Project 2

Simon Kauppinen

- Chapter 7.9 Problem 9
- Chapter 8 Problem 11
- Chapter 9 Problem 7
- Chapter 12 Problem 10

Chapter 7.9 Problem 9

```
library("latex2exp")
library("ISLR2")
library("boot")
library("glue")
library("tinytex")
library("polynom")
library("MASS")
library("splines")
```

a)

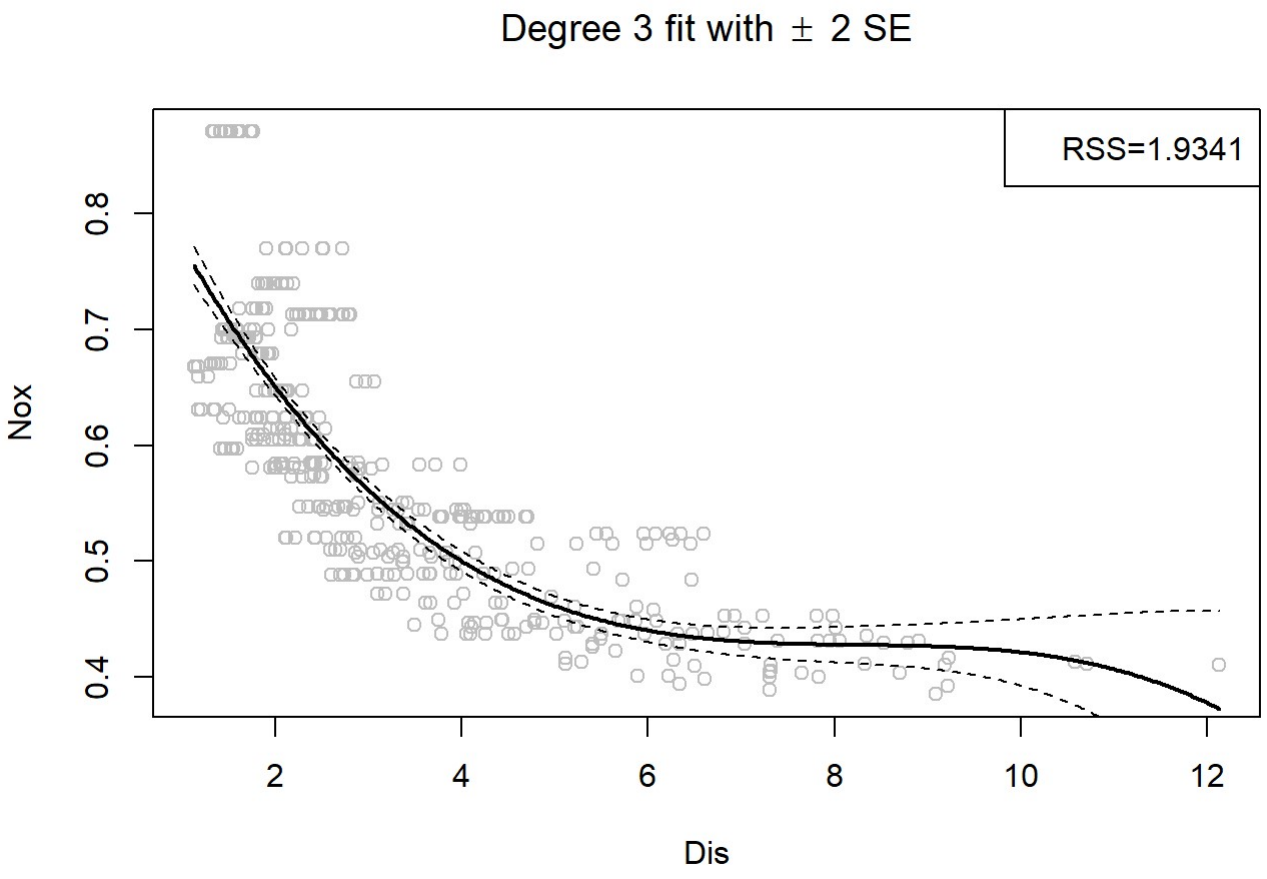
```
#Uses a degree d polynomial to predict nox with dis.
poly.fit <- function(d){
  lm(nox~poly(dis,d), data=Boston)
}

# A set of points along values of dis to be used later for graphing the image of fitted polynomials.
x <- with(Boston, seq(min(dis), max(dis), length.out=400))


#Plots the fitted polynomial with degree d together with the associated RSS.
poly.plot <- function(d){
  pred <- predict(poly.fit(d),newdata=data.frame(dis=x),se.fit=TRUE)
  rss <- round(sum(poly.fit(d)$residuals^2),4)
  plot(nox~dis,data=Boston, main=TeX(sprintf(r'(Degree $%d$ fit with $\pm$ 2 SE)',d)),xlab='Dis',ylab='N
ox',col='grey')
  lines(x, pred$fit, lwd=2)
  lines(x, pred$fit + 2*pred$se.fit, lty="dashed")
  lines(x, pred$fit - 2*pred$se.fit, lty="dashed")
  legend("topright",legend=glue('RSS=',rss))
}

#Plots summary of the fitted cubic polynomial
summary(poly.fit(3))
```

```
##
## Call:
## lm(formula = nox ~ poly(dis, d), data = Boston)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.121130 -0.040619 -0.009738  0.023385  0.194904
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.554695   0.002759 201.021  < 2e-16 ***
## poly(dis, d)1 -2.003096   0.062071 -32.271  < 2e-16 ***
## poly(dis, d)2  0.856330   0.062071  13.796  < 2e-16 ***
## poly(dis, d)3 -0.318049   0.062071  -5.124 4.27e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06207 on 502 degrees of freedom
## Multiple R-squared:  0.7148, Adjusted R-squared:  0.7131
## F-statistic: 419.3 on 3 and 502 DF,  p-value: < 2.2e-16
```

In the summary we can see that the p-value for the polynomial regression and each coefficient is very small (p<0.0001). The coefficients of the cubic polynomial are

