Assignment #4 - SS3850

Hussien Hussien

```
set.seed(1)
suppressMessages(library(MASS))
suppressMessages(library(ISLR))
suppressMessages(attach(Boston))
suppressMessages(attach(OJ))
suppressMessages(library(boot))
suppressMessages(library(splines))
suppressMessages(library(tree))
suppressMessages(library('topicmodels'))
suppressMessages(library(caret))
```

Chapter 7 Q9

This question uses the variables dis (the weighted mean of distances to five Boston employment centers) and nox (nitrogen oxides concen- tration in parts per 10 million) from the Boston data. We will treat `dis` as the predictor and `nox` as the response.

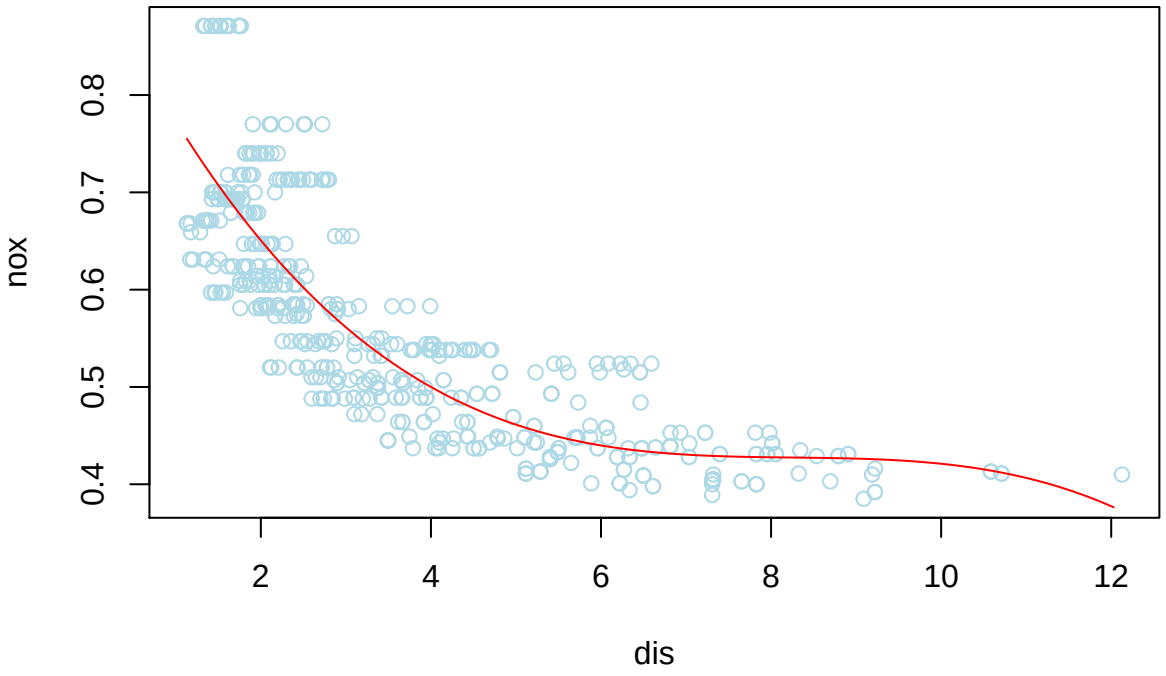
(a) Use the poly() function to fit a cubic polynomial regression to predict nox using dis. Report the regression output, and plot the resulting data and polynomial fits.

```
# TRain model and show regression output
linear.fit = lm(nox ~ poly(dis, 3), data = Boston)
summary(linear.fit)
```

```
##
## Call:
## lm(formula = nox ~ poly(dis, 3), 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, 3)1 -2.003096   0.062071 -32.271  < 2e-16 ***
## poly(dis, 3)2  0.856330   0.062071  13.796  < 2e-16 ***
## poly(dis, 3)3 -0.318049   0.062071  -5.124 4.27e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## 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
```

```
# Set X for plotting
dislim = range(dis)
dis.grid = seq(from = dislim[1], to = dislim[2], by = 0.1)

# Predict and plot predictions
```

```
linear.pred = predict(linear.fit, newdata=list(dis = dis.grid))
plot(nox ~ dis, data = Boston, col = "lightblue")
lines(dis.grid, linear.pred, col = "red", lwd = 1)
```

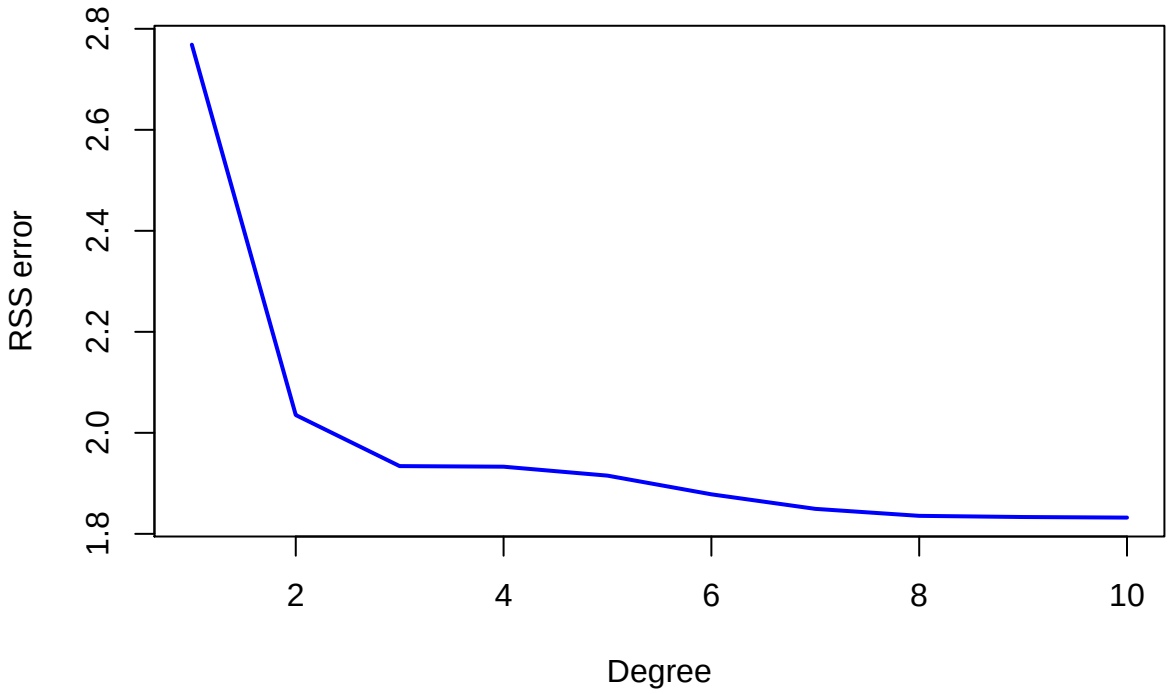


(b) Plot the polynomial fits for a range of different polynomial degrees (say, from 1 to 10), and report the associated residual sum of squares.

```
rss.error = rep(0, 10)
for (i in 1:10) {
    linear.fit = lm(nox ~ poly(dis, i), data = Boston)
    rss.error[i] = sum(linear.fit$residuals^2)
}
rss.error
```

