Predicting the Quality of Wine

6/5/23

DATA 5322

Professor Mendible

Written Homework 4: Unsupervised Learning

Malia Cortez & Ava Delanty

Abstract

This project aims to investigate the thermochemical properties that predict the quality of red wine. The dataset utilized in this study was obtained from Kaggle, originally sourced from the UCI archive [1]. The research covers various machine learning topics, including unsupervised and supervised learning techniques. To conduct exploratory analysis, techniques such as PCA, SVD, K-means clustering, and hierarchical clustering will be employed. Specifically, decision trees and ensemble methods like boosting, bagging, and random forest will be utilized to predict the quality of wine based on its thermochemical properties. The results of the study indicate that the alcohol, volatile acidity, and sulphates play crucial roles in determining the quality of red wine. This study underscores the influence of these factors on an individual's assessment of wine quality based on its chemical properties.

Introduction

This report will commence with an exploratory analysis utilizing unsupervised learning techniques. These techniques encompass PCA, SVD, K-means clustering, and hierarchical clustering. Following the data analysis, a supervised learning model using decision trees will be employed to predict the quality of red wine based on its thermochemical properties. The objective is to develop accurate machine learning models and predictions to identify which response variables in the wine dataset significantly influence the quality of red wine based on people's evaluations.

The red wine dataset consists of 1,599 rows and 12 columns. It includes a column indicating the quality of wine, with values ranging from 3 to 8. A rating of 3 represents the lowest quality, while a rating of 8 represents the highest quality wine. Furthermore, the dataset contains chemical properties of the wine, namely fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, pH, density, sulphates, and alcohol percentage. These findings are of great importance in comprehending the factors that contribute to the superior taste of certain wines in terms of quality, as supported by scientific and chemical properties.

Theoretical Background

Overview: Unsupervised vs. Supervised Learning

This study will utilize both supervised and unsupervised learning techniques. These types of learning differ in several ways. In supervised learning, a predetermined set of predictors are mapped to a particular response through a chosen model. Supervised learning can be used for regression or classification tasks–either predicting a numerical value of a response or predicting

which group a response belongs to, respectively. The model is trained with one set of data, and tested with another set of data. The forms of both the input and output are determined by the user.

In unsupervised learning, there is no determined input or output given to a model. It is conducted to explore the data's underlying structure; this can be difficult to undertake in a supervised setting. Unsupervised learning allows one to easily determine structure, grouping, and relationships between predictors that may otherwise be obscured.

Principal Component Analysis (PCA)

It is common to work with high-dimensional datasets in machine learning spaces. High-dimensional data structures can be difficult to interpret, and relationships between variables can be unclear. PCA, an unsupervised learning technique, reduces the dimensionality of the data and facilitates the exploration of its structure. In PCA, high-dimensional data can be represented in a lower-dimensional space. The original variables are transformed into linear combinations called principal components, and the linear transformations preserve important features and patterns within the data. The dimensionality is reduced by selecting the subset of principal components that contain the most variance.

PCA decomposes the covariance matrix. For each pair of variables, the covariance is calculated, resulting in a covariance matrix for the dataset. Covariance is a metric of how variables tend to behave together. A high positive covariance indicates that the variables either increase or decrease together, while a high negative covariance suggests that the variables are negatively related. The eigenvectors and eigenvalues are derived from the covariance matrix. The eigenvectors correspond to axes in the original space, while the eigenvalues represent the variance explained for a particular eigenvector. These eigenvalues are sorted in descending order to determine the first principal component, second principal component, and so on. These PCA directions represent the most important combinations of variables.

PCA is sensitive to the magnitude of data, thus it is imperative that variables are standardized to ensure no variable dominates in the PCA process due to its large scale.

The location in the principal component space of the m^{th} data point Z can be represented as

$$Z_m = \Sigma^{p}_{j=1} \phi_{jm} X_j$$

Equation 1: Location in Principal Component Space of the Mth Observation

Where there are p total new principal components ϕ along which the data resides.

Singular Value Decomposition (SVD)

Rather than PCA's method of decomposing the covariance matrix directly, SVD decomposes a given matrix into three distinct matrices, the U and V* matrices, and the **Σ** vector. The U matrix represents the location of the matrix in the principal component space. The Σ vector represents the eigenvalues, or the proportion of variance explained. The V* matrix is analogous to ϕ in PCA's principal component direction, which represents the singular value direction.

