<u>DSC 441 – FUNDAMENTAL OF DATA SCIENCE</u>

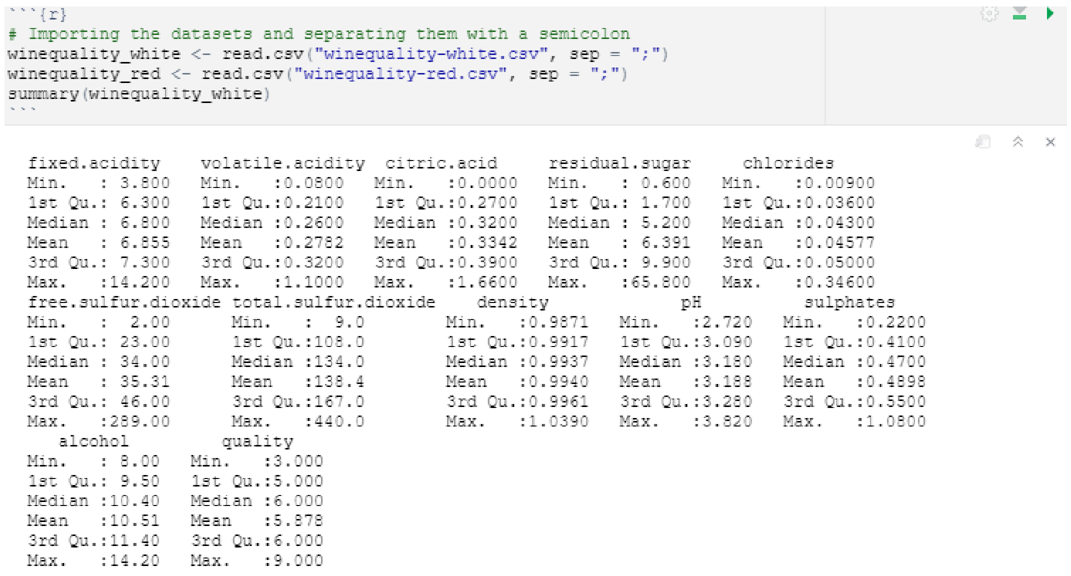
<u>HOMEWORK 4</u>

<u>NAME</u> – Goutham Selvakumar

<u>PROBLEM 1</u>

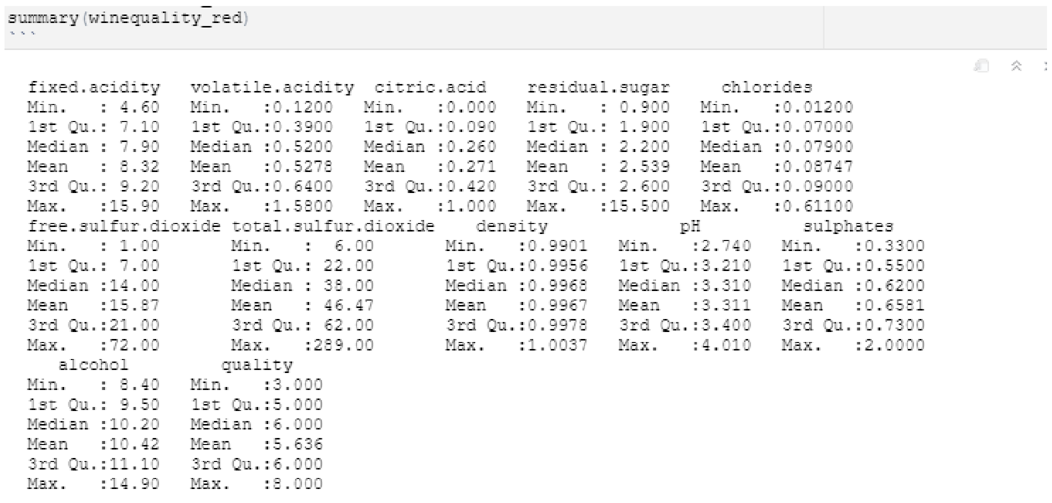
For this problem, you will tune and apply kNN and compare it to other classifiers. We will use the wine quality data, which has a number of measurements about chemical components in wine, plus a quality rating. There are separate files for red and white wines, so the first step is some data preparation

a. **For this problem, you will tune and apply kNN and compare it to other classifiers. We will use the wine quality data, which has a number of measurements about chemical components in wine, plus a quality rating. There are separate files for red and white wines, so the first step is some data preparation.**



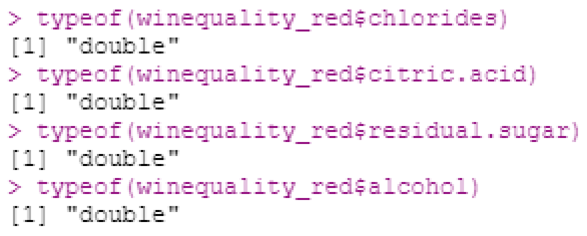
~~~
```{r}
# Importing the datasets and separating them with a semicolon
winequality_white <- read.csv("winequality-white.csv", sep = ";")
winequality_red <- read.csv("winequality-red.csv", sep = ";")
summary(winequality_white)
```
~~~

```
 fixed.acidity    volatile.acidity  citric.acid     residual.sugar     chlorides      
 Min.   : 3.800   Min.   :0.0800   Min.   :0.0000   Min.   : 0.600   Min.   :0.00900  
 1st Qu.: 6.300   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1.700   1st Qu.:0.03600  
 Median : 6.800   Median :0.2600   Median :0.3200   Median : 5.200   Median :0.04300  
 Mean   : 6.855   Mean   :0.2782   Mean   :0.3342   Mean   : 6.391   Mean   :0.04577  
 3rd Qu.: 7.300   3rd Qu.:0.3200   3rd Qu.:0.3900   3rd Qu.: 9.900   3rd Qu.:0.05000  
 Max.   :14.200   Max.   :1.1000   Max.   :1.6600   Max.   :65.800   Max.   :0.34600  
 free.sulfur.dioxide total.sulfur.dioxide    density              pH          sulphates     
 Min.   :  2.00      Min.   :  9.0        Min.   :0.9871   Min.   :2.720   Min.   :0.2200  
 1st Qu.: 23.00      1st Qu.:108.0        1st Qu.:0.9917   1st Qu.:3.090   1st Qu.:0.4100  
 Median : 34.00      Median :134.0        Median :0.9937   Median :3.180   Median :0.4700  
 Mean   : 35.31      Mean   :138.4        Mean   :0.9940   Mean   :3.188   Mean   :0.4898  
 3rd Qu.: 46.00      3rd Qu.:167.0        3rd Qu.:0.9961   3rd Qu.:3.280   3rd Qu.:0.5500  
 Max.   :289.00      Max.   :440.0        Max.   :1.0390   Max.   :3.820   Max.   :1.0800  
    alcohol         quality     
 Min.   : 8.00   Min.   :3.000  
 1st Qu.: 9.50   1st Qu.:5.000  
 Median :10.40   Median :6.000  
 Mean   :10.51   Mean   :5.878  
 3rd Qu.:11.40   3rd Qu.:6.000  
 Max.   :14.20   Max.   :9.000  
```



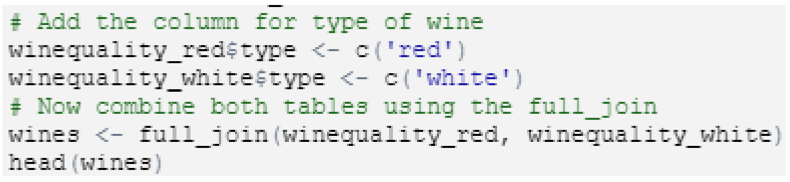
```
summary(winequality_red)
```
```

 fixed.acidity   volatile.acidity  citric.acid    residual.sugar     chlorides
 Min.   : 4.60   Min.   :0.1200   Min.   :0.000   Min.   : 0.900   Min.   :0.01200
 1st Qu.: 7.10   1st Qu.:0.3900   1st Qu.:0.090   1st Qu.: 1.900   1st Qu.:0.07000
 Median : 7.90   Median :0.5200   Median :0.260   Median : 2.200   Median :0.07900
 Mean   : 8.32   Mean   :0.5278   Mean   :0.271   Mean   : 2.539   Mean   :0.08747
 3rd Qu.: 9.20   3rd Qu.:0.6400   3rd Qu.:0.420   3rd Qu.: 2.600   3rd Qu.:0.09000
 Max.   :15.90   Max.   :1.5800   Max.   :1.000   Max.   :15.500   Max.   :0.61100
 free.sulfur.dioxide total.sulfur.dioxide    density             pH          sulphates
 Min.   : 1.00       Min.   :  6.00       Min.   :0.9901   Min.   :2.740   Min.   :0.3300
 1st Qu.: 7.00       1st Qu.: 22.00       1st Qu.:0.9956   1st Qu.:3.210   1st Qu.:0.5500
 Median :14.00       Median : 38.00       Median :0.9968   Median :3.310   Median :0.6200
 Mean   :15.87       Mean   : 46.47       Mean   :0.9967   Mean   :3.311   Mean   :0.6581
 3rd Qu.:21.00       3rd Qu.: 62.00       3rd Qu.:0.9978   3rd Qu.:3.400   3rd Qu.:0.7300
 Max.   :72.00       Max.   :289.00       Max.   :1.0037   Max.   :4.010   Max.   :2.0000
    alcohol         quality
 Min.   : 8.40   Min.   :3.000
 1st Qu.: 9.50   1st Qu.:5.000
 Median :10.20   Median :6.000
 Mean   :10.42   Mean   :5.636
 3rd Qu.:11.10   3rd Qu.:6.000
 Max.   :14.90   Max.   :8.000
```

Now, check the type of data in the columns of the wines dataset

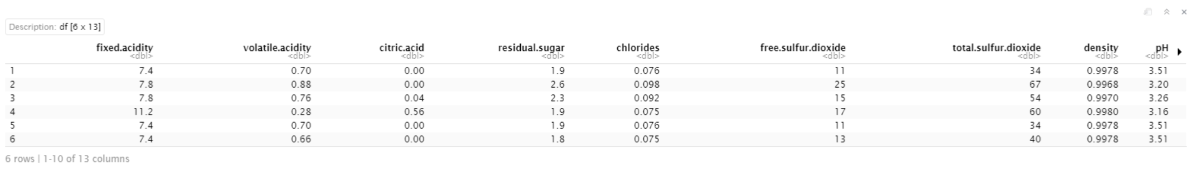


```
> typeof(winequality_red$chlorides)
[1] "double"
> typeof(winequality_red$citric.acid)
[1] "double"
> typeof(winequality_red$residual.sugar)
[1] "double"
> typeof(winequality_red$alcohol)
[1] "double"
```

Note that there is no NAs in the dataset. Now, let's add a column for the types of wines.



```
# Add the column for type of wine
winequality_red$type <- c('red')
winequality_white$type <- c('white')
# Now combine both tables using the full_join
wines <- full_join(winequality_red, winequality_white)
head(wines)
```

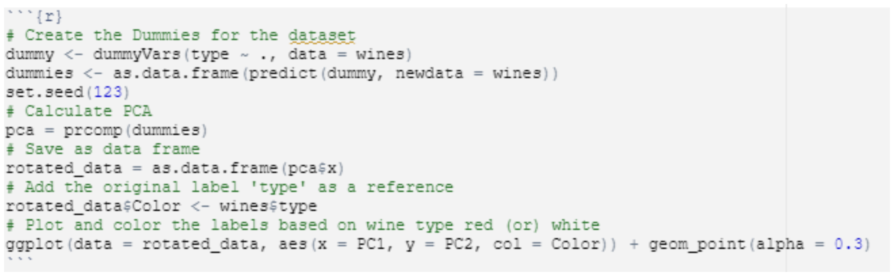


Description: df [6 x 13]

