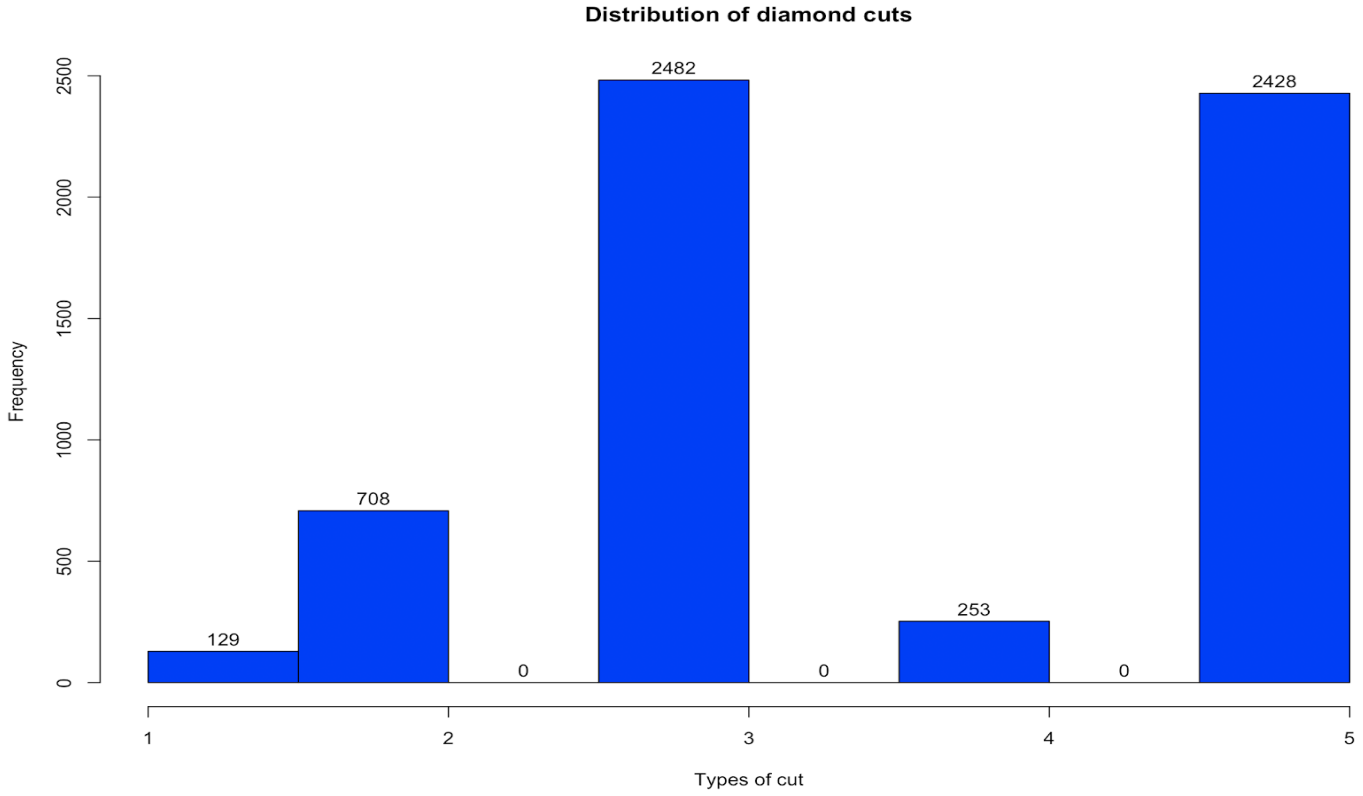
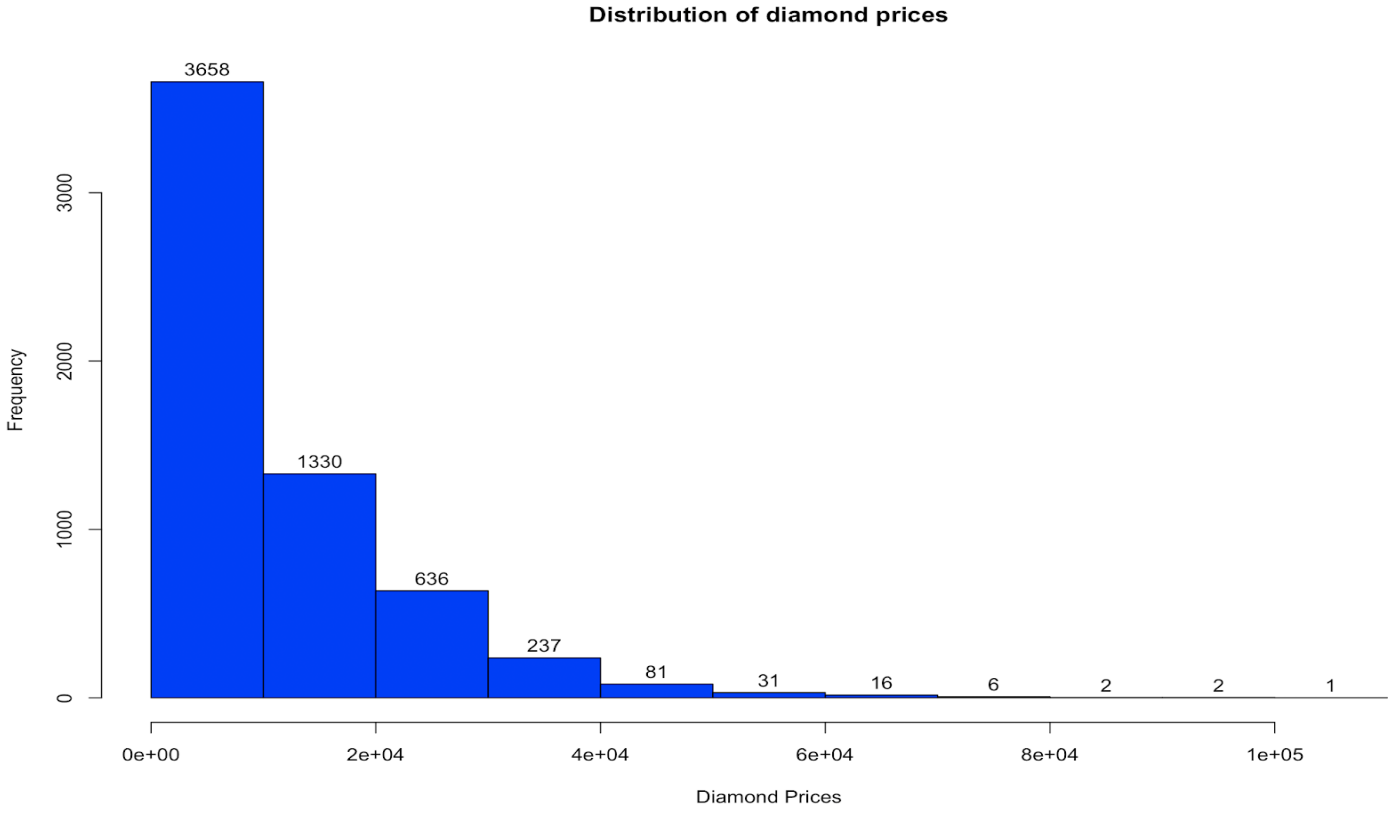
NAME:BHAUMIK ICHHAPORIA **HW#4**
ID: W1549327

1. Ensemble Methods. The 13_Diamond_Data.xls file contains information on 6000 diamonds; read the accompanying README file for a description of the different attributes. The goal in this problem is to build a regression model to predict the wholesale market value of a unique stone.
You can use the read.xls() function from the gdata package to read the data file, or convert it to text from Excel and read it with read.table(). Confirm that variables Cut, Color, Clarity, Polish, Symmetry and Report are categorical. We are going to use the Prediction Rule Ensembles ('pre') package for building rule-ensemble models.

(a) Show the distribution of the "cut" and "Price" attributes.

```
library(gdata)
diamond_data_i<- read.xls("~/Downloads/Diamond_Data.xls", header=TRUE)
diamond_data=diamond_data_i[,-1]
str(diamond_data)
summary(diamond_data)
indices <- sample(1:nrow(diamond_data), size = (0.83334*nrow(diamond_data)))##sampling 80% of the

training_diamond<- diamond_data[indices,] ##putting the sampled data in training vector
test_diamond <- diamond_data[-indices,] ##putting the rest dataset in the test_data
help(hist)
hist(diamond_data$Price, col="blue1", xlab="Diamond Prices",
                         ylab="Frequency", main="Distribution of diamond prices", labels=TRUE)
hist(as.numeric(diamond_data$Cut), col="blue1", xlab="Types of cut", ylab="Frequency",
                         main="Distribution of diamond cuts", labels=TRUE)
```


Distribution of diamond cuts
Frequency
0
500
1000
1500
2000
2500
129
708
0
2482
0
253
0
2428
1
2
3
4
5
Types of cut


Distribution of diamond prices
Frequency
0
1000
2000
3000
3658
1330
636
237
81
31
16
6
2
2
1
0e+00
2e+04
4e+04
6e+04
8e+04
1e+05
Diamond Prices

(b) Use the pre() command to fit a regression model (e.g., family =" gaussian") to a random sample of size 5000. Feel free to experiment with various values of the pre's arguments, but in you final submission use nfolds =10. Use the summary() command to report the number of terms (the ensemble size) and root mean-squared error on the training data.