The singular value decomposition of X is thus represented by

$$X = U\Sigma V^T$$

Equation 2: Singular Value Decomposition of X

Similarly to PCA, the eigenvalues corresponding to the percent variance explained are sorted in descending order to find the first singular value, second singular value, and so on.

K-means Clustering

K-means clustering is an unsupervised machine-learning algorithm used for grouping similar or dissimilar data points into clusters. It operates by iteratively assigning data points to the nearest cluster centroid and updating the centroids based on the mean of the assigned data points. A centroid is a central point within a cluster that represents the center of a group of data points. The user defines the number of clusters to be created. The algorithm initializes the centroids and then calculates the Euclidean distance between each data point and each centroid to assign the point to the nearest cluster. This distance metric may fail when data points are not close to the mean but are clustered according to another metric.

The process continues until convergence, where the centroids stabilize and the assignments remain unchanged. The result is a set of clusters where data points within each cluster are more similar to each other compared to points in other clusters. It is important to note that K-means work only with continuous numerical data. The total within-cluster sum of squares is commonly used as a measure to minimize during the performance of K-means clustering. To avoid undesirable local optima, it is recommended to run the clustering algorithm with a large value of "`nstart()`", such as 20 or 50.

Hierarchical Clustering & Dendrograms

Determining the appropriate number of clusters, especially with a dataset, can be challenging. However, hierarchical clustering offers a solution by aiding in the identification of the optimal number of clusters. Hierarchical clustering constructs a dendrogram, which is a tree diagram illustrating the clustering of observations. It starts from the bottom, identifying similar individual points, and gradually builds the dendrogram. Dendrograms provide insights for various numbers

of clusters simultaneously. By setting a specific cut-off height, the clusters can be determined, analogous to selecting K in k-means clustering. In this method, the most similar points cluster near the bottom, while less similar points form higher branches in the dendrogram. Hierarchical clustering considers all pairwise dissimilarities, treating each observation as its own cluster. The argument "`cutree()`" is utilized to specify the desired number of clusters.

Four linkages are commonly used to calculate distances or similarities between clusters: complete, single, average, and Ward linkage. Single linkage measures the distance between the closest points of two clusters, complete linkage measures the distance between the farthest points, average linkage calculates the average distance between all point pairs, and Ward linkage minimizes within-cluster variance by merging clusters with the least increase in the total within-cluster sum of squares. These linkages play a crucial role in determining the similarity or dissimilarity measures and significantly influence the resulting clustering structure.

Decision Trees: Fitting Classification Trees

Decision trees, a machine learning approach used for classification and regression tasks, will be the method used throughout this study. Decision trees split the data into subsets based on the values of selected features until a decision is reached.Then, the goodness of fit for each split is calculated, a split is chosen and the regions are labeled. The goodness of fit for data can be measured with classification error, Gini index, and entropy. Classification error determines how often each terminal node gives a correct classification which will be used to evaluate the model created in this report. Classification trees specifically have the goal to assign a class label to each input instance. For fitting a classification tree, the algorithm selects a feature that splits the data into subsets with the greatest homogeneity in class labels. The process is repeated recursively.

Pruning

Decision trees, however, have a tendency to overfit the training data, which increases complexity and results in subpar generalization performance on potentially unseen data. Pruning is one of the techniques employed in this study as a result. In order to solve this issue, pruning is performed to remove specific tree components that do not improve the model's overall accuracy. The parameter in pruning that determines the complexity of the tree is α. A higher value of α corresponds to a less complex tree, while a lower value corresponds to a more complex tree. For example, α equaling 0 means that the full complexity of the tree is preserved. Overall, pruning is an excellent technique that will be utilized.

Random Forest

In machine learning, a random forest is essentially a collection (or ensemble) of decision trees, all of which have a say in what the end prediction will be. As stated before, basic decision trees operate by considering all predictors for each split on the tree. For a random forest, the algorithm

only considers a random subset of variables for each tree, while also taking a random sample of observations with replacement. This model has two noteworthy parameters. *m* is the number of predictors considered at each split, and *p* is the number of total predictors in the data set.

Bagging

Bagging, or bootstrap aggregation, is a special instance of the random forest with $m = p$. Basic decision trees are very sensitive to input and prone to overfitting; bagging helps to avoid overfitting. Ignoring some predictors at each split while using random subsets of data allows strong variables that may be overshadowed by dominant variables to have more consideration. This also allows us to create uncorrelated trees as different splits are used, which will improve the model. With bagging and random forests, too many trees cannot be used, but rather computation time will increase. Because of this, it is important to investigate how the out-of-bag error decreases with the number of trees-especially if there are budget constraints. Out-of-bag (OOB) error measures prediction error with boosted trees (more on boosting below), random forests, and other models that use bagging.

