LDA| QDA | SVM | polynomial Regression | B-spline | natrual cubic spline

Yimei Fan

Create a binary variable `crim01`, and regenerate the training set and test set with ratio 2:1.

```
library(MASS)
Boston$crim01[Boston$crim >= median(Boston$crim)] = 1
Boston$crim01[Boston$crim < median(Boston$crim)] = 0

bos_new = Boston[, c("crim01", "indus","nox","age","dis","rad","tax")]

train = sample(1: nrow(bos_new),2/3*nrow(bos_new))
test = (-train)

train_set = bos_new[train,]
test_set = bos_new[test,]
```

Perform LDA on the training data.

The accuracy is 0.852071.
False Positive Rate: 0.2.
False Negative Rate: 0.08860759.

```
lda.fit = lda(crim01~.,data = train_set)
lda.pred = predict(lda.fit, test_set)

lda.class = lda.pred$class

table(lda.class,test_set$crim01)
```

```
##
## lda.class  0  1
##         0 87 13
##         1 10 59
```

```
mean(lda.class == test_set$crim01)
```

```
## [1] 0.8639053
```

```
#False Positive Rate
18/(72+18)
```

```
## [1] 0.2
```

```r
#False Negative Rate
7/(7+72)
```

```
## [1] 0.08860759
```

```
Perform QDA on the training data.

According to (b), except for `tax`, all other variables are well-associated with `crim01`.

The accuracy is 0.887574.
The False Positive Rate is 0.1979167.
The False Negative Rate is 0.02739726.
```

```r
qda.fit = qda(crim01~ indus + nox + age + dis + rad,data = train_set)
qda.pred = predict(qda.fit,test_set)

qda.class = qda.pred$class

table(qda.class, test_set$crim01)
```

```
##
## qda.class  0  1
##         0 94 14
##         1  3 58
```

```r
mean(qda.class == test_set$crim01)
```

```
## [1] 0.8994083
```

```r
#False Positive Rate
19/(77+19)
```

```
## [1] 0.1979167
```

```r
#False Negative Rate
2/(2+71)
```

```
## [1] 0.02739726
```

```
Perform KNN on the training data using cross validation to decide $k$.

From cross-validation result, k =1 will give us the highest accuracy, which is 0.9151796.
```

```r
train.x = train_set[,-1]
test.x = test_set[,-1]
train.y = train_set[,1]
test.y = test_set[,1]

library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```
set.seed(100)
trControl = trainControl(method = "cv",number = 5)
fit_knn_cv = train(as.factor(crim01)~indus+ nox + age + dis + rad +tax,method = "knn",
         tuneGrid = expand.grid(k = 1:10),
         trControl  = trControl,
         metric     = "Accuracy",
         data       = bos_new)
fit_knn_cv
```

```
## k-Nearest Neighbors
##
## 506 samples
##   6 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 404, 405, 404, 406, 405
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.9151796  0.8304011
##    2  0.8815535  0.7631795
##    3  0.9091990  0.8184131
##    4  0.9111404  0.8222900
##    5  0.9091800  0.8183498
##    6  0.9051800  0.8103685
##    7  0.9111602  0.8223192
##    8  0.9092192  0.8184314
##    9  0.9051800  0.8103529
##   10  0.8992976  0.7985882
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 1.
```

```
knn_best = knn(train.x,test.x,train.y,k=1)
```

Fit a support vector classifier to the training data using cross validation.

cost = 5 results in lowest cross-validation error rate, which is 0.1221925.The accuracy of the best mode

```
set.seed(100)
  tune.out = tune(svm, as.factor(crim01)~., data = train_set,kernel = "linear", ranges = list(cost = c
  summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
```

```
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    50
##
## - best performance: 0.1722816
##
## - Detailed performance results:
##       cost     error dispersion
## 1    0.001 0.2016934 0.06478989
## 2    0.010 0.2045455 0.07749428
## 3    0.100 0.1986631 0.06163918
## 4    1.000 0.1723708 0.06153277
## 5    5.000 0.1812834 0.07051983
## 6   10.000 0.1812834 0.06122648
## 7   15.000 0.1812834 0.06122648
## 8   20.000 0.1812834 0.06122648
## 9   50.000 0.1722816 0.05616946
## 10 100.000 0.1723708 0.06467284
```

```
best = tune.out$best.model
pred_svm = predict(best,test_set)

table(predict = pred_svm, truth = test_set$crim01)
```

```
##        truth
## predict  0  1
##       0 88  7
##       1  9 65
```

```
(81+72)/(81+72+9+7)
```

```
## [1] 0.9053254
```

```
when using "polynomial" kernels, cost = 0.1 results in lowest cv error rate, which is 0.5164884, and the
When using "raidal" kernels, cost = 100 results in lowest cv error rate, which is 0.06247772, and the ac
```

```
set.seed(100)
tune_poly = tune(svm,as.factor(crim01)~., data = train_set, kernel = "polynomial", ranges = list(cos


set.seed(100)
tune_radial = tune(svm,as.factor(crim01)~., data = train_set, kernel = "radial", ranges = list(cost

tune_radial$best.parameters$degree
```

```
## NULL
```

```r
par(mfrow = c(2,3))
summary(tune_poly)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.4634581
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-01 0.4634581  0.1026758
## 2 1e+00 0.4634581  0.1026758
## 3 1e+01 0.4634581  0.1026758
## 4 1e+02 0.4634581  0.1026758
## 5 1e+03 0.4634581  0.1026758
```

```r
summary(tune_radial)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.07139037
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-01 0.16051693 0.05574580
## 2 1e+00 0.07139037 0.02919475
## 3 1e+01 0.08012478 0.03450182
## 4 1e+02 0.08021390 0.04507921
## 5 1e+03 0.08636364 0.05242321
```

```r
table(predict(tune_poly$best.model, test_set), test_set$crim01)
```

```
##
##      0  1
##   0  0  0
##   1 97 72
```

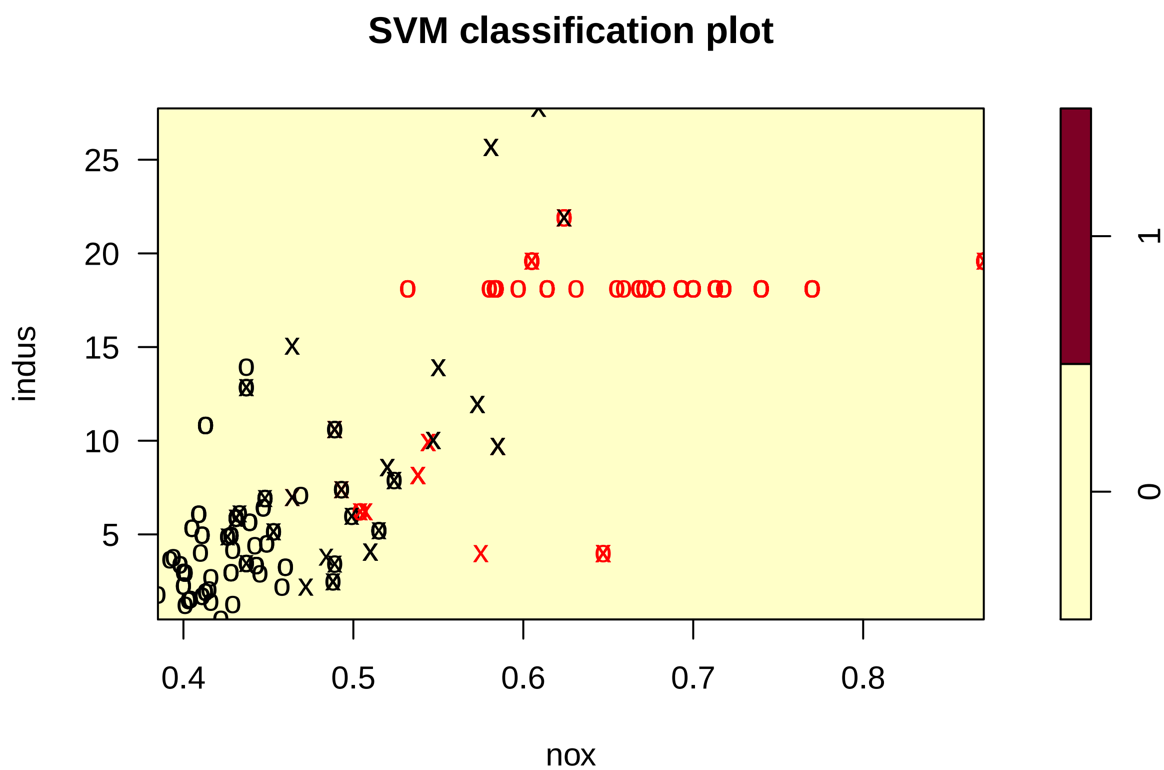
```r
#81/88
table(predict(tune_radial$best.model, test_set),test_set$crim01)
```

```
##
##       0  1
##    0 93  2
##    1  4 70
```

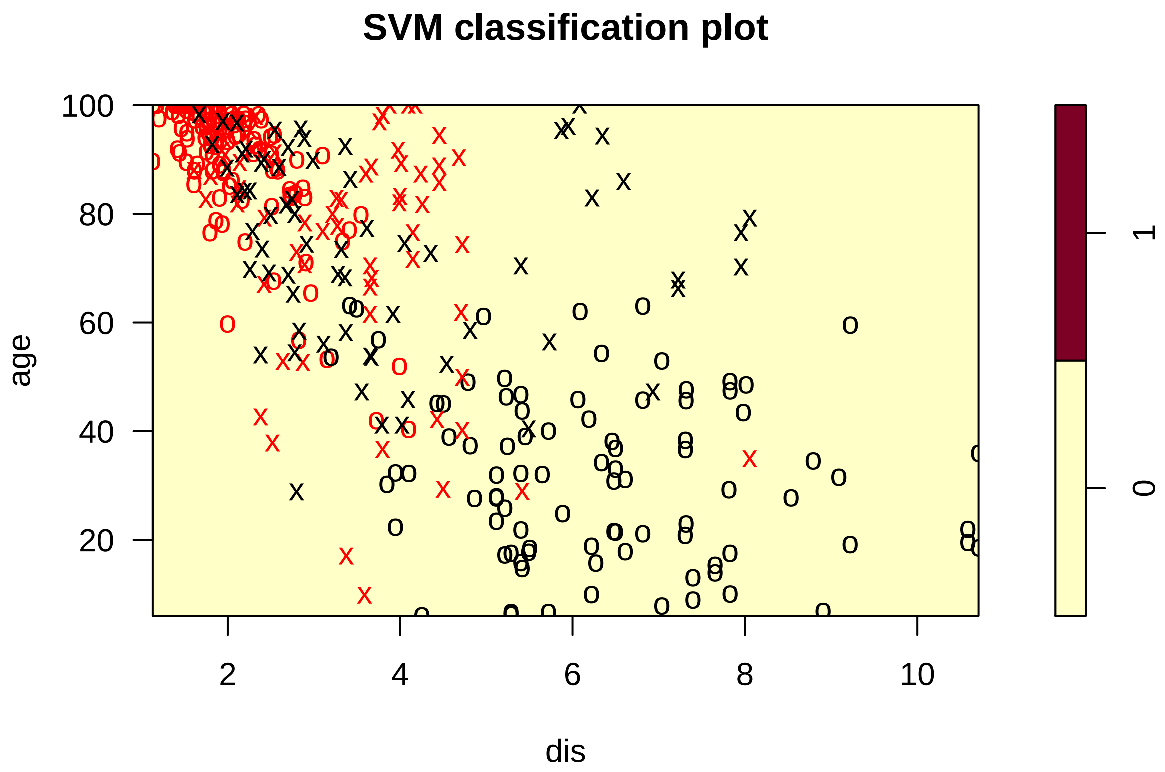
```r
#(84+75)/(84+75+4+6)
```

Fit the svm model with kernel = "linear".

