Unsupervised Learning - K-means Cluster Analysis

Video Game Sales

Gorramuth Prasertkull 6280632

March 27, 2020

1.1 Introduction

This analysis is applied on the Video Games Sales dataset retrieved from https://www.kaggle.com/rush4ratio/video-game-sales-with-ratings. Video games has always been a major interest of millions of people around the world and before every purchases many people would go to popular review sites to evaluate whether a video game is worth buying. However, if you were to make and sell video game, are critic scores important to how well your game sells? Can be be successful with low ratings since there are times where critics don't always agree with users? We will be visualizing the clusters of corresponding scores and sales.

1.2 Hypothesis

I believe that video games with similar global sales have rating characteristics, especially how sometimes users and critics disagree with a game's rating, in common so I decide to analyze it using K-means to see if the data in the clusters have coherent relationships and reveal their similarities of the range of their attributes in scores and sales.

1.3 Data preparation

	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales	Critic_Score	Critic_Count	User_Score	User_Count	Developer	R
	0.06	0.07	0.34	0.01	0.48	NA	NA		NA		
	0.40	0.02	0.00	0.07	0.48	56	18	6.1	9	Amaze Entertainment	
	0.35	0.13	0.00	0.01	0.48	NA	NA	tbd	NA	Vicarious Visions	
	0.35	0.13	0.00	0.01	0.48	62	4	7.8	8	Pocketeers	
ient	0.00	0.00	0.45	0.03	0.48	NA	NA		NA		
	0.01	0.45	0.00	0.03	0.48	85	34	5.8	1345	Ubisoft Montreal	
	0.36	0.11	0.00	0.02	0.48	78	44	6.5	11	EA Tiburon	
	0.45	0.03	0.00	0.01	0.48	NA	NA		NA		

Transformation, Cleansing, and Selection:

The data is in acceptable format, however it is quite dirty. There are fields with blank values, NA values, and the numbers are actually characters. The User_Score column have "tbd" string inside which will not do in our analysis. We will begin cleansing this data and extract our three variables of interest for clustering.

	Critic_Score	User_Score	Global_Sales
6409	47	5.7	0.27
6410	59	6.7	0.27
6411	NA	tbd	0.27
6412	45	6.1	0.27
6413	NA		0.27
6414	69	7.3	0.27
6415	NA		0.27
6416	NA		0.27
6417	87	8.2	0.27

Showing 6,413 to 6,423 of 16,719 entries, 3 total columns

	Critic_Score	User_Score	Global_Sales
1	76	79	82.53
2	82	82	35.52
3	80	79	32.77
4	89	84	29.80
5	58	65	28.92
6	87	83	28.32
7	91	85	23.21
8	80	76	22.70
9	61	62	21.81
10	80	73	21.79

Showing 1 to 11 of 7,055 entries, 3 total columns

Script

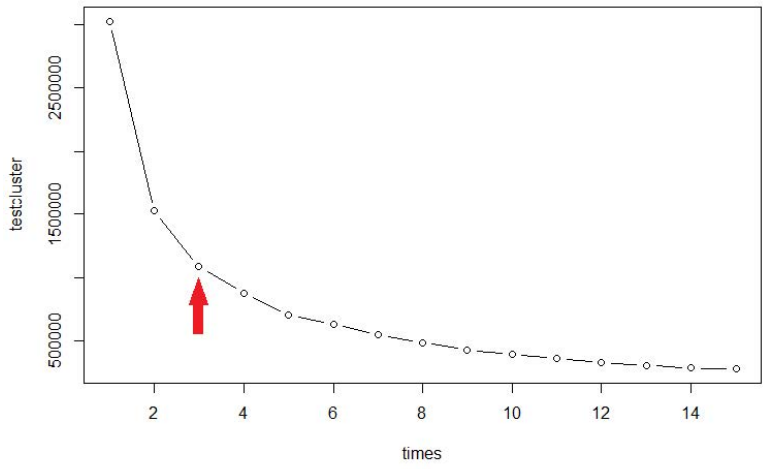
```
extracted<-data%>%select(Critic_Score,User_Score,Global_Sales)
extracted2<-extracted[complete.cases(extracted[ ,1:3]),]
extracted3<-extracted2%>%filter(User_Score!="tbd")
extracted4<-extracted3%>%
mutate(Critic_Score=as.numeric(Critic_Score),User_Score=as.numeric(User_Score),Global_
Sales=as.numeric(Global_Sales))
```

By running the above script, the NA values are omitted and we have selected the two variables of interest which resulted in the data extracted, the characters are now converted to numbers, clean data is created in a form of extracted4 that is now ready for clustering.

1.4 Results

How many clusters should it be?

Firstly, in order to proceed with the K-means algorithm, the optimal number of clusters must be determined before proceeding. The method which shall be used is by graphing the deviation of total squared distance between each clusters. We would want to choose the marginal value of cluster in which the next decrease in the squared distance is negligible.



Script

```
testcluster<-sapply(times,function(k){
  cl<-kmeans(extracted4,k,25)
  cl$tot.withinss
})
plot(times, testcluster, type="b")
```

With the graph, we can now see that the optimal number of clusters is 3 since further decreases are too little to be significant.