Boosting

Boosting is a method in machine learning where the model learns slowly while correcting mistakes along the way to improve the model. First, a small tree is built. Then, errors are calculated, and the tree is scaled based on the performance. Finally, a new tree is built that estimates the residuals. These steps are repeated until the number of trees is sufficiently small or the limit for the number of trees is met. Each tree computed is "shrunk" by the same amount, and the weighted votes of all trees are taken into consideration for the end prediction. Boosting is prone to overfitting, so it is important to cross-validate.

Methodology

Data Cleaning and Processing

The data needed minimal cleaning and processing. There were no missing values present, nor were there inconsistent or nonsensical values. Imperatively, before performing dimensionality reduction, the data was scaled with the `scale = TRUE` parameter for PCA, and with the `scale()` function for SVD.

For the decision tree, random forest, and bagged models, the data was refactored into three quality categories: low quality (3-4), average quality (5-6), and high quality (7-8) wine. This decision was largely due to the disparities in observation counts for each category, as can be seen in the appendix. For the boosted model, these categorizations were not used in favor of the original numerical categorizations.

Unsupervised Learning Models

PCA and SVD

To begin, all variables except for wine quality were selected. All variables were numerical, allowing for easy analysis with PCA and SVD. Both PCA and SVD were conducted after standardizing the data.

For PCA, the parameter `scale = TRUE` was utilized in the `prcomp()` function, which performs PCA on the input and returns a PCA object. To calculate the percent of the variance explained, the PCA object's `sdev` column was used. More specifically, this standard deviation was squared and divided by the sum of the standard deviations squared in order to receive the percent variance. Additionally, the principal component loadings were obtained by extracting the PCA object's `rotation` matrix, with the first column of the matrix corresponding to the first principal component, the second corresponding to the second principal component, and so on. To retrieve the important variables for each principal component, the magnitude of the principal components was calculated with `abs()` and sorted in descending order.

For the SVD analysis, the `scale()` function was first used to standardize the data. Then, the svd() function was run to perform SVD on this scaled data. To obtain the percent of variance explained, the SVD object's `d` vector was used. It was calculated by dividing the values in the object's `d` vector by the sum of the `d` vector. It is important to note here that the `d` vector in this context represents the eigenvalues indicated in Equation 2's parameter Σ.

K-means Clustering

The number of clusters to be performed was determined by checking the dataset after scaling. It was found that 6 clusters would provide the most predictive results (see appendix for plot). The data was then visualized, focusing on the variables alcohol and sulphates, using `nstart 20`. After calculating the total within-group sum of squares, the proportion of variance was interpreted. K-means was then applied to the transformed data in the singular vector space using the same two variables. A confusion matrix was generated to examine the clusters based on wine quality and the means of the variables for each cluster.

K-means Plots

To gain an overall understanding of the dataset, the original data was plotted, with the color representing wine quality. K-means was then performed with 6 clusters to observe its correlation with wine quality. Furthermore, the SVD of the scaled data was plotted on the first and third singular vectors, revealing more distinct separations among the clusters. Subsequently, k-means

was applied to the data in the singular vector space, and the resulting clusters were plotted. This plot accurately displayed the clusters separated based on their wine quality.

Hierarchical Clustering

After creating the k-means plots, hierarchical clustering was performed. Four dendrograms were generated using different linkage methods: complete, single, average, and ward. Scaling was applied to the data using complete linkage and visualized for further analysis. Additionally, a correlation-based distance was computed and plotted using complete linkage. Scaling and creating a correlation-based distance were also performed with average linkage to compare the plots. Due to its lower interpretability compared to other unsupervised learning models, Hierarchical Clustering results will be presented in the appendix rather than the main results section.

Supervised Learning Model

Multi-Class Classification Using Decision Trees

For the multi-class classification task, the first step involved constructing a decision tree without any training or test data. Then, the dataset was split into a 70-30 ratio for training and test sets. Following that, a decision tree was created, and pruning was applied to find the optimal value of K. The cross-validation technique was used to determine if any variables could be eliminated. The test error rates of both pruned and unpruned trees, as well as the training error rates, were compared. Random forest and bagging methods were employed using the training and test split to compare Mean Squared Errors (MSEs), the number of variables used for splitting each tree, and the performance of the models on the test set. A variable importance plot was used to interpret which chemicals have the most influence on wine quality.

