```
dfn <- which.min(MSE) + 2
dfn
```

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

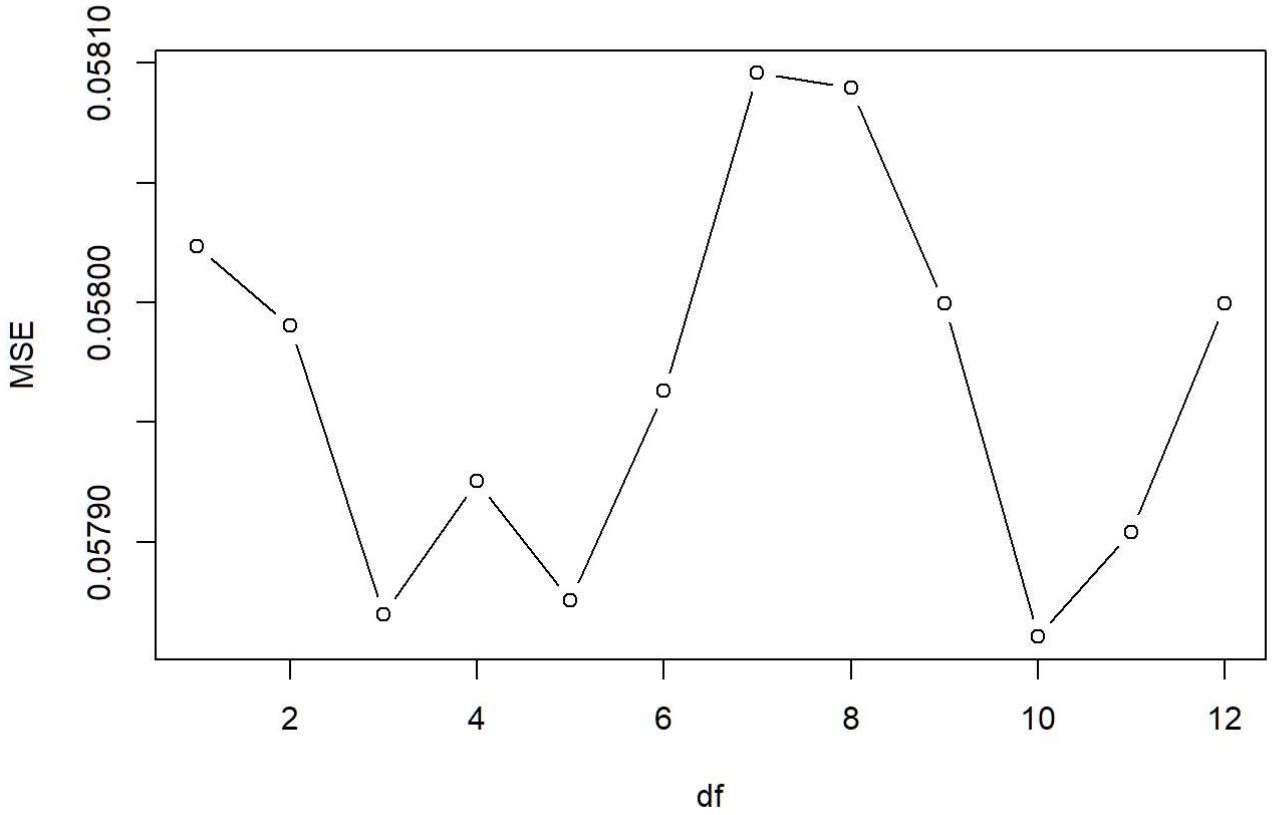
```
MSE[which.min(MSE)]
```

```
## [1] 0.05765122
```

Here, basis spline with df = 8 has the lowest MSE (0.0577), which is better than polynomial fit.

1.3.3 Refit *lstatbs* with Natural Spline

```
MSE <- rep(0,12)
for (i in 1:12){
  Data <- RegressionData
  Data$lstatns <- ns(RegressionData$lstat, df = i)
  fit <- train(nox ~ poly(age,3) + ns(dis,4) + lstatns + medv,
               data = Data,
               method = 'lm',
               trControl=train_control)
  MSE[i] <- fit$results$RMSE
}

plot(1:12, MSE, type='b', xlab = 'df')
```




```
which.min(MSE)
```

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

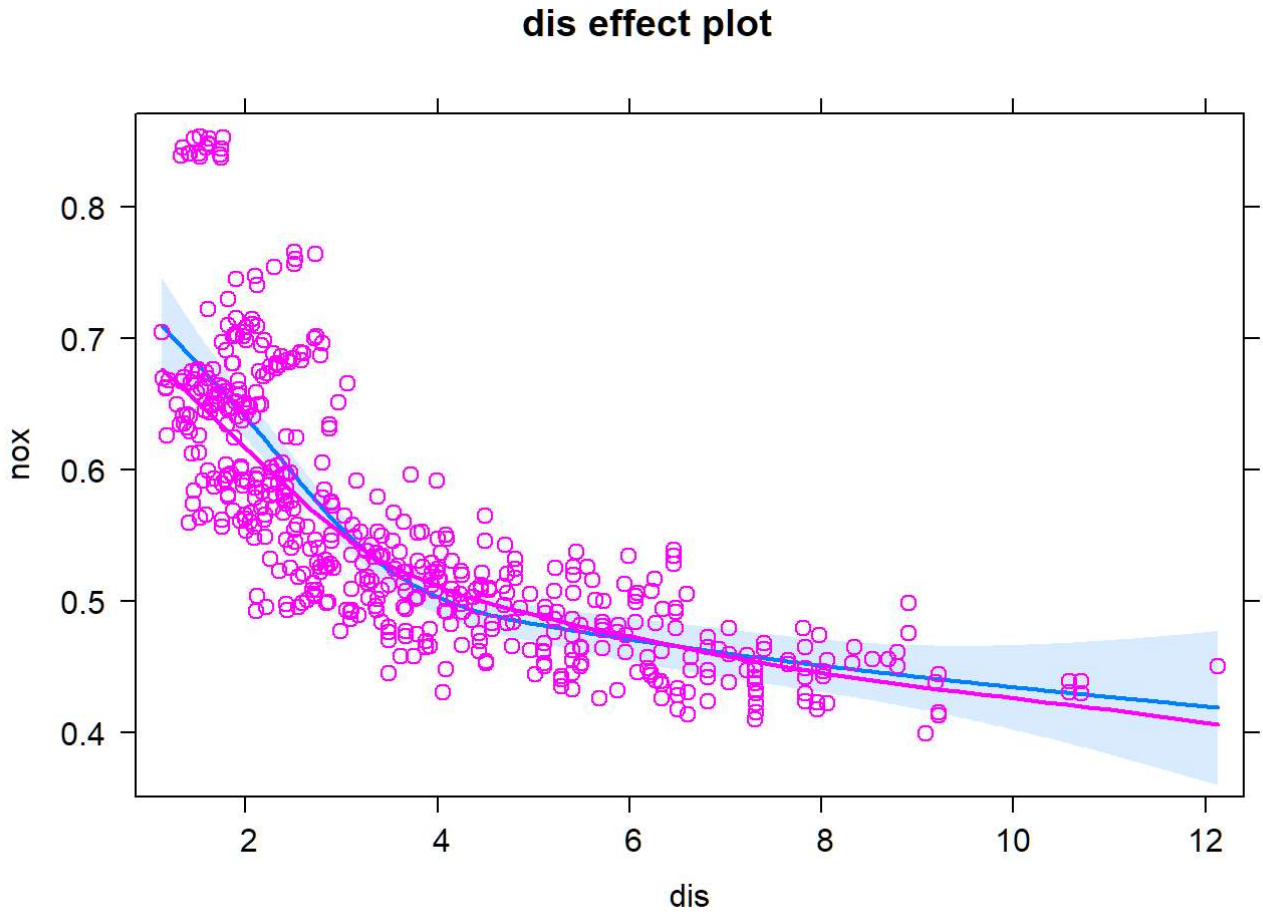
```
MSE[which.min(MSE)]
```

```
## [1] 0.05786041
```

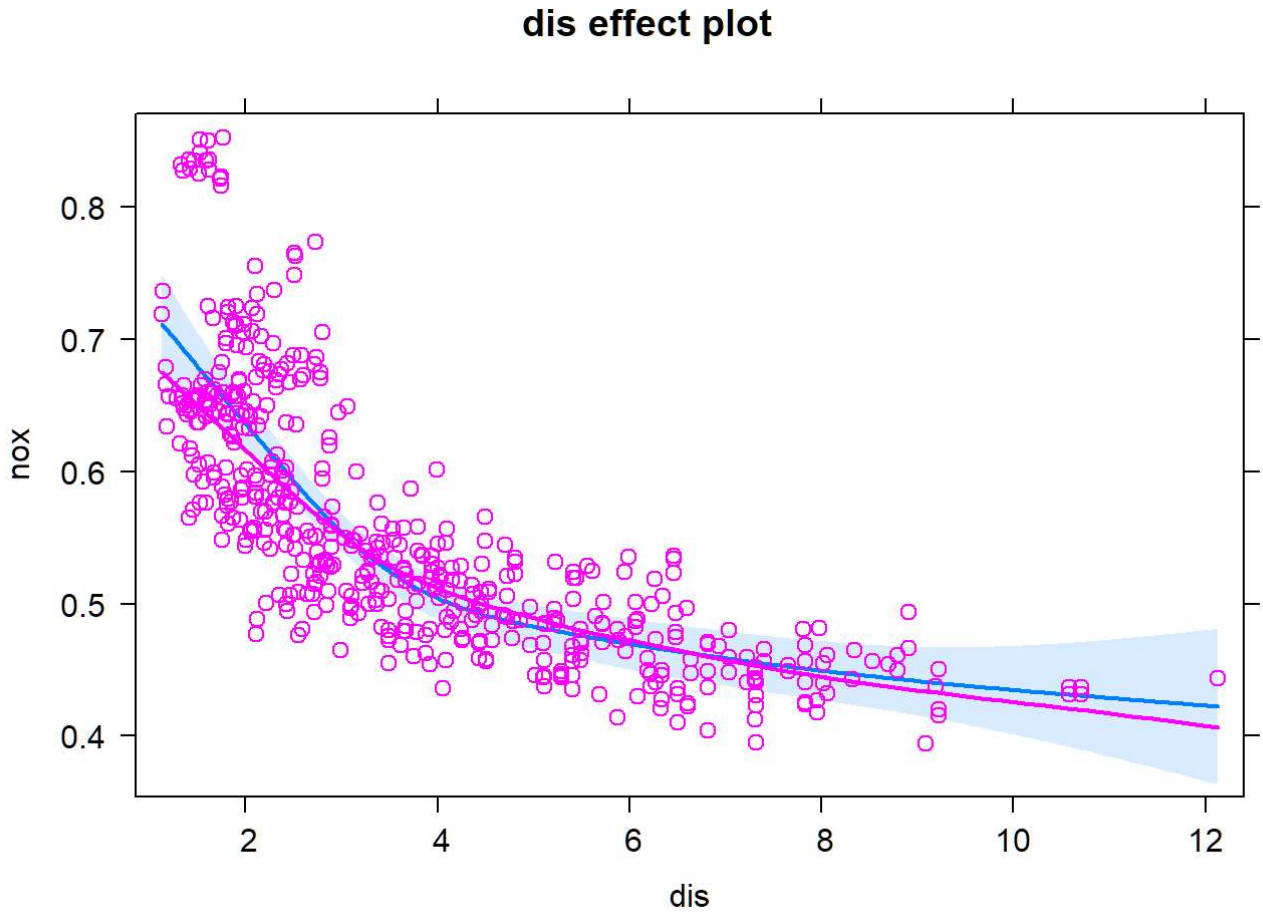
Natural spline with 10 degree is the best among this but has lager MSE than 8 degree basis spine.

Now let's compare between this three fitting methods for *lstat*.

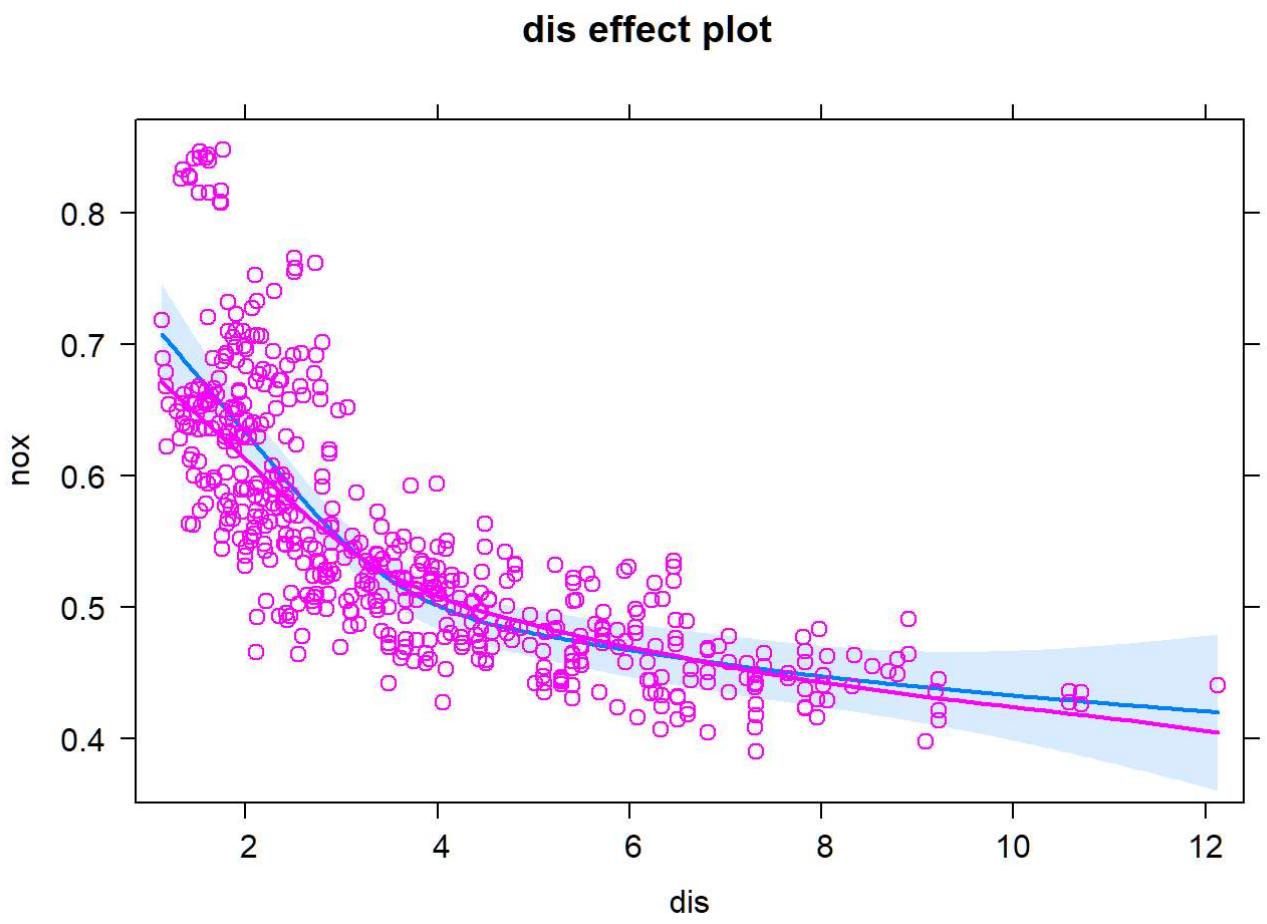
```
lm.lstat2 = lm(nox ~ poly(age,3) + ns(dis,4) + poly(lstat,2) + medv,
                 data = RegressionData)
lm.bs.lstat8 = lm(nox ~ poly(age,3) + ns(dis,4) + bs(lstat,8) + medv,
                 data = RegressionData)
lm.ns.lstat10 = lm(nox ~ poly(age,3) + ns(dis,4) + ns(lstat,10) + medv,
                 data = RegressionData)
plot(effect("dis", mod = lm.lstat2, partial.residuals = TRUE))
```




```
plot(effect("dis", mod = lm.bs.lstat8, partial.residuals = TRUE))
```



```
plot(effect("dis", mod = lm.ns.lstat10, partial.residuals = TRUE))
```

