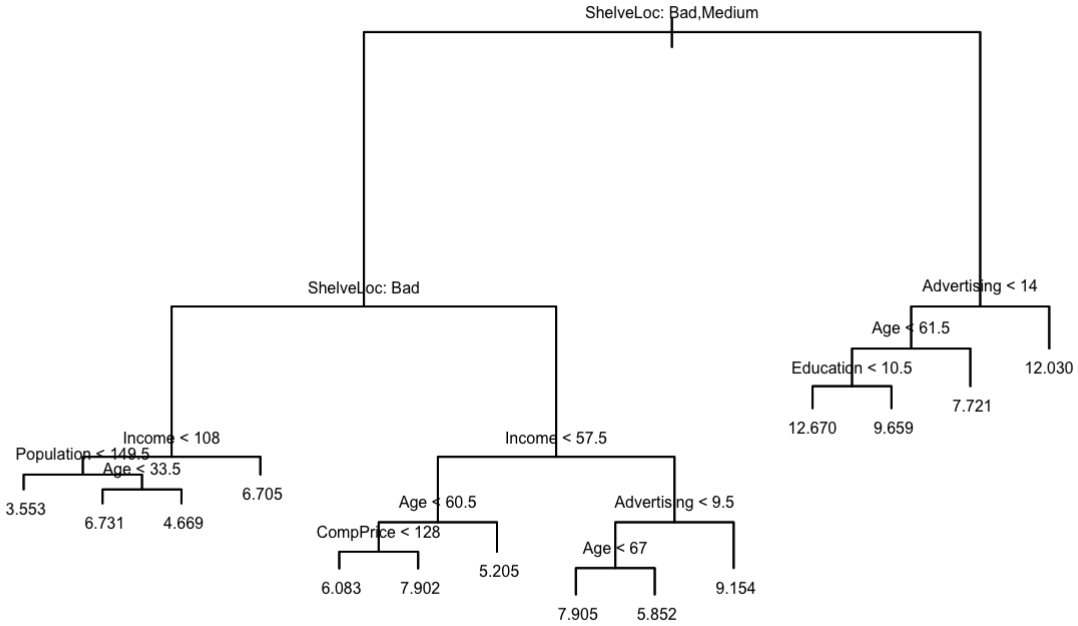
Name: Xin Gao Student ID: 43044879 1(a)

```
library(tree)
train = read.csv("carseatsTrain.csv", header = TRUE)
test = read.csv("carseatsTest.csv", header = TRUE)
train$ShelveLoc = as.factor(train$ShelveLoc)
train$Urban = as.factor(train$Urban)
train$US = as.factor(train$US)
test$ShelveLoc = as.factor(test$ShelveLoc)
test$Urban = as.factor(test$Urban)
test$US = as.factor(test$US)
carseats = tree(Sales~., train)
summary(carseats)
```

```
##
## Regression tree:
## tree(formula = Sales ~ ., data = train)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Income"      "Population"  "Age"         "CompPrice"
## [6] "Advertising" "Education"
## Number of terminal nodes:  14
## Residual mean deviance:  3.717 = 936.6 / 252
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -5.2050 -1.3610 -0.2297  0.0000  1.3120  5.4550
```

```
plot(carseats)
text(carseats, pretty = 0, cex = 0.5)
```



The tree has 14 terminal nodes. “ShelveLoc” “Income” “Population” “Age” “CompPrice” are the most important predictors.

```
tr.pred = predict(carseats, newdata = train)
error.tr = mean((tr.pred - train$Sales)^2)
error.tr # Training MSE
```

```
## [1] 3.521092
```

```
te.pred = predict(carseats, newdata = test)
error.te = mean((te.pred - test$Sales)^2)
error.te # Testing MSE
```

```
## [1] 5.163211
```

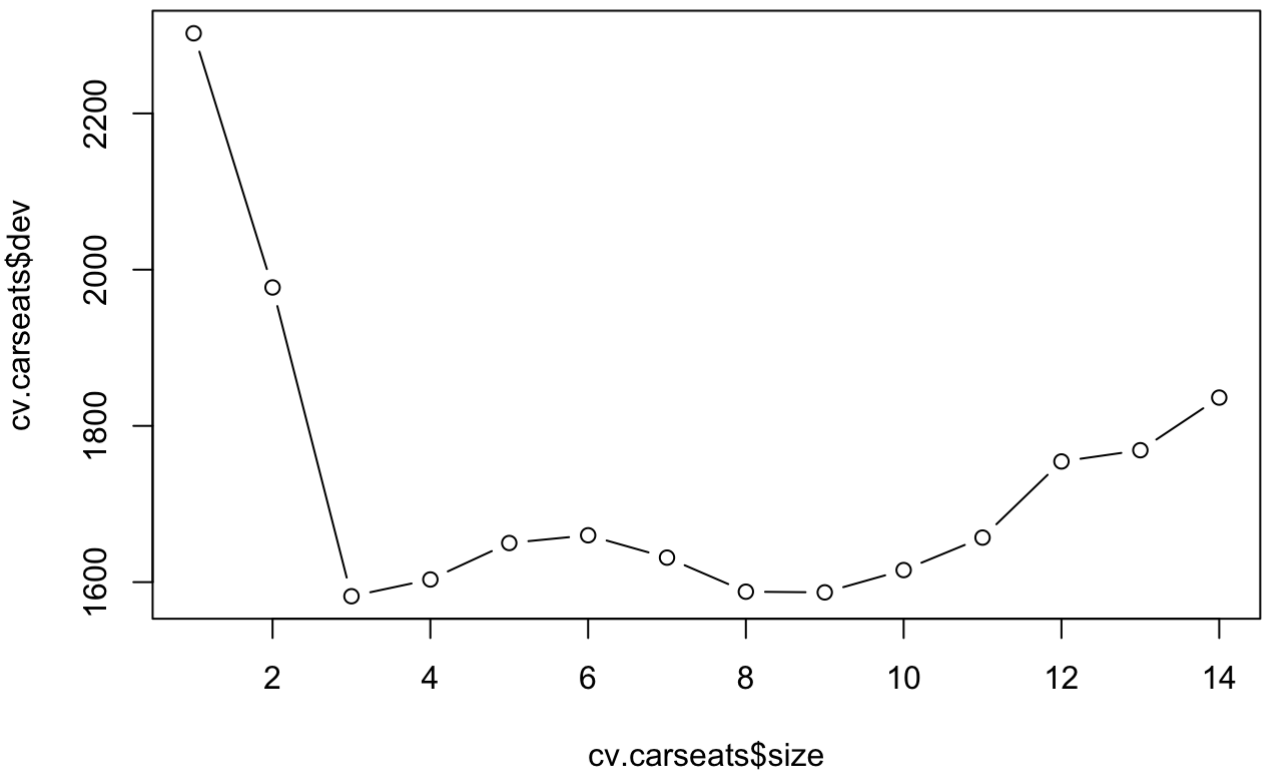
The training MSE is 3.521092, while the test MSE is 5.163211.

1(b)

```
cv.carseats = cv.tree(carseats)
cv.carseats
```

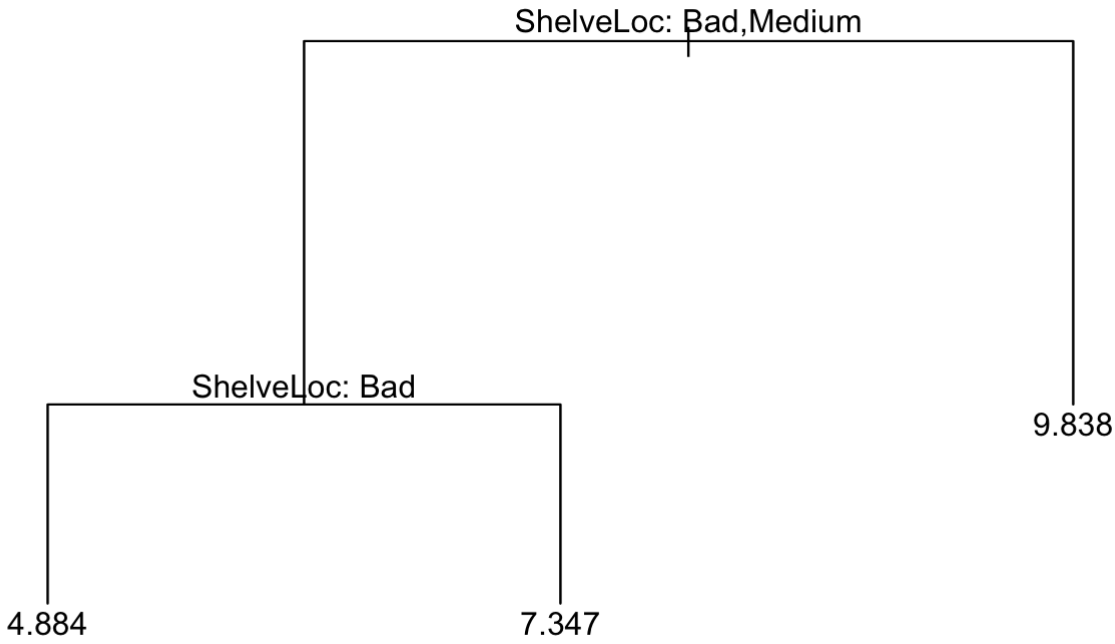
```
## $size
##  [1] 14 13 12 11 10  9  8  7  6  5  4  3  2  1
##
## $dev
##  [1] 1836.220 1768.851 1754.456 1657.014 1615.426 1586.891 1587.768 1631.374
##  [9] 1659.958 1650.061 1603.380 1581.945 1977.223 2302.576
##
## $k
##  [1]      -Inf  23.97843  25.49634  28.75808  30.62082  37.87091  45.41471
##  [8]  50.75054  64.66001  72.35951  78.78095 112.52361 258.45151 471.81572
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
plot(cv.carseats$size, cv.carseats$dev, type='b')
```