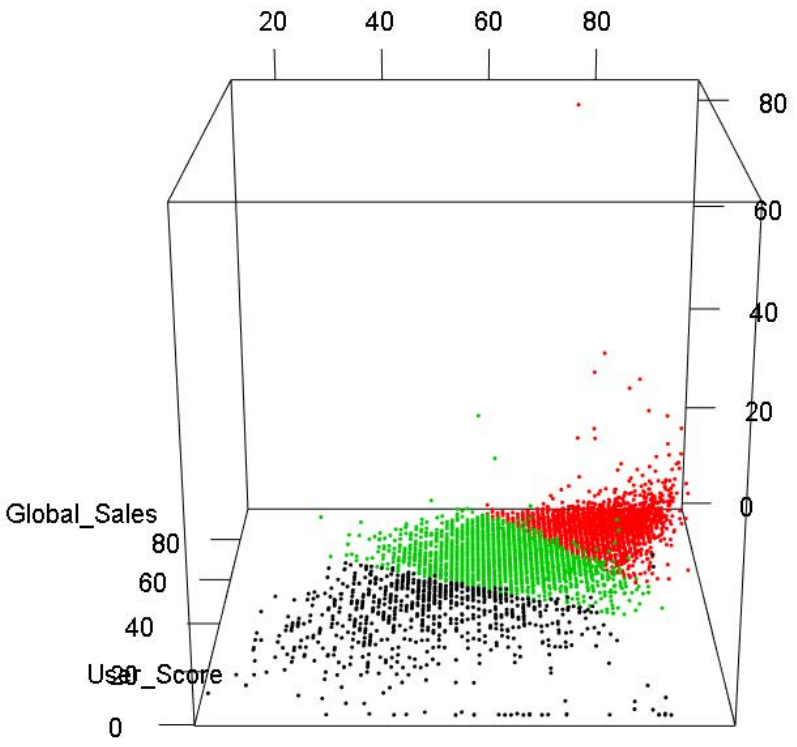
Raw results and visualization

Knowing the number of optimal clusters, we now use the K-means algorithm on the data. The result is repeated 25 times for accuracy which resulted in the following:

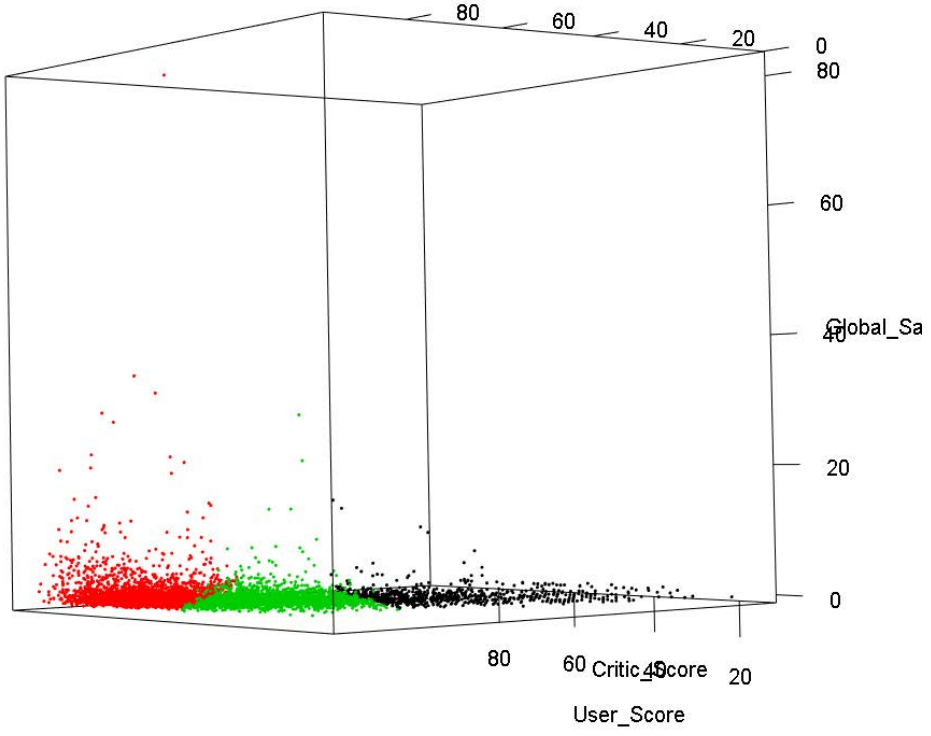
```
> cl
K-means clustering with 3 clusters of sizes 825, 3510, 2720

Cluster means:
  Critic_Score User_Score Global_Sales
1     49.53939   39.74061    0.3706788
2     80.12877   80.10028    1.0806695
3     63.74191   67.30809    0.4777279

Clustering vector:
   [1] 2 2 2 2 3 2 2 2 3 2 2 2 2 2 2 2 2 1 2 3 3 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2 2 3 2 3 3 2 2 2 2 2 2 3 3
  [64] 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 2 2 2 2 2 2 2 3 2 2 2 2 2
 [127] 2 3 2 1 2 2 2 2 2 3 3 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2
 [190] 3 2 2 2 2 2 2 2 2 2 2 2 2 3 3 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 3 2 2 3 2 2 2 2 2 3 2 2 2 2 2 2 2 3 2 2 2 2 2 3 2
 [253] 2 2 1 2 2 2 3 2 3 2 2 2 2 2 2 2 2 2 2 3 2 3 2 2 3 2 2 2 2 2 2 2 2 3 1 3 2 2 2 2 2 3 2 2 2 2 2 2 3 1 2 2 2 2 2 2 3 2 3 2 2 2 3
 [316] 1 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 1 2 2 2 3 2 3 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 3 2 2 2 3 2 2 2
 [379] 2 3 2 3 3 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 3 2 2 1 2 2 1 2 3 2 2 2 2 2 2 2 2 1 2 1 2
 [442] 2 2 3 3 2 3 2 2 2 3 2 3 2 2 1 2 2 1 3 2 3 2 2 3 2 2 2 3 2 2 2 2 2 3 2 2 3 2 2 2 3 2 2 2 3 2 2 2 3 3 2 2 2 3 2 2 2 2 3 2 3 2