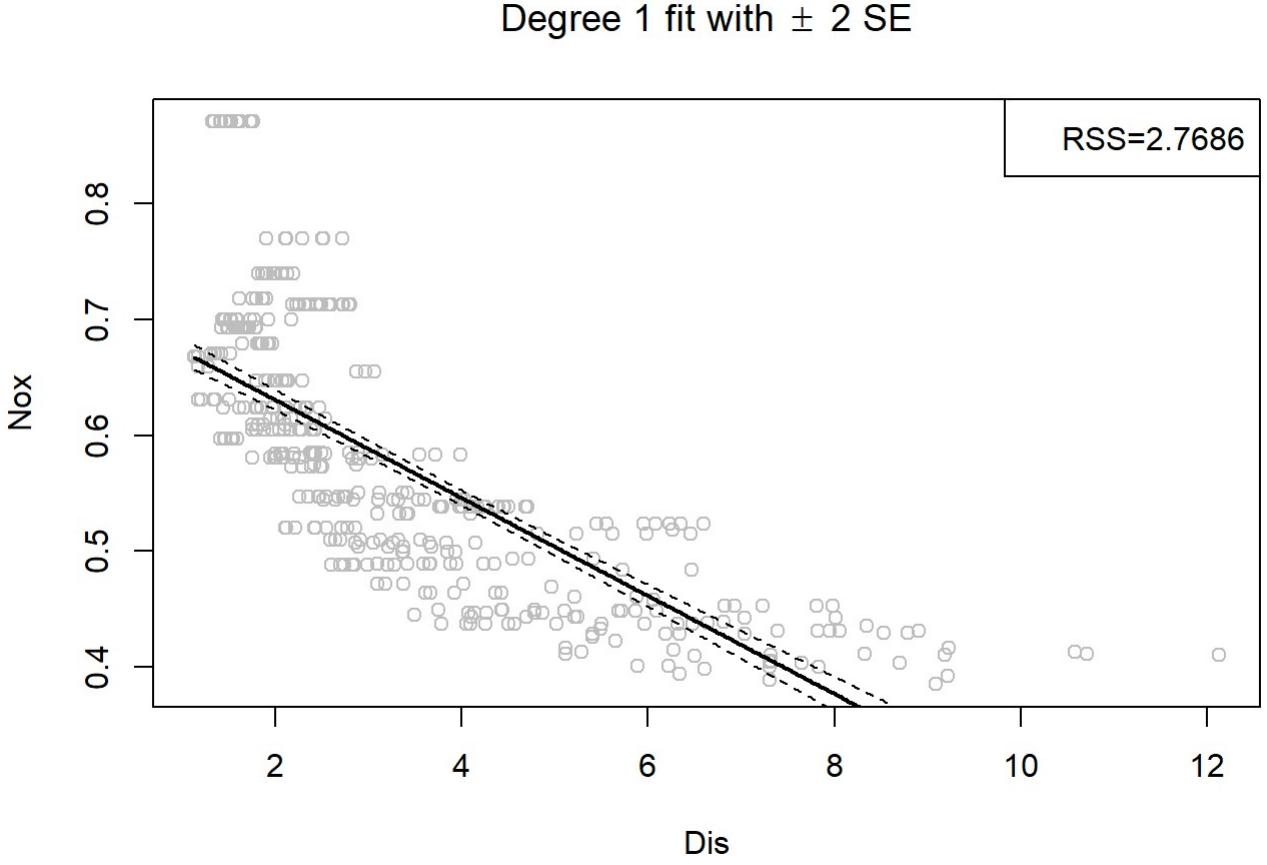
0.5547, -2.0031, 0.8563, -0.318

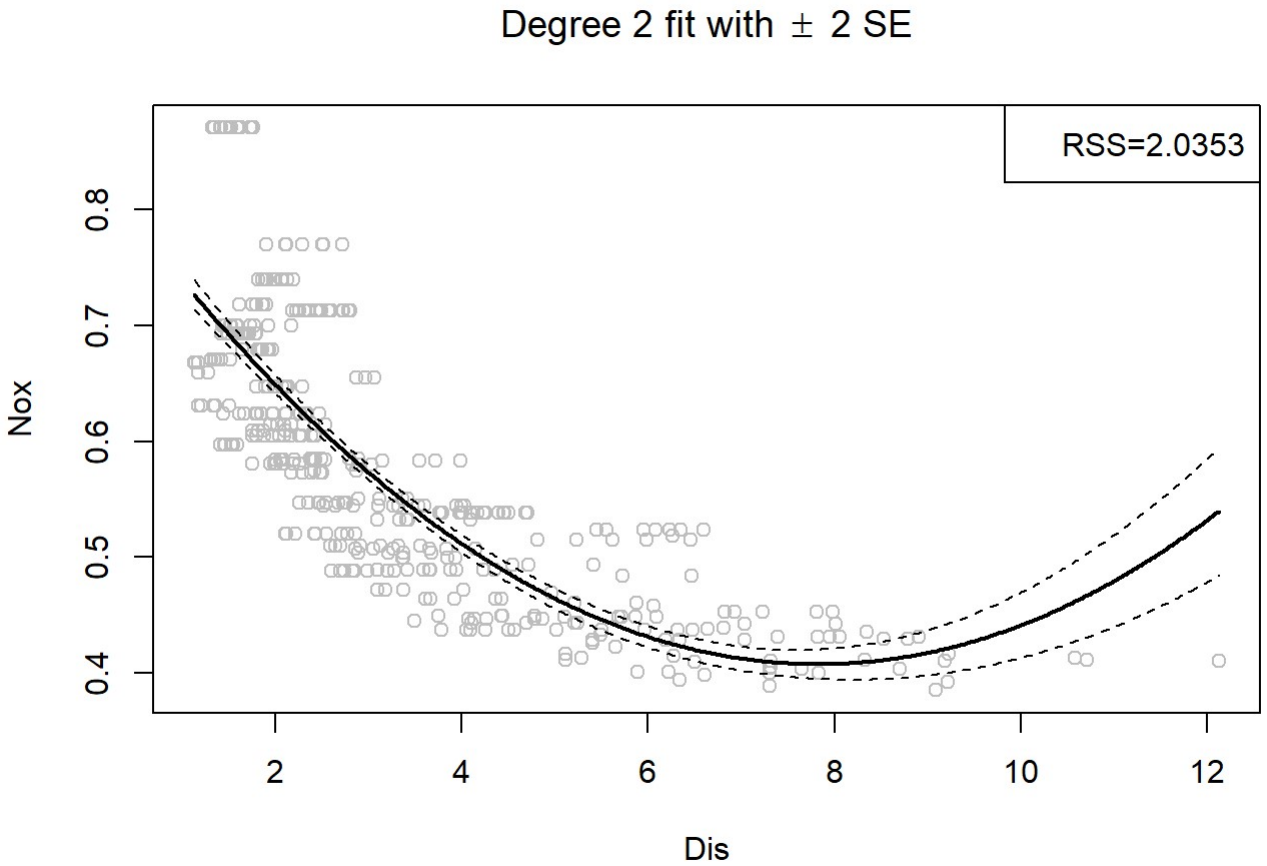
```
poly.plot(3)
```

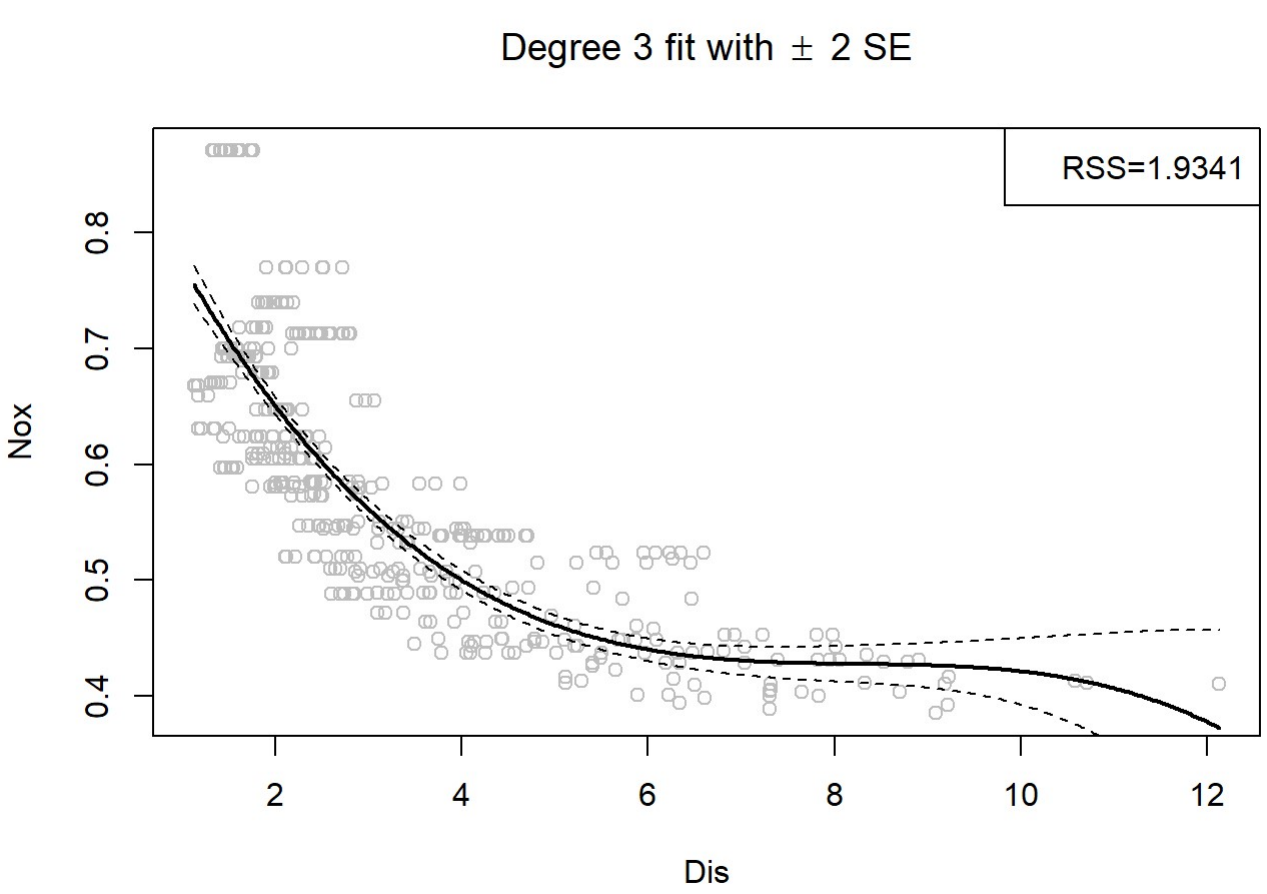


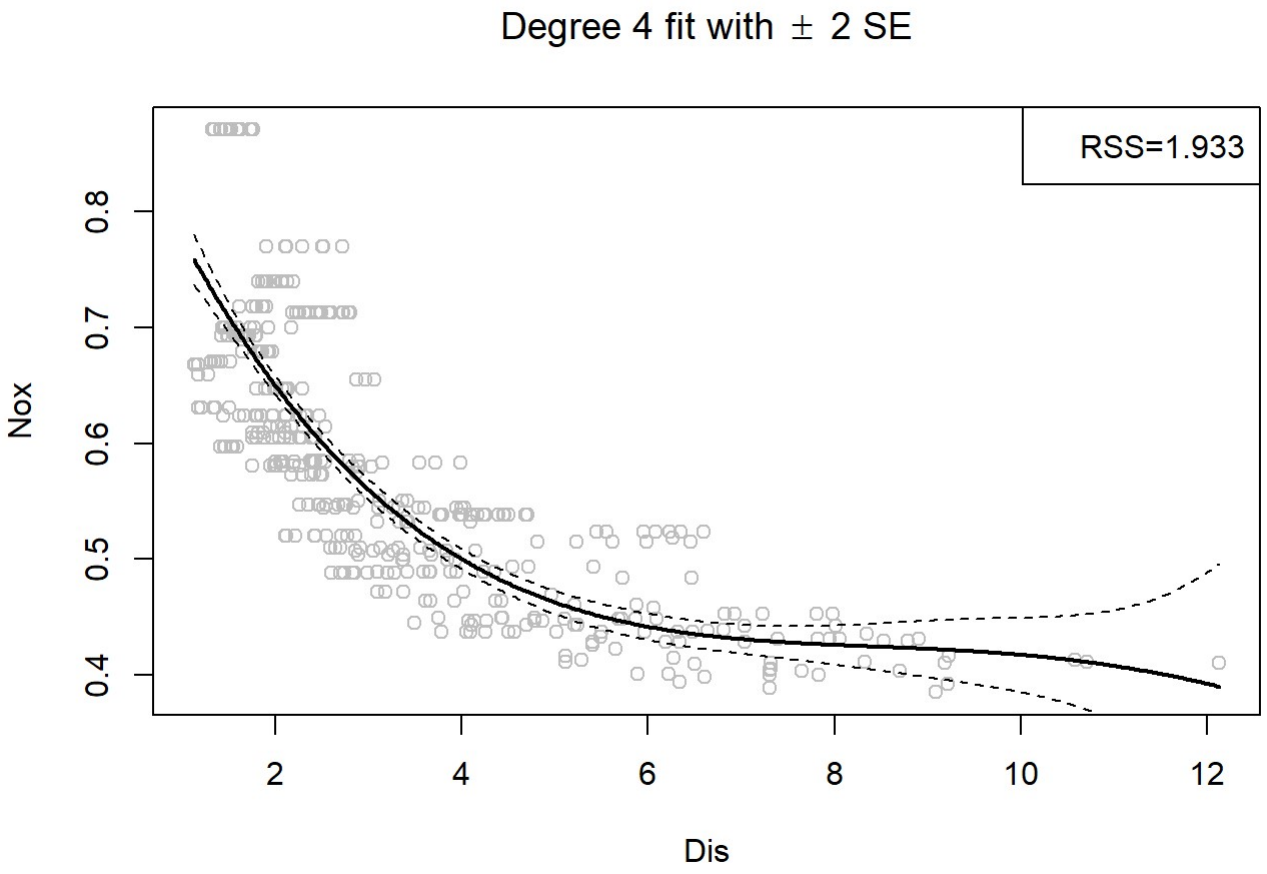
b)

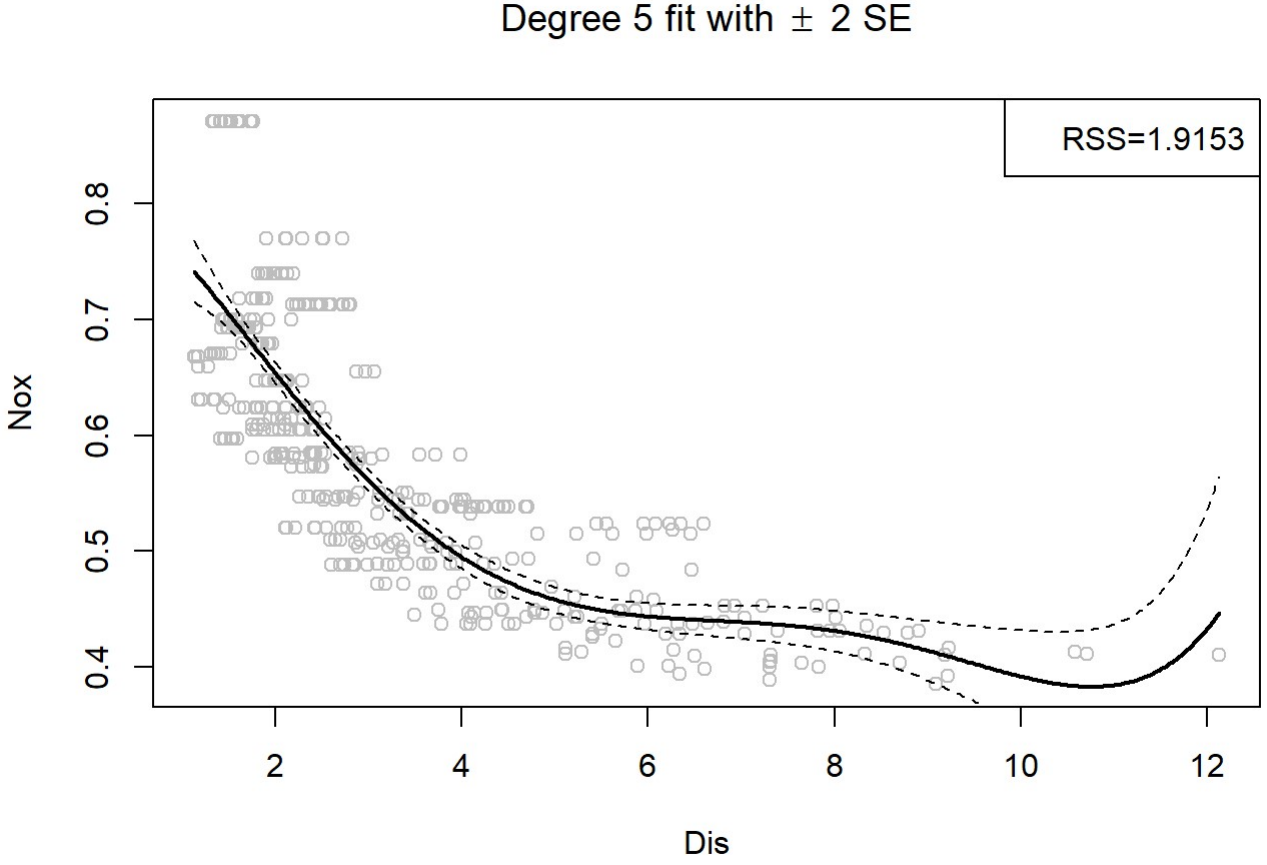
```
#Plots the fitted polynomial for degrees 1-10 with their associated RSS
invisible(sapply(1:10,poly.plot))
```

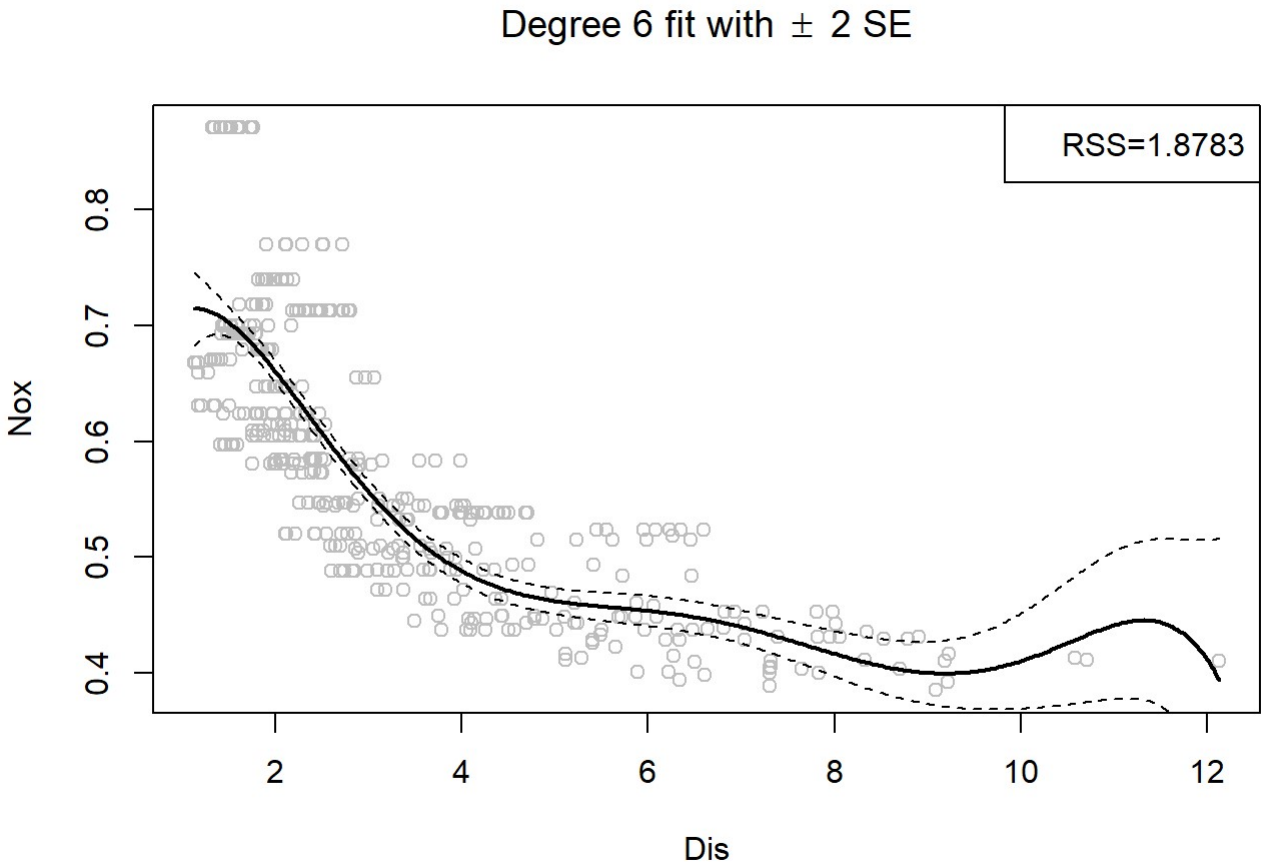

Degree 1 fit with ± 2 SE
RSS=2.7686
Nox
Dis
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12

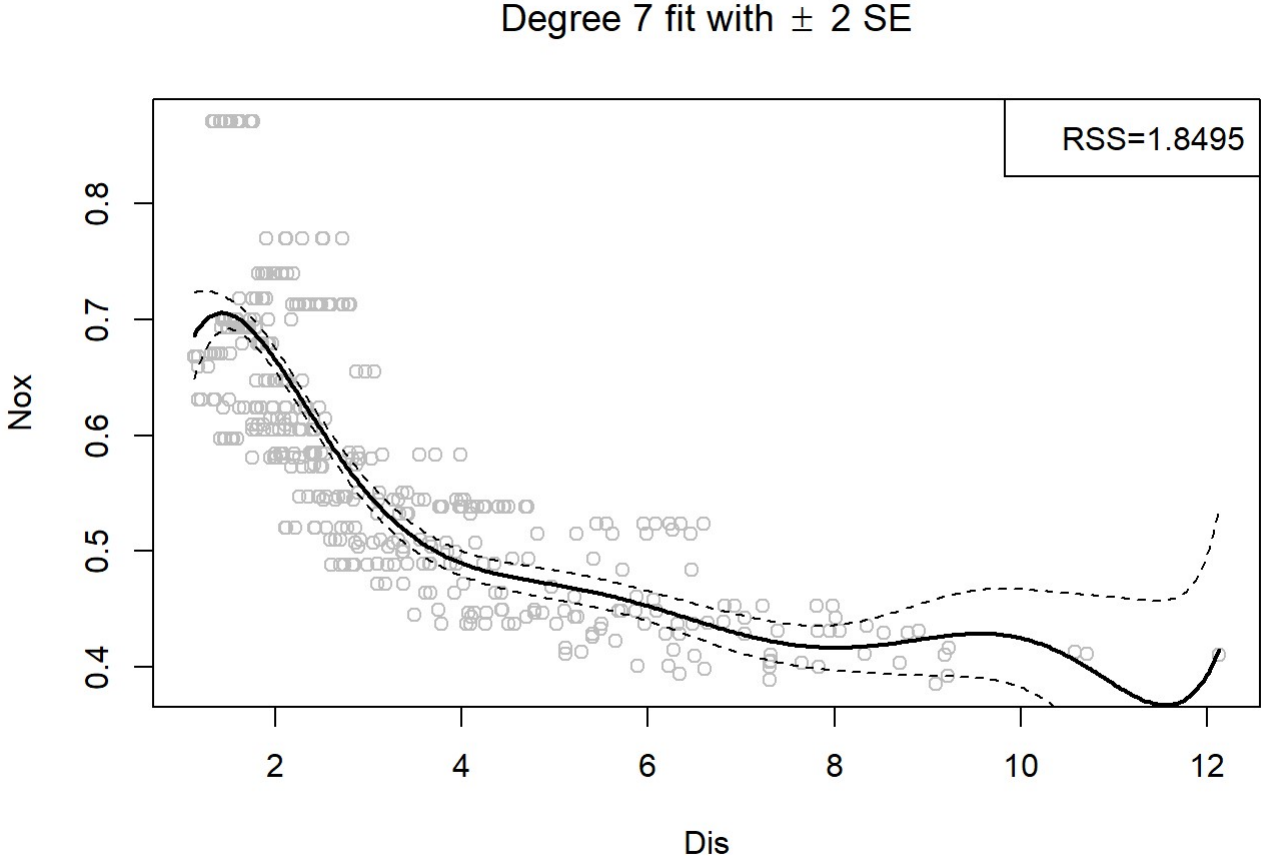

Degree 2 fit with ± 2 SE
RSS=2.0353
Nox
Dis
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12