use best = 3

```
prune.carseats = prune.tree(carseats, best=3)
plot(prune.carseats)
text(prune.carseats, pretty=0)
```




```
te.pred = predict(prune.carseats, newdata = test)
error.te = mean((te.pred-test$Sales)^2)
error.te
```

```
## [1] 5.346983
```

The pruned tree does not perform better because the test error increased slightly.

1(c)

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
set.seed(1)
bag.carseats = randomForest(Sales~., data=train, mtry=9)
bag.carseats
```

```
##
## Call:
##  randomForest(formula = Sales ~ ., data = train, mtry = 9)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 9
##
##           Mean of squared residuals: 5.510135
##                     % Var explained: 34.51
```

```
tr.pred.ba = predict(bag.carseats, newdata=train)
error.tr.ba = mean((tr.pred.ba-train$Sales)^2)
error.tr.ba # Training MSE of Bagging
```

```
## [1] 0.9581012
```

```
te.pred.ba = predict(bag.carseats, newdata=test)
error.te.ba = mean((te.pred.ba-test$Sales)^2)
error.te.ba # Test MSE of Bagging
```

```
## [1] 4.942638
```

```
rf.carseats = randomForest(Sales~., data=train)
tr.pred.rf = predict(rf.carseats, newdata=train)
error.tr.rf = mean((tr.pred.rf-train$Sales)^2)
error.tr.rf  #Training MSE of random forest
```

```
## [1] 1.110016
```

```
te.pred.rf = predict(rf.carseats, newdata=test)
error.te.rf = mean((te.pred.rf-test$Sales)^2)
error.te.rf  #Test MSE of random forest
```

```
## [1] 4.754133
```

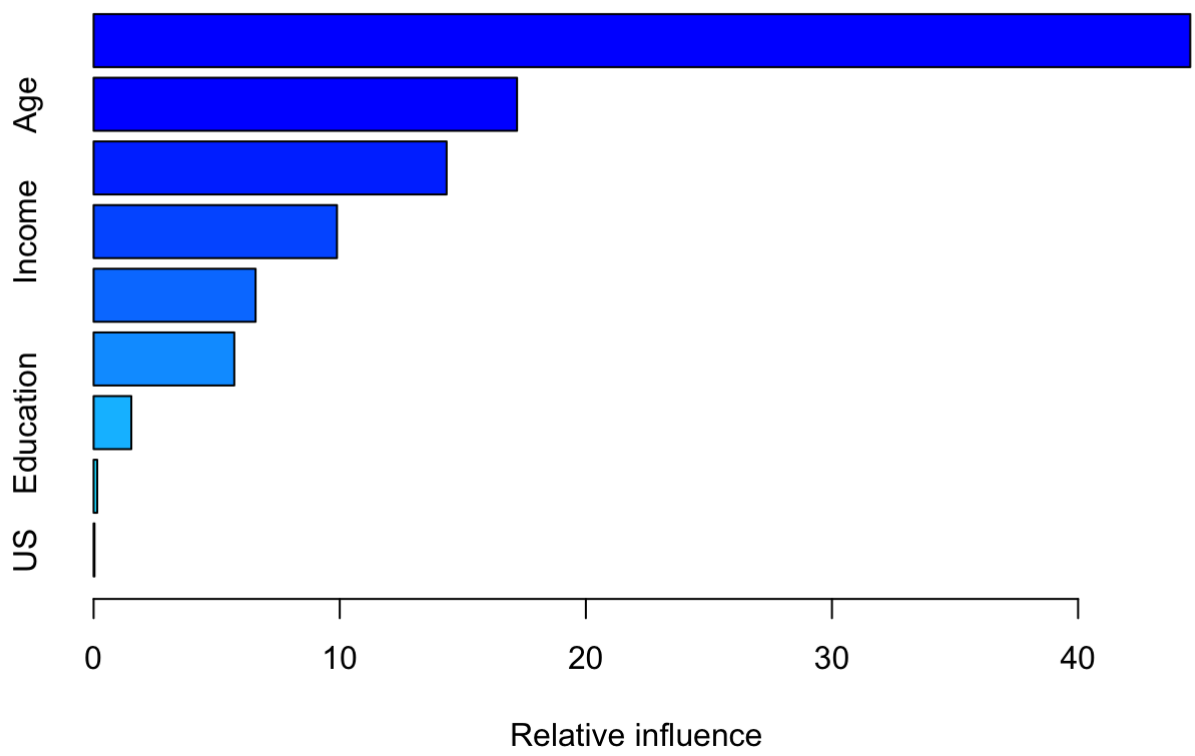
(It seems you don't need to specify the mtry, random forest output the best one. In this case, mtry = 4) Random forest does a bit better than bagging because it has slightly lower test error. So decorrelating trees works for this problem, not much though.

1(d) Tried different settings: set n.trees = 5000, the test error goes slightly up set interation.depth = 4, the test error goes up by 0.4 set shringkage = 0.1, the test error goes up by 1.3 Ultimately, I set the parameters as follow:

```
library(gbm)
```

```
## Loaded gbm 2.1.8
```

```
boost.carseats = gbm(Sales~., data=train, distribution="gaussian", n.trees=1000, inte
raction.depth=1, shrinkage=0.01)
summary(boost.carseats)
```




```
##                    var     rel.inf
## ShelveLoc     ShelveLoc 44.55716254
## Age                 Age 17.20349893
## Advertising Advertising 14.34198735
## Income           Income  9.88753636
## Population   Population  6.58041366
## CompPrice     CompPrice  5.71573643
## Education     Education  1.53149102
## Urban             Urban  0.14616036
## US                   US  0.03601335
```

```
tr.pred.bo = predict(boost.carseats, newdata=train, n.trees=1000)
error.tr.bo = mean((tr.pred.bo-train$Sales)^2)
error.tr.bo  #Training MSE of boosting
```

```
## [1] 3.844696
```

```
te.pred.bo = predict(boost.carseats, newdata=test, n.trees=1000)
error.te.bo = mean((te.pred.bo-test$Sales)^2)
error.te.bo  #Test MSE of boosting
```