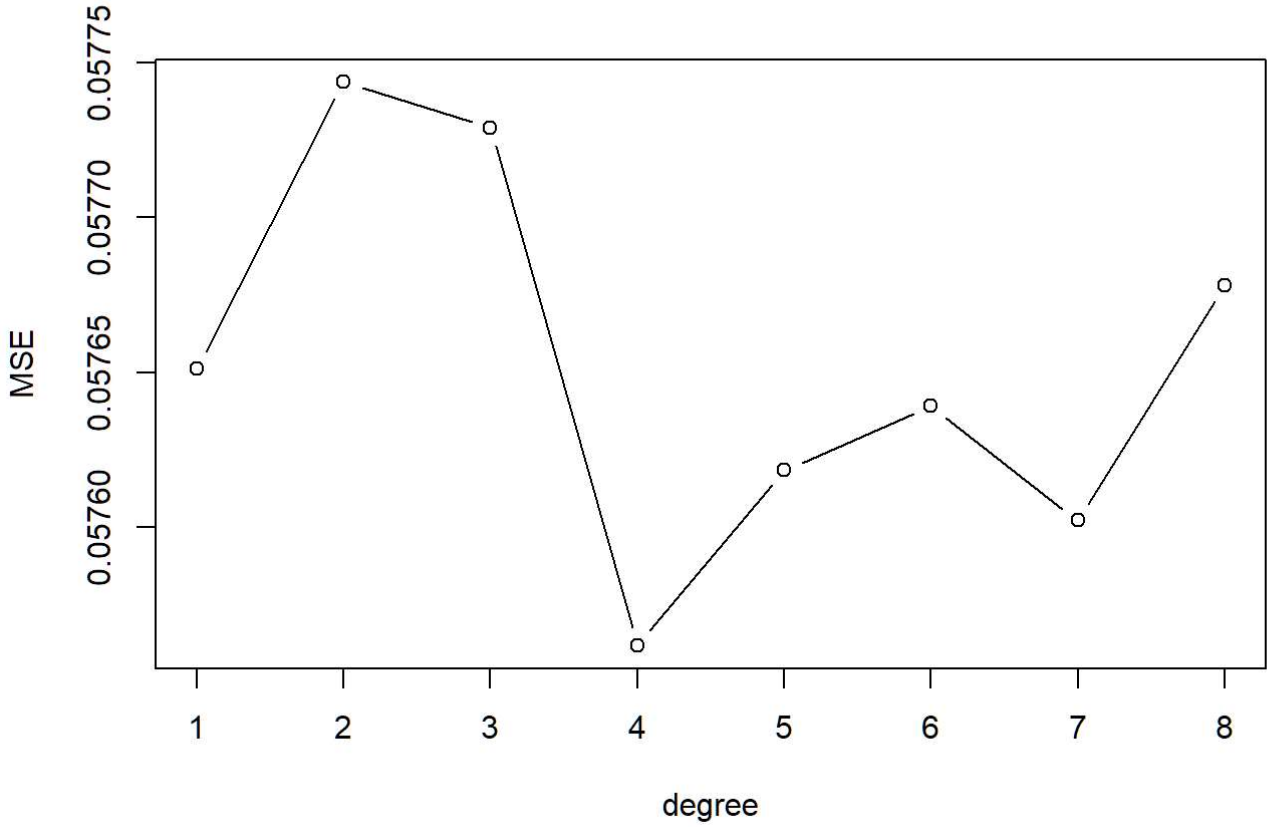


The plots are all good, so we choose bs with 8 df with smallest MSE = 0.0577.

Fitting *medv*

1.4.1 Refit *medv* with Polynomial

```
fit.medv.1 <- train(nox ~ poly(age,3) + ns(dis,4) + bs(lstat,8)
                    + poly(medv,1),
                       data = RegressionData,
                       method = "lm", trControl = train_control)
fit.medv.2 <- train(nox ~ poly(age,3) + ns(dis,4) + bs(lstat,8)
                    + poly(medv,2),
                       data = RegressionData,
                       method = "lm", trControl = train_control)
fit.medv.3 <- train(nox ~ poly(age,3) + ns(dis,4) + bs(lstat,8)
                    + poly(medv,3),
                       data = RegressionData,
                       method = "lm", trControl = train_control)
fit.medv.4 <- train(nox ~ poly(age,3) + ns(dis,4) + bs(lstat,8)
                    + poly(medv,4),
                       data = RegressionData,
                       method = "lm", trControl = train_control)
fit.medv.5 <- train(nox ~ poly(age,3) + ns(dis,4) + bs(lstat,8)
                    + poly(medv,5),
                       data = RegressionData,
                       method = "lm", trControl = train_control)
fit.medv.6 <- train(nox ~ poly(age,3) + ns(dis,4) + bs(lstat,8)
                    + poly(medv,6),
                       data = RegressionData,
                       method = "lm", trControl = train_control)
fit.medv.7 <- train(nox ~ poly(age,3) + ns(dis,4) + bs(lstat,8)
                    + poly(medv,7),
                       data = RegressionData,
                       method = "lm", trControl = train_control)
fit.medv.8 <- train(nox ~ poly(age,3) + ns(dis,4) + bs(lstat,8)
                    + poly(medv,8),
                       data = RegressionData,
                       method = "lm", trControl = train_control)
MSE <- rep(0,8)
MSE[1] <- fit.medv.1$results$RMSE
MSE[2] <- fit.medv.2$results$RMSE
MSE[3] <- fit.medv.3$results$RMSE
MSE[4] <- fit.medv.4$results$RMSE
MSE[5] <- fit.medv.5$results$RMSE
MSE[6] <- fit.medv.6$results$RMSE
MSE[7] <- fit.medv.7$results$RMSE
MSE[8] <- fit.medv.8$results$RMSE

plot(1:8, MSE, type = "b", xlab = "degree")
```




```
which.min(MSE)
```

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

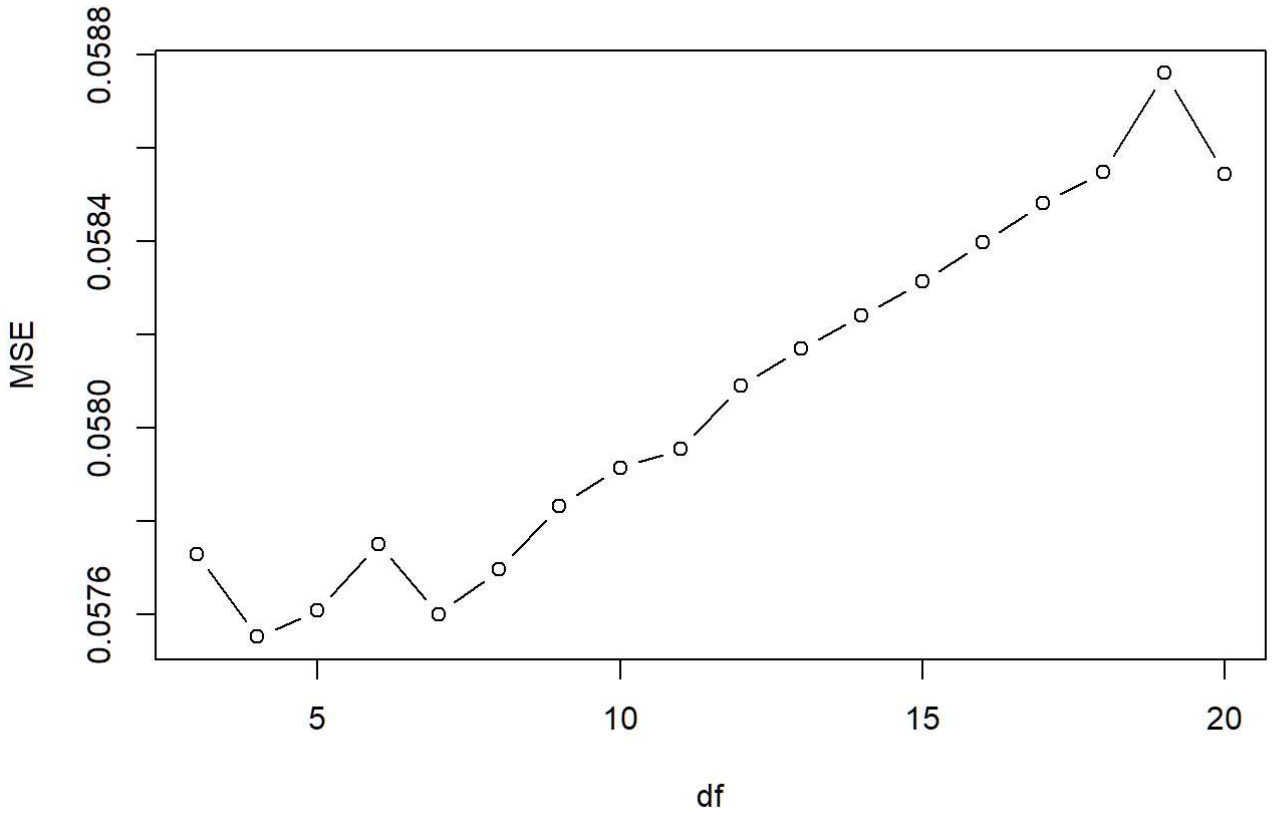
```
MSE[which.min(MSE)]
```

```
## [1] 0.05756186
```

Here, 4th polynomial has smallest MSE, which is 0.0576.

1.4.2 Refit *medv* with Basis Spline.

```
MSE <- rep(0,18)
for (i in 3:20){
  Data <- RegressionData
  Data$medvbs <- bs(RegressionData$medv, df = i)
  fit <- train(nox ~ poly(age,3) + ns(dis,4) + bs(lstat,8) + medvbs,
               data = Data,
               method = 'lm',
               trControl=train_control)
  MSE[i-2] <- fit$results$RMSE
}
plot(3:20, MSE, type='b', xlab = 'df')
```




```
dfn <- which.min(MSE) + 2
dfn
```

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

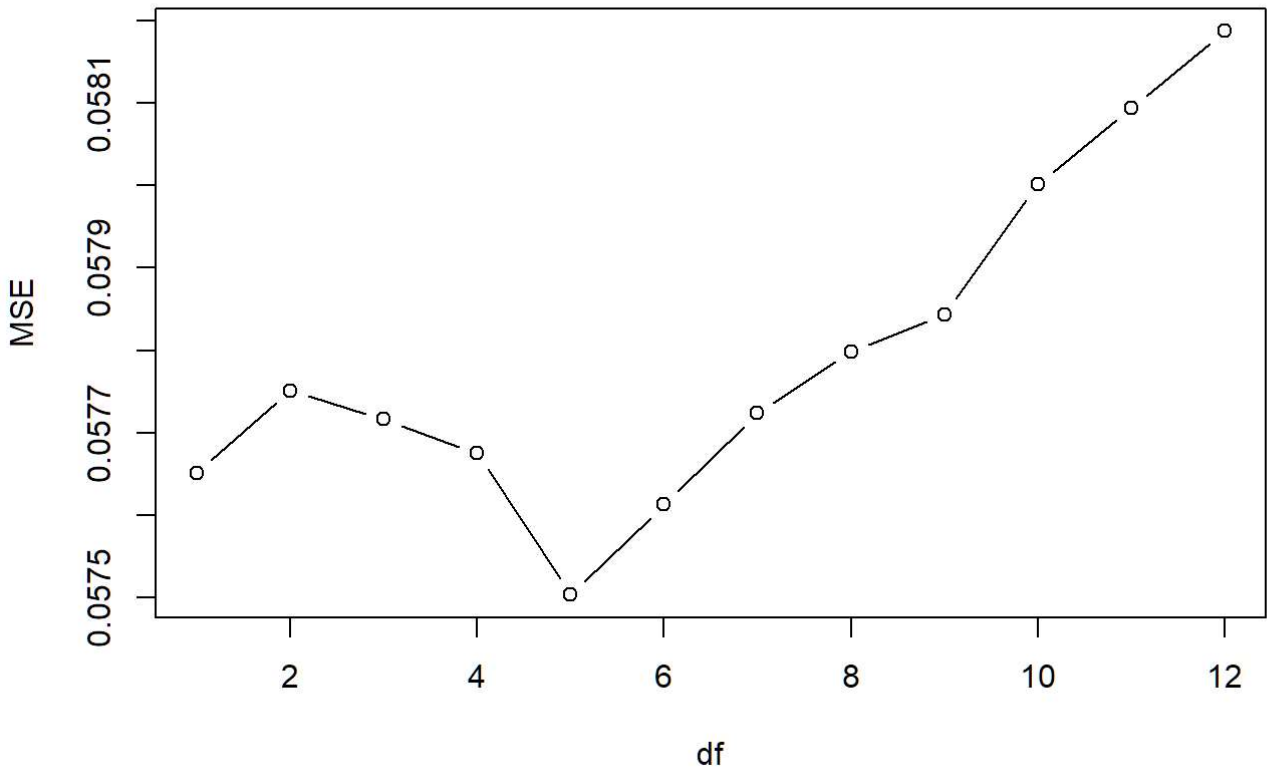
```
MSE[which.min(MSE)]
```

```
## [1] 0.05755134
```

Here, when df = 4, the MSE is the smallest (0.0576).