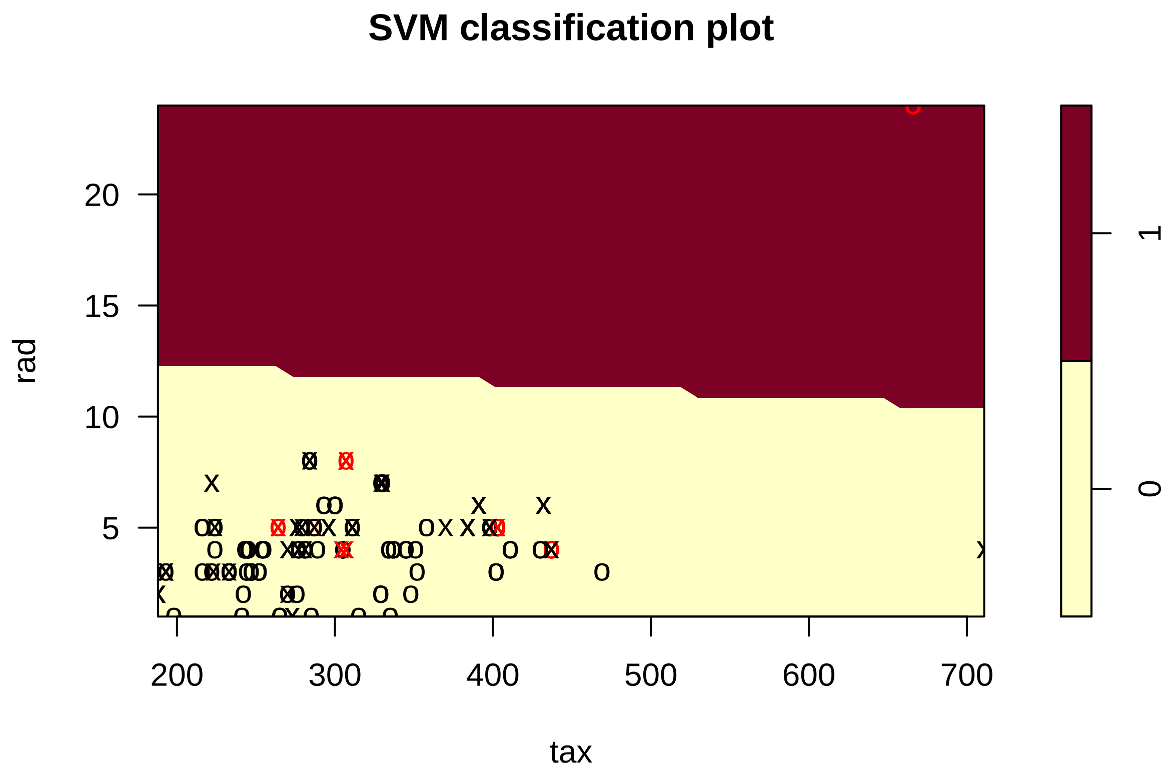
```r
svm_l = svm(as.factor(crim01)~.,data = train_set,kernel = "linear", cost = 0.01,scale = FALSE)

par(mfrow = c(2,2))
plot(svm_l, train_set,indus ~ nox)
```




```r
plot(svm_l, train_set, age ~dis)
```

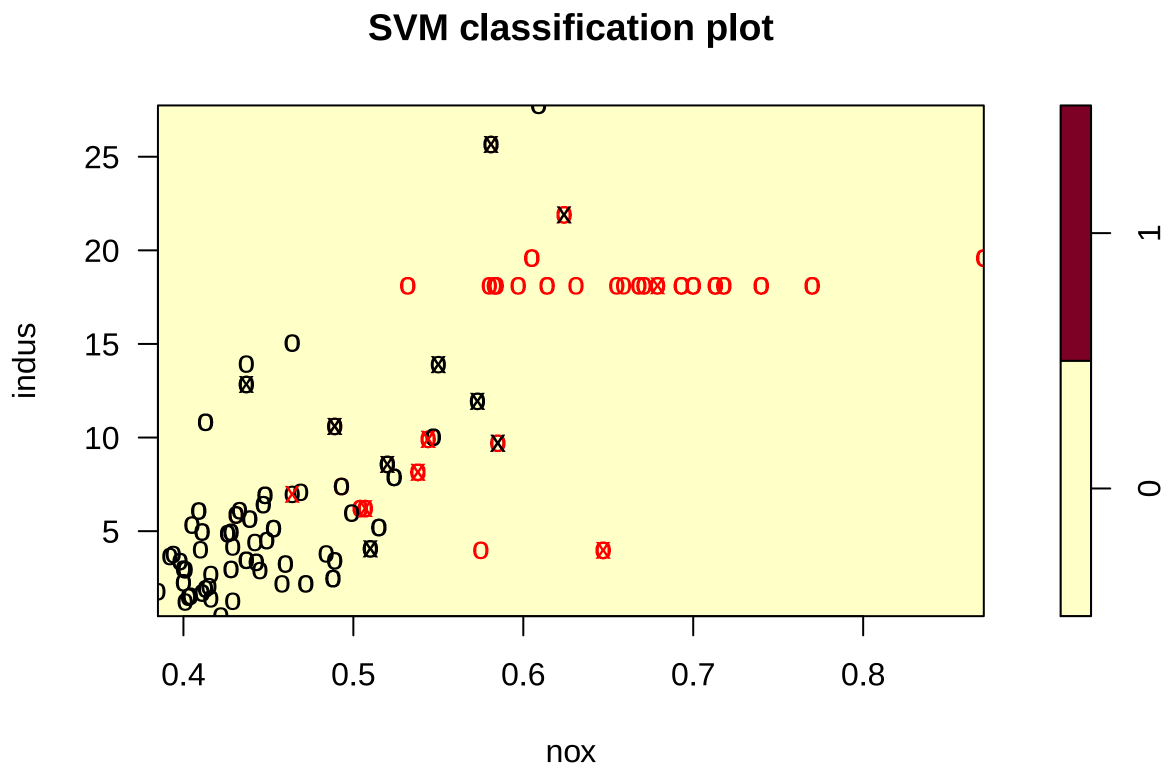



```r
plot(svm_l, train_set, rad ~tax)
```

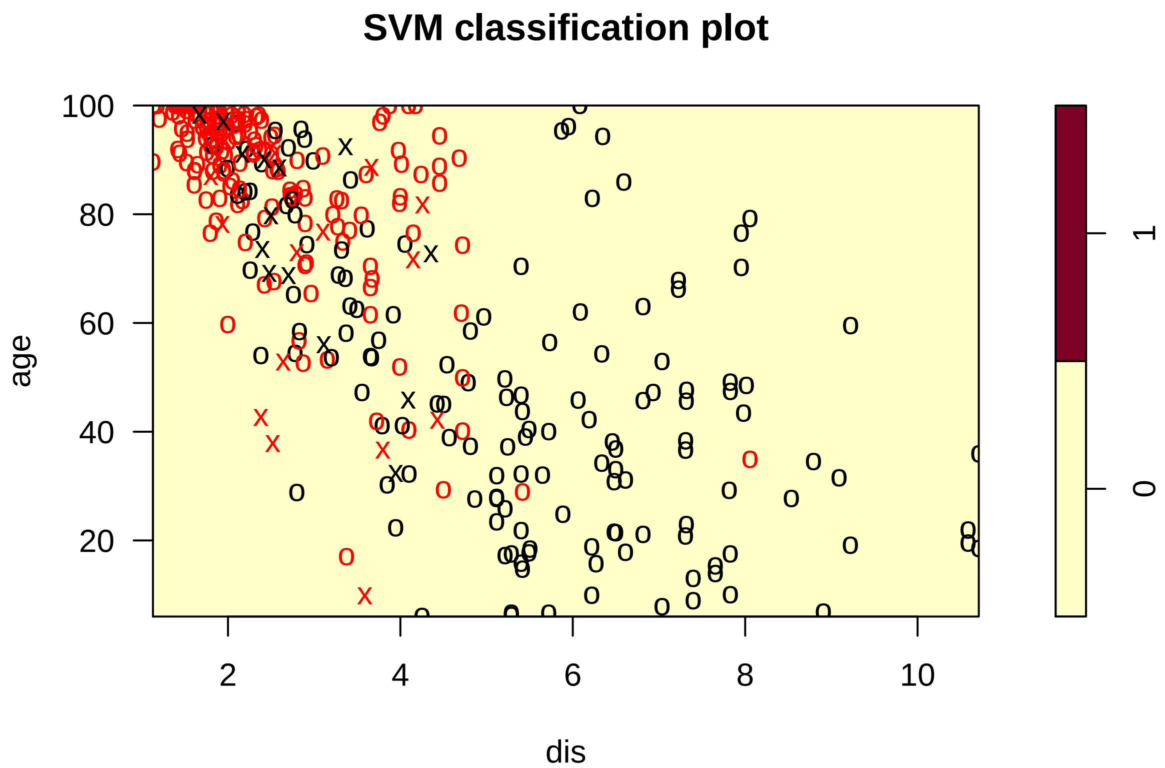


Fit the svm model with kernel = "polynomial"

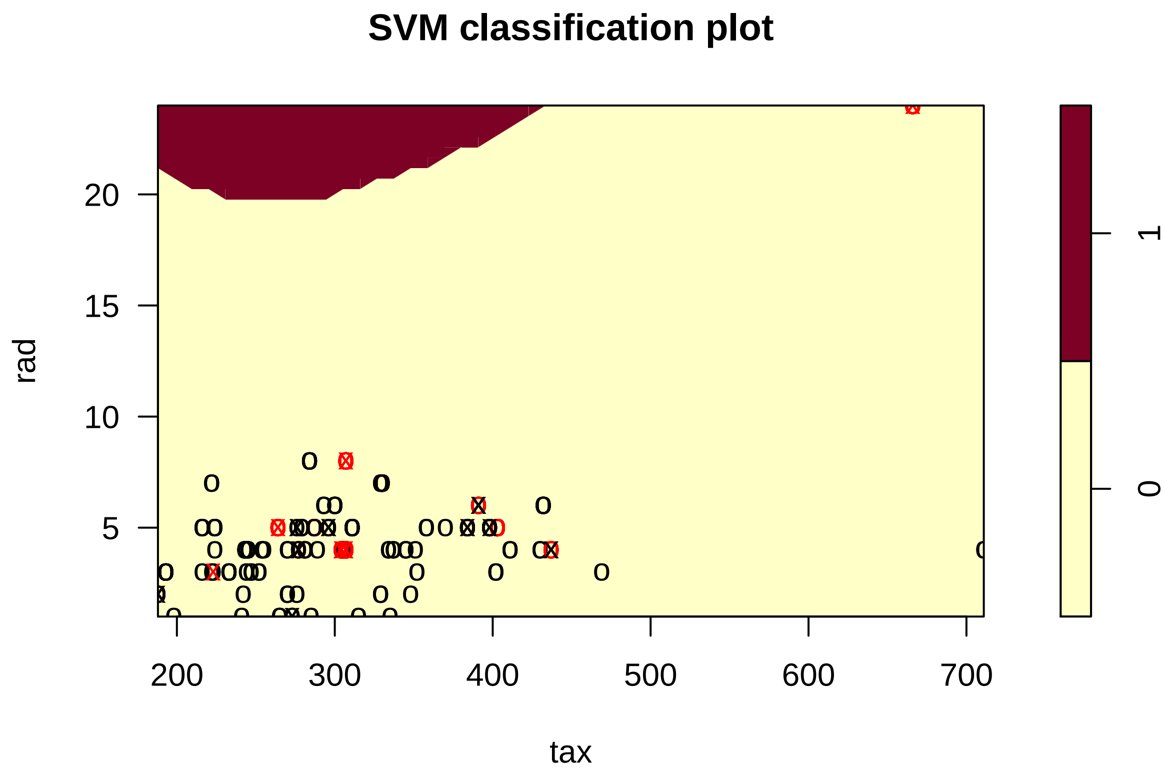
```
svm_p = svm(as.factor(crim01)~.,data = train_set,kernel = "polynomial", cost = 0.1,scale = FALSE)

par(mfrow = c(2,2))
plot(svm_p, train_set,indus ~ nox)
```




```
plot(svm_p, train_set, age ~dis)
```