 [505] 2 2 2 2 3 2 2 2 3 2 2 2 2 3 3 2 2 3 2 2 3 2 2 2 3 2 2 2 2 3 2 2 3 2 2 2 2 2 2 2 3 2 2 3 2 2 3 2 1 3 3 2 2 2 2 2 3 3 2 2 3 2 2 1 2 2 2
 [568] 3 2 3 2 2 3 2 2 2 3 2 2 2 2 3 1 3 2 2 2 3 3 2 2 2 2 2 3 2 2 2 3 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 3 2 3 2 1 2 2 3 2 1 2 2 2 2 2 3 3
 [631] 2 2 2 2 2 3 2 2 2 3 2 3 3 2 2 2 2 2 3 2 2 3 2 2 2 2 2 3 3 2 2 2 2 2 2 2 2 3 2 2 2 3 2 2 2 3 3 2 1 2 3 2 3 2 2 2 3 2 2 2 1 2 2
 [694] 2 2 1 2 2 3 1 2 2 3 3 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 3 3 2 3 2 2 2 2 2 2 2 3 2 2 2 2 1 3 2 2
 [757] 2 2 2 2 2 2 2 2 2 2 2 2 3 3 2 3 2 2 2 3 2 2 2 3 3 2 3 3 3 2 2 2 2 1 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 1 2 1 2 2 2 2 2 2
 [820] 2 3 2 2 2 2 2 2 2 2 3 2 2 2 3 2 2 2 3 3 2 3 3 2 2 2 2 1 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 3 2 2 2 3 3 2 2 3 2 3 1 3 2 2 3 3 2 2 3 1 2 2 2
 [883] 2 2 2 2 2 3 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 3 2 3 2 2 2 2 3 2 2 3 3 2 2 2 3 2 2 2 3 2 2 2 3 2 2 2 3 2 2 2 2 2 2 2 2 2 3 2 2
 [946] 2 2 3 2 2 2 2 2 2 2 2 2 2 3 2 3 2 2 2 2 3 2 2 3 2 3 2 3 2 2 2 3 2 2 2 2 3 2 2 2 3 3 1 2 2 3 2 2 3 2 2 2 2 2 3 2
 [ reached getOption("max.print") -- omitted 6055 entries ]

Within cluster sum of squares by cluster:
[1] 322286.5 333945.8 427349.5
 (between_SS / total_SS =  64.2 %)

Available components:

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

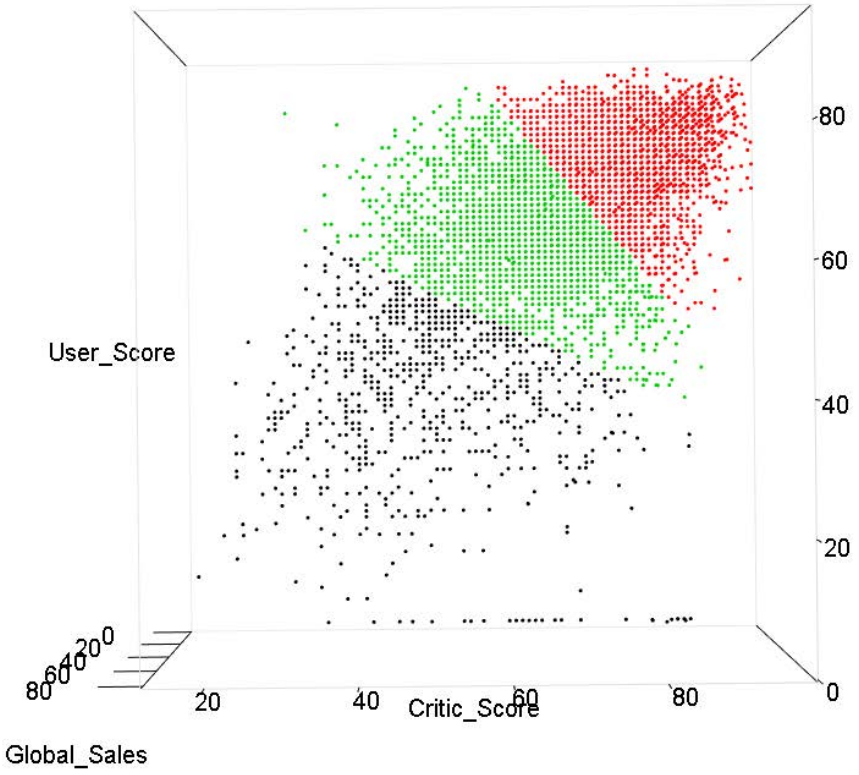
Script