1.4.3 Refit *medv* with Natural Spline.

```
MSE <- rep(0,12)
for (i in 1:12){
  Data <- RegressionData
  Data$medvns <- ns(RegressionData$medv, df = i)
  fit <- train(nox ~ poly(age,3) + ns(dis,4) + bs(lstat,8) + medvns,
               data = Data,
               method = 'lm',
               trControl=train_control)
  MSE[i] <- fit$results$RMSE
}

plot(1:12, MSE, type='b', xlab = 'df')
```




```
which.min(MSE)
```

```
## [1] 5
```

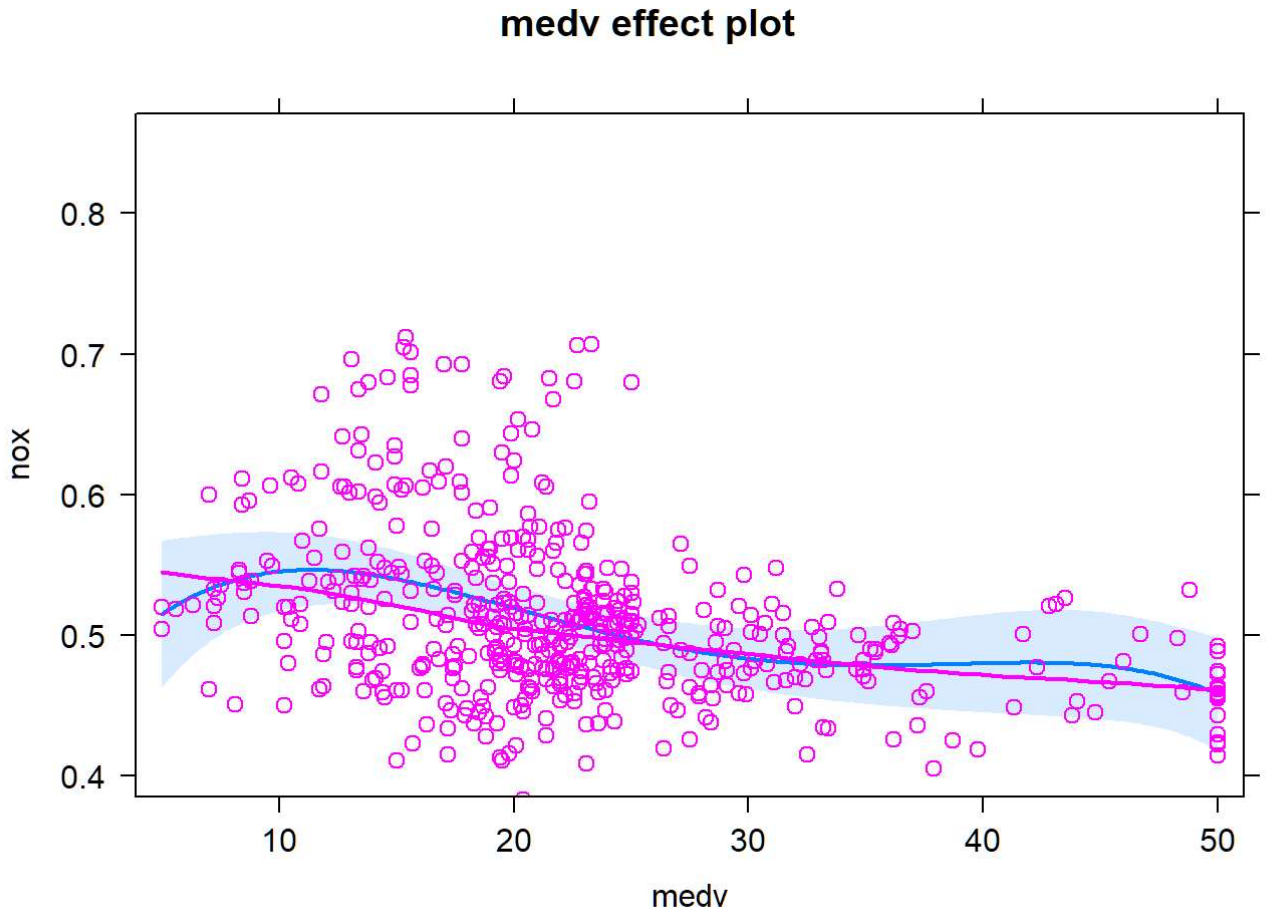
```
MSE[which.min(MSE)]
```

```
## [1] 0.05750367
```

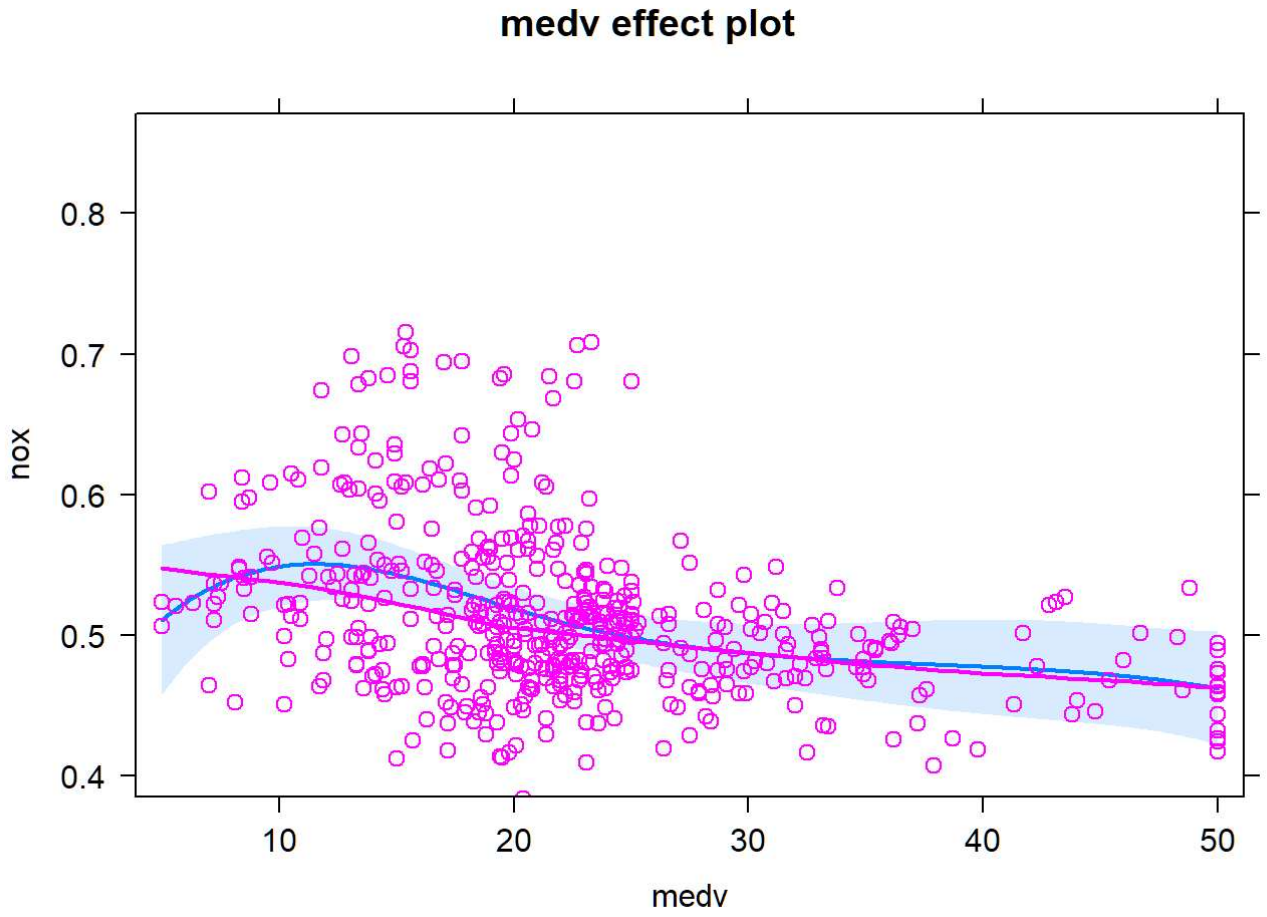
Natural spline with df = 5 has smallest MSE = 0.0575.

Compare between ns(df = 5), bs(df = 4) and 4th polynomia.

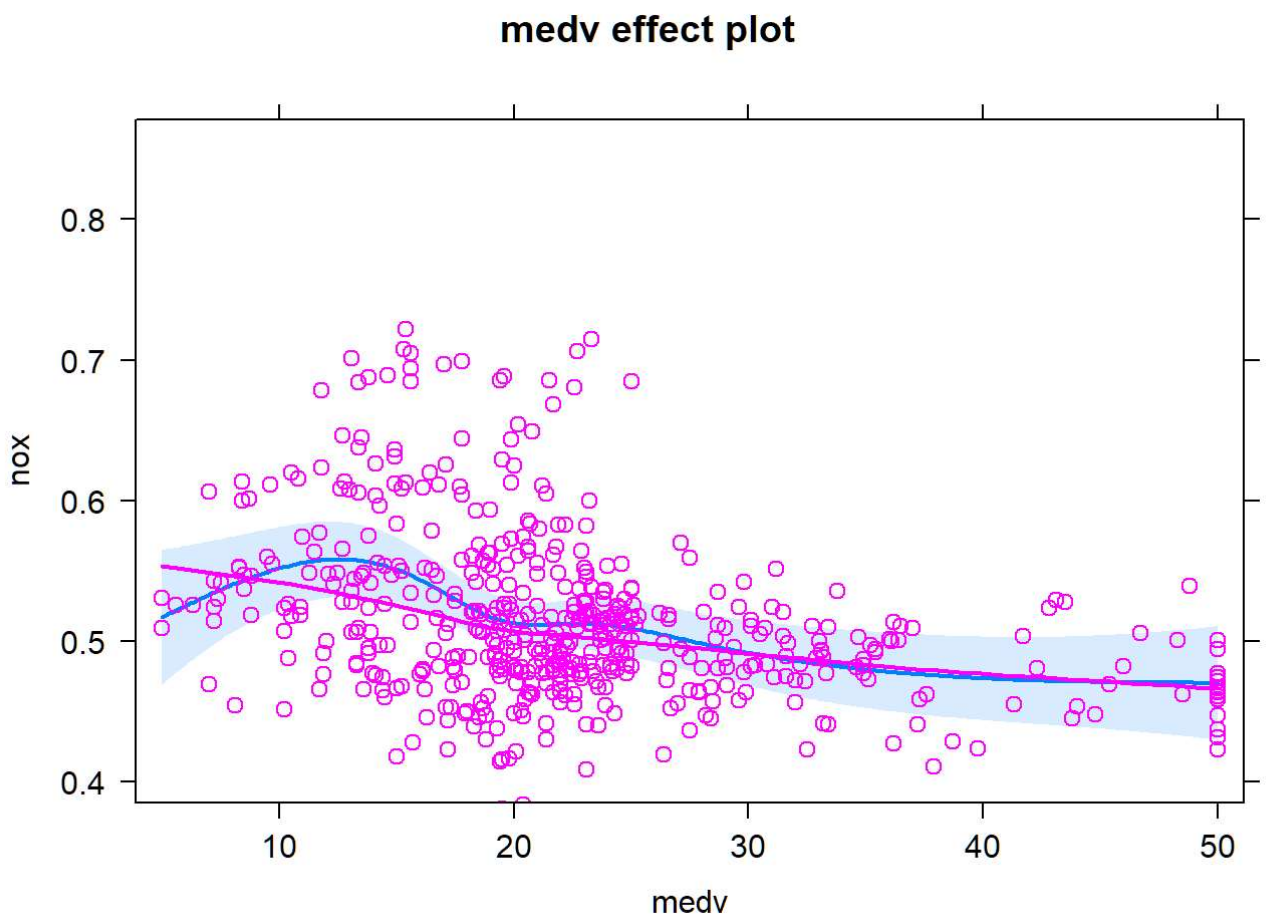
```
lm.fit.medv4 <- lm(nox ~ poly(age,3) + ns(dis,4) + bs(lstat,8)
                   + poly(medv,4),
                      data = RegressionData)
lm.bs.medv4 <- lm(nox ~ poly(age,3) + ns(dis,4) + bs(lstat,8)
                   + bs(medv,4),
                      data = RegressionData)
lm.ns.medv5 <- lm(nox ~ poly(age,3) + ns(dis,4) + bs(lstat,8)
                   + ns(medv,5),
                      data = RegressionData)

plot(effect("medv", mod = lm.fit.medv4, partial.residuals = TRUE))
```




```
plot(effect("medv", mod = lm.bs.medv4, partial.residuals = TRUE))
```



```
plot(effect("medv", mod = lm.ns.medv5, partial.residuals = TRUE))
```

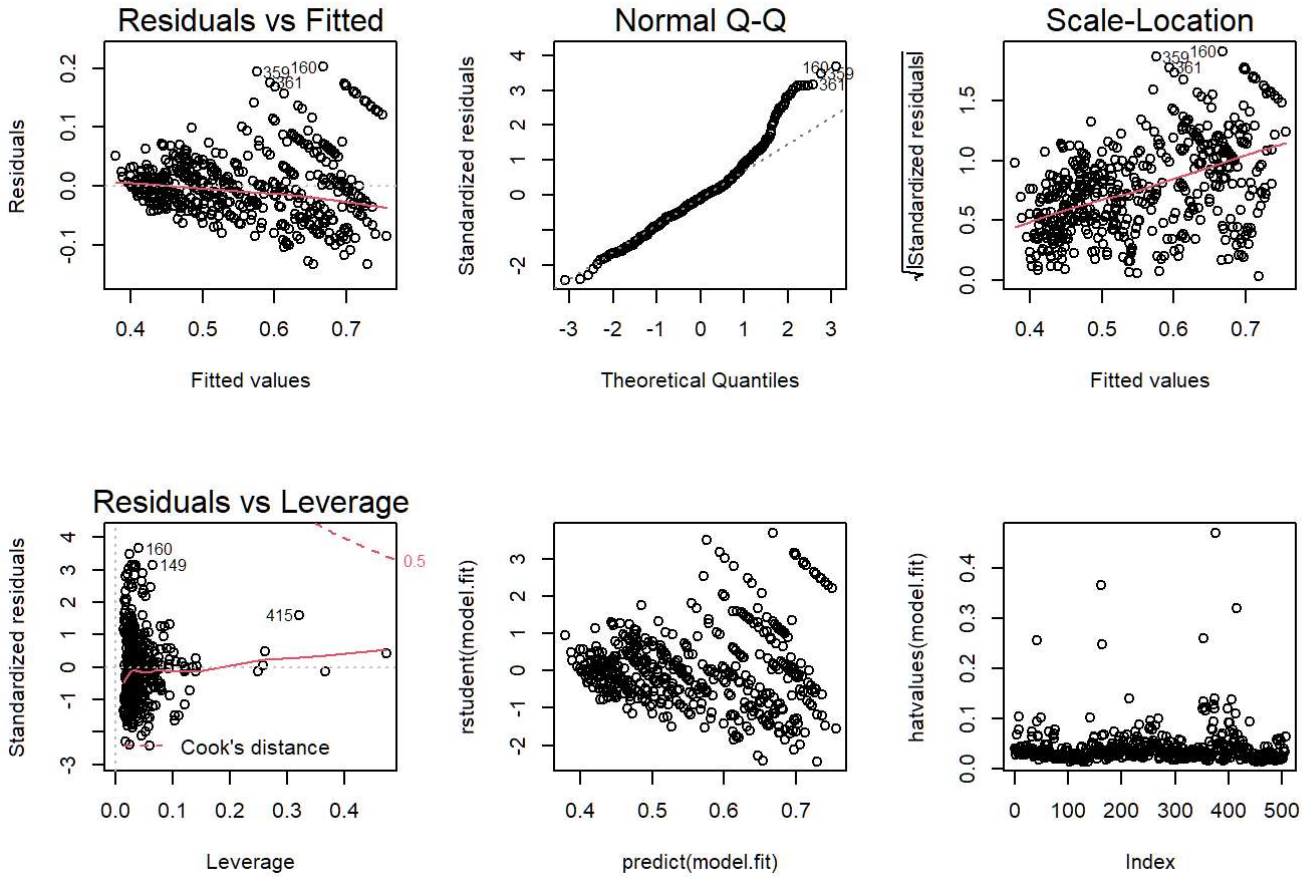


We choose natural spline with 5 df for *medv* which has smallest MSE among all fitting methods.

