DA5030.P3.Parpattedar

Shruti Parpattedar

April 4, 2019

Problem 1

Question 1

Loading the bank dataset.

```
bank <- read.csv("bank.csv", sep = ";")
```

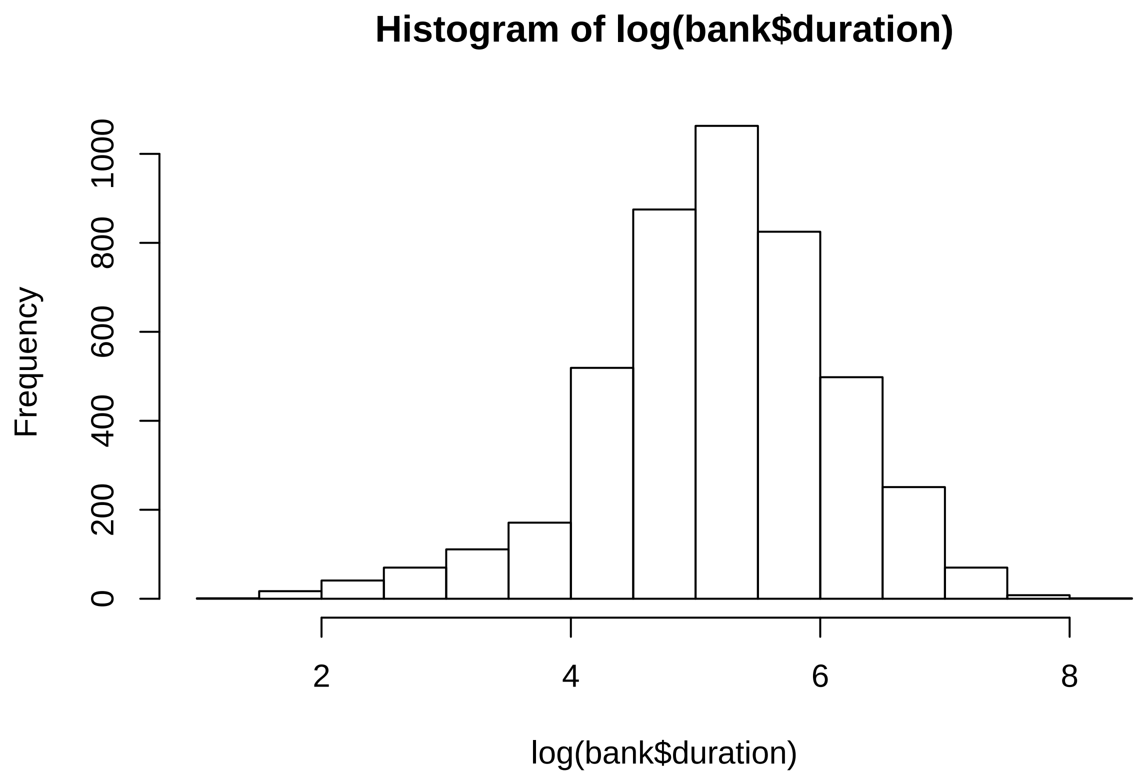
Question 2

Exploring the dataset to check for distributional skew.

```
library(psych)
str(bank)
```

```
## 'data.frame':    4521 obs. of  17 variables:
##  $ age      : int  30 33 35 30 59 35 36 39 41 43 ...
##  $ job      : Factor w/ 12 levels "admin.","blue-collar",..: 11 8 5 5 2 5 7 10 3 8 ...
##  $ marital  : Factor w/ 3 levels "divorced","married",..: 2 2 3 2 2 3 2 2 2 2 ...
##  $ education: Factor w/ 4 levels "primary","secondary",..: 1 2 3 3 2 3 3 2 3 1 ...
##  $ default  : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ balance  : int  1787 4789 1350 1476 0 747 307 147 221 -88 ...
##  $ housing  : Factor w/ 2 levels "no","yes": 1 2 2 2 2 1 2 2 2 2 ...
##  $ loan     : Factor w/ 2 levels "no","yes": 1 2 1 2 1 1 1 1 1 2 ...
##  $ contact  : Factor w/ 3 levels "cellular","telephone",..: 1 1 1 3 3 1 1 1 3 1 ...
##  $ day      : int  19 11 16 3 5 23 14 6 14 17 ...
##  $ month    : Factor w/ 12 levels "apr","aug","dec",..: 11 9 1 7 9 4 9 9 9 1 ...
##  $ duration : int  79 220 185 199 226 141 341 151 57 313 ...
##  $ campaign : int  1 1 1 4 1 2 1 2 2 1 ...
##  $ pdays    : int  -1 339 330 -1 -1 176 330 -1 -1 147 ...
##  $ previous : int  0 4 1 0 0 3 2 0 0 2 ...
##  $ poutcome : Factor w/ 4 levels "failure","other",..: 4 1 1 4 4 1 2 4 4 1 ...
##  $ y        : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
```

```
# pairs.panels(Q1_data[,1:8])
# pairs.panels(data_Q1[,9:17])
hist(log(bank$duration))
```



Question 3

Building a classification model svm from the caret package.

The confusion matrix shows approximately 89% accuracy in the model.

```r
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2

##
## Attaching package: 'ggplot2'

## The following objects are masked from 'package:psych':
##
##     %+%, alpha
```

```r
bank$y <- as.factor(bank$y)
levels(bank$y) <- make.names(levels(bank$y))

set.seed(1)
# Creating a partition such that 2/3rd data is used in training the model and the
# remaining in testing it.
partition <- createDataPartition(y = bank$y, p = 0.66, list = FALSE)
train <- bank[partition,]
test <- bank[-partition,]
```

```r
# Checking if the proportion of outcomes is maintained in the subsets as well.
prop.table(table(bank$y))
```

```
##
##      no     yes
## 0.88476 0.11524
```

```r
prop.table(table(test$y))
```

```
##
##        no       yes
## 0.8848406 0.1151594
```

```r
prop.table(table(train$y))
```

```
##
##        no       yes
## 0.8847185 0.1152815
```

```r
set.seed(1492)
# Using the trainControl function to control the computational nuances of the
# train function
ctrl <- trainControl(method = "cv",
                     n = 3,
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE)

svm_grid <- expand.grid(sigma = c(0.01, 0.05),
                        C = c(0.75, 1, 1.25))

# Using the train function from the caret package with the svmRadial method
model_svm <- train(y ~ .,
              data = train,
              method = "svmRadial",
              preProc = c("center","scale"),
              metric = "ROC",
              trControl = ctrl,
              tuneGrid = svm_grid)
model_svm
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 2984 samples
##   16 predictor
##    2 classes: 'no', 'yes'
##
## Pre-processing: centered (42), scaled (42)
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 1990, 1989, 1989
## Resampling results across tuning parameters:
##
##   sigma  C     ROC        Sens       Spec
##   0.01   0.75  0.8754066  0.9844697  0.1888889
##   0.01   1.00  0.8755390  0.9825758  0.2063056
##   0.01   1.25  0.8754429  0.9799242  0.2151030
##   0.05   0.75  0.8359439  0.9795455  0.1626748
##   0.05   1.00  0.8358382  0.9791667  0.1655988
##   0.05   1.25  0.8353956  0.9776515  0.1742690
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.01 and C = 1.
```

```
# Testing the model on the test subset
svm_pred <- predict(model_svm, test)

# Displaying the accuracy of the model
confusionMatrix(svm_pred, test$y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1336  135
##        yes   24   42
##
##                Accuracy : 0.8966
##                  95% CI : (0.8802, 0.9113)
##     No Information Rate : 0.8848
##     P-Value [Acc > NIR] : 0.07944
##
##                   Kappa : 0.302
##
##  Mcnemar's Test P-Value : < 2e-16
##
##             Sensitivity : 0.9824
##             Specificity : 0.2373
##          Pos Pred Value : 0.9082
##          Neg Pred Value : 0.6364
##              Prevalence : 0.8848
##          Detection Rate : 0.8692
##    Detection Prevalence : 0.9571
##       Balanced Accuracy : 0.6098
##
##        'Positive' Class : no
##
```

Question 4

Building a classification model using a neural network function - nnet.