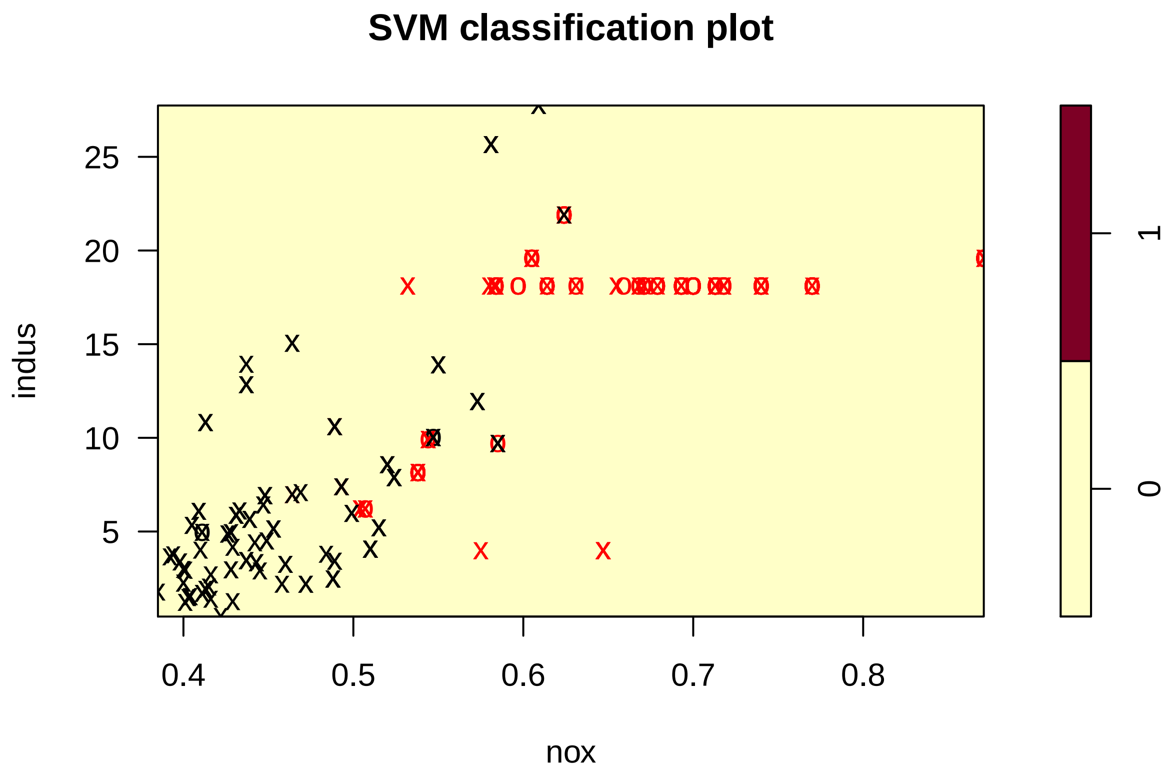



```r
plot(svm_p, train_set, rad ~tax)
```

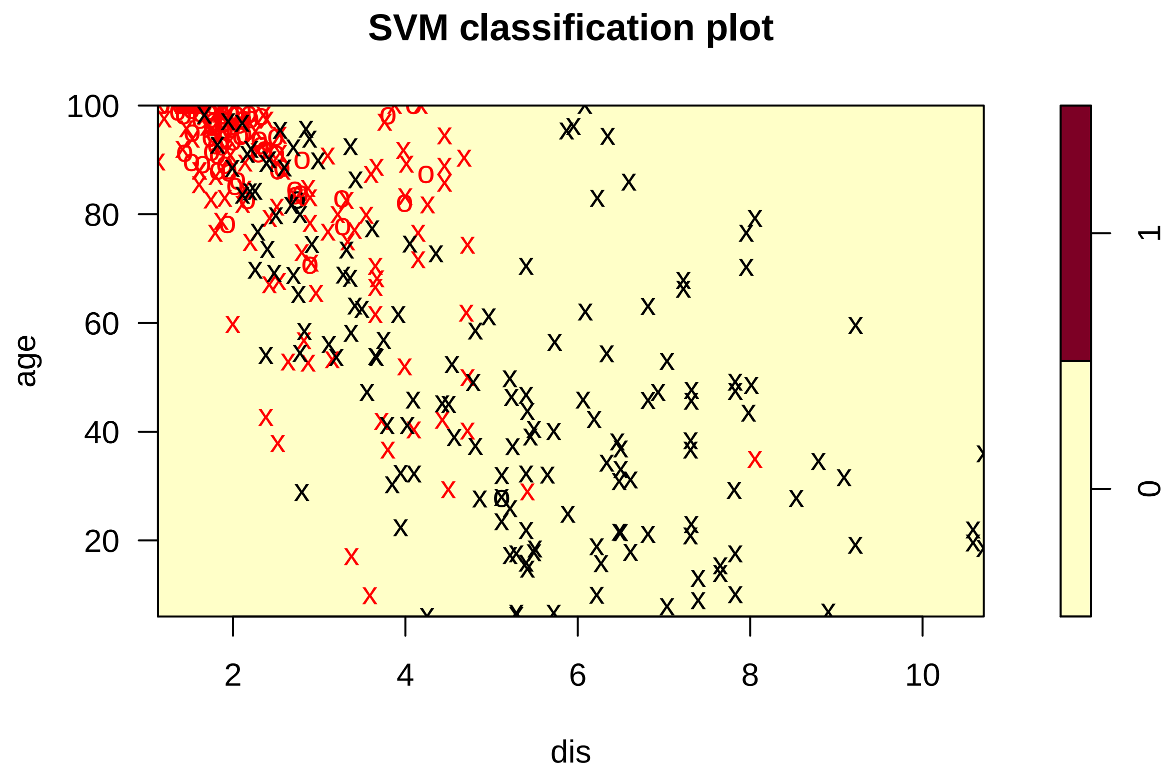


FIt the svm model with kernel = "radial"

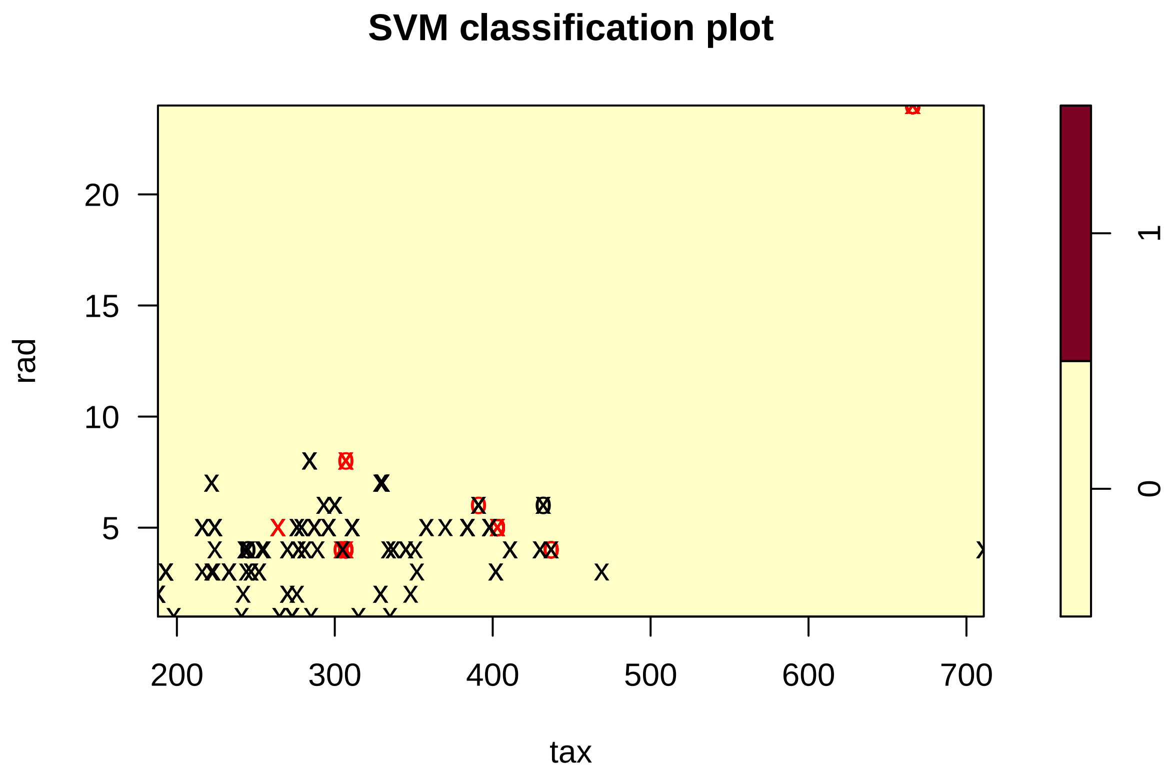
```
svm_r = svm(as.factor(crim01)~.,data = train_set,kernel = "radial", cost = 100,scale = FALSE)

par(mfrow = c(2,2))
plot(svm_r, train_set,indus ~ nox)
```




```
plot(svm_r, train_set, age ~dis)
```




```r
plot(svm_r, train_set, rad ~tax)
```




```
(h) Compare the test errors of LDA, QDA, the best tuned models for KNN, linear SVM, SVM with radial basi

LDA:  1-0.852071 = 0.147929
QDA: 1-0.887574 = 0.112426
KNN: 1-0.9151796 = 0.0848204
linear SVM: 1-0.9053254 = 0.0946746
SVM with  polynomial basis kernel : 1-0.9204545 = 0.0795455
SVM with radial basis kernel : 1-0.9408284  = 0.0591716

The SVM model with radial basis kernel can achieve lowest test error, while LDA can achieve highest tes
```

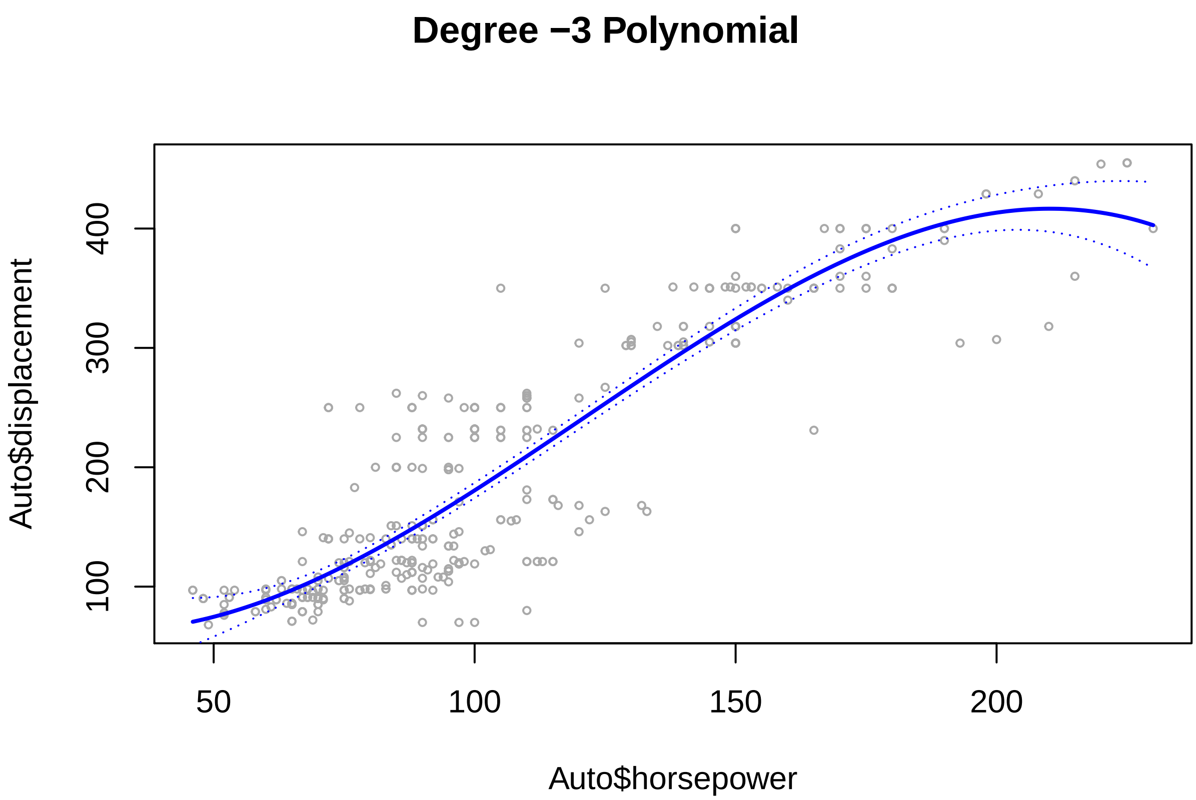
```
 Fit a cubic polynomial regression to predict `displacement` using `horsepower`.

 The model is displacement = 194.4120 + 1856.6053*horsepower -148.3240*horsepower^2 -224.5867*horsepowe
```

```r
poly_fit = lm(displacement~poly(horsepower,3), data = Auto)
summary(poly_fit)
```

```
##
## Call:
## lm(formula = displacement ~ poly(horsepower, 3), data = Auto)
##
```