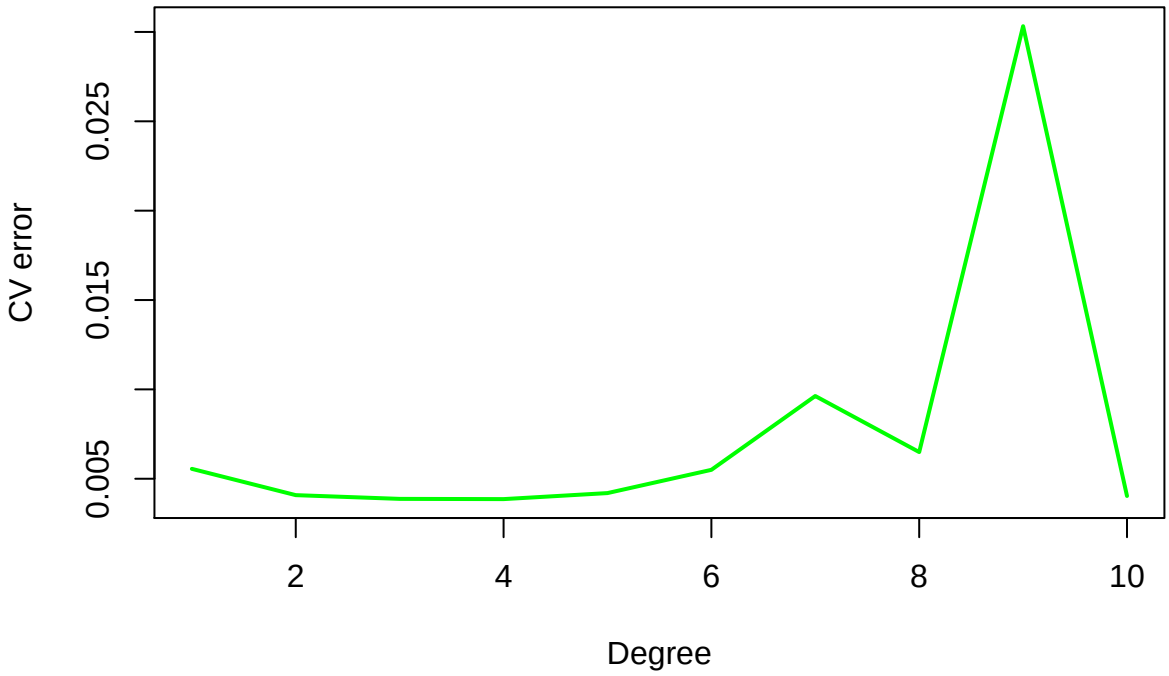
```
##  [1] 2.768563 2.035262 1.934107 1.932981 1.915290 1.878257 1.849484
##  [8] 1.835630 1.833331 1.832171
```

```
plot(1:10, rss.error, xlab = "Degree", ylab = "RSS error", type = "l", pch = 20,
    lwd = 2, col = "blue")
```



(c) Perform cross-validation or another approach to select the optimal degree for the polynomial, and explain your results.

```
cv.error = rep(0, 10)
for (i in 1:10) {
    glm.fit = glm(nox ~ poly(dis, i), data = Boston)
    cv.error[i] = cv.glm(Boston, glm.fit, K = 10)$delta[2]
}
plot(1:10, cv.error, xlab = "Degree", ylab = "CV error", type = "l", pch = 20,
    lwd = 2,col = 'green')
```



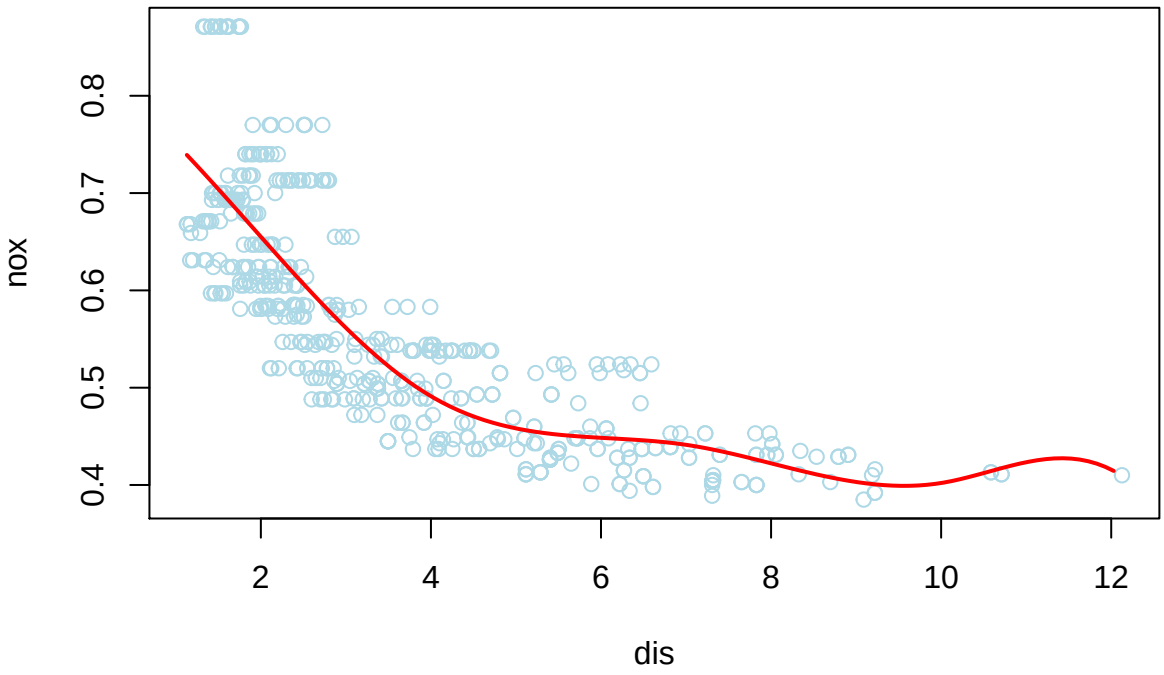
Three looks like the optimal polynomial degree.