The confusion matrix shows approximately 90% accuracy in the model.

```
# Function to normalize data using min-max normalization
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}

# Normalizing the non-categorical featues
bank$age = normalize(bank$age)
bank$balance = normalize(bank$balance)
bank$day = normalize(bank$day)
bank$duration = normalize(bank$duration)
```

```r
bank$campaign = normalize(bank$campaign)
bank$pdays = normalize(bank$pdays)
bank$previous = normalize(bank$previous)

set.seed(142)
# Using the trainControl function to control the computational nuances of the
# train function
ctrl <- trainControl(method = "cv",
                     n = 5,
                     summaryFunction = twoClassSummary,
                     classProbs = TRUE)

nnet_grid <- expand.grid(decay = c(0.5, 0.1),
                         size = c(5, 6, 7))

# Using the train function from the caret package with the nnet method
model_nnet <- train(y ~ .,
              train,
              method = "nnet",
              maxit = 1000,
              tuneGrid = nnet_grid,
              trControl = ctrl,
              preProc = c("center","scale"),
              metric = "ROC")
```

```
## # weights:  221
## initial  value 1777.918537
## iter  10 value 571.662884
## iter  20 value 521.440217
## iter  30 value 494.498693
## iter  40 value 478.065820
## iter  50 value 469.277202
## iter  60 value 458.828105
## iter  70 value 453.421868
## iter  80 value 450.895339
## iter  90 value 449.839804
## iter 100 value 448.925323
## iter 110 value 448.200075
## iter 120 value 447.953634
## iter 130 value 447.877834
## iter 140 value 447.852344
## iter 150 value 447.771931
## iter 160 value 447.656676
## iter 170 value 447.504928
## iter 180 value 447.440940
## iter 190 value 447.421166
## iter 200 value 447.414162
## iter 210 value 447.393454
## iter 220 value 447.323700
## iter 230 value 447.221376
## iter 240 value 447.007319
## iter 250 value 446.732328
## iter 260 value 446.625154
## iter 270 value 446.606168
## iter 280 value 446.603989
## final  value 446.603823
## converged
## # weights:  221
## initial  value 1903.135435
## iter  10 value 528.700519
## iter  20 value 470.470468
## iter  30 value 446.794969
## iter  40 value 432.416236
## iter  50 value 417.259146
## iter  60 value 402.994719
## iter  70 value 395.878756
## iter  80 value 388.671556
## iter  90 value 385.416711
## iter 100 value 381.483391
## iter 110 value 377.459188
## iter 120 value 375.722323
## iter 130 value 374.765115
## iter 140 value 373.785755
## iter 150 value 373.254490
## iter 160 value 372.930469
## iter 170 value 372.652981
## iter 180 value 372.543193
## iter 190 value 372.529408
## iter 200 value 372.525932
## final  value 372.525519
## converged
## # weights:  265
## initial  value 1845.214128
## iter  10 value 549.972648
## iter  20 value 511.684032
## iter  30 value 491.159527
## iter  40 value 467.068989
## iter  50 value 451.486610
## iter  60 value 444.687667
## iter  70 value 437.916253
## iter  80 value 432.078199
## iter  90 value 430.508751
## iter 100 value 429.658409
## iter 110 value 429.010286
## iter 120 value 428.520507
## iter 130 value 427.938577
## iter 140 value 426.583291
## iter 150 value 424.877061
## iter 160 value 424.271015
## iter 170 value 424.117342
## iter 180 value 424.096222
## iter 190 value 424.089080
## iter 200 value 424.081088
## iter 210 value 424.074278
## iter 220 value 424.070072
## iter 230 value 424.068016
## final  value 424.067874
## converged
## # weights:  265
## initial  value 2993.227869
## iter  10 value 533.272794
## iter  20 value 476.136864
## iter  30 value 449.068555
## iter  40 value 428.927352
## iter  50 value 415.484286
## iter  60 value 404.860950
## iter  70 value 392.033136
## iter  80 value 385.521735
## iter  90 value 372.305509
## iter 100 value 360.004442
## iter 110 value 352.545098
## iter 120 value 348.176502
## iter 130 value 338.658465
## iter 140 value 328.335904
## iter 150 value 324.121455
## iter 160 value 321.609331
## iter 170 value 318.821885
## iter 180 value 317.362832
## iter 190 value 315.984597
## iter 200 value 314.895516
## iter 210 value 313.899814
## iter 220 value 312.899162
## iter 230 value 311.816554
## iter 240 value 309.540077
## iter 250 value 306.992225
## iter 260 value 306.369649
## iter 270 value 304.613168
## iter 280 value 304.307065
## iter 290 value 304.148816
## iter 300 value 304.130786
## iter 310 value 304.121007
## iter 320 value 304.035202
## iter 330 value 303.787883
## iter 340 value 303.322567
## iter 350 value 302.079182
## iter 360 value 302.004252
## iter 370 value 302.000170
## final  value 302.000031
## converged
## # weights:  309
## initial  value 2139.750331
## iter  10 value 574.230130
## iter  20 value 513.549599
## iter  30 value 484.522654
## iter  40 value 460.018008
## iter  50 value 447.200808
## iter  60 value 437.456045
## iter  70 value 430.041747
## iter  80 value 427.517940
## iter  90 value 424.210319
## iter 100 value 421.253547
## iter 110 value 419.934171
## iter 120 value 419.086971
## iter 130 value 418.222864
## iter 140 value 414.381434
## iter 150 value 407.289216
## iter 160 value 404.901178
## iter 170 value 404.016133
## iter 180 value 403.153702
## iter 190 value 401.779509
## iter 200 value 400.519384
## iter 210 value 399.819392
## iter 220 value 399.519071
## iter 230 value 397.846114
## iter 240 value 395.006020
## iter 250 value 394.094332
## iter 260 value 393.123317
## iter 270 value 392.808416
## iter 280 value 392.474670
## iter 290 value 392.022901
## iter 300 value 391.648838
## iter 310 value 391.384532
## iter 320 value 390.908141
## iter 330 value 390.772240
## iter 340 value 390.744818
## iter 350 value 390.731485
## iter 360 value 390.728311
## iter 370 value 390.700959
## iter 380 value 390.149298
## iter 390 value 389.997844
## iter 400 value 389.972780
## iter 410 value 389.966608
## iter 420 value 389.955596
## iter 430 value 388.710876
## iter 440 value 388.402366
## iter 450 value 388.342437
## iter 460 value 388.337151
## final  value 388.337124
## converged
## # weights:  309
## initial  value 1221.452171
## iter  10 value 530.812417
## iter  20 value 472.014314
## iter  30 value 416.549097
## iter  40 value 374.278740
## iter  50 value 347.108777
## iter  60 value 333.162257
## iter  70 value 321.584824
## iter  80 value 315.870868
## iter  90 value 312.016134
## iter 100 value 308.626114
## iter 110 value 306.130426
## iter 120 value 304.372539
## iter 130 value 303.114348
## iter 140 value 302.490257
## iter 150 value 302.128338
## iter 160 value 301.715213
## iter 170 value 300.696136
## iter 180 value 300.106695
## iter 190 value 299.312080
## iter 200 value 296.763568
## iter 210 value 294.482222
## iter 220 value 292.262539
## iter 230 value 290.088811
## iter 240 value 289.215002
## iter 250 value 288.780081
## iter 260 value 288.668919
## iter 270 value 288.627695
## iter 280 value 288.109173
## iter 290 value 287.118796
## iter 300 value 286.873194
## iter 310 value 286.370890
## iter 320 value 284.708767
## iter 330 value 282.241370
## iter 340 value 281.198407
## iter 350 value 280.023364
## iter 360 value 279.399867
## iter 370 value 279.149701
## iter 380 value 279.044457
## iter 390 value 279.025163
## iter 400 value 279.021254
## iter 410 value 279.020620
## iter 420 value 279.020352
## iter 430 value 279.020160
## final  value 279.020141
## converged
## # weights:  221
## initial  value 1274.861911
## iter  10 value 595.796189
## iter  20 value 523.922214
## iter  30 value 495.220379
## iter  40 value 474.735112
## iter  50 value 464.272881
## iter  60 value 458.176303
## iter  70 value 454.817223
## iter  80 value 452.982431
## iter  90 value 451.275001
## iter 100 value 450.266162
## iter 110 value 449.441491
## iter 120 value 448.578610
## iter 130 value 446.984709
## iter 140 value 446.168347
## iter 150 value 445.870149
## iter 160 value 445.778258
## iter 170 value 445.753026
## iter 180 value 445.746342
## iter 190 value 445.730247
## iter 200 value 445.499496
## iter 210 value 445.340475
## iter 220 value 445.322360
## iter 230 value 445.321413
## iter 240 value 445.321308
## final  value 445.321265
## converged
## # weights:  221
## initial  value 2800.126127
## iter  10 value 531.353160
## iter  20 value 463.305063
## iter  30 value 432.098717
## iter  40 value 423.954216
## iter  50 value 416.844191
## iter  60 value 408.677668
## iter  70 value 403.522014
## iter  80 value 397.211715
## iter  90 value 391.153254
## iter 100 value 387.121062
## iter 110 value 384.042683
## iter 120 value 378.468879
## iter 130 value 372.851107
## iter 140 value 369.245517
## iter 150 value 365.102656
## iter 160 value 362.913113
## iter 170 value 361.430922
## iter 180 value 360.097739
## iter 190 value 358.039915
## iter 200 value 356.836081
## iter 210 value 354.985929
## iter 220 value 352.248659
## iter 230 value 348.916583
## iter 240 value 348.806511
## iter 250 value 348.786231
## iter 260 value 348.779506
## iter 270 value 348.777730
## iter 280 value 348.776659
## iter 290 value 348.776443
## iter 300 value 348.776315
## iter 310 value 348.776248
## final  value 348.776241
## converged
## # weights:  265
## initial  value 3420.134074
## iter  10 value 575.872144
## iter  20 value 523.635311
## iter  30 value 492.459306
## iter  40 value 472.849868
## iter  50 value 454.819951
## iter  60 value 440.761436
## iter  70 value 436.418272
## iter  80 value 434.455084
## iter  90 value 432.144773
## iter 100 value 429.916850
## iter 110 value 428.089104
## iter 120 value 425.785066
## iter 130 value 424.333758
## iter 140 value 423.757922
## iter 150 value 423.151309
## iter 160 value 422.643635
## iter 170 value 422.377853
## iter 180 value 422.273685
## iter 190 value 422.206592
## iter 200 value 422.117897
## iter 210 value 422.005133
## iter 220 value 421.984838
## iter 230 value 421.981965
## iter 240 value 421.981643
## final  value 421.981626
## converged
## # weights:  265
## initial  value 1063.306809
## iter  10 value 537.265787
## iter  20 value 476.191937
## iter  30 value 450.777453
## iter  40 value 430.309513
## iter  50 value 400.878980
## iter  60 value 380.343486
## iter  70 value 369.556470
## iter  80 value 360.160062
## iter  90 value 351.762466
## iter 100 value 345.373328
## iter 110 value 340.549850
## iter 120 value 335.299197
## iter 130 value 333.181025
## iter 140 value 332.190952
## iter 150 value 331.797938
## iter 160 value 331.568873
## iter 170 value 331.261677
## iter 180 value 331.084725
## iter 190 value 331.033520
## iter 200 value 331.019453
## iter 210 value 330.986220
## iter 220 value 330.981396
## iter 230 value 330.980817
## iter 240 value 330.980688
## iter 240 value 330.980685
## iter 240 value 330.980684
## final  value 330.980684
## converged
## # weights:  309
## initial  value 2649.353621
## iter  10 value 572.271592
## iter  20 value 511.707244
## iter  30 value 480.762422
## iter  40 value 460.223356
## iter  50 value 447.982846
## iter  60 value 441.476320
## iter  70 value 436.148671
## iter  80 value 432.320353
## iter  90 value 430.716431
## iter 100 value 429.706280
## iter 110 value 428.689374
## iter 120 value 427.646335
## iter 130 value 424.704805
## iter 140 value 422.384582
## iter 150 value 421.734652
## iter 160 value 420.225663
## iter 170 value 418.570399
## iter 180 value 417.900082
## iter 190 value 417.232271
## iter 200 value 416.562315
## iter 210 value 416.261707
## iter 220 value 416.143398
## iter 230 value 416.102965
## iter 240 value 416.071181
## iter 250 value 415.975706
## iter 260 value 415.914354
## iter 270 value 415.890984
## iter 280 value 415.888332
## final  value 415.888178
## converged
## # weights:  309
## initial  value 1966.257738
## iter  10 value 544.960797
## iter  20 value 472.037879
## iter  30 value 437.632183
## iter  40 value 396.012751
## iter  50 value 366.829705
## iter  60 value 353.202583
## iter  70 value 342.511368
## iter  80 value 330.561184
## iter  90 value 321.232305
## iter 100 value 315.812956
## iter 110 value 310.265940
## iter 120 value 305.787347
## iter 130 value 300.491715
## iter 140 value 297.702793
## iter 150 value 294.497441
## iter 160 value 290.927070
## iter 170 value 287.508923
## iter 180 value 283.081539
## iter 190 value 280.689100
## iter 200 value 279.529166
## iter 210 value 279.052688
## iter 220 value 278.445123
## iter 230 value 278.139185
## iter 240 value 277.682187
## iter 250 value 276.189916
## iter 260 value 274.155379
## iter 270 value 272.361858
## iter 280 value 270.487078
## iter 290 value 268.031707
## iter 300 value 264.930702
## iter 310 value 262.885937
## iter 320 value 261.747703
## iter 330 value 260.354799
## iter 340 value 259.305918
## iter 350 value 258.851501
## iter 360 value 258.474304
## iter 370 value 258.095538
## iter 380 value 257.561633
## iter 390 value 257.258077
## iter 400 value 257.163000
## iter 410 value 257.141927
## iter 420 value 257.137559
## iter 430 value 257.135427
## iter 440 value 257.133305
## iter 450 value 257.132626
## iter 460 value 257.132513
## iter 460 value 257.132510
## iter 460 value 257.132510
## final  value 257.132510
## converged
## # weights:  221
## initial  value 2090.019805
## iter  10 value 544.082720
## iter  20 value 505.947916
## iter  30 value 488.855013
## iter  40 value 475.983686
## iter  50 value 467.884233
## iter  60 value 461.889832
## iter  70 value 458.468516
## iter  80 value 457.238055
## iter  90 value 456.653676
## iter 100 value 455.952981
## iter 110 value 455.622655
## iter 120 value 455.500878
## iter 130 value 455.449289
## iter 140 value 455.419697
## iter 150 value 455.403227
## iter 160 value 455.396321
## iter 170 value 455.395505
## final  value 455.395468
## converged
## # weights:  221
## initial  value 1365.265123
## iter  10 value 545.788265
## iter  20 value 491.837017
## iter  30 value 443.603671
## iter  40 value 412.107172
## iter  50 value 397.521732
## iter  60 value 387.991637
## iter  70 value 383.096408
## iter  80 value 381.676310
## iter  90 value 381.163510
## iter 100 value 380.674284
## iter 110 value 380.078862
## iter 120 value 379.920859
## iter 130 value 379.883070
## iter 140 value 379.862937
## iter 150 value 379.844328
## iter 160 value 379.715718
## iter 170 value 378.347875
## iter 180 value 377.068019
## iter 190 value 376.836056
## iter 200 value 376.221109
## iter 210 value 374.732703
## iter 220 value 373.043526
## iter 230 value 372.685922
## iter 240 value 372.453644
## iter 250 value 372.382637
## iter 260 value 372.378149
## final  value 372.378042
## converged
## # weights:  265
## initial  value 2599.496060
## iter  10 value 578.935706
## iter  20 value 522.285079
## iter  30 value 491.230290
## iter  40 value 478.597327
## iter  50 value 470.509177
## iter  60 value 463.838614
## iter  70 value 453.764824
## iter  80 value 446.530636
## iter  90 value 440.395673
## iter 100 value 431.673506
## iter 110 value 427.349742
## iter 120 value 425.963559
## iter 130 value 425.226193
## iter 140 value 424.190843
## iter 150 value 421.957631
## iter 160 value 418.158917
## iter 170 value 415.745281
## iter 180 value 415.187601
## iter 190 value 415.026190
## iter 200 value 414.964600
## iter 210 value 414.897418
## iter 220 value 414.838164
## iter 230 value 414.825407
## iter 240 value 414.823071
## iter 250 value 414.822529
## final  value 414.822489
## converged
## # weights:  265
## initial  value 1728.452056
## iter  10 value 521.207640
## iter  20 value 440.011523
## iter  30 value 404.245497
## iter  40 value 377.488117
## iter  50 value 367.126450
## iter  60 value 361.195399
## iter  70 value 356.813741
## iter  80 value 353.189458
## iter  90 value 349.773208
## iter 100 value 347.143395
## iter 110 value 345.168956
## iter 120 value 343.090404
## iter 130 value 341.964861
## iter 140 value 340.191982
## iter 150 value 338.496288
## iter 160 value 337.115304
## iter 170 value 335.621285
## iter 180 value 334.916343
## iter 190 value 334.227785
## iter 200 value 332.105506
## iter 210 value 326.847724
## iter 220 value 322.412919
## iter 230 value 318.627087
## iter 240 value 315.578726
## iter 250 value 313.040935
## iter 260 value 311.272718
## iter 270 value 310.070963
## iter 280 value 308.512100
## iter 290 value 307.266459
## iter 300 value 306.678342
## iter 310 value 305.967443
## iter 320 value 305.312582
## iter 330 value 304.322824
## iter 340 value 303.682135
## iter 350 value 303.413885
## iter 360 value 302.994714
## iter 370 value 302.737729
## iter 380 value 302.469339
## iter 390 value 302.338098
## iter 400 value 302.266002
## iter 410 value 302.242794
## iter 420 value 302.238514
## iter 430 value 302.237610
## iter 440 value 302.237471
## iter 440 value 302.237468
## iter 440 value 302.237468
## final  value 302.237468
## converged
## # weights:  309
## initial  value 2408.347224
## iter  10 value 583.885502
## iter  20 value 509.518152
## iter  30 value 481.525838
## iter  40 value 462.067521
## iter  50 value 445.506798
## iter  60 value 437.321386
## iter  70 value 430.963545
## iter  80 value 424.006010
## iter  90 value 417.434181
## iter 100 value 410.887680
## iter 110 value 405.311842
## iter 120 value 402.097528
## iter 130 value 397.779147
## iter 140 value 394.102469
## iter 150 value 392.371678
## iter 160 value 391.332478
## iter 170 value 390.841214
## iter 180 value 390.294444
## iter 190 value 389.828228
## iter 200 value 389.103908
## iter 210 value 388.538978
## iter 220 value 388.024357
## iter 230 value 387.648769
## iter 240 value 387.014947
## iter 250 value 385.971062
## iter 260 value 385.503230
## iter 270 value 385.380132
## iter 280 value 385.343833
## iter 290 value 385.326010
## iter 300 value 385.322063
## iter 310 value 385.321391
## iter 320 value 385.321058
## final  value 385.320976
## converged
## # weights:  309
## initial  value 1785.136586
## iter  10 value 558.700979
## iter  20 value 478.150215
## iter  30 value 428.641943
## iter  40 value 389.146541
## iter  50 value 367.256912
## iter  60 value 355.330622
## iter  70 value 343.611238
## iter  80 value 333.271665
## iter  90 value 322.969661
## iter 100 value 313.782674
## iter 110 value 304.206214
## iter 120 value 296.155913
## iter 130 value 289.117286
## iter 140 value 282.367361
## iter 150 value 278.495712
## iter 160 value 273.430390
## iter 170 value 268.715446
## iter 180 value 266.295331
## iter 190 value 264.978517
## iter 200 value 263.710469
## iter 210 value 262.761215
## iter 220 value 262.248873
## iter 230 value 261.688869
## iter 240 value 261.228652
## iter 250 value 260.952805
## iter 260 value 260.849952
## iter 270 value 260.805719
## iter 280 value 260.729533
## iter 290 value 260.666445
## iter 300 value 260.652802
## iter 310 value 260.649547
## iter 320 value 260.648845
## final  value 260.648776
## converged
## # weights:  221
## initial  value 1989.207099
## iter  10 value 560.969676
## iter  20 value 522.925657
## iter  30 value 498.630562
## iter  40 value 479.586162
## iter  50 value 469.176931
## iter  60 value 462.749400
## iter  70 value 459.605318
## iter  80 value 457.215903
## iter  90 value 454.804316
## iter 100 value 451.146606
## iter 110 value 445.991715
## iter 120 value 443.412915
## iter 130 value 442.720397
## iter 140 value 441.538727
## iter 150 value 440.717876
## iter 160 value 440.495755
## iter 170 value 440.366555
## iter 180 value 440.214249
## iter 190 value 439.970234
## iter 200 value 439.885375
## iter 210 value 439.717715
## iter 220 value 439.595773
## iter 230 value 439.565393
## iter 240 value 439.561247
## final  value 439.560585
## converged
## # weights:  221
## initial  value 1639.225022
## iter  10 value 523.433340
## iter  20 value 460.440144
## iter  30 value 430.242734
## iter  40 value 409.818631
## iter  50 value 395.438840
## iter  60 value 388.782714
## iter  70 value 385.920135
## iter  80 value 383.112461
## iter  90 value 379.935983
## iter 100 value 377.002318
## iter 110 value 373.374664
## iter 120 value 371.320860
## iter 130 value 370.322653
## iter 140 value 370.033962
## iter 150 value 369.837312
## iter 160 value 369.738036
## iter 170 value 369.718689
## iter 180 value 369.710640
## iter 190 value 369.708859
## iter 200 value 369.708674
## iter 200 value 369.708670
## iter 200 value 369.708669
## final  value 369.708669
## converged
## # weights:  265
## initial  value 1482.665471
## iter  10 value 561.390562
## iter  20 value 513.883819
## iter  30 value 485.974688
## iter  40 value 472.932036
## iter  50 value 458.144488
## iter  60 value 450.762802
## iter  70 value 443.384704
## iter  80 value 438.088543
## iter  90 value 434.843663
## iter 100 value 433.284642
## iter 110 value 432.340377
## iter 120 value 431.598958
## iter 130 value 430.886297
## iter 140 value 429.836780
## iter 150 value 428.644243
## iter 160 value 427.774687
## iter 170 value 427.382679
## iter 180 value 427.076008
## iter 190 value 426.718337
## iter 200 value 426.020976
## iter 210 value 425.545022
## iter 220 value 425.314236
## iter 230 value 425.232560
## iter 240 value 425.149286
## iter 250 value 425.115929
## iter 260 value 425.100817
## iter 270 value 425.006348
## iter 280 value 424.962435
## iter 290 value 424.911247
## iter 300 value 424.890192
## iter 310 value 424.883559
## iter 320 value 424.880820
## iter 330 value 424.880008
## final  value 424.879909
## converged
## # weights:  265
## initial  value 1218.185828
## iter  10 value 518.081926
## iter  20 value 463.271171
## iter  30 value 424.634410
## iter  40 value 400.404566
## iter  50 value 391.686100
## iter  60 value 384.990186
## iter  70 value 379.595420
## iter  80 value 375.954246
## iter  90 value 373.703739
## iter 100 value 371.244092
## iter 110 value 365.883847
## iter 120 value 359.163749
## iter 130 value 351.253583
## iter 140 value 345.143906
## iter 150 value 343.255907
## iter 160 value 342.147048
## iter 170 value 341.336905
## iter 180 value 341.041725
## iter 190 value 340.818421
## iter 200 value 340.720388
## iter 210 value 340.706371
## iter 220 value 340.701602
## iter 230 value 340.699285
## iter 240 value 340.698890
## final  value 340.698820
## converged
## # weights:  309
## initial  value 1257.875695
## iter  10 value 580.293933
## iter  20 value 515.854780
## iter  30 value 482.330509
## iter  40 value 463.240279
## iter  50 value 447.779804
## iter  60 value 430.518480
## iter  70 value 421.742475
## iter  80 value 416.626747
## iter  90 value 413.168265
## iter 100 value 410.593383
## iter 110 value 408.325730
## iter 120 value 407.633636
## iter 130 value 407.441190
## iter 140 value 407.276790
## iter 150 value 407.044190
## iter 160 value 406.863598
## iter 170 value 406.815841
## iter 180 value 406.796802
## iter 190 value 406.785318
## iter 200 value 406.780512
## iter 210 value 406.778685
## iter 220 value 406.777942
## final  value 406.777703
## converged
## # weights:  309
## initial  value 966.207815
## iter  10 value 532.254755
## iter  20 value 452.863635
## iter  30 value 415.456733
## iter  40 value 396.741372
## iter  50 value 382.796251
## iter  60 value 367.408069
## iter  70 value 357.253924
## iter  80 value 344.835210
## iter  90 value 332.270512
## iter 100 value 321.902969
## iter 110 value 311.398038
## iter 120 value 302.762768
## iter 130 value 298.316337
## iter 140 value 294.894413
## iter 150 value 292.954325
## iter 160 value 292.409310
## iter 170 value 292.183520
## iter 180 value 291.558870
## iter 190 value 291.049665
## iter 200 value 288.426342
## iter 210 value 286.382737
## iter 220 value 285.948269
## iter 230 value 285.758821
## iter 240 value 285.687328
## iter 250 value 285.656584
## iter 260 value 285.560855
## iter 270 value 285.160630
## iter 280 value 285.042302
## iter 290 value 285.011345
## iter 300 value 285.004170
## iter 310 value 285.002316
## iter 320 value 285.001778
## final  value 285.001699
## converged
## # weights:  221
## initial  value 2693.118444
## iter  10 value 626.117740
## iter  20 value 548.947523
## iter  30 value 516.485985
## iter  40 value 492.544796
## iter  50 value 476.006632
## iter  60 value 467.997739
## iter  70 value 460.243678
## iter  80 value 456.917265
## iter  90 value 455.680167
## iter 100 value 454.492334
## iter 110 value 452.711120
## iter 120 value 451.561231
## iter 130 value 451.276559
## iter 140 value 451.224141
## iter 150 value 451.204144
## iter 160 value 451.186429
## iter 170 value 451.158235
## iter 180 value 451.142669
## iter 190 value 451.139574
## iter 200 value 451.139366
## iter 200 value 451.139364
## iter 200 value 451.139364
## final  value 451.139364
## converged
## # weights:  221
## initial  value 1476.201664
## iter  10 value 514.586344
## iter  20 value 455.787278
## iter  30 value 417.015989
## iter  40 value 397.311333
## iter  50 value 390.607737
## iter  60 value 386.477977
## iter  70 value 384.615018
## iter  80 value 380.685597
## iter  90 value 378.341693
## iter 100 value 376.567403
## iter 110 value 375.519303
## iter 120 value 373.549395
## iter 130 value 372.095963
## iter 140 value 371.408453
## iter 150 value 370.925473
## iter 160 value 370.699252
## iter 170 value 370.418761
## iter 180 value 370.300227
## iter 190 value 370.263177
## iter 200 value 370.239862
## iter 210 value 370.232733
## iter 220 value 370.231534
## iter 230 value 370.231402
## final  value 370.231395
## converged
## # weights:  265
## initial  value 3233.504700
## iter  10 value 602.306260
## iter  20 value 542.653329
## iter  30 value 510.340418
## iter  40 value 485.125526
## iter  50 value 463.699268
## iter  60 value 458.251769
## iter  70 value 454.786117
## iter  80 value 451.111363
## iter  90 value 446.379642
## iter 100 value 443.109373
## iter 110 value 438.274767
## iter 120 value 435.743123
## iter 130 value 434.716616
## iter 140 value 433.885002
## iter 150 value 433.482345
## iter 160 value 433.250419
## iter 170 value 432.730068
## iter 180 value 432.188049
## iter 190 value 431.971790
## iter 200 value 431.814429
## iter 210 value 431.745595
## iter 220 value 431.719038
## iter 230 value 431.338447
## iter 240 value 430.857661
## iter 250 value 430.759231
## iter 260 value 430.735602
## iter 270 value 430.729499
## iter 280 value 430.724968
## iter 290 value 430.724572
## final  value 430.724562
## converged
## # weights:  265
## initial  value 2098.054157
## iter  10 value 691.600753
## iter  20 value 586.086246
## iter  30 value 517.080915
## iter  40 value 463.953601
## iter  50 value 432.411508
## iter  60 value 412.464580
## iter  70 value 400.273657
## iter  80 value 393.294393
## iter  90 value 386.359941
## iter 100 value 376.999462
## iter 110 value 356.735200
## iter 120 value 345.156369
## iter 130 value 337.724859
## iter 140 value 333.049934
## iter 150 value 330.290719
## iter 160 value 328.896840
## iter 170 value 326.874371
## iter 180 value 325.650604
## iter 190 value 324.484116
## iter 200 value 323.595825
## iter 210 value 322.670945
## iter 220 value 321.099951
## iter 230 value 320.620800
## iter 240 value 319.832028
## iter 250 value 319.481157
## iter 260 value 319.336505
## iter 270 value 319.244646
## iter 280 value 319.204746
## iter 290 value 319.125401
## iter 300 value 319.063401
## iter 310 value 319.048204
## iter 320 value 319.046252
## final  value 319.046156
## converged
## # weights:  309
## initial  value 927.407915
## iter  10 value 587.416015
## iter  20 value 499.907268
## iter  30 value 468.012145
## iter  40 value 436.747121
## iter  50 value 426.326633
## iter  60 value 419.575045
## iter  70 value 414.252981
## iter  80 value 410.950922
## iter  90 value 409.155124
## iter 100 value 408.273336
## iter 110 value 407.921868
## iter 120 value 407.823548
## iter 130 value 407.634389
## iter 140 value 407.240598
## iter 150 value 406.875085
## iter 160 value 406.590147
## iter 170 value 406.007692
## iter 180 value 404.928677
## iter 190 value 403.788333
## iter 200 value 402.997179
## iter 210 value 402.712542
## iter 220 value 402.539516
## iter 230 value 402.388578
## iter 240 value 401.996157
## iter 250 value 401.757976
## iter 260 value 401.323942
## iter 270 value 401.279661
## iter 280 value 401.266498
## iter 290 value 401.256768
## iter 300 value 401.255669
## final  value 401.255509
## converged
## # weights:  309
## initial  value 1277.836763
## iter  10 value 531.671448
## iter  20 value 469.218740
## iter  30 value 413.861598
## iter  40 value 378.209317
## iter  50 value 359.378184
## iter  60 value 340.762114
## iter  70 value 330.572983
## iter  80 value 324.556960
## iter  90 value 321.817344
## iter 100 value 319.095107
## iter 110 value 315.347384
## iter 120 value 312.594827
## iter 130 value 310.796654
## iter 140 value 309.590183
## iter 150 value 308.983722
## iter 160 value 308.662057
## iter 170 value 308.416687
## iter 180 value 308.278885
## iter 190 value 308.160879
## iter 200 value 308.011853
## iter 210 value 307.751010
## iter 220 value 307.674677
## iter 230 value 307.653220
## iter 240 value 307.625588
## iter 250 value 307.598739
## iter 260 value 307.588119
## iter 270 value 307.584364
## iter 280 value 307.583785
## iter 290 value 307.583729
## final  value 307.583718
## converged
## # weights:  265
## initial  value 3755.121815
## iter  10 value 815.205088
## iter  20 value 720.372898
## iter  30 value 663.978495
## iter  40 value 628.268544
## iter  50 value 601.563582
## iter  60 value 590.167161
## iter  70 value 582.581732
## iter  80 value 577.323862
## iter  90 value 573.613100
## iter 100 value 570.169738
## iter 110 value 567.126790
## iter 120 value 564.962338
## iter 130 value 564.008503
## iter 140 value 563.505185
## iter 150 value 562.949152
## iter 160 value 562.194335
## iter 170 value 561.774924
## iter 180 value 561.605432
## iter 190 value 561.569885
## iter 200 value 561.555258
## iter 210 value 561.551159
## iter 220 value 561.550228
## iter 230 value 561.550047
## iter 230 value 561.550041
## iter 230 value 561.550040
## final  value 561.550040
## converged
```

```
model_nnet
```

```
## Neural Network
##
## 2984 samples
##   16 predictor
##    2 classes: 'no', 'yes'
##
## Pre-processing: centered (42), scaled (42)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 2388, 2387, 2387, 2387, 2387
## Resampling results across tuning parameters:
##
##   decay  size  ROC        Sens       Spec
##   0.1    5     0.8235708  0.9382576  0.4041773
##   0.1    6     0.8294158  0.9325758  0.4098465
##   0.1    7     0.7903993  0.9242424  0.3981245
##   0.5    5     0.8530150  0.9553030  0.3719949
##   0.5    6     0.8534917  0.9507576  0.3575021
##   0.5    7     0.8350937  0.9484848  0.3577579
##
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were size = 6 and decay = 0.5.
```

```
# Testing the model on the test subset
nnet_pred <- predict(model_nnet, test)

# Displaying the accuracy of the model
```

```r
confusionMatrix(nnet_pred, test$y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1306   96
##        yes   54   81
##
##                Accuracy : 0.9024
##                  95% CI : (0.8865, 0.9168)
##     No Information Rate : 0.8848
##     P-Value [Acc > NIR] : 0.015536
##
##                   Kappa : 0.466
##
##  Mcnemar's Test P-Value : 0.000815
##
##             Sensitivity : 0.9603
##             Specificity : 0.4576
##          Pos Pred Value : 0.9315
##          Neg Pred Value : 0.6000
##              Prevalence : 0.8848
##          Detection Rate : 0.8497
##    Detection Prevalence : 0.9122
##       Balanced Accuracy : 0.7090
##
##        'Positive' Class : no
##
```