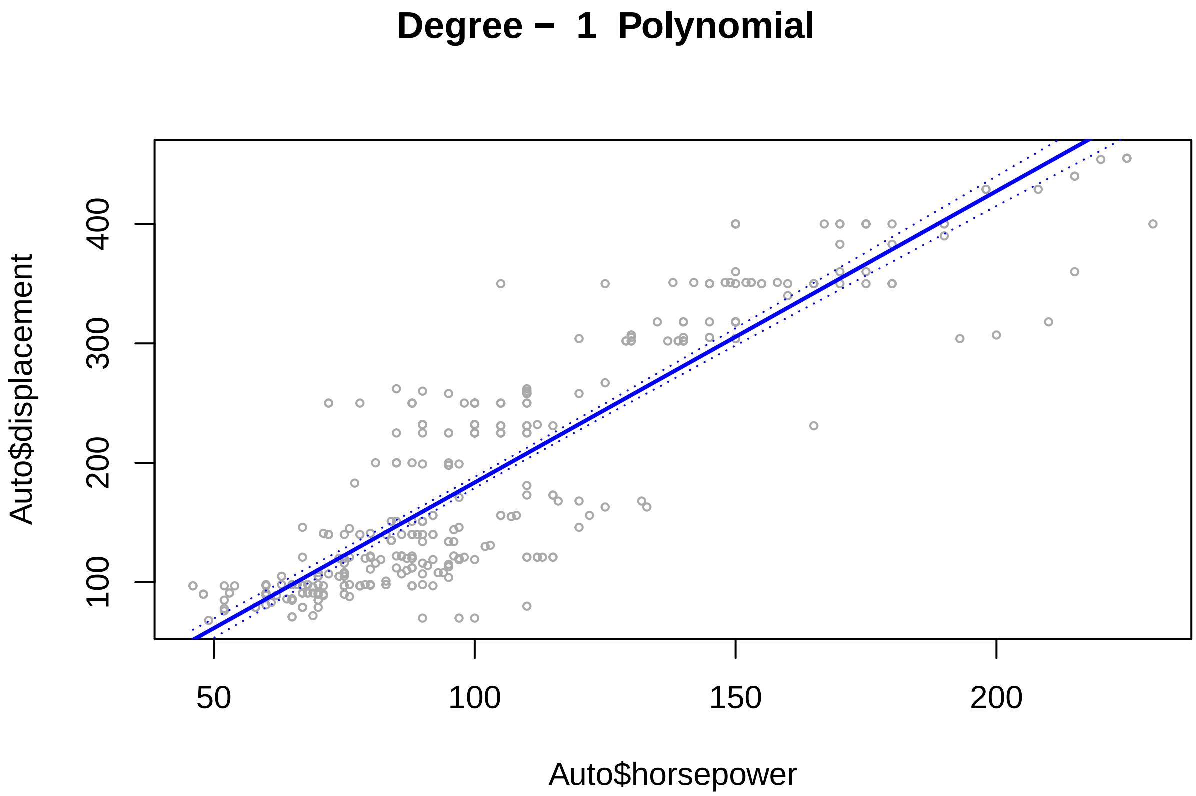
```
## Residuals:
##      Min       1Q   Median       3Q      Max
## -129.656  -21.818   -3.818   26.498  155.207
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)            194.412      2.239  86.847  < 2e-16 ***
## poly(horsepower, 3)1 1856.605     44.321  41.890  < 2e-16 ***
## poly(horsepower, 3)2 -148.324     44.321  -3.347 0.000898 ***
## poly(horsepower, 3)3 -224.587     44.321  -5.067 6.26e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 44.32 on 388 degrees of freedom
## Multiple R-squared:  0.822,  Adjusted R-squared:  0.8206
## F-statistic: 597.2 on 3 and 388 DF,  p-value: < 2.2e-16
```

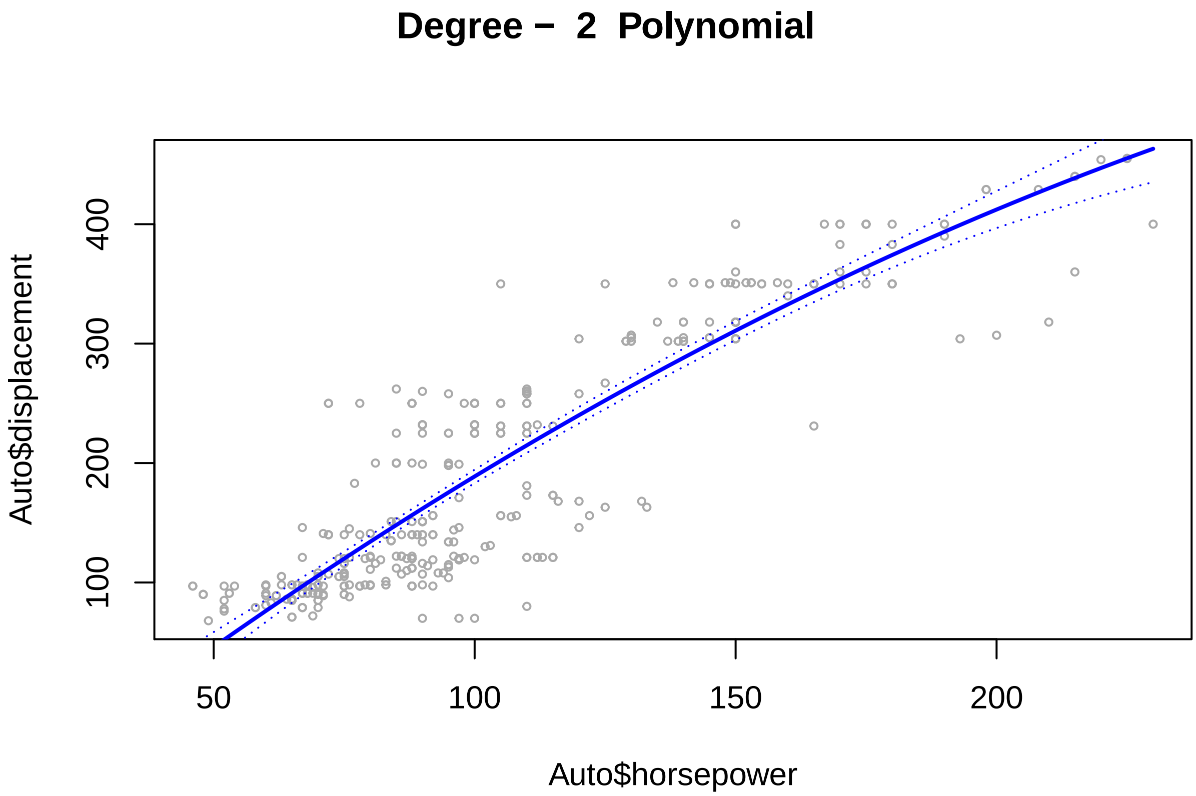
```
horselim = range(Auto$horsepower)
horse.grid = seq(from = horselim[1], to = horselim[2])
pred_poly = predict(poly_fit, newdata = list(horsepower = horse.grid), se = TRUE)
bands =cbind(pred_poly$fit +2* pred_poly$se.fit ,pred_poly$fit -2* pred_poly$se.fit)

par(mar=c(4.5,4.5,1,1) ,oma=c(0,0,4,0))
plot(Auto$horsepower ,Auto$displacement ,xlim= range(Auto$horsepower),cex =.5,col=" darkgrey ")
title(" Degree -3 Polynomial ",outer=T)
lines(horse.grid ,pred_poly$fit ,lwd=2,col="blue")
matlines (horse.grid ,bands ,lwd=1, col=" blue",lty=3)
```



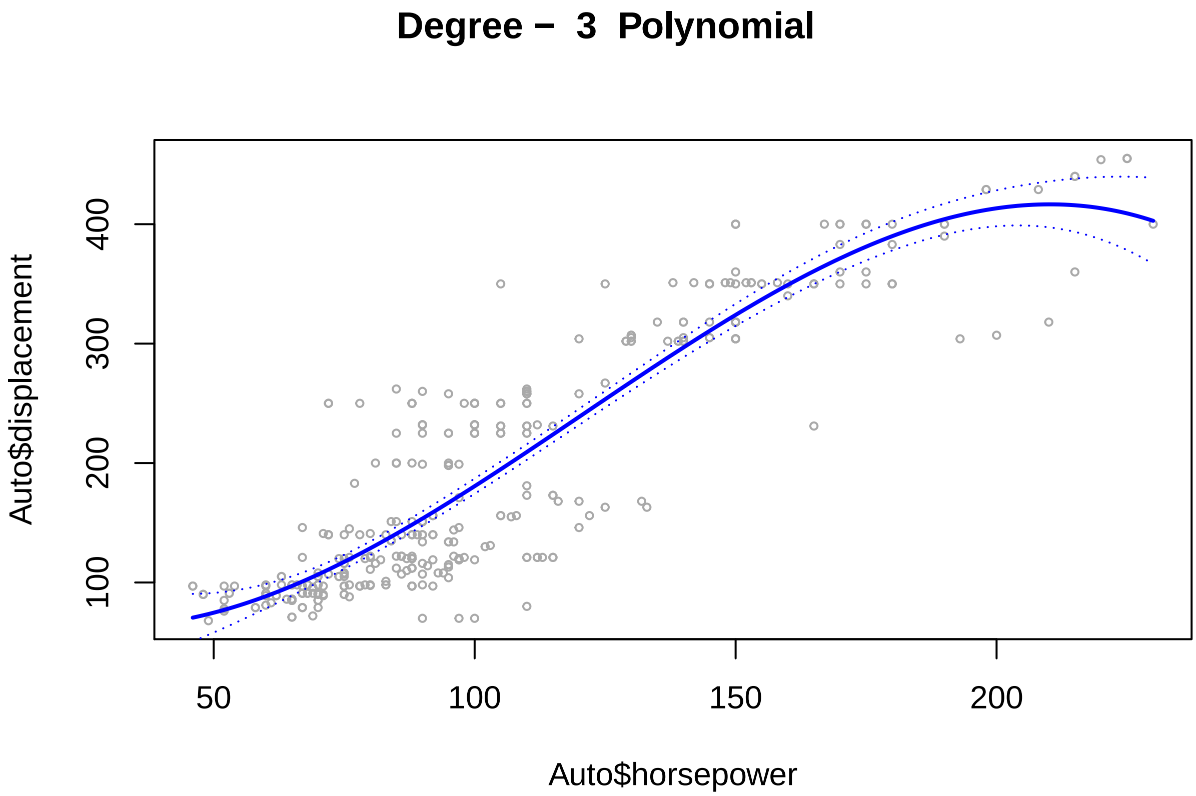
Here are the plots for polynomial fits for a range of different polynomial degrees.