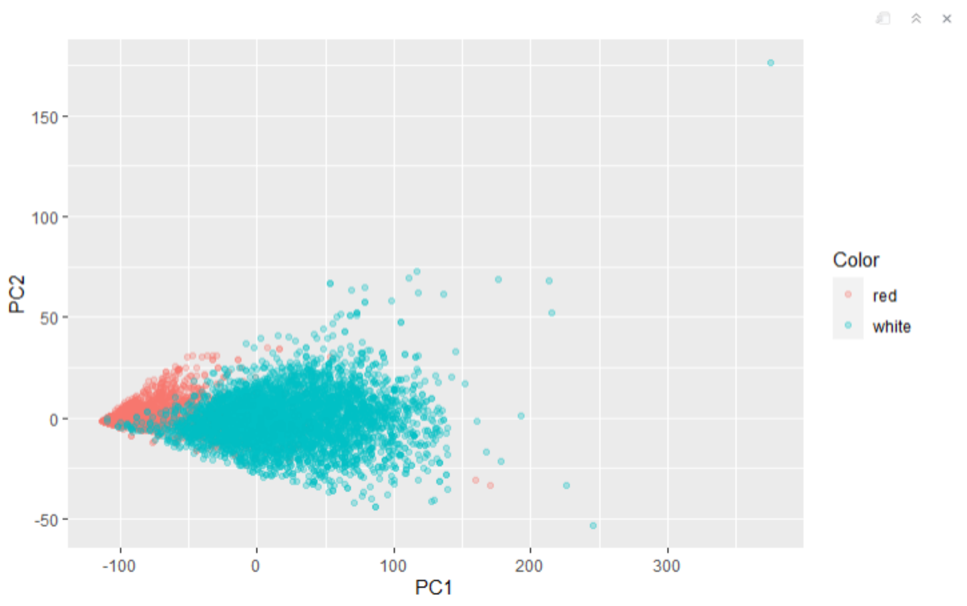
	fixed.acidity <dbl>	volatile.acidity <dbl>	citric.acid <dbl>	residual.sugar <dbl>	chlorides <dbl>	free.sulfur.dioxide <dbl>	total.sulfur.dioxide <dbl>	density <dbl>	pH <dbl>
1	7.4	0.70	0.00	1.9	0.076	11	34	0.9978	3.51
2	7.8	0.88	0.00	2.6	0.098	25	67	0.9968	3.20
3	7.8	0.76	0.04	2.3	0.092	15	54	0.9970	3.26
4	11.2	0.28	0.56	1.9	0.075	17	60	0.9980	3.16
5	7.4	0.70	0.00	1.9	0.076	11	34	0.9978	3.51
6	7.4	0.66	0.00	1.8	0.075	13	40	0.9978	3.51

6 rows | 1-10 of 13 columns

b. Use PCA to create a projection of the data to 2D and show a scatterplot with color showing the wine type.



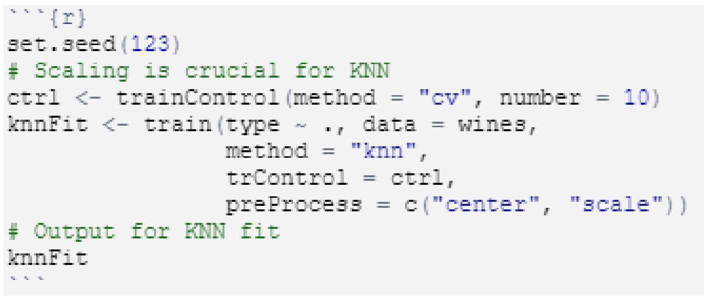
~~~
```{r}
# Create the Dummies for the dataset
dummy <- dummyVars(type ~ ., data = wines)
dummies <- as.data.frame(predict(dummy, newdata = wines))
set.seed(123)
# Calculate PCA
pca = prcomp(dummies)
# Save as data frame
rotated_data = as.data.frame(pca$x)
# Add the original label 'type' as a reference
rotated_data$Color <- wines$type
# Plot and color the labels based on wine type red (or) white
ggplot(data = rotated_data, aes(x = PC1, y = PC2, col = Color)) + geom_point(alpha = 0.3)
```
~~~



c. **We are going to try kNN, SVM and decision trees on this data. Based on the 'shape' of the data in the visualization from (b), which do you think will do best and why?**

I think the best method in this case may be to use KNN. The wines dataset is somewhat large so KNN may be faster and more efficient solution. Also, the non-linear relationship will allow for a more accurate prediction by using the nearest K neighbour to predict the type of wine. I've performed some sample work below that helped me also get to this conclusion, although SVM and KNN performed similarly in regards to accuracy. However, SVM's patterns may be limited in this case (we can discover this later).

<u>Finding the K nearest neighbour:</u>



~~~
```{r}
set.seed(123)
# Scaling is crucial for KNN
ctrl <- trainControl(method = "cv", number = 10)
knnFit <- train(type ~ ., data = wines,
                method = "knn",
                trControl = ctrl,
                preProcess = c("center", "scale"))
# Output for KNN fit
knnFit
```
~~~

```
k-Nearest Neighbors

6497 samples
  12 predictor
   2 classes: 'red', 'white'

Pre-processing: centered (12), scaled (12)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 5848, 5847, 5847, 5848, 5847, 5848, ...
Resampling results across tuning parameters:

  k  Accuracy   Kappa
  5  0.9923051  0.9792070
  7  0.9929205  0.9808827
  9  0.9930741  0.9813154

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 9.
```

SVM:

```
# Fit the Model
svm1 <- train(type ~., data = wines, method = "svmLinear")
# Evaluate the Fit
svm1
```

```
Support Vector Machines with Linear Kernel

6497 samples
  12 predictor
   2 classes: 'red', 'white'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 6497, 6497, 6497, 6497, 6497, 6497, ...
Resampling results:

  Accuracy  Kappa
  0.994937  0.9862915

Tuning parameter 'C' was held constant at a value of 1
```

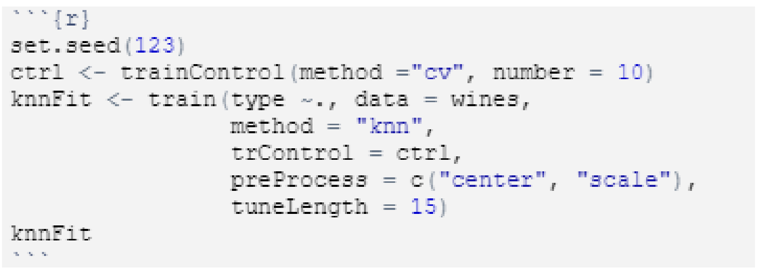
Decision Tree:

```
# Evaluation Method
train_control = trainControl(method = "cv", number = 10)
# Fit the Model
tree1 <- train(type ~., data = wines, method = "rpart", trControl = train_control)
tree1
```

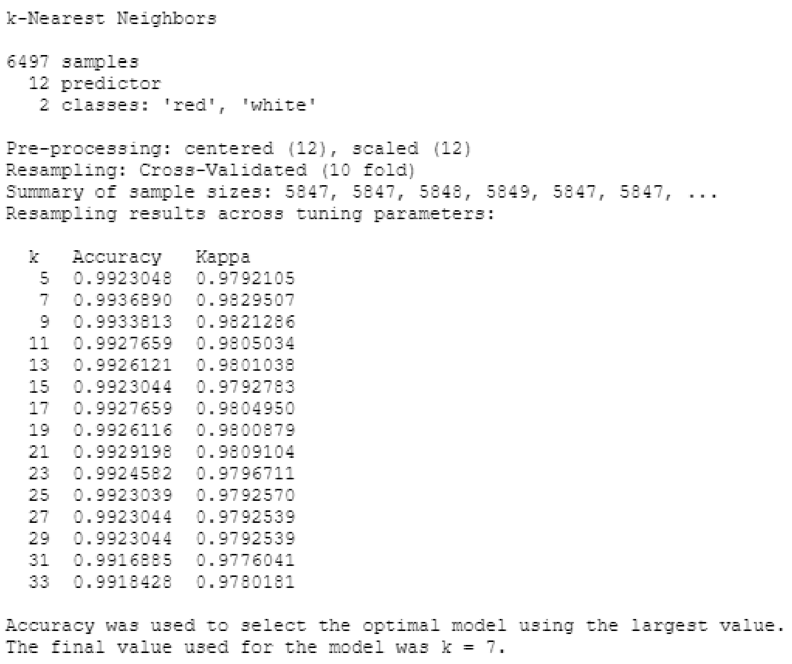
```
CART

6497 samples
  12 predictor
   2 classes: 'red', 'white'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 5847, 5848, 5847, 5847, 5848, 5847, ...
Resampling results across tuning parameters:

  cp          Accuracy   Kappa
  0.06253909  0.9465933  0.8498495
  0.06754221  0.9348963  0.8147365
  0.70043777  0.7855788  0.1500825

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.06253909.
```

d. Use kNN (tune k), use decision trees (basic rpart method is fine), and SVM (tune C) to predict type from the rest of the variables. Compare the accuracy values – is this what you expected? Can you explain it? Note: you will need to fix the columns names for rpart because it is not able to handle the underscores. This code will do the trick (assuming you called your data wine_quality): colnames(wine_quality) <- make.names(colnames(wine_quality)).

<u>KNN:</u>

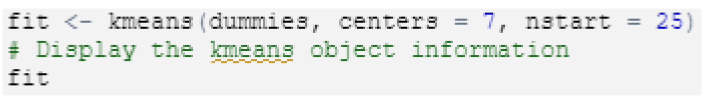


~~~
```{r}
set.seed(123)
ctrl <- trainControl(method ="cv", number = 10)
knnFit <- train(type ~., data = wines,
                method = "knn",
                trControl = ctrl,
                preProcess = c("center", "scale"),
                tuneLength = 15)
knnFit
```
~~~

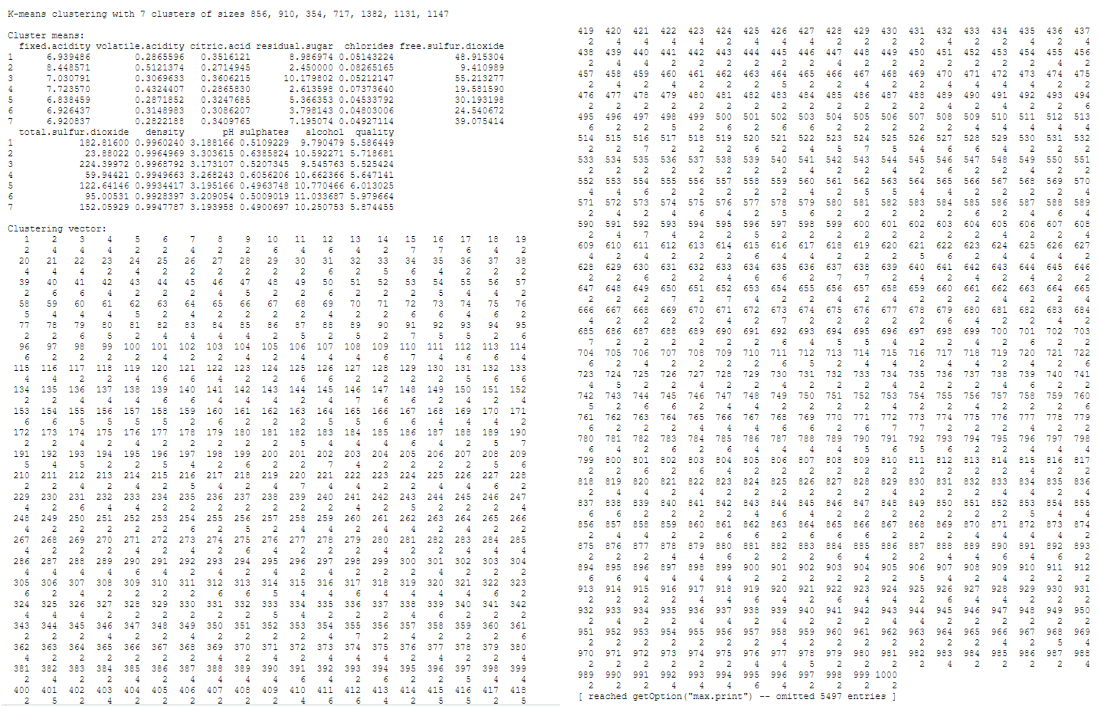


```
k-Nearest Neighbors

6497 samples
  12 predictor
   2 classes: 'red', 'white'

Pre-processing: centered (12), scaled (12)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 5847, 5847, 5848, 5849, 5847, 5847, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   5  0.9923048  0.9792105
   7  0.9936890  0.9829507
   9  0.9933813  0.9821286
  11  0.9927659  0.9805034
  13  0.9926121  0.9801038
  15  0.9923044  0.9792783
  17  0.9927659  0.9804950
  19  0.9926116  0.9800879
  21  0.9929198  0.9809104
  23  0.9924582  0.9796711
  25  0.9923039  0.9792570
  27  0.9923044  0.9792539
  29  0.9923044  0.9792539
  31  0.9916885  0.9776041
  33  0.9918428  0.9780181

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 7.
```