```
cl<-kmeans(extracted4,3,25)
cl
```



Critic_Score





Script

```
library(rgl)
plot3d(extracted4,col=cl$cluster)
```

Summary

From the visualization, where x=Critic_Score, y=User_Score, and z=Global_Sales(millions), the clusters can be grouped into the following categories:

1) Red: High critic score - High user score - High global sales
2) Green: Medium critic score - Medium user score - Medium global sales
3) Black: Low critic score - Low user score - Low global sales

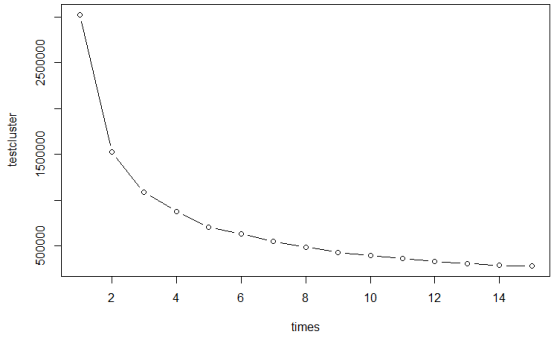
It turns out there are no clusters that would reveal special insights into games with characteristics such as low critic score and lower user score with high sales. The relationship of the clusters turns out to be linear. There are outliers in that regard but it appears to be too little to be significant to be another cluster. Unsurprisingly, video games with high scores have the most variation of sales since that is what most people tend to buy. Category 1 games may potentially be games that are created by hobbyists and independent developers though decent budget games can also fall here which might explain outliers. I can now classify category 1 games as ending at the range of 0-80 critic rating scores and 0-60 user scores with least amount of variation in sales. In category 2, the critic score remains in the same range as category 1, however the difference is in user score where the range is now from 40-90 with slight increase in average sales. where there are high rating, it appears that the deciding factor which makes a game category 2 is the user ratings.

Despite higher overall rating, category 2 which have slightly better sales, is lot denser which means that this seems to be the standard category of average-performing games. Category 2 are most likely games from independent developers that have little exposure and games with standard production values. As for the high score categories, those that have high global sales are quite sparse. Category 3 are games with upwards of 4 million sales with more the highest user rating ranges and minimum critic scores. It is most likely that category 3 games are most likely big-budget video games with high market exposure and high production values with exceptional game play that are worldwide hits. The clusters reveal that there exist common relationship between game global sales and review scores. As a matter of fact, similar games with high user scores are ones that can achieve high global sales. The cluster allows me to derive the various cutoff point for each category for each parameters such as realizing that user scores appears to be the deciding factor for the market performance of games. Many of the games do share similar characteristic in rating and sales which explains their success.

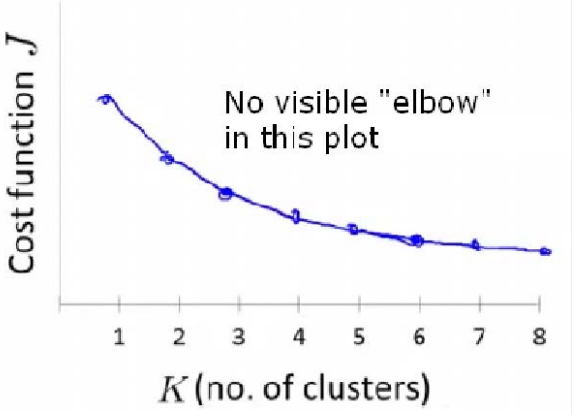
1.5 Measuring and Results Discussion

Model evaluation and success/failure discussion

K-means perform fairly well on this task and clusters than I expected for a linear relationship. However I was hoping to find abnormal clusters, but it created partitioning for categories which allowed me to clearly see the differences of the parameters between those cluster than just normal scatter plots. I initially categorized the data points as just 3 groups by looking at it. In terms of clusters, the elbows have visible distinctions and are not smooth parabolic.

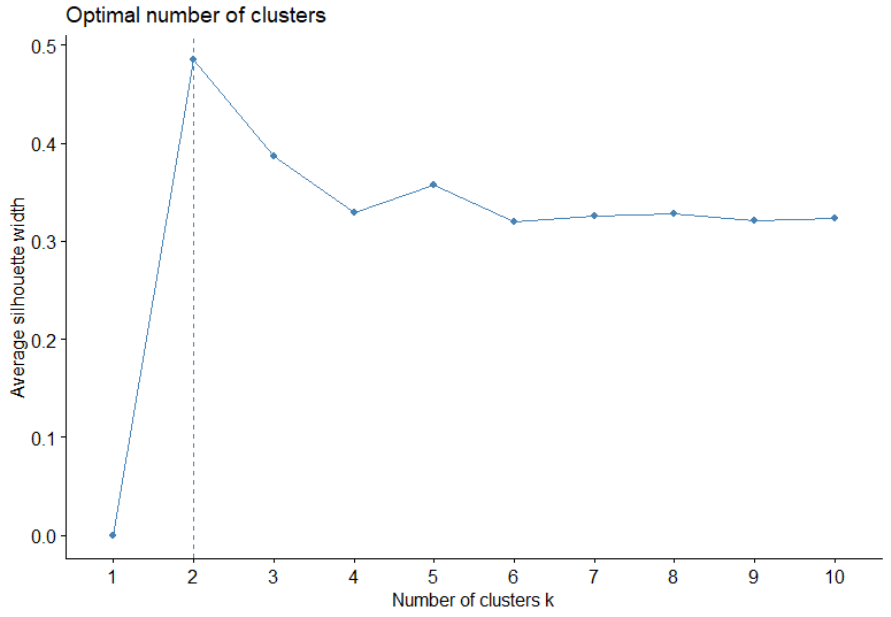


VS



The groupings provided are also quite logical because algorithm successfully broke down each level(low,medium,high) of sales and scores to fit each groups. However, it appears that Category 3 are not tightly clustered together and have very large gaps compared to category 1 and 2.

?q_ kc_qspc rm rcqr rfc cddcargtclcqq md I+kc_lq dmp rfgq r_qi 2 _bbgrgml_j rcqrq sqgle qgjfmscrrc kcrfmb _lb Bsll glbcvcq _pc p_l,



Script
fviz_nbclust(extracted4, FUNcluster = kmeans, method = "silhouette")

This method argues that 2 clusters are optimal for this dataset since the silhouette method would select cluster composition with the highest silhouette width. However, 2 clusters would not reveal any significant insight so I believe that 3 clusters is the minimum I'm using.

```
> dunn1
[1] 0.01155253
```

```
> stats<-cls.scatt.data(extracted4,cl$cluster,dist="euclidean")
> stats$intracls.complete
          c1       c2       c3
[1,] 85.04117 86.56115 78.39276
> stats$intercls.centroid
          c1       c2       c3
c1  0.00000 50.64699 31.01111
c2 50.64699  0.00000 20.79743
c3 31.01111 20.79743  0.00000
>
```

Script
dunn1 <- dunn(clusters = cl$cluster, Data=extracted2)
stats<-cls.scatt.data(extracted4,cl$cluster, dist="euclidean")
stats$intracls.complete
stats$intercls.centroid

The dunn index is quite low which is bad since its one of the indicator for cluster quality. The inter-distance between cluster centroids are also relatively close to each other while there is a large intra-distance of up to 86.5 between the two most remote objects of the cluster. which is bad since clusters should be distinct from each other. The extreme proximity of clusters in this model means that there the dividing line or the gap in which to group each dataset is so small meaning that values are too similar to each other between categories.

The proximity of values means that changes in the algorithm could cause drastic shift in data point between categories which can cause inaccuracy in repeated testing. The large intra distance means that data point can be very extreme from each other indicating high diameter variability when it should be minimized. More variability means the value ranges are nonstandard and dissimilar data are forcibly clustered. Clusters should be homogeneous to each other, meaning that the more uniform the overall size of each cluster, the more similarity those data points the data have. The grouping of the data points are acceptable, however many of the data might be too close to each other and too uniform in general which makes the existence of cluster hard to identify for the clustering model.

Full Script