->indices <- sample(1:nrow(diamond_data), size =(0.83334*nrow(diamond_data))) #5000 data
->training_diamond<- diamond_data[indices,] ##putting the sampled data in training vector
->test_diamond <- diamond_data[-indices,] ##putting the rest dataset in the test_data

-> pre_model <- pre(Price ~ ., data = training_diamond,family = gaussian,nfolds = 10)

```
> str(training_diamond)
'data.frame':   5000 obs. of  8 variables:
 $ Carat.Weight: num  0.9 2.31 0.91 1.28 0.9 0.91 1.32 1.51 2.2 0.79 ...
 $ Cut         : Factor w/ 5 levels "Fair","Good",..: 2 3 5 3 5 3 3 1 5 5 ...
 $ Color       : Factor w/ 6 levels "D","E","F","G",..: 3 5 1 4 2 6 4 5 4 4 ...
 $ Clarity     : Factor w/ 7 levels "FL","IF","SI1",..: 3 6 3 4 7 7 6 4 3 3 ...
 $ Polish      : Factor w/ 4 levels "EX","G","ID",..: 4 3 1 1 2 3 1 4 2 1 ...
 $ Symmetry    : Factor w/ 4 levels "EX","G","ID",..: 2 3 1 4 4 3 1 1 2 4 ...
 $ Report      : Factor w/ 2 levels "AGSL","GIA": 2 1 2 2 2 1 2 2 2 2 ...
 $ Price       : num  4004 28792 5029 9640 5987 ...
> cat_var <- c("Cut", "Color", "Clarity", "Polish", "Symmetry", "Report")
>
> pre_model <- pre(Price ~ ., data = training_diamond,family = gaussian,nfolds = 10)
> summary(pre_model)

Final ensemble with cv error within 1se of minimum:
  lambda =  5.290334
  number of terms = 230
  mean cv error (se) = 1247903 (113491.5)

  cv error type : Mean-Squared Error
```

Number of terms=230
Root mean square error=1247903

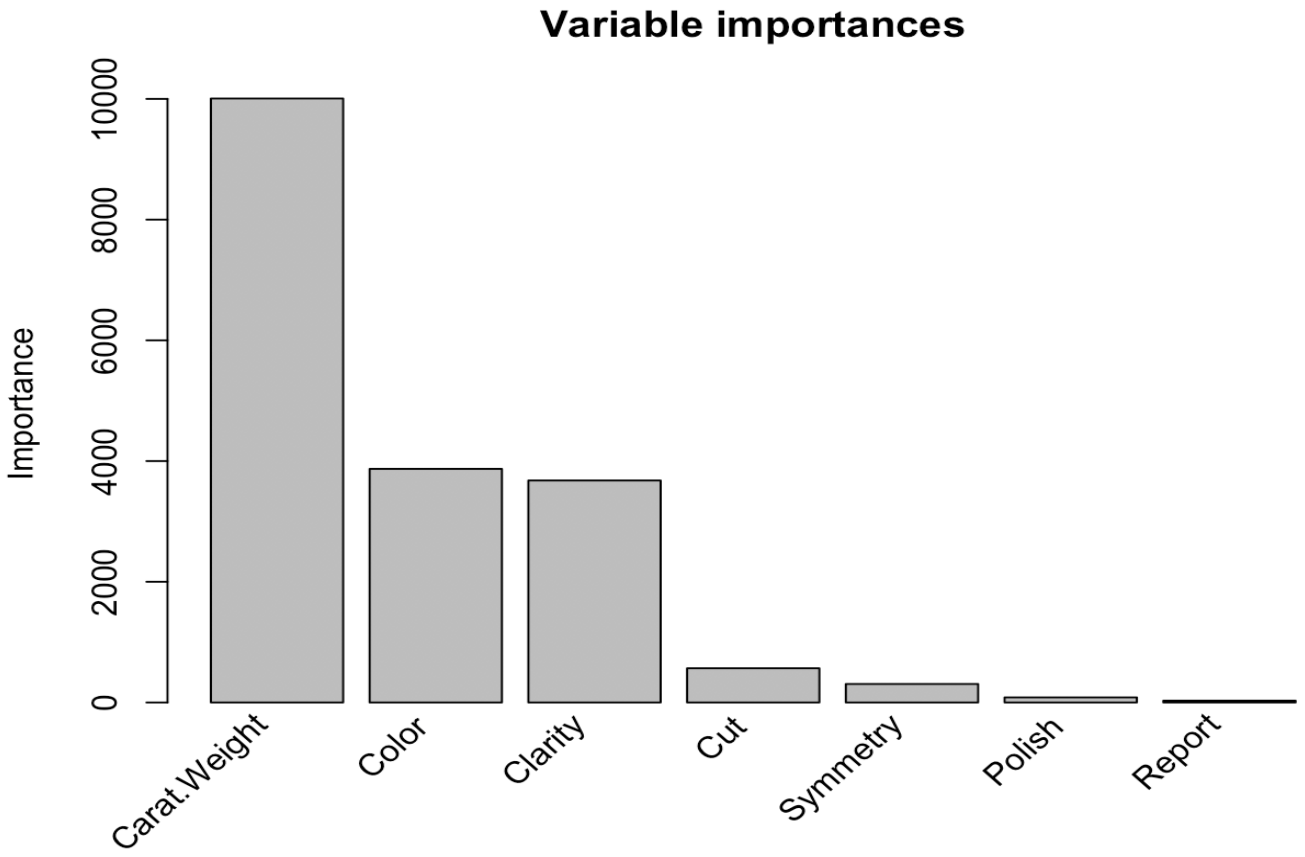
(c) Use the print() command to view the top 10 rules. i) Was there a linear term in the model? Explain its coefficient. ii) Explain one of the other rules.

```
> head(pre_model$rules,10)
         rule
rule1   rule1
rule2   rule2
rule3   rule3
rule4   rule4
rule5   rule5
rule6   rule6
rule7   rule7
rule8   rule8
rule9   rule9
rule10 rule10

description
rule1                                                                                                  Carat.Weight
<= 1.71
rule2                                                                                 Carat.Weight <= 1.71 & Carat.Weight
<= 1.2
rule3                                                  Carat.Weight <= 1.71 & Carat.Weight <= 1.2 & Carat.Weight
<= 0.96
rule4                                                   Carat.Weight <= 1.71 & Carat.Weight <= 1.2 & Carat.Weight
> 0.96
rule5                                                                                  Carat.Weight <= 1.71 & Carat.Weigh
t > 1.2
rule6                        Carat.Weight <= 1.71 & Carat.Weight > 1.2 & Clarity %in% c("FL", "IF", "VVS1",
"VVS2")
rule7                               Carat.Weight <= 1.71 & Carat.Weight > 1.2 & Clarity %in% c("SI1", "VS1",
"VS2")
rule8                                                   Carat.Weight > 1.71 & Clarity %in% c("FL", "IF", "VVS1",
"VVS2")
rule9                          Carat.Weight > 1.71 & Clarity %in% c("FL", "IF", "VVS1", "VVS2") & Color %in%
c("D")
rule10 Carat.Weight > 1.71 & Clarity %in% c("FL", "IF", "VVS1", "VVS2") & Color %in% c("E", "F", "G", "H
", "I")
```

The linear term in the model is Carat Weight.
The Rule-10 in the above picture shows that the price of the diamond can be predicted using the carat weight, which is further split into clarity and Color. This shows that according to this role Carat weight, clarity and color are the most important predictors.

(d) Use the importance() command to variable importance plot. Which are the top three most important variables in determining a diamond's price?

-> importance(pre_model)



(e) Report the error on the test set using "Average Absolute Error"

```
> test_data_x <- test_diamond[,1:7]
> test_data_y <-test_diamond[,8]
>
> yhat <- predict(pre_model,test_data_x)
> abs.test.ensemble <- 1/1000*sum(abs(test_data_y - yhat))
> print(abs.test.ensemble)
[1] 588.8536
```

"Average Absolute Error" = 588.8536

(f) Fit a decision tree to the same training data and report the error of the best pruned tree.

```r
library(rpart)
#converting appropriate x attributes to factors

training_diamond[,2] <- as.factor(training_diamond[,2])
training_diamond[,3] <- as.factor(training_diamond[,3])
training_diamond[,4] <- as.factor(training_diamond[,4])
training_diamond[,5] <- as.factor(training_diamond[,5])
training_diamond[,6] <- as.factor(training_diamond[,6])
training_diamond[,7] <- as.factor(training_diamond[,7])


# fitting a decision tree to training data and plotting complexity parameter
classifier <- rpart(Price~., method = "class", data = training_diamond, cp=0.0001)
y_pred = predict(classifier, newdata = test_diamond,type = 'class')
print(y_pred)

#getting the optimal cp value from cp table to prune the tree
View(classifier$cptable)
print(classifier$cptable[which.min(classifier$cptable[,"xerror"]),"CP"])
plotcp(classifier)
#pruning the tree
prune_tree<- prune(classifier, cp = 0.0001001402)

#plotting the pruned tree
plot(prune_tree, uniform=TRUE,
     main="Pruned Classification Tree" )
text(prune_tree)
summary(prune_tree)
print(prune_tree)
prune_pred <- predict(prune_tree, test_diamond, type = "vector")
abs.test.tree <- 1/1000*sum(abs(test_diamond$Price - prune_pred))


print(abs.test.tree)
```