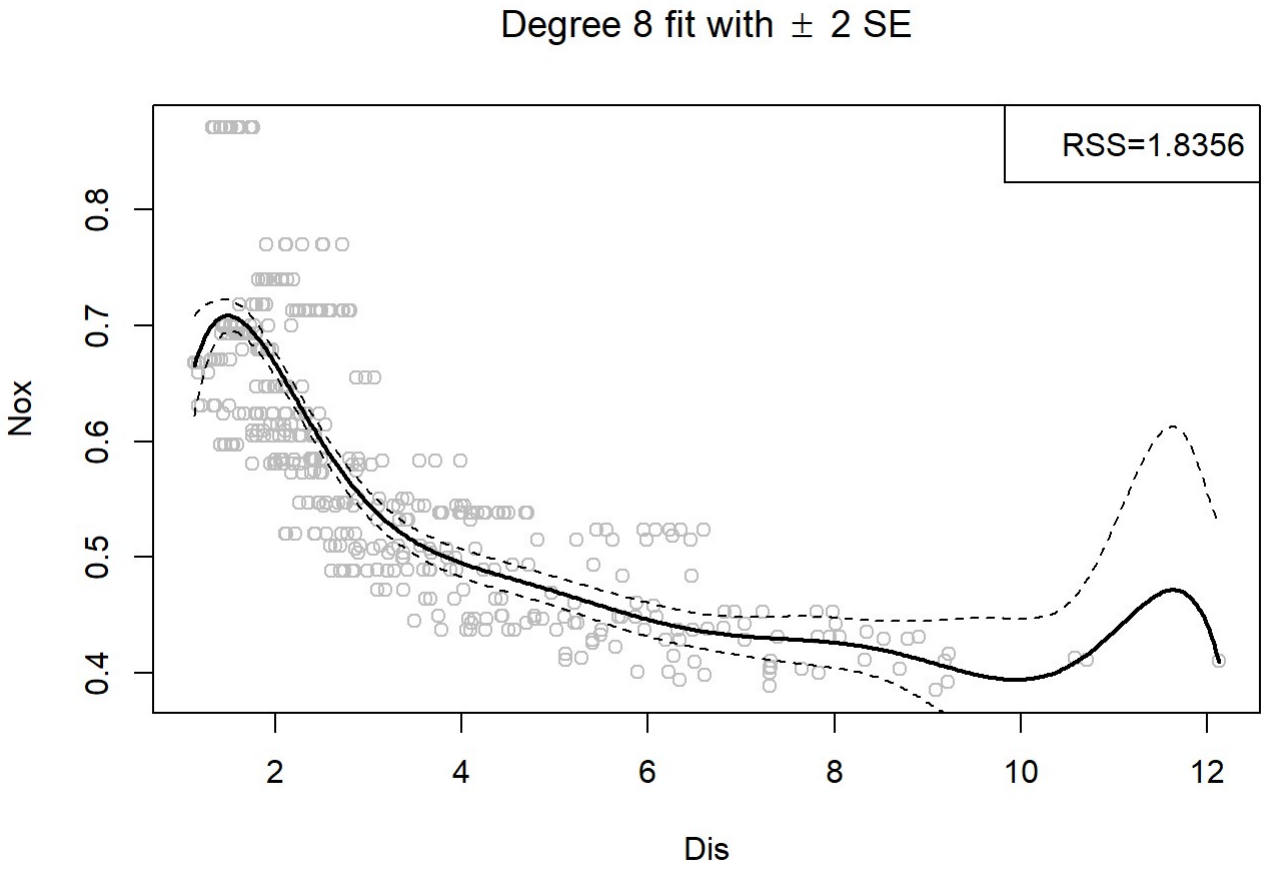

Degree 3 fit with ± 2 SE
RSS=1.9341
Nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
Dis

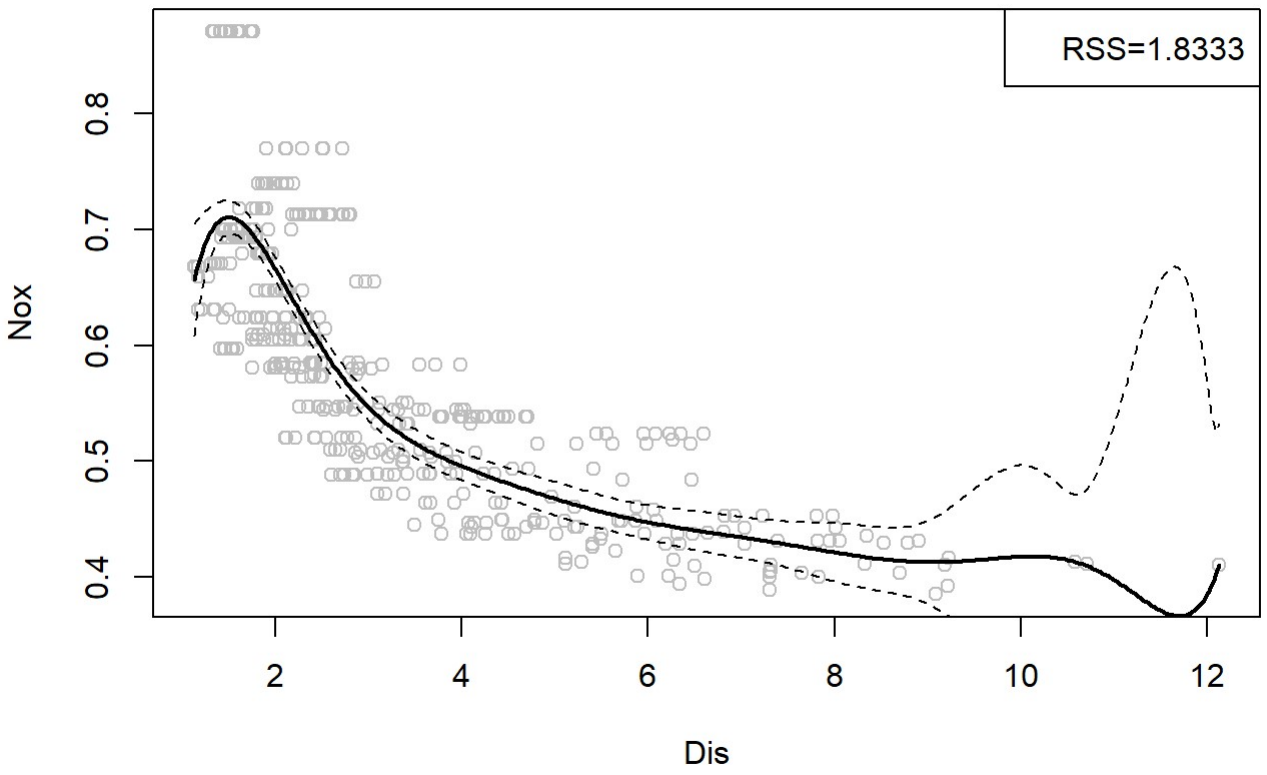

Degree 4 fit with ± 2 SE
RSS=1.933
Nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
Dis

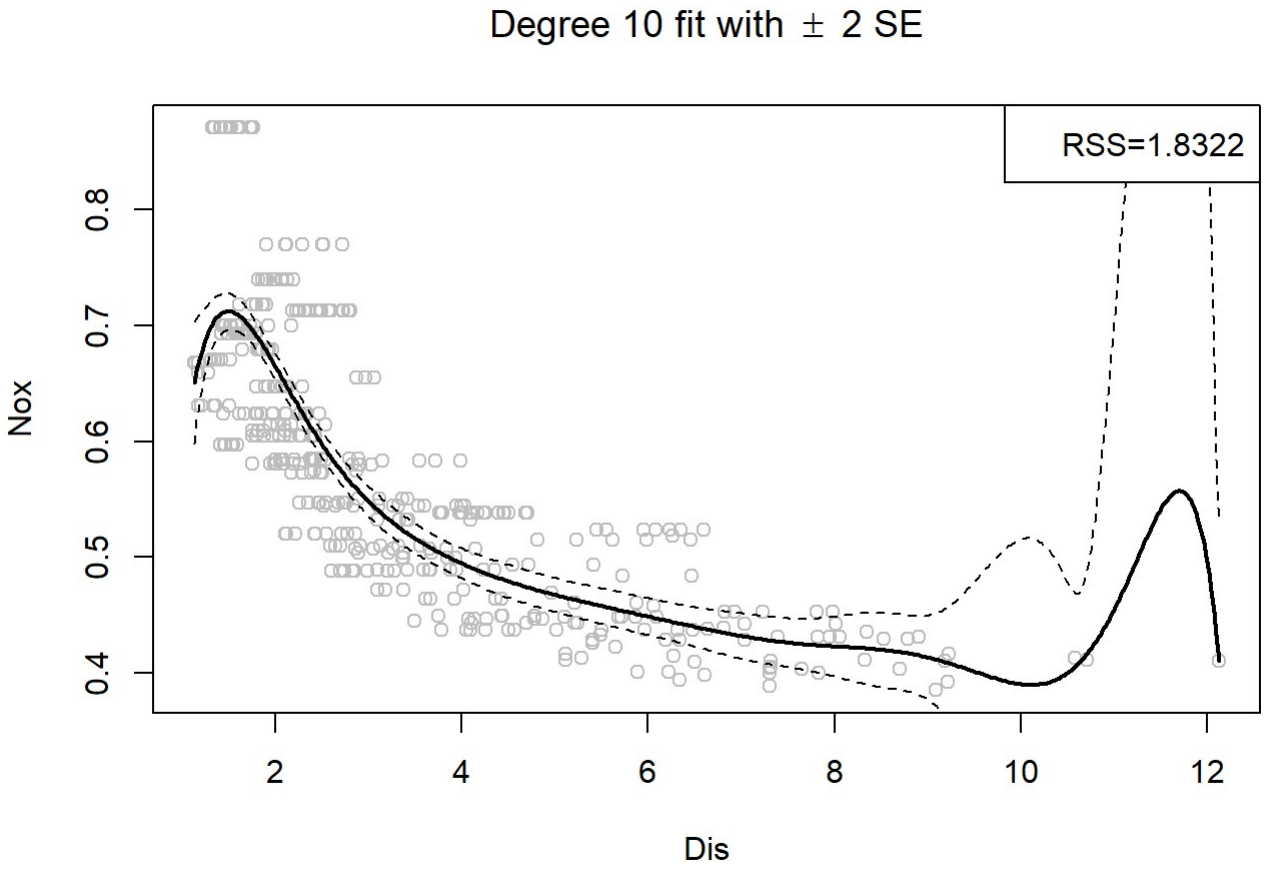

Degree 5 fit with ± 2 SE
RSS=1.9153
Nox
Dis
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12


Degree 6 fit with ± 2 SE
RSS=1.8783
Nox
Dis
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12


Degree 7 fit with ± 2 SE
RSS=1.8495
Nox
Dis
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12


Degree 8 fit with ± 2 SE
RSS=1.8356
Nox
Dis
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12


Degree 9 fit with ± 2 SE
RSS=1.8333
Nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
Dis


Degree 10 fit with ± 2 SE
RSS=1.8322
Nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
Dis

c)

```
set.seed(1)

get_optimal_d <- function(){

  #While loop to determine the highest degree such that all predictors are significant.
  max_degree <- 1
  while(TRUE){
    if(all(summary(poly.fit(max_degree))$coefficients[,4] < 0.05)==TRUE){
      max_degree <- max_degree +1
    } else {
      max_degree <- max_degree -1
      break}
  }

  # Performs 10-fold cross-validation for each polynomial up to the max degree and collects their associ
ated RSS.
  cv.error <- sapply(1:max_degree,function(d){
  fit <- glm(nox~poly(dis,d),data=Boston)
  cv.glm(Boston,fit,K=10)$delta[1]
  })

  #Returns the degree with the smallest CV error.
  return(which.min(cv.error))
}

get_optimal_d()
```

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

Based on cross-validation on the polynomials with only significant predictors, the degree of the polynomial with the smallest CV error is 3