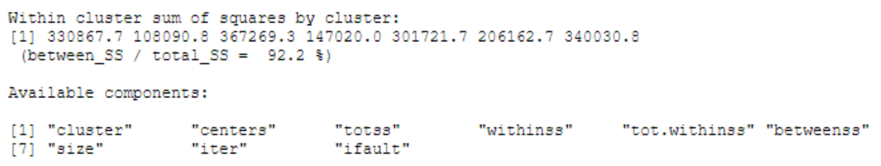
<u>For k = 7:</u>



```
fit <- kmeans(dummies, centers = 7, nstart = 25)
# Display the kmeans object information
fit
```



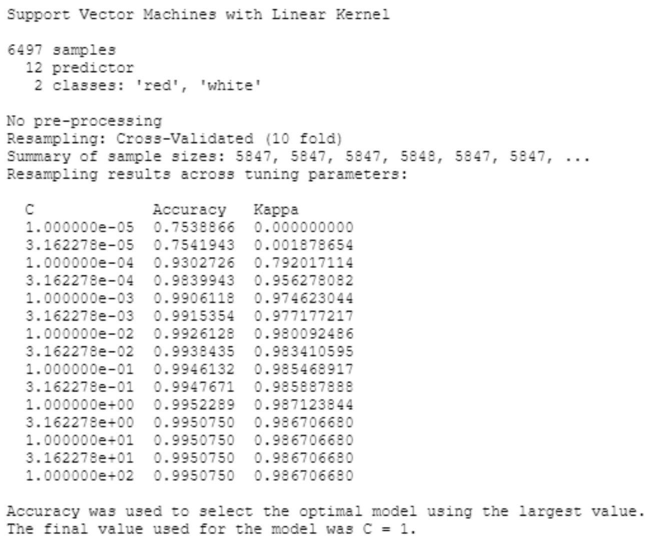
```
K-means clustering with 7 clusters of sizes 856, 910, 354, 717, 1382, 1131, 1147

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar  chlorides free.sulfur.dioxide
1      6.939486        0.2865596   0.3516121       8.986974 0.05143224           48.915304
2      8.448571        0.5121374   0.2714945       2.450000 0.08265165            9.410989
3      7.030791        0.3069633   0.3606215      10.179802 0.05212147           55.213277
4      7.723570        0.4324407   0.2865830       2.613598 0.07373640           19.581590
5      6.838459        0.2871852   0.3247685       5.366353 0.04533792           30.193198
6      6.926437        0.3148983   0.3086207       3.798143 0.04803006           24.540672
7      6.920837        0.2822188   0.3409765       7.195074 0.04927114           39.075414
  total.sulfur.dioxide   density       pH sulphates   alcohol  quality
1            182.81600 0.9960240 3.188166 0.5109229  9.790479 5.586449
2             23.88022 0.9964969 3.303615 0.6385824 10.592271 5.718681
3            224.39972 0.9968792 3.173107 0.5207345  9.545763 5.525424
4             59.94421 0.9949663 3.268243 0.6056206 10.662366 5.647141
5            122.64146 0.9934417 3.195166 0.4963748 10.770466 6.013025
6             95.00531 0.9928397 3.209054 0.5009019 11.033687 5.979664
7            152.05929 0.9947787 3.193958 0.4900697 10.250753 5.874455

Clustering vector:
    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19
    2    4    4    4    2    2    4    2    2    6    4    6    4    2    7    7    6    4    2
   20   21   22   23   24   25   26   27   28   29   30   31   32   33   34   35   36   37   38
    4    4    4    2    4    2    2    2    2    2    2    6    2    5    6    4    2    2    2
   39   40   41   42   43   44   45   46   47   48   49   50   51   52   53   54   55   56   57
    2    6    6    4    2    2    2    4    5    2    2    6    2    2    2    5    4    4    2
   58   59   60   61   62   63   64   65   66   67   68   69   70   71   72   73   74   75   76
    5    4    4    4    5    2    4    2    2    2    2    4    2    2    6    6    4    6    2
   77   78   79   80   81   82   83   84   85   86   87   88   89   90   91   92   93   94   95
    2    2    6    5    2    4    4    4    4    2    5    2    5    2    7    5    5    2    6
   96   97   98   99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114
    6    2    2    2    2    4    2    2    4    2    4    4    4    6    7    4    6    6    4
  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133
    4    4    2    2    4    6    6    4    2    2    6    6    2    2    2    2    5    6    6
  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152
    2    2    4    4    4    6    6    4    4    4    2    4    7    6    6    2    4    2    4
  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171
    6    6    5    5    5    5    2    6    2    2    2    5    5    6    6    4    4    4    2
  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190
    2    2    4    2    4    2    2    2    2    2    5    4    4    4    6    4    2    5    7
  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209
    5    4    5    2    2    5    4    2    6    2    2    7    4    2    2    2    2    5    6
  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228
    2    2    4    2    4    2    5    4    2    4    7    4    4    2    2    4    4    6    2
  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247
    4    2    6    4    4    2    2    2    2    2    2    2    4    2    5    2    2    2    4
  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266
    4    2    2    2    2    2    6    2    5    2    4    4    2    4    4    2    4    2    2
  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285
    4    2    2    2    4    2    4    2    6    4    2    2    2    4    2    2    4    4    4
  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304
    4    4    4    4    6    4    2    4    2    4    2    4    2    2    2    4    2    2    2
  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323
    6    2    4    2    2    2    2    6    6    5    4    4    6    4    4    4    4    6    2
  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342
    4    4    4    2    2    2    2    2    2    5    4    2    2    2    4    6    2    2    2
  343  344  345  346  347  348  349  350  351  352  353  354  355  356  357  358  359  360  361
    2    2    2    4    2    2    4    2    2    2    2    4    7    2    4    2    2    2    6
  362  363  364  365  366  367  368  369  370  371  372  373  374  375  376  377  378  379  380
    4    2    2    2    2    2    2    4    2    4    2    4    4    4    2    4    2    2    4
  381  382  383  384  385  386  387  388  389  390  391  392  393  394  395  396  397  398  399
    2    4    2    2    4    2    4    4    4    4    6    4    2    6    2    2    5    4    4
  400  401  402  403  404  405  406  407  408  409  410  411  412  413  414  415  416  417  418
    2    5    2    4    2    2    2    2    2    2    4    6    6    4    2    5    5    2    5
  419  420  421  422  423  424  425  426  427  428  429  430  431  432  433  434  435  436  437
    2    4    4    4    4    2    4    4    4    2    2    2    2    4    2    2    4    2    4
  438  439  440  441  442  443  444  445  446  447  448  449  450  451  452  453  454  455  456
    2    4    4    2    2    2    2    2    2    4    2    2    2    2    2    2    2    4    2
  457  458  459  460  461  462  463  464  465  466  467  468  469  470  471  472  473  474  475
    2    4    2    4    2    2    2    5    2    2    4    2    2    4    4    4    4    2    2
  476  477  478  479  480  481  482  483  484  485  486  487  488  489  490  491  492  493  494
    2    2    2    2    2    2    2    2    2    2    2    2    2    4    2    4    2    2    6
  495  496  497  498  499  500  501  502  503  504  505  506  507  508  509  510  511  512  513
    6    2    2    5    2    6    2    6    6    2    2    2    2    2    4    4    4    4    4
  514  515  516  517  518  519  520  521  522  523  524  525  526  527  528  529  530  531  532
    2    2    7    2    2    2    6    2    4    5    7    5    4    6    6    4    2    2    2
  533  534  535  536  537  538  539  540  541  542  543  544  545  546  547  548  549  550  551
    2    2    2    2    2    2    2    2    4    2    2    4    2    6    2    2    4    2    2
  552  553  554  555  556  557  558  559  560  561  562  563  564  565  566  567  568  569  570
    4    4    6    2    2    2    2    2    4    2    5    5    4    4    2    2    2    2    4
  571  572  573  574  575  576  577  578  579  580  581  582  583  584  585  586  587  588  589
    2    4    2    4    6    4    2    5    6    2    2    2    2    2    6    2    4    6    4
  590  591  592  593  594  595  596  597  598  599  600  601  602  603  604  605  606  607  608
    2    4    7    4    2    2    5    2    2    2    2    2    2    2    2    4    2    2    4
  609  610  611  612  613  614  615  616  617  618  619  620  621  622  623  624  625  626  627
    4    2    4    2    2    2    6    4    4    2    2    2    5    6    2    4    4    4    2
  628  629  630  631  632  633  634  635  636  637  638  639  640  641  642  643  644  645  646
    2    2    6    2    2    4    6    6    2    7    7    2    4    2    4    2    4    2    2
  647  648  649  650  651  652  653  654  655  656  657  658  659  660  661  662  663  664  665
    2    2    2    7    2    7    4    2    2    4    2    4    2    2    2    4    2    2    4
  666  667  668  669  670  671  672  673  674  675  676  677  678  679  680  681  682  683  684
    4    2    2    2    2    4    2    7    2    2    2    2    2    6    4    2    2    4    2
  685  686  687  688  689  690  691  692  693  694  695  696  697  698  699  700  701  702  703
    7    2    2    2    2    2    2    6    4    5    5    4    2    2    4    2    6    2    2
  704  705  706  707  708  709  710  711  712  713  714  715  716  717  718  719  720  721  722
    6    2    4    2    2    2    2    6    5    2    4    4    2    4    4    2    4    2    6
  723  724  725  726  727  728  729  730  731  732  733  734  735  736  737  738  739  740  741
    4    5    2    2    4    2    2    4    2    2    2    4    2    2    2    4    6    2    2
  742  743  744  745  746  747  748  749  750  751  752  753  754  755  756  757  758  759  760
    5    2    6    6    2    4    4    2    2    2    2    4    2    2    4    2    2    2    6
  761  762  763  764  765  766  767  768  769  770  771  772  773  774  775  776  777  778  779
    6    2    2    2    4    4    4    6    6    2    6    7    7    2    2    2    4    2    2
  780  781  782  783  784  785  786  787  788  789  790  791  792  793  794  795  796  797  798
    6    4    2    6    2    6    4    4    4    4    5    6    5    6    2    2    4    4    4
  799  800  801  802  803  804  805  806  807  808  809  810  811  812  813  814  815  816  817
    2    2    6    2    6    4    2    2    2    2    2    2    2    2    2    2    4    2    2
  818  819  820  821  822  823  824  825  826  827  828  829  830  831  832  833  834  835  836
    2    4    4    2    4    2    2    2    2    2    2    4    2    2    2    4    4    2    4
  837  838  839  840  841  842  843  844  845  846  847  848  849  850  851  852  853  854  855
    6    6    2    2    2    2    4    6    4    2    2    2    2    2    2    2    5    4    4
  856  857  858  859  860  861  862  863  864  865  866  867  868  869  870  871  872  873  874
    2    4    4    2    2    6    6    2    6    6    6    2    2    2    4    4    4    4    2
  875  876  877  878  879  880  881  882  883  884  885  886  887  888  889  890  891  892  893
    2    2    2    4    4    6    2    2    2    6    4    2    2    4    4    6    4    6    2
  894  895  896  897  898  899  900  901  902  903  904  905  906  907  908  909  910  911  912
    6    6    4    4    4    4    2    2    2    2    2    2    5    4    2    4    2    2    2
  913  914  915  916  917  918  919  920  921  922  923  924  925  926  927  928  929  930  931
    2    2    2    2    4    4    6    4    2    6    4    4    2    6    4    4    2    2    2
  932  933  934  935  936  937  938  939  940  941  942  943  944  945  946  947  948  949  950
    2    4    2    2    4    4    2    4    2    2    2    4    4    2    2    2    2    2    2
  951  952  953  954  955  956  957  958  959  960  961  962  963  964  965  966  967  968  969
    2    2    2    2    2    2    2    4    2    2    2    2    2    2    2    4    2    5    4
  970  971  972  973  974  975  976  977  978  979  980  981  982  983  984  985  986  987  988
    2    2    2    2    2    2    4    4    5    2    2    2    2    4    2    2    2    2    4
  989  990  991  992  993  994  995  996  997  998  999 1000 
    2    2    2    4    4    4    6    4    2    2    2    2 
 [ reached getOption("max.print") -- omitted 5497 entries ]
```



```
Within cluster sum of squares by cluster:
[1] 330867.7 108090.8 367269.3 147020.0 301721.7 206162.7 340030.8
 (between_SS / total_SS =  92.2 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"   
[7] "size"         "iter"         "ifault"      
```