```
## [1] 4.70364
```

Training MSE is 3.856231, test MSE is 4.70364. Boosted regression tree has the lowest test MSE.

1(e) Boosted regression tree performs the best. The most important predictors are ShelveLoc, Age, Advertising.

2(a) see last page

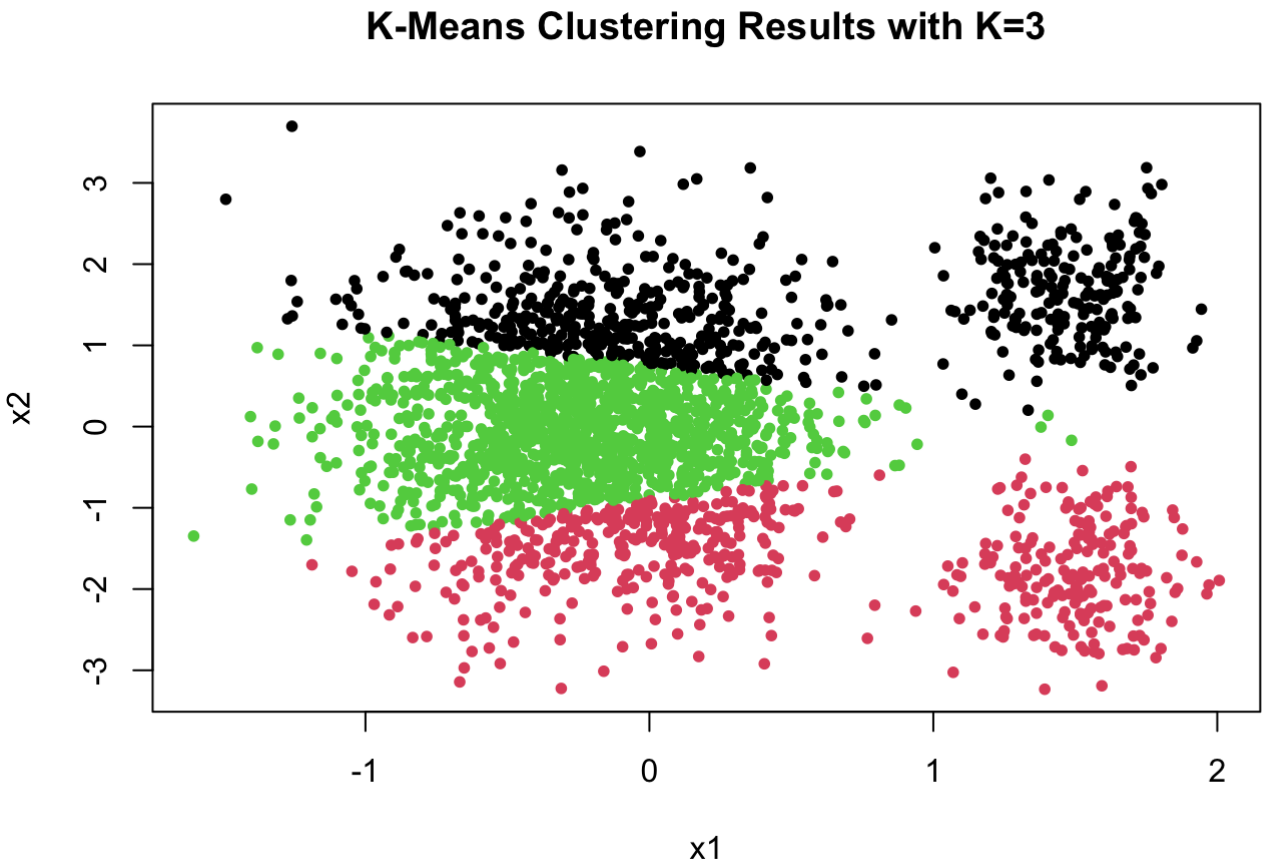
2(b) σ{d,e} = 5 σ{b} = 8 σ{b,d,e} = 3

c({d,e} -> {b}) = 3/5 = 0.6 lift({d,e} -> {b}) = 0.6/(8/10) = 0.75

The chance of occurrence of {b} when {d,e} are on transaction is 0.6. lift({d,e} -> {b}) < 1 means {d,e} and {b}) are negatively correlated. The occurrence of {d,e} inhibits the occurrence of {b}.

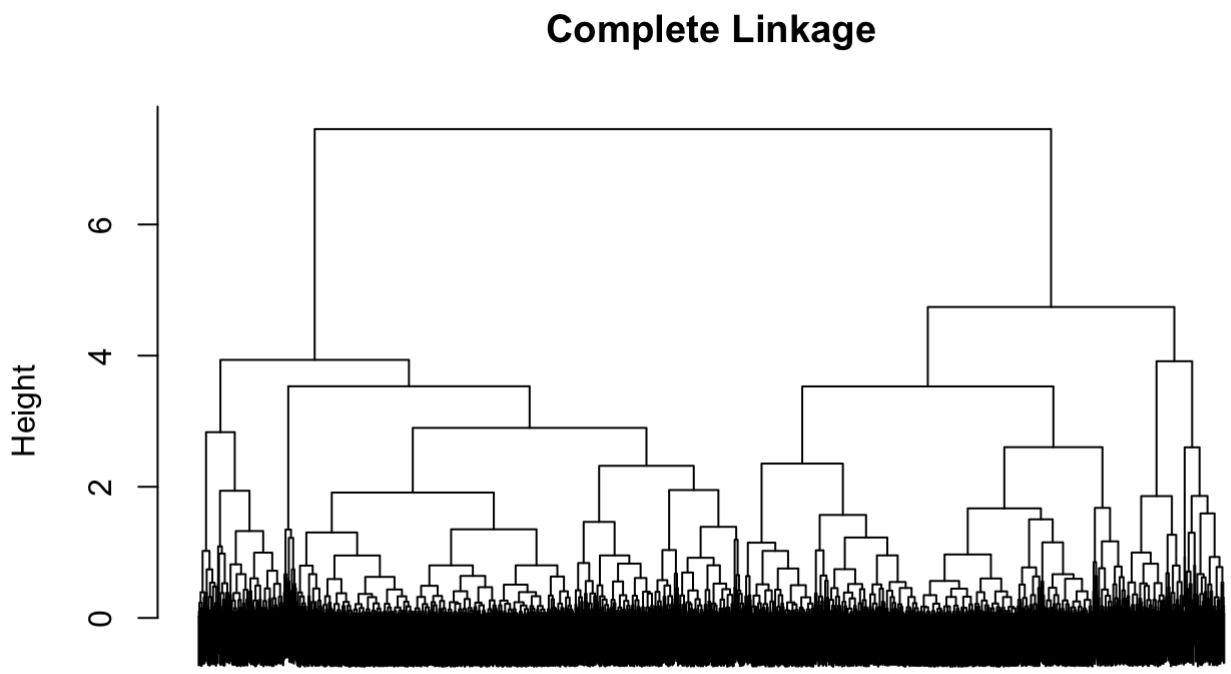
3(a)