```
library(factoextra)
library(clValid)
library(tidyverse)
library(clv)
data<-read.csv("Video_Games_Sales_as_at_22_Dec_2016.csv")
extracted<-data%>%select(Critic_Score,User_Score,Global_Sales)
extracted2<-extracted[complete.cases(extracted[ ,1:3]),]
extracted3<-extracted2%>%filter(User_Score!="tbd")
extracted4<-extracted3%>%
mutate(Critic_Score=as.numeric(Critic_Score),User_Score=as.numeric(User_Score),Global
_Sales=as.numeric(Global_Sales))
times<-1:15
testcluster<-sapply(times,function(k){
cl<-kmeans(extracted4,k,25)
cl$tot.withinss
})
plot3d(extracted4,col=cl$cluster)
fviz_nbclust(extracted4, FUNcluster = kmeans, method = "silhouette")
dunn1 <- dunn(clusters = cl$cluster, Data=extracted4)
stats<-cls.scatt.data(extracted4,cl$cluster, dist="euclidean")
stats$intracls.complete
stats$intercls.centroid
```

Supervised Learning - K-Nearest Neighbors Classification Analysis

Red Wine Quality

Gorramuth Prasertkull 6280632

March 27, 2020

2.1 Introduction

This analysis is applied on the Red Wine Quality dataset retrieved from https://www.kaggle.com/uciml/red-wine-quality-cortez-et-al-2009. Among all alcoholic beverages, I have found myself to fancy wine, in particular red wine. It tastes better than beer or vodka, plus its pretty elegant. Since there are so many types of red wine in the world, all with varying compositions, I have decided that finding a way to classify wine quality would be interesting. Specifically, I am interested in the relationship between the acidity of wine and its quality since it is believed that wines with high acidity "taste crispier and more tart" and might age better according to tradition.

2.2 Hypothesis

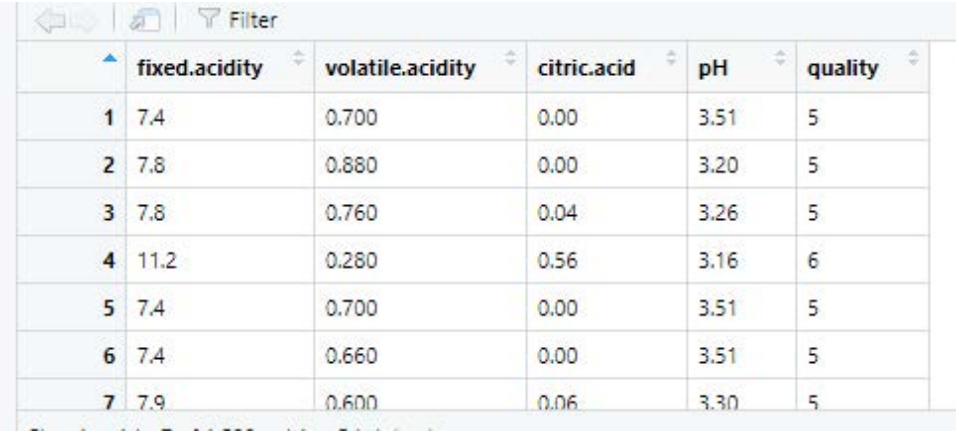
I believe that acidity affects the quality of wines. Perhaps the model associate higher acidity would result in better quality of wine. Thus, I will create a classifier to predict the quality of wine from its acid values.

2.3 Data preparation

	fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides	free.sulfur.dioxide	total.sulfur.dioxide	density	pH	sulphates	alcohol	quality
1	7.4	0.700	0.00	1.90	0.076	11	34	0.9978	3.51	0.56	9.4	5
2	7.8	0.880	0.00	2.60	0.098	25	67	0.9968	3.20	0.68	9.8	5
3	7.8	0.760	0.04	2.30	0.092	15	54	0.9970	3.26	0.65	9.8	5
4	11.2	0.280	0.56	1.90	0.075	17	60	0.9980	3.16	0.58	9.8	6
5	7.4	0.700	0.00	1.90	0.076	11	34	0.9978	3.51	0.56	9.4	5
6	7.4	0.660	0.00	1.80	0.075	13	40	0.9978	3.51	0.56	9.4	5
7	7.9	0.600	0.06	1.60	0.069	15	59	0.9964	3.30	0.46	9.4	5
8	7.3	0.650	0.00	1.20	0.065	15	21	0.9946	3.39	0.47	10.0	7
9	7.8	0.580	0.02	2.00	0.073	9	18	0.9968	3.36	0.57	9.5	7

Transformation and Selection:

The retrieved data is structured in a tidy format that is ready clustering. However, we will be selecting specific features: fixed.acidity, volatile.acidity, citric.acid, pH as the values for training and testing. The label that we will be using in this case would be quality(1-10) from least to excellent.



Filter

	fixed.acidity	volatile.acidity	citric.acid	pH	quality
1	7.4	0.700	0.00	3.51	5
2	7.8	0.880	0.00	3.20	5
3	7.8	0.760	0.04	3.26	5
4	11.2	0.280	0.56	3.16	6
5	7.4	0.700	0.00	3.51	5
6	7.4	0.660	0.00	3.51	5
7	7.9	0.600	0.06	3.30	5

Script

```
data2<-data%>%select(fixed.acidity,volatile.acidity,citric.acid,pH,quality)
data2<-data2%>%filter(!is.na(fixed.acidity),!is.na(volatile.acidity),!is.na(citric.acid),!
is.na(pH))
```

None of the values in this dataset appears to be missing. The data have been converted to a form that is now ready to be trained.

Data Allocation:

The next step is to allocate data for the KNN algorithm with parts of the dataset. The ratio of training and testing that we will be using in this analysis is 80:20. Of the 1599 data points 1279 randomly selected data will be the training set and the other 320 data points shall be used for model validation. I have partitioned the non-repeating data indexes for use in the algorithm as shown below.

Script