SVM:

```
# I decided to use the Grid Search here to try different values of C
grid <- expand.grid(C = 10^seq(-5,2,0.5))
#  Fit the Model
svm_grid <- train(type ~., data = wines, method = "svmLinear",
                  trControl = train_control, tuneGrid = grid)
# View grid search result
svm_grid
```



```
Support Vector Machines with Linear Kernel 

6497 samples
  12 predictor
   2 classes: 'red', 'white' 

No pre-processing
Resampling: Cross-Validated (10 fold) 
Summary of sample sizes: 5847, 5847, 5847, 5848, 5847, 5847, ... 
Resampling results across tuning parameters:

  C             Accuracy   Kappa      
  1.000000e-05  0.7538866  0.000000000
  3.162278e-05  0.7541943  0.001878654
  1.000000e-04  0.9302726  0.792017114
  3.162278e-04  0.9839943  0.956278082
  1.000000e-03  0.9906118  0.974623044
  3.162278e-03  0.9915354  0.977177217
  1.000000e-02  0.9926128  0.980092486
  3.162278e-02  0.9938435  0.983410595
  1.000000e-01  0.9946132  0.985468917
  3.162278e-01  0.9947671  0.985887888
  1.000000e+00  0.9952289  0.987123844
  3.162278e+00  0.9950750  0.986706680
  1.000000e+01  0.9950750  0.986706680
  3.162278e+01  0.9950750  0.986706680
  1.000000e+02  0.9950750  0.986706680

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was C = 1.
```

Decision Trees:

```r
# Evaluation Method
train_control = trainControl(method = "cv", number = 10)
# Fit the Model
tree1 <- train(type ~., data = wines, method = "rpart", trControl = train_control)
# Evaluate the Fit
tree1
```

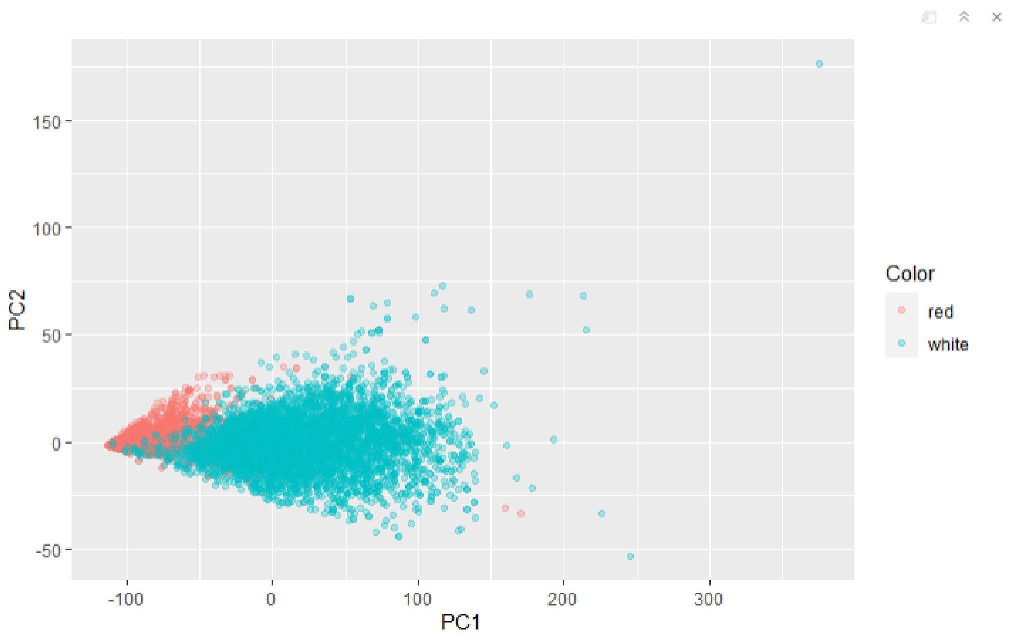
```
CART

6497 samples
  12 predictor
   2 classes: 'red', 'white'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 5847, 5847, 5847, 5847, 5848, 5847, ...
Resampling results across tuning parameters:

  cp          Accuracy   Kappa
  0.06253909  0.9448994  0.8427741
  0.06754221  0.9344300  0.8125794
  0.70043777  0.8370148  0.3857512

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.06253909.
```

Accuracy seems to be comparable in SVM and KNN, although not decision trees. I think the dataset may be too complex/large for a decision tree to handle. However, SVM and KNN seem to perform better. Although, due to the non-linear nature of this dataset, probably a KNN solution may be the best.

e. Use the same already computed PCA again to show a scatter plot of the data and to visualize the labels for kNN, decision tree and SVM. Note that you do not need to recreate the PCA projection, you have already done this in 1b. Here, you just make a new visualization for each classifier using its labels for color (same points but change the color). Map the color results to the classifier, that is use the "predict" function to predict the class of your data, add it to your data frame and use it as a color. This is done for KNN in the tutorial, it should be similar for the others. Consider and explain the differences in how these classifiers performed.

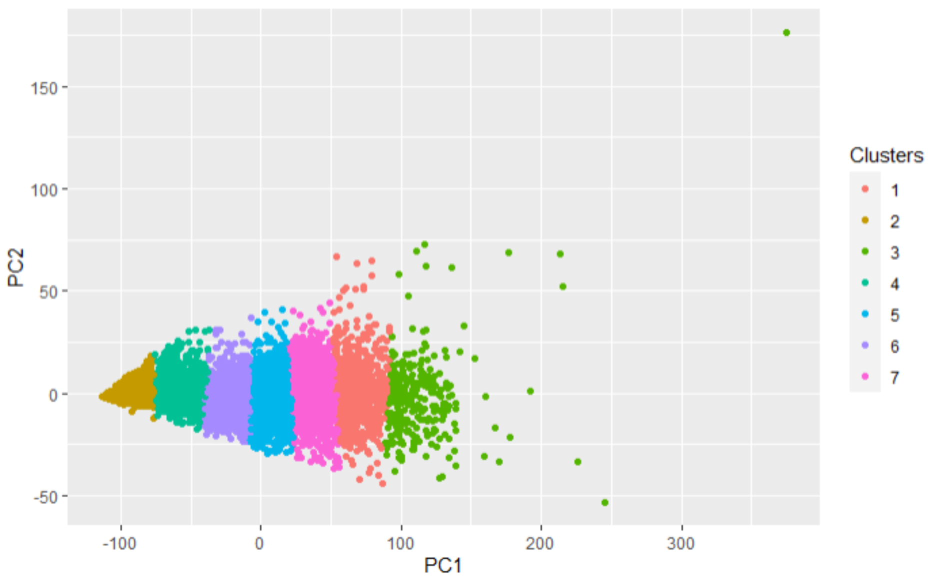
<u>**Using PCA:**</u>

```r
# Plot and color the labels based on Wine type red (or) white
ggplot(data = rotated_data, aes(x = PC1, y = PC2, col = Color)) + geom_point(alpha = 0.3)
```



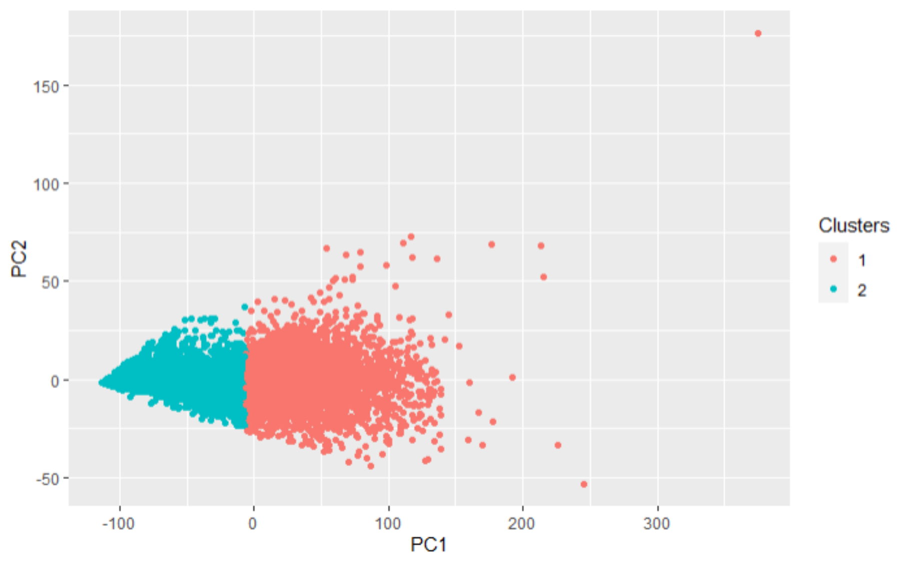
Using KNN:

```
# Assign clusters as a new column
rotated_data$Clusters = as.factor(fit$cluster)
# Plot and color by labels
ggplot(data = rotated_data, aes(x = PC1, y = PC2 , col = Clusters)) + geom_point()
```



PCA seemed to depict the clusters into two uneven clusters, although we are aware that there is far more white wine samples than there is red wine in the dataset. The KNN clustering method provided an interesting distribution, although I prefer PCA as it seems to make more sense with the predicting of the red (or) white wine type. I think trying a different K value, perhaps smaller, may provide us with the better results. For the sake of testing it, I will try k = 2 below, which seems to cluster the data more realistically:

```
fit2 <- kmeans(dummies, centers = 2, nstart = 25)
# Assign clusters as a new column
rotated_data$Clusters = as.factor(fit2$cluster)
# Plot and color by the labels
ggplot(data = rotated_data, aes(x = PC1, y = PC2, col = Clusters)) + geom_point()
```



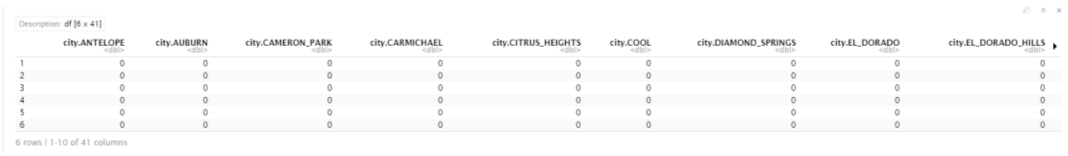
PROBLEM – 2

In this question we will use the Sacramento data, which covers available housing in the region of that city. The variables include numerical information about the size of the housing and its price, as well as categorical information like zip code (there are a large but limited number in the area), and the type of unit (condo vs house (coded as residential))

~~~
```{r}
data("Sacramento")
# Remove the Zipcode, and lon for simplicity
cpsacramento <- Sacramento %>% select(-c("latitude", "longitude", "zip"))
```
~~~

a. **Load the data from the tidyverse library with the data ("Sacramento") command and you should have a variable Sacramento. Because we have categoricals, convert them to dummy variables.**

```
# Type is the largest variable to predict
dummy <- dummyVars(type ~., data = cpsacramento)
dummies <- as.data.frame(predict(dummy, newdata = cpsacramento))
head(dummies)
```



Description: df [6 x 41]