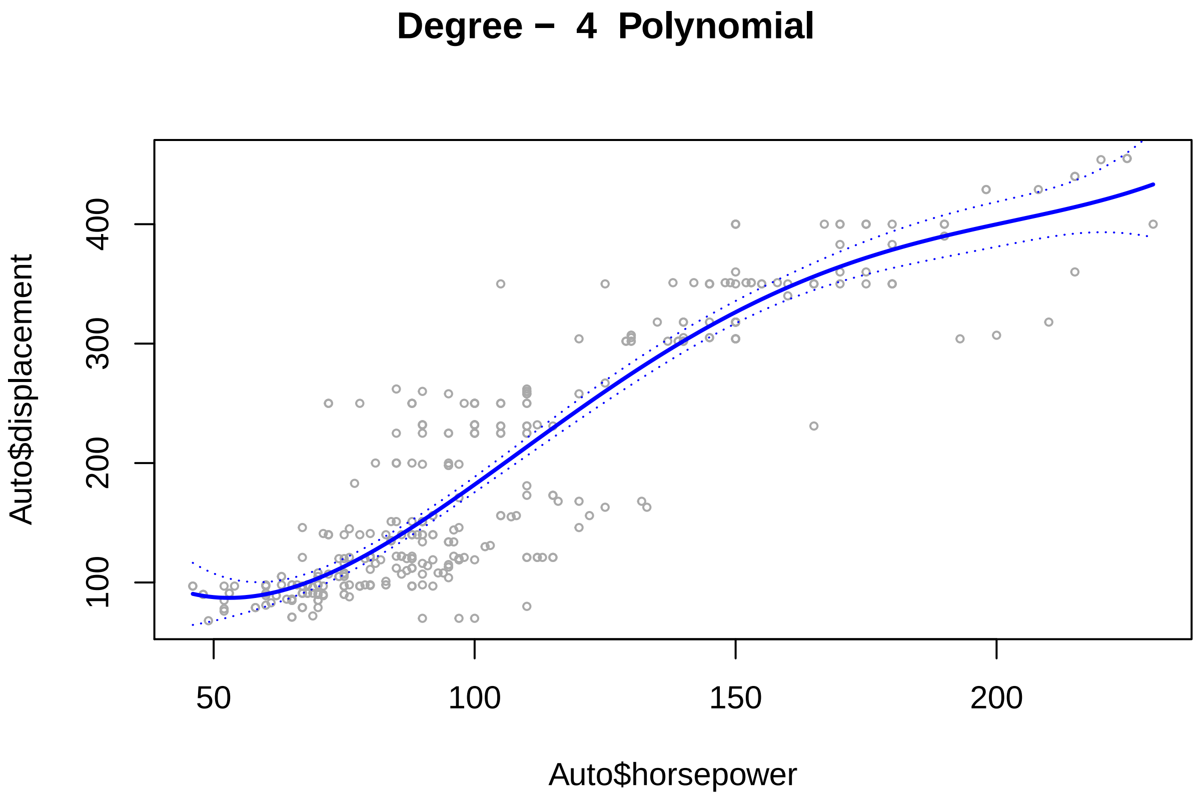
```
vec = c()

for (i in 1:10){

poly_fit = lm(displacement~poly(horsepower,i), data = Auto)


horselim = range(Auto$horsepower)
horse.grid = seq(from = horselim[1], to = horselim[2])
pred_poly = predict(poly_fit, newdata = list(horsepower = horse.grid), se = TRUE)
bands =cbind(pred_poly$fit +2* pred_poly$se.fit ,pred_poly$fit -2* pred_poly$se.fit)

par(mar=c(4.5,4.5,1,1) ,oma=c(0,0,4,0))
plot(Auto$horsepower ,Auto$displacement ,xlim= range(Auto$horsepower),cex =.5,col=" darkgrey ")
title(paste(" Degree - ",i," Polynomial "),outer=T)
lines(horse.grid ,pred_poly$fit ,lwd=2,col="blue")
matlines (horse.grid ,bands ,lwd=1, col=" blue",lty=3)

vec[i] = sum((resid(poly_fit))^2)

}
```


Degree − 1 Polynomial
Auto$displacement
100
200
300
400
50
100
150
200
Auto$horsepower

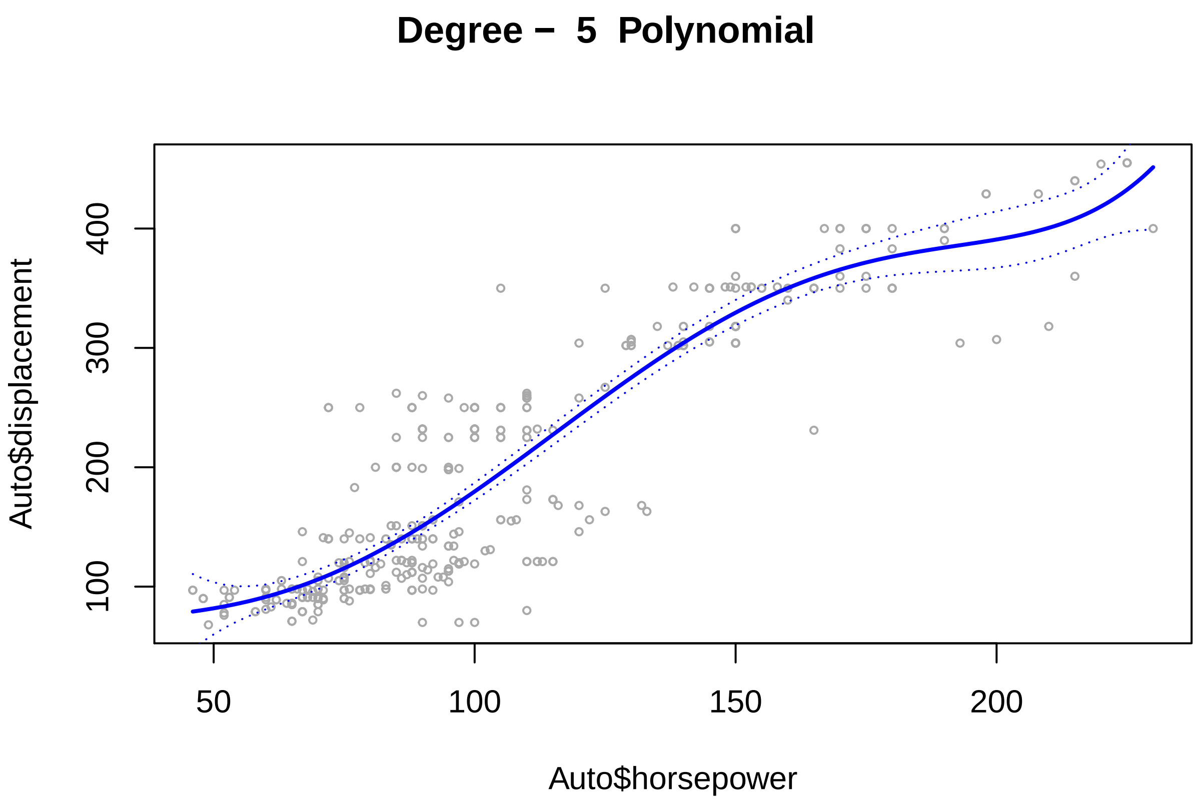


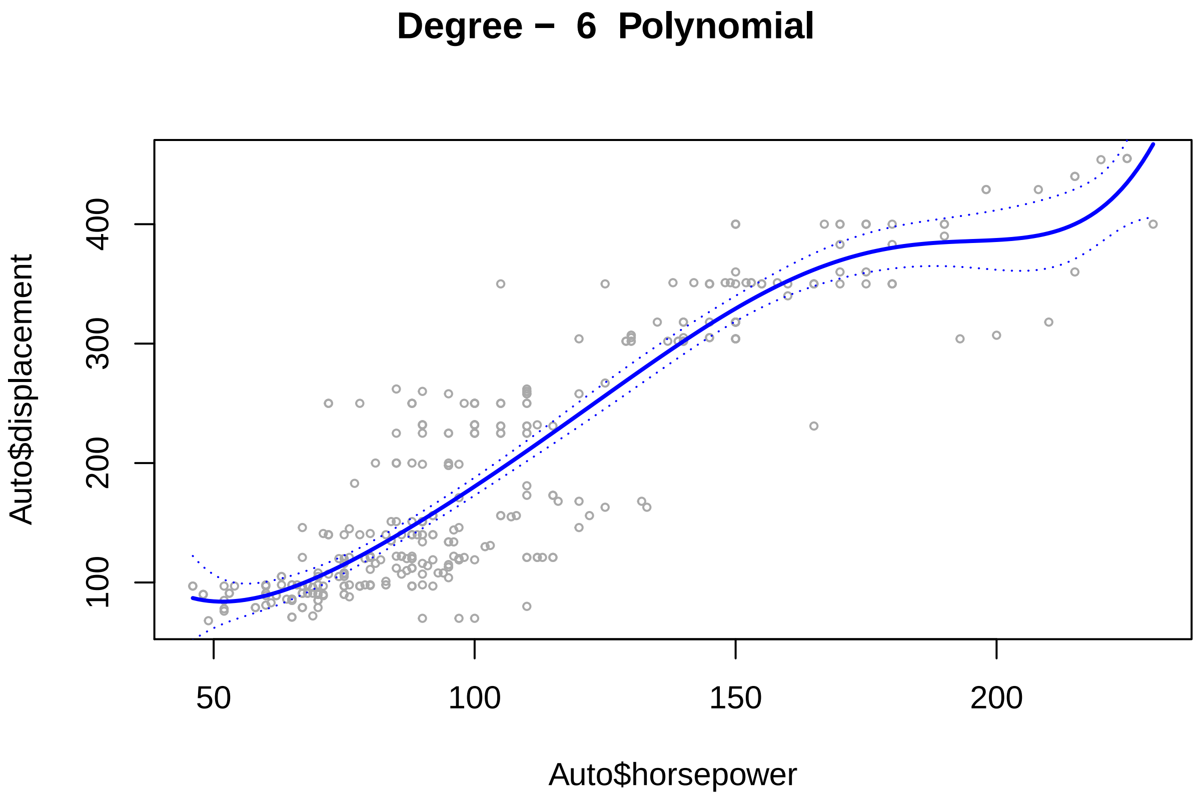
Degree − 2 Polynomial
Auto$displacement
100
200
300
400
50
100
150
200
Auto$horsepower