Whole Model Build and Assumption Check 2

Thus, the whole model has been build. Now, we are supposed to check the assumption for this final model.

```
model.fit <- lm.ns.medv5 <- lm(nox ~ poly(age,3) + ns(dis,4) + bs(lstat,8)
                  + ns(medv,5),
                     data = RegressionData)

par(mfrow = c(2, 3))
plot(model.fit)
plot(predict(model.fit), rstudent(model.fit))
plot(hatvalues(model.fit))
```



All these plots suggested that the effect of leverage point are not noticeable and residuals has not significant pattern.

Test model with different number of input variables

Best Subsets

```
## 0 inputs
mod.0 <- lm(nox ~ 1, data = RegressionData)

## 1 inputs
mod.1.1 <- lm(nox ~ poly(age,3), data = RegressionData)
mod.1.2 <- lm(nox ~ ns(dis,4), data = RegressionData)
mod.1.3 <- lm(nox ~ bs(lstat,8), data = RegressionData)
mod.1.4 <- lm(nox ~ ns(medv,5), data = RegressionData)
which.min(c(deviance(mod.1.1), deviance(mod.1.2),
            deviance(mod.1.3),deviance(mod.1.4)))
```

```
## [1] 2
```

```
# mod.1.2 is the winner

## 2 inputs
mod.2.1 <- lm(nox ~ poly(age,3) + ns(dis,4), data = RegressionData)
mod.2.2 <- lm(nox ~ poly(age,3) + bs(lstat,8), data = RegressionData)
mod.2.3 <- lm(nox ~ poly(age,3) + ns(medv,5), data = RegressionData)
mod.2.4 <- lm(nox ~ ns(dis,4) + bs(lstat,8), data = RegressionData)
mod.2.5 <- lm(nox ~ ns(dis,4) + ns(medv,5), data = RegressionData)
mod.2.6 <- lm(nox ~ ns(medv,5) + bs(lstat,8), data = RegressionData)
which.min(c(deviance(mod.2.1), deviance(mod.2.2),
            deviance(mod.2.3),deviance(mod.2.4),
            deviance(mod.2.5),deviance(mod.2.6)))
```

```
## [1] 5
```

```
# mod.2.5 is the winner

## 3 inputs
mod.3.1 <- gam(nox ~ poly(age,3) + ns(dis,4) + bs(lstat,8),
               data = RegressionData)
mod.3.2 <- lm(nox ~ poly(age,3) + ns(dis,4) + ns(medv,5),
               data = RegressionData)
mod.3.3 <- lm(nox ~ poly(age,3) + bs(lstat,8) + ns(medv,5),
               data = RegressionData)
mod.3.4 <- lm(nox ~ ns(dis,4) + bs(lstat,8) + ns(medv,5),
               data = RegressionData)
which.min(c(deviance(mod.3.1), deviance(mod.3.2),
            deviance(mod.3.3),deviance(mod.3.4)))
```

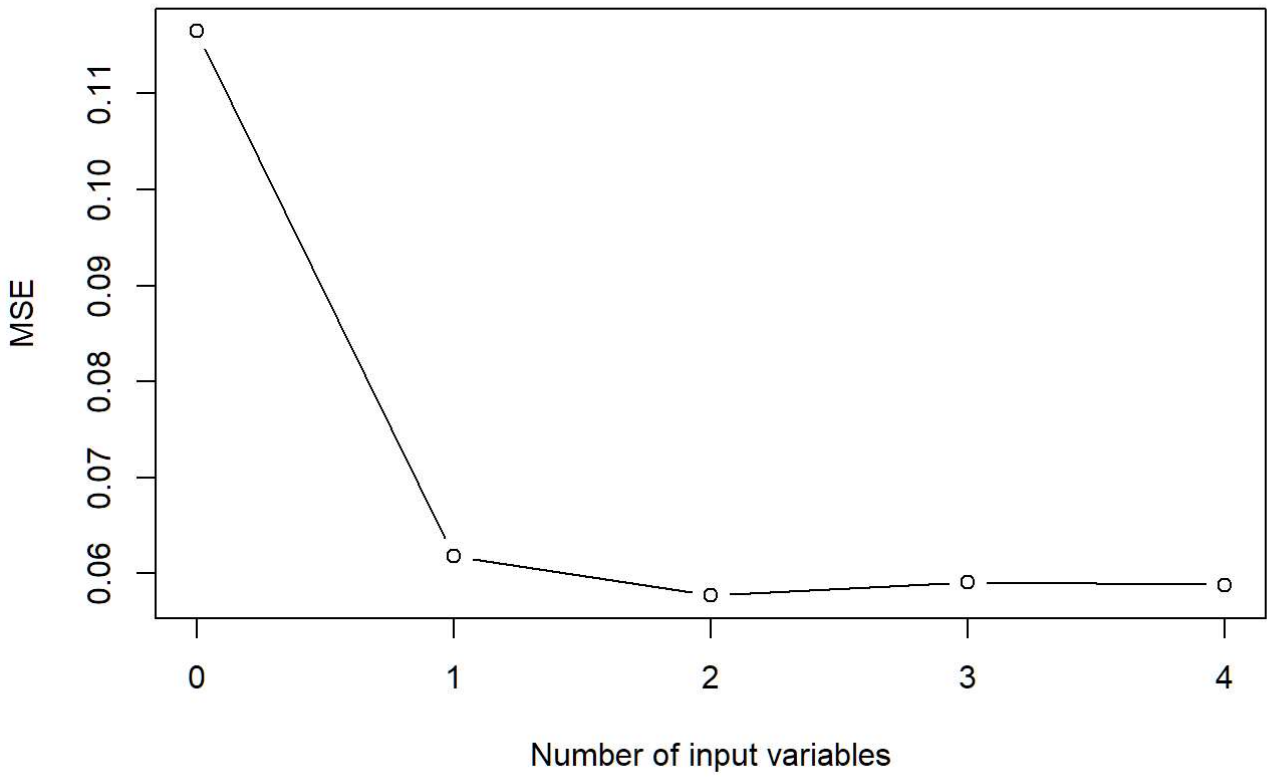
```
## [1] 4
```

```
# mod.3.4 is the winner

## 4 inputs
mod.4 <- lm(nox ~ poly(age,3) + ns(dis,4) + bs(lstat,8)
                        + ns(medv,5),
               data = RegressionData)
```

The mod.1.2 is the best in 2 variables model; mod.2.5 is the best in 3 variables model; the mod.3.4 is the best in 3 variables model. Now, let's comapre between these three model and null model as well as full model.

Cross-validation

```
RegressionData$const <- rep(0,dim(RegressionData)[1])
fit.0 <- train(nox ~ const,
               data = RegressionData, method = "lm",
               trCrontol = train_control)
fit.1 <- train(nox ~ ns(dis,4),
               data = RegressionData, method = "lm",
               trCrontol = train_control)
fit.2 <- train(nox ~ ns(dis,4) + ns(medv,5),
               data = RegressionData, method = "lm",
               trCrontol = train_control)
fit.3 <- train(nox ~ ns(dis,4) + bs(lstat,8) + ns(medv,5),
               data = RegressionData, method = "lm",
               trCrontol = train_control)
fit.4 <- train(nox ~ poly(age,3) + ns(dis,4) + bs(lstat,8)
                       + ns(medv,5),
              data = RegressionData, method = "lm",
               trCrontol = train_control)

mod.comp <- data.frame(MSE = c(fit.0$results$RMSE, fit.1$results$RMSE, fit.2$results$RMSE, fi
t.3$results$RMSE, fit.4$results$RMSE),
                       Inputs = c(0,1,2,3,4))
plot(mod.comp$Inputs, mod.comp$MSE, type = "b", xlab = "Number of input variables", ylab = "M
SE")
```




```
dfn_best <- mod.comp$Inputs[which.min(mod.comp$MSE)]

mod.comp$MSE[3]
```

```
## [1] 0.05771516
```

```
mod.comp$MSE[5]
```

```
## [1] 0.05883094
```

```
# Also, we can use ANOVA to compare the model performance
anova(mod.1.2, mod.2.5, mod.3.4, mod.4)
```

```
## Analysis of Variance Table
##
## Model 1: nox ~ ns(dis, 4)
## Model 2: nox ~ ns(dis, 4) + ns(medv, 5)
## Model 3: nox ~ ns(dis, 4) + bs(lstat, 8) + ns(medv, 5)
## Model 4: nox ~ poly(age, 3) + ns(dis, 4) + bs(lstat, 8) + ns(medv, 5)
##   Res.Df    RSS Df Sum of Sq       F    Pr(>F)
## 1    501 1.8858
## 2    496 1.6329  5  0.252865 15.8171 1.925e-14 ***
## 3    488 1.5780  8  0.054916  2.1469   0.03029 *
## 4    485 1.5507  3  0.027307  2.8469   0.03715 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Although the 4 input variables still have the lowest MSE, however, based on the ground of parsimony and if the truth is that we need to make a decision (choose to focus on something at expense of other parts) about the plan of city development, it would be better choosing the 2 input variables model, which is mod.2.5.

Meanwhile, for the results of ANOVA, we could see that the p-value comparing the 2-variable Model to the 3-variable Model is very low (1.925e-14), while the p-value between model 2-variable and model 3-variable as well as the 4-variable model is not that low (actually, if we consider the multiple comparison problem, the corrected p-value should be beyond 0.05). So, maybe a 2 variable model is enough in this case.

Now, let's calculate the parameter for this model.

```
summary(mod.2.5)
```

```
##
## Call:
## lm(formula = nox ~ ns(dis, 4) + ns(medv, 5), data = RegressionData)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.131035 -0.034707 -0.004922  0.027763  0.213256
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.698846   0.020492  34.104  < 2e-16 ***
## ns(dis, 4)1  -0.208664   0.015378 -13.569  < 2e-16 ***
## ns(dis, 4)2  -0.248323   0.016899 -14.695  < 2e-16 ***
## ns(dis, 4)3  -0.347434   0.032891 -10.563  < 2e-16 ***
## ns(dis, 4)4  -0.272266   0.029825  -9.129  < 2e-16 ***
## ns(medv, 5)1  0.011512   0.019666   0.585 0.558546
## ns(medv, 5)2  0.002627   0.022267   0.118 0.906144
## ns(medv, 5)3 -0.063655   0.018634  -3.416 0.000688 ***
## ns(medv, 5)4  0.035936   0.043541   0.825 0.409580
## ns(medv, 5)5 -0.069431   0.014723  -4.716 3.13e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.05738 on 496 degrees of freedom
## Multiple R-squared:  0.7592, Adjusted R-squared:  0.7548
## F-statistic: 173.7 on 9 and 496 DF,  p-value: < 2.2e-16
```

So our final predicted model for nitrogen oxides concentration includes weighted mean of distances to five Boston employment centers and median value of owner-occupied homes, these two variables can explain 75.48% variance of nitrogen oxides concentration in Boston.

Classification Problem

Because the classification problem is not quite dissimilar to regression mode, here we omit most of the narrative and just stress the different parts.

Read in and Tidy Data

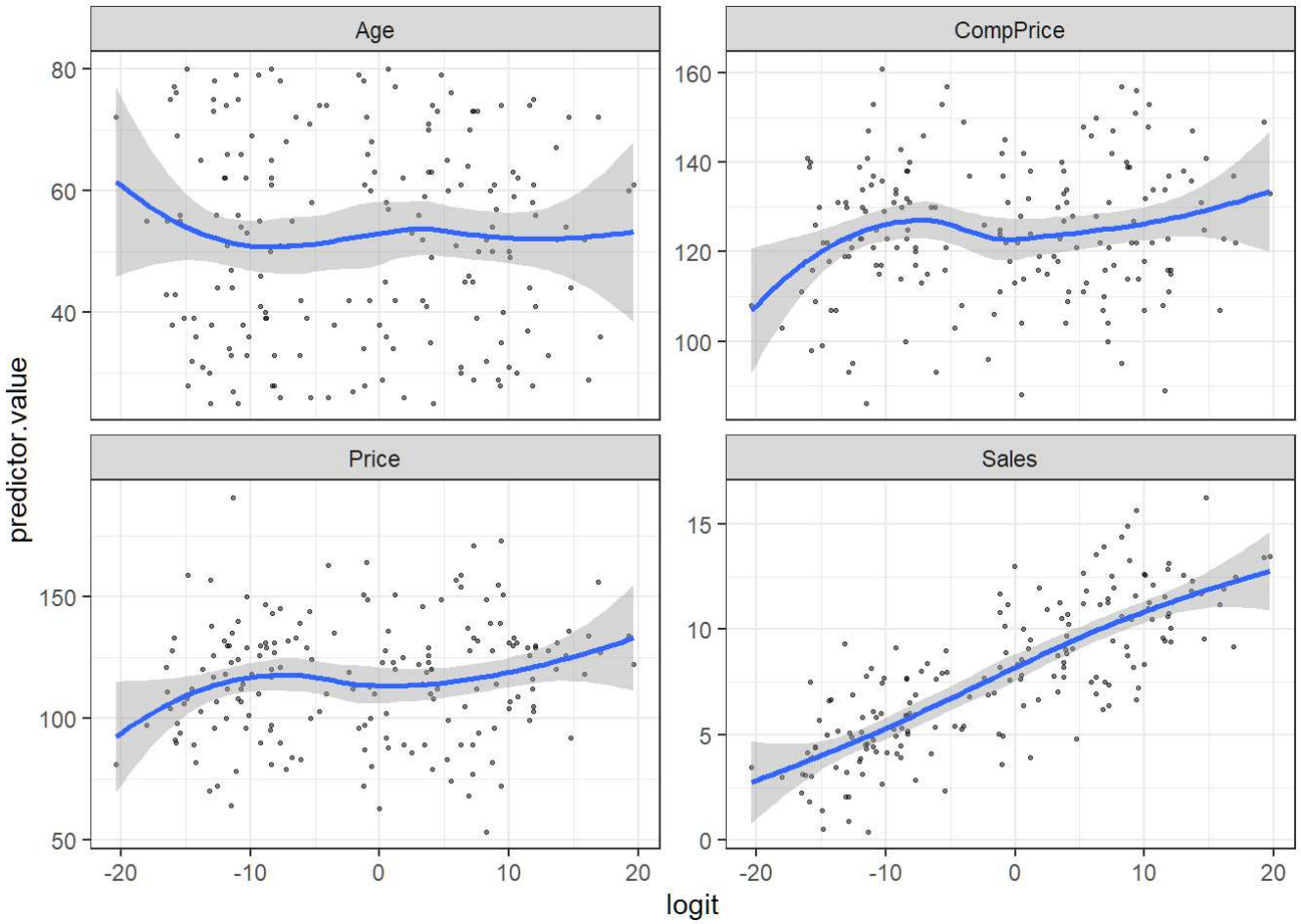
```
ClassificationData <- Carseats[Carseats$ShelveLoc=="Good"|Carseats$ShelveLoc=="Bad",]
ClassificationData <- ClassificationData[,c(1,2,6,7,8)]