```
set.seed(666)
split=0.8
training<-sample(nrow(data2),size=split*nrow(data2),replace=FALSE)
testing<-setdiff(seq_len(nrow(data2)),training)
trainsort<-sort(training)
testsort<-sort(testing)
```

```
> testsort
  [1]    2   10   12   14   18   22   29   31   32   33   37   66   74   81   82   88   89   97   98  116  121  124  129  130  136  137
 [27]  138  151  152  168  177  181  183  185  189  200  203  209  211  215  217  220  226  232  233  235  242  243  247  249  256  264
 [53]  285  296  297  298  302  312  314  315  319  326  329  331  339  340  345  352  354  365  371  375  378  381  384  386  389  390
 [79]  395  397  414  416  418  419  432  435  437  438  444  450  451  462  465  466  476  479  480  482  484  485  488  490  492  499
[105]  502  508  519  523  527  530  531  532  533  537  539  542  553  557  564  568  573  576  577  585  586  589  591  598  603  607
[131]  628  630  631  654  659  664  666  671  681  685  697  711  716  720  721  725  730  731  733  735  740  741  749  750  751  755
[157]  764  766  780  787  796  798  807  808  809  820  821  831  834  836  842  848  858  863  864  865  866  874  883  887  888  889
[183]  890  891  900  905  908  910  916  919  921  926  937  947  949  952  953  954  959  960  966  974  978  983  984  985  992  994
[209] 1010 1018 1032 1039 1048 1051 1061 1064 1065 1079 1087 1093 1099 1103 1106 1107 1109 1112 1113 1122 1123 1125 1133 1137 1146 1153
[235] 1156 1160 1164 1168 1170 1178 1182 1187 1195 1198 1201 1223 1232 1238 1241 1242 1249 1254 1258 1262 1286 1290 1298 1305 1308 1313
[261] 1315 1318 1321 1322 1337 1339 1346 1350 1354 1355 1357 1358 1368 1370 1371 1372 1374 1376 1382 1384 1390 1391 1394 1395 1397 1409
[287] 1411 1413 1420 1423 1430 1435 1441 1447 1457 1459 1460 1468 1469 1473 1475 1482 1483 1492 1498 1499 1539 1542 1553 1564 1569 1572
[313] 1575 1576 1580 1587 1591 1594 1598 1599
>
```

```
> trainsort
   [1]    1    3    4    5    6    7    8    9   11   13   15   16   17   19   20   21   23   24   25   26   27   28   30   34   35   36
  [27]   38   39   40   41   42   43   44   45   46   47   48   49   50   51   52   53   54   55   56   57   58   59   60   61   62   63
  [53]   64   65   67   68   69   70   71   72   73   75   76   77   78   79   80   83   84   85   86   87   90   91   92   93   94   95
  [79]   96   99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  117  118  119  120  122  123  125  126
 [105]  127  128  131  132  133  134  135  139  140  141  142  143  144  145  146  147  148  149  150  153  154  155  156  157  158  159
 [131]  160  161  162  163  164  165  166  167  169  170  171  172  173  174  175  176  178  179  180  182  184  186  187  188  190  191
 [157]  192  193  194  195  196  197  198  199  201  202  204  205  206  207  208  210  212  213  214  216  218  219  221  222  223  224
 [183]  225  227  228  229  230  231  234  236  237  238  239  240  241  244  245  246  248  250  251  252  253  254  255  257  258  259
 [209]  260  261  262  263  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  286  287
 [235]  288  289  290  291  292  293  294  295  299  300  301  303  304  305  306  307  308  309  310  311  313  316  317  318  320  321
 [261]  322  323  324  325  327  328  330  332  333  334  335  336  337  338  341  342  343  344  346  347  348  349  350  351  353  355
 [287]  356  357  358  359  360  361  362  363  364  366  367  368  369  370  372  373  374  376  377  379  380  382  383  385  387  388
 [313]  391  392  393  394  396  398  399  400  401  402  403  404  405  406  407  408  409  410  411  412  413  415  417  420  421  422
 [339]  423  424  425  426  427  428  429  430  431  433  434  436  439  440  441  442  443  445  446  447  448  449  452  453  454  455
 [365]  456  457  458  459  460  461  463  464  467  468  469  470  471  472  473  474  475  477  478  481  483  486  487  489  491  493
 [391]  494  495  496  497  498  500  501  503  504  505  506  507  509  510  511  512  513  514  515  516  517  518  520  521  522  524
 [417]  525  526  528  529  534  535  536  538  540  541  543  544  545  546  547  548  549  550  551  552  554  555  556  558  559  560
 [443]  561  562  563  565  566  567  569  570  571  572  574  575  578  579  580  581  582  583  584  587  588  590  592  593  594  595
 [469]  596  597  599  600  601  602  604  605  606  608  609  610  611  612  613  614  615  616  617  618  619  620  621  622  623  624
 [495]  625  626  627  629  632  633  634  635  636  637  638  639  640  641  642  643  644  645  646  647  648  649  650  651  652  653
 [521]  655  656  657  658  660  661  662  663  665  667  668  669  670  672  673  674  675  676  677  678  679  680  682  683  684  686
 [547]  687  688  689  690  691  692  693  694  695  696  698  699  700  701  702  703  704  705  706  707  708  709  710  712  713  714
 [573]  715  717  718  719  722  723  724  726  727  728  729  732  734  736  737  738  739  742  743  744  745  746  747  748  752  753
 [599]  754  756  757  758  759  760  761  762  763  765  767  768  769  770  771  772  773  774  775  776  777  778  779  781  782  783
 [625]  784  785  786  788  789  790  791  792  793  794  795  797  799  800  801  802  803  804  805  806  810  811  812  813  814  815
 [651]  816  817  818  819  822  823  824  825  826  827  828  829  830  832  833  835  837  838  839  840  841  843  844  845  846  847
 [677]  849  850  851  852  853  854  855  856  857  859  860  861  862  867  868  869  870  871  872  873  875  876  877  878  879  880
 [703]  881  882  884  885  886  892  893  894  895  896  897  898  899  901  902  903  904  906  907  909  911  912  913  914  915  917
 [729]  918  920  922  923  924  925  927  928  929  930  931  932  933  934  935  936  938  939  940  941  942  943  944  945  946  948
 [755]  950  951  955  956  957  958  961  962  963  964  965  967  968  969  970  971  972  973  975  976  977  979  980  981  982  986
 [781]  987  988  989  990  991  993  995  996  997  998  999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1011 1012 1013 1014 1015
 [807] 1016 1017 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1033 1034 1035 1036 1037 1038 1040 1041 1042 1043 1044
 [833] 1045 1046 1047 1049 1050 1052 1053 1054 1055 1056 1057 1058 1059 1060 1062 1063 1066 1067 1068 1069 1070 1071 1072 1073 1074 1075
 [859] 1076 1077 1078 1080 1081 1082 1083 1084 1085 1086 1088 1089 1090 1091 1092 1094 1095 1096 1097 1098 1100 1101 1102 1104 1105 1108
 [885] 1110 1111 1114 1115 1116 1117 1118 1119 1120 1121 1124 1126 1127 1128 1129 1130 1131 1132 1134 1135 1136 1138 1139 1140 1141 1142
 [911] 1143 1144 1145 1147 1148 1149 1150 1151 1152 1154 1155 1157 1158 1159 1161 1162 1163 1165 1166 1167 1169 1171 1172 1173 1174 1175
 [937] 1176 1177 1179 1180 1181 1183 1184 1185 1186 1188 1189 1190 1191 1192 1193 1194 1196 1197 1199 1200 1202 1203 1204 1205 1206 1207
 [963] 1208 1209 1210 1211 1212 1213 1214 1215 1216 1217 1218 1219 1220 1221 1222 1224 1225 1226 1227 1228 1229 1230 1231 1233 1234 1235
 [989] 1236 1237 1239 1240 1243 1244 1245 1246 1247 1248 1250 1251
 [ reached getOption("max.print") -- omitted 279 entries ]
```

2.4 Results

Raw results and visualization

Script

result<-knn(data2[training,-5],data2[testing,-5],data2$quality[training])