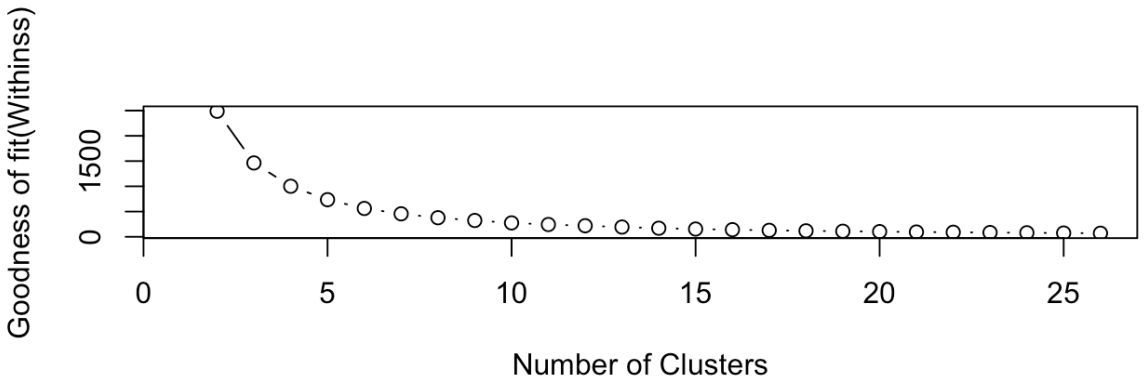
```
> summary(prune_tree)
Call:
rpart(formula = Price ~ ., data = training_diamond, method = "class",
    cp = 1e-04)
  n= 5000

             CP nsplit rel error    xerror         xstd
1  0.0010012014      0 1.0000000 1.0012014 0.0000000000
2  0.0008009612      1 0.9989988 1.0012014 0.0000000000
3  0.0007342144      7 0.9941930 0.9997998 0.0005294150
4  0.0006007209     13 0.9897877 0.9969964 0.0009156873
5  0.0005339741     27 0.9813777 0.9963957 0.0009786159
6  0.0005006007     33 0.9781738 0.9959952 0.0010183710
7  0.0004004806     37 0.9761714 0.9959952 0.0010183710
8  0.0003754505    148 0.9317181 0.9915899 0.0013806303
9  0.0003504205    156 0.9287145 0.9909892 0.0014226900
10 0.0003337338    160 0.9273128 0.9909892 0.0014226900
11 0.0003003604    166 0.9253104 0.9907889 0.0014364251
12 0.0002002403    172 0.9235082 0.9907889 0.0014364251
13 0.0001001402    397 0.8784541 0.9891870 0.0015417202

Variable importance
Carat.Weight        Color      Clarity       Polish          Cut     Symmetry       Report
          30           16           15           13           12           11            2
```

```
> prune_pred <- predict(prune_tree, test_diamond, type = "vector")
> abs.test.tree <- 1/1000*sum(abs(test_diamond$Price - prune_pred))
> print(abs.test.tree)
[1] 966.909
```

2. Partition-based Clustering. Use kmeans() to find clusters in all the letters data. Do not include the first column. One of the values returned by this function is withinss - the within-cluster sum of squares for each cluster. We use the average withinss as a "goodness of fit" criterion for a given value of K

(a) Repeat kmeans for K=2,…,26, computing and saving J(C) in each case.

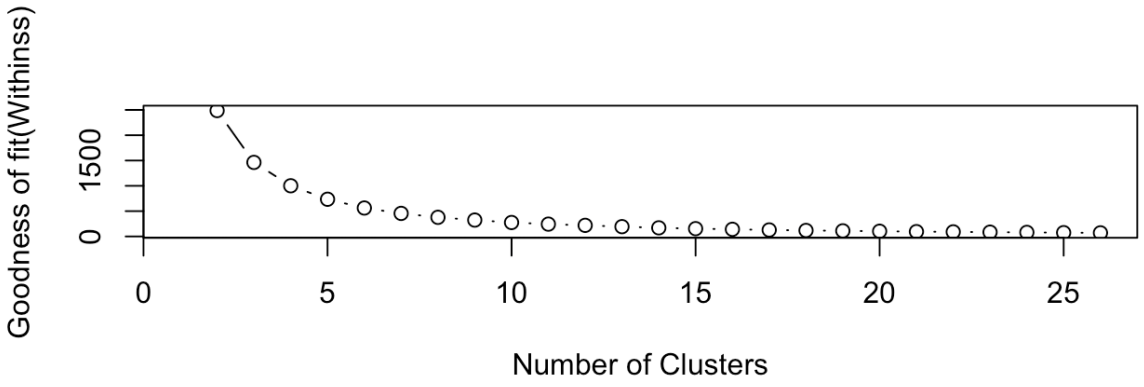
```r
set.seed(123)

#Loading the "letters Data"
az_char<- read.table("~/Downloads/az-5000.txt", header=TRUE)

#Removing the first column of matrix
az_char_c<- az_char[,-1]
dim(az_char_c)
head(az_char_c)

#performing kmeans and determining the number of clusters
fit <- vector()

for (i in 2:26)
{
  kmeans_output <- kmeans(az_char_c, centers=i, iter.max=26)
 fit[i] <- (1/i)*sum(kmeans(az_char_c, centers=i)$withinss)
}

#plotting goodness of fit vs. number of clusters(K)
par(mfrow=c(2,1))
plot(1:26, fit, type="b", xlab="Number of Clusters", ylab="Goodness of fit(Withinss)")
```

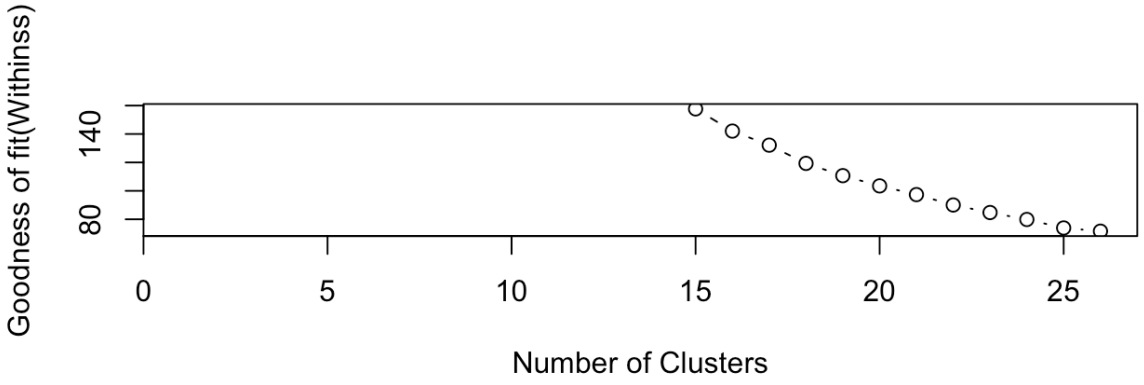


(b) Plot the goodness of fit values vs. K twice. First for all K, then for K=15,…,26 only. – do

you notice any “steps” that might suggest the number of “natural” clusters?

```
#performing kmeans from 15 to 26 for determining the number of clusters
fit2 <- vector()
for (i in 15:26)
{
  fit2[i] <- (1/i)*sum(kmeans(az_char_c, centers=i)$withinss)
}
#plotting goodness of fit vs. number of clusters(K)
plot(1:26, fit2, type="b", xlab="Number of Clusters", ylab="Goodness of fit(Withinss)")
```





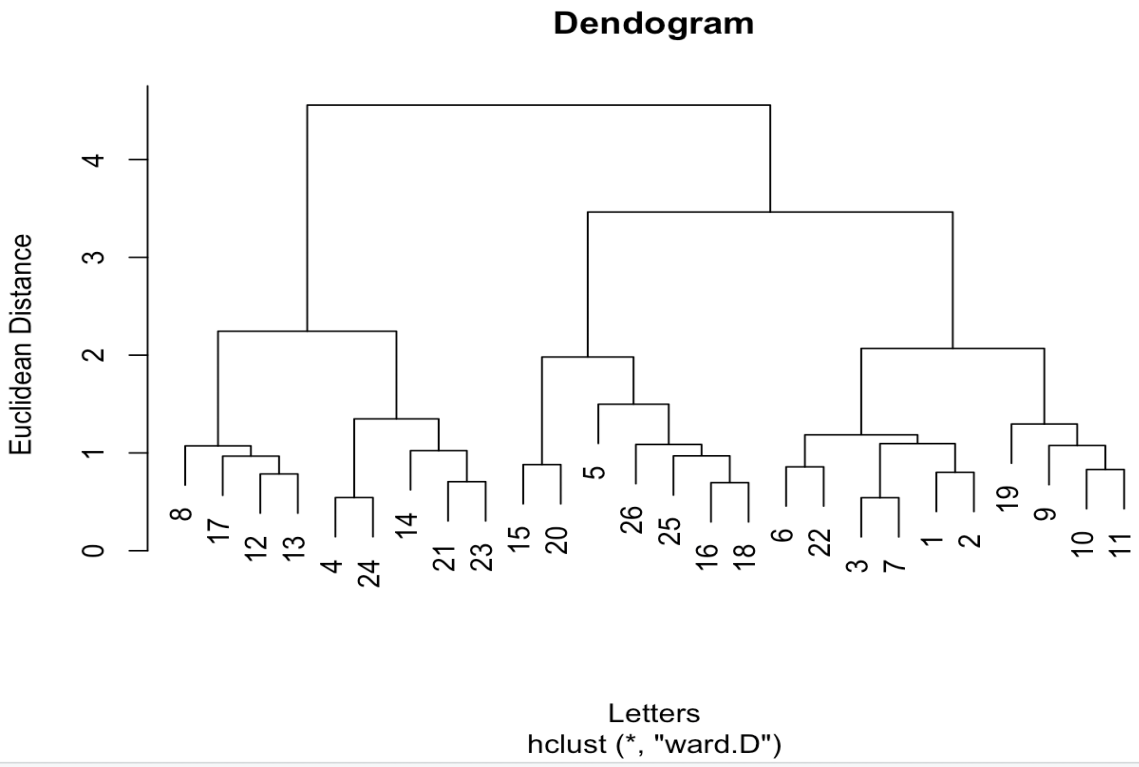
Obervation:

The 23rd letter – ‘w’ indicates number of natural clusters
Reason: After plotting K = 15 to 26, we observe an ‘*elbow’* in the shape of the graph at 23 – which suggests the number of natural clusters.