(d) Use the bs() function to fit a regression spline to predict nox using dis. Report the output for the fit using four degrees of freedom. How did you choose the knots? Plot the resulting fit.

```
# Fit regression spline and output summary.. Since dis has a range of about 1-13 we'll try to pick spli
spline.fit = lm(nox ~ bs(dis, df = 4, knots = c(4, 7, 10)), data = Boston)
summary(spline.fit)
```

```
##
## Call:
## lm(formula = nox ~ bs(dis, df = 4, knots = c(4, 7, 10)), data = Boston)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.124592 -0.040324 -0.008719  0.024749  0.192906
##
## Coefficients:
##                                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)                          0.73918    0.01332  55.510  < 2e-16
## bs(dis, df = 4, knots = c(4, 7, 10))1 -0.08839    0.02506  -3.526  0.00046
## bs(dis, df = 4, knots = c(4, 7, 10))2 -0.31363    0.01684 -18.625  < 2e-16
## bs(dis, df = 4, knots = c(4, 7, 10))3 -0.27037    0.02791  -9.686  < 2e-16
## bs(dis, df = 4, knots = c(4, 7, 10))4 -0.37989    0.03801  -9.995  < 2e-16
## bs(dis, df = 4, knots = c(4, 7, 10))5 -0.28983    0.06615  -4.381 1.44e-05
## bs(dis, df = 4, knots = c(4, 7, 10))6 -0.32971    0.06324  -5.214 2.71e-07
##
## (Intercept)                          ***
## bs(dis, df = 4, knots = c(4, 7, 10))1 ***
## bs(dis, df = 4, knots = c(4, 7, 10))2 ***
## bs(dis, df = 4, knots = c(4, 7, 10))3 ***
## bs(dis, df = 4, knots = c(4, 7, 10))4 ***
## bs(dis, df = 4, knots = c(4, 7, 10))5 ***
## bs(dis, df = 4, knots = c(4, 7, 10))6 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.06185 on 499 degrees of freedom
## Multiple R-squared:  0.7185, Adjusted R-squared:  0.7151
## F-statistic: 212.3 on 6 and 499 DF,  p-value: < 2.2e-16
```

```
# Predict using regression spline and plot over data
spline.pred = predict(spline.fit, newdata=list(dis = dis.grid))
plot(nox ~ dis, data = Boston, col = "lightblue")
lines(dis.grid, spline.pred, col = "red", lwd = 2)
```

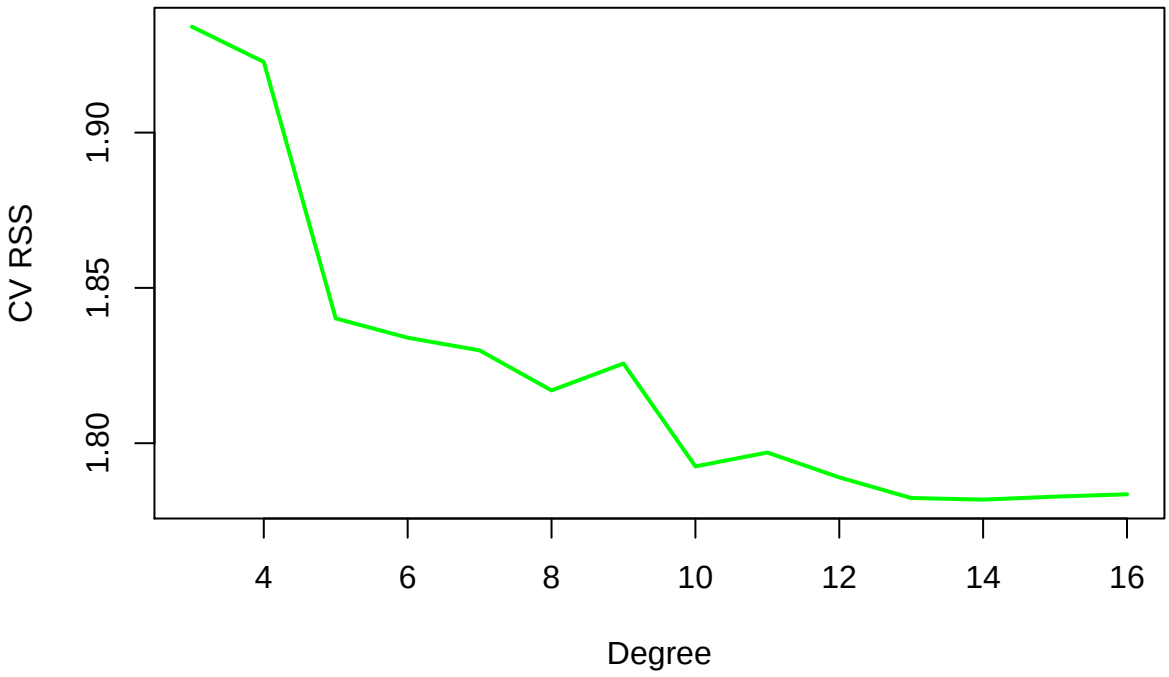


(e) Now fit a regression spline for a range of degrees of freedom, and plot the resulting fits and report the resulting RSS. Describe the results obtained.

```
spline.rss = rep(0, 16)
for (i in 3:16) {
    spline.fit = lm(nox ~ bs(dis, df = i), data = Boston)
    spline.rss[i] = sum(spline.fit$residuals^2)
}
spline.rss[-c(1, 2)]
```

```
##  [1] 1.934107 1.922775 1.840173 1.833966 1.829884 1.816995 1.825653
##  [8] 1.792535 1.796992 1.788999 1.782350 1.781838 1.782798 1.783546
```

```
plot(3:16, spline.rss[-c(1, 2)], xlab = "Degree", ylab = "CV RSS", type = "l", pch = 20,
    lwd = 2,col = 'green')
```



It looks like the results keep getting better until we hit degree=13 then experience a gradualy increase in RSS... It appears that 13 has the best training RSS but probably overfit.

(f) Perform cross-validation or another approach in order to select the best degrees of freedom for a regression spline on this data. Describe your results.