Question 5

Comparing the accuracy of the two models based on the absolute accuracy and AUC.

```r
library(ROCR)
```

```
## Loading required package: gplots
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```r
# Accuracy of the SVM model
paste("Accuracy of SVM Model = ", mean(svm_pred == test$y))
```

```
## [1] "Accuracy of SVM Model =  0.896551724137931"
```

```r
# Accuracy of the ANN model
paste("Accuracy of ANN Model = ", mean(nnet_pred == test$y))
```

```
## [1] "Accuracy of ANN Model =  0.902407286922576"
```

```r
# Generating probabilities
svm_prob <- predict(model_svm, test, type = "prob")
nnet_prob <- predict(model_nnet, test, type = "prob")
```

```r
# Prediction function used to transform the probability list into a standardized format
svm_input <- prediction(svm_prob[,2], test$y)
nnet_input <- prediction(nnet_prob[,2], test$y)

# Performance function used to evaluate the AUC
svm_auc <- performance(svm_input, "auc")
nnet_auc <- performance(nnet_input, "auc")

# Displaying the AUC values
paste("AUC of SVM Model = ", svm_auc@y.values)
```

```
## [1] "AUC of SVM Model =  0.893639913592537"
```

```r
paste("AUC of ANN Model = ", nnet_auc@y.values)
```

```
## [1] "AUC of ANN Model =  0.899725822532398"
```

Problem 2

Using the wine dataset and applying kmeans clustering algorithm and visualizing the results.

Step 1

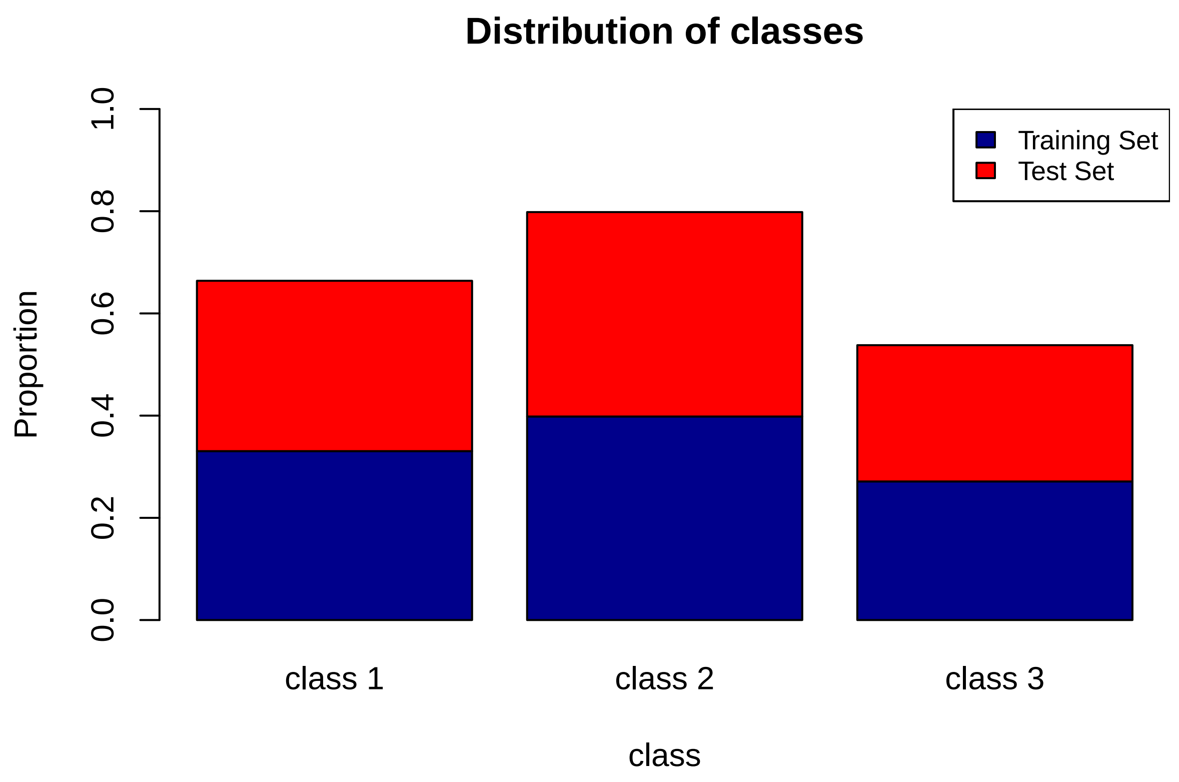
Loading the dataset and setting the column headers.

```r
Q2_data <- read.csv("wine-data.csv")
colnames(Q2_data) <- c("class", "Alcohol", "Malic acid", "Ash",
                       "Alcalinity_of_ash", "Magnesium", "Total_phenols",
                       "Flavanoids", "Nonflavanoid_phenols", "Proanthocyanins",
                       "Color_intensity", "Hue", "OD280/OD315", "Proline")

# Initializing lists to summarize the accuracies
acc_train <- rep(0,6)
acc_test <- rep(0,6)
```

Step 2

Splitting the dataset into training and testing datasets and verifying the proportion of the three classes in the two subsets

```r
library(caTools)

# Splitting the data into training and testing datasets
samp_splt <- sample.split(Q2_data$class, SplitRatio = 2/3, group = Q2_data$class)
train_data <- subset(Q2_data, samp_splt == TRUE)
test_data <- subset(Q2_data, samp_splt == FALSE)

# Visual representation of equal distribution of cultivars in training/test set
mat = matrix(0,2,3)
l = list()

for (i in 1:3){
  l[[i]] = c(sum(train_data$class == i)/nrow(train_data),
             sum(test_data$class == i)/nrow(test_data))
  mat[,i] = l[[i]]
}

rownames(mat) = c("Training Set","Test Set")
colnames(mat) = c("class 1","class 2","class 3")

barplot(mat, beside = FALSE, col = c("darkblue","red"),
        ylim = c(0,1), ylab = "Proportion", xlab = "class",
        main = "Distribution of classes")

legend("topright", legend = c("Training Set","Test Set"),
       cex = 0.8, fill=c("darkblue","red"))
```



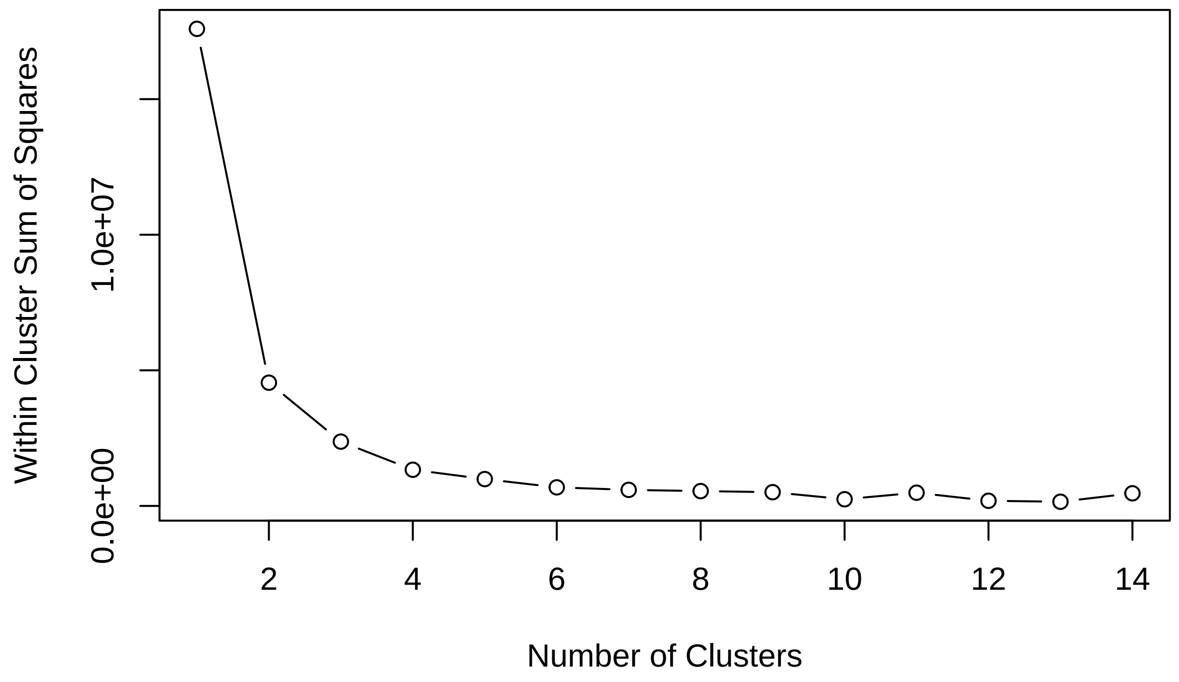
Step 3

Summarizing the data and determining the number of clusters to be used.

```
summary(Q2_data)
```

```
##      class          Alcohol        Malic acid         Ash
##  Min.   :1.000   Min.   :11.03   Min.   :0.740   Min.   :1.360
##  1st Qu.:1.000   1st Qu.:12.36   1st Qu.:1.603   1st Qu.:2.210
##  Median :2.000   Median :13.05   Median :1.865   Median :2.360
##  Mean   :1.938   Mean   :13.00   Mean   :2.336   Mean   :2.367
##  3rd Qu.:3.000   3rd Qu.:13.68   3rd Qu.:3.083   3rd Qu.:2.558
##  Max.   :3.000   Max.   :14.83   Max.   :5.800   Max.   :3.230
##  Alcalinity_of_ash   Magnesium      Total_phenols     Flavanoids
##  Min.   :10.60     Min.   : 70.00   Min.   :0.980   Min.   :0.340
##  1st Qu.:17.20     1st Qu.: 88.00   1st Qu.:1.742   1st Qu.:1.205
##  Median :19.50     Median : 98.00   Median :2.355   Median :2.135
##  Mean   :19.49     Mean   : 99.74   Mean   :2.295   Mean   :2.029
##  3rd Qu.:21.50     3rd Qu.:107.00   3rd Qu.:2.800   3rd Qu.:2.875
##  Max.   :30.00     Max.   :162.00   Max.   :3.880   Max.   :5.080
##  Nonflavanoid_phenols Proanthocyanins Color_intensity       Hue
##  Min.   :0.1300       Min.   :0.410   Min.   : 1.280   Min.   :0.4800
##  1st Qu.:0.2700       1st Qu.:1.250   1st Qu.: 3.220   1st Qu.:0.7825
##  Median :0.3400       Median :1.555   Median : 4.690   Median :0.9650
##  Mean   :0.3619       Mean   :1.591   Mean   : 5.058   Mean   :0.9574
##  3rd Qu.:0.4375       3rd Qu.:1.950   3rd Qu.: 6.200   3rd Qu.:1.1200
##  Max.   :0.6600       Max.   :3.580   Max.   :13.000   Max.   :1.7100
##   OD280/OD315       Proline
##  Min.   :1.270   Min.   : 278.0
##  1st Qu.:1.938   1st Qu.: 500.5
##  Median :2.780   Median : 673.5
##  Mean   :2.612   Mean   : 746.9
##  3rd Qu.:3.170   3rd Qu.: 985.0
##  Max.   :4.000   Max.   :1680.0
```

```r
# Calculating Within Cluster Sum of Squares
wcss <- (nrow(Q2_data)-1)*sum(apply(Q2_data,2,var))
for (i in 2:14) wcss[i] <- sum(kmeans(Q2_data[,1:14],
                                       centers=i)$withinss)

plot(1:14, wcss, type="b", xlab="Number of Clusters",
     ylab="Within Cluster Sum of Squares")
```



Step 4

Raw Data and Euclidean Distance

Creating a list of 100 possible ways of clustering, using different seeds and euclidean distance.

Choosing the one/s with minimal total WCSS.

```r
library(cclust)
L1 = list()
totw1 = list()

for (i in 1:100) {
  set.seed(i)
  # Using the cclust function to cluster the data
  L1[[i]] = cclust(as.matrix(train_data[,2:14]), 3,
                   method = "kmeans", dist = "euclidean")
  totw1[[i]] = sum(L1[[i]]$withinss)
}

# Finding the minimal total WCSS
min_ss = min(unlist(totw1))

for (i in 1:100){
  if (totw1[[i]] == min_ss){
    pred_train1 = predict(L1[[i]], newdata = as.matrix(train_data[,2:14]))
    pred_test1 = predict(L1[[i]], newdata = as.matrix(test_data[,2:14]))
    # print(i)
    # print(table(train_data[,1],pred_train1$cluster))
    # print(table(test_data[,1],pred_test1$cluster))
  }
}

# Choosing L1[[3]]
chosen_pred1train = predict(L1[[3]], newdata = as.matrix(train_data[,2:14]))
chosen_pred1test = predict(L1[[3]], newdata = as.matrix(test_data[,2:14]))

table(train_data[,1],chosen_pred1train$cluster)
```

```
##
##      1  2  3
##   1 30  0  9
##   2  1 29 17
##   3  0 16 16
```

```
table(test_data[,1], chosen_pred1test$cluster)
```

```
##
##      1  2  3
##   1 15  0  5
##   2  0 21  3
##   3  0  5 11
```

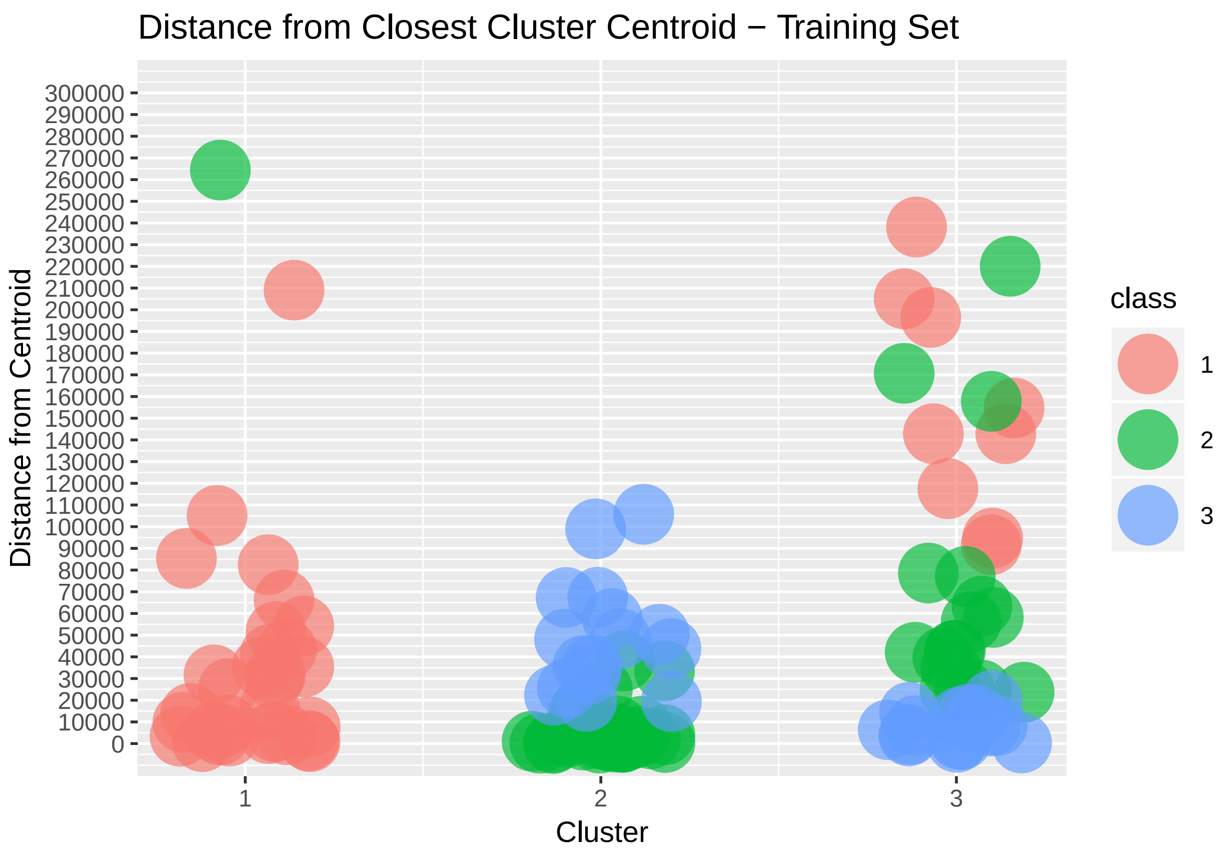
```
# Assigning accuracies
acc_train[1] <- mean(train_data[,1] == chosen_pred1train$cluster)
acc_test[1] <- mean(test_data[,1] == chosen_pred1test$cluster)
L1[[3]]$centers
```

```
##      Alcohol Malic acid      Ash Alcalinity_of_ash Magnesium Total_phenols
## 2   13.80613   1.748387 2.447742          17.15484 105.45161      2.807097
## 85  12.65111   2.410000 2.282222          20.27778  93.62222      1.991556
## 131 12.84357   2.181190 2.377381          19.78095 103.16667      2.157619
##     Flavanoids Nonflavanoid_phenols Proanthocyanins Color_intensity
## 2     2.948065            0.2993548        1.871613        5.469032
## 85    1.585333            0.3764444        1.329778        4.356222
## 131   1.633810            0.3900000        1.537381        5.076905
##           Hue OD280/OD315   Proline
## 2   1.1154839    3.100968 1222.8065
## 85  0.9711111    2.334444  472.9111
## 131 0.9337143    2.428571  739.2857
```