```
> result
   [1] 5 5 5 5 6 6 6 6 5 5 6 5 7 6 5 5 5 6 7 5 5 5 5 5 5 6 6 6 5 5 7 5 5 5 6 5 6 6 5 5 6 5 5 6 5 6 5 5 6 6 7 6 6 6 7 6 6 7 5 5 5 7 6 5 7 5
  [67] 5 5 6 5 5 6 6 5 6 6 5 6 6 6 6 7 5 6 6 5 6 5 7 6 6 5 6 6 6 6 6 5 6 5 5 5 6 7 8 6 7 5 5 5 4 5 6 5 6 5 5 5 6 4 5 6 5 5 6 6 6 6 7 6 6 5 6 6 6
 [133] 6 6 6 5 5 4 5 6 5 3 6 5 5 6 5 5 5 6 7 6 5 5 5 6 5 6 6 6 5 5 7 6 5 6 7 5 6 5 5 5 6 7 6 5 7 5 5 6 7 6 6 6 5 5 5 5 5 5 7 7 7 6 7 6 5 7 5 5 6 5 5 5
 [199] 6 6 6 5 7 7 6 5 7 7 7 6 5 5 5 5 6 5 5 5 5 5 7 6 5 6 6 5 6 5 6 5 5 5 7 5 5 5 7 6 6 5 6 6 6 5 7 6 7 5 6 6 6 6 6 5 6 5 5 5 5 6 7 7 6 7 6 6 6 6 5
 [265] 6 7 5 5 5 6 5 6 5 4 5 7 5 4 5 5 5 5 5 6 5 6 5 5 5 6 5 6 6 5 6 6 6 6 5 6 6 5 6 5 6 5 8 5 5 5 5 5 5 5 5 5 5 7 5 6 6 7 5
Levels: 3 4 5 6 7 8
>
```

This is the raw result of the prediction the KNN classifier made for the test dataset. Now we compare the predicted quality to the actual quality of the testing set.

```
> view(data2)
> comparison<-result==data2$quality[testing]
> comparison
   [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
  [23]  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE
  [45]  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE
  [67] FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE
  [89]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE FALSE
 [111]  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE
 [133]  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE
 [155] FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE
 [177]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE
 [199]  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE
 [221]  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
 [243]  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE
 [265]  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
 [287] FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE
 [309]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE
> sum(comparison)
[1] 190
>
```

Script

```
comparison<-result==data2$quality[testing]
sum(comparison)
```

```
> mean(comparison)
[1] 0.59375
>
```

Summary

The result shows that the classifier is only around 60% accurate, only 190 correct guesses out of 320, which is to say that there is slightly more than half chance of the algorithm correctly classifying the wine from its various acidity levels to the quality.

Future prediction could be a hit-or-miss and perhaps there are other attributes which could help to improve its accuracy. But since this analysis is concerned about the liquidity itself the interpretation is that this figure could represent the relationship between the relevance of acidity to wine quality which would result in the verdict of being a somewhat significant factor of determining wine quality. Yet, the prediction based on acidities alone is not good enough to be reliable.

However to test out my assumption that high acidity affect wine quality, I created another fictional dataset with values that are high and low values of acidity and standardize pH levels to test out the classifier with the original training and testing set

Script

```
volatile.acidity<-c(0.76,0.88,0.2,0.11)
fixed.acidity<-c(10.1,11.6,3.2,2.1)
citric.acid<-c(0.99,0.71,0.01,0.3)
pH<-c(3.01,3.04,3.11,3.21)
data3<-data.frame(fixed.acidity,volatile.acidity,citric.acid,pH)
result2<-knn(data2[training,-5],data3,data2$quality[training])
```

```
> result2
[1] 7 6 4 4
Levels: 3 4 5 6 7 8
>
```

It would appear that the first two values with high acid levels are classified as having quite more quality than the following two values which were assigned low acidity value.

2.5 Measuring and Results Discussion

Model evaluation and success/failure discussion

In evaluation of the analysis model that was created firstly the most apparent measure that will be used is the error and accuracy rating of the model.