3. Hierarchical Clustering. Use hclust() with the dAverage distance method to form a dendogram of the 26 means from the last run of k-means. These means are given in the ‘centers’ value returned by kmeans() -- i.e., kmeans$centers.

```r
az_char <- read.table("~/Downloads/az-5000.txt" , TRUE)

#Removing the first column of letters data frame
new_az_char <- az_char[,-1]
dim(new_az_char)
head(new_az_char,5)

#performing kmeans and determining the number of clusters
fit <- vector()
for (i in 2:26)
{
  kmeans_output <- kmeans(new_az_char, centers=i, iter.max=26)
}

#ward. D method hieracrhical clustering and showing the dendrogram with cluster
#index on each leaf

d <- dist(kmeans_output$centers, method = "euclidean")
dendogram <- hclust(d, method="ward.D")
plot(dendogram,
     main=paste('Dendogram'),
     xlab ="Letters",
     ylab="Euclidean Distance")
```

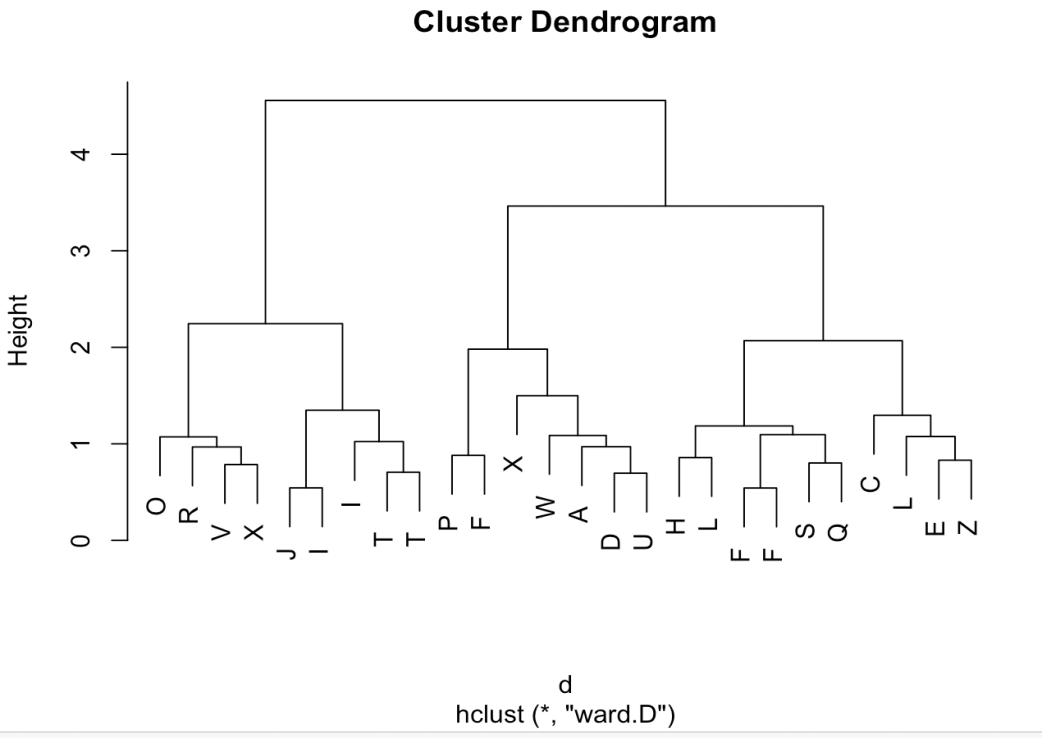
(a) Show the dendogram. There should be a cluster number (or index) on each leaf.



(b) Assigning letter labels to each center (node in the dendogram). Function kmeans() also returns a `cluster' vector with integers indicating the cluster to which each point is allocated. Build a 26x26 matrix (i.e., letters by cluster-number) to determine the most common letter

associated with each given cluster. Show the dendogram above with these letters instead of the cluster numbers.

```
#creating a 26 x 26 matrix (letters by cluster numbers) by extracting cluster vector
#from kmeans_output and
#letter vector from character data set and populating the matrix using a for loop from 1 to 5000
#for each character sample

letter_mat <- az_char[,1]
num_cluster <- kmeans_output$cluster
initial_mat <- matrix(0,26,26)
rownames(initial_mat) <- LETTERS

for(k in 1:5000)
{
  initial_mat[letter_mat[k], num_cluster[k]] <-  initial_mat[letter_mat[k], num_cluster[k]] + 1
}

#showing the dendogram with letters by getting the letter position where maximum of the
#26 x 26 letter
#matrix occurs plotting the fit with the corresponding label
letters <- c()
for(i in 1:26)
{
  letters[i] <- which.max(initial_mat[,i])
}

plot(dendogram, labels=LETTERS[letters])


tab <- read.table(file, sep="\t", header=FALSE, comment.char="#",
                  na.strings=".", stringsAsFactors=FALSE,
                  quote="", fill=FALSE)
```



(c) Report two observations (analysis) of what the dendogram tells you (anything you find

interesting).
Ans.
It shows that J and I are rooted under one branch which means they are similar enough. Same is the case with P and F.

(d) In the dendogram with letter labels you will notice that some letters are missing and some are duplicated. Determine to what cluster we should assign each missing letter.
Ans.

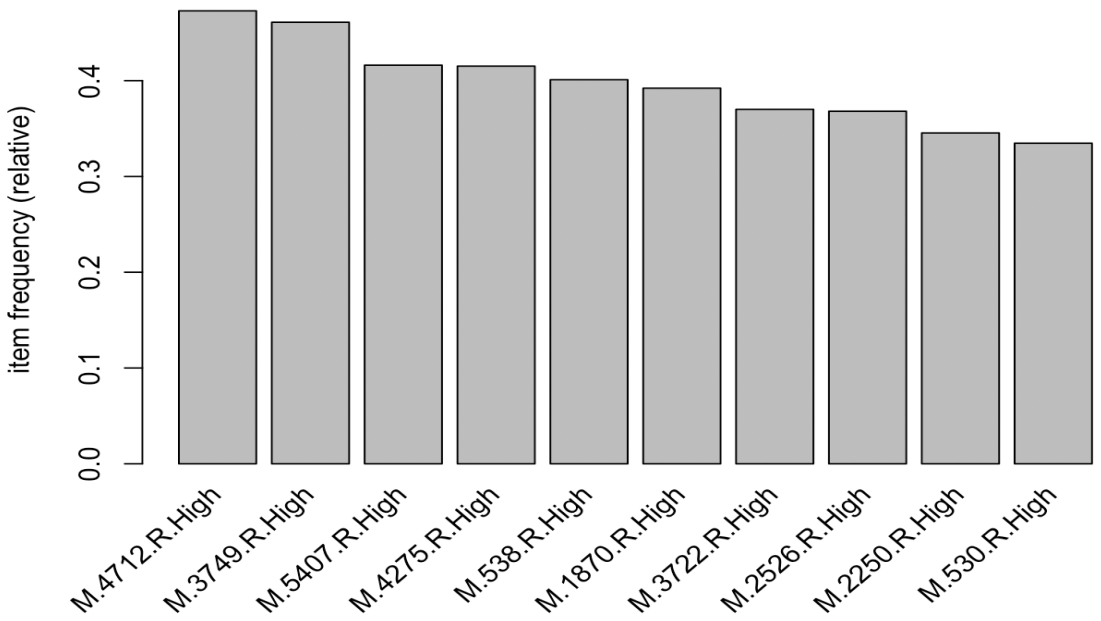
```
> print(initial_mat)
  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26]
A    7    0    0   12    0    0    0    1    0     0     9     0     0    13     1     4    89    32     0     5     4     0     0     1     0     0
B    0    5    1    0  138    1    0    3    0     2     0     0     0     0     1     4     0     0     0     0     0    17    11     0     1     0
C    3    0    0    3    0    0    3    0    0    10     0     0     0     0     1     0     1     0     0     0   177     0     1     0     0     0
D    6    1    1    0    1    1    2    1    0     0     0     1     0    30     3     0    19     0     1   116     0     1     1     0     1     1
E    1    1    0    0    0    2    0    8    0   169     1     0     0     0     0     0     0     3     2     0     5     0     3     0     0     0
F    0    0    0   98    0    7    1    1   14     1     1     1     0     3    13     0     0     0    27     0     1     0    19     0     0     0
G    0  101    0   36    0    0    1    0    0     0     0     0     0     3     2     0     0     2     0     0     0    31     0    15     0     1
H    3    0    0    0  143    0    4    1    1     2     0     0     0     2     0     9     3     1     0     0     1     0     8     0     0     0
I    2    0   40    0    0    1    9    1    2     0     0     2   113     0    20     0     0     0     1     0     0     0     1     0     0     0
J    0    2   10    4    0    1    0    0    0     0     0     1   152     1     9     0     0     0     0     0     0    13     0     0     0     1
K    1    4    0    0  110    0    3    2    2     0     0     0     0     3     2    28     0     1     0     0     0     0     6     9     0     2
L    1    0    0    0    2    0  148   17    1     0     0     0     0     0     0     0     0     0     0     1     3     0    29     0     0     0
M  122    0    0    0    0    0    0    3    0     0     0     0     0     1     0    64     1     0     0     1     0     0     0     0     0     0
N   19    0    1    0   10    0    0    2    5     0     0     2     0     0     0   117     5     0     1     0     0     0     0     2     0     0
O    2    2    0    2    0   15    0    4    1     0    12     3     0     0     2     0     2     0     1     1     3     1     0     0   155     5
P    0    2    0    0    0    2    1    1  120     0     0     4     0     0     2     1     1     0    73     0     0     0     0     0     0     1
Q    0   18    0   17    1    0    0    2    2     0     0     0     0     0     0     2     0     0     0     0     0     1     3   151     0     0
R    1    0    0    1    1    1    0    9    7     3     0   161     0     0    13     0     0     1     7     0     0     0     1     1     0    10
S    0    1    0    3    0    1    4    8    0     0     0     1     0     0     0     0     0     1     0     0     0   186     1     1     0     0
T    0    0   56    8    0   22    3    3    0     1     0     5     3     6    69     0     0     1     0     0     0     0     6     0     5     3
U   17    0    1    0   17    1    0    1    0     0     0     0     0     0     0   143    10     0     0     1     0     0     0     3     0    26
V    2    0    0    0    0    0    0    0    0     0     0    17     0     0     6     4     0     0     0     0     0     0     0     1     0   170
W  169    0    0    0    0    0    0    0    0     0     0     0     0     0     0     4     0     0     0     0     0     0     0     0     0    25
X    3    5    0    1    0    5    0   11    0     0    52     2     0    59     8     3     5     0     0     1     2     0     0     3     4    19
Y    0  150    0   20    0    0    1    1    0     0     0     2     0     0     0     0     0     1     0     0     0    10     0     6     0     0
Z    0    1    0    9    0   41    0    0    0     0     0     0     0     0     1     0     0   108     0     0     0     0     0     0     0     0
```