Step 5

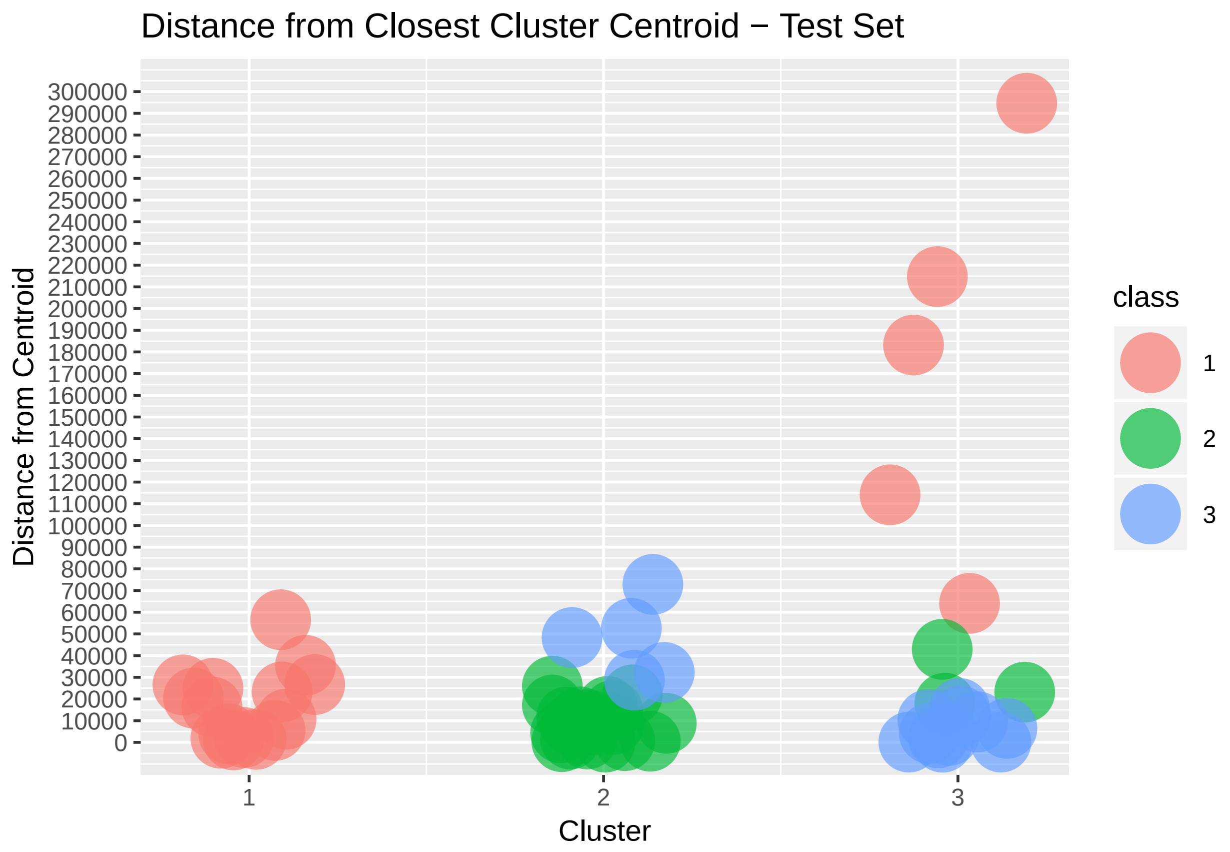
Visualizing the results using a jitter plot for the training and testing datasets

```
library(ggplot2)

class1train_raw = subset(train_data, train_data[,1] == 1)
class2train_raw = subset(train_data, train_data[,1] == 2)
class3train_raw = subset(train_data, train_data[,1] == 3)

# Calculating the distance from the centroids
class1train_raw$sse = apply(class1train_raw[,2:14], 1,
                             function(x) sum( (x-L1[[3]]$centers[1,])^2 ))
class2train_raw$sse = apply(class2train_raw[,2:14], 1,
                             function(x) sum( (x-L1[[3]]$centers[2,])^2 ))
class3train_raw$sse = apply(class3train_raw[,2:14], 1,
                             function(x) sum( (x-L1[[3]]$centers[3,])^2 ))

sse_train_raw = rbind(class1train_raw, class2train_raw, class3train_raw)

sse_train_raw$cluster = jitter(chosen_pred1train$cluster)
sse_train_raw$class = cut(sse_train_raw$class, c(.5,1.5,2.5,3.5),
                           right=FALSE, labels=c(1:3))

# Jitter plot to visualize distance from closest cluster centroid
# for training set
jitplot_train_raw = qplot(cluster, sse, data = sse_train_raw,
                           color = class, alpha = I(2/3), size = I(10))
jitplot_train_raw + coord_cartesian(ylim=c(0, 300000)) +
  scale_y_continuous(breaks=seq(0, 300000, 10000)) +
  scale_x_continuous(breaks=seq(1,3,1)) + xlab("Cluster") +
  ylab("Distance from Centroid") +
  ggtitle("Distance from Closest Cluster Centroid - Training Set")
```



```
# For testing set
class1test_raw = subset(test_data, test_data[,1] == 1)
class2test_raw = subset(test_data, test_data[,1] == 2)
class3test_raw = subset(test_data, test_data[,1] == 3)
```

```r
class1test_raw$sse = apply(class1test_raw[,2:14], 1,
                           function(x) sum( (x-L1[[3]]$centers[1,])^2 ))
class2test_raw$sse = apply(class2test_raw[,2:14], 1,
                           function(x) sum( (x-L1[[3]]$centers[2,])^2 ))
class3test_raw$sse = apply(class3test_raw[,2:14], 1,
                           function(x) sum( (x-L1[[3]]$centers[3,])^2 ))

sse_test_raw = rbind(class1test_raw, class2test_raw, class3test_raw)

sse_test_raw$cluster = jitter(chosen_pred1test$cluster)
sse_test_raw$class = cut(sse_test_raw$class, c(.5,1.5,2.5,3.5),
                         right=FALSE, labels=c(1:3))

# Jitter plot to visualize distance from closest cluster centroid for testing set
jitplot_test_raw = qplot(cluster, sse, data = sse_test_raw,
                         color=class, alpha = I(2/3), size = I(10))
jitplot_test_raw + coord_cartesian(ylim=c(0, 300000)) +
  scale_y_continuous(breaks=seq(0, 300000, 10000)) +
  scale_x_continuous(breaks=seq(1,3,1)) + xlab("Cluster") +
  ylab("Distance from Centroid") +
  ggtitle("Distance from Closest Cluster Centroid - Test Set")
```



Step 6

Raw Data & Manhattan Distance

Using manhattan distance for the clustering.

```r
L1_manh = list()
totw1_manh = list()

for (i in 1:100) {
  set.seed(i)
  L1_manh[[i]] = cclust(as.matrix(train_data[,2:14]), 3,
                        method = "kmeans", dist = "manhattan")
  totw1_manh[[i]] = sum(L1_manh[[i]]$withinss)
}

min_ss_manh = min(unlist(totw1_manh))

for (i in 1:100){
  if (totw1_manh[[i]] == min_ss_manh){
    pred_train1_manh = predict(L1_manh[[i]],
                               newdata = as.matrix(train_data[,2:14]))
    pred_test1_manh = predict(L1_manh[[i]],
                              newdata = as.matrix(test_data[,2:14]))
    # print(i)
    # print(table(train_data[,1],pred_train1_manh$cluster))
    # print(table(test_data[,1],pred_test1_manh$cluster))
  }
}

# Choose L1_manh[[30]] as the best clustering among the obtained clusterings
chosen_pred1train_manh = predict(L1_manh[[30]],
                                 newdata = as.matrix(train_data[,2:14]))
chosen_pred1test_manh = predict(L1_manh[[30]],
                                newdata = as.matrix(test_data[,2:14]))

table(train_data[,1],chosen_pred1train_manh$cluster)
```

```
##
##      1  2  3
##   1 30  0  9
##   2  1 29 17
##   3  0 13 19
```

```r
table(test_data[,1], chosen_pred1test_manh$cluster)
```

```
##
##      1  2  3
##   1 16  0  4
##   2  0 21  3
##   3  0  5 11
```

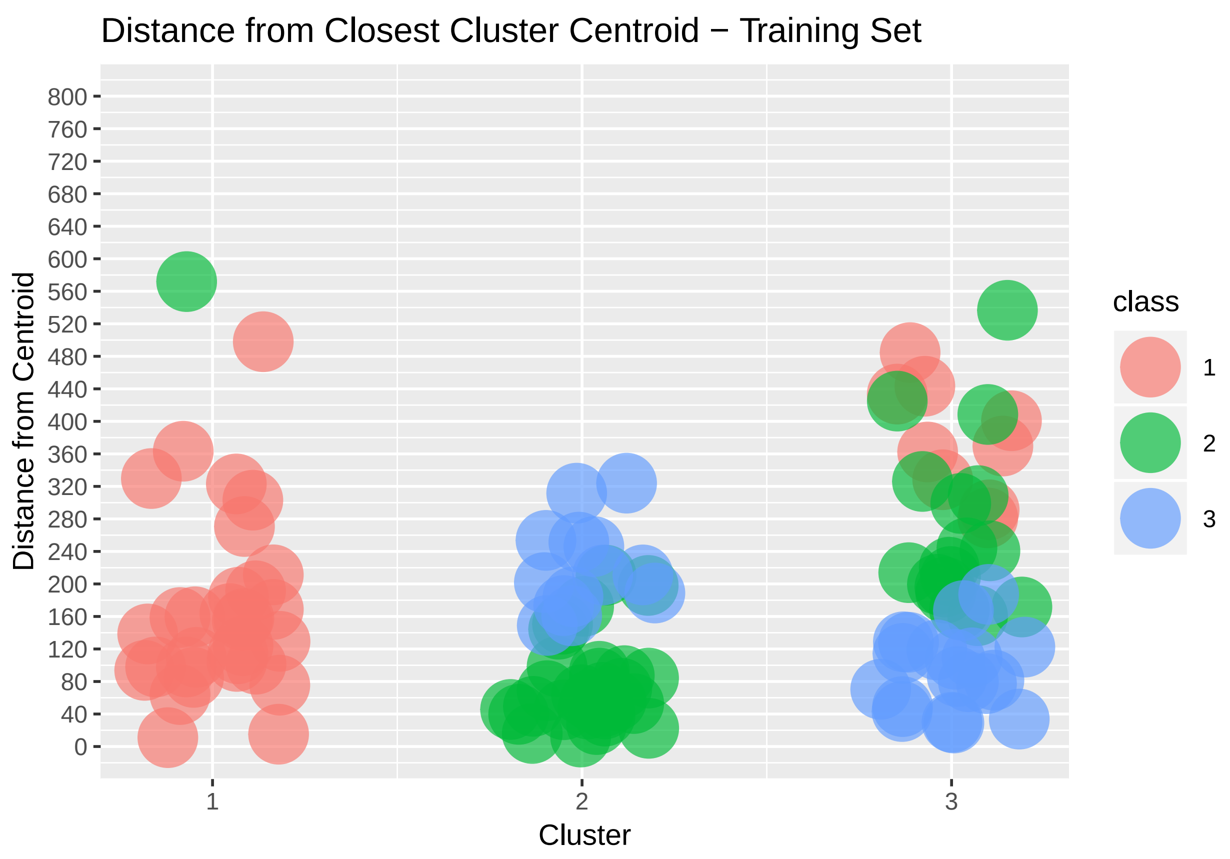
```r
# Assigning accuracies
acc_train[3] <- mean(train_data[,1] == chosen_pred1train_manh$cluster)
acc_test[3] <- mean(test_data[,1] == chosen_pred1test_manh$cluster)
L1_manh[[30]]$centers
```

```
##    Alcohol Malic acid   Ash Alcalinity_of_ash Magnesium Total_phenols
## 18  13.760       1.73 2.450              16.8     101.0         2.800
## 92  12.565       1.89 2.295              19.8      88.5         1.905
## 60  12.840       1.81 2.360              20.0     100.0          2.050
##    Flavanoids Nonflavanoid_phenols Proanthocyanins Color_intensity  Hue
## 18       2.91                 0.29            1.86            5.40 1.10
## 92       1.58                 0.34            1.35            3.67 0.98
## 60       1.30                 0.37            1.40            4.50 0.99
##    OD280/OD315 Proline
## 18       2.900    1195
## 92       2.355     480
## 60       2.310     710
```

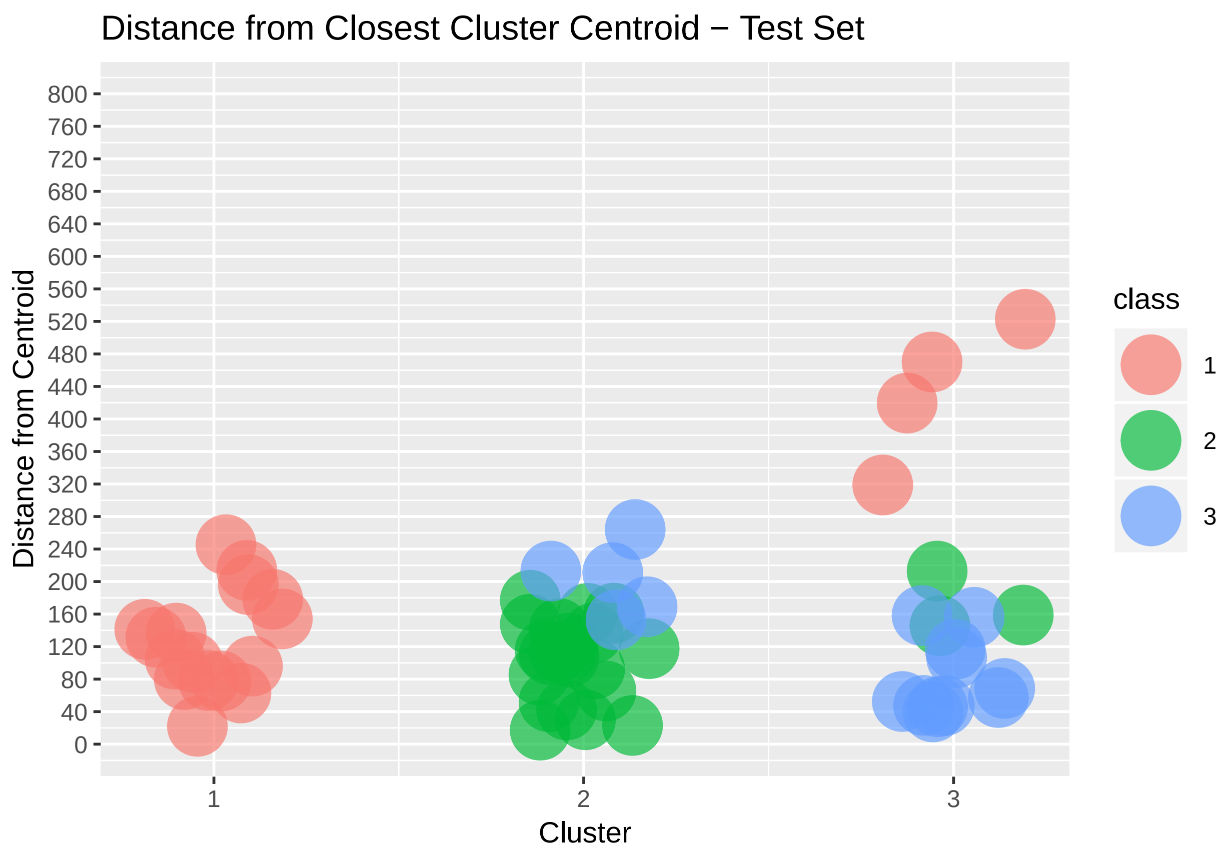
Step 7

Visualizing the results using a jitter plot for the training and testing datasets

```
# Jitter plot for the training set
class1train_raw_manh = subset(train_data, train_data[,1] == 1)
class2train_raw_manh = subset(train_data, train_data[,1] == 2)
class3train_raw_manh = subset(train_data, train_data[,1] == 3)

class1train_raw_manh$sse = apply(class1train_raw_manh[,2:14],
                                 1, function(x)
                                   sum( abs(x - L1_manh[[30]]$centers[1,]) ))
class2train_raw_manh$sse = apply(class2train_raw_manh[,2:14],
                                 1, function(x)
                                   sum( abs(x - L1_manh[[30]]$centers[2,]) ))
class3train_raw_manh$sse = apply(class3train_raw_manh[,2:14],
                                 1, function(x)
                                   sum( abs(x - L1_manh[[30]]$centers[3,]) ))

sse_train_raw_manh = rbind(class1train_raw_manh,
                           class2train_raw_manh, class3train_raw_manh)

sse_train_raw_manh$cluster = jitter(chosen_pred1train_manh$cluster)
sse_train_raw_manh$class = cut(sse_train_raw_manh$class,
                               c(.5,1.5,2.5,3.5), right=FALSE, labels=c(1:3))

jitplot_train_raw_manh = qplot(cluster, sse, data = sse_train_raw_manh,
                               color=class, alpha = I(2/3), size = I(10))
jitplot_train_raw_manh + coord_cartesian(ylim=c(0, 800)) +
  scale_y_continuous(breaks=seq(0, 800, 40)) +
  scale_x_continuous(breaks=seq(1,3,1)) + xlab("Cluster") +
  ylab("Distance from Centroid") +
  ggtitle("Distance from Closest Cluster Centroid - Training Set")
```



```
# Jitter plot for the test set
class1test_raw_manh = subset(test_data, test_data[,1] == 1)
class2test_raw_manh = subset(test_data, test_data[,1] == 2)
class3test_raw_manh = subset(test_data, test_data[,1] == 3)

class1test_raw_manh$sse = apply(class1test_raw_manh[,2:14],
                                1, function(x)
                                  sum( abs(x - L1_manh[[30]]$centers[1,]) ))
class2test_raw_manh$sse = apply(class2test_raw_manh[,2:14],
                                1, function(x)
                                  sum( abs(x - L1_manh[[30]]$centers[2,]) ))
class3test_raw_manh$sse = apply(class3test_raw_manh[,2:14],
                                1, function(x)
                                  sum( abs(x - L1_manh[[30]]$centers[3,]) ))

sse_test_raw_manh = rbind(class1test_raw_manh,
                          class2test_raw_manh, class3test_raw_manh)

sse_test_raw_manh$cluster = jitter(chosen_pred1test_manh$cluster)
sse_test_raw_manh$class = cut(sse_test_raw_manh$class,
                              c(.5,1.5,2.5,3.5), right=FALSE, labels=c(1:3))

jitplot_test_raw_manh = qplot(cluster, sse, data = sse_test_raw_manh,
                              color=class, alpha = I(2/3), size = I(10))
jitplot_test_raw_manh + coord_cartesian(ylim=c(0, 800)) +
  scale_y_continuous(breaks=seq(0, 800, 40)) +
  scale_x_continuous(breaks=seq(1,3,1)) + xlab("Cluster") +
  ylab("Distance from Centroid") +
  ggtitle("Distance from Closest Cluster Centroid - Test Set")
```