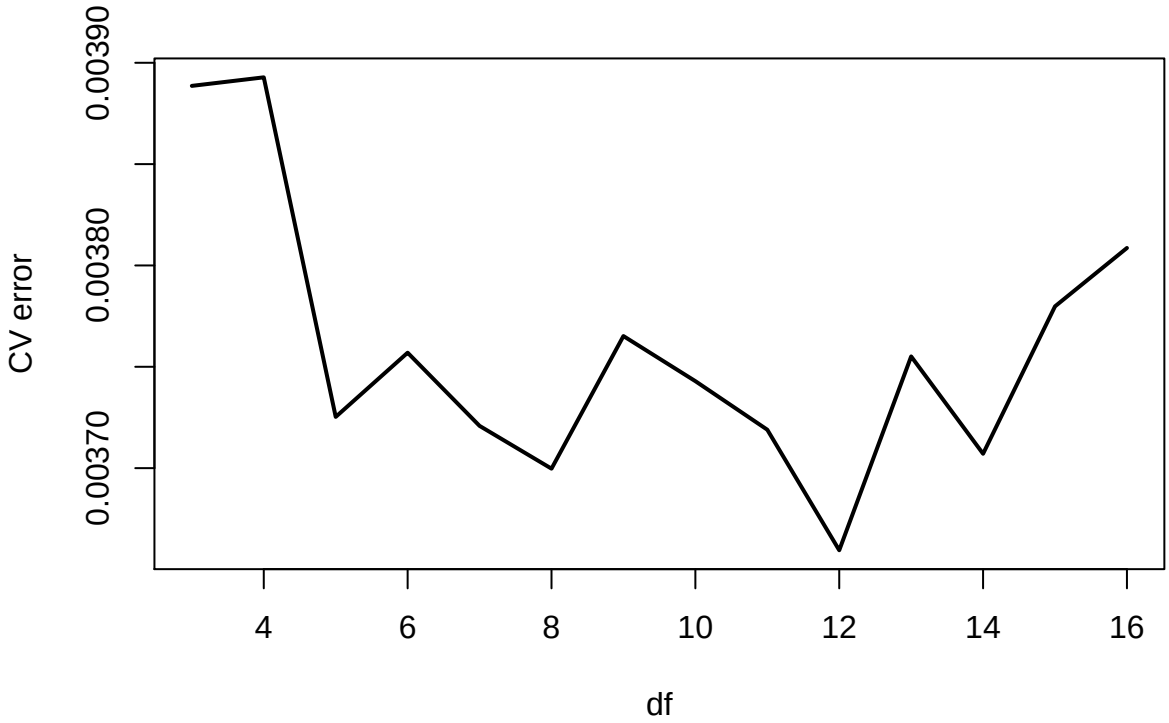
```
spline.cverror = rep(0, 16)
for (i in 3:16) {
    spline.fit = glm(nox ~ bs(dis, df = i), data = Boston)
    spline.cverror[i] = cv.glm(Boston, spline.fit, K = 10)$delta[1]
}
```

```
## Warning in bs(dis, degree = 3L, knots = numeric(0), Boundary.knots =
## c(1.1296, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = numeric(0), Boundary.knots =
## c(1.1296, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = numeric(0), Boundary.knots =
## c(1.137, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = numeric(0), Boundary.knots =
## c(1.137, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`50%` = 3.0992), Boundary.knots =
## c(1.137, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`50%` = 3.0992), Boundary.knots =
## c(1.137, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(`50%` = 3.2157), Boundary.knots =
## c(1.1296, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`50%` = 3.2157), Boundary.knots =
## c(1.1296, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`33.33333%` = 2.354, `66.66667%`
## = 4.2474: some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`33.33333%` = 2.354, `66.66667%`
## = 4.2474: some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`33.33333%` = 2.4212, `66.66667%`
## = 4.38856666666667: some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`33.33333%` = 2.4212, `66.66667%`
## = 4.38856666666667: some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`25%` = 2.1105, `50%` = 3.2721, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`25%` = 2.1105, `50%` = 3.2721, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`25%` = 2.1, `50%` = 3.1323,
## `75%` = 5.118: some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`25%` = 2.1, `50%` = 3.1323,
## `75%` = 5.118: some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`20%` = 1.92938, `40%` =
## 2.55946, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`20%` = 1.92938, `40%` =
## 2.55946, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`20%` = 1.93736, `40%` =
## 2.59666, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`20%` = 1.93736, `40%` =
## 2.59666, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`16.66667%` = 1.86156666666667, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(`16.66667%` = 1.86156666666667, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`14.28571%` = 1.79777142857143, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`14.28571%` = 1.79777142857143, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`14.28571%` = 1.7936, `28.57143%`
## = 2.16771428571429, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`14.28571%` = 1.7936, `28.57143%`
## = 2.16771428571429, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`12.5%` = 1.734325, `25%` =
## 2.0941, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`12.5%` = 1.734325, `25%` =
## 2.0941, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`12.5%` = 1.751575, `25%` =
## 2.1084, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`12.5%` = 1.751575, `25%` =
## 2.1084, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`11.11111%` = 1.71552222222222, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`11.11111%` = 1.71552222222222, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`11.11111%` = 1.66286666666667, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`11.11111%` = 1.66286666666667, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`10%` = 1.62008, `20%` =
## 1.92938, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`10%` = 1.62008, `20%` =
## 1.92938, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`10%` = 1.6283, `20%` = 1.9512, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`10%` = 1.6283, `20%` = 1.9512, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(dis, degree = 3L, knots = c(`9.090909%` = 1.61225454545455, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`9.090909%` = 1.61225454545455, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`9.090909%` = 1.61066363636364, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`9.090909%` = 1.61066363636364, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`8.333333%` = 1.60476666666667, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`8.333333%` = 1.60476666666667, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`8.333333%` = 1.5881, `16.66667%`
## = 1.82231666666667, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`8.333333%` = 1.5881, `16.66667%`
## = 1.82231666666667, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`7.692308%` = 1.58949230769231, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`7.692308%` = 1.58949230769231, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`7.692308%` = 1.5741, `15.38462%`
## = 1.8209, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`7.692308%` = 1.5741, `15.38462%`
## = 1.8209, : some 'x' values beyond boundary knots may cause ill-conditioned
## bases

## Warning in bs(dis, degree = 3L, knots = c(`7.142857%` = 1.54201428571429, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`7.142857%` = 1.54201428571429, :
## some 'x' values beyond boundary knots may cause ill-conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`7.142857%` = 1.5768, `14.28571%`
## = 1.81652857142857, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases

## Warning in bs(dis, degree = 3L, knots = c(`7.142857%` = 1.5768, `14.28571%`
## = 1.81652857142857, : some 'x' values beyond boundary knots may cause ill-
## conditioned bases
plot(3:16, spline.cverror[-c(1, 2)], lwd = 2, type = "l", xlab = "df", ylab = "CV error")
```



The data looks alot less monotonic than the above CV graphs. It does appear that there is a clear minimum at DF=10.

Chapter 8 Q9

This problem involves the OJ data set which is part of the ISLR package. (a) Create a training set containing a random sample of 800 obser- vations, and a test set containing the remaining observations.

```
set.seed(1013)

train = sample(dim(OJ)[1], 800)
OJ.train = OJ[train, ]
OJ.test = OJ[-train, ]
```

(b) Fit a tree to the training data, with Purchase as the response and the other variables as predictors. Use the summary() function to produce summary statistics about the tree, and describe the results obtained. What is the training error rate? How many terminal nodes does the tree have?

```
tree.model = tree(Purchase ~ ., data = OJ.train)
summary(tree.model)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = OJ.train)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "PriceDiff"     "ListPriceDiff" "SalePriceMM"
## Number of terminal nodes:  7
## Residual mean deviance:  0.7564 = 599.8 / 793
## Misclassification error rate: 0.1612 = 129 / 800
```

It has 7 terminal nodes. Training error rate (misclassification error) for the tree is 0.1612.