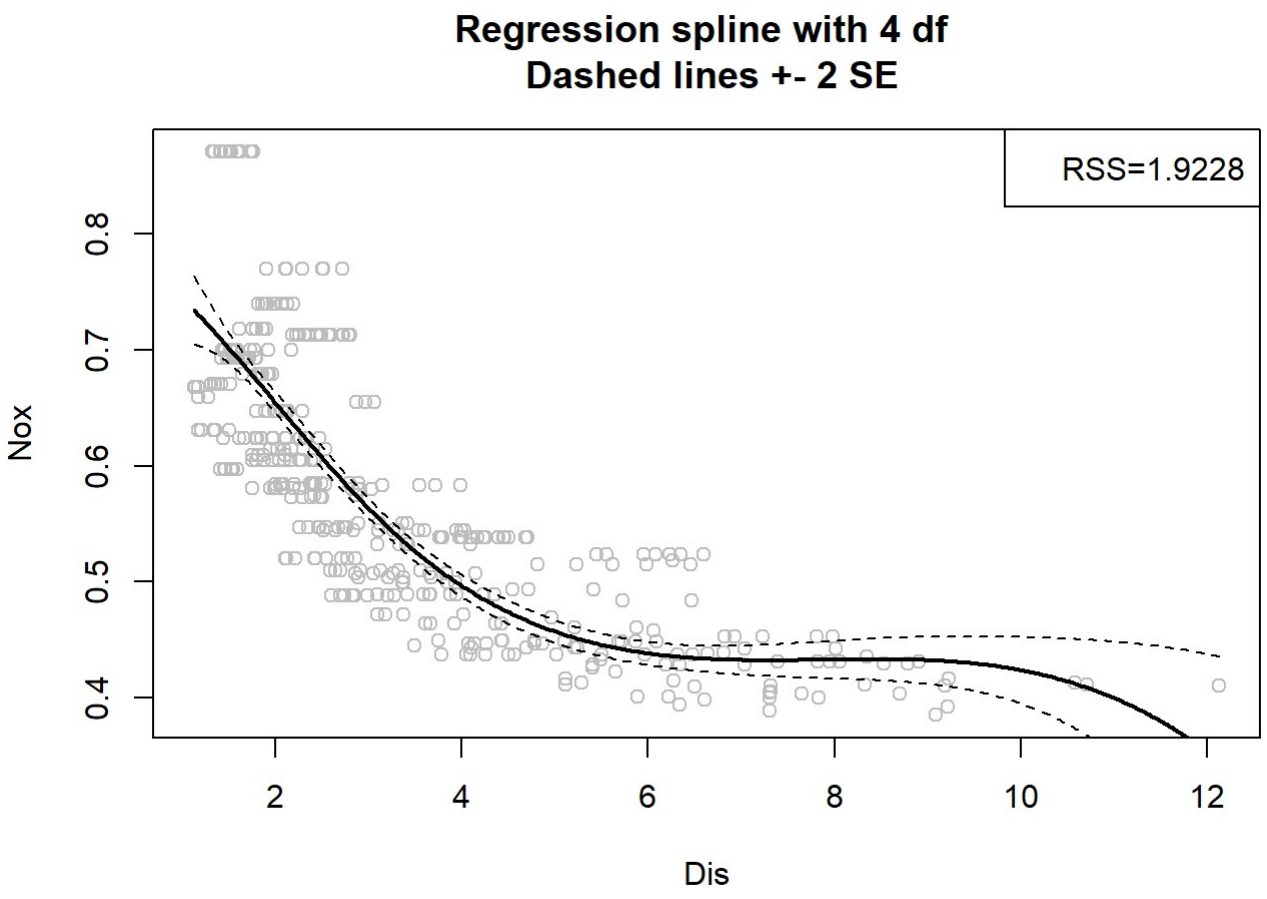
d)

```
#Knots are placed at equally-spaced quantiles of dis by the 'df' argument in bs.
spline.fit <- function(df){
  return(lm(nox~bs(dis,df=df), data=Boston))
}

#Function for plotting the regression spline with df degrees of freedom
spline.plot <- function(df){
  spline.pred <- predict(spline.fit(df),newdata=data.frame(dis=x),se.fit=TRUE)
  rss <- round(sum(summary(spline.fit(df))$residuals^2),4)
  plot(Boston$dis,Boston$nox,xlab='Dis',ylab='Nox',col='grey',
       main=paste('Regression spline with', df, 'df','\n Dashed lines +- 2 SE'))
  lines(x,spline.pred$fit,lwd=2)
  lines(x,spline.pred$fit + 2 *spline.pred$se.fit, lty="dashed")
  lines(x,spline.pred$fit - 2 *spline.pred$se.fit, lty="dashed")
  legend("topright", legend=glue('RSS=', rss))
}

print(summary(spline.fit(4)))
```

```
##
## Call:
## lm(formula = nox ~ bs(dis, df = df), data = Boston)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.124622 -0.039259 -0.008514  0.020850  0.193891
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)         0.73447    0.01460  50.306  < 2e-16 ***
## bs(dis, df = df)1 -0.05810    0.02186  -2.658  0.00812 **
## bs(dis, df = df)2 -0.46356    0.02366 -19.596  < 2e-16 ***
## bs(dis, df = df)3 -0.19979    0.04311  -4.634 4.58e-06 ***
## bs(dis, df = df)4 -0.38881    0.04551  -8.544  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06195 on 501 degrees of freedom
## Multiple R-squared:  0.7164, Adjusted R-squared:  0.7142
## F-statistic: 316.5 on 4 and 501 DF,  p-value: < 2.2e-16
```

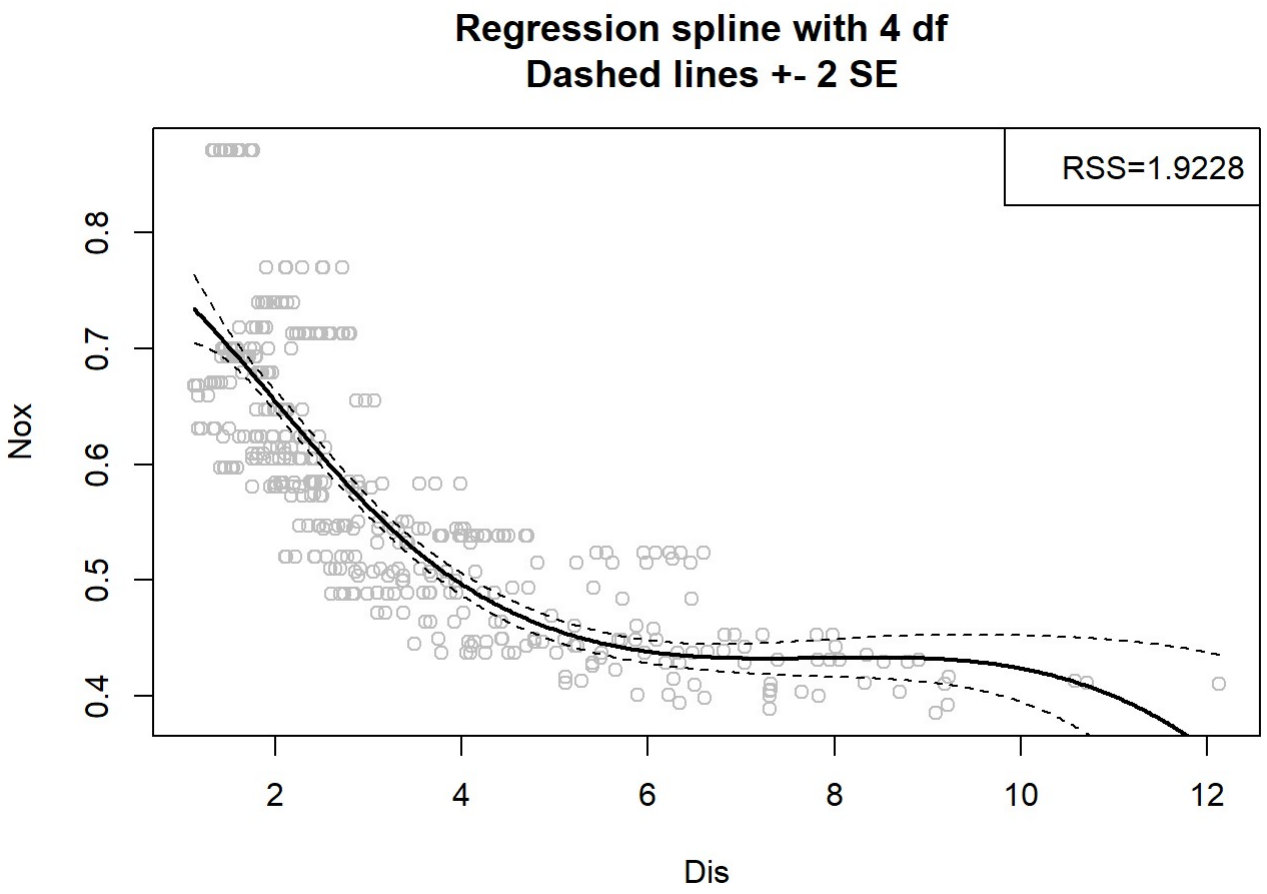
Summary shows that the regression spline with four degrees of freedom has Adjusted R^2 0.7142, p < 2.2e^-16. The coefficients 0.7345, -0.0581, -0.4636, -0.1998, -0.3888 are all significant.