```
> data2[testing,] %
# A tibble: 6 x 2
  quality count
    <int> <int>
1       3     1
2       4     9
3       5   139
4       6   137
5       7    33
6       8     1
>
```

```
> summary(comparison)
   Mode   FALSE    TRUE
logical     130     190
>
```

Script
data2[testing,] %>%group_by(quality)%>%summarise(count=n())
summary(comparison)

Accuracy: 59.375%
Error: 40.625%

The model is not terrible but it is not good either. There is a very large 40% chance for the model to fail which is almost half of the time. It could possibly be fine tuned to improve further.

```
> split=0.9
> training<-sample(nrow(data2),size=split*nrow(data2),replace=FALSE)
> testing<-setdiff(seq_len(nrow(data2)),training)
> result<-knn(data2[training,-5],data2[testing,-5],data2$quality[training])
> comparison<-result==data2$quality[testing]
> sum(comparison)
[1] 96
```

I reapplied the dataset to KNN with a 90% training set this time. However it turns out that the algorithm the result is still similar providing an accuracy of 60.37%(96/159). It would appears that this is the limit of what the algorithm can achieve with acid values. Inclusion of other parameters would be necessary to improve the model's accuracy further.

The next measure of evaluation is through the confusion matrix in terms of accuracy, precision, and recall.

```
> confusionMatrix(result22,ref)
Confusion Matrix and Statistics

          Reference
Prediction  3  4  5  6  7  8
         3  0  0  1  0  0  0
         4  1  0  3  1  0  0
         5  0  1 97 44  5  0
         6  0  6 30 76 11  1
         7  0  2  8 14 17  0
         8  0  0  0  2  0  0

Overall Statistics

               Accuracy : 0.5938
                 95% CI : (0.5377, 0.648)
    No Information Rate : 0.4344
    P-Value [Acc > NIR] : 7.361e-09

                  Kappa : 0.3457

 Mcnemar's Test P-Value : NA
```

```
> conf$byClass
          Sensitivity  Precision    Recall   Accuracy
Class: 3   0.0000000  0.0000000 0.0000000  0.4984326
Class: 4   0.0000000  0.0000000 0.0000000  0.4919614
Class: 5   0.6978417  0.6598639 0.6978417  0.7107993
Class: 6   0.5547445  0.6129032 0.5547445  0.6462247
Class: 7   0.5151515  0.4146341 0.5151515  0.7157639
Class: 8   0.0000000  0.0000000 0.0000000  0.4968652
>
```

Script

```
ibrary(caret)
ref<-as.factor(data2$quality[testing])
result22<-as.factor(result)
confusionMatrix(result22,ref)
conf<-confusionMatrix(result22,ref)
conf$byClass
```

In general the key performance indicator values of this models appears to be more accurate at quality 5 and 6 wine where there are many training datasets available. Quality 3,4, and 8 wines have only have accuracy statistics due to the lack of sample. The accuracy measures the chances of the wine quality being identified for each quality level.

Precision

Precision measures the accuracy true positive against false positive among the predicted positive values. It would appear that quality 5 and 6 wine have around 40% chances of false positives while quality 7 have up to 60% false positives. The false percentages of each class mean that these are the chances that other wines qualities are wrongly categories into the current category. Quality 5 and 6 wine appears to have the least chance of wrong wine quality with this model.

Recall
measures the ratio of true positive against false negative for the wine qualities. Basically it means that the chances of the wine quality actually being a category but not being categorized as its respective quality. Having high recall means there is a lesser chance of wine being mismatched to other wine quality. In this case quality 5 still have the least class of being mis-categorized while 6 and 7 performed similarly.

Overall, the model is passable for the purpose of classifying wines from its acid composition. There is still a large margin of error and its prediction can be unwieldy but it is does not guess everything wrongly. For the results that it produces, most of the statistics would hover around the 60% margin in many areas of accuracy rating. I would trust this model 60% of the time and its assessment of my dummy dataset is satisfactory.

Full Script

```
library(tidyverse)
library(class)
data<-read.csv("winequality-red.csv")
view(data)
data2<-data%>%select(fixed.acidity,volatile.acidity,citric.acid,pH,quality)
data2<-data2%>%filter(!is.na(fixed.acidity),!is.na(volatile.acidity),!is.na(citric.acid),!
is.na(pH))
split=0.8
training<-sample(nrow(data2),size=split*nrow(data2),replace=FALSE)
testing<-setdiff(seq_len(nrow(data2)),training)
result<-knn(data2[training,-5],data2[testing,-5],data2$quality[training])
comparison<-result==data2$quality[testing]
sum(comparison)

set.volatile.acidity<-c(0.76,0.88,0.2,0.11)
fixed.acidity<-c(10.1,11.6,3.2,2.1)
citric.acid<-c(0.99,0.71,0.01,0.3)
pH<-c(3.01,3.04,3.11,3.21)
data3<-data.frame(fixed.acidity,volatile.acidity,citric.acid,pH)
result2<-knn(data2[training,-5],data3,data2$quality[training])
data2[testing,] %>%group_by(quality)%>%summarise(count=n())
summary(comparison)

library(caret)
ref<-as.factor(data2$quality[testing])
result22<-as.factor(result)
confusionMatrix(result22,ref)
conf<-confusionMatrix(result22,ref)
conf$byClass
```