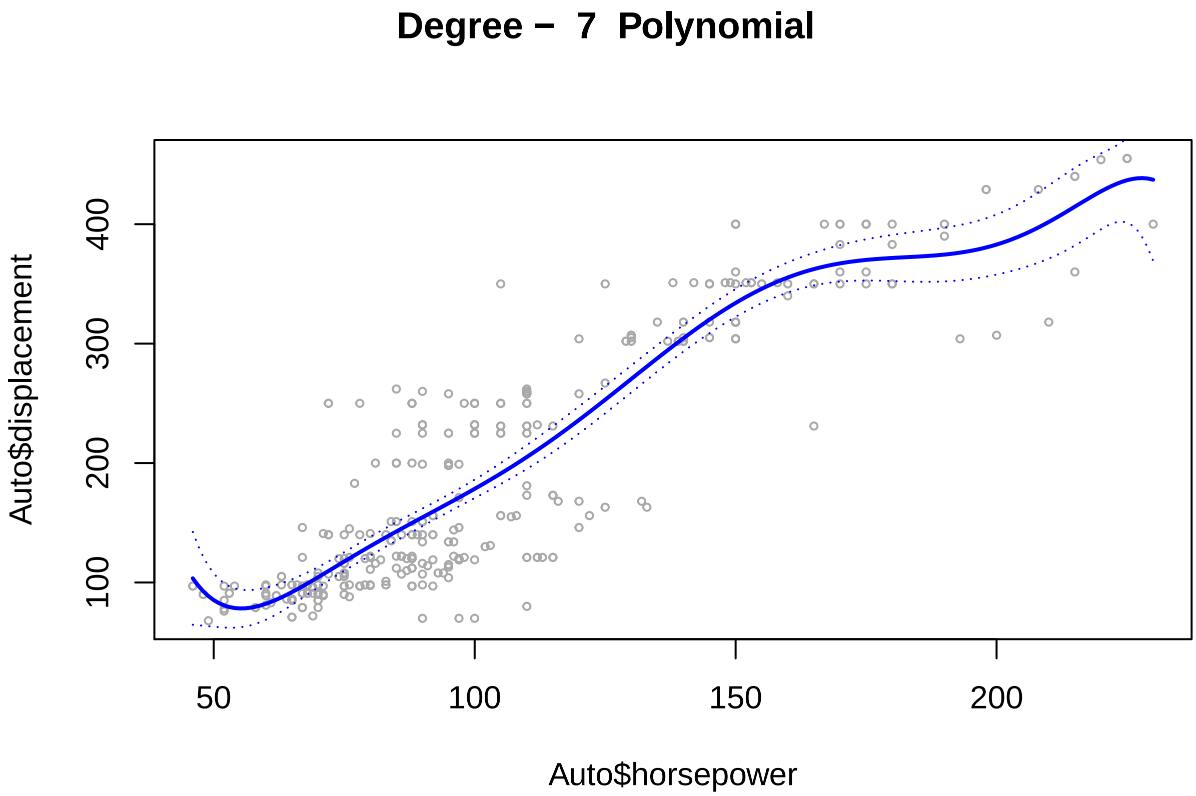

Degree − 3 Polynomial
Auto$displacement
100
200
300
400
50
100
150
200
Auto$horsepower

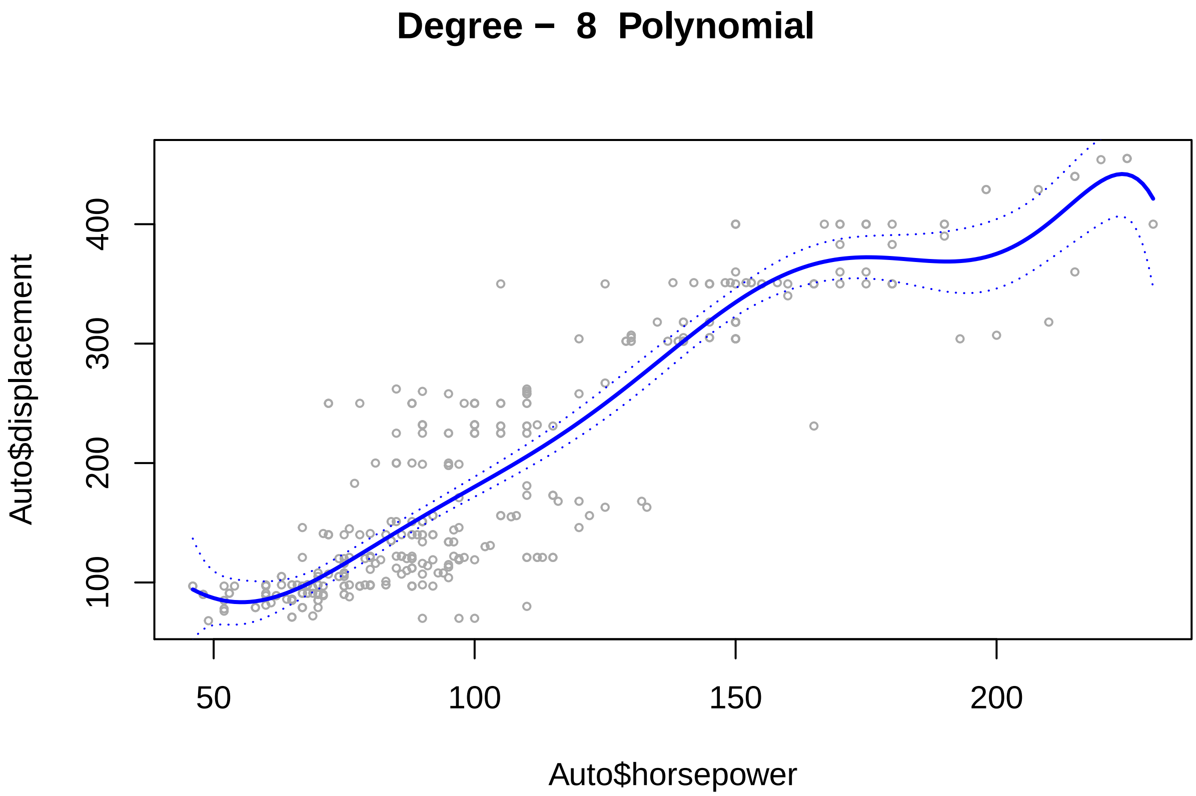


Degree − 4 Polynomial
Auto$displacement
100
200
300
400
50
100
150
200
Auto$horsepower

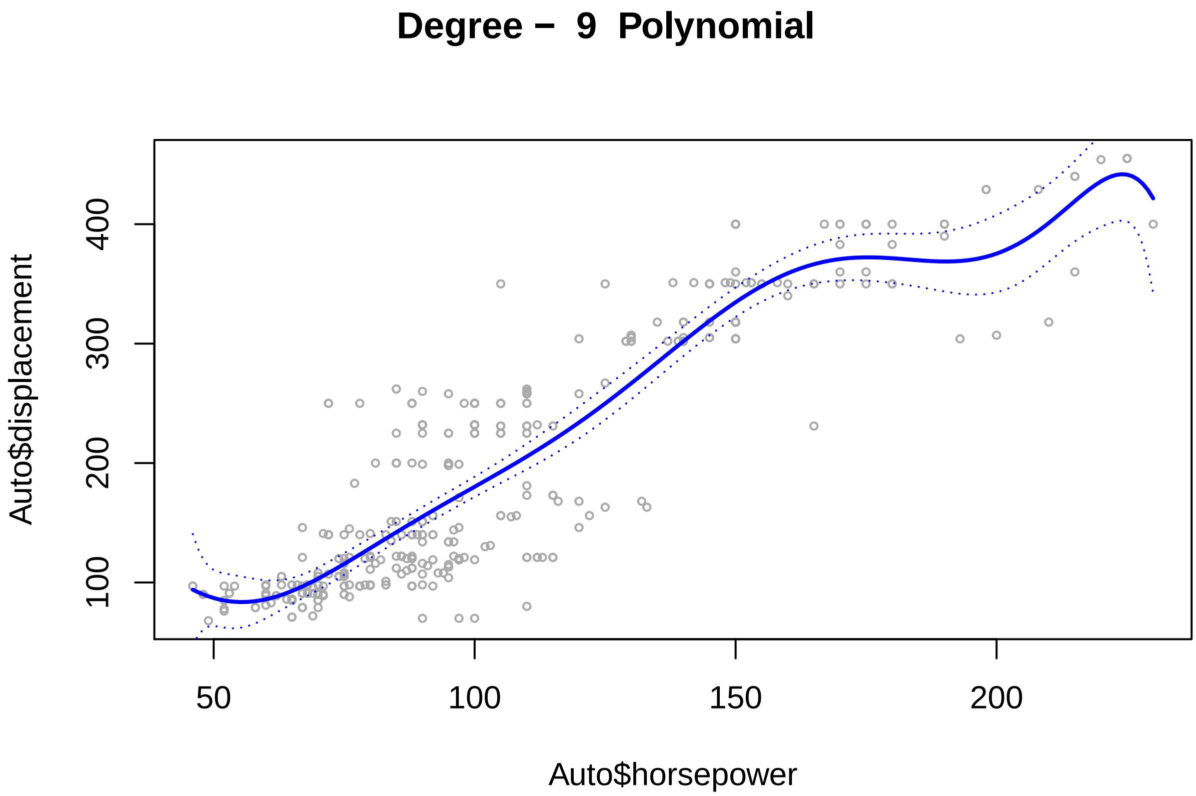

Degree − 5 Polynomial
Auto$displacement
100
200
300
400
50
100
150
200
Auto$horsepower

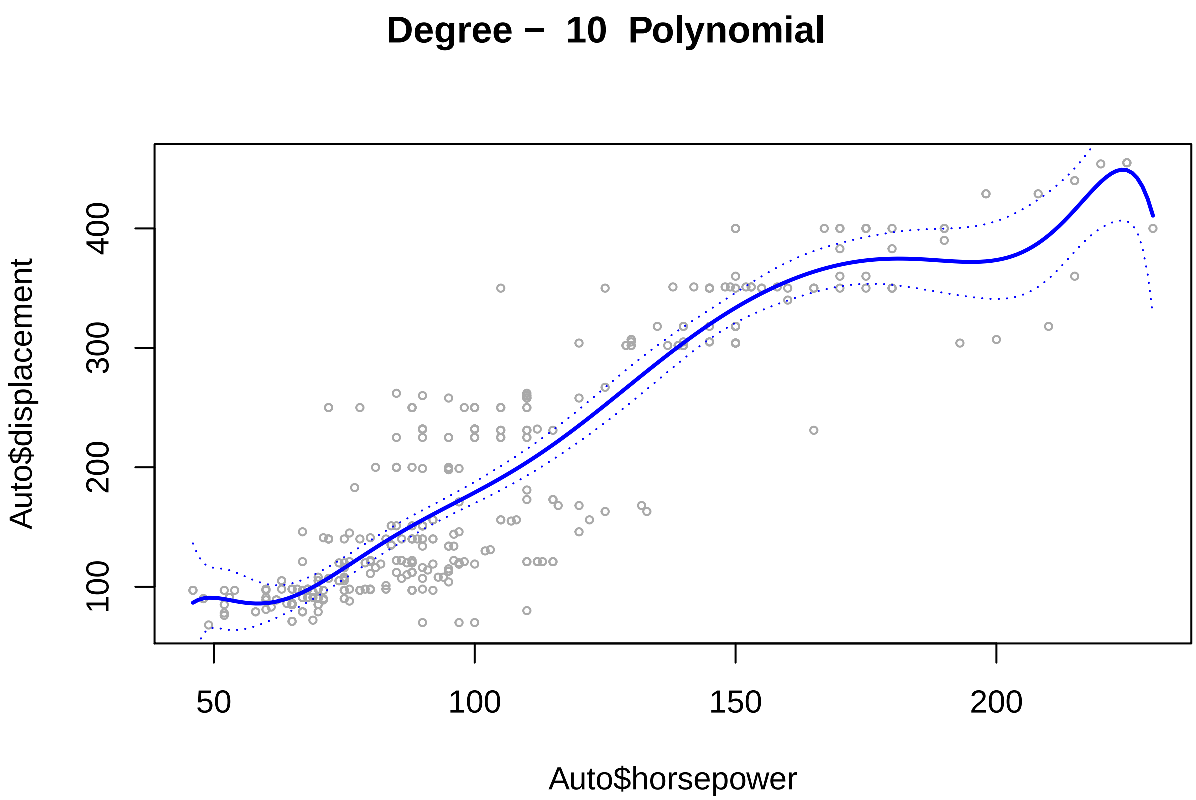

Degree − 6 Polynomial
Auto$displacement
100
200
300
400
50
100
150
200
Auto$horsepower


Degree − 7 Polynomial
Auto$displacement
100
200
300
400
50
100
150
200
Auto$horsepower



Degree − 8 Polynomial
Auto$displacement
100
200
300
400
50
100
150
200
Auto$horsepower