We can see 2 T's, 2 I's in the dendogram.
G is missing, Y is missing and K is missing.
As can be seen from above matrix G with highest no of cluster value 101 should be assigned to cluster of B.
Y should be assigned with cluster of B with value of 150.
K must be assigned to E with cluster no 110.

4. Association Rules. The data files "movies.txt" and "ratingsAsBasket.txt" contain movie ratings from Netflix (Be sure to read the license note from that company which accompanies the data.). The goal of this problem is to learn how to use the Apriori algorithms to find association rules. Each line in "ratingsAsBasket.txt" represents all the movie ratings provided by a single user. Not all users rated the same number of movies. Movies are represented by numbers and ratings have been summarized by three values: 'Low", "Medium", and "High." Thus, a "basket" will be the set of all movies rated by one person, and the "items" are the pairs <movieId, rating indicator>.
The R function apriori from the arules package provides the apriori functionality using Borgelt's Implementation.

(a) Use the read.transactions() command to load the data into R. Show a summary of the dataset with the command summary(). Using information from this summary, find out:
- Number of "baskets"
- Describe most frequent item (i.e., title, rating, frequency)
- Minimum/Maximum/Average number of movies rated by one rater

```r
set.seed(1)

#Loading the movies data and creating a summary
library(arules)
movies_data<- read.transactions("~/Downloads/ratingsAsBasket.txt", format = "basket", sep=NULL)
inspect(movies_data)
summary(movies_data)
itemFrequencyPlot(movies_data,topN=10)
```



```
> summary(movies_data)
transactions as itemMatrix in sparse format with
 10000 rows (elements/itemsets/transactions) and
 15500 columns (items) and a density of 0.009911529

most frequent items:
M.4712.R.High M.3749.R.High M.5407.R.High M.4275.R.High  M.538.R.High       (Other)
         4729          4610          4162          4152          4010       1514624

element (itemset/transaction) length distribution:
sizes
  20   21   22   23   24   25   26   27   28   29   30   31   32   33   34   35   36   37   38
  64  110   77   71   81   71   77  100   96   85  108  112   99  100  110   93   83   84   95
  39   40   41   42   43   44   45   46   47   48   49   50   51   52   53   54   55   56   57
 115   80  110   80   99   83   84   72   69   82   66   88   71   66   60   67   71   76   61
  58   59   60   61   62   63   64   65   66   67   68   69   70   71   72   73   74   75   76
  81   76   60   58   55   74   48   57   68   54   48   44   48   56   60   50   63   43   56
  77   78   79   80   81   82   83   84   85   86   87   88   89   90   91   92   93   94   95
  56   52   53   44   51   43   36   34   64   34   41   42   43   39   46   27   42   40   36
  96   97   98   99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114
  42   38   31   37   36   36   40   41   36   42   38   36   35   40   33   38   37   34   32
 115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133
  35   34   30   41   32   38   33   39   24   29   20   37   31   32   20   40   20   33   25
 134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152
  26   24   26   30   24   29   26   18   22   30   24   19   28   23   11   24   26   27   27
 153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171
  20   35   21   27   21   19   23   21   11   18   24   24   22   15   29   13   17   26   16
 172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190
  20   16   10   13   22   23   16   19   15   14   10   24   20   21   16   17   15   12   14
 191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209
  16   15   13   13   13   19   15   12    7   14   24   13   15    9   11   17   14   12   13
 210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228
  19   12   10   14    9    5   14   16   12   15   17   16   12   14    8   13   11   14    7
 229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247
  14   11   15   18   12    8   15   12   16    8    2    7    7   10   13   16   15    8   11
 248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266
  10    7    8   12   13    9   11    9    6    9   15   11   11   11    9    5    3    9    9
 267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285
   7    8    8    5    5    9    7    2    4   12    6   13    8    6    6    6    8    5    5
```

```
 583  586  588  589  590  592  593  595  597  599  600  602  603  604  606  608  609  610  611
   4    1    1    1    2    1    1    3    1    1    2    2    2    1    1    3    1    2    1
 612  613  614  615  616  617  618  620  621  622  624  625  628  629  630  632  634  635  637
   1    1    2    2    2    2    3    2    1    1    1    2    1    3    3    1    1    1    1
 638  639  640  641  642  643  644  645  646  647  650  652  653  655  657  658  659  660  661
   3    3    1    1    2    1    1    1    2    2    1    1    2    2    2    1    2    3    1
 662  663  665  667  668  669  672  675  676  677  679  681  686  687  692  694  695  697  698
   1    1    1    1    2    1    1    1    1    2    3    1    1    1    1    1    1    2    1
 701  703  705  707  708  709  712  715  716  718  721  722  723  727  728  729  730  731  732
   2    1    1    3    1    1    3    1    1    1    5    1    1    1    3    1    1    1    1
 733  734  736  737  738  741  742  745  746  750  751  753  754  755  758  759  761  762  763
   1    1    1    1    3    1    1    1    2    1    1    1    1    1    1    3    3    2    2
 764  766  769  771  774  775  776  783  787  788  789  792  793  794  797  806  810  811  815
   2    1    1    1    1    1    2    1    2    1    2    1    1    2    1    2    2    1    1
 818  819  820  821  823  830  832  834  835  837  838  840  842  843  845  847  849  853  855
   2    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    2    1    1
 857  860  863  864  871  878  885  892  896  901  906  907  912  923  924  925  927  932  934
   1    1    2    1    1    2    1    1    1    1    1    1    1    1    1    1    1    1    1
 937  946  952  953  962  964  982  985  986  988  994  995  999 1000 1002 1006 1017 1018 1019
   1    1    1    1    1    4    1    1    2    1    1    1    1    1    1    1    3    1    1
1023 1025 1028 1040 1042 1050 1056 1058 1061 1062 1068 1082 1083 1086 1097 1101 1105 1117 1120
   1    1    1    1    1    1    1    1    1    1    1    1    1    2    1    1    1    1    1
1121 1130 1132 1156 1162 1163 1164 1176 1179 1181 1182 1186 1207 1208 1209 1212 1216 1219 1225
   1    1    1    1    1    1    1    1    1    1    1    2    1    1    1    1    1    1    1
1230 1245 1255 1272 1285 1292 1306 1315 1325 1339 1359 1366 1392 1443 1482 1493 1513 1539 1558
   1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    2    1
1565 1648 1653 1658 1666 1709 1852 1945 1972 2003 2027 2087 2106 2267 2289
   1    1    1    1    1    1    1    1    1    1    1    1    1    1    1

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   20.0    47.0    92.0   153.6   183.0  2289.0

includes extended item information - examples:
      labels
1 M.1.R.High
2  M.1.R.Low
3  M.1.R.Med
```

The most frequent item is the Matrix (movieID - 4712) with 4729 ratings of "High"
The mininmum number of movies rated by one user is 20
The maximum number of movies rated by one user is 2289
The average number of movies rated by one user is 153.6
The number of "baskets" is 10000