Step 8

Scaled Data & Euclidean Distance

```
# Scaling the data
mins = sapply(train_data, min)
ranges = sapply(train_data,function(x)diff(range(x)))
train_set_scaled = as.data.frame(scale(train_data, center = mins, scale = ranges))
test_data_scaled = as.data.frame(scale(test_data, center = mins, scale = ranges))

train_set_scaled[,1] = train_data[,1]
test_data_scaled[,1] = test_data[,1]

L2 = list()
totw2 = list()

for (i in 1:100) {
  set.seed(i)
  L2[[i]] = cclust(as.matrix(train_set_scaled[,2:14]), 3,
                   method = "kmeans", dist = "euclidean")
  totw2[[i]] = sum(L2[[i]]$withinss)
}
```

```
min_ss2 = min(unlist(totw2))

for (i in 1:100){
  if (totw2[[i]] == min_ss2){
    pred_train2 = predict(L2[[i]], newdata = as.matrix(train_set_scaled[,2:14]))
    pred_test2 = predict(L2[[i]], newdata = as.matrix(test_data_scaled[,2:14]))
    # print(i)
    # print(table(train_data[,1],pred_train2$cluster))
    # print(table(test_data[,1],pred_test2$cluster))
  }
}

# Choosing L2[[13]] as the most suitable result
chosen_pred2train = predict(L2[[13]], newdata = as.matrix(train_set_scaled[,2:14]))
chosen_pred2test = predict(L2[[13]], newdata = as.matrix(test_data_scaled[,2:14]))

table(train_set_scaled[,1],chosen_pred2train$cluster)
```

```
##
##      1  2  3
##   1 39  0  0
##   2  5  2 40
##   3  0 32  0
```

```
table(test_data_scaled[,1], chosen_pred2test$cluster)
```

```
##
##      1  2  3
##   1 20  0  0
##   2  1  1 22
##   3  0 16  0
```

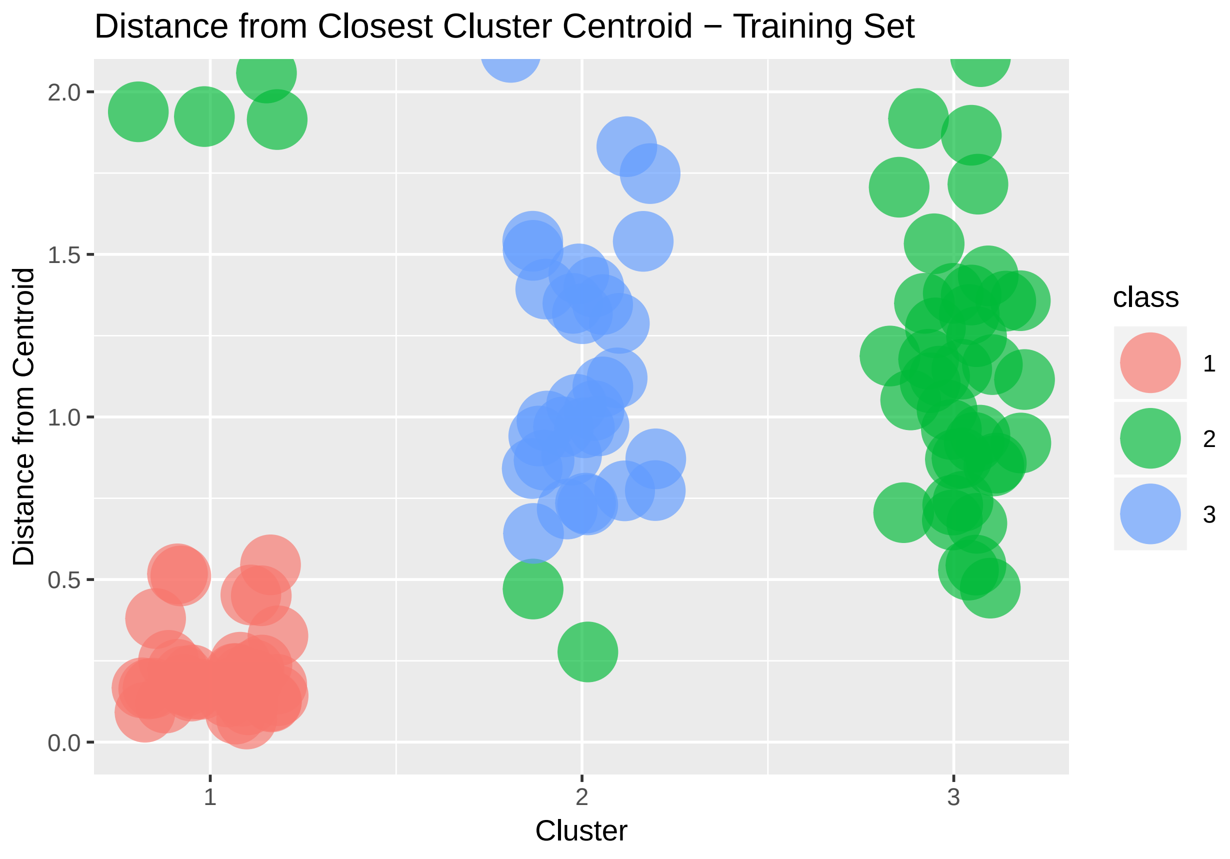
```
# Assigning accuracies
acc_train[2] <- mean(train_set_scaled[,1] == chosen_pred2train$cluster)
acc_test[2] <- mean(test_data_scaled[,1] == chosen_pred2test$cluster)

L2[[13]]$centers
```

```
##      Alcohol Malic acid       Ash Alcalinity_of_ash Magnesium
## 15 0.6341688  0.1991342 0.5981183         0.3679709 0.3957510
## 87 0.4453912  0.4992997 0.5654649         0.5509400 0.3069054
## 80 0.2132399  0.1641234 0.4467742         0.4469072 0.2698370
##    Total_phenols Flavanoids Nonflavanoid_phenols Proanthocyanins
## 15     0.6348590  0.7198658            0.3156089       0.5185303
## 87     0.2609141  0.1468950            0.6098779       0.2589670
## 80     0.4006098  0.4149721            0.4202830       0.3771777
##    Color_intensity       Hue OD280/OD315   Proline
## 15       0.3020144 0.6534677   0.6841492 0.5787836
## 87       0.4504493 0.2204366   0.1471666 0.2358605
## 80       0.1141208 0.6622680   0.5263736 0.1870007
```

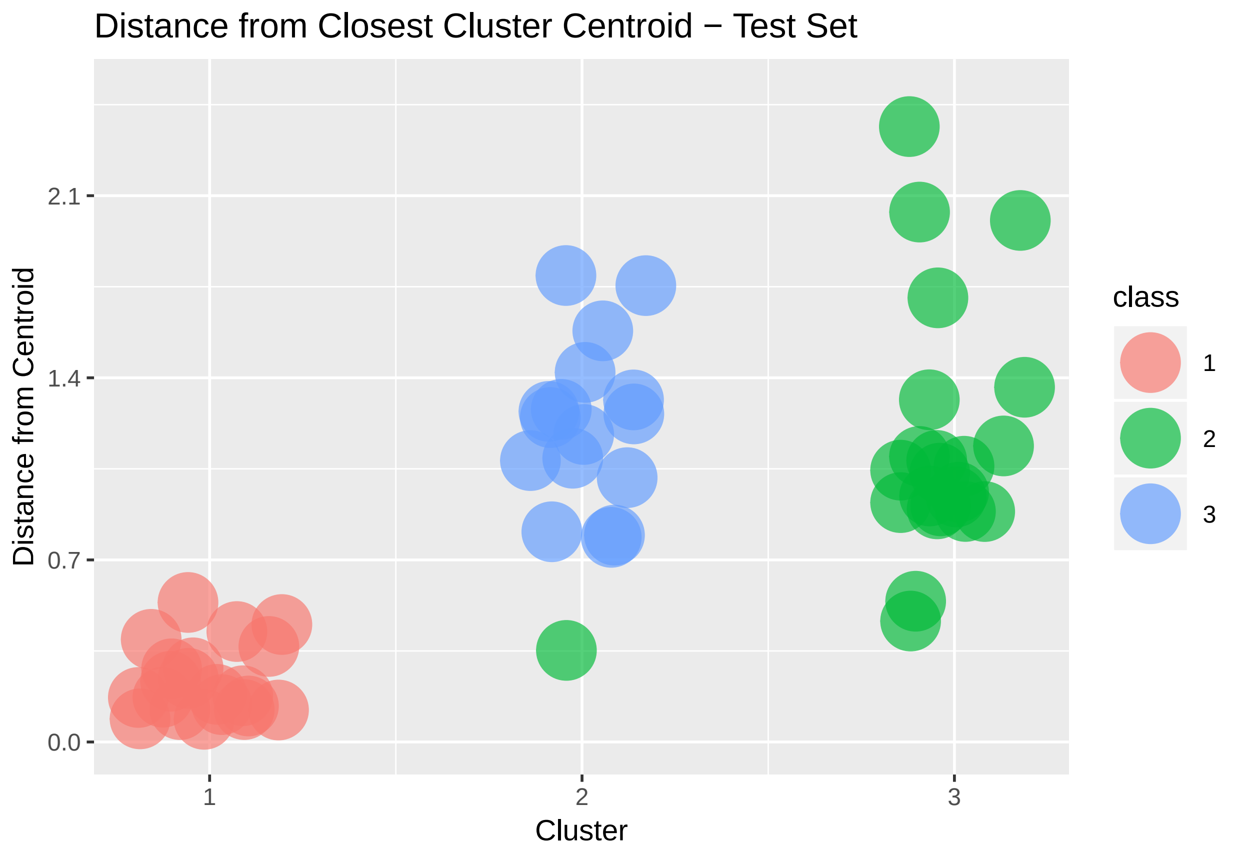
Step 9

Visualizing the results using a jitter plot for the training and testing datasets

```
# Jitter plot for the training set
class1train = subset(train_set_scaled, train_set_scaled[,1] == 1)
class2train = subset(train_set_scaled, train_set_scaled[,1] == 2)
class3train = subset(train_set_scaled, train_set_scaled[,1] == 3)

class1train$sse = apply(class1train[,2:14], 1, function(x)
  sum( (x-L2[[13]]$centers[1,])^2 ))
class2train$sse = apply(class2train[,2:14], 1, function(x)
  sum( (x-L2[[13]]$centers[2,])^2 ))
class3train$sse = apply(class3train[,2:14], 1, function(x)
  sum( (x-L2[[13]]$centers[3,])^2 ))

sse_train = rbind(class1train, class2train, class3train)

sse_train$cluster = jitter(chosen_pred2train$cluster)
sse_train$class = cut(sse_train$class, c(.5,1.5,2.5,3.5),
                      right=FALSE, labels=c(1:3))

jitplot_train = qplot(cluster, sse, data = sse_train, color=class,
                      alpha = I(2/3), size = I(10))
jitplot_train + coord_cartesian(ylim=c(0, 2)) +
  scale_y_continuous(breaks=seq(0, 2, .5)) +
  scale_x_continuous(breaks=seq(1,3,1)) + xlab("Cluster") +
  ylab("Distance from Centroid") +
  ggtitle("Distance from Closest Cluster Centroid - Training Set")
```



```r
# Jitter plot for the testing set
class1test = subset(test_data_scaled, test_data_scaled[,1] == 1)
class2test = subset(test_data_scaled, test_data_scaled[,1] == 2)
class3test = subset(test_data_scaled, test_data_scaled[,1] == 3)

class1test$sse = apply(class1test[,2:14], 1, function(x)
  sum( (x-L2[[13]]$centers[1,])^2 ))
class2test$sse = apply(class2test[,2:14], 1, function(x)
  sum( (x-L2[[13]]$centers[2,])^2 ))
class3test$sse = apply(class3test[,2:14], 1, function(x)
  sum( (x-L2[[13]]$centers[3,])^2 ))

sse_test = rbind(class1test, class2test, class3test)

sse_test$cluster = jitter(chosen_pred2test$cluster)
sse_test$class = cut(sse_test$class, c(.5,1.5,2.5,3.5),
                     right=FALSE, labels=c(1:3))

jitplot_test = qplot(cluster, sse, data = sse_test,
                     color=class, alpha = I(2/3), size = I(10))
jitplot_test + coord_cartesian(ylim=c(0, 2.5)) +
  scale_y_continuous(breaks=seq(0, 5, .7)) +
  scale_x_continuous(breaks=seq(1,3,1)) + xlab("Cluster") +
  ylab("Distance from Centroid") +
  ggtitle("Distance from Closest Cluster Centroid - Test Set")
```



Step 10

Scaled Data & Manhattan Distance

```
L2_manh = list()
totw2_manh = list()

for (i in 1:100) {
  set.seed(i)
  L2_manh[[i]] = cclust(as.matrix(train_set_scaled[,2:14]), 3,
                        method = "kmeans", dist = "manhattan")
  totw2_manh[[i]] = sum(L2_manh[[i]]$withinss)
}

min_ss2_manh = min(unlist(totw2_manh))

for (i in 1:100){
  if (totw2_manh[[i]] == min_ss2_manh){
    pred_train2_manh = predict(L2_manh[[i]],
                               newdata = as.matrix(train_set_scaled[,2:14]))
    pred_test2_manh = predict(L2_manh[[i]],
                              newdata = as.matrix(test_data_scaled[,2:14]))
    # print(i)
    # print(table(train_data[,1],pred_train2_manh$cluster))
    # print(table(test_data[,1],pred_test2_manh$cluster))
  }
}

chosen_pred2train_manh = predict(L2_manh[[4]],
                                 newdata = as.matrix(train_set_scaled[,2:14]))
chosen_pred2test_manh = predict(L2_manh[[4]],
                                newdata = as.matrix(test_data_scaled[,2:14]))

table(train_set_scaled[,1], chosen_pred2train_manh$cluster)
```

```
##
##      1  2  3
##   1 39  0  0
##   2  3  2 42
##   3  0 32  0
```

```
table(test_data_scaled[,1], chosen_pred2test_manh$cluster)
```

```
##
##      1  2  3
##   1 20  0  0
##   2  2  1 21
##   3  0 16  0
```

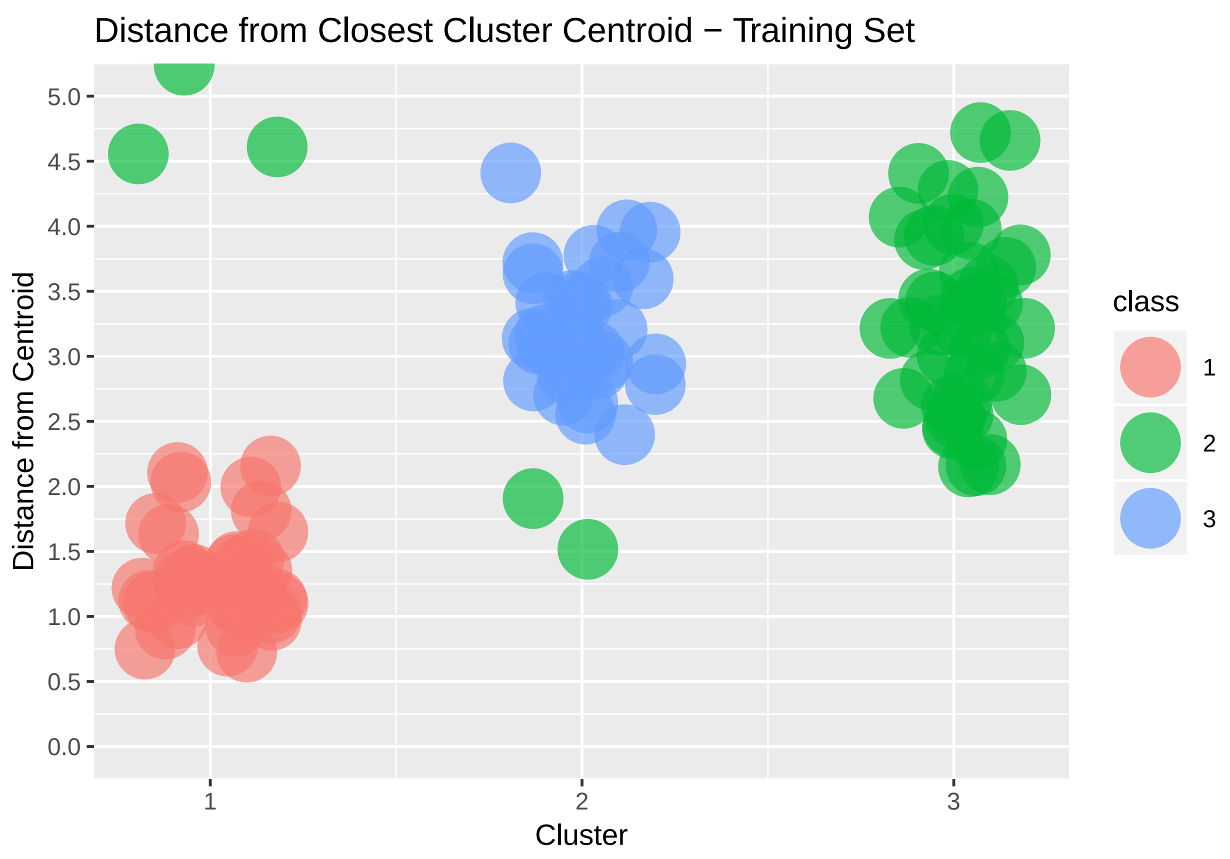
```
# Assigning accuracies
acc_train[4] <- mean(train_set_scaled[,1] == chosen_pred2train_manh$cluster)
acc_test[4] <- mean(test_data_scaled[,1] == chosen_pred2test_manh$cluster)

L2_manh[[4]]$centers
```

```
##      Alcohol Malic acid       Ash Alcalinity_of_ash Magnesium
## 36  0.6542056  0.1850649 0.6075269         0.3350515 0.3478261
## 149 0.4267913  0.5010823 0.5483871         0.5360825 0.2934783
## 144 0.2087227  0.1352814 0.4704301         0.4278351 0.2065217
##     Total_phenols Flavanoids Nonflavanoid_phenols Proanthocyanins
## 36      0.6341463  0.7130919            0.3018868       0.5052265
## 149     0.2421603  0.1197772            0.6981132       0.2299652
## 144     0.3919861  0.4178273            0.3773585       0.3658537
##     Color_intensity       Hue OD280/OD315   Proline
## 36        0.2984014 0.6340206   0.6941392 0.5720399
## 149       0.3619005 0.1907216   0.1227106 0.2296719
## 144       0.1047957 0.6752577   0.5457875 0.1690442
```