Degree − 9 Polynomial
Auto$displacement
100
200
300
400
50
100
150
200
Auto$horsepower




```
print(vec)
```

```
## [1] 834610.5 812610.5 762171.3 751387.0 748198.0 746313.1 738708.4 736536.6
## [9] 736535.5 735121.6
```

```
Using cross-validation to select the optimal degree for the polynomial.

From the results, when degree for polynomial is 6, the model can achieve lowest sum of square error, whi
```

```
When knot is 9, the sum of square error is smallest, which is 53371.92.
```

```
ssr_bs = c()
for (i in 1:10){

  set.seed(12)
  testIdx = which(folds == i, arr.ind = TRUE)
  testData_bs = Auto[testIdx,]
  trainData_bs = Auto[-testIdx,]

  fit_bs = lm(displacement ~ bs(horsepower, df = 4, knots = c(i),degree = 3, intercept = FALSE), da
```

```r
    predd_bs =  predict(fit_bs, newdata = testData_bs, se = T)

    ssr_bs[i] = sum((predd_bs$fit-testData_bs$displacement)^2)
  }
```

```
## Warning in predict.lm(fit_bs, newdata = testData_bs, se = T): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(fit_bs, newdata = testData_bs, se = T): prediction from a
## rank-deficient fit may be misleading

## Warning in bs(horsepower, degree = 3L, knots = 3L, Boundary.knots = c(46, : some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in predict.lm(fit_bs, newdata = testData_bs, se = T): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(fit_bs, newdata = testData_bs, se = T): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(fit_bs, newdata = testData_bs, se = T): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(fit_bs, newdata = testData_bs, se = T): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(fit_bs, newdata = testData_bs, se = T): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(fit_bs, newdata = testData_bs, se = T): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(fit_bs, newdata = testData_bs, se = T): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(fit_bs, newdata = testData_bs, se = T): prediction from a
## rank-deficient fit may be misleading
```

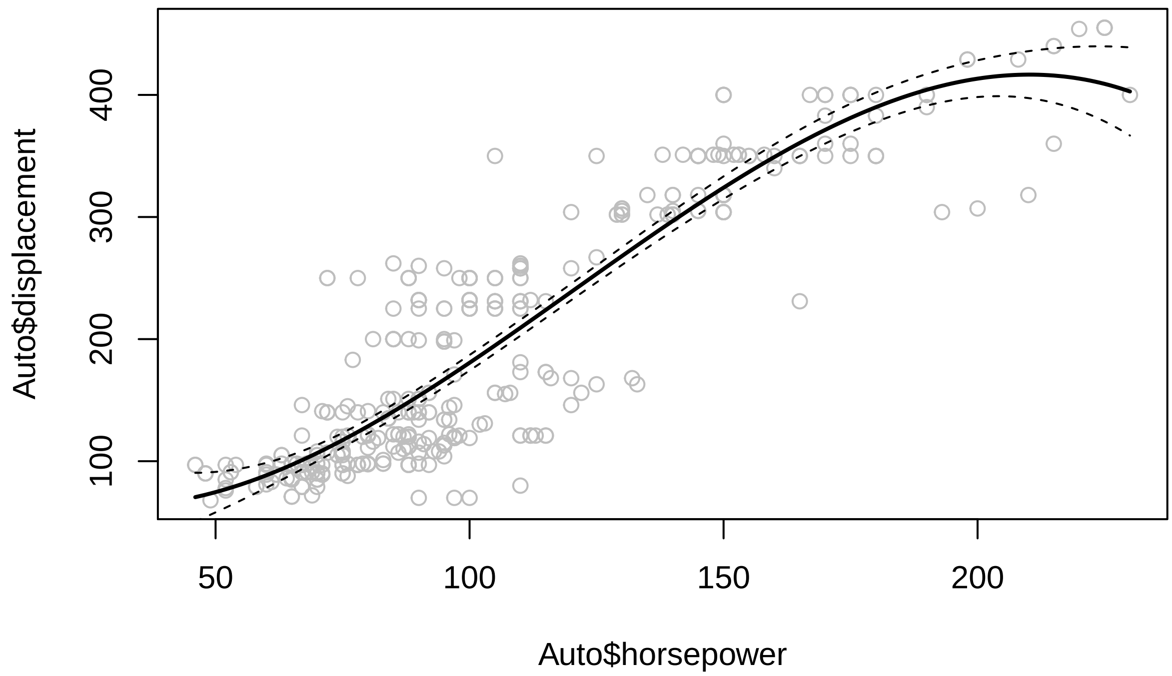
```r
  ssr_bs
```

```
##  [1]  74774.23  84923.59  64515.42  69066.66 105084.51  56636.71  89553.52
##  [8]  80334.43  48581.00 100365.81
```

```r
  fit_B = lm(displacement ~ bs(horsepower, df = 4, knots = 9,degree = 3, intercept = FALSE), data = Au
  pred_B = predict(fit_B, newdata = list(horsepower = horse.grid), se = T)
```

```
## Warning in predict.lm(fit_B, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading
```

```
plot(Auto$horsepower,Auto$displacement, col = "grey")
lines(horse.grid,pred_B$fit,lwd=2)
lines(horse.grid,pred_B$fit+2*pred_B$se,lty="dashed")
lines(horse.grid,pred_B$fit-2*pred_B$se,lty="dashed")
```

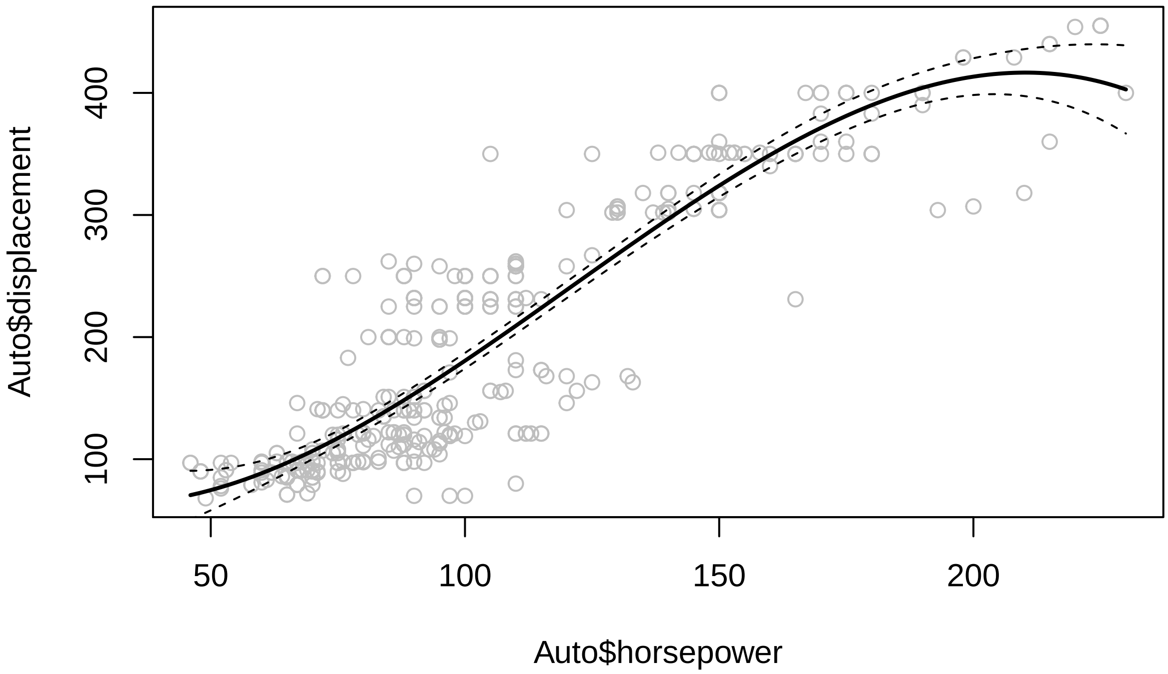


Fit a cubic B-Spline for a range of degrees of freedom from 1 to 10.

```
df_ssr = c()
for (i in 1:10){
fit_B1 = lm(displacement ~ bs(horsepower, df = i, knots = 1,degree = 3, intercept = FALSE), data = 
pred_B1 = predict(fit_B1, newdata = list(horsepower = horse.grid), se = T)
plot(Auto$horsepower,Auto$displacement, col = "grey")
lines(horse.grid,pred_B1$fit,lwd=2)
lines(horse.grid,pred_B1$fit+2*pred_B1$se,lty="dashed")
lines(horse.grid,pred_B1$fit-2*pred_B1$se,lty="dashed")

df_ssr[i] = sum(resid(fit_B1)^2)
}
```

```
## Warning in predict.lm(fit_B1, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(fit_B1, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading
```

```
## Warning in predict.lm(fit_B1, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading
```




```
## Warning in predict.lm(fit_B1, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(fit_B1, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(fit_B1, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(fit_B1, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(fit_B1, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(fit_B1, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(fit_B1, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading
```

```r
df_ssr
```

```
##  [1] 762171.3 762171.3 762171.3 762171.3 762171.3 762171.3 762171.3 762171.3
##  [9] 762171.3 762171.3
```

Perform cross-validation to select the best degrees of freedom for a cubic B-Spline.

When degree of freedom equals to 9, the sum of square error is smallest, which is 53371.92.

```r
ssr_bs1 = c()
for (i in 1:10){

  set.seed(11)
  testIdx = which(folds == i, arr.ind = TRUE)
  testData_bs1 = Auto[testIdx,]
  trainData_bs1 = Auto[-testIdx,]

  fit_bs1 = lm(displacement ~ bs(horsepower, df = i, knots = 5,degree = 3, intercept = FALSE), data


  predd_bs1 =  predict(fit_bs1, newdata = testData_bs1, se = T)

  ssr_bs[i] = sum((predd_bs1$fit-testData_bs1$displacement)^2)
}
```

```
## Warning in predict.lm(fit_bs1, newdata = testData_bs1, se = T): prediction from
## a rank-deficient fit may be misleading

## Warning in predict.lm(fit_bs1, newdata = testData_bs1, se = T): prediction from
## a rank-deficient fit may be misleading

## Warning in bs(horsepower, degree = 3L, knots = 5, Boundary.knots = c(46, : some
## 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in predict.lm(fit_bs1, newdata = testData_bs1, se = T): prediction from
## a rank-deficient fit may be misleading

## Warning in predict.lm(fit_bs1, newdata = testData_bs1, se = T): prediction from
## a rank-deficient fit may be misleading

## Warning in predict.lm(fit_bs1, newdata = testData_bs1, se = T): prediction from
## a rank-deficient fit may be misleading

## Warning in predict.lm(fit_bs1, newdata = testData_bs1, se = T): prediction from
## a rank-deficient fit may be misleading

## Warning in predict.lm(fit_bs1, newdata = testData_bs1, se = T): prediction from
## a rank-deficient fit may be misleading

## Warning in predict.lm(fit_bs1, newdata = testData_bs1, se = T): prediction from
## a rank-deficient fit may be misleading
```

```
## Warning in predict.lm(fit_bs1, newdata = testData_bs1, se = T): prediction from
## a rank-deficient fit may be misleading

## Warning in predict.lm(fit_bs1, newdata = testData_bs1, se = T): prediction from
## a rank-deficient fit may be misleading
```