(b) Use the apriori() command to build association rules from the dataset (default values for the parameters are ok but feel free to experiment). Use the inspect() function to view the actual results of the modeling.
- Show the top-10 rules
- Substitute the movie titles for one of these top-10 rules. Comment.

```
set.seed(1)

#Loading the movies data and creating a summary
library(arules)
movies_data<- read.transactions("~/Downloads/ratingsAsBasket.txt", format = "basket", sep=NULL)
summary(movies_data)
itemFrequencyPlot(movies_data,topN=10)
#The most frequent item is the Matrix (movieID - 4712) with 4729 ratings of "High"
#The mininmum number of movies rated by one user is
#The number of "baskets" is

#building the association rules from the movies dataset
rules_association <- apriori(movies_data, parameter = list(support= 0.1, confidence = 0.78,
                                                           target = "rules"))
summary(rules_association)

inspect(head(rules_association,10))
```

```
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target   ext
       0.78    0.1    1 none FALSE            TRUE       5     0.1      1     10  rules FALSE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 1000

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[15500 item(s), 10000 transaction(s)] done [0.47s].
sorting and recoding items ... [253 item(s)] done [0.02s].
creating transaction tree ... done [0.01s].
checking subsets of size 1 2 3 4 5 6 done [0.50s].
writing ... [931 rule(s)] done [0.00s].
creating S4 object  ... done [0.02s].
> summary(rules_association)
set of 931 rules

rule length distribution (lhs + rhs):sizes
  2   3   4   5
 11 320 547  53

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   2.00    3.00    4.00    3.69    4.00    5.00

summary of quality measures:
    support         confidence          lift           count
 Min.   :0.1000   Min.   :0.7800   Min.   :1.650   Min.   :1000
 1st Qu.:0.1038   1st Qu.:0.7919   1st Qu.:1.942   1st Qu.:1038
 Median :0.1087   Median :0.8080   Median :2.042   Median :1087
 Mean   :0.1122   Mean   :0.8118   Mean   :2.103   Mean   :1122
 3rd Qu.:0.1169   3rd Qu.:0.8279   3rd Qu.:2.255   3rd Qu.:1169
 Max.   :0.1754   Max.   :0.8806   Max.   :3.143   Max.   :1754

mining info:
        data ntransactions support confidence
 movies_data         10000     0.1       0.78
>
> inspect(head(rules_association,10))
     lhs                rhs              support confidence lift     count
[1]  {M.4021.R.High} => {M.4275.R.High} 0.1096  0.7890569  1.900426 1096
[2]  {M.3670.R.High} => {M.1870.R.High} 0.1121  0.7888811  2.011425 1121
[3]  {M.3816.R.High} => {M.3749.R.High} 0.1230  0.8698727  1.886926 1230
[4]  {M.2528.R.High} => {M.4275.R.High} 0.1042  0.7876039  1.896927 1042
[5]  {M.3978.R.High} => {M.4275.R.High} 0.1309  0.7885542  1.899215 1309
[6]  {M.496.R.High}  => {M.4275.R.High} 0.1071  0.7951002  1.914981 1071
[7]  {M.4033.R.High} => {M.4275.R.High} 0.1235  0.8178808  1.969848 1235
[8]  {M.2175.R.High} => {M.2526.R.High} 0.1405  0.8126084  2.207575 1405
[9]  {M.3656.R.High} => {M.2526.R.High} 0.1421  0.7907624  2.148227 1421
[10] {M.3605.R.High} => {M.4275.R.High} 0.1611  0.7824186  1.884438 1611
```

The user who purchase Movie 4021- The Sting is most likely to purchase the Movie 4275 - The Silence of the Lambs.

(c) Explain what is “lift’ of a rule. Use the subset() command to list all the rules which have a lift greater that 3.0

Ans.
Lift is a measure of the performance of a targeting model (association rule) at predicting or classifying cases as having an enhanced response (with respect to the population as a whole), measured against a random choice targeting model.
- Substitute the movie titles for one of these rules. Comment.

```
> rules_lift <- subset(rules_association, subset = lift > 3)
> inspect(rules_lift)
     lhs                                                rhs             support confidence lift
[1]  {M.1817.R.High,M.647.R.High}                    => {M.646.R.High} 0.1026  0.8234350  3.057687
[2]  {M.2936.R.High,M.647.R.High}                    => {M.646.R.High} 0.1164  0.8185654  3.039604
[3]  {M.2250.R.High,M.2936.R.High,M.647.R.High}      => {M.646.R.High} 0.1025  0.8464079  3.142993
[4]  {M.2250.R.High,M.2749.R.High,M.647.R.High}      => {M.646.R.High} 0.1006  0.8293487  3.079646
[5]  {M.2526.R.High,M.2749.R.High,M.647.R.High}      => {M.646.R.High} 0.1007  0.8440905  3.134387
[6]  {M.2250.R.High,M.2526.R.High,M.647.R.High}      => {M.646.R.High} 0.1158  0.8324946  3.091328
[7]  {M.2250.R.High,M.5407.R.High,M.647.R.High}      => {M.646.R.High} 0.1038  0.8166798  3.032602
[8]  {M.1870.R.High,M.2250.R.High,M.647.R.High}      => {M.646.R.High} 0.1084  0.8181132  3.037925
[9]  {M.2250.R.High,M.4275.R.High,M.647.R.High}      => {M.646.R.High} 0.1157  0.8390138  3.115536
[10] {M.2250.R.High,M.4712.R.High,M.647.R.High}      => {M.646.R.High} 0.1130  0.8242159  3.060586
[11] {M.2526.R.High,M.5407.R.High,M.647.R.High}      => {M.646.R.High} 0.1012  0.8214286  3.050236
[12] {M.1870.R.High,M.2526.R.High,M.647.R.High}      => {M.646.R.High} 0.1072  0.8195719  3.043341
[13] {M.2526.R.High,M.4275.R.High,M.647.R.High}      => {M.646.R.High} 0.1119  0.8369484  3.107866
[14] {M.2526.R.High,M.4712.R.High,M.647.R.High}      => {M.646.R.High} 0.1075  0.8231240  3.056532
[15] {M.4275.R.High,M.5407.R.High,M.647.R.High}      => {M.646.R.High} 0.1066  0.8149847  3.026308
[16] {M.1870.R.High,M.4275.R.High,M.647.R.High}      => {M.646.R.High} 0.1085  0.8238421  3.059198
[17] {M.4275.R.High,M.4712.R.High,M.647.R.High}      => {M.646.R.High} 0.1112  0.8261516  3.067774
     count
[1]  1026
[2]  1164
[3]  1025
[4]  1006
[5]  1007
[6]  1158
[7]  1038
[8]  1084
[9]  1157
[10] 1130
[11] 1012
[12] 1072
[13] 1119
[14] 1075
[15] 1066
[16] 1085
[17] 1112
```

```r
#finding all the rules with lift greater than 3.
rules_lift <- subset(rules_association, subset = lift > 3)
inspect(rules_lift)
summary(rules_lift)
```

```
> summary(rules_lift)
set of 17 rules

rule length distribution (lhs + rhs):sizes
 3  4
 2 15

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.000   4.000   4.000   3.882   4.000   4.000

summary of quality measures:
    support         confidence          lift           count
 Min.   :0.1006   Min.   :0.8150   Min.   :3.026   Min.   :1006
 1st Qu.:0.1026   1st Qu.:0.8196   1st Qu.:3.043   1st Qu.:1026
 Median :0.1075   Median :0.8238   Median :3.059   Median :1075
 Mean   :0.1079   Mean   :0.8270   Mean   :3.071   Mean   :1079
 3rd Qu.:0.1119   3rd Qu.:0.8325   3rd Qu.:3.091   3rd Qu.:1119
 Max.   :0.1164   Max.   :0.8464   Max.   :3.143   Max.   :1164

mining info:
        data ntransactions support confidence
 movies_data         10000     0.1       0.78
```

5. Collaborative Filtering Recommender Systems. The R package recommenderlab provides functionality for Collaborative Filtering style of recommender systems. Each line in “ratings.txt” represents one movie rating provided by a single user in the form of <user_id, movie_id, rating> Tupples.

a) Use the scan() command to load the data into R as a list. The use the sparseMatrix() from the Matrix package to convert this list into a sparse matrix. Show the dimensions of the resulting matrix, and fill the row and column names (simple numeric labels will suffice).

```
#5
library("recommenderlab")
movie_rating_data <- scan("~/Downloads/ratings.txt",sep = "|",
                          what = list(userid = 0,movieid = 0, rating=0))
data_sparse <- sparseMatrix(i=as.integer(movie_rating_data$userid),
                            j=as.integer(movie_rating_data$movieid) ,
                            x = as.integer(movie_rating_data$rating),
                            dimnames = list(userid = paste("user_no", 1:10000, sep = ''),
                        movieid= paste("movies_no", 1:7223, sep = '')))
print(data_sparse)
str(data_sparse)
dim(data_sparse)
```

```
> movie_rating_data <- scan("~/Downloads/ratings.txt",sep = "|",
+                           what = list(userid = 0,movieid = 0, rating=0))
Read 1536287 records
> data_sparse <- sparseMatrix(i=as.integer(movie_rating_data$userid),
+                             j=as.integer(movie_rating_data$movieid) ,
+                             x = as.integer(movie_rating_data$rating),
+                             dimnames = list(userid = paste("user_no", 1:10000, sep = ''),
+                         movieid= paste("movies_no", 1:7223, sep = '')))
> print(data_sparse)
10000 x 7223 sparse Matrix of class "dgCMatrix"
   [[ suppressing 41 column names ‘movies_no1’, ‘movies_no2’, ‘movies_no3’ ... ]]
   [[ suppressing 41 column names ‘movies_no1’, ‘movies_no2’, ‘movies_no3’ ... ]]

user_no1  . . . . . ......
user_no2  . . . . . ......
user_no3  . . . . . ......
user_no4  . . . . . ......
user_no5  . . . . . ......
user_no6  . . . . . ......
user_no7  . . . . . ......
user_no8  . . . . . ......
user_no9  . . . . . ......
user_no10 . . . . . ......
user_no11 . . . . . ......
user_no12 . . . . . ......

 ..............................
 ........suppressing columns and rows in show(); maybe adjust 'options(max.print= *, width = *)'
 ..............................

   [[ suppressing 41 column names ‘movies_no1’, ‘movies_no2’, ‘movies_no3’ ... ]]

user_no9989  . . . . . ......
user_no9990  . . . . . ......
user_no9991  . . . . . ......
user_no9992  . . . . . ......
user_no9993  . . . . . ......
user_no9994  . . . . . ......
user_no9995  . . . . . ......
user_no9996  . . . . . ......
user_no9997  . . . . . ......
user_no9998  . . . . . ......
user_no9999  . . . . . ......
user_no10000 . . . . . ......
[illegible]  . . . . . ......
> str(data_sparse)
Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
  ..@ i       : int [1:1536287] 540 559 603 929 1167 1588 1902 2255 2571 2812 ...
  ..@ p       : int [1:7224] 0 26 28 28 30 30 30 37 37 37 ...
  ..@ Dim     : int [1:2] 10000 7223
  ..@ Dimnames:List of 2
  .. ..$ userid : chr [1:10000] "user_no1" "user_no2" "user_no3" "user_no4" ...
  .. ..$ movieid: chr [1:7223] "movies_no1" "movies_no2" "movies_no3" "movies_no4" ...
  ..@ x       : num [1:1536287] 5 4 3 1 5 5 3 5 4 5 ...
  ..@ factors : list()
> dim(data_sparse)
[1] 10000  7223
```