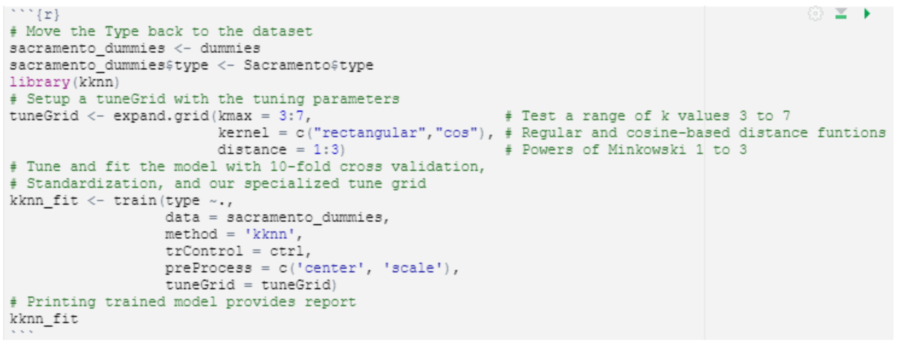
	city.ANTELOPE	city.AUBURN	city.CAMERON_PARK	city.CARMICHAEL	city.CITRUS_HEIGHTS	city.COOL	city.DIAMOND_SPRINGS	city.EL_DORADO	city.EL_DORADO_HILLS
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0

6 rows | 1-10 of 41 columns

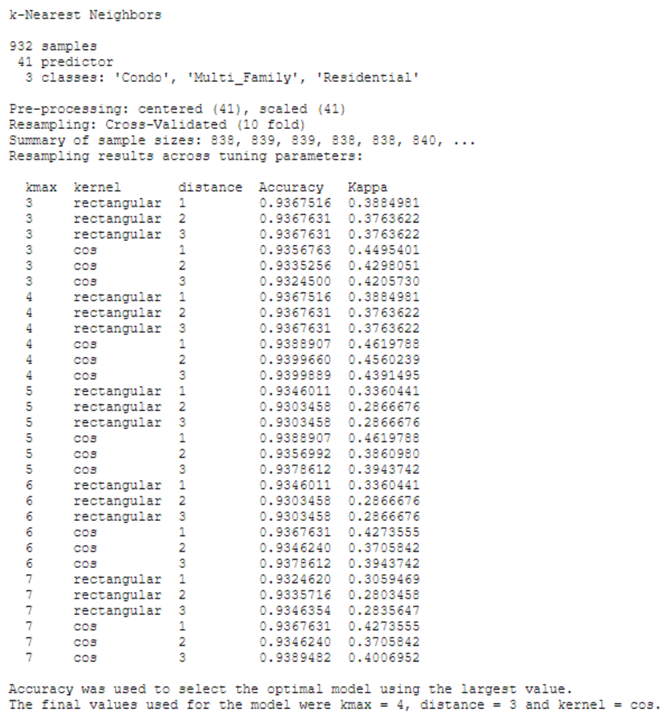
b. With kNN, because of the high dimensionality, which might be a good choice for the distance function?

It's hard to tell without actually trying the different metrics, although using Minkowski is a fairly common in high dimensional data.

c. Use kNN to classify this data with type as the label. Tune the choice of k plus the type of distance function. Report your results – what values for these parameters were tried, which were chosen, and how did they perform with accuracy?



~~~
```{r}
# Move the Type back to the dataset
sacramento_dummies <- dummies
sacramento_dummies$type <- Sacramento$type
library(kknn)
# Setup a tuneGrid with the tuning parameters
tuneGrid <- expand.grid(kmax = 3:7,                        # Test a range of k values 3 to 7
                        kernel = c("rectangular","cos"), # Regular and cosine-based distance funtions
                        distance = 1:3)                    # Powers of Minkowski 1 to 3
# Tune and fit the model with 10-fold cross validation,
# Standardization, and our specialized tune grid
kknn_fit <- train(type ~.,
                  data = sacramento_dummies,
                  method = 'kknn',
                  trControl = ctrl,
                  preProcess = c('center', 'scale'),
                  tuneGrid = tuneGrid)
# Printing trained model provides report
kknn_fit
```
~~~



```
k-Nearest Neighbors

932 samples
 41 predictor
  3 classes: 'Condo', 'Multi_Family', 'Residential'

Pre-processing: centered (41), scaled (41)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 838, 839, 839, 838, 838, 840, ...
Resampling results across tuning parameters:

  kmax  kernel       distance  Accuracy   Kappa
  3     rectangular  1         0.9367516  0.3884981
  3     rectangular  2         0.9367631  0.3763622
  3     rectangular  3         0.9367631  0.3763622
  3     cos          1         0.9356763  0.4495401
  3     cos          2         0.9335256  0.4298051
  3     cos          3         0.9324500  0.4205730
  4     rectangular  1         0.9367516  0.3884981
  4     rectangular  2         0.9367631  0.3763622
  4     rectangular  3         0.9367631  0.3763622
  4     cos          1         0.9388907  0.4619788
  4     cos          2         0.9399660  0.4560239
  4     cos          3         0.9399889  0.4391495
  5     rectangular  1         0.9346011  0.3360441
  5     rectangular  2         0.9303458  0.2866676
  5     rectangular  3         0.9303458  0.2866676
  5     cos          1         0.9388907  0.4619788
  5     cos          2         0.9356992  0.3860980
  5     cos          3         0.9378612  0.3943742
  6     rectangular  1         0.9346011  0.3360441
  6     rectangular  2         0.9303458  0.2866676
  6     rectangular  3         0.9303458  0.2866676
  6     cos          1         0.9367631  0.4273555
  6     cos          2         0.9346240  0.3705842
  6     cos          3         0.9378612  0.3943742
  7     rectangular  1         0.9324620  0.3059469
  7     rectangular  2         0.9335716  0.2803458
  7     rectangular  3         0.9346354  0.2835647
  7     cos          1         0.9367631  0.4273555
  7     cos          2         0.9346240  0.3705842
  7     cos          3         0.9389482  0.4006952

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were kmax = 4, distance = 3 and kernel = cos.
```

Here we tried different Minkowski distances from 1 to 3, and tested regular and cosine distance functions. K-max of 6, with a distance of 2 (Euclidean), and a cosine based distance function were the results. The cosine's distance function kappa value generally performed better, although accuracy was comparable on both.

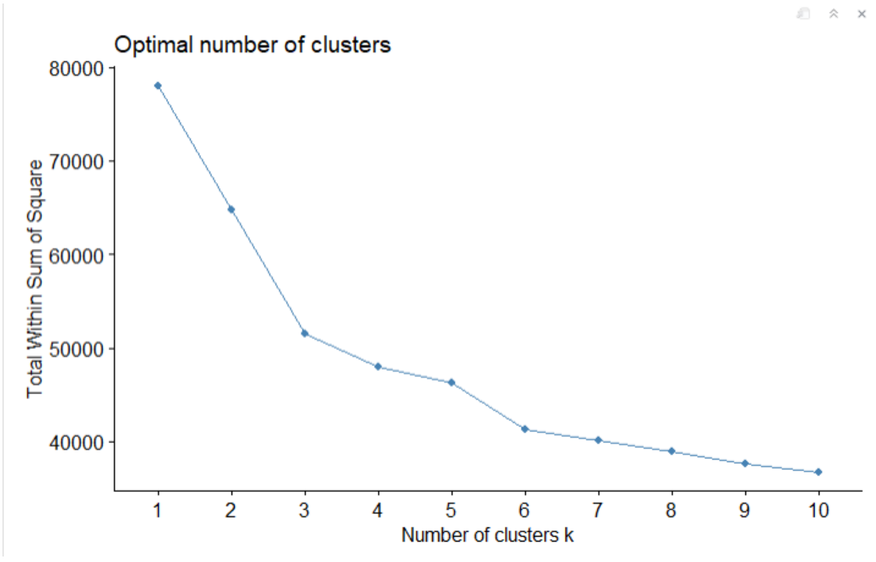
PROBLEM – 3

In this problem we will continue with the wine quality data from Problem 1, but this time we will use clustering. Do not forget to remove the type variable before clustering because that would be cheating by using the label to perform clustering.

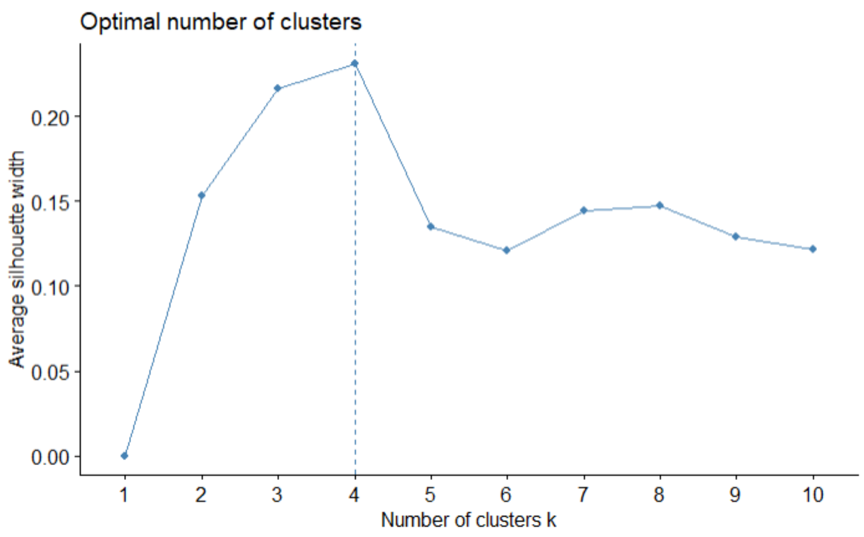
```r
# Copy wines dataset and remove type
cpwines <- wines
cpwines <- cpwines %>% select(-c("type"))
```

a. Use k-means to cluster the data. Show your usage of silhouette and the elbow method to pick the best number of clusters. Make sure it is using multiple restarts.

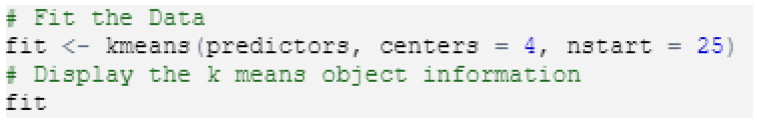
```r
df <- cpwines
# Set seed
set.seed(123)
# Center scale allows us to standardize the data
preproc <- preProcess(df, method = c("center", "scale"))
# We have to call predict to fit our data based on preprocessing
predictors <- predict(preproc, df)
# Find the knee
fviz_nbclust(predictors, kmeans, method = "wss")
```



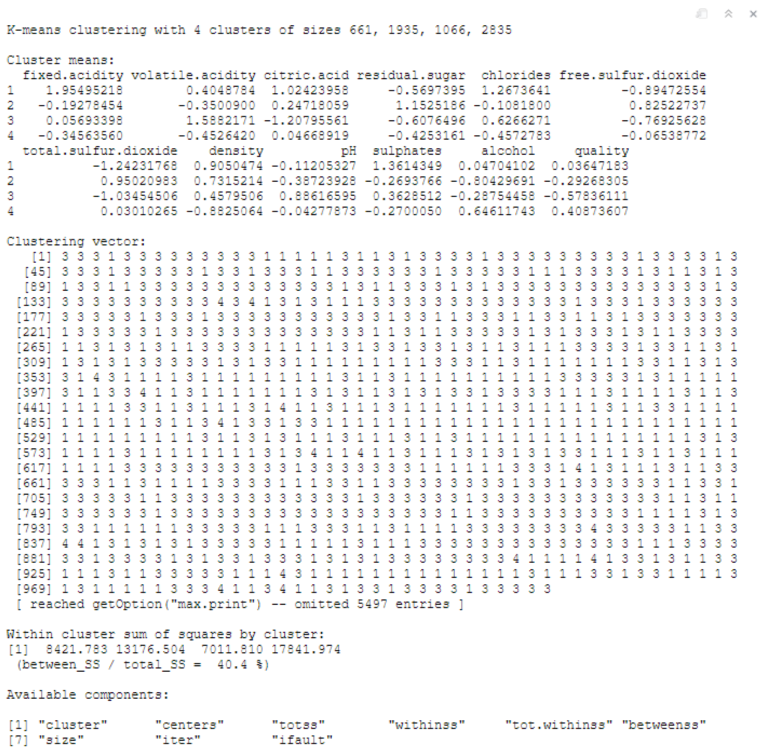
```r
fviz_nbclust(predictors, kmeans, method = "silhouette")
```