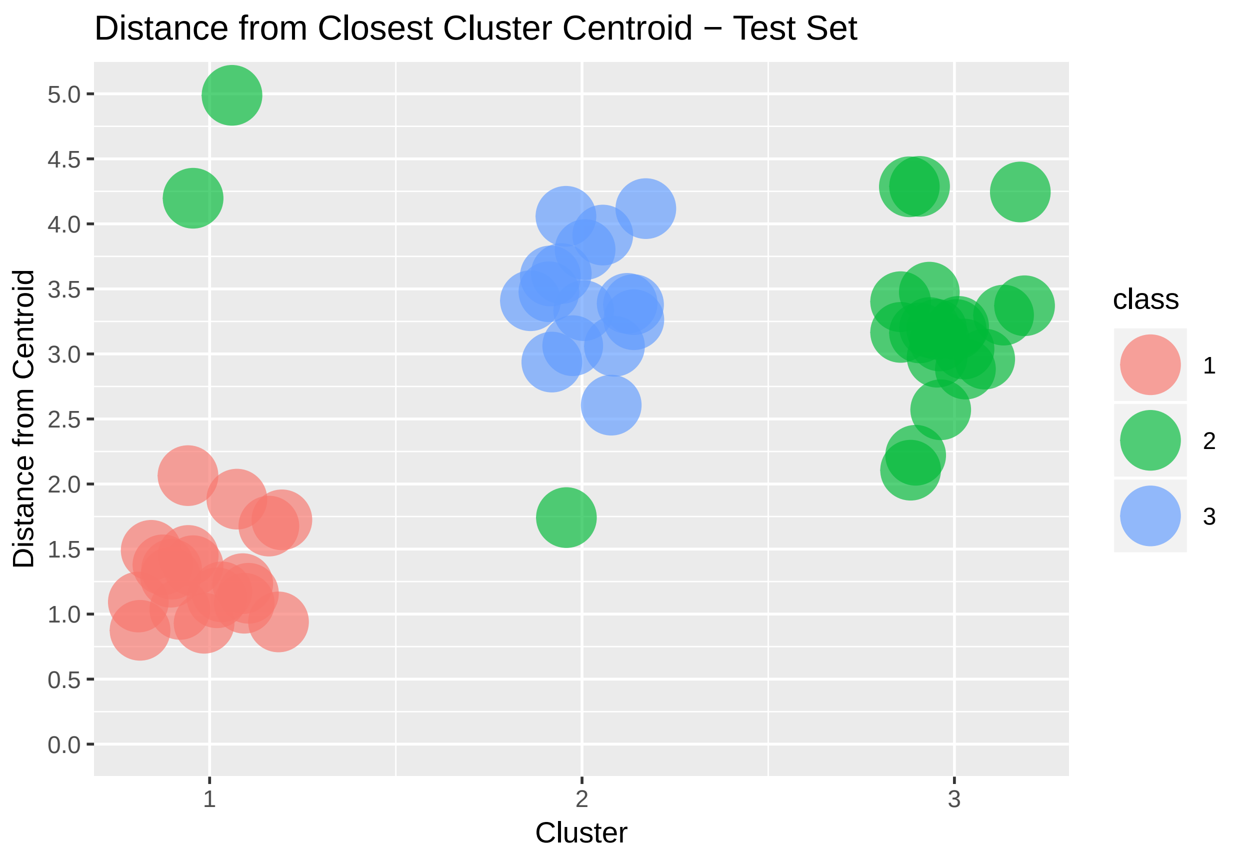
Step 11

Visualizing the results using a jitter plot for the training and testing datasets

```
# Jitter plot for the training set
class1train_scaled_manh = subset(train_set_scaled, train_set_scaled[,1] == 1)
class2train_scaled_manh = subset(train_set_scaled, train_set_scaled[,1] == 2)
class3train_scaled_manh = subset(train_set_scaled, train_set_scaled[,1] == 3)

class1train_scaled_manh$sse = apply(class1train[,2:14],
                                    1, function(x)
                                      sum( abs(x - L2_manh[[4]]$centers[1,]) ))
class2train_scaled_manh$sse = apply(class2train[,2:14],
                                    1, function(x)
                                      sum( abs(x - L2_manh[[4]]$centers[2,]) ))
class3train_scaled_manh$sse = apply(class3train[,2:14],
                                    1, function(x)
                                      sum( abs(x - L2_manh[[4]]$centers[3,]) ))

sse_train_scaled_manh = rbind(class1train_scaled_manh, class2train_scaled_manh,
                              class3train_scaled_manh)

sse_train_scaled_manh$cluster = jitter(chosen_pred2train_manh$cluster)
sse_train_scaled_manh$class = cut(sse_train_scaled_manh$class,
                                  c(.5,1.5,2.5,3.5), right=FALSE, labels=c(1:3))

jitplot_train_scaled_manh = qplot(cluster, sse, data = sse_train_scaled_manh,
                                  color=class, alpha = I(2/3), size = I(10))
jitplot_train_scaled_manh + coord_cartesian(ylim=c(0, 5)) +
  scale_y_continuous(breaks=seq(0, 5, .5)) +
  scale_x_continuous(breaks=seq(1,3,1)) + xlab("Cluster") +
  ylab("Distance from Centroid") +
  ggtitle("Distance from Closest Cluster Centroid - Training Set")
```



```
# Jitter plot for the testing set
class1test_scaled_manh = subset(test_data_scaled, test_data_scaled[,1] == 1)
class2test_scaled_manh = subset(test_data_scaled, test_data_scaled[,1] == 2)
class3test_scaled_manh = subset(test_data_scaled, test_data_scaled[,1] == 3)

class1test_scaled_manh$sse = apply(class1test_scaled_manh[,2:14],
                                   1, function(x)
                                     sum( abs(x-L2_manh[[4]]$centers[1,]) ))
class2test_scaled_manh$sse = apply(class2test_scaled_manh[,2:14],
                                   1, function(x)
                                     sum( abs(x-L2_manh[[4]]$centers[2,]) ))
class3test_scaled_manh$sse = apply(class3test_scaled_manh[,2:14],
                                   1, function(x)
                                     sum( abs(x-L2_manh[[4]]$centers[3,]) ))

sse_test_scaled_manh = rbind(class1test_scaled_manh,
                             class2test_scaled_manh, class3test_scaled_manh)

sse_test_scaled_manh$cluster = jitter(chosen_pred2test_manh$cluster)
sse_test_scaled_manh$class = cut(sse_test_scaled_manh$class,
                                 c(.5,1.5,2.5,3.5), right=FALSE, labels=c(1:3))

jitplot_test_scaled_manh = qplot(cluster, sse, data = sse_test_scaled_manh,
                                 color=class, alpha = I(2/3), size = I(10))
jitplot_test_scaled_manh + coord_cartesian(ylim=c(0, 5)) +
  scale_y_continuous(breaks=seq(0, 5, .5)) +
  scale_x_continuous(breaks=seq(1,3,1)) + xlab("Cluster") +
  ylab("Distance from Centroid") +
  ggtitle("Distance from Closest Cluster Centroid - Test Set")
```



Step 12

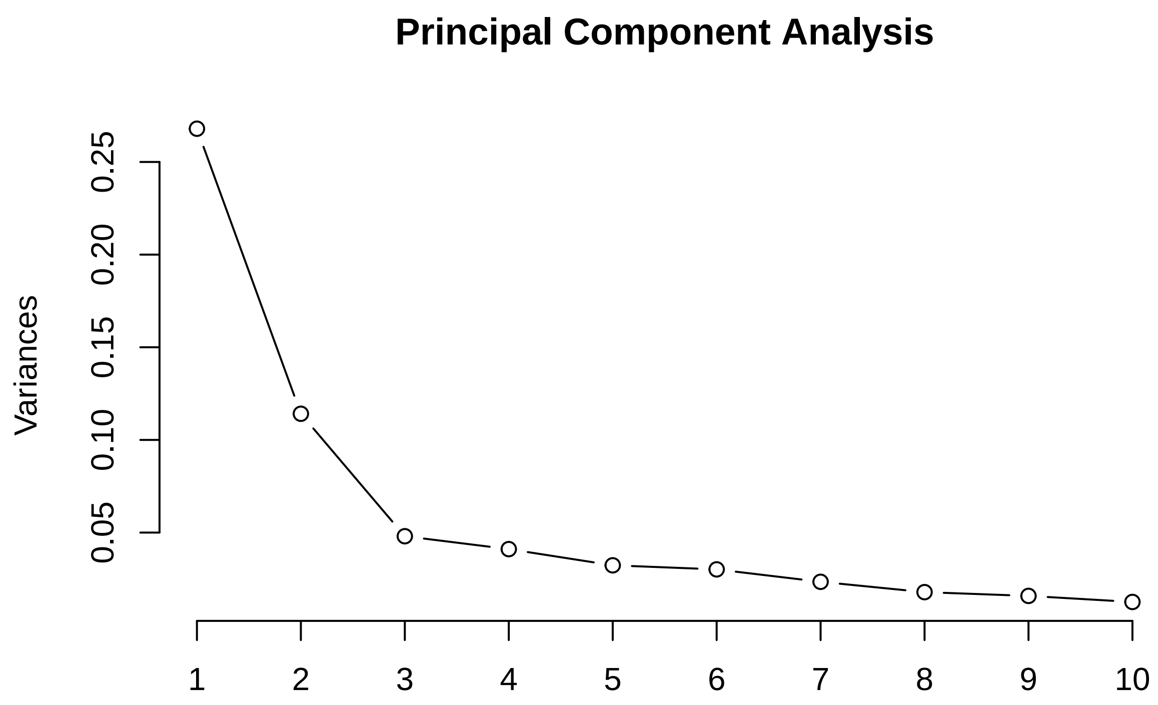
Principal Component Analysis

Summarizing the data and plotting the PCA

```
# PCA on the trained, scaled dataset
A1 = prcomp(train_set_scaled[,2:14])

# Summary of the results
summary(A1)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4     PC5     PC6
## Standard deviation     0.5176 0.3378 0.21917 0.20268 0.17981 0.17362
## Proportion of Variance 0.4296 0.1829 0.07701 0.06586 0.05184 0.04833
## Cumulative Proportion  0.4296 0.6125 0.68950 0.75536 0.80719 0.85552
##                            PC7     PC8     PC9    PC10    PC11    PC12
## Standard deviation     0.15309 0.13365 0.12580 0.11210 0.09135 0.08414
## Proportion of Variance 0.03757 0.02864 0.02537 0.02015 0.01338 0.01135
## Cumulative Proportion  0.89309 0.92173 0.94710 0.96725 0.98062 0.99197
##                           PC13
## Standard deviation     0.07075
## Proportion of Variance 0.00803
## Cumulative Proportion  1.00000
```

```
plot(A1, type="l", main = "Principal Component Analysis")
```



Step 13

Running the kmeans algorithm and displaying the best clustering

```
# The training data is going to be the first two PCs
train.data = data.frame(A1$x)
train.data = train.data[,1:2]
train.data$class = train_data$class

# Testing the data
test.data = predict(A1, newdata = test_data_scaled[,2:14])
test.data = as.data.frame(test.data)
test.data = test.data[,1:2]
test.data$class = test_data$class

L4 = list()
totw4 = list()

for (i in 1:100) {
  set.seed(i)
  L4[[i]] = cclust(as.matrix(train.data)[,1:2], 3,
                   method = "kmeans", dist = "euclidean")
  totw4[[i]] = sum(L4[[i]]$withinss)
}

min_ss4 = min(unlist(totw4))

for (i in 1:100){
  if (totw4[[i]] == min_ss4){
    pred_train4 = predict(L4[[i]], newdata = as.matrix(train.data)[,1:2])
    pred_test4 = predict(L4[[i]], newdata = as.matrix(test.data)[,1:2])
    # print(i)
    # print(table(train_data[,1],pred_train4$cluster))
    # print(table(test_data[,1],pred_test4$cluster))
  }
}

# Choosing L4[[3]]
chosen_pred4train = predict(L4[[3]], newdata = as.matrix(train.data)[,1:2])
chosen_pred4test = predict(L4[[3]], newdata = as.matrix(test.data)[,1:2])

table(train_data[,1],chosen_pred4train$cluster)
```

```
##
##       1  2  3
##    1 39  0  0
##    2  5 40  2
##    3  0  0 32
```

```
table(test_data[,1], chosen_pred4test$cluster)
```

```
##
##       1  2  3
##    1 20  0  0
##    2  1 22  1
##    3  0  0 16
```

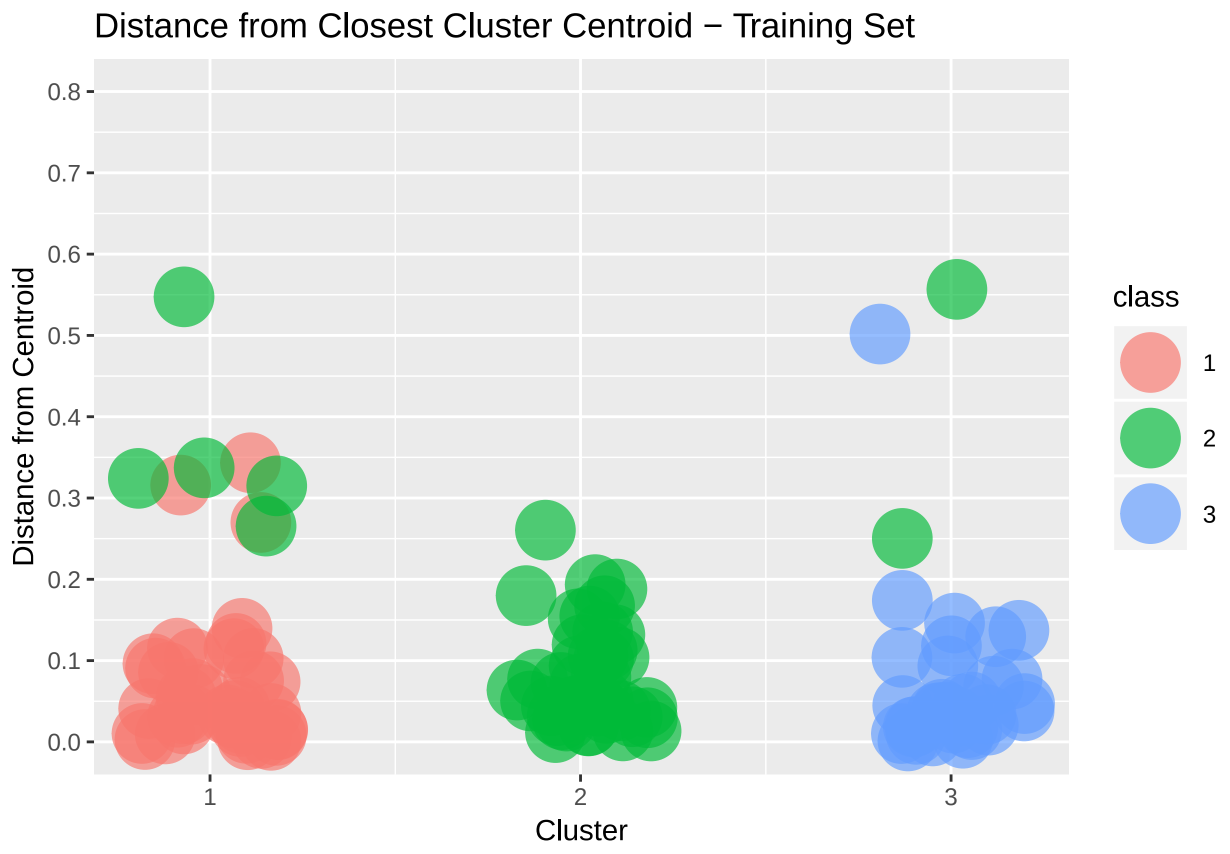
```
# Assigning accuracies
acc_train[5] <- mean(train_data[,1] == chosen_pred4train$cluster)
acc_test[5] <- mean(test_data[,1] == chosen_pred4test$cluster)

L4[[3]]$centers
```

```
##             PC1        PC2
## 2   -0.52448105 -0.1738982
## 85   0.01919093  0.3857910
## 131  0.65616262 -0.2288271
```