b) Create a user-based collaborative filtering ("UBCF") model and use it to identify the top 5 recommended movies for user #10000.

```
r_movie<- as(data_sparse, "realRatingMatrix")
head(as(r_movie, "data.frame"))
model_movie<- Recommender(data = r_movie, method = "UBCF",
                          parameter = list(method = "cosine"))
names(getModel(model_movie))

recc_predicted <- predict(object = model_movie,
                          newdata = r_movie[10000,] ,n = 5)
as(recc_predicted, "list")
recc_predicted1 <- predict(object = model_movie,
                          newdata = r_movie[500,] ,n = 1)
as(recc_predicted1, "list")
```

```
> as(recc_predicted, "list")
$user_no10000
[1] "movies_no5407" "movies_no4712" "movies_no4716" "movies_no1401" "movies_no538"
```

Movies are - Saving Private Ryan, The Matrix, The Mummy, Antz, True Lies.

c) What is the highest predicted rating movie for user #500?

```
> as(recc_predicted1, "list")
$user_no500
[1] "movies_no4716"
```

Movie 4716 - The Mummy.

6. Text Mining. The R package tm provides functionality for text mining applications. The data directories “rec.autos” and “rec.motorcycles” contain a subset of the 20 Newsgroups data set.

(a) Use the DirSource() and Corpus() tm commands to load the data into R (you may want to use the reader = readPlain argument in the call to Corpus()). Confirm all the documents were read into the corpus by
- Showing the output of length(news.corpus), where news.corpus is the variable where you stored the output of Corpus()
- Printing the corpus entry corresponding to document rec.autos/103806 (hint: use names(news.corpus) and [[]] indexing to access corpus entries).

```
library(tm)
pathtodata= c("~/Downloads/rec.autos","~/Downloads/rec.motorcycles")
newsCorpus = Corpus(DirSource(pathtodata, recursive = TRUE),
                    readerControl = list(reader = readPlain))
length(newsCorpus)
names(newsCorpus)
```

```
> pathtodata= c("~/Downloads/rec.autos","~/Downloads/rec.motorcycles")
> newsCorpus = Corpus(DirSource(pathtodata, recursive = TRUE),
+                     readerControl = list(reader = readPlain))
> length(newsCorpus)
[1] 1986
```

(b) Use the tm_map() command to apply the following preprocessing transformations of the corpus (in order): removePunctuation, removeNumbers, tolower, removeWords (with argument stopwords("english")) and stripWhitespace. After each step, print document rec.autos/103806 to

confirm the effect of the operation.

```
> strwrap(newsCorpus[["103806"]])
 [1] "From: cheekeen@tartarus.uwa.edu.au (Desmond Chan) Subject: Re: Honda clutch chatter"
 [2] "Organization: The University of Western Australia Lines: 8 NNTP-Posting-Host:"
 [3] "tartarus.uwa.edu.au X-Newsreader: NN version 6.4.19 #1"
 [4] ""
 [5] "I also experience this kinda problem in my 89 BMW 318. During cold start ups, the clutch"
 [6] "seems to be sticky and everytime i drive out, for about 5km, the clutch seems to stick"
 [7] "onto somewhere that if i depress the clutch, the whole chassis moves along. But after"
 [8] "preheating, it becomes smooth again. I think that your suggestion of being some humudity"
 [9] "is right but there should be some remedy. I also found out that my clutch is already"
[10] "thin but still alright for a couple grand more!"
>
> newsCorpus = tm_map(newsCorpus,removePunctuation)
> strwrap(newsCorpus[["103806"]])
 [1] "From cheekeentartarusuwaeduau Desmond Chan Subject Re Honda clutch chatter Organization"
 [2] "The University of Western Australia Lines 8 NNTPPostingHost tartarusuwaeduau XNewsreader"
 [3] "NN version 6419 1"
 [4] ""
 [5] "I also experience this kinda problem in my 89 BMW 318 During cold start ups the clutch"
 [6] "seems to be sticky and everytime i drive out for about 5km the clutch seems to stick"
 [7] "onto somewhere that if i depress the clutch the whole chassis moves along But after"
 [8] "preheating it becomes smooth again I think that your suggestion of being some humudity"
 [9] "is right but there should be some remedy I also found out that my clutch is already thin"
[10] "but still alright for a couple grand more"
>
> newsCorpus = tm_map(newsCorpus,removeNumbers)
> strwrap(newsCorpus[["103806"]])
 [1] "From cheekeentartarusuwaeduau Desmond Chan Subject Re Honda clutch chatter Organization"
 [2] "The University of Western Australia Lines NNTPPostingHost tartarusuwaeduau XNewsreader"
 [3] "NN version"
 [4] ""
 [5] "I also experience this kinda problem in my BMW During cold start ups the clutch seems to"
 [6] "be sticky and everytime i drive out for about km the clutch seems to stick onto"
 [7] "somewhere that if i depress the clutch the whole chassis moves along But after"
 [8] "preheating it becomes smooth again I think that your suggestion of being some humudity"
 [9] "is right but there should be some remedy I also found out that my clutch is already thin"
[10] "but still alright for a couple grand more"
>
> newsCorpus = tm_map(newsCorpus,tolower)
> strwrap(newsCorpus[["103806"]])
 [1] "from cheekeentartarusuwaeduau desmond chan subject re honda clutch chatter organization"
 [2] "the university of western australia lines nntppostinghost tartarusuwaeduau xnewsreader"
 [3] "nn version"
 [4] ""
 [5] "i also experience this kinda problem in my bmw during cold start ups the clutch seems to"
 [6] "be sticky and everytime i drive out for about km the clutch seems to stick onto"
 [7] "somewhere that if i depress the clutch the whole chassis moves along but after"
 [8] "preheating it becomes smooth again i think that your suggestion of being some humudity"
 [9] "is right but there should be some remedy i also found out that my clutch is already thin"
[10] "but still alright for a couple grand more"
>
```

```
[10] "  but still alright for a couple grand more"
> newsCorpus = tm_map(newsCorpus,removeWords,stopwords("english"))
>  strwrap(newsCorpus[["103806"]])
[1] "cheekeentartarusuwaeduau desmond chan subject re honda clutch chatter organization"
[2] "university western australia lines nntppostinghost tartarusuwaeduau xnewsreader nn"
[3] "version"
[4] ""
[5] "also experience kinda problem bmw cold start ups clutch seems sticky everytime drive km"
[6] "clutch seems stick onto somewhere depress clutch whole chassis moves along preheating"
[7] "becomes smooth think suggestion humudity right remedy also found clutch already thin"
[8] "still alright couple grand"
>
> newsCorpus = tm_map(newsCorpus,stripWhitespace)
> strwrap(newsCorpus[["103806"]])
[1] "cheekeentartarusuwaeduau desmond chan subject re honda clutch chatter organization"
[2] "university western australia lines nntppostinghost tartarusuwaeduau xnewsreader nn"
[3] "version also experience kinda problem bmw cold start ups clutch seems sticky everytime"
[4] "drive km clutch seems stick onto somewhere depress clutch whole chassis moves along"
[5] "preheating becomes smooth think suggestion humudity right remedy also found clutch"
[6] "already thin still alright couple grand"
```