```
data = read.csv("A3data2.csv", header=TRUE)
km.out = kmeans(data[,1:2], 3, nstart=100)
plot(data[,1:2], col=(km.out$cluster), main = "K-Means Clustering Results with K=3",
 pch = 20)
```

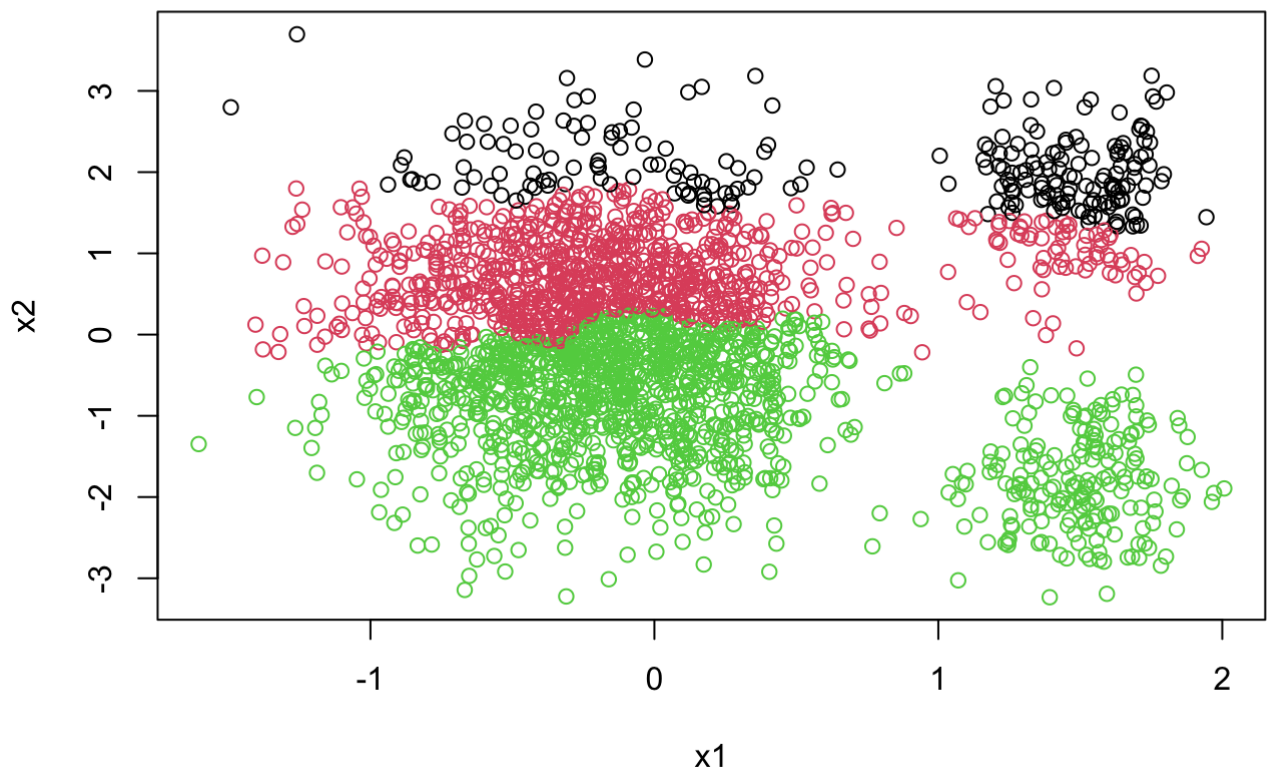


3(b)

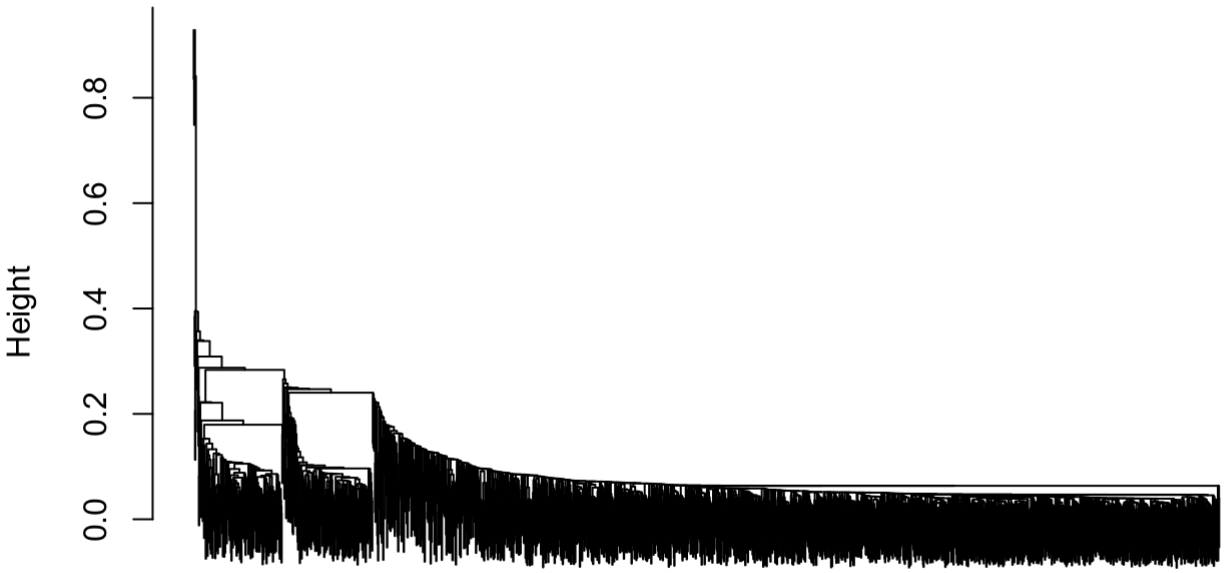
```
hc.complete = hclust(dist(data[,1:2]), method = "complete")
plot(hc.complete, main = "Complete Linkage", xlab = "", sub = "", labels = FALSE)
```



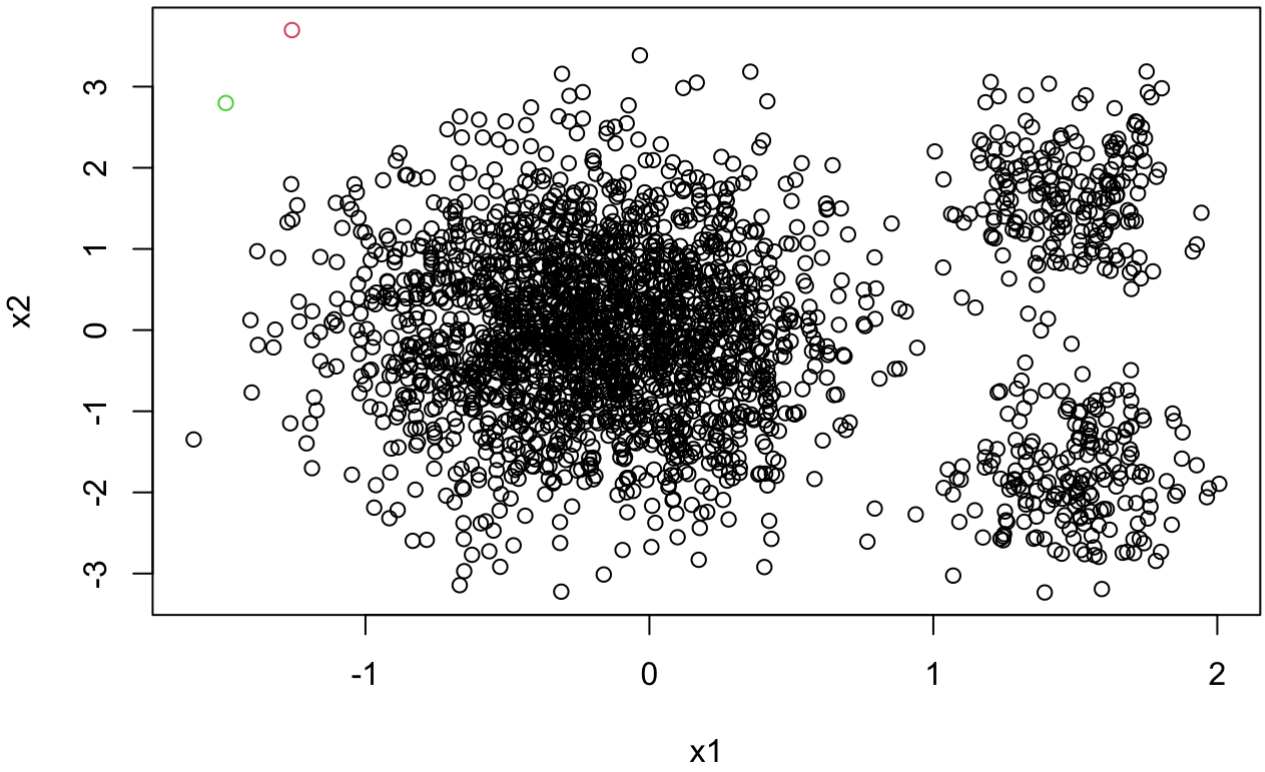

```
plot(data[,1:2], col=cutree(hc.complete, 3))
```



```
hc.single = hclust(dist(data[,1:2]), method = "single")
plot(hc.single, main = "Single Linkage", xlab = "", sub = "", labels = FALSE)
```




```
plot(data[,1:2], col=cutree(hc.single, 3))
```



3(c)

```
table(km.out$cluster, data[,3])
```

```
##
##        1    2    3
##   1  199  409    0
##   2    0  333  198
##   3    1 1258    2
```