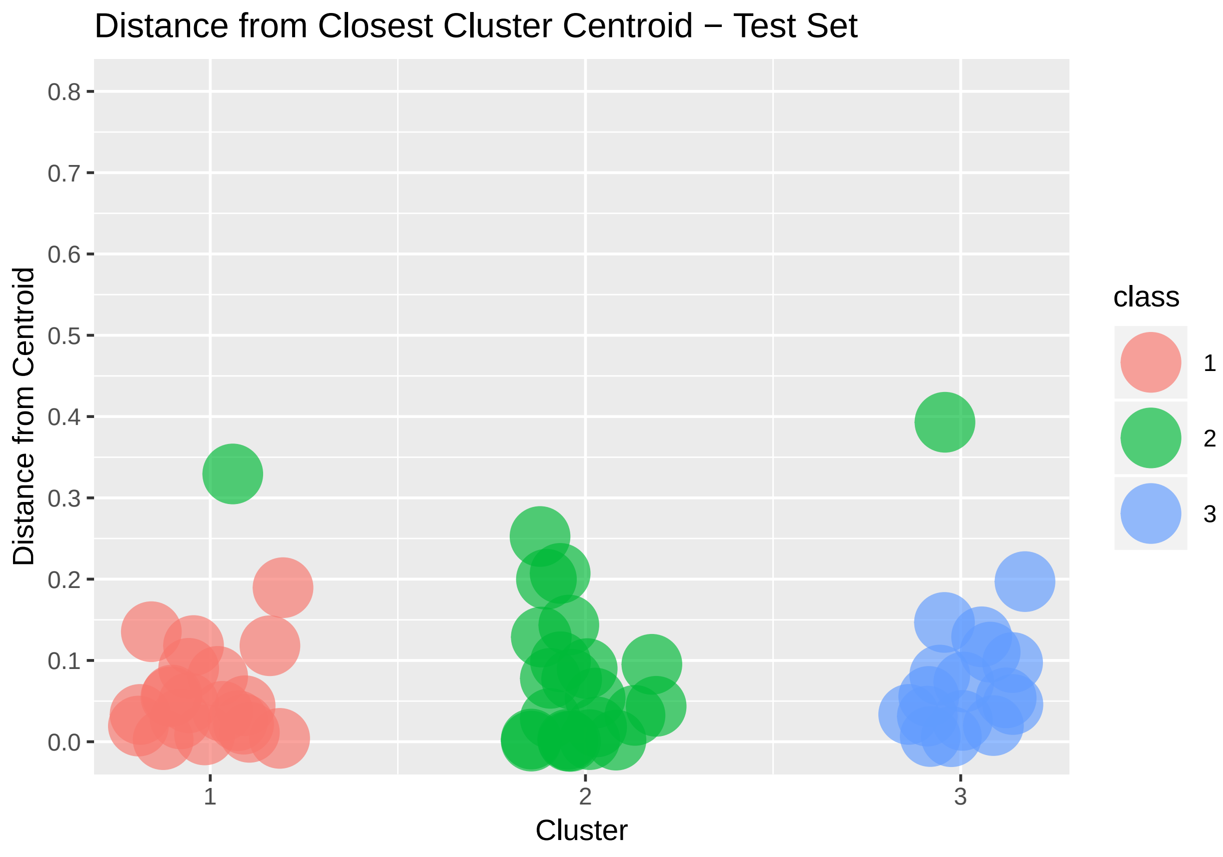
Step 14

Visualizing the results using a jitter plot for the training and testing datasets

```
# Jitter plot for the training set
class1train_pca = subset(train.data, train.data[,3] == 1)
class2train_pca = subset(train.data, train.data[,3] == 2)
class3train_pca = subset(train.data, train.data[,3] == 3)

class1train_pca$sse = apply(class1train_pca[,c(1,2)], 1,
                            function(x) sum( (x-L4[[3]]$centers[1,])^2 ))
class2train_pca$sse = apply(class2train_pca[,c(1,2)], 1,
                            function(x) sum( (x-L4[[3]]$centers[2,])^2 ))
class3train_pca$sse = apply(class3train_pca[,c(1,2)], 1,
                            function(x) sum( (x-L4[[3]]$centers[3,])^2 ))

sse_train_pca = rbind(class1train_pca, class2train_pca, class3train_pca)

sse_train_pca$cluster = jitter(chosen_pred4train$cluster)
sse_train_pca$class = cut(sse_train_pca$class, c(.5,1.5,2.5,3.5),
                          right=FALSE, labels=c(1:3))

jitplot_train_pca = qplot(cluster, sse, data = sse_train_pca,
                          color=class, alpha = I(2/3), size = I(10))
jitplot_train_pca + coord_cartesian(ylim=c(0, .8)) +
  scale_y_continuous(breaks=seq(0, .8, .1)) +
  scale_x_continuous(breaks=seq(1,3,1)) + xlab("Cluster") +
  ylab("Distance from Centroid") +
  ggtitle("Distance from Closest Cluster Centroid - Training Set")
```



```
# Jitter plot for the testing set
class1test_pca = subset(test.data, test.data[,3] == 1)
class2test_pca = subset(test.data, test.data[,3] == 2)
class3test_pca = subset(test.data, test.data[,3] == 3)

class1test_pca$sse = apply(class1test_pca[,c(1,2)], 1,
                           function(x) sum( (x-L4[[3]]$centers[1,])^2 ))
class2test_pca$sse = apply(class2test_pca[,c(1,2)], 1,
                           function(x) sum( (x-L4[[3]]$centers[2,])^2 ))
class3test_pca$sse = apply(class3test_pca[,c(1,2)], 1,
                           function(x) sum( (x-L4[[3]]$centers[3,])^2 ))

sse_test_pca = rbind(class1test_pca, class2test_pca, class3test_pca)

sse_test_pca$cluster = jitter(chosen_pred4test$cluster)
sse_test_pca$class = cut(sse_test_pca$class, c(.5,1.5,2.5,3.5),
                         right=FALSE, labels=c(1:3))

jitplot_test_pca = qplot(cluster, sse, data = sse_test_pca,
                         color=class, alpha = I(2/3), size = I(10))
jitplot_test_pca + coord_cartesian(ylim=c(0, .8)) +
  scale_y_continuous(breaks=seq(0, .8, .1)) +
  scale_x_continuous(breaks=seq(1,3,1)) + xlab("Cluster") +
  ylab("Distance from Centroid") +
  ggtitle("Distance from Closest Cluster Centroid - Test Set")
```



Step 15

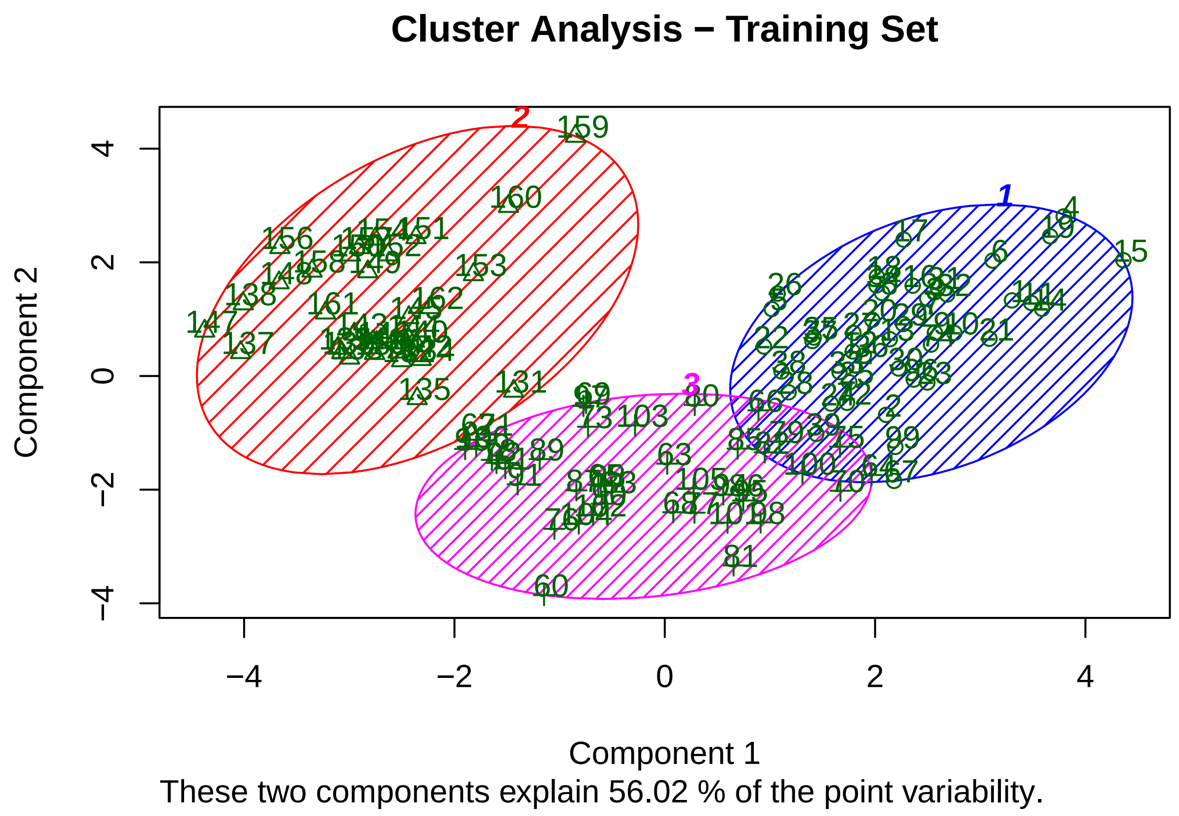
Cluster Analysis

```
library(clusterSim)
```

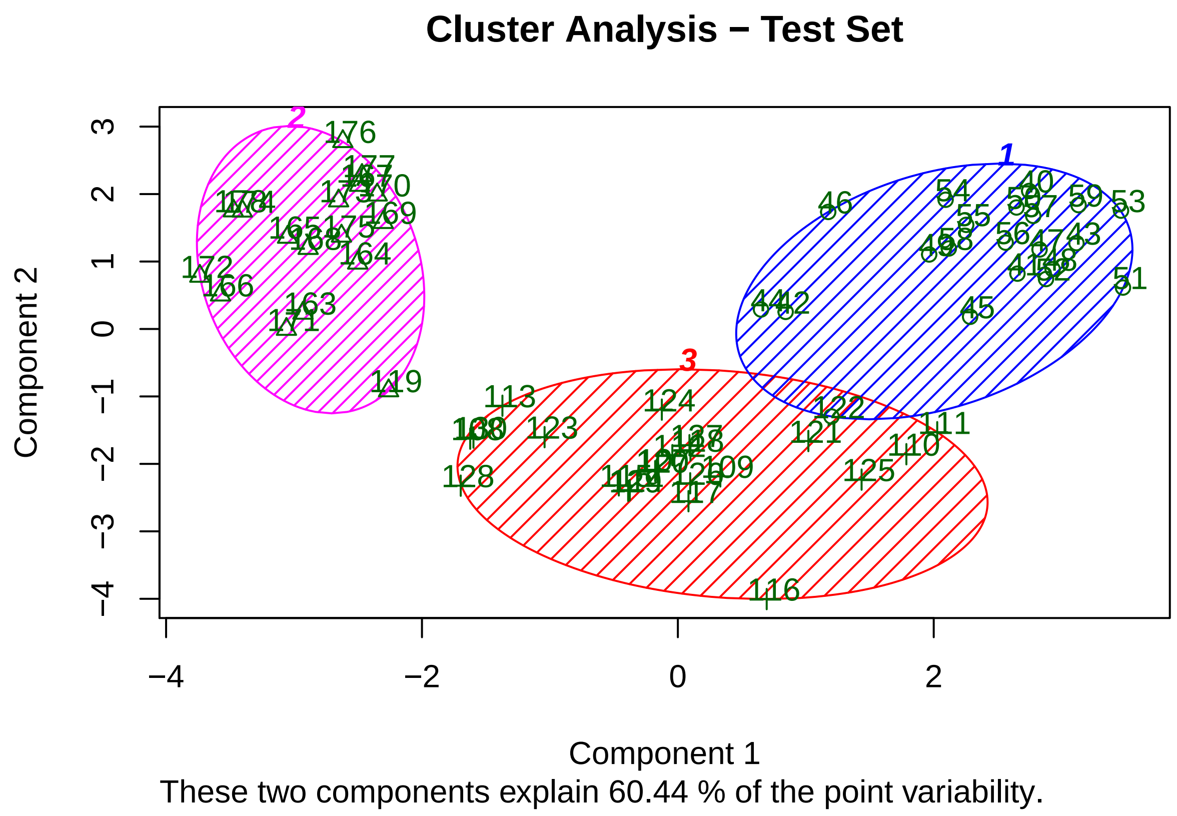
```
## Loading required package: cluster
```

```
## Loading required package: MASS
```

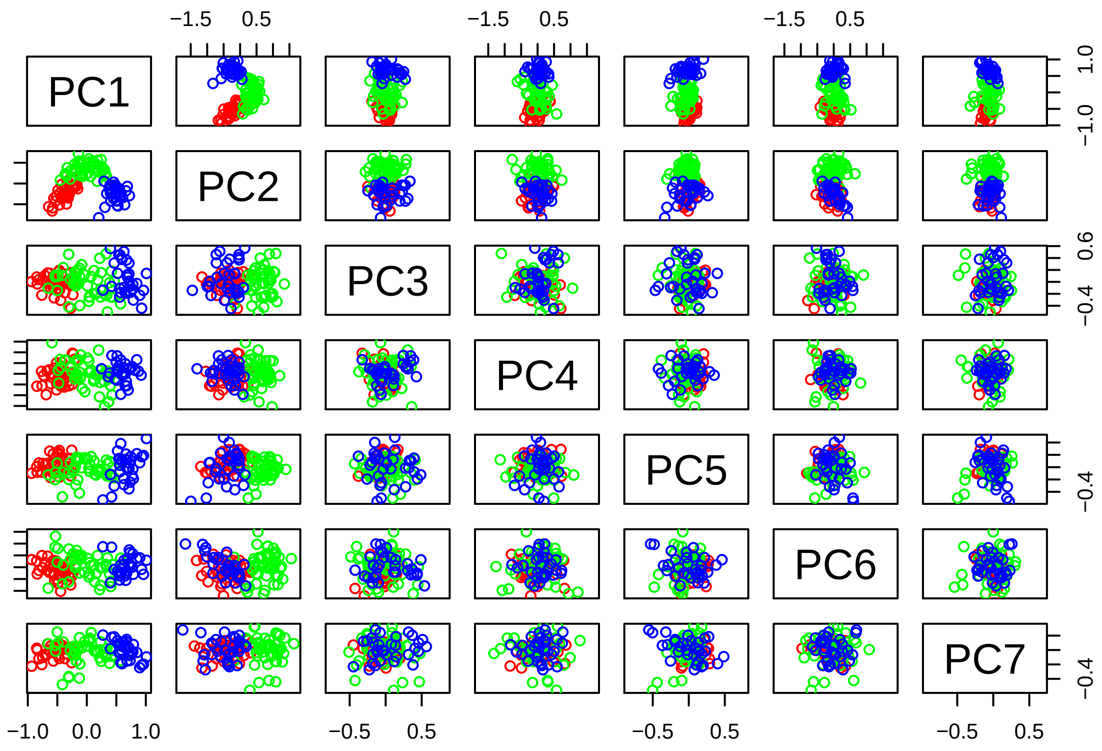
```
library(scatterplot3d)

# Cluster analysis
l = pred_train4$cluster
m = pred_test4$cluster

clusplot(train_set_scaled[,2:14], l, color=T, shade=T, labels = 2,
         lines = 0, main = "Cluster Analysis - Training Set")
```



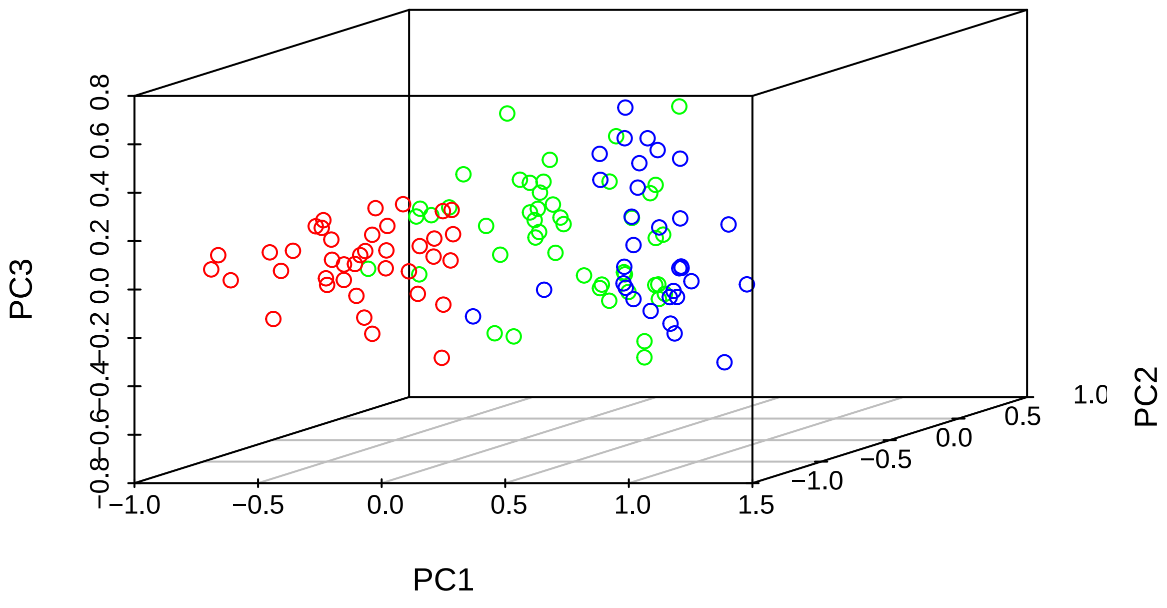
```
clusplot(test_data_scaled[,2:14], m, color=T, shade=T, labels = 2,
         lines = 0, main = "Cluster Analysis - Test Set")
```



```r
# Pairwise plot
pairs(A1$x[,1:7], col = rainbow(3)[train_set_scaled[,1]], asp = 1)
```



```
# 3-D scatterplot
scatterplot3d(A1$x[,c(1,2,3)], color=rainbow(3)[train_set_scaled[,1]])
```



Step 16

Independent Component Analysis

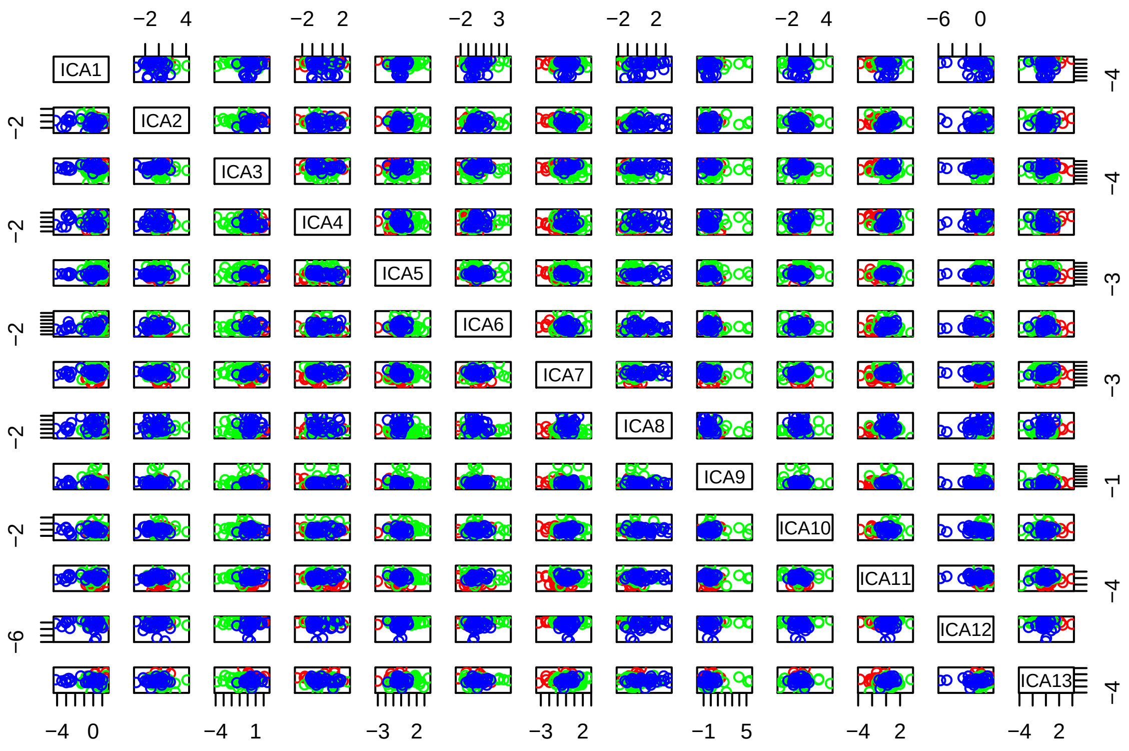
```
library(fastICA)
set.seed(25)

# Preprocessing the training data
preprocessParams = preProcess(train_data[,2:14],
                              method=c("center", "scale", "ica"), n.comp=13)
print(preprocessParams)
```

```
## Created from 118 samples and 13 variables
##
## Pre-processing:
##   - centered (13)
##   - independent component signal extraction (13)
##   - ignored (0)
##   - scaled (13)
##
## ICA used 13 components
```