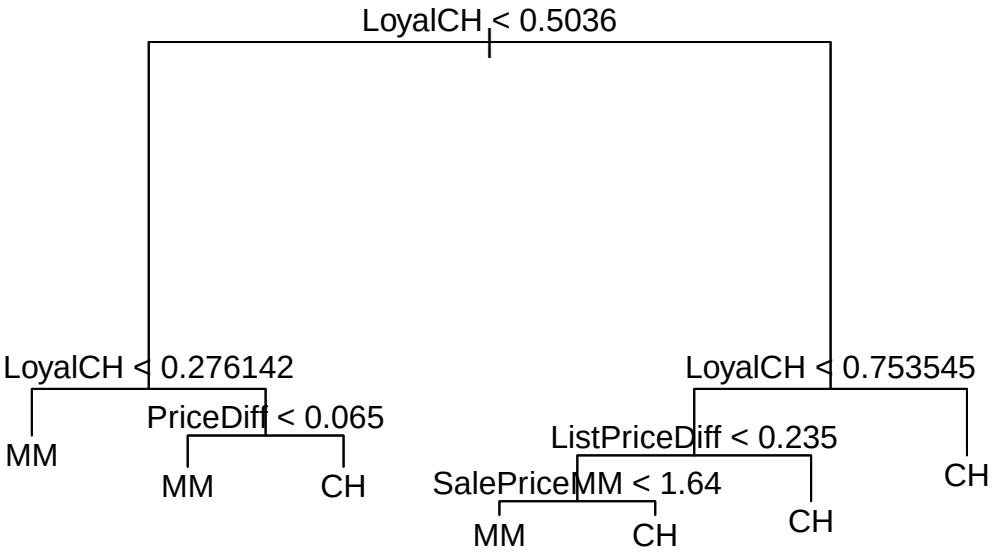
(c) Type in the name of the tree object in order to get a detailed text output. Pick one of the terminal nodes, and interpret the information displayed.

```
tree.model
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1069.00 CH ( 0.61125 0.38875 )
##    2) LoyalCH < 0.5036 344  407.30 MM ( 0.27907 0.72093 )
##      4) LoyalCH < 0.276142 163  121.40 MM ( 0.12270 0.87730 ) *
##      5) LoyalCH > 0.276142 181  246.30 MM ( 0.41989 0.58011 )
##       10) PriceDiff < 0.065 75   75.06 MM ( 0.20000 0.80000 ) *
##       11) PriceDiff > 0.065 106  144.50 CH ( 0.57547 0.42453 ) *
##    3) LoyalCH > 0.5036 456  366.30 CH ( 0.86184 0.13816 )
##      6) LoyalCH < 0.753545 189  224.30 CH ( 0.71958 0.28042 )
##       12) ListPriceDiff < 0.235 79  109.40 MM ( 0.48101 0.51899 )
##         24) SalePriceMM < 1.64 22   20.86 MM ( 0.18182 0.81818 ) *
##         25) SalePriceMM > 1.64 57   76.88 CH ( 0.59649 0.40351 ) *
##       13) ListPriceDiff > 0.235 110   75.81 CH ( 0.89091 0.10909 ) *
##      7) LoyalCH > 0.753545 267   85.31 CH ( 0.96255 0.03745 ) *
```

(d) Create a plot of the tree, and interpret the results.

```
plot(tree.model); text(tree.model)
```



Since LoyalCH splits the tree the most, we can say that it is the most important feature. This alone could perform a classification but in the case that it doesn't we observe and split at either PriceDiff or ListPriceDiff.. If ListPriceDiff is below 0.235 then we consult SalePriceMM.

(e) Predict the response on the test data, and produce a confusion matrix comparing the test labels to the predicted test labels. What is the test error rate?

```
tree.pred = predict(tree.model, OJ.test, type = "class")

confusionMatrix(tree.pred, reference = OJ.test$Purchase)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  CH  MM
##         CH 149  30
```

```
##         MM  15  76
##
##                Accuracy : 0.8333
##                  95% CI : (0.7834, 0.8758)
##     No Information Rate : 0.6074
##     P-Value [Acc > NIR] : 6.513e-16
##
##                   Kappa : 0.6416
##
##  Mcnemar's Test P-Value : 0.03689
##
##             Sensitivity : 0.9085
##             Specificity : 0.7170
##          Pos Pred Value : 0.8324
##          Neg Pred Value : 0.8352
##              Prevalence : 0.6074
##          Detection Rate : 0.5519
##    Detection Prevalence : 0.6630
##       Balanced Accuracy : 0.8128
##
##        'Positive' Class : CH
##
```

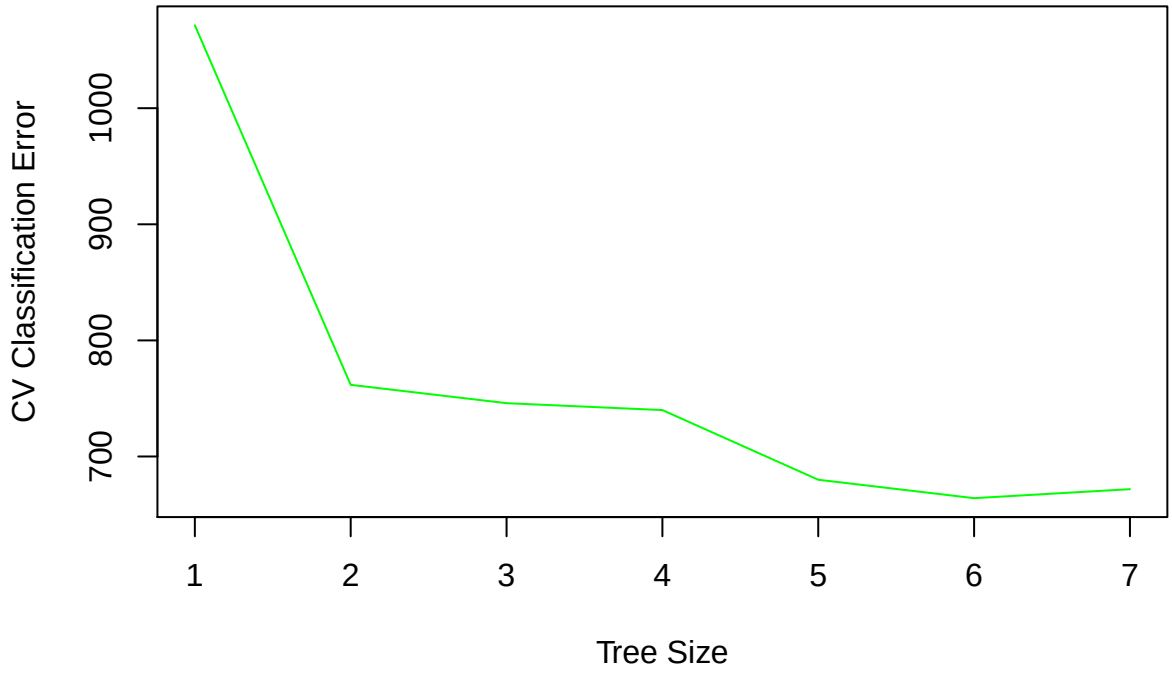
Error rate (1-accuracy): 1 - 0.8333 = 0.167

(f) Apply the cv.tree() function to the training set in order to determine the optimal tree size.

```
tree.cv = cv.tree(tree.model, FUN = prune.tree)
```

(g) Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis.

```
plot(tree.cv$size, tree.cv$dev, type = "l", xlab = "Tree Size", ylab = "CV Classification Error",col='g
```



(h) Which tree size corresponds to the lowest cross-validated classi- fication error rate?