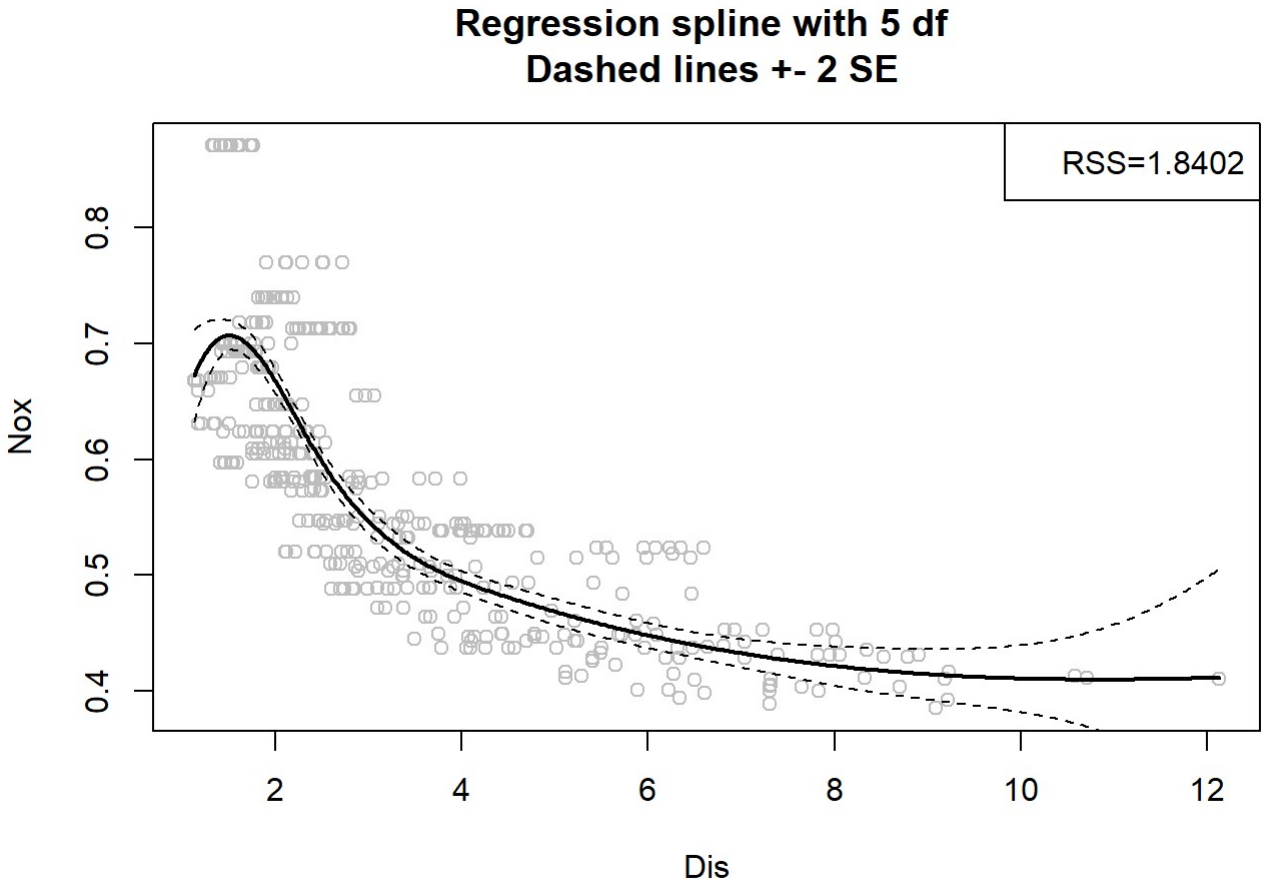
```
spline.plot(4)
```

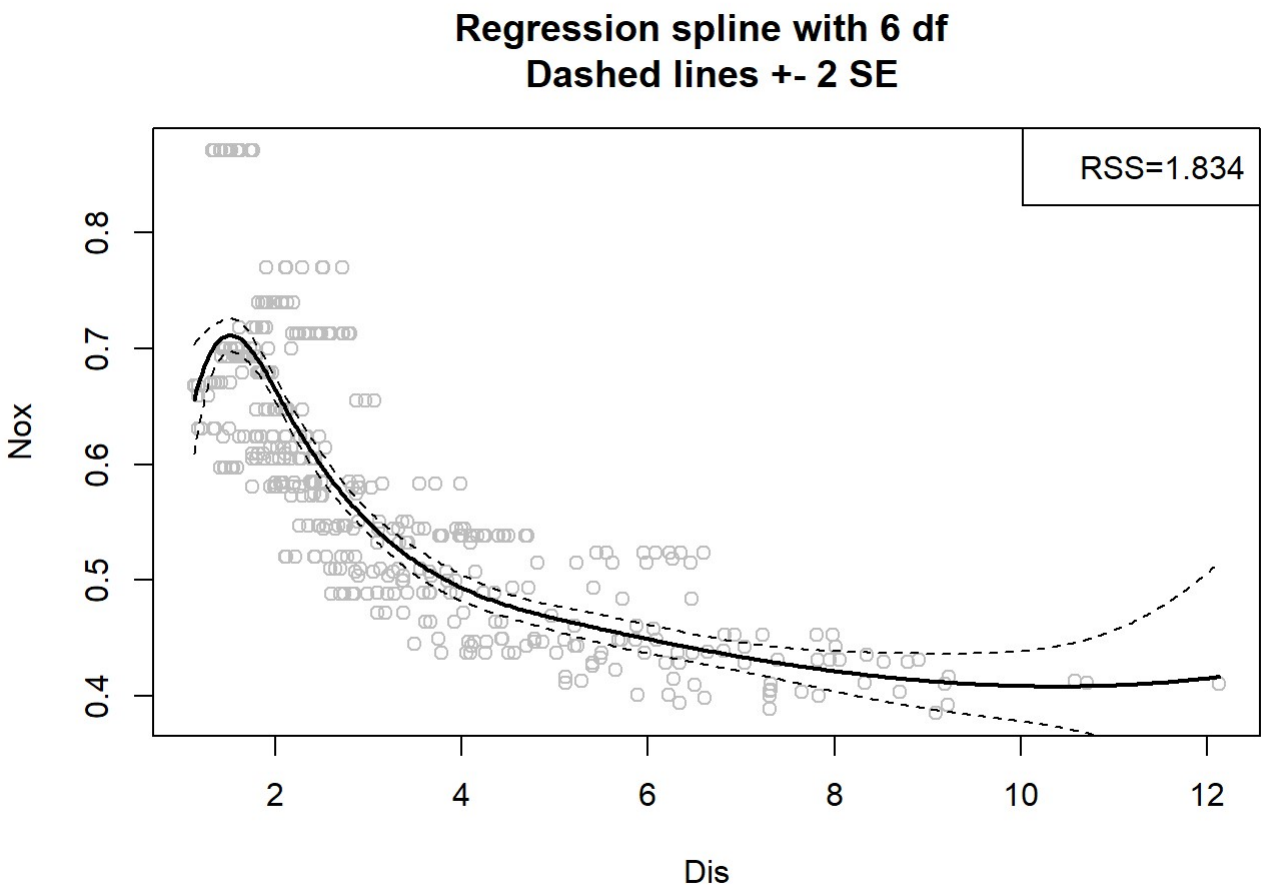


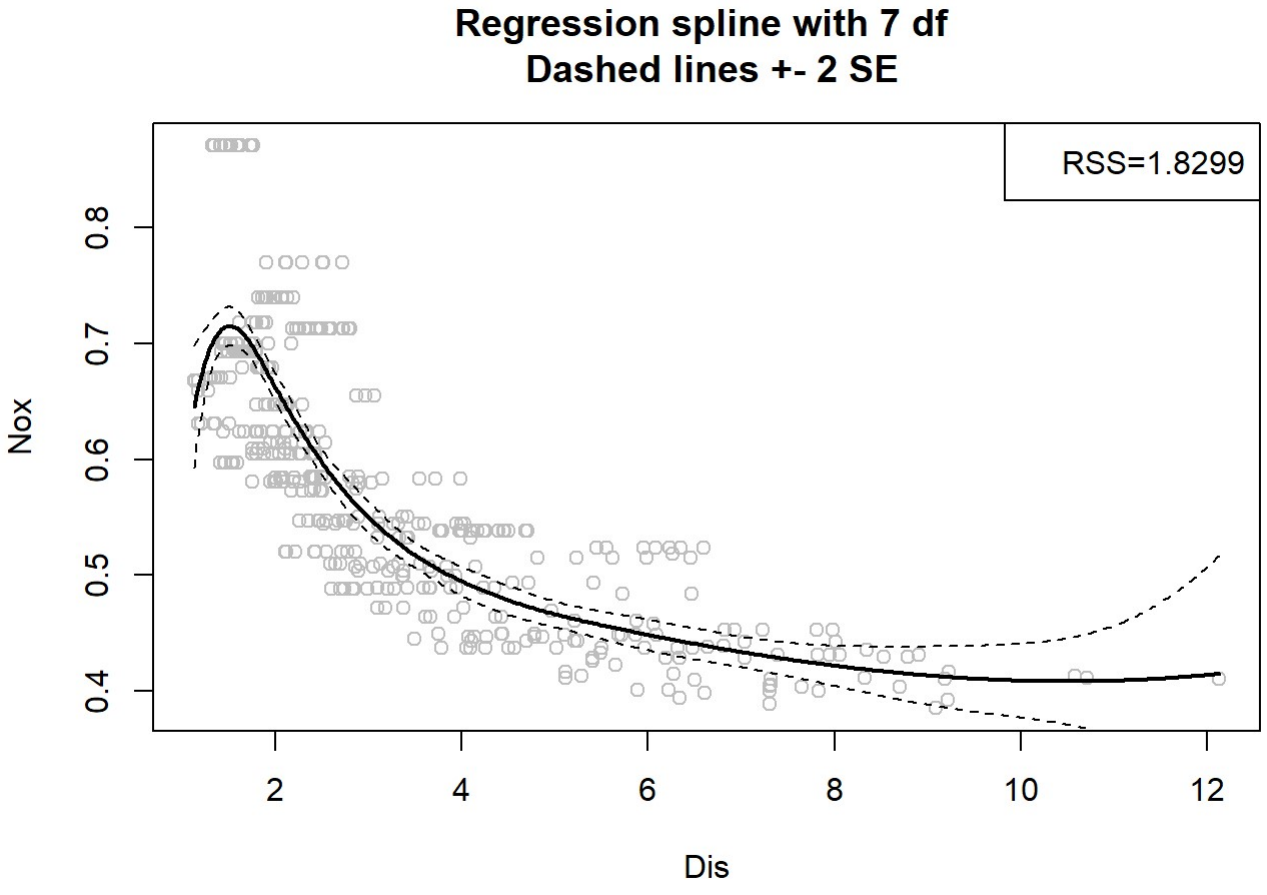
e)

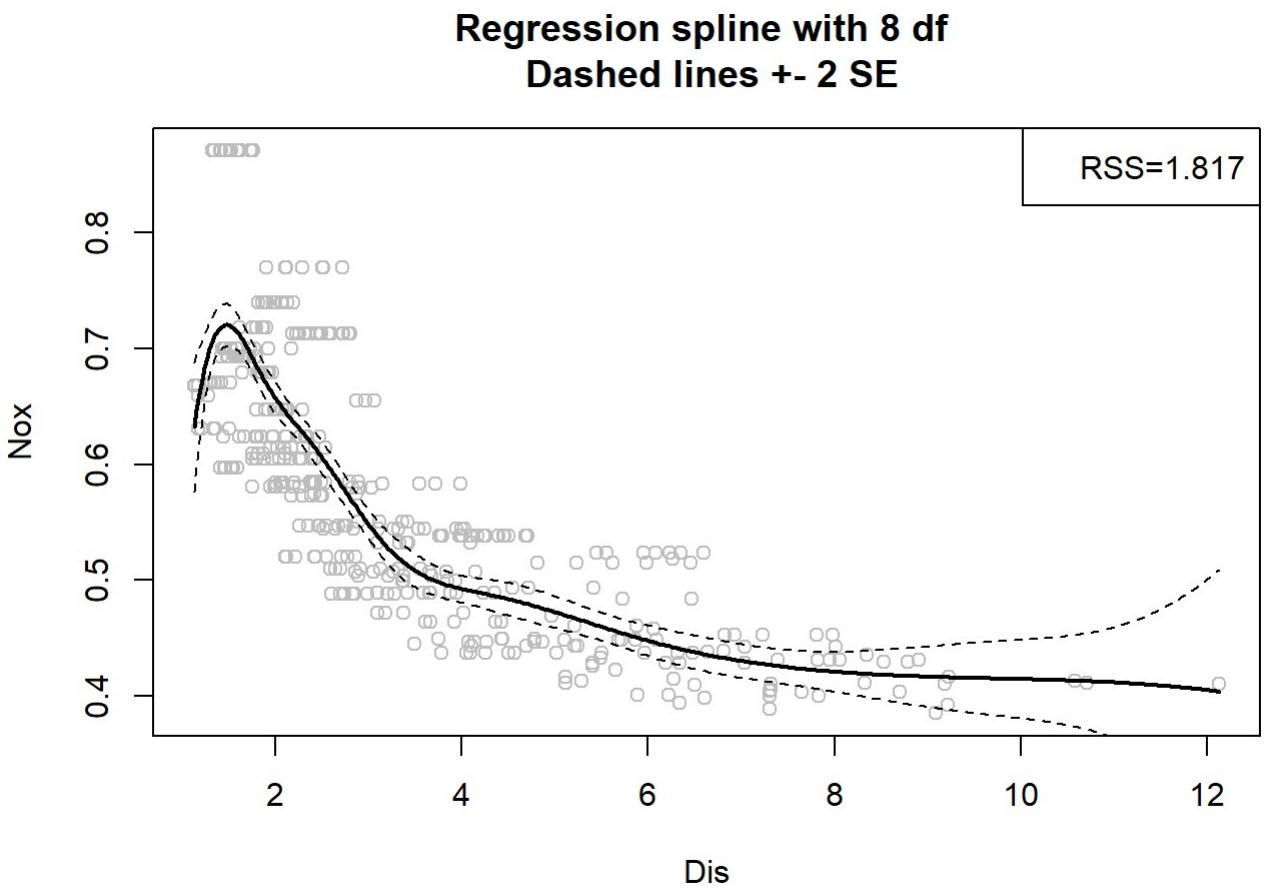
```
invisible(sapply(4:10,spline.plot))
```

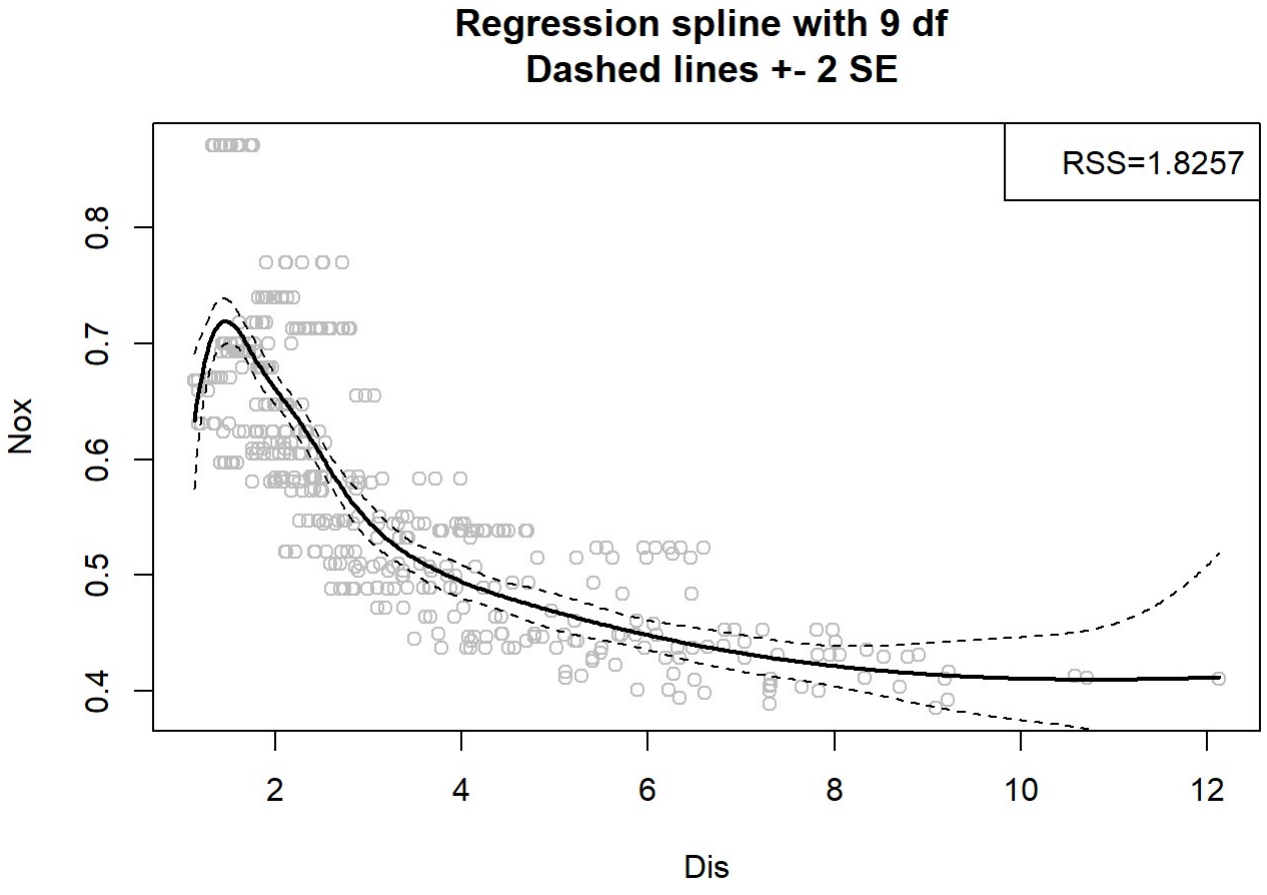

Regression spline with 4 df
Dashed lines +- 2 SE
RSS=1.9228
Nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
Dis

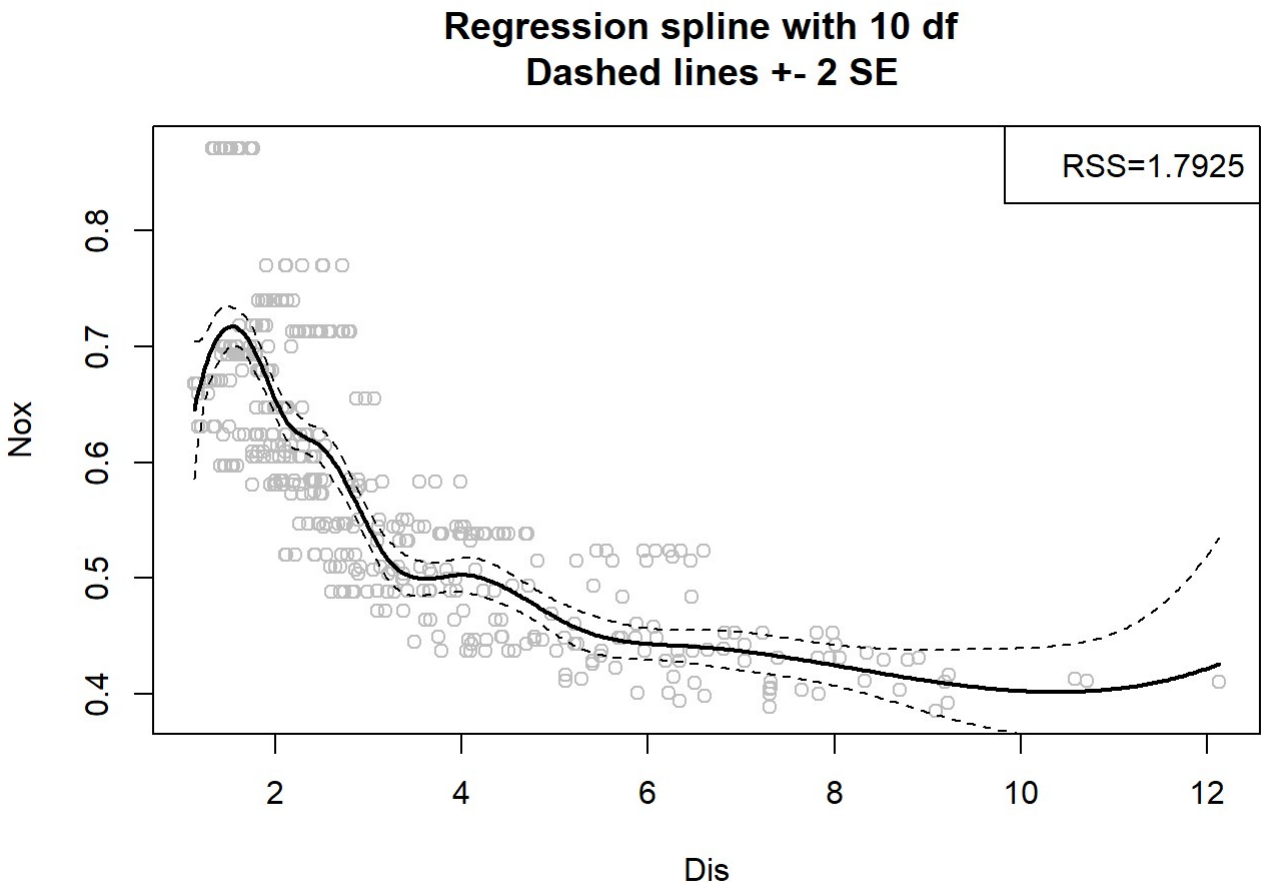

Regression spline with 5 df
Dashed lines +- 2 SE
RSS=1.8402
Nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
Dis


Regression spline with 6 df
Dashed lines +- 2 SE
RSS=1.834
Nox
Dis
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12


Regression spline with 7 df
Dashed lines +- 2 SE
RSS=1.8299
Nox
Dis
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12


Regression spline with 8 df
Dashed lines +- 2 SE
RSS=1.817
Nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
Dis


Regression spline with 9 df
Dashed lines +- 2 SE
RSS=1.8257
Nox
0.4
0.5
0.6
0.7
0.8
2
4
6
8
10
12
Dis