Additionally, the boosting method was applied to the same training and test set using the Gaussian distribution parameter, an interaction depth of 4, and 1000 trees. After computing the MSE, adjustments were made to the shrinkage and number of trees to enhance the model's accuracy. When the number of trees was set to 100 with the default shrinkage, the resulting MSE was higher, leading to the abandonment of that particular model. By changing the shrinkage to 0.01, the model's accuracy and MSE improved.

Computational Results

Exploratory Analysis

PCA and SVD Plots

Figure 1 contains the proportion of variance explained along the 11 total principal components.

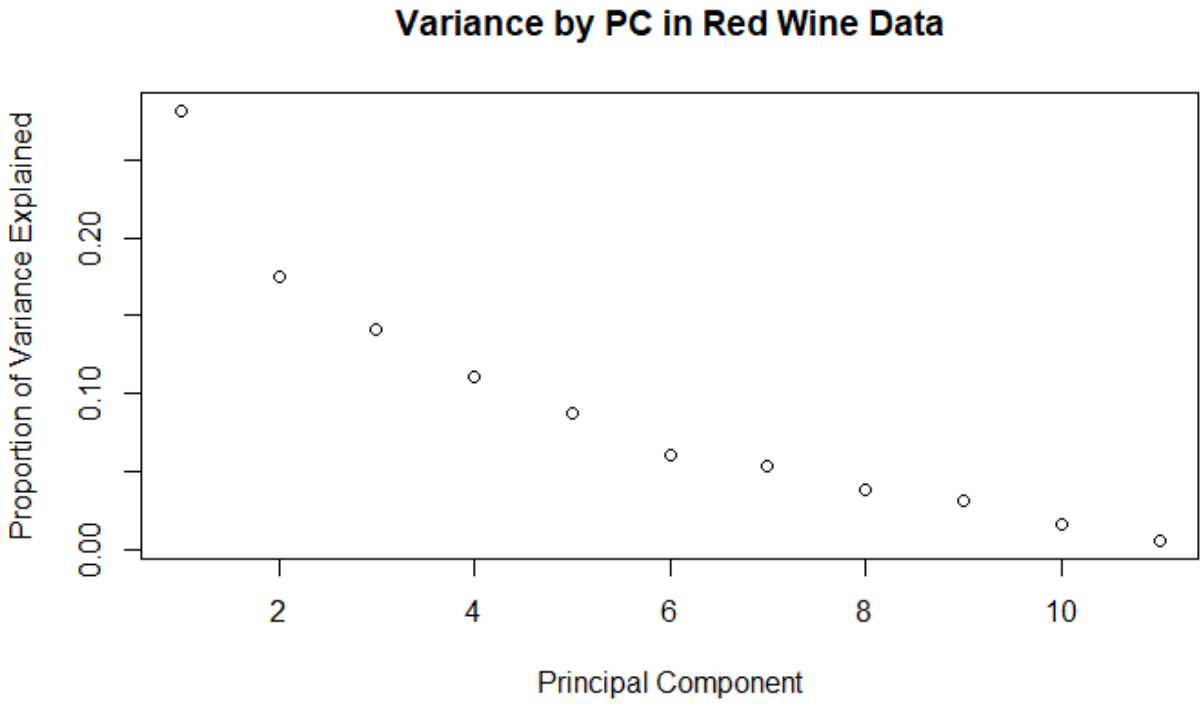


Figure 1: Proportion of Variance Explained by Principal Component

The first principal component accounts for over 25% of the variance, while the second principal component accounts for just under 20% of the variance. There is a smooth and gradual downward curve with this plot. The proportion of variance explained continues to gradually decline.

Figure 2 contains the cumulative percent variance explained by the 11 principal components.