It looks like this tree reaches a low point in CV classification error at a tree size of 6.

(i) Produce a pruned tree corresponding to the optimal tree size obtained using cross-validation. If cross-validation does not lead to selection of a pruned tree, then create a pruned tree with five terminal nodes.

```
tree.pruned = prune.tree(tree.model, best = 6)
```

(j) Compare the training error rates between the pruned and unpruned trees. Which is higher?

```
summary(tree.pruned)
```

```
##
## Classification tree:
## snip.tree(tree = tree.model, nodes = 12L)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "PriceDiff"     "ListPriceDiff"
## Number of terminal nodes:  6
## Residual mean deviance:  0.7701 = 611.5 / 794
## Misclassification error rate: 0.175 = 140 / 800
```

The training error rate of the non-pruned tree: 0.1612 The training error rate of the pruned tree: 0.175 Difference: 0.0138 increase in training error rate

(k) Compare the test error rates between the pruned and unpruned trees. Which is higher?

```
unpruned.prediction = predict(tree.model, OJ.test, type = "class") #Make Prediction
unpruned.errors = sum(OJ.test$Purchase != unpruned.prediction) # Count Errors
print("Unpruned Test Error Rate")
```

```
## [1] "Unpruned Test Error Rate"
```

```
unpruned.errors/length(unpruned.prediction) # Compute test error rate
```

```
## [1] 0.1666667
```

```
pruned.prediction = predict(tree.pruned, OJ.test, type = "class") #Make Prediction
pruned.errors = sum(OJ.test$Purchase != pruned.prediction) # Count Errors
print("Pruned Test Error Rate")
```

```
## [1] "Pruned Test Error Rate"
```

```
pruned.errors/length(pruned.prediction) # Compute
```

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

The Pruned Tree has a higher test and training error rate.

Chapter 10

Conduct K-mean and Hierarchical clustering on the wine data set (attached) or any other dataset you like.

```
library(tidyverse)
```

```
## Registered S3 method overwritten by 'cli':
##   method     from
##   print.tree tree
```

```
## -- Attaching packages ---------------------------------------------------- tidyverse 1.2.1 --
```

```
## v tibble  2.1.3     v purrr   0.3.2
```

```
## v tidyr   1.0.0     v dplyr   0.8.5
## v readr   1.3.1     v stringr 1.4.0
## v tibble  2.1.3     v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x purrr::lift()   masks caret::lift()
## x dplyr::select() masks MASS::select()
```

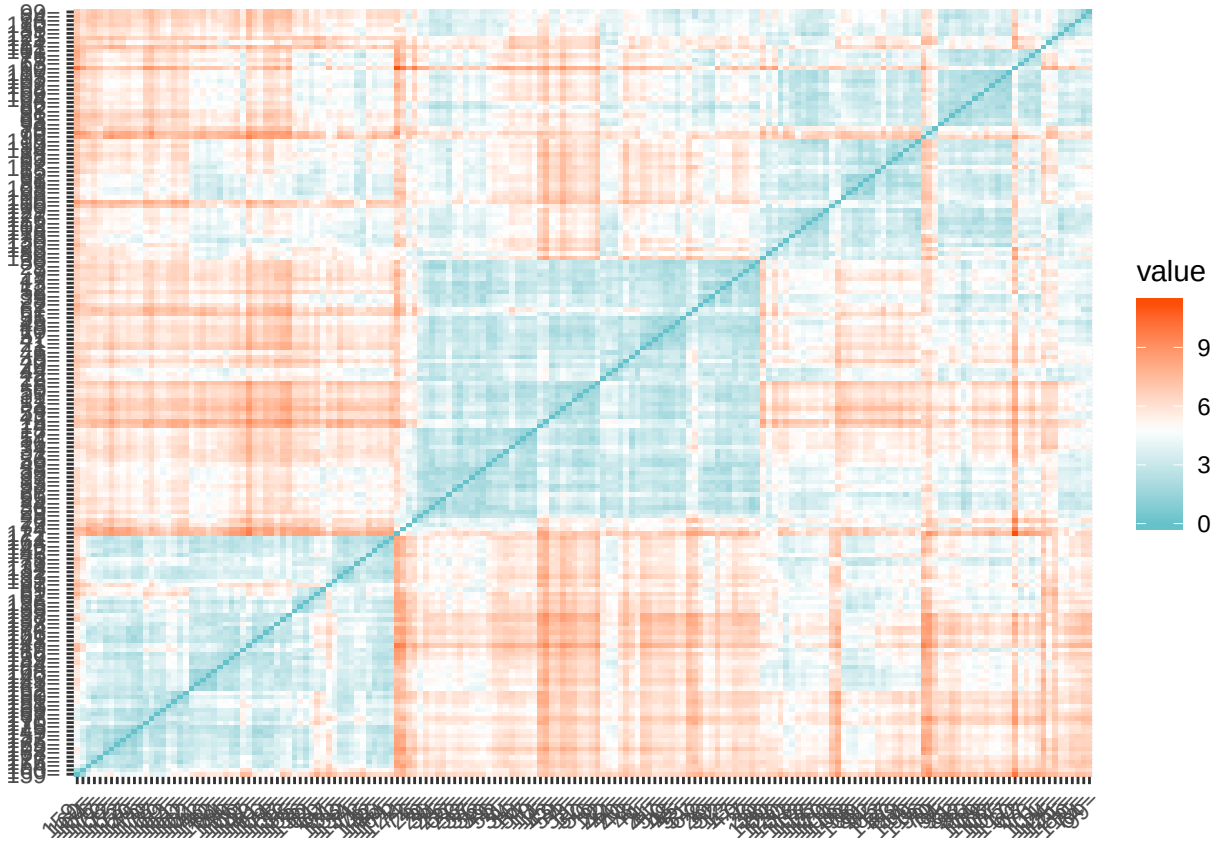
```
library(cluster)    # clustering algorithms
library(factoextra) # clustering algorithms & visualization
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
library(MASS)
```

##K-mean

```
wine <- read.delim("wine.txt", sep = ",", dec = ".", header = FALSE) #Read Data
wine.scale<-scale(wine[,-1]) #Scale Data

distance <- get_dist(wine.scale) #Get Distances

fviz_dist(distance, gradient = list(low = "#00AFBB", mid = "white", high = "#FC4E07"))
```




```
k3 <- kmeans(wine.scale, centers = 3, nstart = 25)
str(k3)
```