```
table(cutree(hc.complete, 3), data[,3])
```

```
##
##        1    2    3
##   1  132   95    0
##   2   68  825    2
##   3    0 1080  198
```

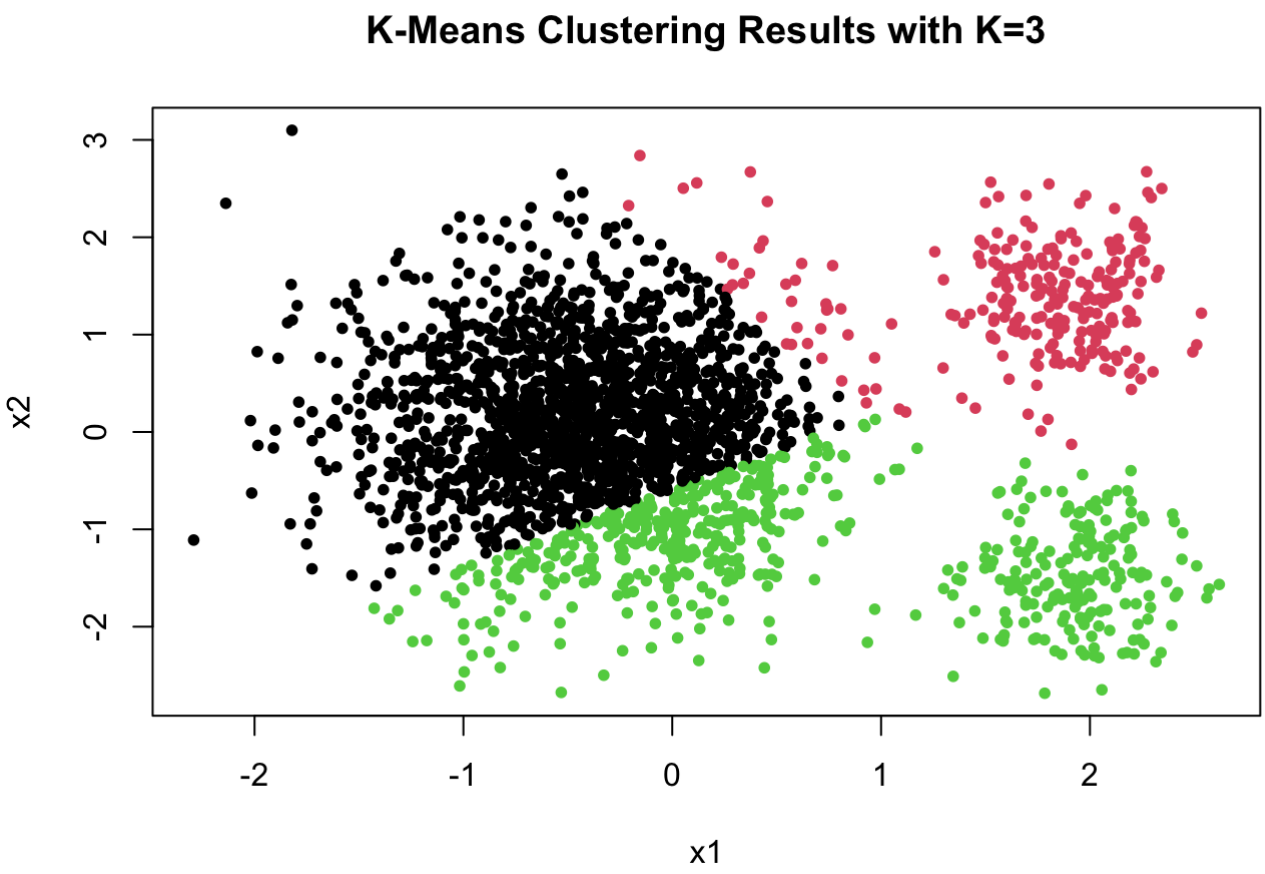
```
table(cutree(hc.single, 3), data[,3])
```

```
##
##        1    2    3
##   1  200 1998  200
##   2    0    1    0
##   3    0    1    0
```

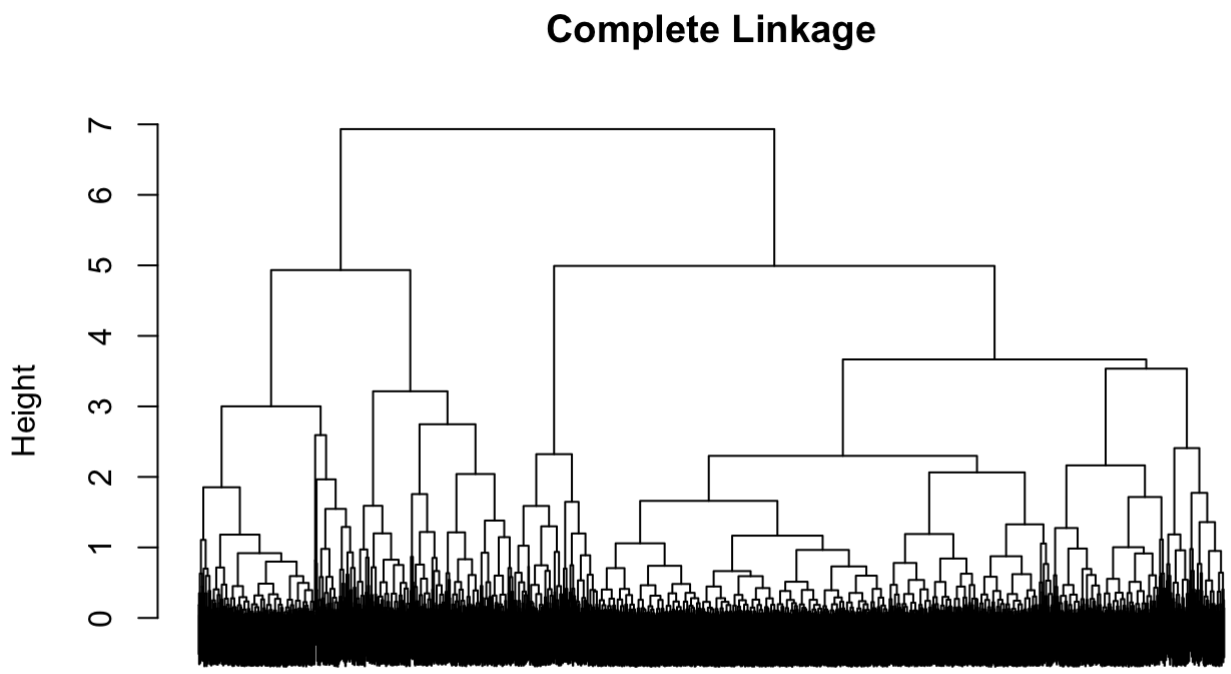
k-means clustering correctly predicted 534 points out of 2400. (it would change evrytime run it) Complete linkage correctly predicted 959 points out of 2400. Single linkage correctly predicted 201 points out of 2400. Complete linkage has the lowest error rate, however the plot still does not look good. It seperates the largest cluster into three groups.The single linkage probably was misguided by several outliers and just simply cluster nearly all the points in one cluster. K means works well in the two small clusters, however, in the big cluster, it seperated it into three groups like complete linkage. This is probably one of the drawbacks of k means because it is not easy for it to distinguish big clusters. Overall, I think none of them performs well in this data set.

3(d)

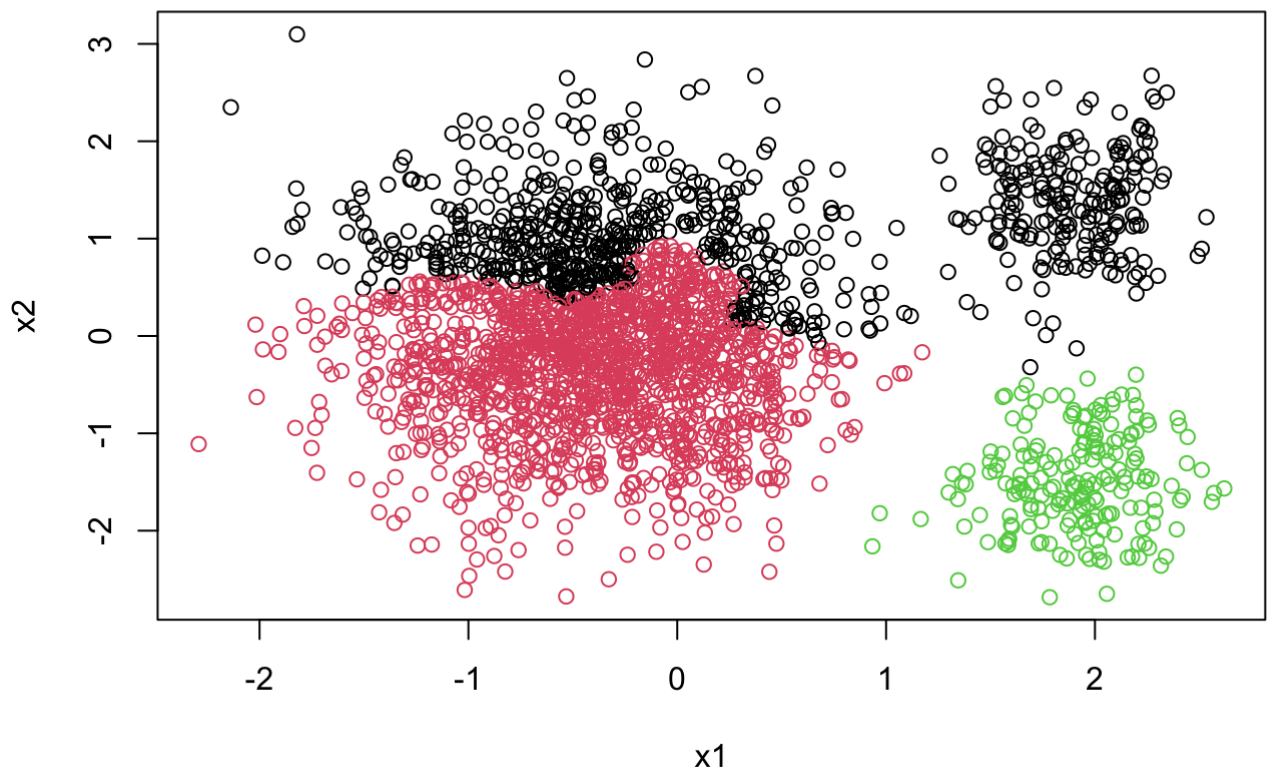
```
cluster = data[,3]
data = scale(data[,1:2], center=TRUE, scale=TRUE)
data = data.frame(data, cluster)
km.out = kmeans(data[,1:2], 3, nstart=100)
plot(data[,1:2], col=(km.out$cluster), main = "K-Means Clustering Results with K=3",
 pch = 20)
```



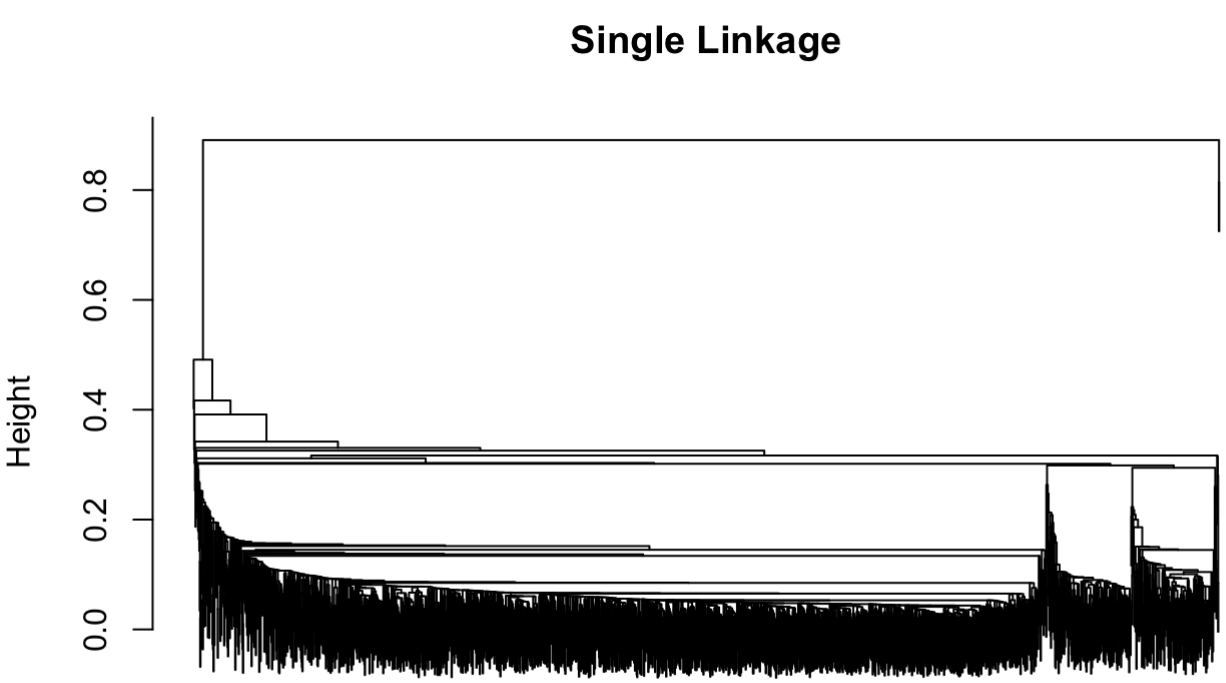