From the plots we see that the RSS usually decreases as the df increases, however the polynomials also seem increasingly overfitted.

```
set.seed(1)

get_optimal_df <- function(){

  #While loop to determine the largest degrees of freedom such that all predictors are significant.
  max_df <- 4
  while(TRUE){
    if(all(summary(spline.fit(max_df))$coefficients[,4] < 0.05)==TRUE){
      max_df <- max_df +1
    } else {
      max_df <- max_df -1
      break}
  }

  # Performs 10-fold cross-validation for each regression spline up to the max degrees of freedom and co
llects their associated RSS.
  cv.error <- sapply(4:max_df,function(d){
  fit <- glm(nox~bs(dis,df=d),data=Boston)
  cv.glm(Boston,fit,K=10)$delta[1]
  })

  #Returns the number of degrees of freedom associated with the smallest CV error.
  return(which.min(cv.error)+3)
}

print(get_optimal_df())
```

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

```
print(summary(spline.fit(get_optimal_df())))
```

```
##
## Call:
## lm(formula = nox ~ bs(dis, df = df), data = Boston)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.132814 -0.039097 -0.008521  0.023493  0.197761
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)        0.67248    0.01996  33.699  < 2e-16 ***
## bs(dis, df = df)1  0.08311    0.02875   2.891  0.00401 **
## bs(dis, df = df)2 -0.13460    0.01985  -6.782 3.35e-11 ***
## bs(dis, df = df)3 -0.25505    0.03477  -7.336 8.95e-13 ***
## bs(dis, df = df)4 -0.26785    0.04059  -6.599 1.06e-10 ***
## bs(dis, df = df)5 -0.26103    0.05214  -5.007 7.69e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06067 on 500 degrees of freedom
## Multiple R-squared:  0.7286, Adjusted R-squared:  0.7259
## F-statistic: 268.5 on 5 and 500 DF,  p-value: < 2.2e-16
```

Based on cross-validation on the range of degrees of freedom with only significant predictors, the number of df with the smallest CV error is 5. From the plots in d) we can see that it has 1.8402 RSS

Chapter 8 Problem 11

a)

```
library("ISLR2")
library("gbm")

data <- Caravan
data$Purchase <- as.integer(data$Purchase == "Yes") #Transforms Purchase values from no and yes to 0 and
1
training_data <- data[1:1000,]
test_data <- data[-(1:1000),]
```

b)

```
set.seed(1)
boost.caravan <- gbm(Purchase ~ . , data = training_data, n.trees=1000,
                     interaction.depth = 2, shrinkage = 0.01, distribution = "bernoulli")
#Fits a boosting model to the training pdata with 1000 trees, 0.01 shrinkage and 2 split per tree.

head(summary(boost.caravan, plotit=FALSE)) # Prints the six most important predictors
```

```
##               var  rel.inf
## PPERSAUT PPERSAUT 9.881968
## MKOOPKLA MKOOPKLA 7.090575
## MOPLHOOG MOPLHOOG 5.973837
## PBRAND     PBRAND 5.427953
## MGODGE     MGODGE 5.375333
## MBERMIDD MBERMIDD 3.948474
```

The most important predictors seem to be PPERSAUT, MKOOPKLA, MOPLHOOG, PBRAND, MGODGE, MBERMIDD

c)

```
yhat.boost <- predict(boost.caravan, newdata=test_data, n.trees=1000, type="response")

yhat.boost <- ifelse(yhat.boost > 0.2, 1, 0)

confusionmatrix <- table(predicted=yhat.boost, true=test_data$Purchase)

print(confusionmatrix)
```

```
##          true
## predicted    0    1
##         0 4359  252
##         1  174   37
```

The fraction of people predicted to make a purchase by the boosted tree that in fact made one is $\frac{37}{174+37} = 0.1754$

```
lr.fit <- glm(Purchase ~ ., data=training_data, family='binomial')
lr.probs <- predict(lr.fit, test_data, type="response")
lr.pred <- ifelse(lr.probs > 0.5, 1, 0)
lr.confusionmatrix <- table(predicted = lr.pred, true= test_data$Purchase)

print(lr.confusionmatrix)
```

```
##          true
## predicted    0    1
##         0 4446  274
##         1   87   15
```

The fraction of people predicted to make a purchase by logistic regression that in fact made one is $\frac{15}{87+15} = 0.1471$

```
library("class")
library("kknn")

get_optimal_k <- function(){

train.X <- training_data[,1:ncol(training_data)-1] # X values in training data
train.Y <- training_data$Purchase # Y values in training data

cv.error <- sapply(seq(1,40,2),function(k){
  set.seed(1)
  return(mean(knn.cv(train.X,train.Y,k,l=0,prob=FALSE,use.all=TRUE) != train.Y))
})
# Produces a list cv.error containing the MSE of a range of k via CV

return(seq(1,40,2)[which.min(cv.error)])
}
#Returns the k with lowest CV MSE


# Given k knn.pred returns the predicted Purchase in test data via k nearest neighbors.
knn.pred <- function(k){

  train.X <- training_data[,1:ncol(training_data)-1] #X values in training data
  train.Y <- training_data$Purchase # Y values in training data
  test.X <- test_data[,1:ncol(test_data)-1] # X values in test data

  return(knn(train.X, test.X, train.Y, k=k))
}



knn.confusionmatrix <- table(Predicted=knn.pred(get_optimal_k()), True=test_data$Purchase)

print(knn.confusionmatrix)
```

```
##          True
## Predicted    0    1
##         0 4502  285
##         1   31    4
```

The fraction of people predicted to make a purchase by knn that in fact made one is $\frac{4}{31+4} = 0.1143$

Between the three models, the boosted tree had the best true positive to false positive ratio and KNN the worst. Perhaps this is an indication that the relationship between X and Y is discontinuous in nature?

Chapter 9 Problem 7

a)

```
library("ISLR2")
library("e1071")
```

```
data <- Auto[,-which(names(Auto)=="mpg")]
mpg01 <- ifelse(Auto$mpg > median(Auto$mpg), 1, 0)
df <- data.frame(data,mpg01=as.factor(mpg01))
```

b)

```
set.seed(1)
tune.linear <- tune(
  svm,
  mpg01 ~ .,
  data   = df,
  kernel = "linear",
  ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100))
)

summary(tune.linear)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.08673077
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-03 0.13525641 0.05661708
## 2 1e-02 0.08923077 0.04698309
## 3 1e-01 0.08673077 0.04040897
## 4 1e+00 0.09961538 0.04923181
## 5 5e+00 0.11230769 0.05826857
## 6 1e+01 0.11237179 0.05701890
## 7 1e+02 0.11750000 0.06208951
```

```
print(tune.linear$best.model)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = mpg01 ~ ., data = df, ranges = list(cost = c(0.001,
##     0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##
## Number of Support Vectors:  109
```

Here we can see that the best model is the one with cv error = 0.0867308. The model has cost = 0.1 and 109 support vectors.

c)

```
set.seed(1)
tune.polynomial <- tune(
  svm,
  mpg01 ~ .,
  data   = df,
  kernel = "polynomial",
  ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100),
                degree = c(1,2,3,4,5,6))
)

summary(tune.polynomial)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##   100      1
##
## - best performance: 0.08173077
##
## - Detailed performance results:
##     cost degree      error dispersion
## 1  1e-03      1 0.55115385 0.04366593
## 2  1e-02      1 0.55115385 0.04366593
## 3  1e-01      1 0.28596154 0.10442771
## 4  1e+00      1 0.10717949 0.04299154
## 5  5e+00      1 0.08673077 0.04551036
## 6  1e+01      1 0.08416667 0.04010502
## 7  1e+02      1 0.08173077 0.03986661
## 8  1e-03      2 0.55115385 0.04366593
## 9  1e-02      2 0.55115385 0.04366593
## 10 1e-01      2 0.55115385 0.04366593
## 11 1e+00      2 0.55115385 0.04366593
## 12 5e+00      2 0.55115385 0.04366593
## 13 1e+01      2 0.52064103 0.08505283
## 14 1e+02      2 0.31673077 0.09410274
## 15 1e-03      3 0.55115385 0.04366593
## 16 1e-02      3 0.55115385 0.04366593
## 17 1e-01      3 0.55115385 0.04366593
## 18 1e+00      3 0.55115385 0.04366593
## 19 5e+00      3 0.55115385 0.04366593
## 20 1e+01      3 0.55115385 0.04366593
## 21 1e+02      3 0.40326923 0.10793388
## 22 1e-03      4 0.55115385 0.04366593
## 23 1e-02      4 0.55115385 0.04366593
## 24 1e-01      4 0.55115385 0.04366593
## 25 1e+00      4 0.55115385 0.04366593
## 26 5e+00      4 0.55115385 0.04366593
## 27 1e+01      4 0.55115385 0.04366593
## 28 1e+02      4 0.55115385 0.04366593
## 29 1e-03      5 0.55115385 0.04366593
## 30 1e-02      5 0.55115385 0.04366593
## 31 1e-01      5 0.55115385 0.04366593
## 32 1e+00      5 0.55115385 0.04366593
## 33 5e+00      5 0.55115385 0.04366593
## 34 1e+01      5 0.55115385 0.04366593
## 35 1e+02      5 0.55115385 0.04366593
## 36 1e-03      6 0.55115385 0.04366593
## 37 1e-02      6 0.55115385 0.04366593
## 38 1e-01      6 0.55115385 0.04366593
## 39 1e+00      6 0.55115385 0.04366593
## 40 5e+00      6 0.55115385 0.04366593
## 41 1e+01      6 0.55115385 0.04366593
## 42 1e+02      6 0.55115385 0.04366593
```

```
print(tune.polynomial$best.model)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = mpg01 ~ ., data = df, ranges = list(cost = c(0.001,
##     0.01, 0.1, 1, 5, 10, 100), degree = c(1, 2, 3, 4, 5, 6)), kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  100
##      degree:  1
##      coef.0:  0
##
## Number of Support Vectors:  107
```

Here we can see that the best model with cv.error = 0.0817308 is the one with cost = 100 and degree = 1. The model has 107 support vectors.

```
set.seed(1)
tune.radial <- tune(
  svm,
  mpg01 ~ .,
  data   = df,
  kernel = "radial",
  ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100),
                gamma = c(0.5, 1, 2,3,4))
)

summary(tune.radial)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10     1
##
## - best performance: 0.07897436
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-03   0.5 0.55115385 0.04366593
## 2  1e-02   0.5 0.55115385 0.04366593
## 3  1e-01   0.5 0.08410256 0.04164179
## 4  1e+00   0.5 0.08673077 0.04708817
## 5  5e+00   0.5 0.08660256 0.04514614
## 6  1e+01   0.5 0.09173077 0.04008042
## 7  1e+02   0.5 0.09429487 0.03796985
## 8  1e-03   1.0 0.55115385 0.04366593
## 9  1e-02   1.0 0.55115385 0.04366593
## 10 1e-01   1.0 0.55115385 0.04366593
## 11 1e+00   1.0 0.07903846 0.04891067
## 12 5e+00   1.0 0.08147436 0.04910668
## 13 1e+01   1.0 0.07897436 0.04869339
## 14 1e+02   1.0 0.07897436 0.04869339
## 15 1e-03   2.0 0.55115385 0.04366593
## 16 1e-02   2.0 0.55115385 0.04366593
## 17 1e-01   2.0 0.55115385 0.04366593
## 18 1e+00   2.0 0.13769231 0.06926822
## 19 5e+00   2.0 0.13512821 0.06692968
## 20 1e+01   2.0 0.13512821 0.06692968
## 21 1e+02   2.0 0.13512821 0.06692968
## 22 1e-03   3.0 0.55115385 0.04366593
## 23 1e-02   3.0 0.55115385 0.04366593
## 24 1e-01   3.0 0.55115385 0.04366593
## 25 1e+00   3.0 0.37012821 0.14598387
## 26 5e+00   3.0 0.32935897 0.14522774
## 27 1e+01   3.0 0.32935897 0.14522774
## 28 1e+02   3.0 0.32935897 0.14522774
## 29 1e-03   4.0 0.55115385 0.04366593
## 30 1e-02   4.0 0.55115385 0.04366593
## 31 1e-01   4.0 0.55115385 0.04366593
## 32 1e+00   4.0 0.47955128 0.05564953
## 33 5e+00   4.0 0.47698718 0.06085690
## 34 1e+01   4.0 0.47698718 0.06085690
## 35 1e+02   4.0 0.47698718 0.06085690
```

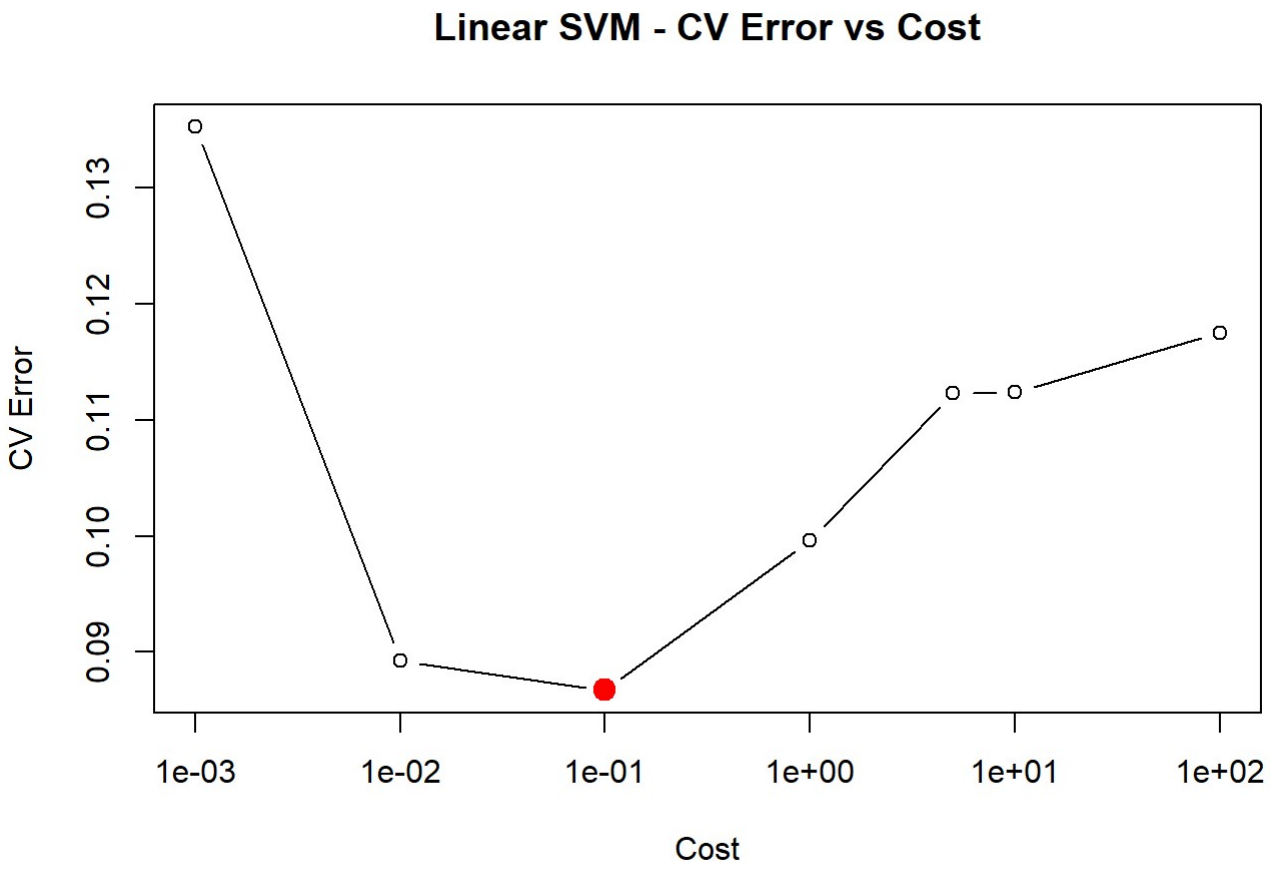
```
print(tune.radial$best.model)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = mpg01 ~ ., data = df, ranges = list(cost = c(0.001,
##     0.01, 0.1, 1, 5, 10, 100), gamma = c(0.5, 1, 2, 3, 4)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##
## Number of Support Vectors:  379
```