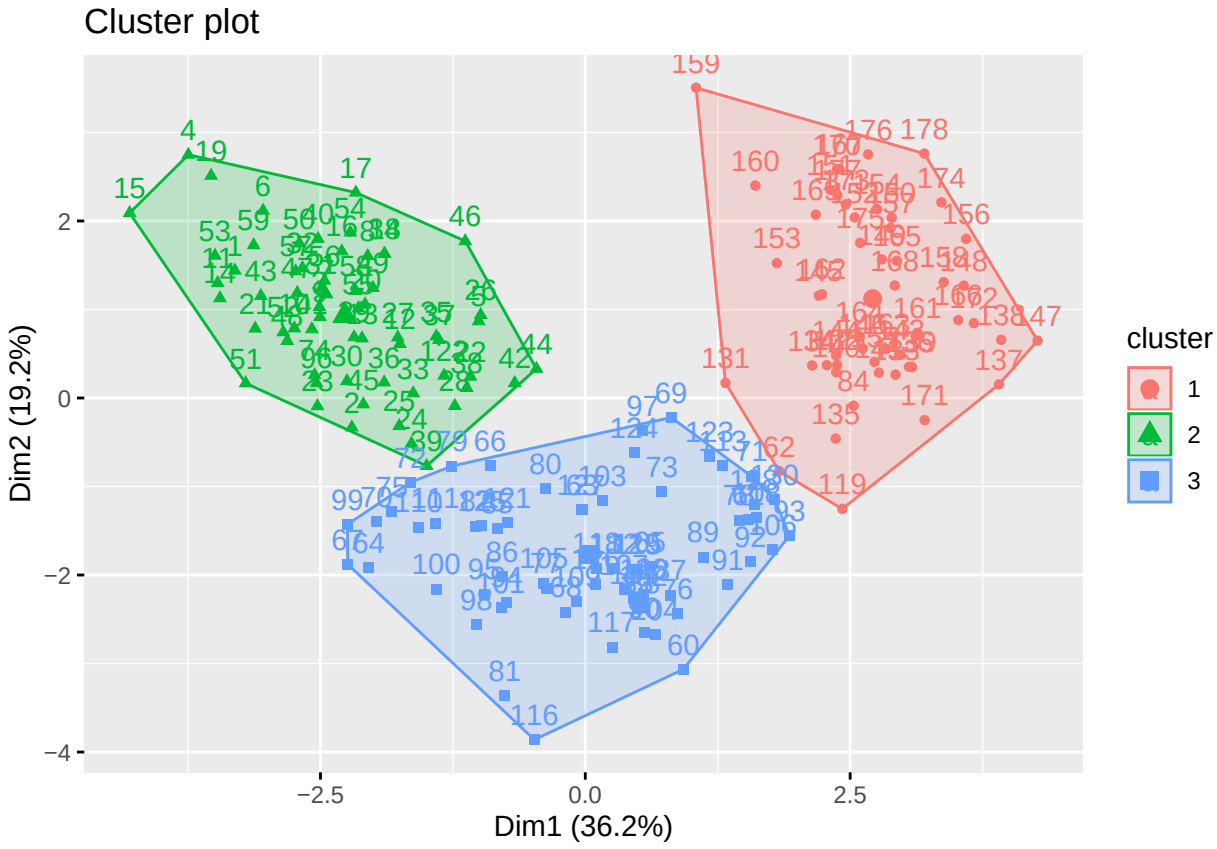
```
## List of 9
```

```
##  $ cluster     : int [1:178] 2 2 2 2 2 2 2 2 2 2 ...
##  $ centers     : num [1:3, 1:13] 0.164 0.833 -0.923 0.869 -0.303 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:3] "1" "2" "3"
##   .. ..$ : chr [1:13] "V2" "V3" "V4" "V5" ...
##  $ totss       : num 2301
##  $ withinss    : num [1:3] 326 386 559
##  $ tot.withinss: num 1271
##  $ betweenss   : num 1030
##  $ size        : int [1:3] 51 62 65
##  $ iter        : int 2
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"
```

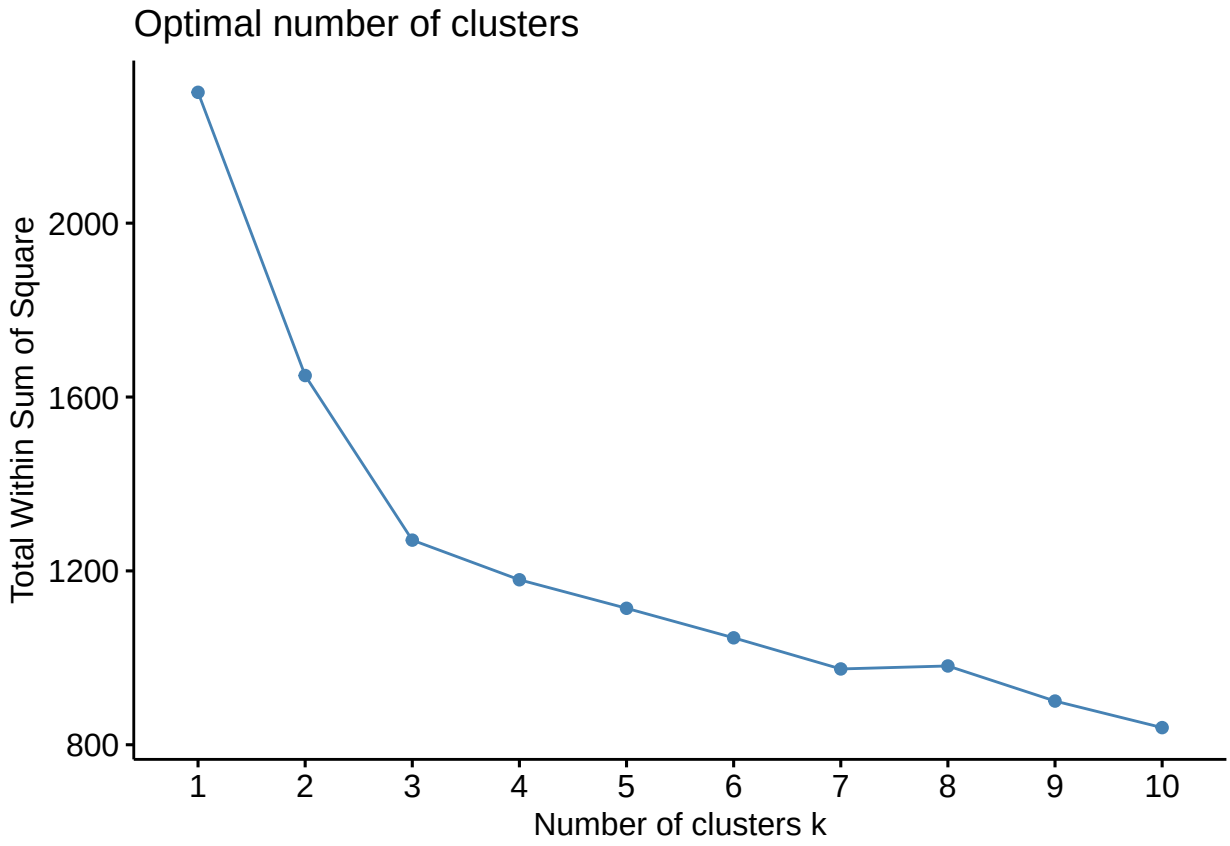
```
k3$cluster
```

```
##   [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [36] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 1 3 3 3 3 3 3 3 3
##  [71] 3 3 3 2 3 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3
## [106] 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 3 2 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1
## [141] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [176] 1 1 1
```

```
fviz_cluster(k3, data = wine.scale)
```