(c) Use the DocumentTermMatrix() command to create a document-term matrix with TFIDF weights (also use minWordLength = 1 and minDocFreq = 1 arguments). Then
- Use dim() to report the dimensions of the resulting matrix. This is a very sparse matrix with far fewer non-zero entries than the dim() values would suggest.
- Use the inspect() command to confirm that the terms 'bmw' and 'clutch' are present in our rec.autos/103806 document but 'mother' isn't.

```
> docTermMatrix = DocumentTermMatrix(newsCorpus,control = list(weighting = weightTfIdf,
+                                                              minWordLength = 1,minDocFreq = 1))
> inspect(docTermMatrix )
<<DocumentTermMatrix (documents: 1986, terms: 22213)>>
Non-/sparse entries: 173995/43941023
Sparsity           : 100%
Maximal term length: 163
Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
Sample             :
        Terms
Docs            bike        can         car distribution        just       like         new
  101568 0.000000000 0.00000000 0.000000000  0.000000000 0.000000000 0.00000000 0.000000000
  101650 0.000000000 0.00000000 0.019006909  0.020020736 0.000000000 0.00000000 0.000000000
  102768 0.000000000 0.00000000 0.004650627  0.000000000 0.000000000 0.00000000 0.011635455
  102946 0.000000000 0.00000000 0.034286972  0.000000000 0.000000000 0.00000000 0.000000000
  103219 0.000000000 0.01621437 0.020572183  0.021669503 0.000000000 0.00000000 0.000000000
  103414 0.000000000 0.00000000 0.000000000  0.016157085 0.014095805 0.01282276 0.000000000
  104312 0.006975325 0.00428907 0.001813937  0.001910693 0.000000000 0.00000000 0.009076621
  104863 0.000000000 0.00000000 0.000000000  0.000000000 0.008034609 0.00000000 0.000000000
  104878 0.033705329 0.00000000 0.000000000  0.000000000 0.000000000 0.00000000 0.016447110
  105154 0.000000000 0.00000000 0.000000000  0.000000000 0.000000000 0.00000000 0.000000000
        Terms
Docs             one  university        will
  101568 0.000000000 0.000000000 0.000000000
  101650 0.000000000 0.017346743 0.000000000
  102768 0.000000000 0.000000000 0.000000000
  102946 0.000000000 0.000000000 0.000000000
  103219 0.000000000 0.000000000 0.000000000
  103414 0.013415659 0.000000000 0.000000000
  104312 0.000000000 0.000000000 0.002011246
  104863 0.007646926 0.007979502 0.000000000
  104878 0.000000000 0.023998501 0.014577757
  105154 0.000000000 0.067910653 0.041251949

> dim(docTermMatrix)
[1]  1986 22213
>
```

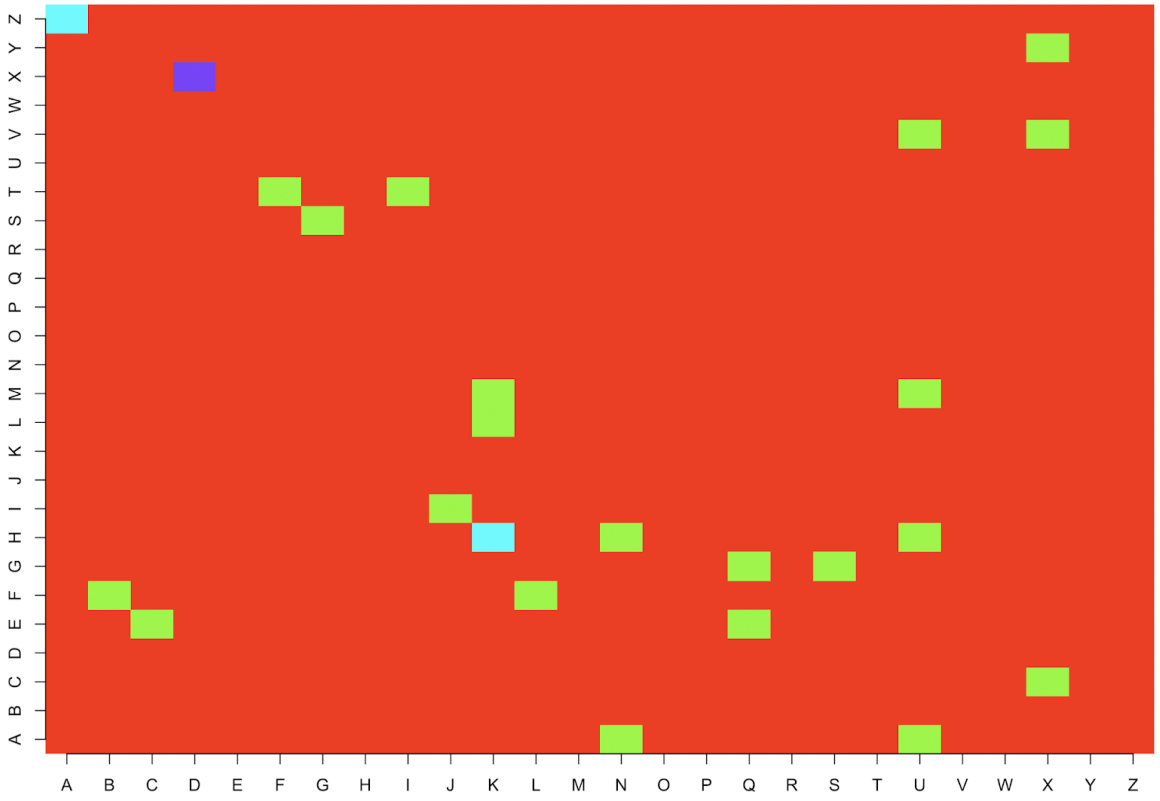
```
> x=inspect(docTermMatrix["103806",])
<<DocumentTermMatrix (documents: 1, terms: 22213)>>
Non-/sparse entries: 50/22163
Sparsity           : 100%
Maximal term length: 163
Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
Sample             :
        Terms
Docs       alright cheekeentartarusuwaeduau    clutch   depress   desmond  humudity preheating
  103806 0.1468542               0.1746605 0.4289088 0.1514688 0.1746605 0.1922044  0.1922044
        Terms
Docs        remedy tartarusuwaeduau      thin
  103806 0.1468542        0.1746605 0.1514688
> inspect(docTermMatrix["103806","bmw"])
<<DocumentTermMatrix (documents: 1, terms: 1)>>
Non-/sparse entries: 1/0
Sparsity           : 0%
Maximal term length: 3
Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
Sample             :
        Terms
Docs            bmw
  103806 0.05710895
> inspect(docTermMatrix["103806","clutch"])
<<DocumentTermMatrix (documents: 1, terms: 1)>>
Non-/sparse entries: 1/0
Sparsity           : 0%
Maximal term length: 6
Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
Sample             :
        Terms
Docs        clutch
  103806 0.4289088
> inspect(docTermMatrix["103806","mother"])
<<DocumentTermMatrix (documents: 1, terms: 1)>>
Non-/sparse entries: 0/1
Sparsity           : 100%
Maximal term length: 6
Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
Sample             :
        Terms
Docs     mother
  103806      0
```

7. Visualization. The R function image makes an image of a matrix. You can use it to visualize the off-diagonal elements in the (test set) confusion matrix – i.e., to easily spot the most problematic letters. To this end you should clear (e.g., set to 0) the diagonal elements before invoking the command.
(a) Turn in an image showing the non-zero entries colored. Use 4 colors only and no numeric ticks or labels. Use the axis command to add the letter labels to the plot.

```r
az_char <- read.table("~/Downloads/az-5000.txt", header = TRUE)
dim(Chars)
indices <- sample(1:nrow(az_char), size = (0.8*nrow(az_char)))##sampling 80% of the
data
training <- az_char[indices,] ##putting the sampled data in training vector
test <- az_char[-indices,] ##putting the rest dataset in the test_data

prior_vector <- c(rep(1/26, each=26))
print(prior_vector)

require(MASS)
lda_model <- lda(formula = char~., data = training, prior = prior_vector)
print(lda_model)

predict_lda_test =predict(lda_model, newdata = test)
conf_matrix_test <- table(test$char, predict_lda_test$class)
diag(conf_matrix_test) <- 0
print(conf_matrix_test)
labelpos <- 0:25
labelpos_std <- labelpos/25

image(conf_matrix_test,col=rainbow(4),axes=FALSE)

axis(1, labelpos_std, labels=LETTERS[1:26])
axis(2, labelpos_std, labels=LETTERS[1:26])
```

(b) Identify the color corresponding to the worst confusions and list the corresponding character Pairs.

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	0	0	0	2	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6
b	0	0	0	0	1	3	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
c	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
d	2	1	1	0	0	0	0	2	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	10	0	0
e	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f	1	0	0	1	0	0	0	0	0	0	0	2	0	0	0	1	0	0	0	3	0	0	0	0	2	0
g	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	2	0	4	1	0	0	0	1	2	0
h	1	0	0	0	1	0	0	0	0	0	2	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0
i	0	0	0	0	0	0	0	0	0	2	0	1	0	0	0	1	0	0	0	3	0	0	0	0	0	0
j	0	2	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	1	0
k	0	0	0	0	0	1	0	6	0	0	0	3	4	0	0	0	0	0	0	0	1	0	1	0	0	0
l	0	0	1	1	2	3	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
m	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0
n	3	0	0	1	0	0	0	3	0	0	2	0	2	0	0	0	0	0	0	0	1	0	0	0	0	0
o	0	0	1	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0
p	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	2	0	1	0	0	0	0	0	0
q	0	0	0	0	3	2	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0
r	0	0	0	0	0	0	0	0	2	0	0	0	0	2	0	0	2	0	0	1	1	1	0	0	0	0
s	0	0	0	0	0	0	3	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0
t	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1	2	0	0
u	3	0	0	0	0	0	0	3	0	0	1	0	3	1	1	0	0	0	0	0	0	4	0	0	0	0
v	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	2	0	1	0	0	0
w	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
x	0	2	4	1	1	0	0	0	0	1	2	0	0	0	0	0	0	2	0	1	1	5	0	0	4	0
y	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	2	2	0	0	0	0	0	0
z	0	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0



The confusion shown here is in between letters- X & D.