```
hc.complete = hclust(dist(data[,1:2]), method = "complete")
plot(hc.complete, main = "Complete Linkage", xlab = "", sub = "", labels = FALSE)
```



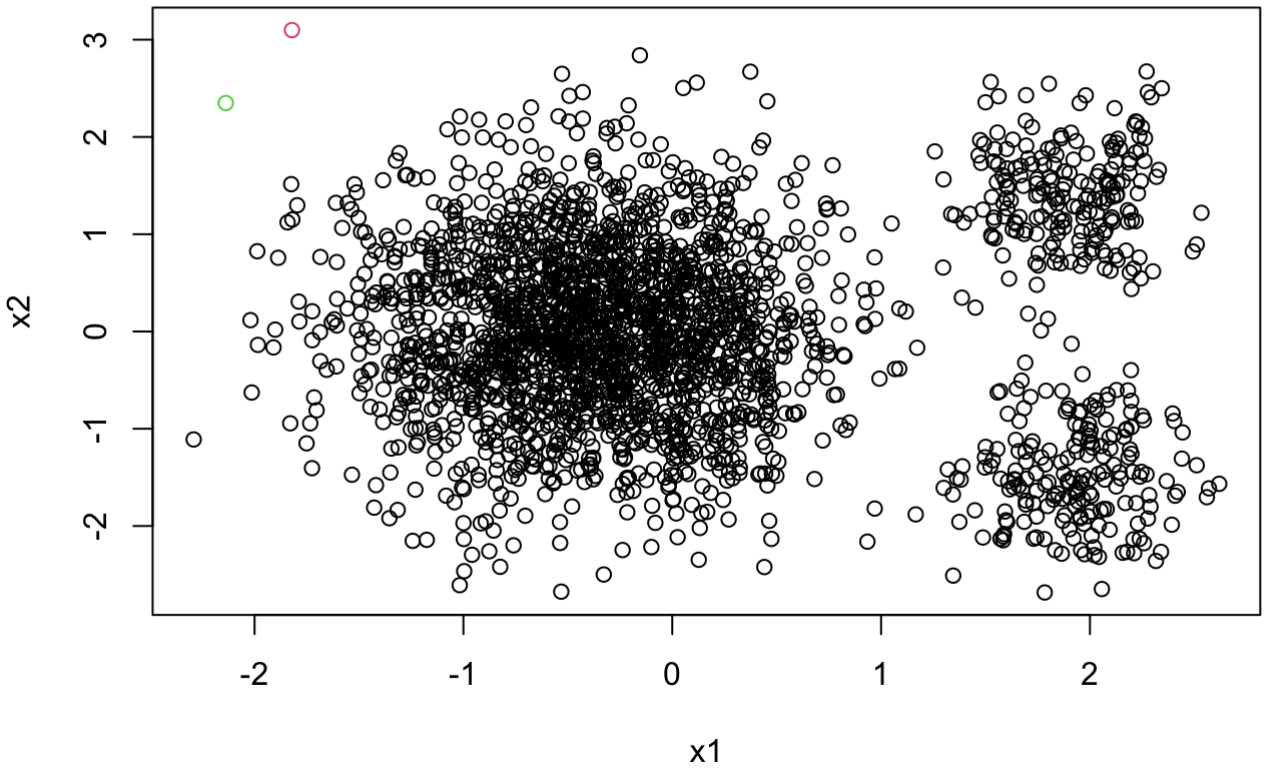

```
plot(data[,1:2], col=cutree(hc.complete, 3))
```



```
hc.single = hclust(dist(data[,1:2]), method = "single")
plot(hc.single, main = "Single Linkage", xlab = "", sub = "", labels = FALSE)
```




```
plot(data[,1:2], col=cutree(hc.single, 3))
```




```
table(km.out$cluster, data[,3])
```

```
##
##         1    2    3
##   1     0 1598    0
##   2   200   44    2
##   3     0  358  198
```