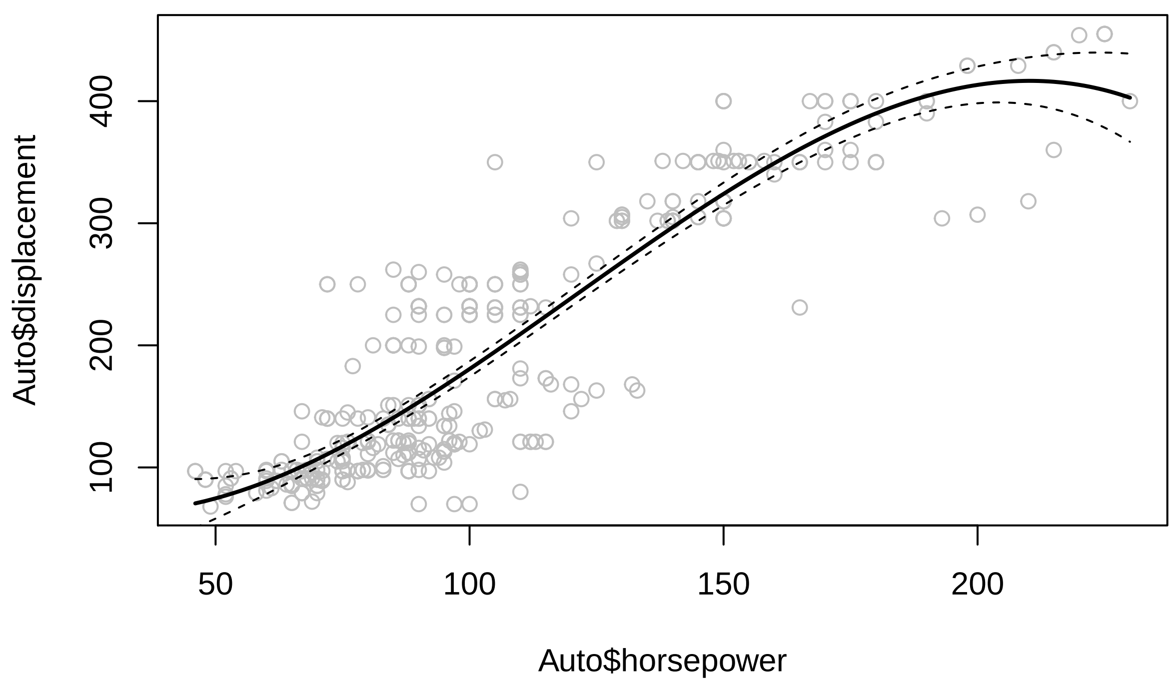
```
ssr_bs
```

```
##  [1]  74774.23  84923.59  64515.42  69066.66 105084.51  56636.71  89553.52
##  [8]  80334.43  48581.00 100365.81
```

```
fit_B1 = lm(displacement ~ bs(horsepower, df = 5, knots = 5,degree = 3, intercept = FALSE), data = A
pred_B1 = predict(fit_B1, newdata = list(horsepower = horse.grid), se = T)
```

```
## Warning in predict.lm(fit_B1, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading
```

```
plot(Auto$horsepower,Auto$displacement, col = "grey")
lines(horse.grid,pred_B1$fit,lwd=2)
lines(horse.grid,pred_B1$fit+2*pred_B1$se,lty="dashed")
lines(horse.grid,pred_B1$fit-2*pred_B1$se,lty="dashed")
```

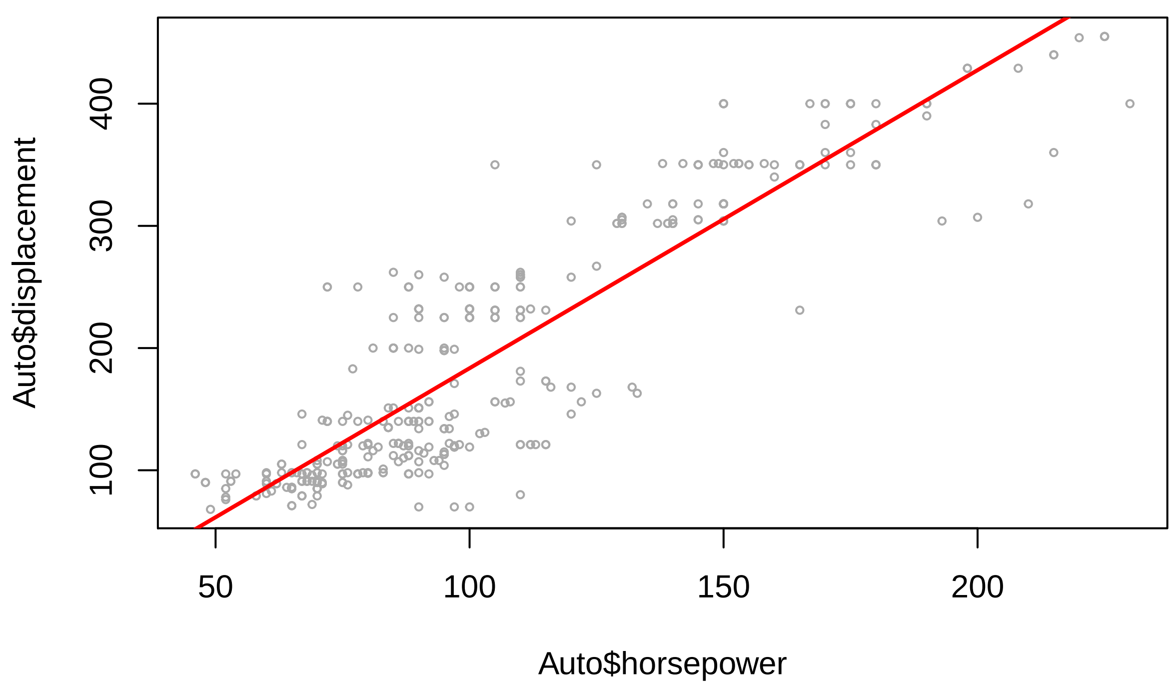



```
Use the `ns()` function to fit a natural cubic spline.

The sum of squared error is 834660.8.
```

```r
fit_ns=lm(displacement~ns(horsepower,df=4,knots = 3),data = Auto)
pred_ns=predict(fit_ns,newdata=list(horsepower=horse.grid),se=T)
```

```
## Warning in predict.lm(fit_ns, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading
```

```r
 plot(Auto$horsepower,Auto$displacement,xlim=horselim,cex=.5,col="darkgrey")
lines(horse.grid, pred_ns$fit,col="red",lwd=2)
```




```r
sum((resid(fit_ns))^2)
```

```
## [1] 834610.5
```

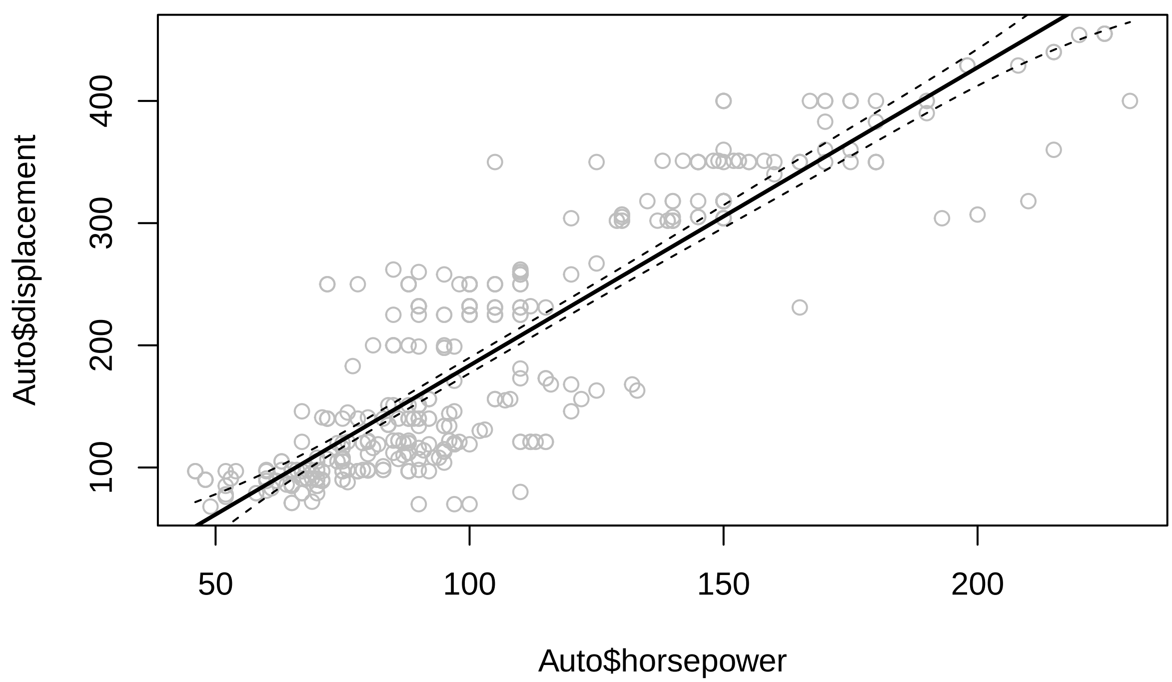
Fit a natural cubic spline for a range of degrees of freedom from 1 to 10.

The resulting RSS are the same, 834660.8.

```r
  rss = c()
  for (i in 1:10){
  fitNS = lm(displacement ~ ns(horsepower, df = i, knots = 1, intercept = FALSE), data = Auto)
  predNS = predict(fitNS, newdata = list(horsepower = horse.grid), se = T)
  plot(Auto$horsepower,Auto$displacement, col = "grey")
  lines(horse.grid,predNS$fit,lwd=2)
```

```r
    lines(horse.grid,predNS$fit+2*pred_B1$se,lty="dashed")
    lines(horse.grid,predNS$fit-2*pred_B1$se,lty="dashed")

    rss[i] = sum(resid(fitNS)^2)
    }
```

```
## Warning in predict.lm(fitNS, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(fitNS, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(fitNS, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading
```




```
## Warning in predict.lm(fitNS, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(fitNS, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(fitNS, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading
```

```
## Warning in predict.lm(fitNS, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(fitNS, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(fitNS, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(fitNS, newdata = list(horsepower = horse.grid), se = T):
## prediction from a rank-deficient fit may be misleading
```

```r
rss
```

```
##  [1] 834610.5 834610.5 834610.5 834610.5 834610.5 834610.5 834610.5 834610.5
##  [9] 834610.5 834610.5
```

Perform cross-validation to select the best degrees of freedom for a natural cubic spline.

When degree of freedom equals to 5, the sum of squared error is smallest, which is 64785.82.

```r
ssr_nss = c()
for (i in 1:10){

  set.seed(11)
  testIdx = which(folds == i, arr.ind = TRUE)
  testData_nss = Auto[testIdx,]
  trainData_nss = Auto[-testIdx,]

  fit_nss = lm(displacement ~ ns(horsepower, df = i, knots =1, intercept = FALSE), data = trainData_


  predd_nss =  predict(fit_nss, newdata = testData_nss, se = T)

  ssr_nss[i] = sum((predd_nss$fit-testData_nss$displacement)^2)
}
```

```
## Warning in predict.lm(fit_nss, newdata = testData_nss, se = T): prediction from
## a rank-deficient fit may be misleading

## Warning in predict.lm(fit_nss, newdata = testData_nss, se = T): prediction from
## a rank-deficient fit may be misleading

## Warning in predict.lm(fit_nss, newdata = testData_nss, se = T): prediction from
## a rank-deficient fit may be misleading

## Warning in predict.lm(fit_nss, newdata = testData_nss, se = T): prediction from
## a rank-deficient fit may be misleading

## Warning in predict.lm(fit_nss, newdata = testData_nss, se = T): prediction from
## a rank-deficient fit may be misleading

## Warning in predict.lm(fit_nss, newdata = testData_nss, se = T): prediction from
```

```
## a rank-deficient fit may be misleading

## Warning in predict.lm(fit_nss, newdata = testData_nss, se = T): prediction from
## a rank-deficient fit may be misleading

## Warning in predict.lm(fit_nss, newdata = testData_nss, se = T): prediction from
## a rank-deficient fit may be misleading

## Warning in predict.lm(fit_nss, newdata = testData_nss, se = T): prediction from
## a rank-deficient fit may be misleading

## Warning in predict.lm(fit_nss, newdata = testData_nss, se = T): prediction from
## a rank-deficient fit may be misleading
```

```
ssr_nss
```

```
##  [1]  85704.73  85582.74  84389.01  72397.32 104210.76  62677.81  99188.55
##  [8]  80063.60  59610.95 105655.39
```