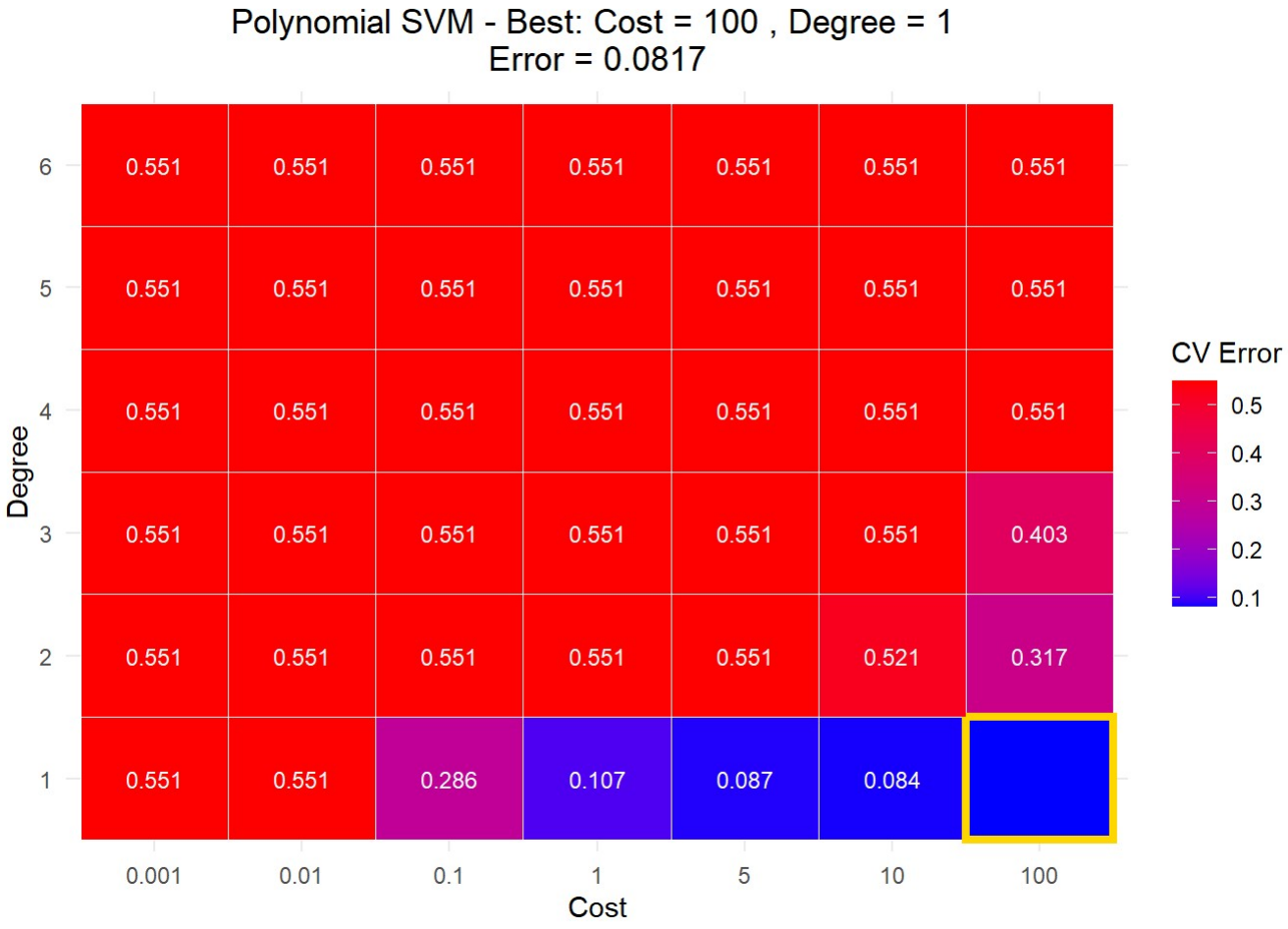
Here we can see that the best model with cv.error = 0.0789744 is the one with cost = 10 and gamma = 1. The model has 379 support vectors.

d)

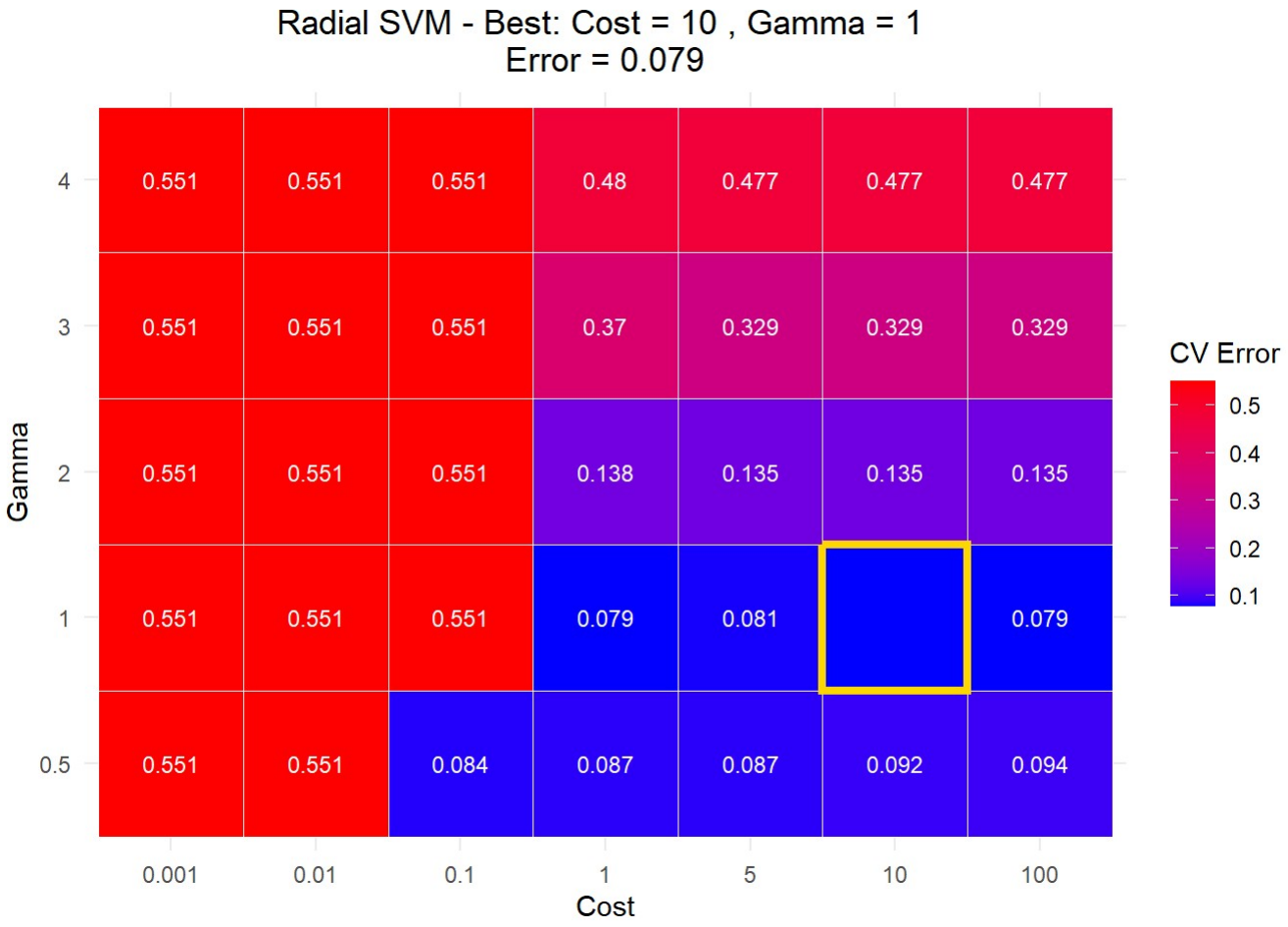
```
library("ggplot2")
library("reshape2")

# Plot cross-validation errors for linear kernel
plot(tune.linear$performances$cost, tune.linear$performances$error,
     type = "b", log = "x", xlab = "Cost", ylab = "CV Error",
     main = "Linear SVM - CV Error vs Cost")
points(tune.linear$best.parameters$cost,
       tune.linear$performances[tune.linear$performances$cost == tune.linear$best.parameters$cost, "erro
r"],
       col = "red", pch = 19, cex = 1.5)
```



```
# Heatmaps with best parameters highlighted

# Polynomial SVM heatmap with best parameter highlighted
polynomial_perf <- tune.polynomial$performances
best_poly_cost <- tune.polynomial$best.parameters$cost
best_poly_degree <- tune.polynomial$best.parameters$degree

polynomial_heatmap <- ggplot(polynomial_perf, aes(x = factor(cost), y = factor(degree), fill = error)) +
  geom_tile(color = "white") +
  scale_fill_gradient(low = "blue", high = "red", name = "CV Error") +
  geom_text(aes(label = round(error, 3)), color = "white", size = 3) +
  # Highlight the best parameters
  geom_tile(data = polynomial_perf[polynomial_perf$cost == best_poly_cost &
                                   polynomial_perf$degree == best_poly_degree, ],
            color = "gold", size = 1.5) +
  labs(title = paste("Polynomial SVM - Best: Cost =", best_poly_cost,
                     ", Degree =", best_poly_degree, "\nError =",
                     round(min(polynomial_perf$error), 4)),
       x = "Cost",
       y = "Degree") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))

print(polynomial_heatmap)
```



```
# Radial SVM heatmap with best parameter highlighted
radial_perf <- tune.radial$performances
best_radial_cost <- tune.radial$best.parameters$cost
best_radial_gamma <- tune.radial$best.parameters$gamma

radial_heatmap <- ggplot(radial_perf, aes(x = factor(cost), y = factor(gamma), fill = error)) +
  geom_tile(color = "white") +
  scale_fill_gradient(low = "blue", high = "red", name = "CV Error") +
  geom_text(aes(label = round(error, 3)), color = "white", size = 3) +
  # Highlight the best parameters
  geom_tile(data = radial_perf[radial_perf$cost == best_radial_cost &
                               radial_perf$gamma == best_radial_gamma, ],
            color = "gold", size = 1.5) +
  labs(title = paste("Radial SVM - Best: Cost =", best_radial_cost,
                     ", Gamma =", best_radial_gamma, "\nError =",
                     round(min(radial_perf$error), 4)),
       x = "Cost",
       y = "Gamma") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))

print(radial_heatmap)
```



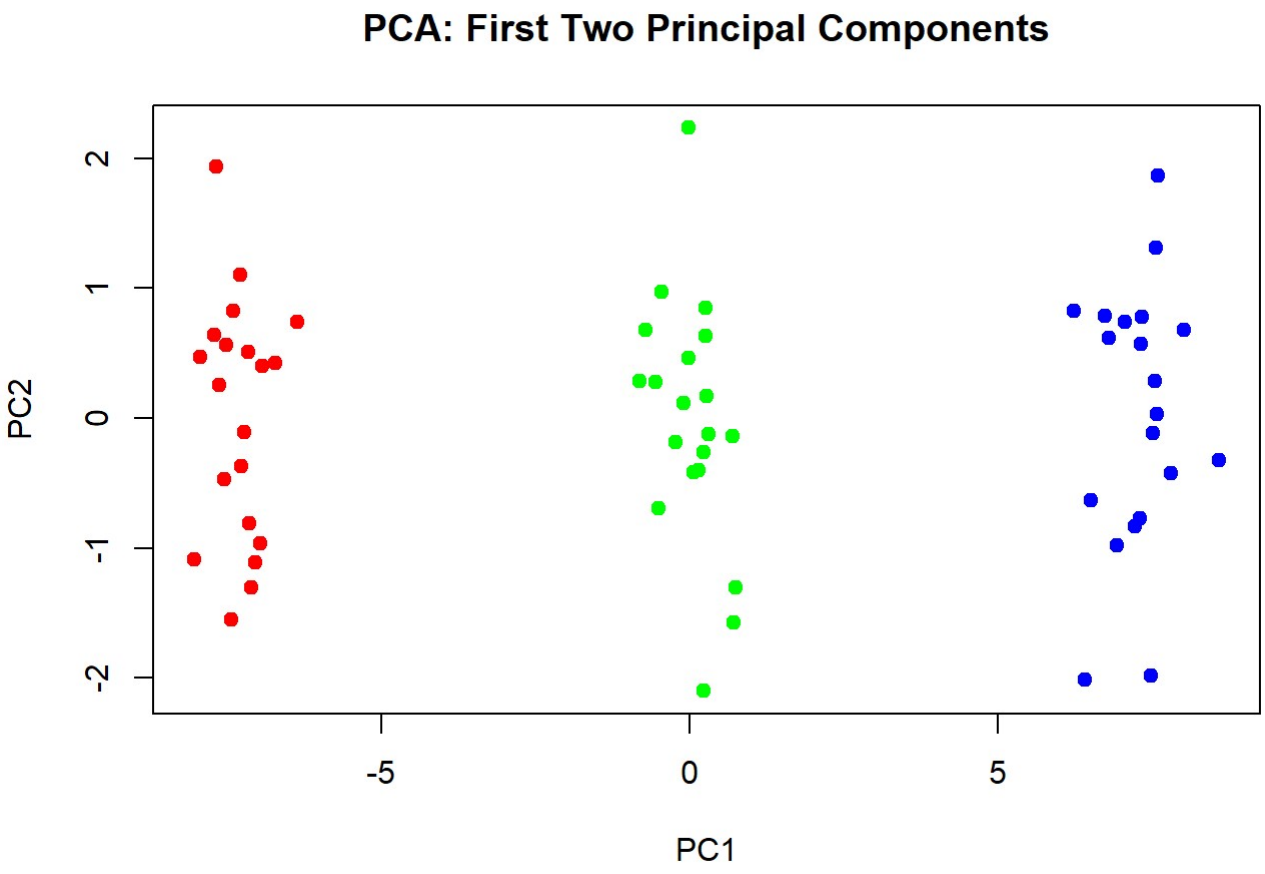
Chapter 12 Problem 10

a)

```
set.seed(1)              # for reproducibility

X <- matrix(rnorm(20*3*50), nrow = 60, ncol = 50) # 60x50 Matrix with random values from standard normal distribution.
X[1:20, ] <- X[1:20,] + 2   # class 1 centered around mean 2
X[21:40, ] <- X[21:40,] -2   # class 2 centered around mean -2
# class 3 left with mean 0
```

b)

```
# Performs PCA with scaling
pr.out <- prcomp(X, scale. = TRUE)

# Colors for each class
cls <- rep(c("red", "blue", "green"), each = 20)

# Plot first two principal components colored by true class
plot(pr.out$x[,1], pr.out$x[,2], col = cls, pch = 19,
     xlab = "PC1", ylab = "PC2",
     main = "PCA: First Two Principal Components")
```



c)

```
set.seed(1)
km.out <- kmeans(X, 3, nstart = 20)
table(Predicted = km.out$cluster, True = rep(1:3, each = 20))
```