The silhouette method suggests a k = 4, and the elbow method also suggests a k = 4 may be reasonable.



```
# Fit the Data
fit <- kmeans(predictors, centers = 4, nstart = 25)
# Display the k means object information
fit
```



```
K-means clustering with 4 clusters of sizes 661, 1935, 1066, 2835

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar  chlorides free.sulfur.dioxide
1    1.95495218        0.4048784  1.02423958     -0.5697395  1.2673641         -0.89472554
2   -0.19278454       -0.3500900  0.24718059      1.1525186 -0.1081800          0.82522737
3    0.05693398        1.5882171 -1.20795561     -0.6076496  0.6266271         -0.76925628
4   -0.34563560       -0.4526420  0.04668919     -0.4253161 -0.4572783         -0.06538772
  total.sulfur.dioxide    density          pH  sulphates     alcohol     quality
1          -1.24231768  0.9050474 -0.11205327  1.3614349  0.04704102  0.03647183
2           0.95020983  0.7315214 -0.38723928 -0.2693766 -0.80429691 -0.29268305
3          -1.03454506  0.4579506  0.88616595  0.3628512 -0.28754458 -0.57836111
4           0.03010265 -0.8825064 -0.04277873 -0.2700050  0.64611743  0.40873607

Clustering vector:
   [1] 3 3 3 1 3 3 3 3 3 3 3 3 3 1 1 1 1 1 3 1 1 3 1 3 3 3 3 1 3 3 3 3 3 3 3 3 3 1 3 3 3 3 1 3
  [45] 3 3 3 1 3 3 3 3 3 1 3 3 1 3 3 3 1 1 3 3 3 3 3 3 1 3 3 3 3 3 1 1 1 3 3 3 3 1 3 1 1 3 1 3
  [89] 1 3 3 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 1 1 3 3 3 1 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3
 [133] 3 3 3 3 3 3 3 3 3 3 4 3 4 1 3 1 3 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 3 3 1 3 3 3 3 3 3
 [177] 3 3 3 3 3 1 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 1 3 3 1 1 3 3 3 1 1 3 3 1 1 3 1 3 3 3 3 3 3 3
 [221] 1 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 3 1 1 3 3 3 3 3 1 3 1 3 3 3 1 3 1 1 3 3 3 3
 [265] 1 1 3 1 3 1 3 1 1 3 3 3 3 1 1 1 1 1 3 1 3 3 1 3 3 1 3 1 1 3 1 1 1 3 3 3 3 1 3 3 1 1 3 1
 [309] 1 3 1 3 1 3 3 3 3 3 1 3 1 3 3 1 1 1 1 1 1 1 1 1 3 3 3 1 1 3 1 1 1 1 1 1 1 3 3 1 1 3 1 3
 [353] 3 1 4 3 1 1 1 1 3 1 1 1 1 1 1 1 1 1 3 1 1 3 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 1 3 1 1 1 1 1
 [397] 3 1 1 3 3 4 1 1 3 1 1 1 1 1 1 1 3 1 3 1 1 3 1 3 1 3 3 1 3 3 3 3 1 1 1 3 1 1 1 1 3 1 1 3
 [441] 1 1 1 1 3 3 1 1 3 1 1 1 3 1 4 1 1 3 1 1 1 3 1 1 1 1 1 1 1 3 1 1 1 1 1 3 1 1 3 3 1 1 1 1
 [485] 1 1 1 1 1 1 3 1 1 3 4 1 3 3 1 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [529] 1 1 1 1 1 1 1 1 1 3 1 1 3 1 3 1 1 1 3 1 1 1 3 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 3
 [573] 1 1 1 1 3 1 1 1 1 1 1 1 1 3 1 3 4 1 1 4 1 1 3 1 1 1 3 1 3 1 3 1 3 3 1 1 1 3 1 1 3 1 1 1
 [617] 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3 3 1 1 1 1 1 1 3 3 3 1 4 1 3 1 1 1 3 1 1 3 3
 [661] 3 3 3 1 1 3 1 1 1 1 3 3 3 3 1 1 1 3 3 1 1 3 3 3 3 3 3 3 3 1 3 3 1 3 3 3 3 3 3 1 1 3 3 1
 [705] 3 3 3 3 3 1 1 3 3 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 1 1 3 1 1
 [749] 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 3 3 3 3 3 3 3 3 3 3 1 1 1 1 3 1 3
 [793] 3 3 1 1 1 1 1 1 3 3 3 3 3 1 1 1 3 3 3 1 1 3 1 1 1 1 3 3 3 3 3 3 3 3 4 3 3 3 3 3 1 1 3 3
 [837] 4 4 1 3 1 3 1 3 1 3 3 3 3 3 1 1 1 1 1 3 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 3 3 3 3
 [881] 3 3 1 3 3 3 3 1 3 1 3 3 1 3 3 3 1 3 1 3 1 3 3 3 3 3 3 3 4 1 1 1 1 4 1 3 3 1 3 1 1 3 3
 [925] 1 1 1 3 1 1 3 3 3 3 3 1 1 1 4 3 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 3 3 1 3 3 1 1 1 1 3
 [969] 1 3 1 1 1 1 1 3 3 3 4 1 1 3 4 1 1 3 1 3 3 1 3 3 3 3 1 3 3 3 3 3
 [ reached getOption("max.print") -- omitted 5497 entries ]

Within cluster sum of squares by cluster:
[1]  8421.783 13176.504  7011.810 17841.974
 (between_SS / total_SS =  40.4 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
```

b. Use hierarchical agglomerative clustering (HAC) to cluster the data. Try at least 2 distance functions and at least 2 linkage functions (cluster distance functions), for a total of 4 parameter combinations. For each parameter combination, perform the clustering.

Euclidean and complete linkage:

```
dist_mat <- dist(predictors, method = 'euclidean')
# Determine assembly/agglomeration method and run hclust
hfit1 <- hclust(dist_mat, method = 'complete')
hfit1
```

```
Call:
hclust(d = dist_mat, method = "complete")

Cluster method   : complete
Distance         : euclidean
Number of objects: 6497
```

Euclidean and average linkage:

```
dist_mat <- dist(predictors, method = 'euclidean')
# Determine assembly/agglomeration method and run hclust
hfit2 <- hclust(dist_mat, method = 'average')
hfit2
```

```
Call:
hclust(d = dist_mat, method = "average")

Cluster method   : average
Distance         : euclidean
Number of objects: 6497
```

Manhattan and complete linkage:

```
dist_mat <- dist(predictors, method = 'manhattan')
# Determine assembly/agglomeration method and run hclust (average uses mean)
hfit3 <- hclust(dist_mat, method = 'complete')
hfit3
```

```
Call:
hclust(d = dist_mat, method = "complete")

Cluster method   : complete
Distance         : manhattan
Number of objects: 6497
```

Manhattan and average linkage:

```
dist_mat <- dist(predictors, method = 'manhattan')
# Determine assembly/agglomeration method and run hclust (average uses mean)
hfit4 <- hclust(dist_mat, method = 'average')
hfit4
```

```
Call:
hclust(d = dist_mat, method = "average")

Cluster method   : average
Distance         : manhattan
Number of objects: 6497
```

```
# Build the new model
h1 <- cutree(hfit1, k=4)
h2 <- cutree(hfit2, k=4)
h3 <- cutree(hfit3, k=4)
h4 <- cutree(hfit4, k=4)
```

c. **Compare the k-means and HAC clusterings by creating a crosstabulation between their labels.**

```
#Redefining the fit (I think I missed it previously)
fit <- kmeans(predictors, centers = 4, nstart = 25)
#Create a dataframe for the results
result1 <- data.frame(WineType = wines$type, HAC1 = h1, Kmeans = fit$cluster)
result2 <- data.frame(WineType = wines$type, HAC2 = h2, Kmeans = fit$cluster)
result3 <- data.frame(WineType = wines$type, HAC3 = h3, Kmeans = fit$cluster)
result4 <- data.frame(WineType = wines$type, HAC4 = h4, Kmeans = fit$cluster)
```

```
#Crosstab for HAC
result1 %>% group_by(HAC1) %>% select(HAC1, WineType) %>% table()
```

```
    WineType
HAC1  red white
   1 1597  4896
   2    2     0
   3    0     1
   4    0     1
```

```
result2 %>% group_by(HAC2) %>% select(HAC2, WineType) %>% table()
```

```
    WineType
HAC2  red white
   1 1575  4895
   2   24     1
   3    0     1
   4    0     1
```

```
result3 %>% group_by(HAC3) %>% select(HAC3, WineType) %>% table()
```

```
    WineType
HAC3  red white
   1 1595  4889
   2    4     0
   3    0     8
   4    0     1
```

```
result4 %>% group_by(HAC4) %>% select(HAC4, WineType) %>% table()
```

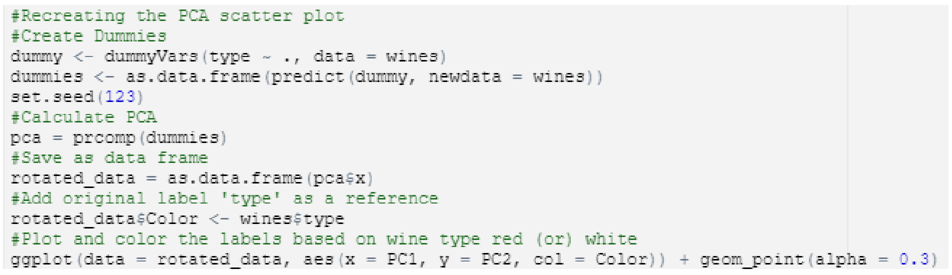
```
    WineType
HAC4  red white
   1 1576  4896
   2   23     0
   3    0     1
   4    0     1
```

```
#Crosstab for K Means
result <- data.frame(Type = wines$type, Kmeans = fit$cluster)
result %>% group_by(Kmeans) %>% select(Kmeans, Type) %>% table()
```

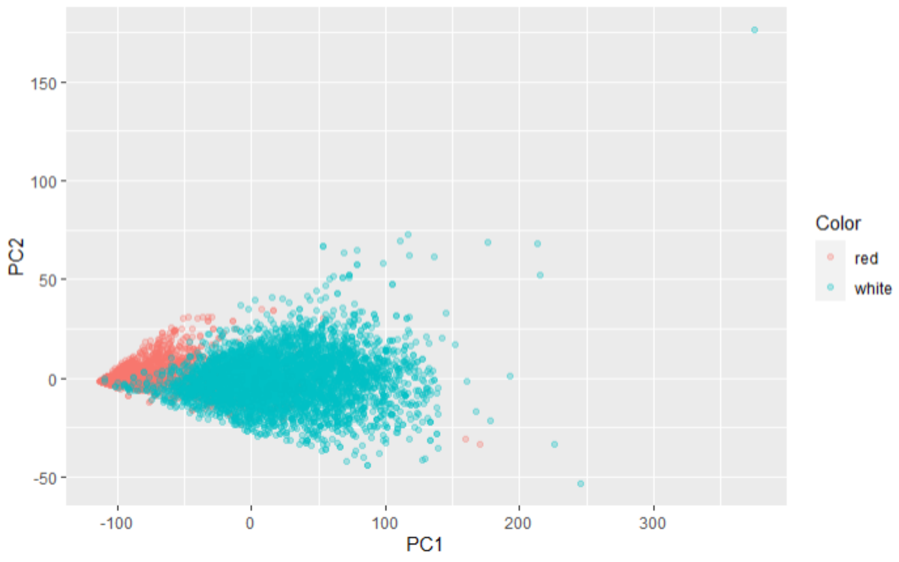
```
      Type
Kmeans  red white
     1    3  1932
     2  609    52
     3  927   139
     4   60  2775
```

d. **For comparison – use PCA to visualize the data in a scatterplot. Create 3 separate plots: use the color of the points to show (1) the type label, (2) the k-means cluster labels and (3) the HAC cluster labels.**