ClassificationData$ShelveLoc <- droplevels(ClassificationData$ShelveLoc)

ClassificationData$ShelveLoc<- factor(ClassificationData$ShelveLoc, levels = c("Bad", "Good"
), labels = c('0', '1'))

str(ClassificationData)
```

```
## 'data.frame':    181 obs. of  5 variables:
##  $ Sales    : num  9.5 11.22 4.15 10.81 11.85 ...
##  $ CompPrice: num  138 111 141 124 136 121 117 115 107 118 ...
##  $ Price    : num  120 83 128 72 120 100 94 86 118 110 ...
##  $ ShelveLoc: Factor w/ 2 levels "0","1": 1 2 1 1 2 1 2 2 2 2 ...
##  $ Age      : num  42 65 38 78 67 26 50 53 52 63 ...
```

Assumption test 1

Linearity assumption

```
model <- glm(ShelveLoc ~., data = ClassificationData,
               family = binomial)
# Select only numeric predictors

probabilities <- predict(model, type = "response")
predicted.classes <- ifelse(probabilities > 0.5, "Bad", "Good")

# Bind the logit and tidying the data for plot
mydata <- ClassificationData %>%
  dplyr::select_if(is.numeric)
predictors <- colnames(mydata)
mydata <- mydata %>%
  mutate(logit = log(probabilities/(1-probabilities))) %>%
  gather(key = "predictors", value = "predictor.value", -logit)

# Create the scatter plots:
ggplot(mydata, aes(logit, predictor.value))+
  geom_point(size = 0.5, alpha = 0.5) +
  geom_smooth(method = "loess") +
  theme_bw() +
  facet_wrap(~predictors, scales = "free_y")
```



The smoothed scatter plots show that these variables are all quite linearly associated with *ShelveLoc* in logit scale, especially for the *Sales*. But the relationship between Age/ComPrice doesn't seem like linear or related. However, the *Price* variables are not 100% perfect, so I think it's fine to try to fit with other methods for those three variables.

Multicollinearity test

Now, let's check the Multicollinearity problems.

```
model <- glm(ShelveLoc ~., data = ClassificationData,
               family = binomial)
car::vif(model)
```

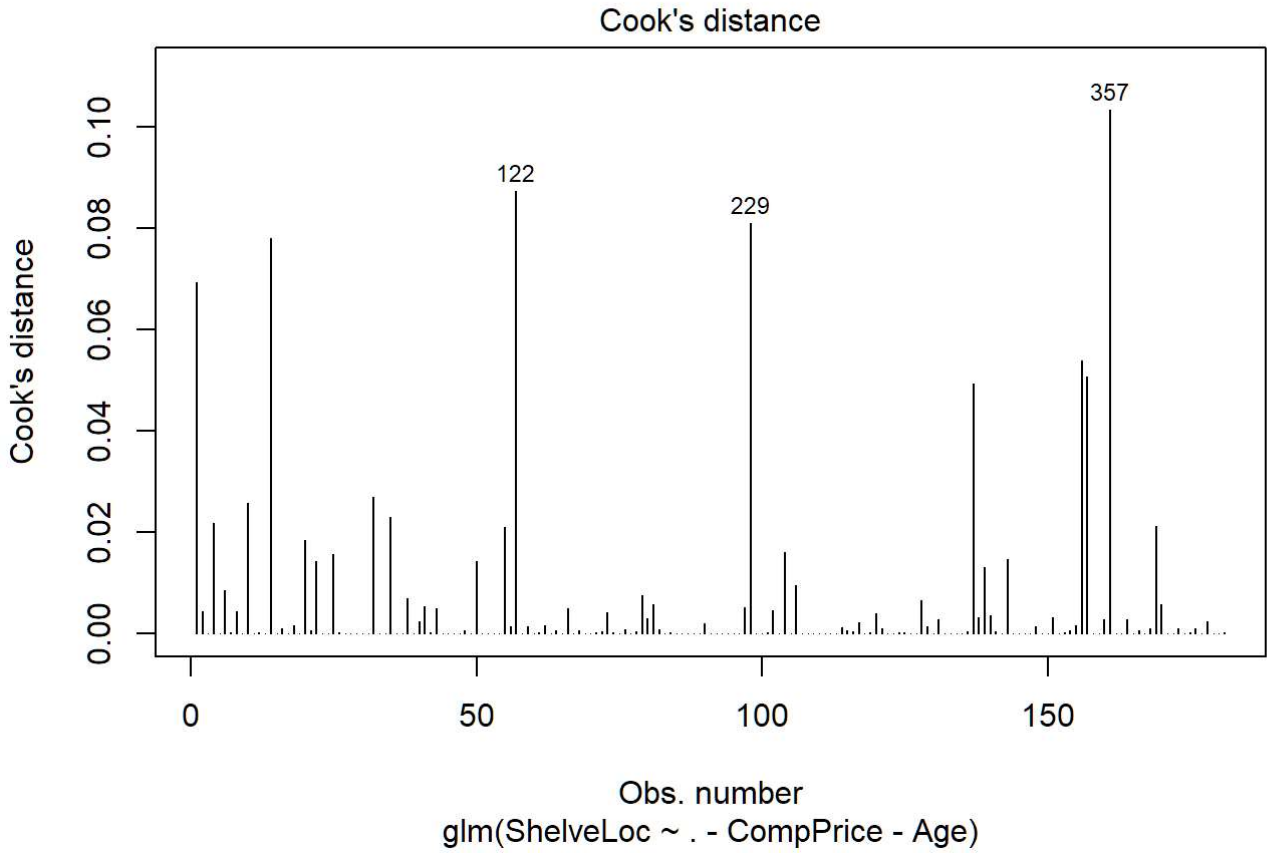
```
##     Sales CompPrice     Price       Age
## 26.637956  6.243205 30.044999  3.804585
```

The vif of *Sales* and *Price* as well as *ComPrice* >5. We should drop *Price*, which one has highest VIF. Thus, if later we want choose a linear model, we should at least drop several variables. Here, I decided to drop Age and CompPrice because these two variables seems like not associated with outcome.

```
model <- glm(ShelveLoc ~.-CompPrice-Age, data = ClassificationData,
               family = binomial)
car::vif(model)
```

```
##   Sales   Price
## 3.15735 3.15735
```

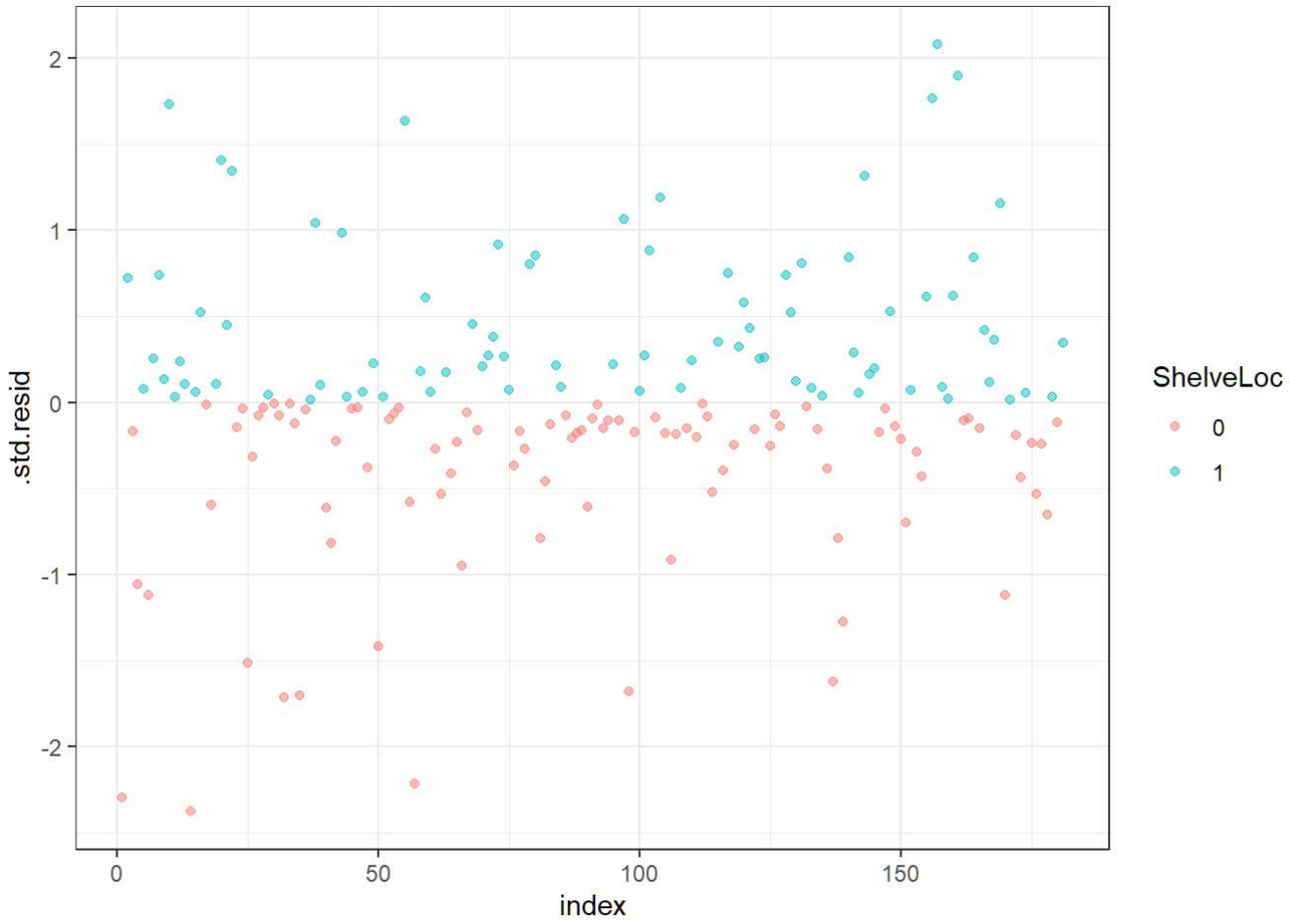
```
# Examine treme data by the Cook’s distance values.
plot(model, which = 4, id.n = 3)
```




```
# Extract model results
model.data <- augment(model) %>%
  mutate(index = 1:n())

model.data %>% top_n(3, .cooksd)
```

```
## # A tibble: 3 x 13
##   .rownames ShelveLoc Sales CompPrice Price   Age .fitted .resid .std.resid
##   <chr>     <fct>     <dbl>     <dbl> <dbl> <dbl>   <dbl>  <dbl>      <dbl>
## 1 122       0         11.7        125    87    28   2.30   -2.19      -2.22
## 2 229       0          5.4        149   163    26   0.983  -1.61      -1.68
## 3 357       1          3.58       142   164    72  -1.49    1.84       1.90
## # ... with 4 more variables: .hat <dbl>, .sigma <dbl>, .cooksd <dbl>,
## #   index <int>
```

```
# Plot the standardized residuals:
ggplot(model.data, aes(index, .std.resid)) +
  geom_point(aes(color = ShelveLoc), alpha = .5) +
  theme_bw()
```