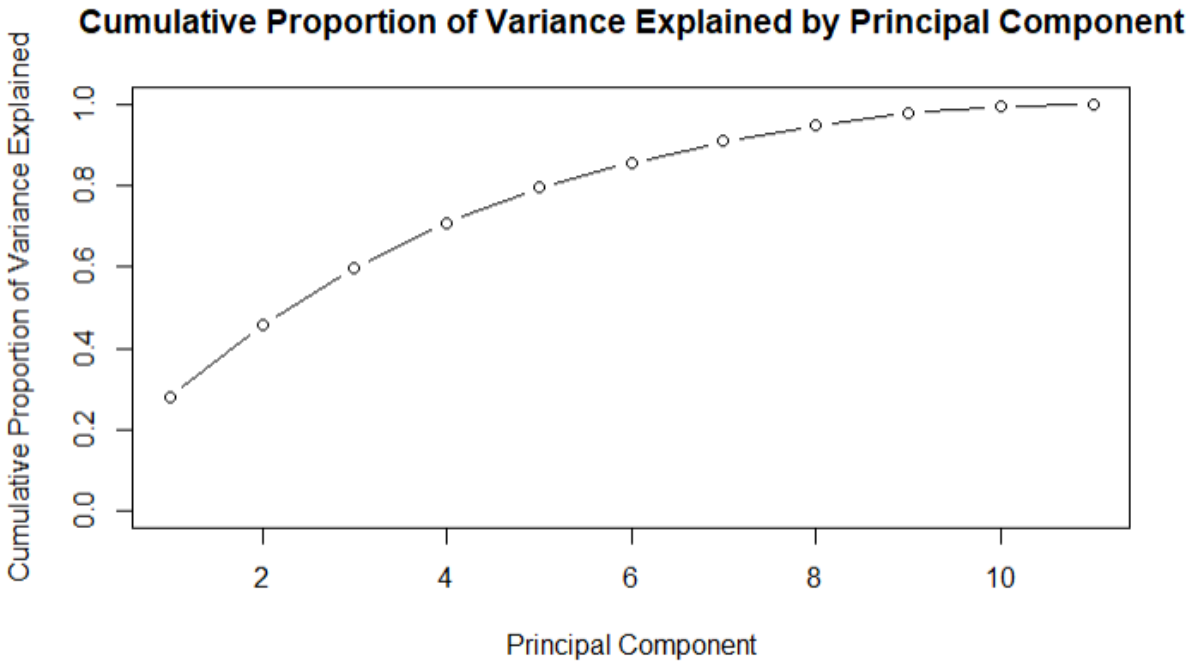


Figure 2: Cumulative Proportion of Variance Explained by Principal Component

The first six principal components account for just over 80% of the variance in the data. The cumulative proportion of variance explained has a smooth upward curve. Eventually, all 11 components account for 100% of the variance.

Important variables in the first two principal components were examined. For the first principal component, fixed acidity, citric acid, pH, and density were the top four most important variables. For the second principal component, total sulfur dioxide, free sulfur dioxide, alcohol, and volatile acidity were the top four variables.

Figure 3 plots the data along these first two principal components, colored by wine quality.