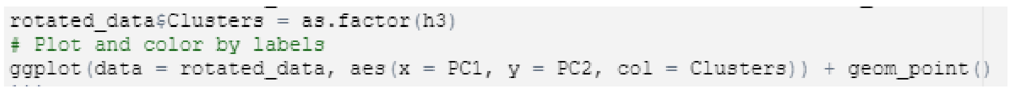
<u>PCA:</u>



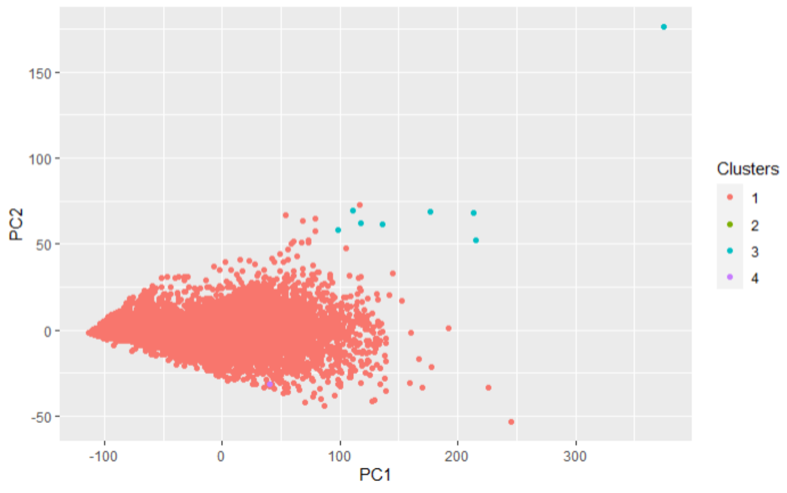
```
#Recreating the PCA scatter plot
#Create Dummies
dummy <- dummyVars(type ~ ., data = wines)
dummies <- as.data.frame(predict(dummy, newdata = wines))
set.seed(123)
#Calculate PCA
pca = prcomp(dummies)
#Save as data frame
rotated_data = as.data.frame(pca$x)
#Add original label 'type' as a reference
rotated_data$Color <- wines$type
#Plot and color the labels based on wine type red (or) white
ggplot(data = rotated_data, aes(x = PC1, y = PC2, col = Color)) + geom_point(alpha = 0.3)
```



<u>H3:</u>

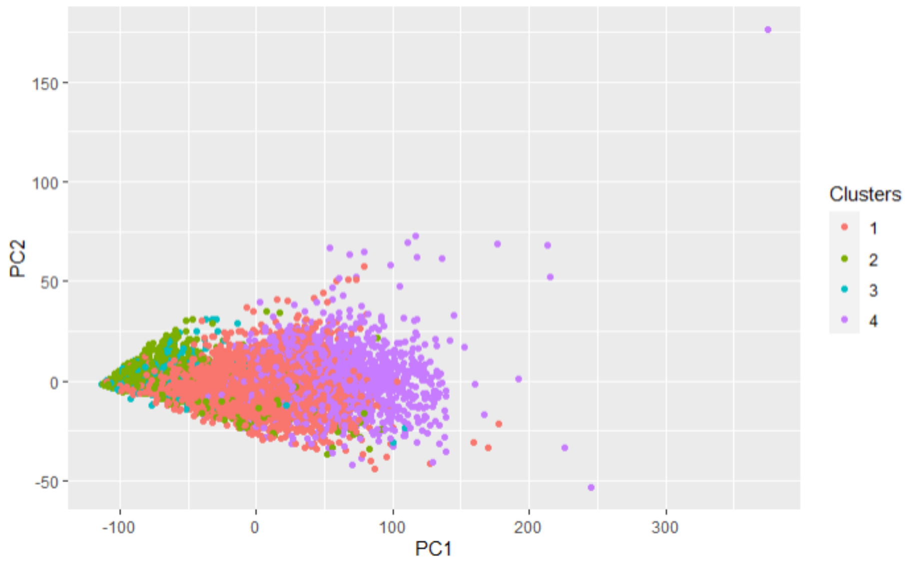


```
rotated_data$Clusters = as.factor(h3)
# Plot and color by labels
ggplot(data = rotated_data, aes(x = PC1, y = PC2, col = Clusters)) + geom_point()
```



K-means:

```
rotated_data$Clusters = as.factor(fit$cluster)
# Plot and color by labels
ggplot(data = rotated_data, aes(x = PC1, y = PC2, col = Clusters)) + geom_point()
```

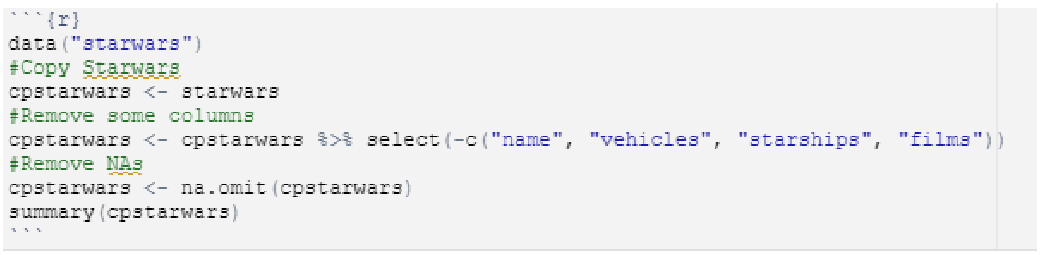


e. Consider the results of C and D and explain the differences between the clustering results in terms of how the algorithms work.

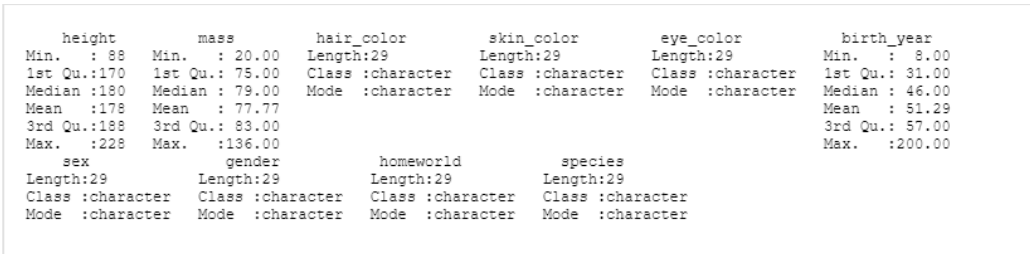
K-means uses a pre-specified K value, while HAC doesn't. We can see that the clustering method seems to be less random in the K-means method. Since HAC is arranged more like a 'tree', it seemed to struggling with clustering the data. K-means divided the data into non-overlapping clusters, which provided a more reasonable look at the dataset. Regardless, I still find PCA to be more sensible visually in this case, even though I don’t think it performed as well as k-means did in this case as it didn’t handle the complexity of the dataset as well.

PROBLEM – 4

Back to the Starwars data from a previous assignment! Remember that the variable that lists the actual names and the variables that are actually lists will be a problem, so remove them (name, films, vehicles, starships). Make sure to double check the types of the variables, i.e., that they are numerical or factors as you expect.



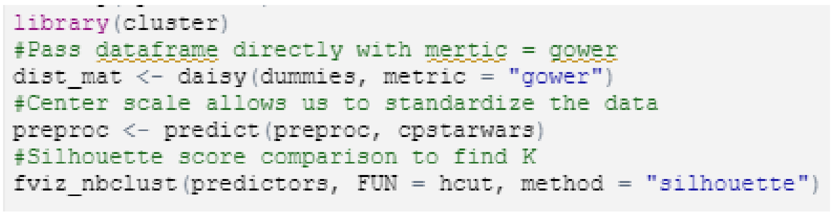
~~~
```{r}
data("starwars")
#Copy Starwars
cpstarwars <- starwars
#Remove some columns
cpstarwars <- cpstarwars %>% select(-c("name", "vehicles", "starships", "films"))
#Remove NAs
cpstarwars <- na.omit(cpstarwars)
summary(cpstarwars)
```
~~~



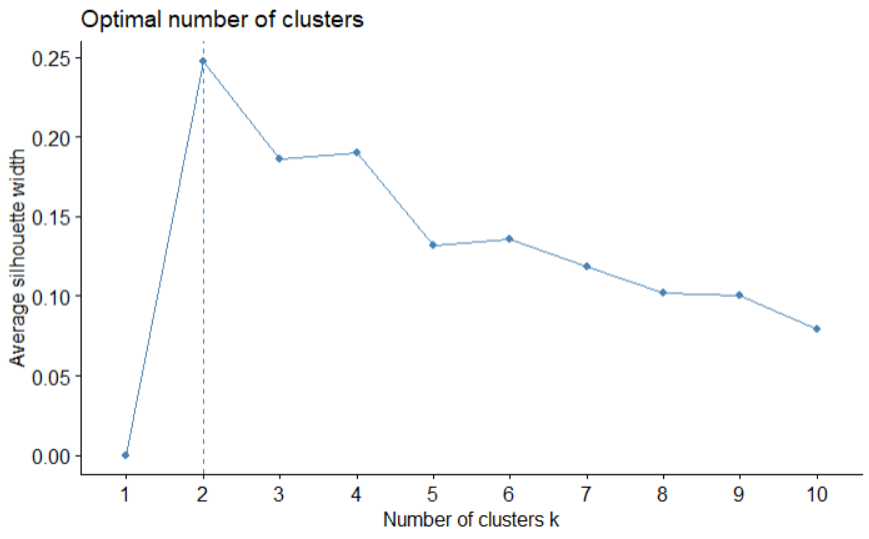
```
    height          mass         hair_color         skin_color         eye_color          birth_year    
 Min.   : 88   Min.   : 20.00   Length:29          Length:29          Length:29          Min.   :  8.00  
 1st Qu.:170   1st Qu.: 75.00   Class :character   Class :character   Class :character   1st Qu.: 31.00  
 Median :180   Median : 79.00   Mode  :character   Mode  :character   Mode  :character   Median : 46.00  
 Mean   :178   Mean   : 77.77                                                            Mean   : 51.29  
 3rd Qu.:188   3rd Qu.: 83.00                                                            3rd Qu.: 57.00  
 Max.   :228   Max.   :136.00                                                            Max.   :200.00  
     sex               gender           homeworld           species         
 Length:29          Length:29          Length:29          Length:29         
 Class :character   Class :character   Class :character   Class :character  
 Mode  :character   Mode  :character   Mode  :character   Mode  :character  
```

a. **Use hierarchical agglomerative clustering to cluster the Starwars data. This time we can leave the categorical variables in place, because we will use the gower metric from daisy in the cluster library to get the distances. Use average linkage. Determine the best number of clusters.**

I had some trouble trying to run this without converting the categoricals to dummies, so I decided to proceed using the dummy variables.



```
library(cluster)
#Pass dataframe directly with mertic = gower
dist_mat <- daisy(dummies, metric = "gower")
#Center scale allows us to standardize the data
preproc <- predict(preproc, cpstarwars)
#Silhouette score comparison to find K
fviz_nbclust(predictors, FUN = hcut, method = "silhouette")
```



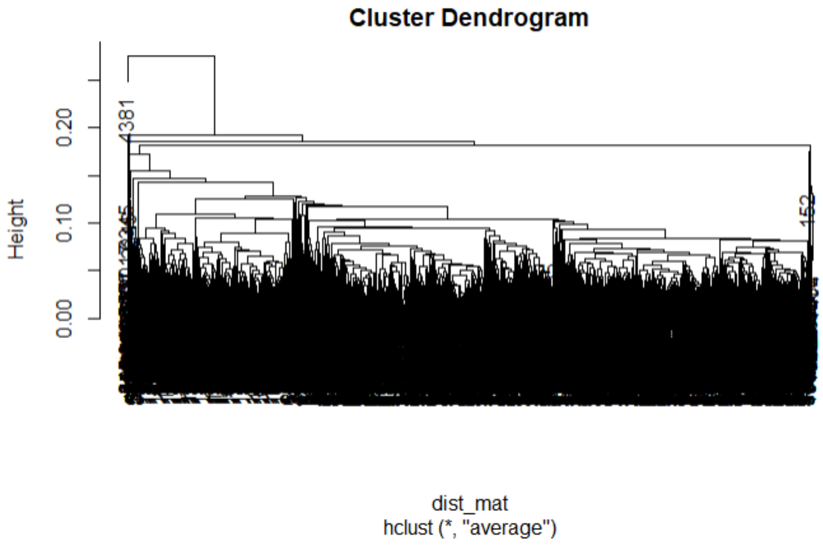
<u>Clusters:</u>

```
#Determine the assembly/agglomeration method and run hclust
hfit <- hclust(dist_mat, method = 'average')
#Build the new model
h2 <- cutree(hfit, k=2)
summary(h2)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
      1       1       1       1       1       2 
```

b. **Produce the dendogram for (a). How might an anomaly show up in a dendogram? Do you see a Starwars character who does not seem to fit in easily? What is the advantage of considering anomalies this way as opposed to looking for unusual values relative to the mean and standard deviations, as we considered earlier in the course? Disadvantages?**