```
# Filter potential influential data points with abs(.std.res) > 3:
model.data %>%
  filter(abs(.std.resid) > 3)
```

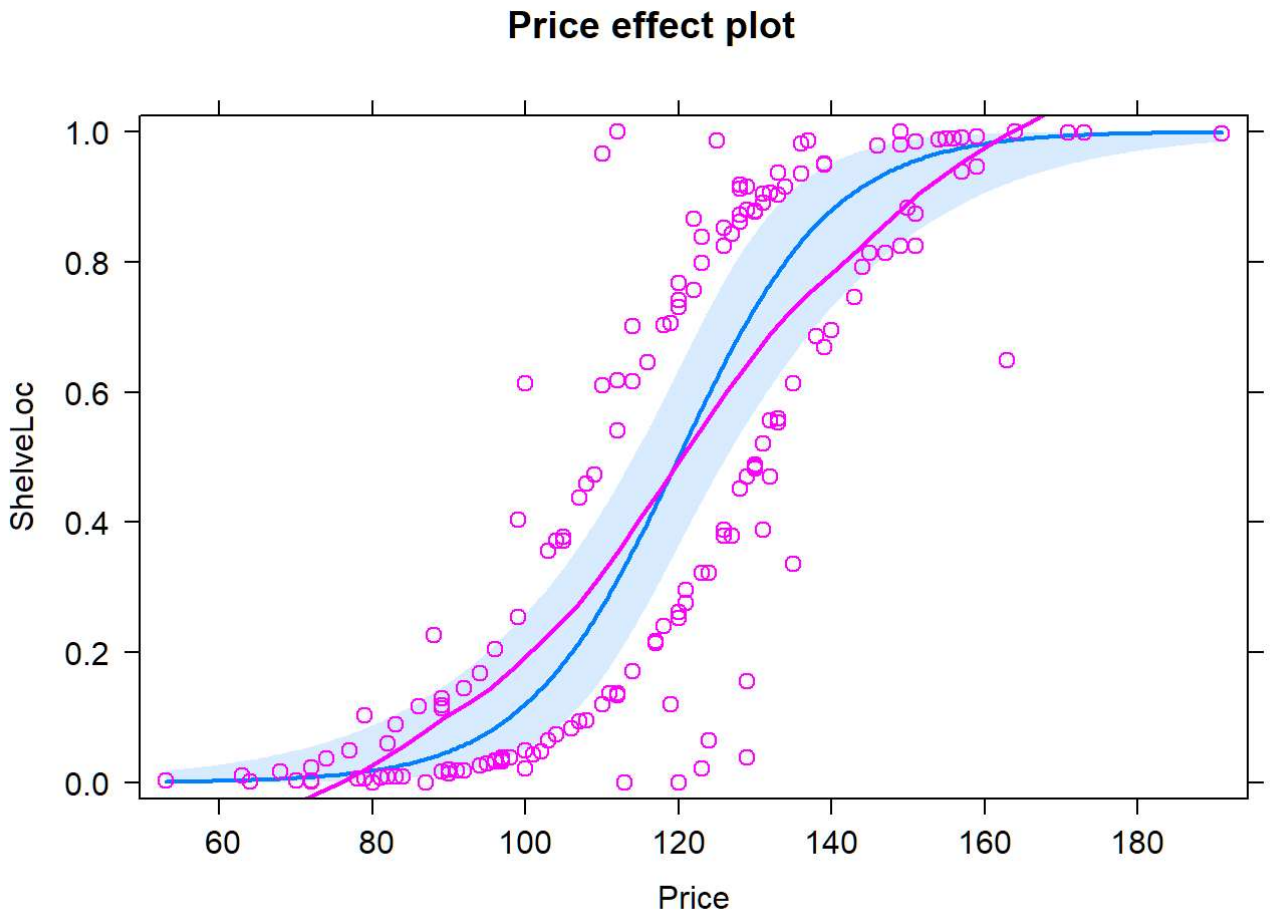
```
## # A tibble: 0 x 13
## # ... with 13 variables: .rownames <chr>, ShelveLoc <fct>, Sales <dbl>,
## #   CompPrice <dbl>, Price <dbl>, Age <dbl>, .fitted <dbl>, .resid <dbl>,
## #   .std.resid <dbl>, .hat <dbl>, .sigma <dbl>, .cooksd <dbl>, index <int>
```

As results above, there is some outliers, but no influential observations in our data.

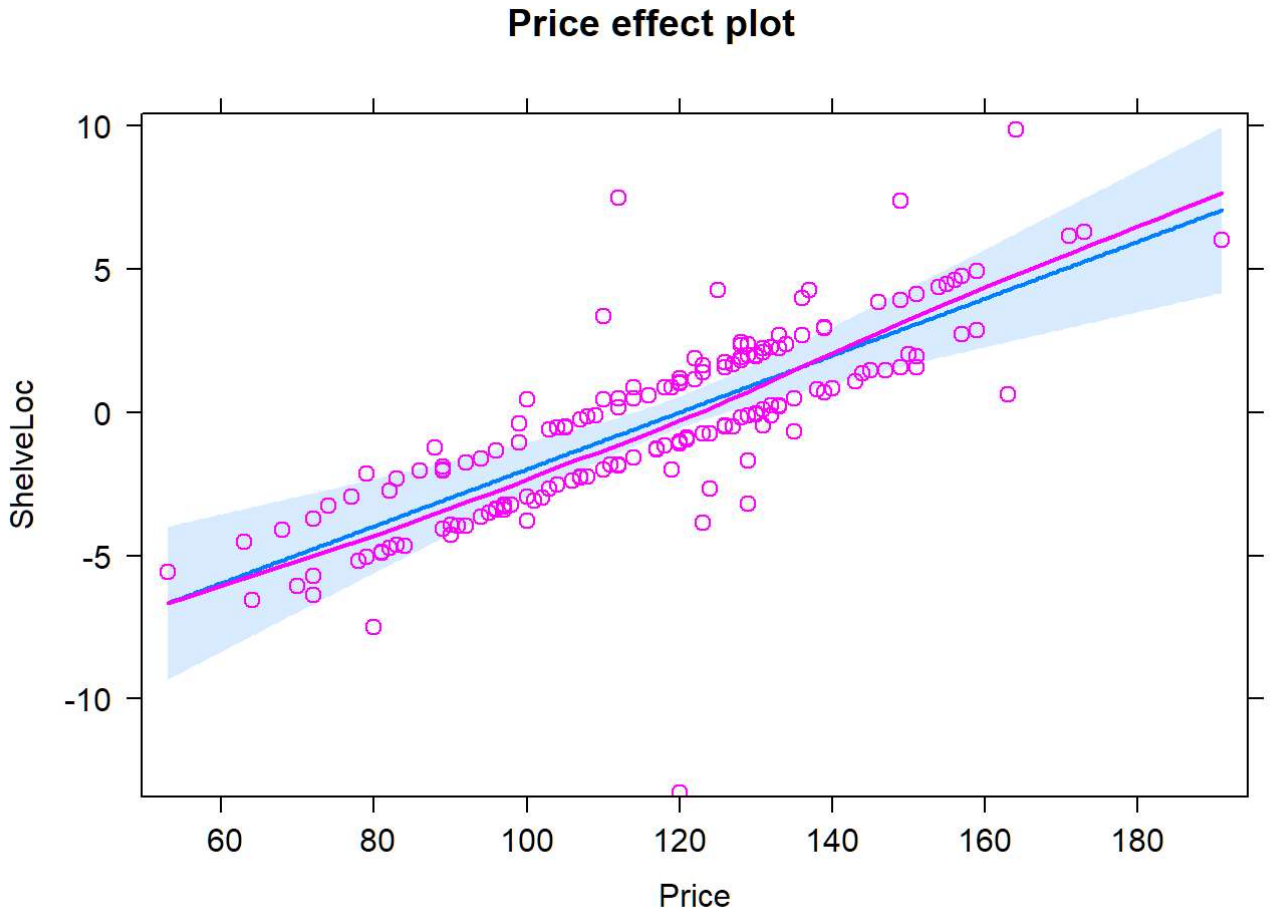
Fitting *Price*

Again, let's plot the residual value of the model in two different scales.

```
plot(effect("Price", mod = model, partial.residuals=TRUE), type="response")
```



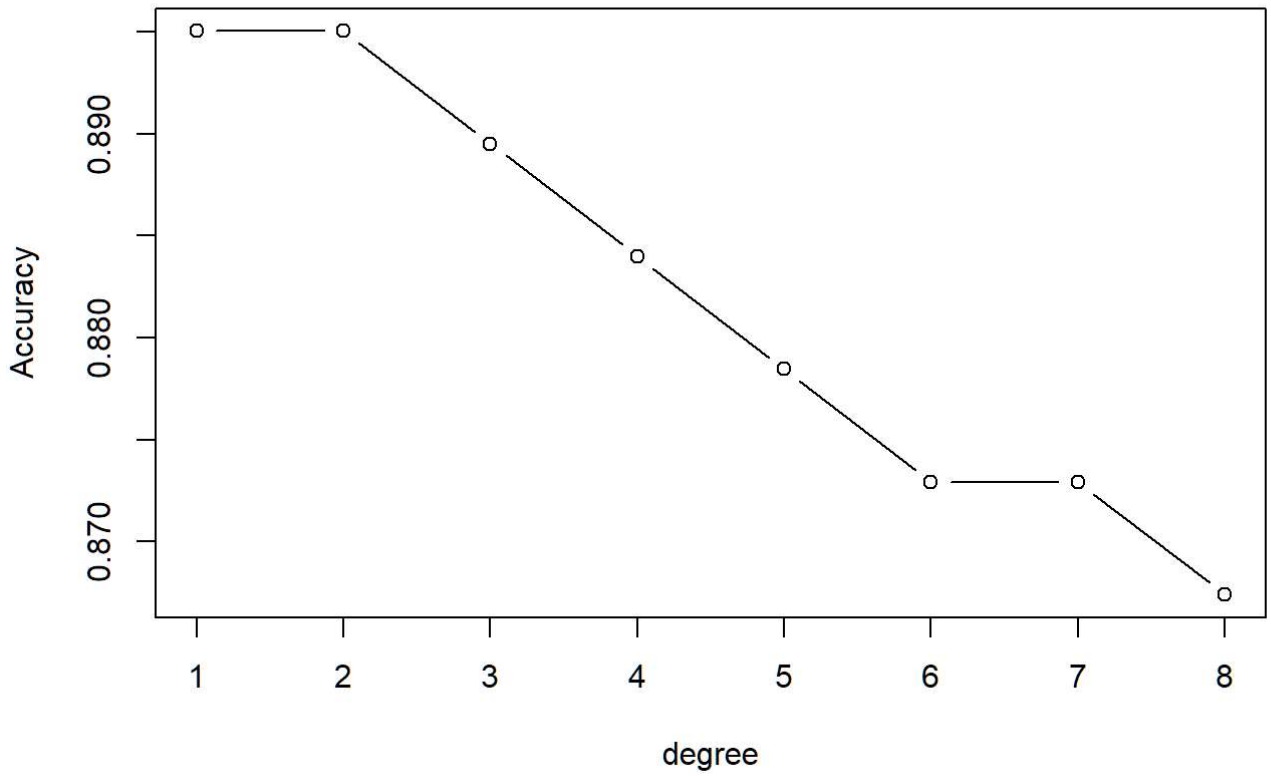

```
plot(effect("Price", mod = model, partial.residuals=TRUE), type="link")
```



It seems like there is no non-linear problems for *Price* in logic scales.

2.1.1 Refit *Price* with Polynomial

```
set.seed(1234)
train_control <- trainControl(method="LOOCV")

fit.1 <- train(ShelveLoc ~ Sales + Price,
               method = "glm",
               data = ClassificationData,
               trControl = train_control,
               family = "binomial")
fit.2 <- train(ShelveLoc ~ Sales + poly(Price,2),
               method = "glm",
               data = ClassificationData,
               trControl = train_control,
               family = "binomial")
fit.3 <- train(ShelveLoc ~ Sales + poly(Price,3),
               method = "glm",
               data = ClassificationData,
               trControl = train_control,
               family = "binomial")
fit.4 <- train(ShelveLoc ~ Sales + poly(Price,4),
               method = "glm",
               data = ClassificationData,
               trControl = train_control,
               family = "binomial")
fit.5 <- train(ShelveLoc ~ Sales + poly(Price,5),
               method = "glm",
               data = ClassificationData,
               trControl = train_control,
               family = "binomial")
fit.6 <- train(ShelveLoc ~ Sales + poly(Price,6),
               method = "glm",
               data = ClassificationData,
               trControl = train_control,
               family = "binomial")
fit.7 <- train(ShelveLoc ~ Sales + poly(Price,7),
               method = "glm",
               data = ClassificationData,
               trControl = train_control,
               family = "binomial")
fit.8 <- train(ShelveLoc ~ Sales + poly(Price,8),
               method = "glm",
               data = ClassificationData,
               trControl = train_control,
               family = "binomial")

Accuracy <- rep(0,8)
Accuracy[1] <- fit.1$results$Accuracy
Accuracy[2] <- fit.2$results$Accuracy
Accuracy[3] <- fit.3$results$Accuracy
Accuracy[4] <- fit.4$results$Accuracy
Accuracy[5] <- fit.5$results$Accuracy
Accuracy[6] <- fit.6$results$Accuracy
Accuracy[7] <- fit.7$results$Accuracy
Accuracy[8] <- fit.8$results$Accuracy

plot(1:8, Accuracy, type = "b", xlab = "degree")
```




```
which.max(Accuracy)
```

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

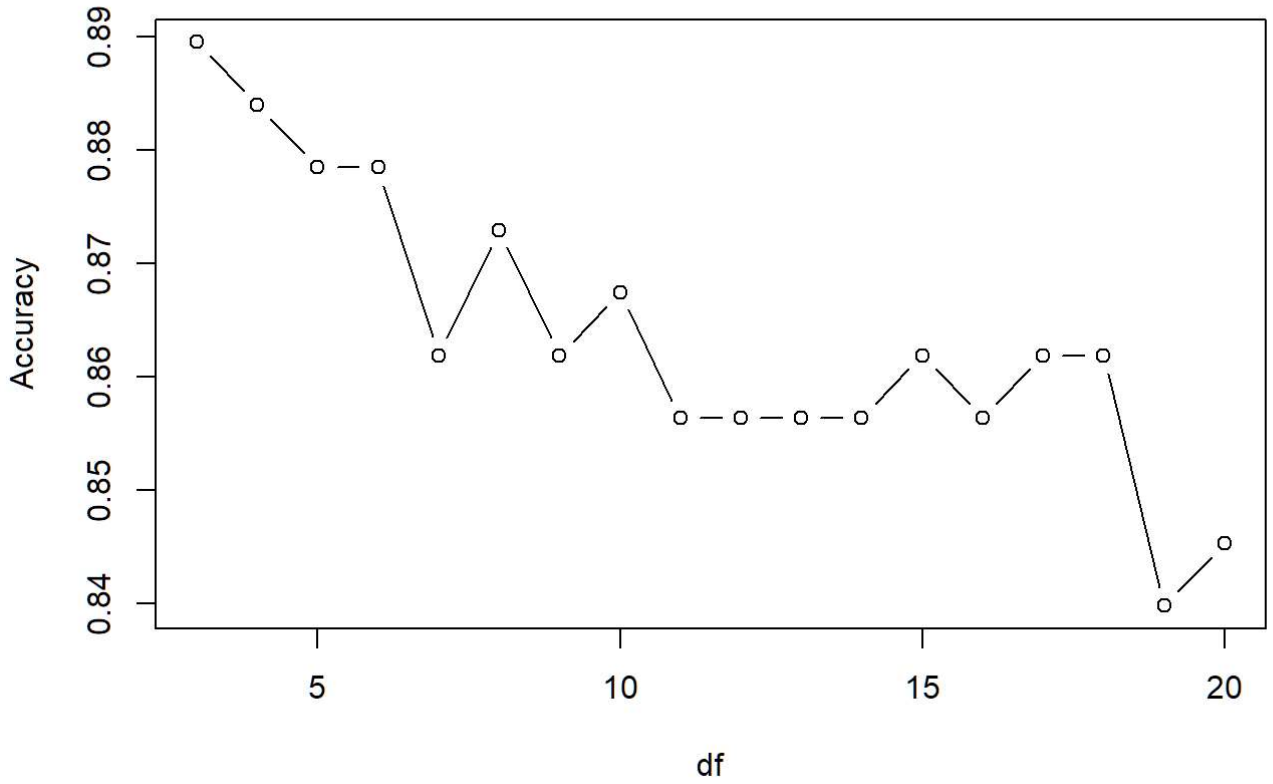
```
Accuracy[which.max(Accuracy)]
```

```
## [1] 0.8950276
```

We choose linear model for *Price* (Accuracy = 0.895).

2.1.2 Refit *Price* with Basis Spline

```
Accuracy<- rep(0,18)

for (i in 3:20){
  Data <- ClassificationData
  ClassificationData$Pricebs <- bs(ClassificationData$Price, df = i)
  fit <- train(ShelveLoc ~ Sales + Pricebs,
               method = "glm",
               data = ClassificationData,
               trControl = train_control,
               family = "binomial")
  Accuracy[i-2] <- fit$results$Accuracy
}
plot(3:20, Accuracy, type='b', xlab = 'df')
```