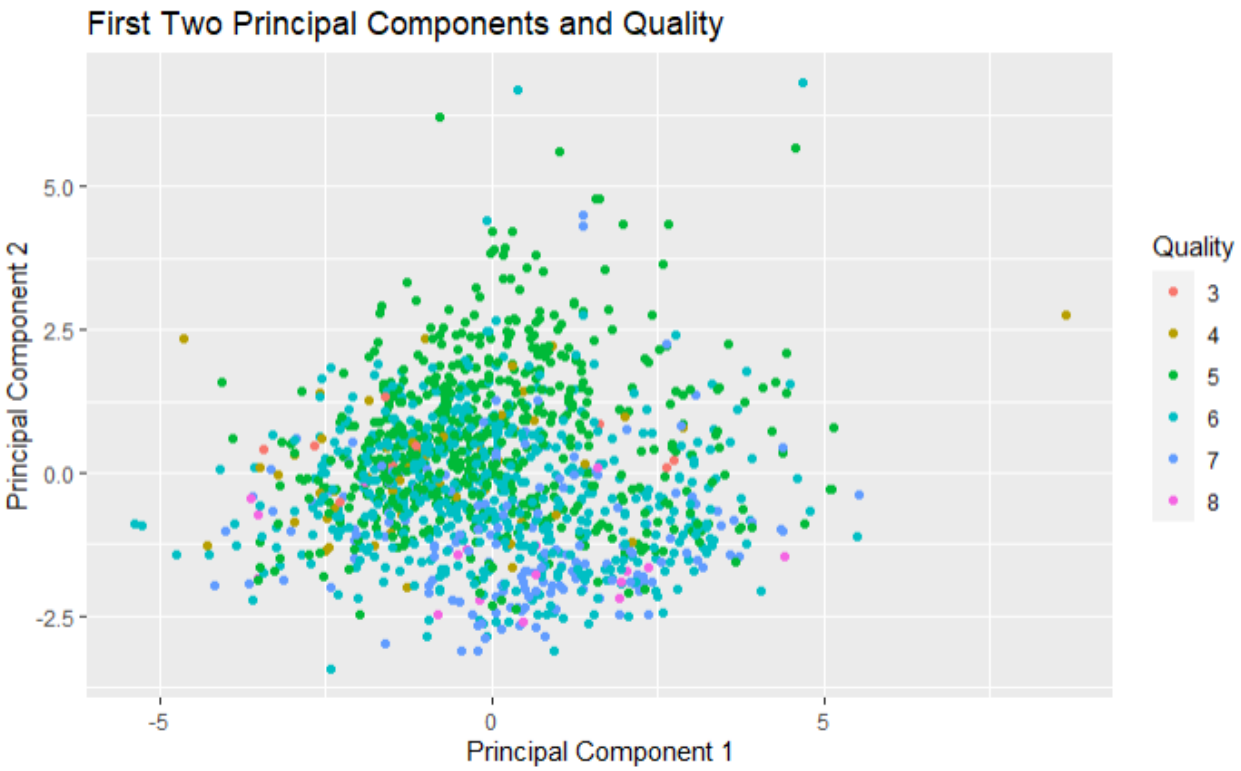


Figure 3: Principal Components 1 and 2 Colored by Quality

From this principal component plot, not much structure can be seen from the data. The quality of wine has many overlapping points and distinct clusters are not present.

To compare to the principal component plot, the most important variables from principal components 1 and 2 (fixed acidity and total sulfur dioxide, respectively) were plotted in their raw form in Figure 4 below.

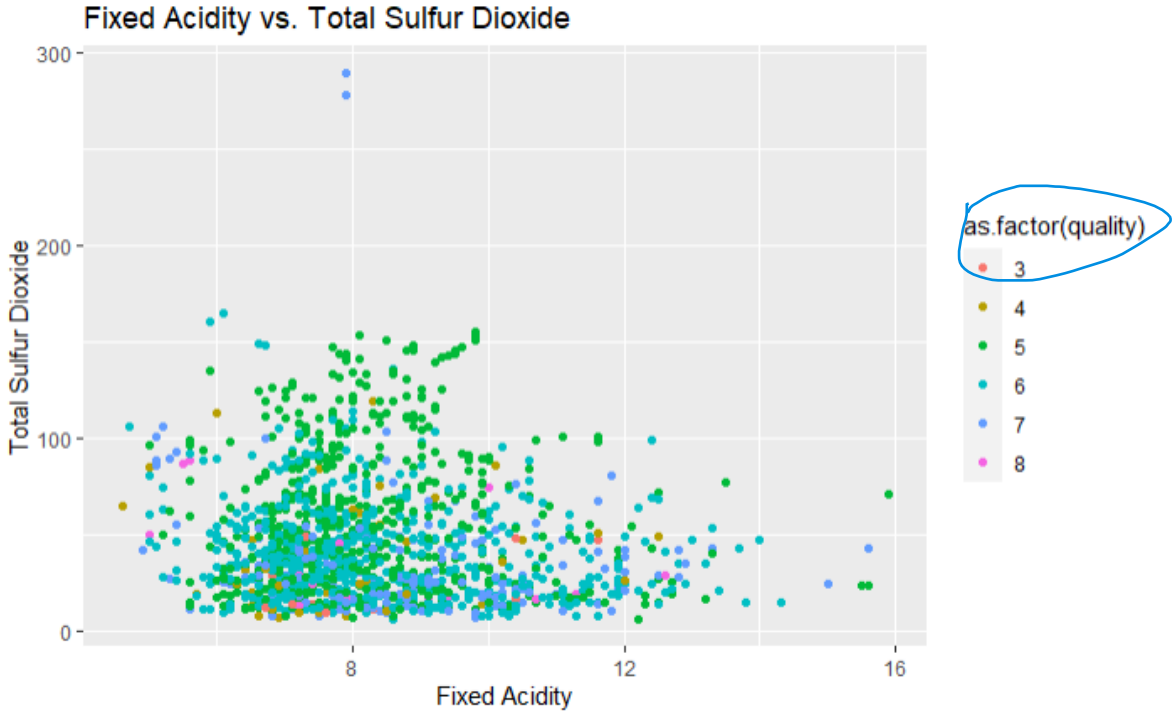


Figure 4: Fixed Acidity vs. Total Sulfur Dioxide

Compared with the principal component plot, the raw plot of fixed acidity against total sulfur dioxide contains a much more significant concentration of points.