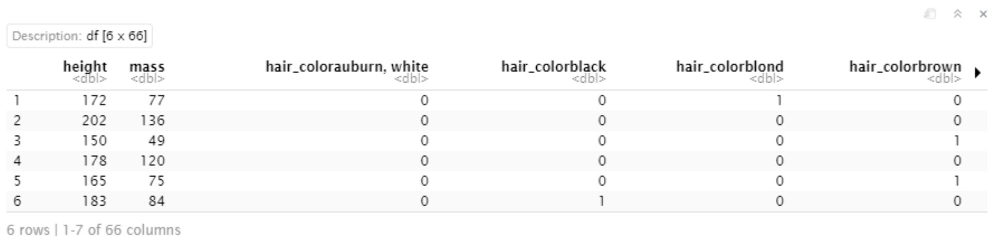
```
hfit <- hclust(dist_mat, method = 'average')
plot(hfit)
```



Own branch with no relationship to other members of the dataset. Based on this dataset, 21, 9, and 18 may have anomalous features. The advantages of this dendogram is the ability to view anomalies directly without reviewing the actual dataset in a tabulation. However, we're only able to view the anomalies here based on height, which may be lacking in regards to other features like eye color (or) mass.

c. Use dummy variables to make this data fully numeric and then use k-means to cluster. Choose the best number of clusters.

```
dummy <- dummyVars(gender ~ ., data = cpstarwars)
dummies <- as.data.frame(predict(dummy, newdata = cpstarwars))
head(dummies)
```

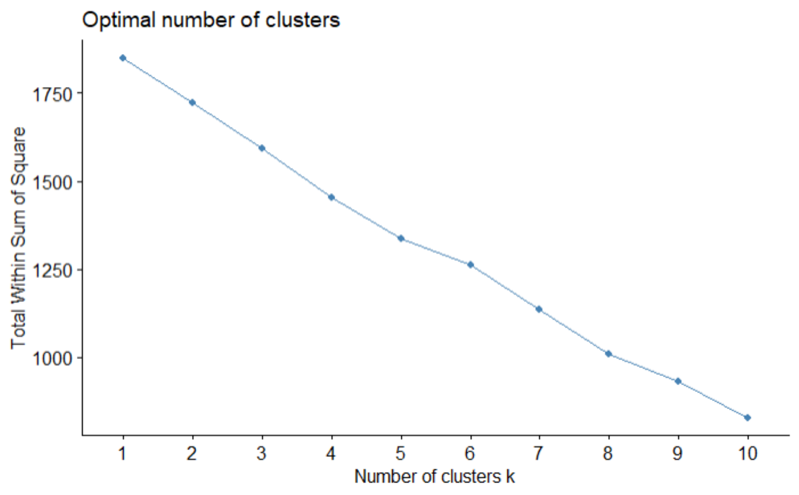


Description: df [6 x 66]

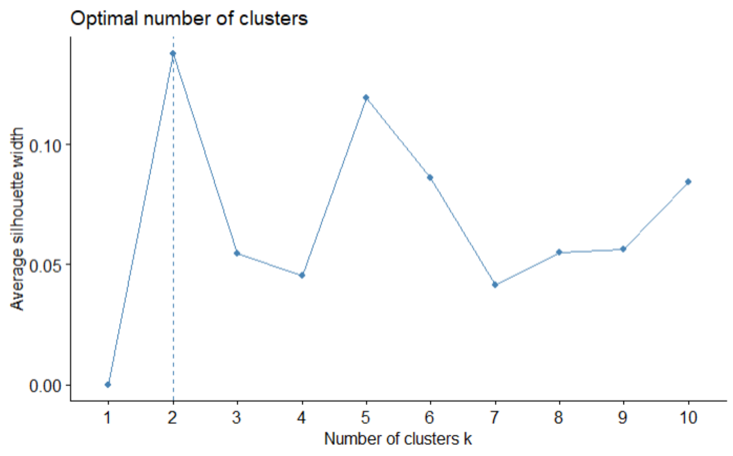
	height <dbl>	mass <dbl>	hair_colorauburn, white <dbl>	hair_colorblack <dbl>	hair_colorblond <dbl>	hair_colorbrown <dbl>
1	172	77	0	0	1	0
2	202	136	0	0	0	0
3	150	49	0	0	0	1
4	178	120	0	0	0	0
5	165	75	0	0	0	1
6	183	84	0	1	0	0

6 rows | 1-7 of 66 columns

```
#Create a predictors file using dummies
predictors <- dummies
#Set seed
set.seed(123)
#Center scale allows us to standardize the data
preproc <- preProcess(predictors, method = c("center", "scale"))
#We have to call predict to fit our data based on preprocessing
predictors <- predict(preproc, predictors)
#Find the knee
fviz_nbclust(predictors, kmeans, method = "wss")
```

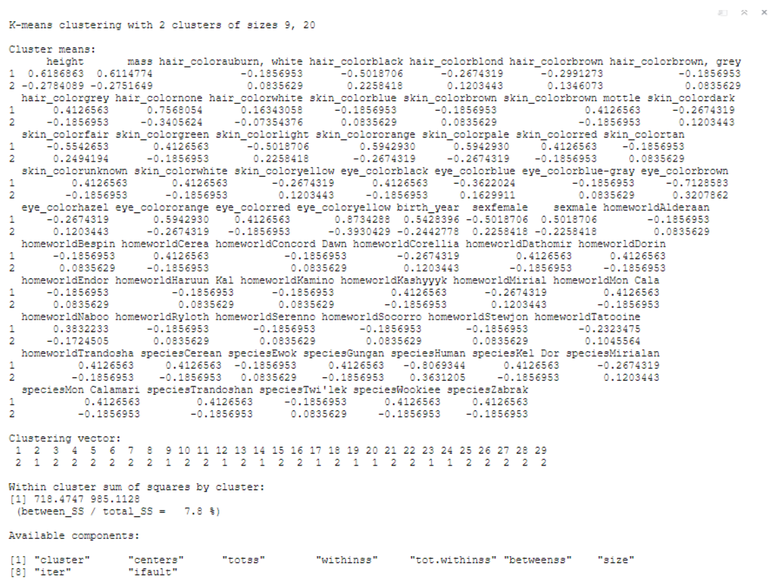


```
fviz_nbclust(predictors, kmeans, method = "silhouette")
```



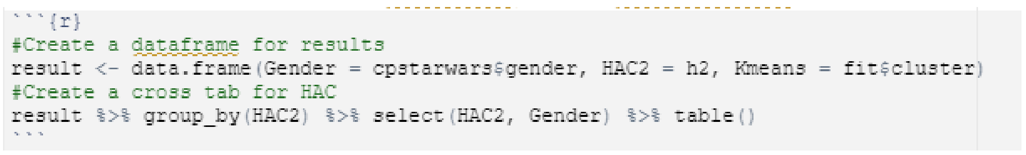
K=2 will be the distance we are using here,

```
#Fit the data
fit <- kmeans(predictors, centers = 2, nstart = 25)
#Display the kmeans object information
fit
```



```
K-means clustering with 2 clusters of sizes 9, 20

Cluster means:
      height       mass hair_colorauburn, white hair_colorblack hair_colorblond hair_colorbrown hair_colorbrown, grey
1  0.6186863  0.6114774              -0.1856953      -0.5018706      -0.2674319      -0.2991273            -0.1856953
2 -0.2784089 -0.2751649               0.0835629       0.2258418       0.1203443       0.1346073             0.0835629
  hair_colorgrey hair_colornone hair_colorwhite skin_colorblue skin_colorbrown skin_colorbrown mottle skin_colordark
1      0.4126563      0.7568054      0.16343058     -0.1856953      -0.1856953             0.4126563     -0.2674319
2     -0.1856953     -0.3405624     -0.07354376      0.0835629       0.0835629            -0.1856953      0.1203443
  skin_colorfair skin_colorgreen skin_colorlight skin_colororange skin_colorpale skin_colorred skin_colortan
1     -0.5542653       0.4126563      -0.5018706        0.5942930      0.5942930     0.4126563    -0.1856953
2      0.2494194      -0.1856953       0.2258418       -0.2674319     -0.2674319    -0.1856953     0.0835629
  skin_colorunknown skin_colorwhite skin_coloryellow eye_colorblack eye_colorblue eye_colorblue-gray eye_colorbrown
1         0.4126563       0.4126563       -0.2674319      0.4126563    -0.3622024         -0.1856953     -0.7128583
2        -0.1856953      -0.1856953        0.1203443     -0.1856953     0.1629911          0.0835629      0.3207862
  eye_colorhazel eye_colororange eye_colorred eye_coloryellow birth_year  sexfemale    sexmale homeworldAlderaan
1     -0.2674319       0.5942930    0.4126563       0.8734288  0.5428396 -0.5018706  0.5018706        -0.1856953
2      0.1203443      -0.2674319   -0.1856953      -0.3930429 -0.2442778  0.2258418 -0.2258418         0.0835629
  homeworldBespin homeworldCerea homeworldConcord Dawn homeworldCorellia homeworldDathomir homeworldDorin
1      -0.1856953      0.4126563            -0.1856953        -0.2674319         0.4126563      0.4126563
2       0.0835629     -0.1856953             0.0835629         0.1203443        -0.1856953     -0.1856953
  homeworldEndor homeworldHaruun Kal homeworldKamino homeworldKashyyyk homeworldMirial homeworldMon Cala
1     -0.1856953          -0.1856953      -0.1856953         0.4126563      -0.2674319         0.4126563
2      0.0835629           0.0835629       0.0835629        -0.1856953       0.1203443        -0.1856953
  homeworldNaboo homeworldRyloth homeworldSerenno homeworldSocorro homeworldStewjon homeworldTatooine
1      0.3832233      -0.1856953       -0.1856953       -0.1856953       -0.1856953        -0.2323475
2     -0.1724505       0.0835629        0.0835629        0.0835629        0.0835629         0.1045564
  homeworldTrandosha speciesCerean speciesEwok speciesGungan speciesHuman speciesKel Dor speciesMirialan
1          0.4126563     0.4126563  -0.1856953     0.4126563   -0.8069344      0.4126563      -0.2674319
2         -0.1856953    -0.1856953   0.0835629    -0.1856953    0.3631205     -0.1856953       0.1203443
  speciesMon Calamari speciesTrandoshan speciesTwi'lek speciesWookiee speciesZabrak
1           0.4126563         0.4126563     -0.1856953      0.4126563     0.4126563
2          -0.1856953        -0.1856953      0.0835629     -0.1856953    -0.1856953

Clustering vector:
 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 27 28 29 
 2  1  2  2  2  2  2  2  1  2  2  1  2  1  2  2  1  2  1  1  2  2  1  1  2  2  2  2  2 

Within cluster sum of squares by cluster:
[1] 718.4747 985.1128
 (between_SS / total_SS =   7.8 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"        
[8] "iter"         "ifault"      
```

d. **Compare the HAC and k-means clusterings with a crosstabulation.**



~~~
```{r}
#Create a dataframe for results
result <- data.frame(Gender = cpstarwars$gender, HAC2 = h2, Kmeans = fit$cluster)
#Create a cross tab for HAC
result %>% group_by(HAC2) %>% select(HAC2, Gender) %>% table()
```
~~~