```
dfn <- which.max(Accuracy) + 2
dfn
```

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

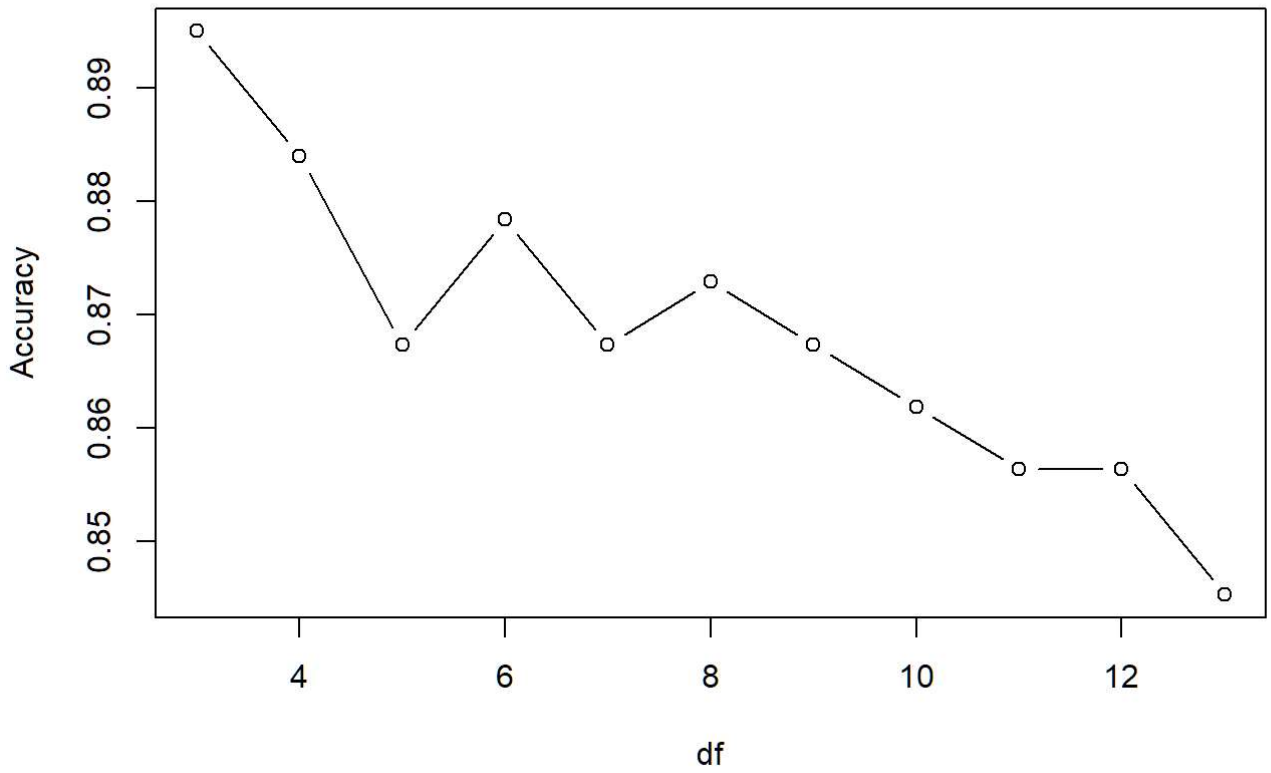
```
Accuracy[which.max(Accuracy)]
```

```
## [1] 0.8895028
```

We choose basis spline with df = 3 for *Price* (Accuracy = 0.890).

2.1.3 Refit *Price* with Nature Spline

```
Accuracy <- rep(0,10)
for (i in 3:13){
  Data <- ClassificationData
  ClassificationData$Pricens <- ns(ClassificationData$Price, df = i)
  fit <- train(ShelveLoc ~ Sales + Pricens,
               method = "glm",
               data = ClassificationData,
               trControl = train_control,
               family = "binomial")
  Accuracy[i-2] <- fit$results$Accuracy
}
plot(3:13, Accuracy, type='b', xlab = 'df')
```




```
dfn <- which.max(Accuracy) + 2
dfn
```

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

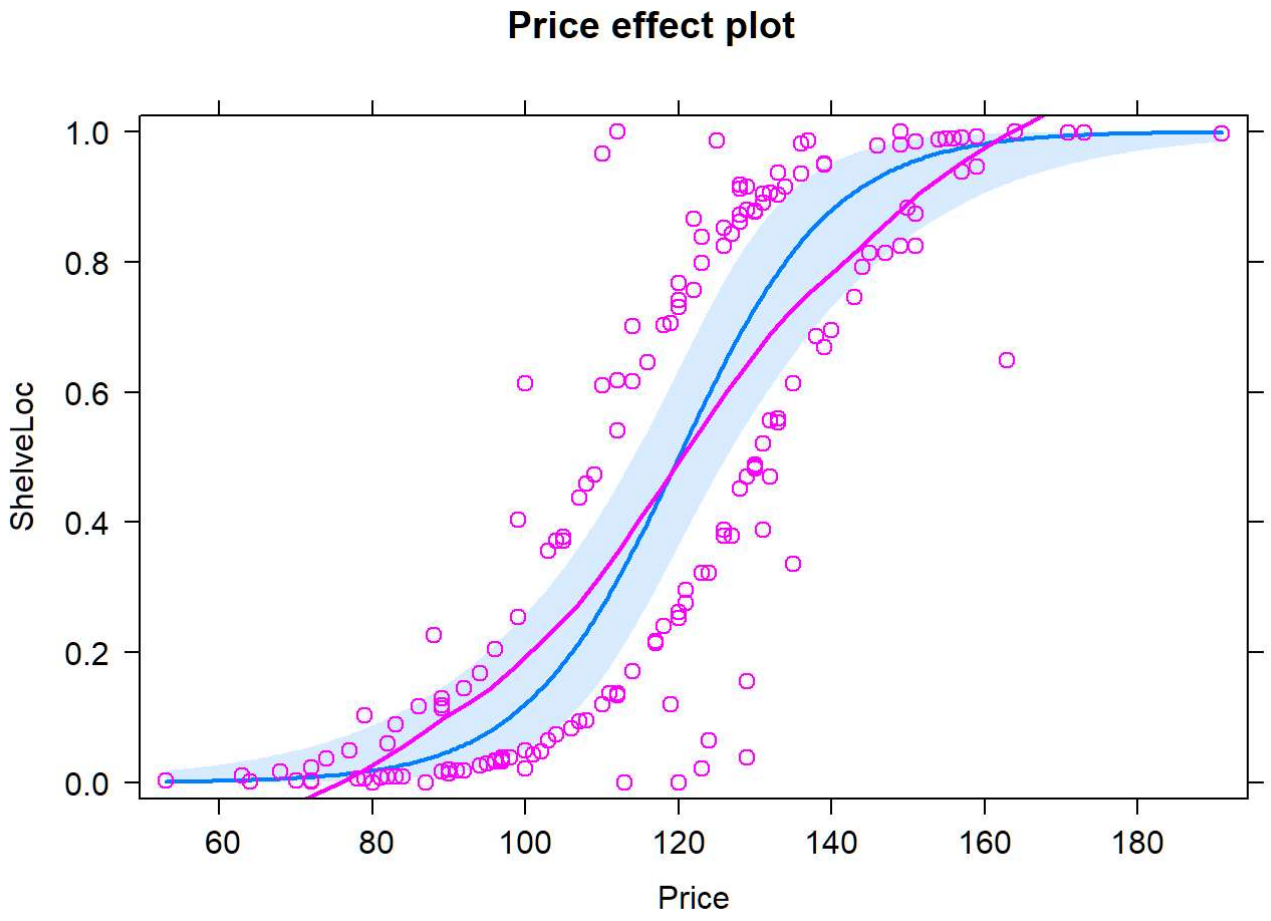
```
Accuracy[which.max(Accuracy)]
```

```
## [1] 0.8950276
```

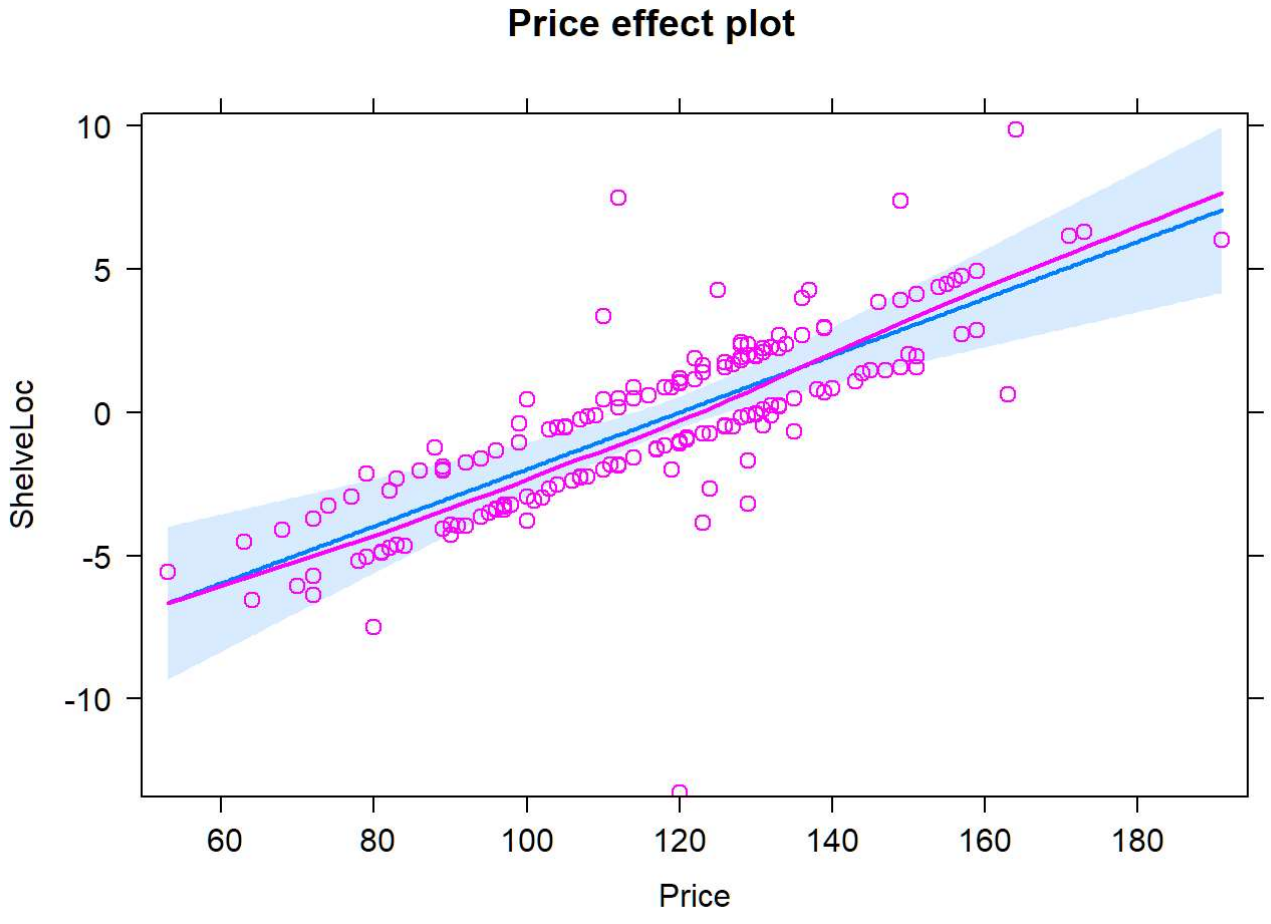
Here we choose natural spline with df = 3 (Accuracy = 0.895).

Let's compare linear fit and natural spline fit.

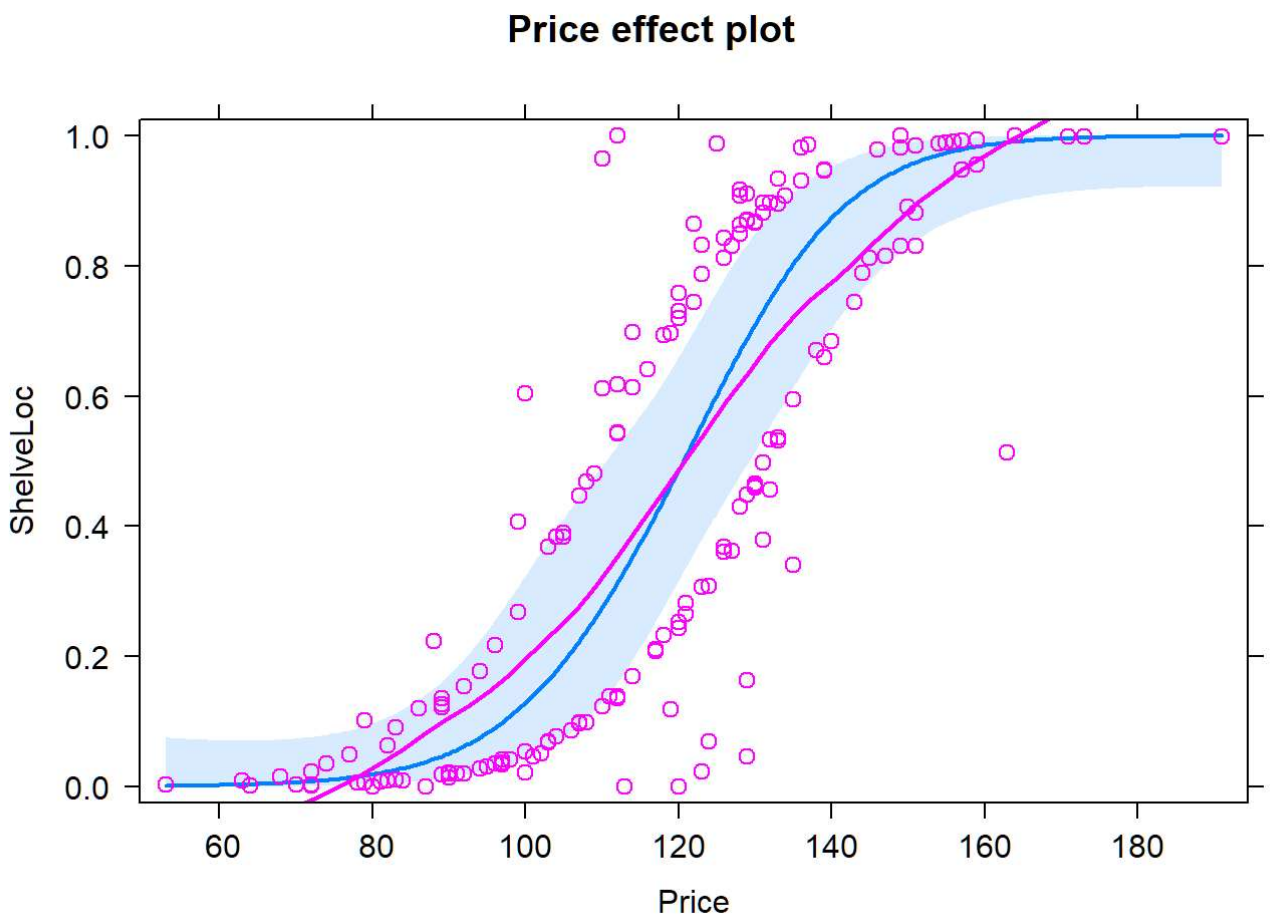
```
model.lm.1 <-glm(ShelveLoc ~ Sales + Price,
               data = ClassificationData,
               family = "binomial")