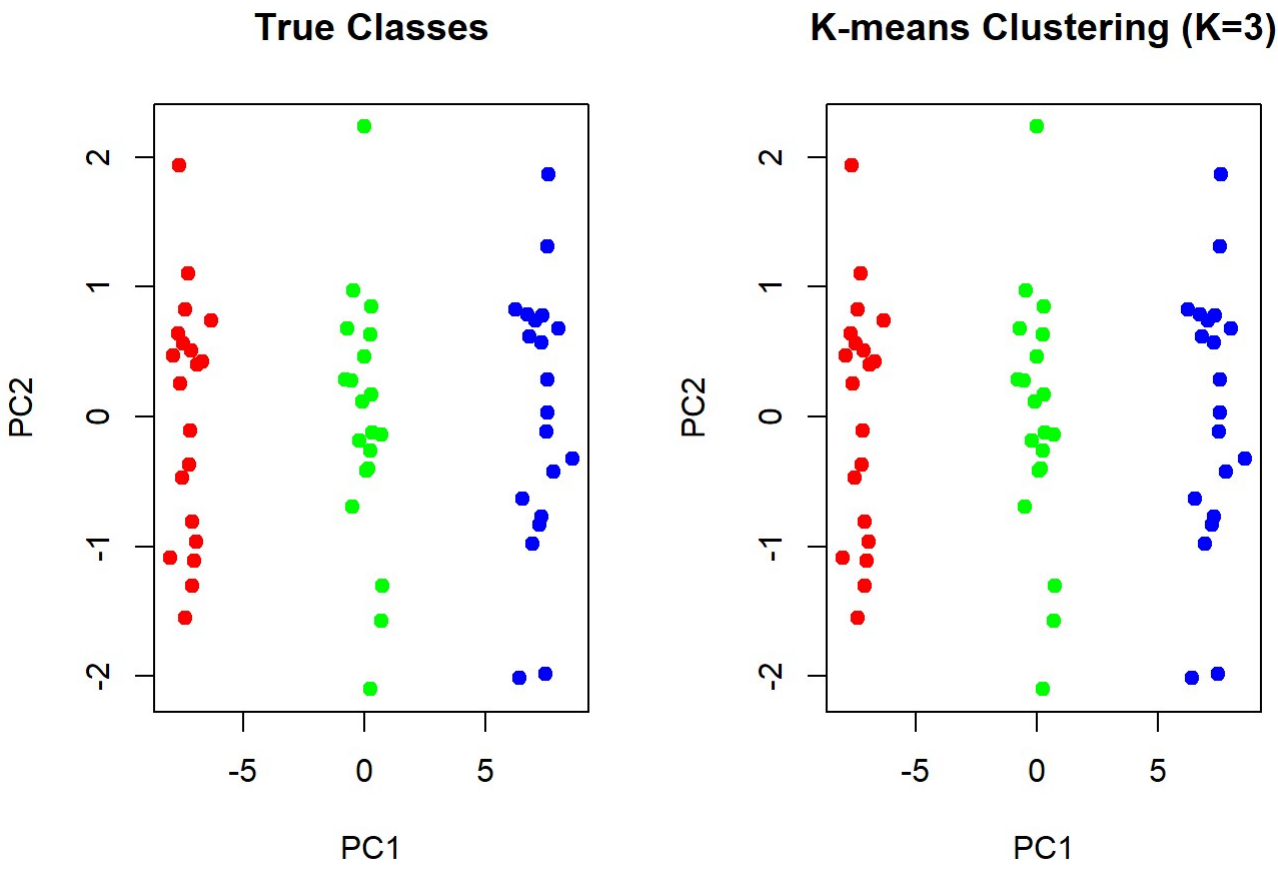
```
##          True
## Predicted  1  2  3
##         1  0  0 20
##         2 20  0  0
##         3  0 20  0
```

From the table we can see that the clustering correctly groups observations even though the cluster labels are not aligned with the class labels.

```
library("StratigrapheR")
# Shift cluster labels to align with class labels
km.out$cluster <- shift(km.out$cluster, n=20)
table(Predicted=km.out$cluster, True= rep(1:3, each=20))
```

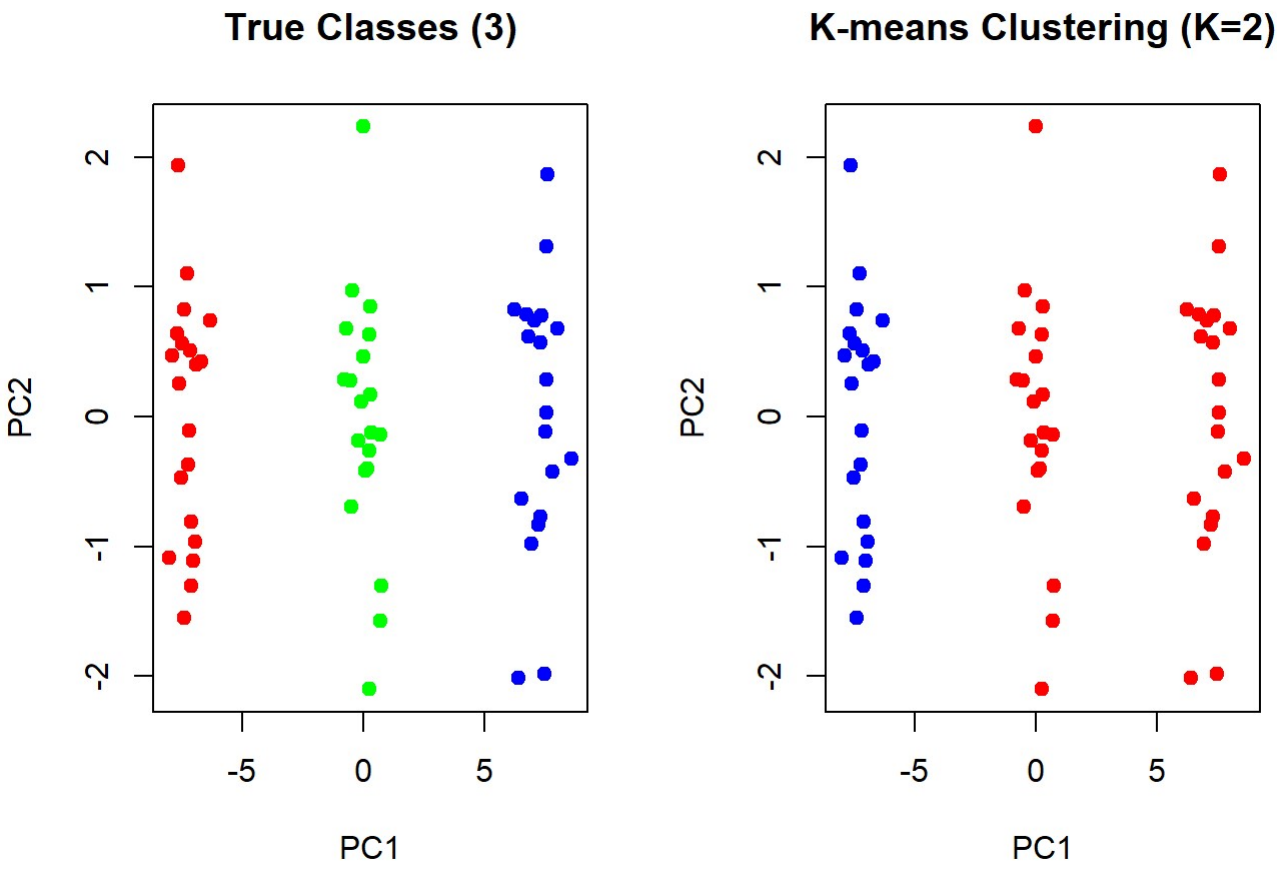
```
##          True
## Predicted  1  2  3
##         1 20  0  0
##         2  0 20  0
##         3  0  0 20
```

```
# True class vector and color function
class_vec <- rep(1:3, each = 20)
Cols <- function(x) c("red", "blue", "green")[x]

# Side-by-side plots for comparison
par(mfrow = c(1, 2))

# Left plot: True classes colored by actual labels
plot(pr.out$x[, 1:2],
     col = Cols(class_vec),
     pch = 19,
     xlab = "PC1", ylab = "PC2",
     main = "True Classes")

# Right plot: K-means clustering results colored by cluster assignment
plot(pr.out$x[, 1:2],
     col = Cols(km.out$cluster),
     pch = 19,
     xlab = "PC1", ylab = "PC2",
     main = "K-means Clustering (K=3)")
```



d)

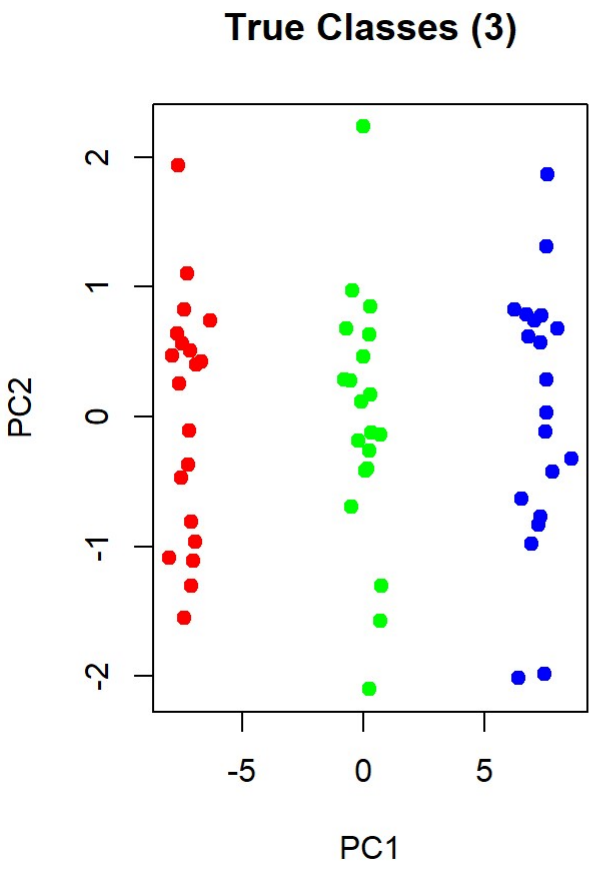
```
# Perform K-means clustering with K=2 (fewer than true classes)
set.seed(1)
km.out <- kmeans(X, 2, nstart = 20)

# Side-by-side plots
par(mfrow = c(1, 2))

# Left plot: True classes (3 classes)
plot(pr.out$x[, 1:2],
     col = Cols(class_vec),
     pch = 19,
     xlab = "PC1", ylab = "PC2",
     main = "True Classes (3)")

# Right plot: K-means with K=2 (clusters forced into 2 groups)
plot(pr.out$x[, 1:2],
     col = Cols(km.out$cluster),
     pch = 19,
     xlab = "PC1", ylab = "PC2",
     main = "K-means Clustering (K=2)")
```

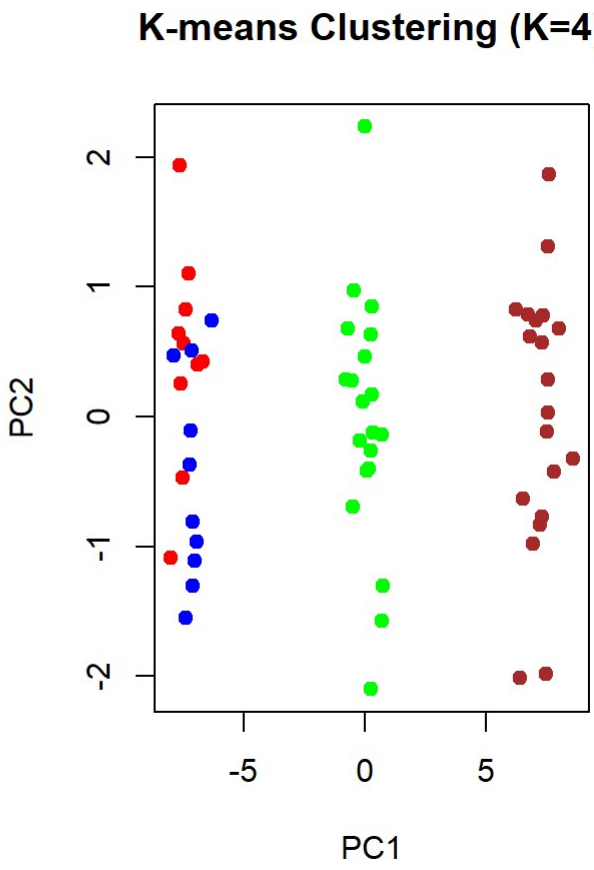


Here we can see that the K-means clustering grouped observations in the first class as one cluster and the last two classes as the second cluster.

e)

```
set.seed(1)
km.out <- kmeans(X, 4, nstart = 20)
Cols2 <- function(x)c("red", "blue", "green","brown")[x] # One color for each cluster label

# Side-by-side plots
par(mfrow = c(1, 2))

# Left plot: True classes (3 classes)
plot(pr.out$x[, 1:2],
     col = Cols(class_vec),
     pch = 19,
     xlab = "PC1", ylab = "PC2",
     main = "True Classes (3)")

# Right plot: K-means with K=4
plot(pr.out$x[, 1:2],
     col = Cols2(km.out$cluster),
     pch = 19,
     xlab = "PC1", ylab = "PC2",
     main = "K-means Clustering (K=4)")
```





Here we can see that the second and third classes were correctly grouped into own clusters and the first class was split between the remaining two clusters.