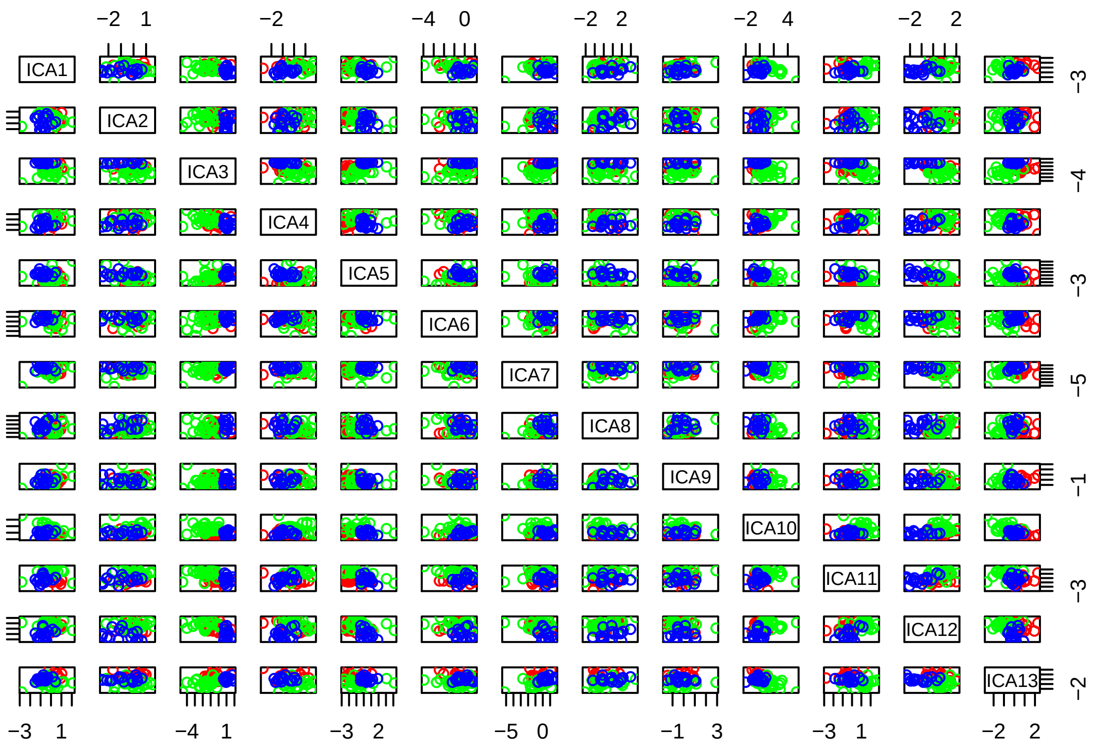
```
transf = predict(preprocessParams, train_data[,2:14])
summary(transf)
```

```
##       ICA1                ICA2                 ICA3
##  Min.   :-4.1655   Min.   :-3.338140   Min.   :-3.89367
##  1st Qu.:-0.2308   1st Qu.:-0.461490   1st Qu.:-0.34429
##  Median : 0.1797   Median : 0.004328   Median : 0.07589
##  Mean   : 0.0000   Mean   : 0.000000   Mean   : 0.00000
##  3rd Qu.: 0.6068   3rd Qu.: 0.541886   3rd Qu.: 0.46206
##  Max.   : 1.4804   Max.   : 4.140119   Max.   : 2.47614
##       ICA4               ICA5                ICA6               ICA7
##  Min.   :-2.5296   Min.   :-3.05922   Min.   :-2.3660   Min.   :-3.30894
##  1st Qu.:-0.8040   1st Qu.:-0.59078   1st Qu.:-0.5523   1st Qu.:-0.57548
##  Median :-0.1049   Median :-0.02111   Median :-0.1079   Median : 0.02371
##  Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.00000
##  3rd Qu.: 0.8752   3rd Qu.: 0.46258   3rd Qu.: 0.3122   3rd Qu.: 0.63174
##  Max.   : 2.5181   Max.   : 3.50362   Max.   : 4.2649   Max.   : 2.76448
##       ICA8               ICA9              ICA10              ICA11
##  Min.   :-1.9546   Min.   :-1.6305   Min.   :-3.12039   Min.   :-3.73406
##  1st Qu.:-0.6398   1st Qu.:-0.4750   1st Qu.:-0.46214   1st Qu.:-0.43463
##  Median :-0.2142   Median :-0.1719   Median :-0.03187   Median : 0.06484
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.00000
##  3rd Qu.: 0.3648   3rd Qu.: 0.2638   3rd Qu.: 0.57263   3rd Qu.: 0.58315
##  Max.   : 3.4338   Max.   : 5.5143   Max.   : 4.60653   Max.   : 3.56476
##      ICA12              ICA13
##  Min.   :-5.7128   Min.   :-3.78518
##  1st Qu.:-0.2599   1st Qu.:-0.52731
##  Median : 0.1165   Median : 0.08488
##  Mean   : 0.0000   Mean   : 0.00000
##  3rd Qu.: 0.6020   3rd Qu.: 0.57162
##  Max.   : 1.5694   Max.   : 3.92867
```

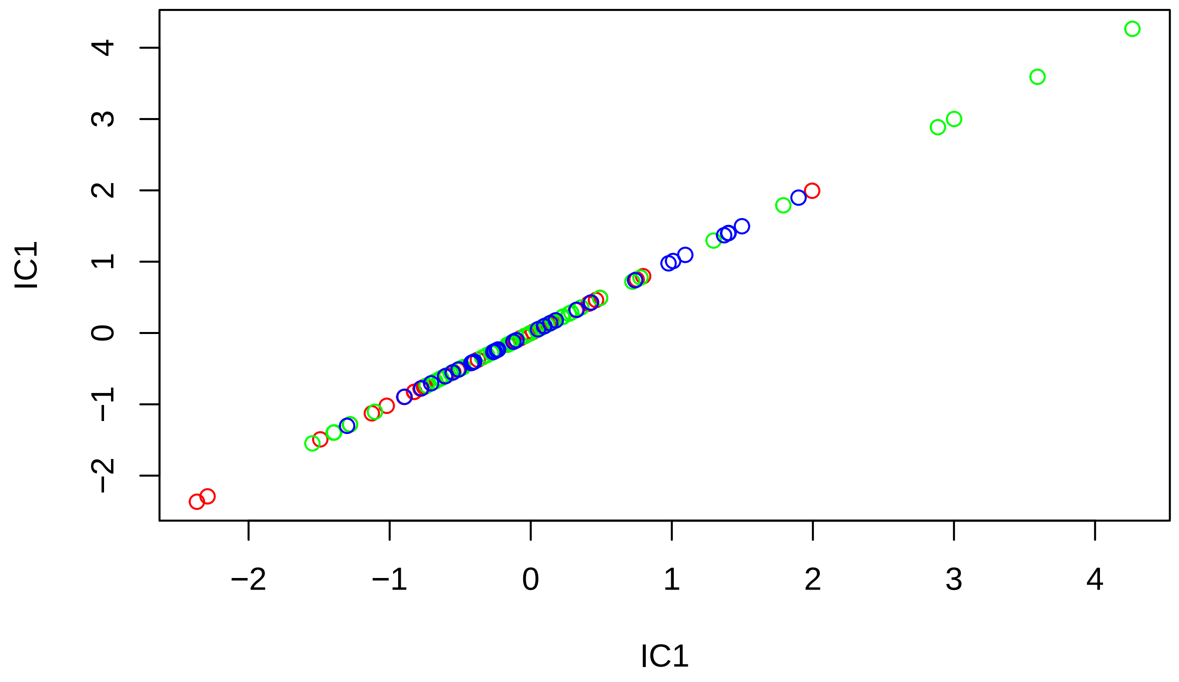
```r
pairs(transf, col = rainbow(3)[train_set_scaled[,1]])
```



```r
test.data2 = predict(preprocessParams, newdata = test_data[,2:14])
pairs(test.data2, col = rainbow(3)[test_data[,1]])
```



```
# Plotting the IC6 against itself
plot(transf[,6], transf[,6], col=rainbow(3)[train_set_scaled[,1]],
     xlab="IC1", ylab="IC1")
```



Step 17

Results of the clustering

```
# Adding a new column "class"
transf$class = train_data$class
test.data2$class = test_data$class

M = transf[,c(6,8,14)]
N = test.data2[,c(6,8,14)]

L4_ica = list()
totw4_ica = list()

for (i in 1:100) {
  set.seed(i)
  L4_ica[[i]] = cclust(as.matrix(M)[,c(1,2)], 3,
                       method = "kmeans", dist = "euclidean")
  totw4_ica[[i]] = sum(L4_ica[[i]]$withinss)
}

min_ss4_ica = min(unlist(totw4_ica))

for (i in 1:100){
  if (totw4_ica[[i]] == min_ss4_ica){
    pred_train4_ica = predict(L4_ica[[i]], newdata = as.matrix(M)[,c(1,2)])
    pred_test4_ica = predict(L4_ica[[i]], newdata = as.matrix(N)[,c(1,2)])
    # print(i)
    # print(table(train_data[,1],pred_train4_ica$cluster))
    # print(table(test_data[,1],pred_test4_ica$cluster))
  }
}

chosen_pred4train_ica = predict(L4_ica[[54]], newdata = as.matrix(M)[,c(1,2)])
chosen_pred4test_ica = predict(L4_ica[[54]], newdata = as.matrix(N)[,c(1,2)])

table(train_data[,1],chosen_pred4train_ica$cluster)
```

```
##
##      1  2  3
##   1 38  0  1
##   2 36  3  8
##   3 13 11  8
```

```
table(test_data[,1], chosen_pred4test_ica$cluster)
```

```
##
##      1  2  3
##   1 14  5  1
##   2 17  7  0
##   3  9  6  1
```

```
# Assigning accuracies
acc_train[6] <- mean(train_data[,1] == chosen_pred4train_ica$cluster)
acc_test[6] <- mean(test_data[,1] == chosen_pred4test_ica$cluster)

L4_ica[[54]]$centers
```

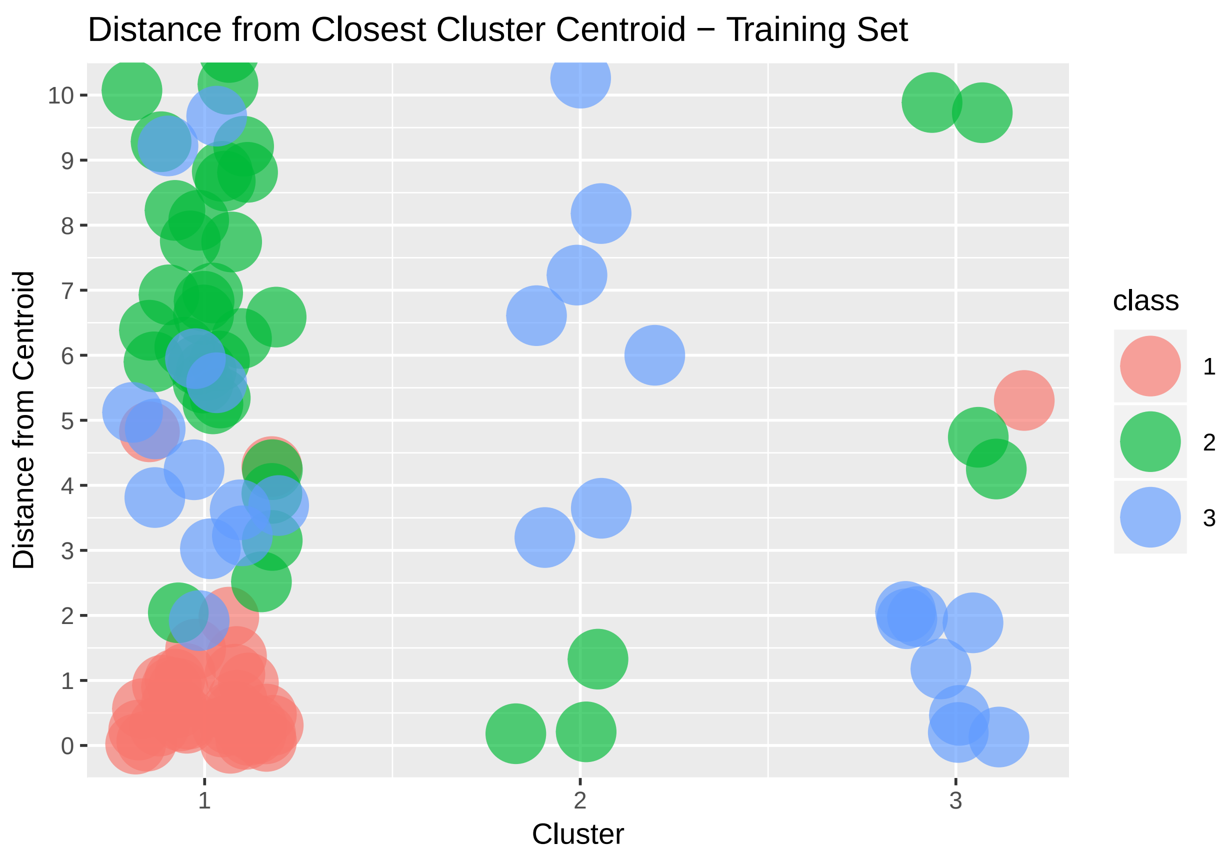
```
##          ICA6        ICA8
## 24 -0.3017335 -0.36322806
## 11 -0.3204543  2.13741126
## 30  1.8080693  0.09865195
```

Step 18

Visualizing the results using a jitter plot for the training and testing datasets

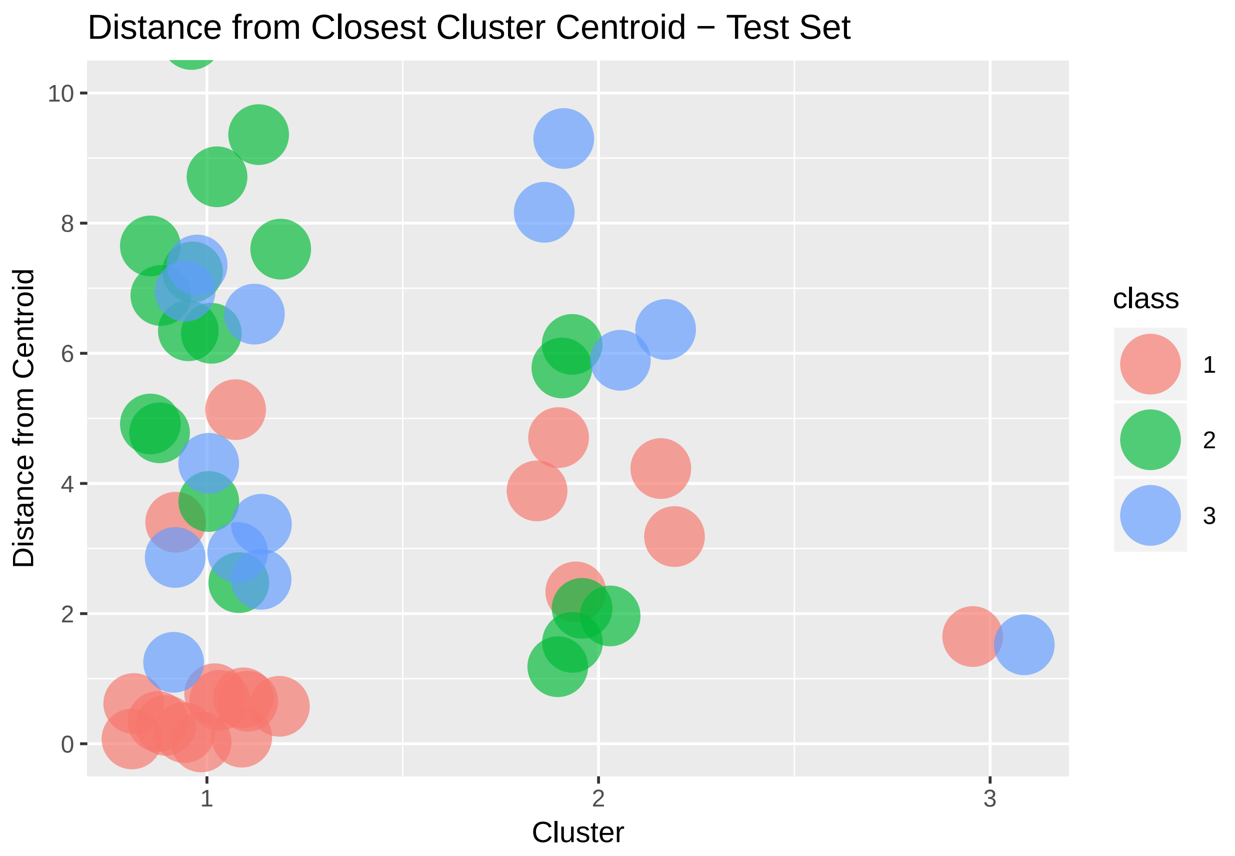
```
# Jitter plot for training dataset
class1train_ica = subset(M, M[,3] == 1)
class2train_ica = subset(M, M[,3] == 2)
class3train_ica = subset(M, M[,3] == 3)

class1train_ica$sse = apply(class1train_ica[,c(1,2)], 1, function(x)
  sum( (x-L4_ica[[54]]$centers[1,])^2 ))
class2train_ica$sse = apply(class2train_ica[,c(1,2)], 1, function(x)
  sum( (x-L4_ica[[54]]$centers[2,])^2 ))
class3train_ica$sse = apply(class3train_ica[,c(1,2)], 1, function(x)
  sum( (x-L4_ica[[54]]$centers[3,])^2 ))

sse_train_ica = rbind(class1train_ica, class2train_ica, class3train_ica)

sse_train_ica$cluster = jitter(chosen_pred4train_ica$cluster)
```

```
sse_train_ica$class = cut(sse_train_ica$class, c(.5,1.5,2.5,3.5),
                          right=FALSE, labels=c(1:3))

jitplot_train_ica = qplot(cluster, sse, data = sse_train_ica,
                          color=class, alpha = I(2/3), size = I(10))
jitplot_train_ica + coord_cartesian(ylim=c(0, 10)) +
  scale_y_continuous(breaks=seq(0, 10, 1)) +
  scale_x_continuous(breaks=seq(1,3,1)) + xlab("Cluster") +
  ylab("Distance from Centroid") +
  ggtitle("Distance from Closest Cluster Centroid - Training Set")
```



```
# Jitter plot for testing data
class1test_ica = subset(N, N[,3] == 1)
class2test_ica = subset(N, N[,3] == 2)
class3test_ica = subset(N, N[,3] == 3)

class1test_ica$sse = apply(class1test_ica[,c(1,2)], 1,
                           function(x) sum( (x-L4_ica[[54]]$centers[1,])^2 ))
class2test_ica$sse = apply(class2test_ica[,c(1,2)], 1,
                           function(x) sum( (x-L4_ica[[54]]$centers[2,])^2 ))
class3test_ica$sse = apply(class3test_ica[,c(1,2)], 1,
                           function(x) sum( (x-L4_ica[[54]]$centers[3,])^2 ))

sse_test_ica = rbind(class1test_ica, class2test_ica, class3test_ica)

sse_test_ica$cluster = jitter(chosen_pred4test_ica$cluster)
```

```
sse_test_ica$class = cut(sse_test_ica$class, c(.5,1.5,2.5,3.5),
                         right=FALSE, labels=c(1:3))

jitplot_test_ica = qplot(cluster, sse, data = sse_test_ica,
                         color=class, alpha = I(2/3), size = I(10))
jitplot_test_ica + coord_cartesian(ylim=c(0, 10)) +
  scale_y_continuous(breaks=seq(0, 10, 2)) +
  scale_x_continuous(breaks=seq(1,3,1)) + xlab("Cluster") +
  ylab("Distance from Centroid") +
  ggtitle("Distance from Closest Cluster Centroid - Test Set")
```



Summary of results

```
sets <- c("Raw data and Euclidean Distance",
          "Scaled data and Euclidean Distance",
          "Raw data and Manhattan Distance",
          "Scaled data and Manhattan Distance",
          "PCA", "ICA")

# Summarizing the results
df <- data.frame(Results = sets, 'Training Set' = acc_train,
                 'Testing Set' = acc_test)
df
```

```
##                                Results Training.Set Testing.Set
## 1     Raw data and Euclidean Distance     0.6355932   0.7833333
## 2 Scaled data and Euclidean Distance    0.3474576   0.3500000
## 3    Raw data and Manhattan Distance    0.6610169   0.8000000
## 4 Scaled data and Manhattan Distance    0.3474576   0.3500000
## 5                                PCA    0.9406780   0.9666667
## 6                                ICA    0.4152542   0.3666667
```