model.ns.3 <- glm(ShelveLoc ~ Sales + ns(Price,3),
               data = ClassificationData,
               family = "binomial")
plot(effect("Price", model.lm.1, partial.residuals=TRUE), type="response")
```



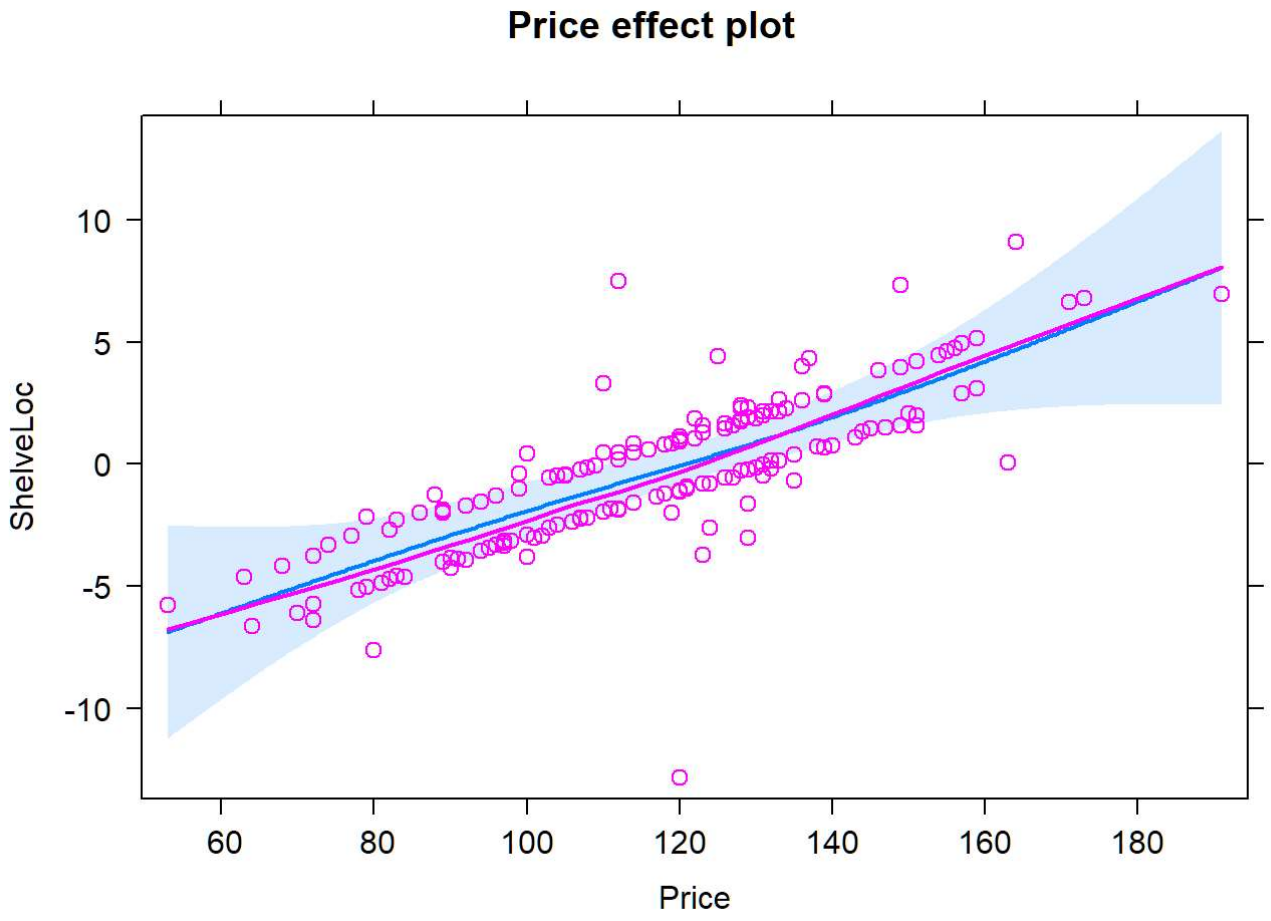

```
plot(effect("Price", model.lm.1, partial.residuals=TRUE), type="link")
```



```
plot(effect("Price", model.ns.3, partial.residuals=TRUE), type="response")
```




```
plot(effect("Price", model.ns.3, partial.residuals=TRUE), type="link")
```



No obvious different, so here, I decided to choose linear fitting for *Price*.

Now that *Price* variable has been determined and I think there is no nonlinear problem for *Sales* variable, thus, the final classification predicted model has been decided.

Best subset selection

Model list

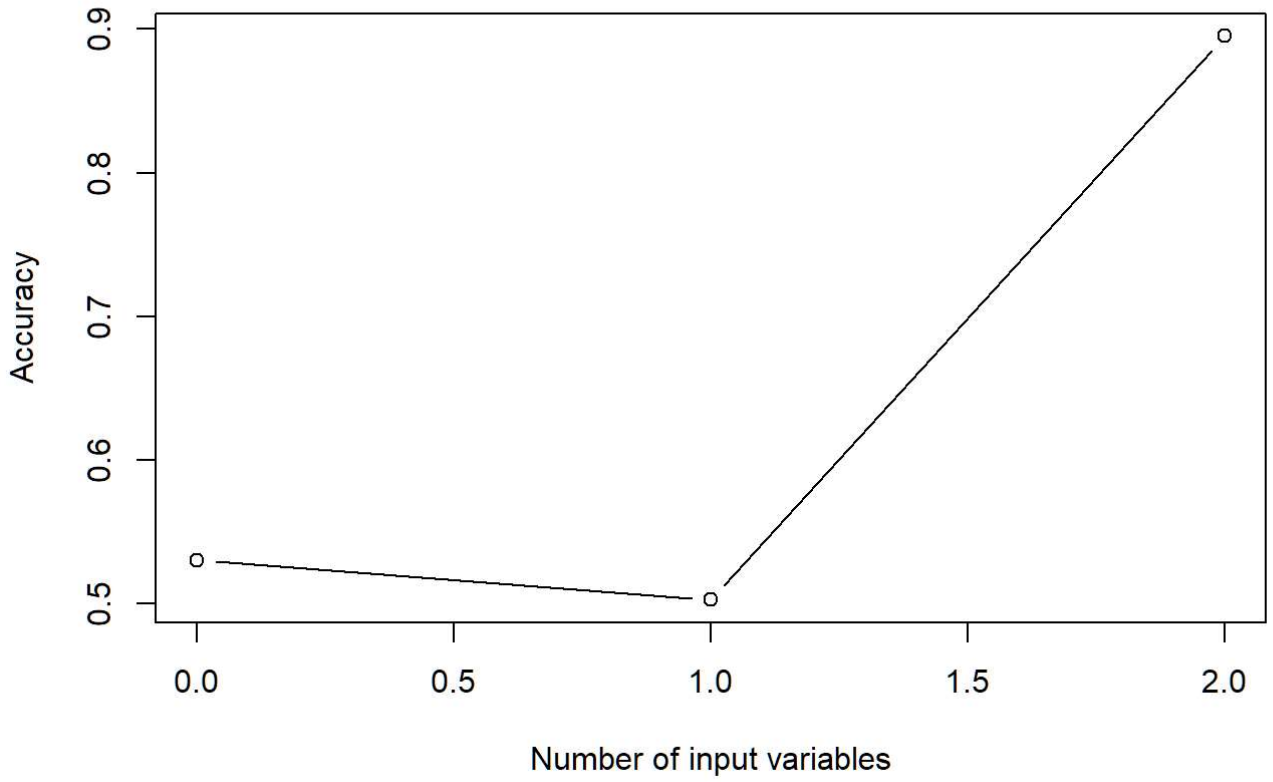
```
## 0 puts
mod.0 <- glm(ShelveLoc ~ 1,
             data = ClassificationData,
             family = "binomial")
## 1 puts
mod.1.1 <- glm(ShelveLoc ~ Sales,
             data = ClassificationData,
             family = "binomial")
mod.1.2 <- glm(ShelveLoc ~ Price,
             data = ClassificationData,
             family = "binomial")
which.max(c(deviance(mod.1.1), deviance(mod.1.2)))
```

```
## [1] 2
```

```
## 2 puts
mod.2.1 <- glm(ShelveLoc ~ Sales + Price,
             data = ClassificationData,
             family = "binomial")
```

Cross-validation

```
ClassificationData$const <- rep(0,dim(ClassificationData)[1])
fit.0 <- train(ShelveLoc ~ const,
               data = ClassificationData,
               method = "glm",
               trControl = train_control,
               family = "binomial")
fit.1 <- train(ShelveLoc ~ Price,
               data = ClassificationData,
               method = "glm",
               trControl = train_control,
               family = "binomial")
fit.2 <- train(ShelveLoc ~ Sales + Price,
               data = ClassificationData,
               method = "glm",
               trControl = train_control,
               family = "binomial")

mod.comp <- data.frame(Accuracy = c(fit.0$results$Accuracy,
                                    fit.1$results$Accuracy,
                                    fit.2$results$Accuracy),
                       Inputs = c(0,1,2))
plot(mod.comp$Inputs, mod.comp$Accuracy, type = "b", xlab = "Number of input variables", ylab
= "Accuracy")
```



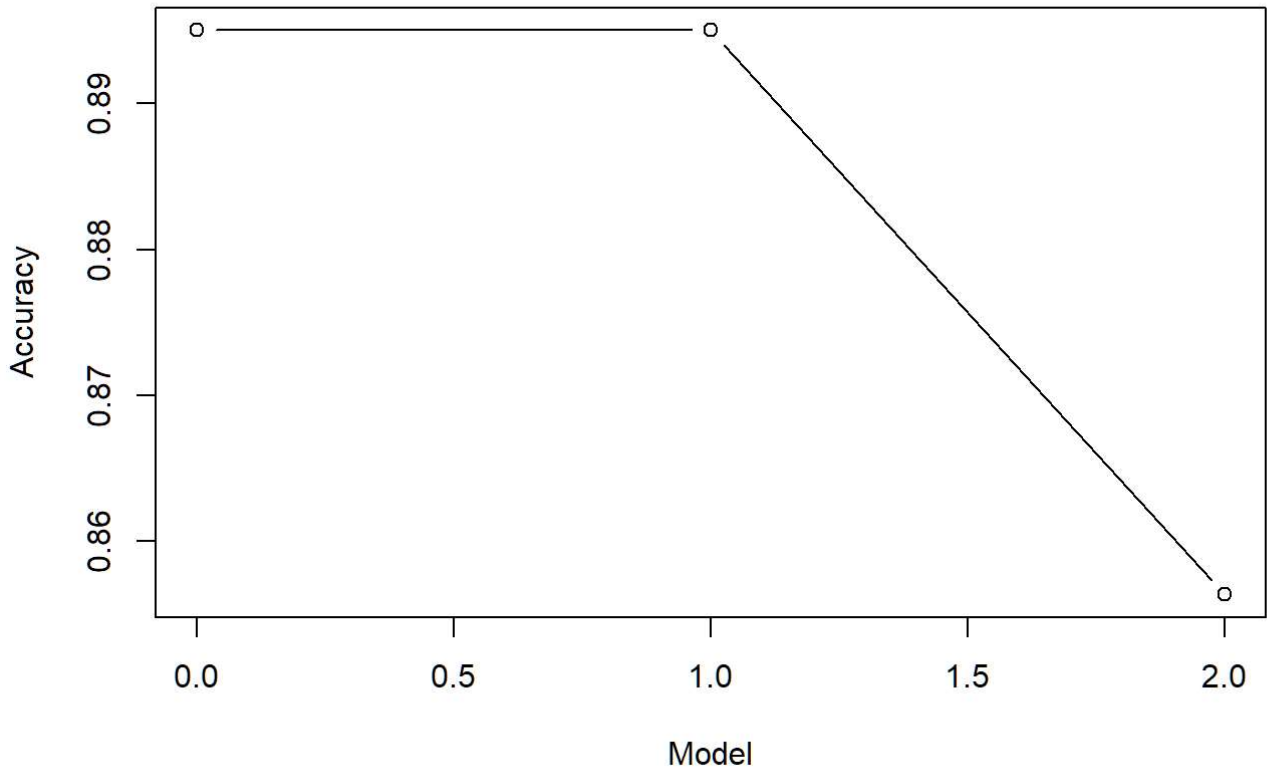

```
dfn_best <- mod.comp$Inputs[which.max(mod.comp$Accuracy)]
dfn_best
```

```
## [1] 2
```

This result suggested that the 2-variables model is the best.

Test cllassification model with QDA and KNN

```
fit.QDA <- train(ShelveLoc ~ Sales + Price,
                 data = ClassificationData,
                 method = "qda",
                 trControl = train_control)
fit.KNN <- train(ShelveLoc ~ Sales + Price,
                 data = ClassificationData,
                 method = "knn",
                 trControl = train_control,
                 preProcess = c("center","scale"))

mod.comp <- data.frame(Accuracy = c(fit.2$results$Accuracy,
                                    fit.QDA$results$Accuracy,
                                    fit.KNN$results$Accuracy[1]),
                       Inputs = c(0,1,2))

plot(mod.comp$Inputs, mod.comp$Accuracy, type = "b", xlab = "Model", ylab = "Accuracy")
```




```
dfn_best <- mod.comp$Inputs[which.max(mod.comp$Accuracy)]
dfn_best
```

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

These results suggested 2-variable model fitted by GLM is still the best.

Interpretation for Classification model

```
model = mod.2.1
summary(model)
```

```
##
## Call:
## glm(formula = ShelveLoc ~ Sales + Price, family = "binomial",
##     data = ClassificationData)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.35839  -0.20948  -0.02382   0.25590   2.05785
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -22.83091    3.97778  -5.740 9.49e-09 ***
## Sales         1.41315    0.22515   6.276 3.46e-10 ***
## Price         0.09928    0.02014   4.930 8.22e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 250.250  on 180  degrees of freedom
## Residual deviance:  83.767  on 178  degrees of freedom
## AIC: 89.767
##
## Number of Fisher Scoring iterations: 7
```

According to the estimated parameter in the table above, we could know each one-unit change in *Sales* will increase the log odds of choosing “good location” by 1.413, and each one-unit change in *Price* will increase the log odds of choosing “good location” by 0.099. Both p-value < 0.05 indicates that it is somewhat significant in determining the location.