```
table(cutree(hc.complete, 3), data[,3])
```

```
##
##         1    2    3
##   1   200  535    3
##   2     0 1464    0
##   3     0    1  197
```

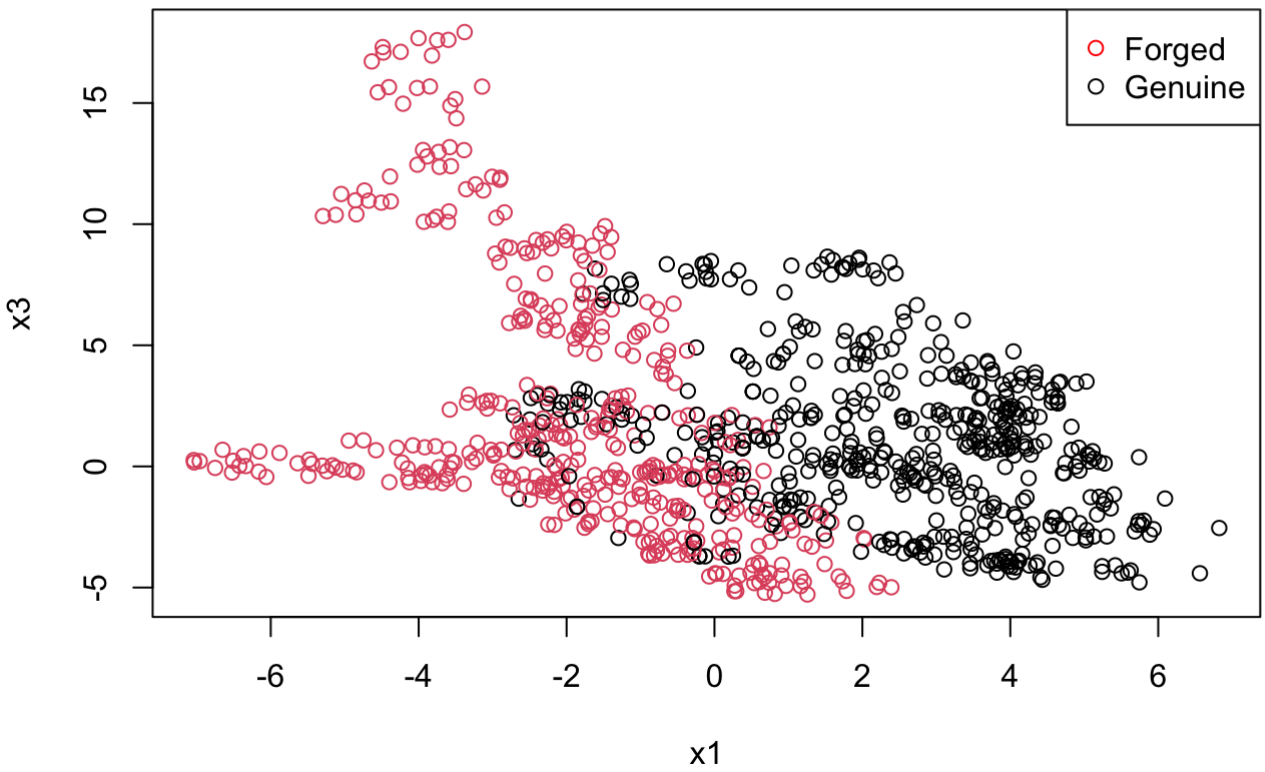
```
table(cutree(hc.single, 3), data[,3])
```

```
##
##         1    2    3
##   1   200 1998  200
##   2     0    1    0
##   3     0    1    0
```

Only the complete linkage improved a lot. Other two does not improve.

4(a)

```
train = read.csv("BankTrain.csv", header=TRUE)
train$y = as.factor(train$y)
test = read.csv("BankTest.csv", header=TRUE)
test$y = as.factor(test$y)
plot(train$x1, train$x3, col=train$y, xlab="x1", ylab="x3")
legend("topright", legend=c("Forged","Genuine"), col=c("red","black"), pch=21)
```



From the chart, a separating hyperplane does not exist because there is not a clear seperate between the two clusters.

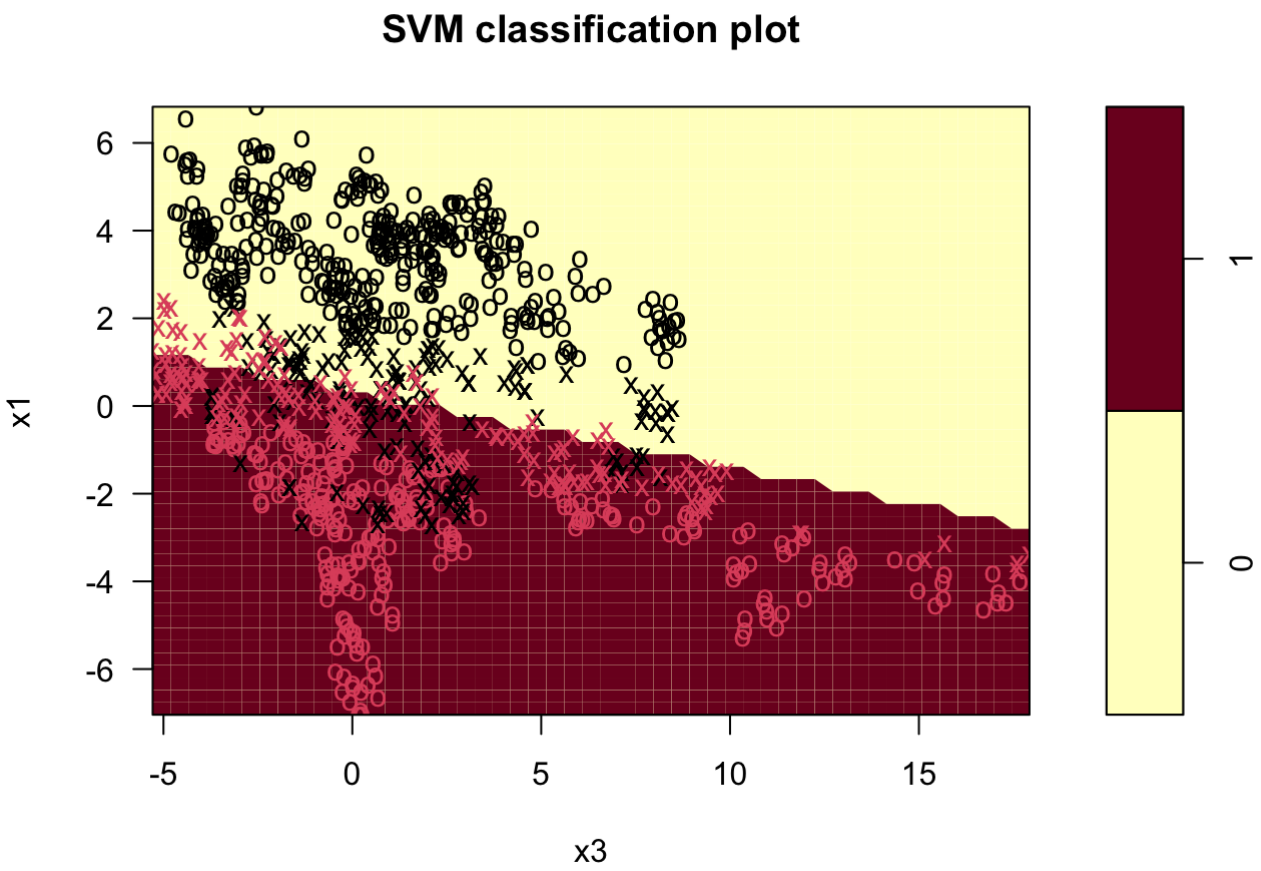
4(b)

```
library(e1071)
set.seed(2)
tune.out = tune(svm, y~x1+x3, data=train, kernel="linear", ranges=list(cost=c(0.01,0.
1,1,10,100,1000)))
bestmod = tune.out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ x1 + x3, data = train, ranges = list(cost =
c(0.01,
##     0.1, 1, 10, 100, 1000)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.1
##
## Number of Support Vectors:  350
##
##  ( 175 175 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

Best cost: 0.1

```
plot(bestmod, train[,c(1,3,5)])
```




```
ypred = predict(bestmod, test)
table(predict=ypred, truth=test$y)
```

```
##         truth
## predict   0   1
##       0 197  11
##       1  39 165
```

```
362/412
```

```
## [1] 0.8786408
```

362 observations are predicted right, while 50 observations are predicted wrong. The accuracy is 0.8786408.