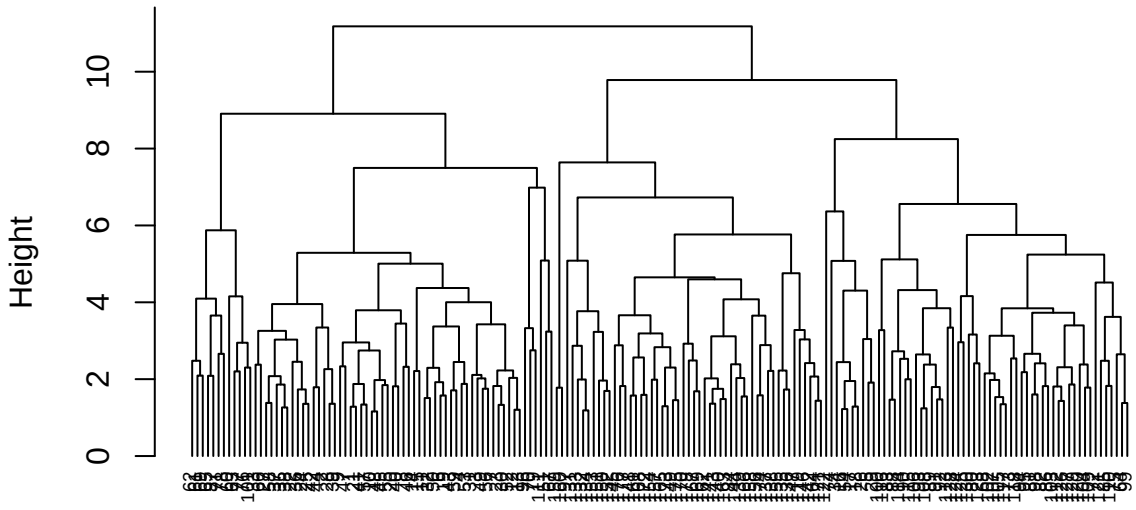



```
fviz_nbclust(wine.scale, kmeans, method = "wss")
```



##Hierarchical Clustering

```
wine.scale<-scale(wine[,-1]) #Scale Data
d <- dist(wine.scale, method = "euclidean") #  Compute Dissimilarity matrix


hc1 <- hclust(d, method = "complete" ) # Use Dissimilarity Matric to Hierarchicaly cluster using Comple


plot(hc1, cex = 0.6, hang = -1)# Plot the obtained dendrogram
```



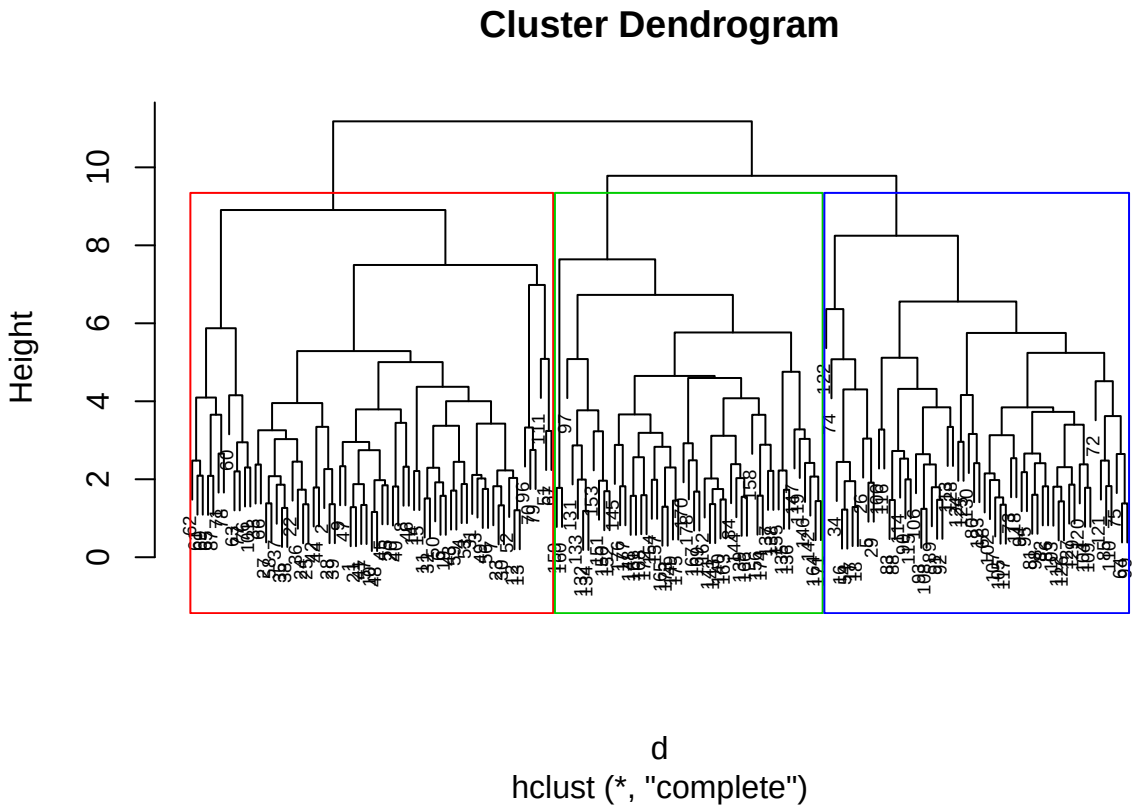

```
# Compute with agnes
hc2 <- agnes(wine.scale, method = "complete")

# Agglomerative coefficient
hc2$ac
```

```
## [1] 0.815931
```

```
##clustering with Dendrograms

sub_grp <- cutree(hc1, k = 3)
table(sub_grp)
```

```
## sub_grp
##  1  2  3
## 69 58 51
```

```
plot(hc1, cex = 0.6)
rect.hclust(hc1, k = 3, border = 2:4)
```




```
fviz_cluster(list(data = wine.scale, cluster = sub_grp))
```