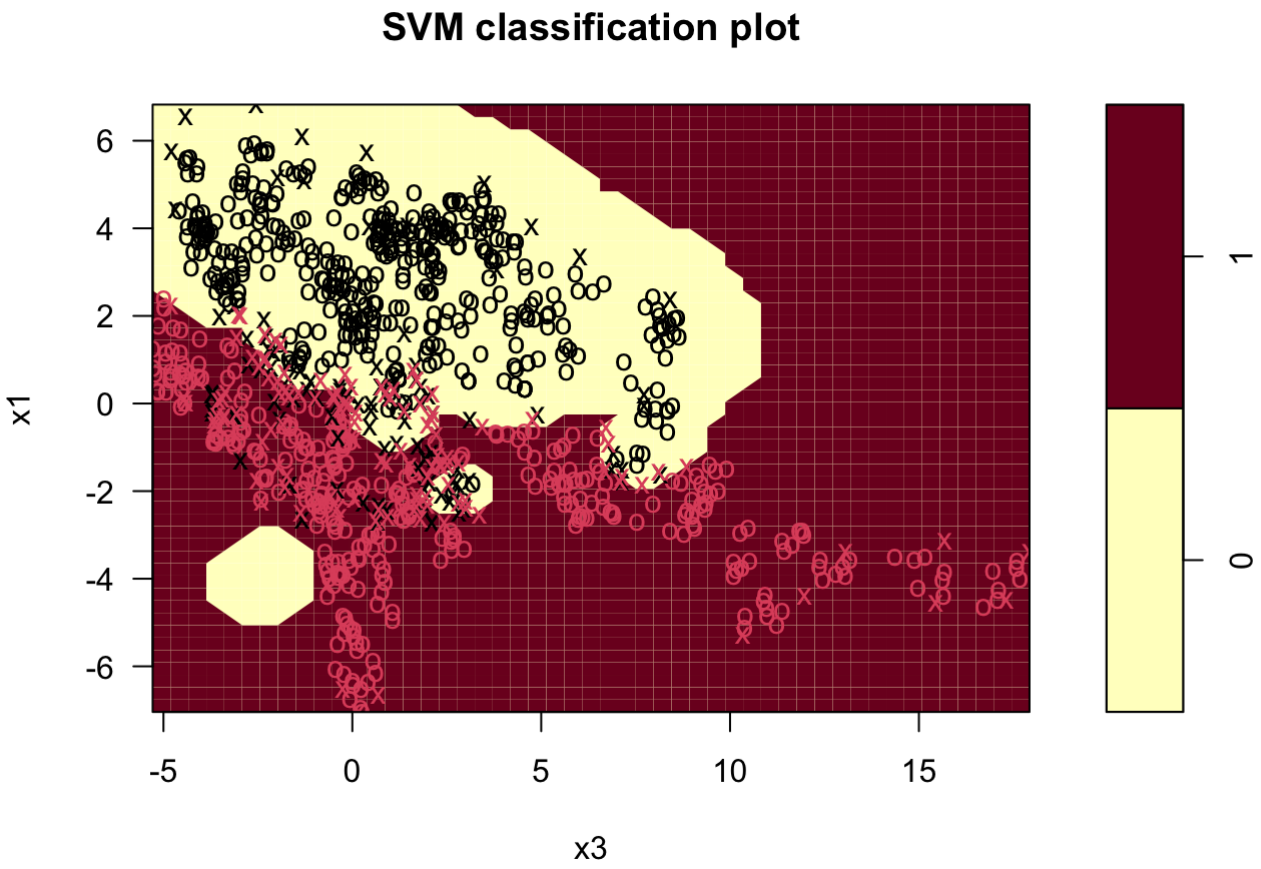
4(c)

```
set.seed(3)
tune.out = tune(svm, y~x1+x3, data=train, kernel="radial", ranges=list(cost=c(0.01,0.
1,1,10,100,1000), gamma=c(0.5,1,2,3,4)))
bestmod = tune.out$best.mod
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   100     4
##
## - best performance: 0.096875
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-02   0.5 0.12291667 0.03668087
## 2  1e-01   0.5 0.10416667 0.02734031
## 3  1e+00   0.5 0.10208333 0.03212509
## 4  1e+01   0.5 0.10625000 0.03286711
## 5  1e+02   0.5 0.09895833 0.03153802
## 6  1e+03   0.5 0.10416667 0.02734031
## 7  1e-02   1.0 0.11770833 0.02691812
## 8  1e-01   1.0 0.10208333 0.02635231
## 9  1e+00   1.0 0.10416667 0.03366444
## 10 1e+01   1.0 0.10104167 0.03221878
## 11 1e+02   1.0 0.10937500 0.03076397
## 12 1e+03   1.0 0.11041667 0.03227486
## 13 1e-02   2.0 0.17395833 0.04911691
## 14 1e-01   2.0 0.10208333 0.03019037
## 15 1e+00   2.0 0.10625000 0.03212509
## 16 1e+01   2.0 0.10416667 0.02905070
## 17 1e+02   2.0 0.10104167 0.02600692
## 18 1e+03   2.0 0.10000000 0.02560985
## 19 1e-02   3.0 0.32812500 0.05794555
## 20 1e-01   3.0 0.10312500 0.02964635
## 21 1e+00   3.0 0.10729167 0.03259083
## 22 1e+01   3.0 0.10729167 0.02506261
## 23 1e+02   3.0 0.10104167 0.01967252
## 24 1e+03   3.0 0.10000000 0.01914451
## 25 1e-02   4.0 0.42708333 0.06569753
## 26 1e-01   4.0 0.10416667 0.03330439
## 27 1e+00   4.0 0.10520833 0.02923685
## 28 1e+01   4.0 0.09895833 0.02154457
## 29 1e+02   4.0 0.09687500 0.02252934
## 30 1e+03   4.0 0.10208333 0.02071727
```

Best cost: 100 Best gama: 0.5

```
plot(bestmod, train[,c(1,3,5)])
```




```
ypred = predict(bestmod, test)
table(predict=ypred, truth=test$y)
```

```
##         truth
## predict   0   1
##       0 211   9
##       1  25 167
```

```
378/412
```

```
## [1] 0.9174757
```

SVM performs better because it creates a non-linear boundary such that it fits the data more precisely.

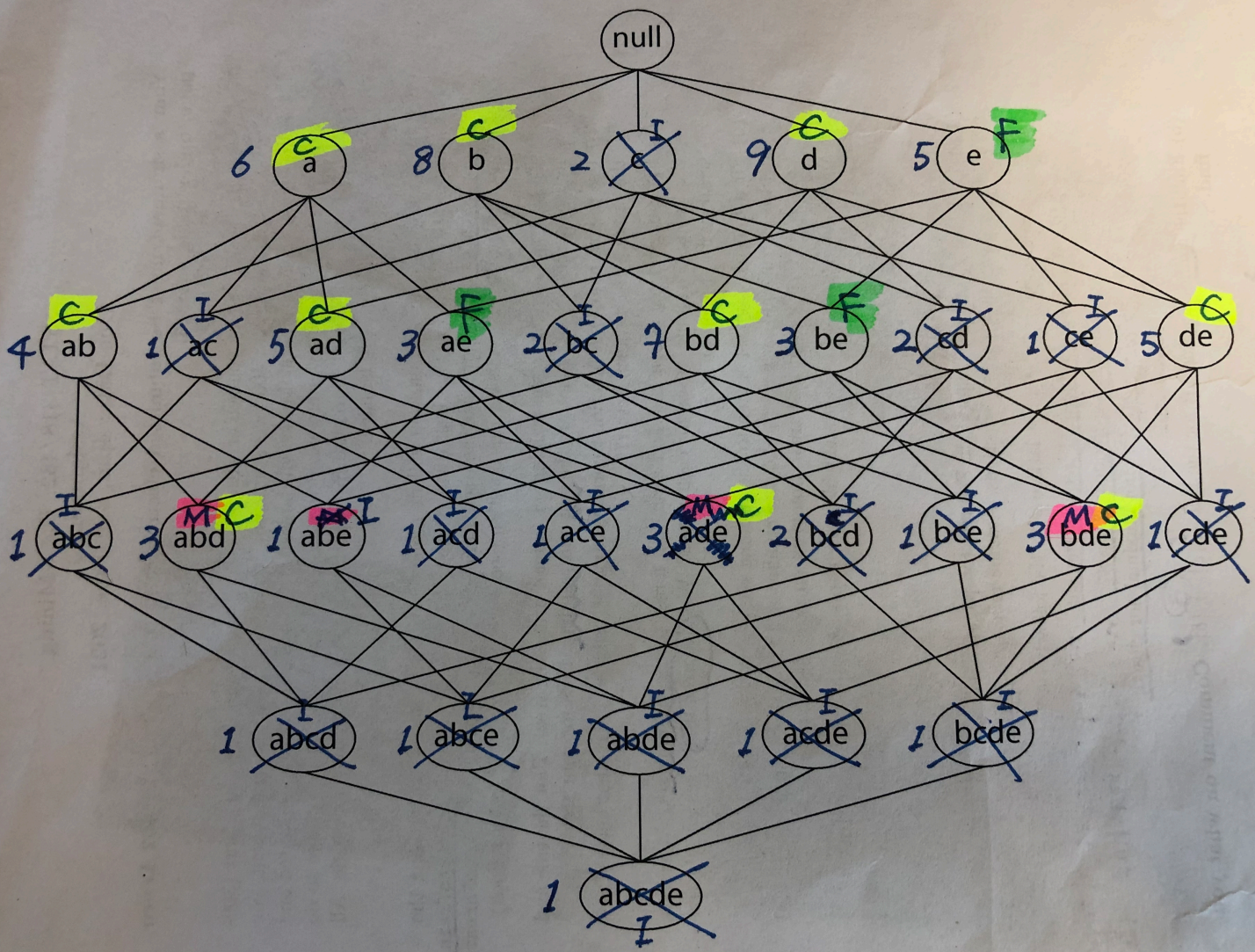


Figure 1: Itemset lattice for Question 3.

University of Canterbury, Thomas Li, 2021