K-Means Plots

K-means were applied to the data in the singular vector space, and the resulting clusters were plotted. Using the variables fixed acidity and total sulfur dioxide that were important from the PCA analysis this was the resulting plot. Additionally, the variables volatile acidity and sulphates were plotted which were the top two variables from the decision tree modeling. It is noticeable that the clusters are still overlapping slightly but are more separated than the raw plots of the dataset.

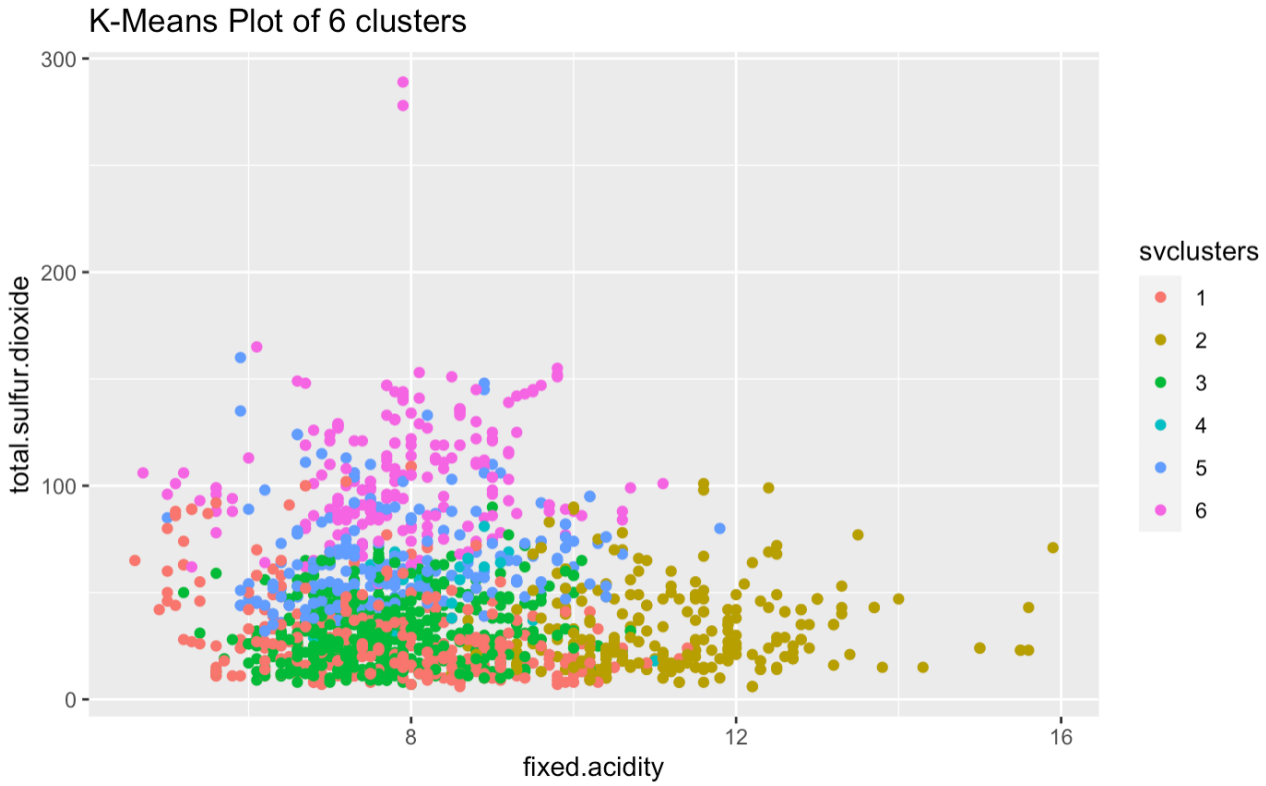


Figure 5: K-Means Plot with Fixed Acidity and Total Sulfur Dioxide

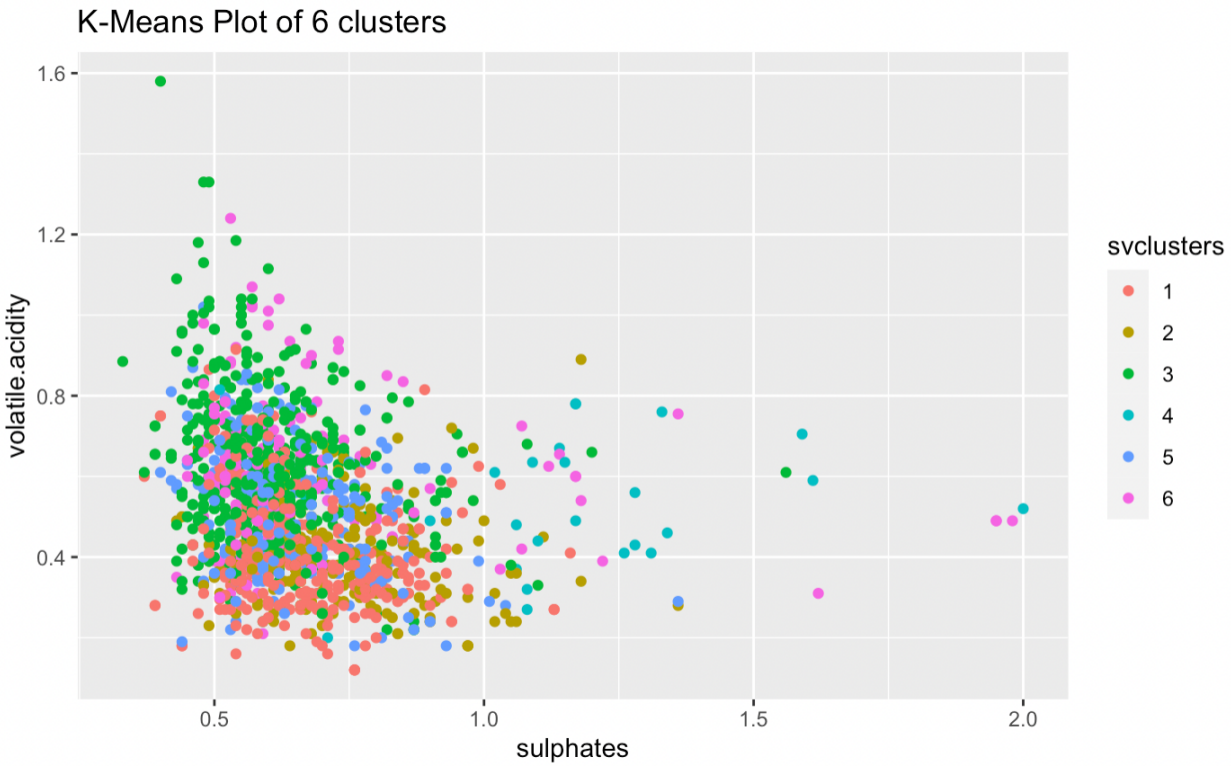


Figure 6: K-Means Plot with Sulphates and Volatile Acidity

K-Means Clustering

Plotting with the variables alcohol and sulphates and using nstart 20 the total within-group sum of squares was 9358.22. The proportion of variance was then interpreted and was found that the variance starts to decrease and around an index of 7 for the most efficient results. After performing k-means on the transformed locations of the data in the singular vector space, the total within-group sum of squares was 7.85 significantly lower than the previous k-means cluster plot. A confusion matrix was then computed between the VD-based clusters and the wine quality which determined that the 6 clusters were of sizes 412, 310, 232, 46, 25, and 574. The quality of wines wasn't distributed equally based off the matrix.

Supervised Model Decision Trees

Multi-Class Classification

Problem: Predicting the Quality of Wine based on Thermochemical Properties

Decision Trees

Results of the tree model without a training and test set gave a training error rate of .15 or 15%. The decision tree that was produced used 6 different variables. The variables were alcohol,

sulphates, total sulfur dioxide, volatile acidity, fixed acidity, and pH. The training and test tree gave a training error rate of .14 or 14% with 6 different variables. The variables were alcohol, total sulfur dioxide, sulphates, volatile acidity, fixed acidity, and citric acid. Estimating the test error gave an accuracy of 83.54% in the test data.

After cross-validating and finding the optimal K, 5 variables were used in the pruned tree. The variables were alcohol, sulphates, total sulfur dioxide, volatile acidity, and fixed acidity. The pruned tree training error rate was .15 or 15%, therefore performing slightly worse than the previous tree model. Additionally, the test error gave an accuracy of 82.91% of the test data which therefore the pruned tree performed worse by around 1%.

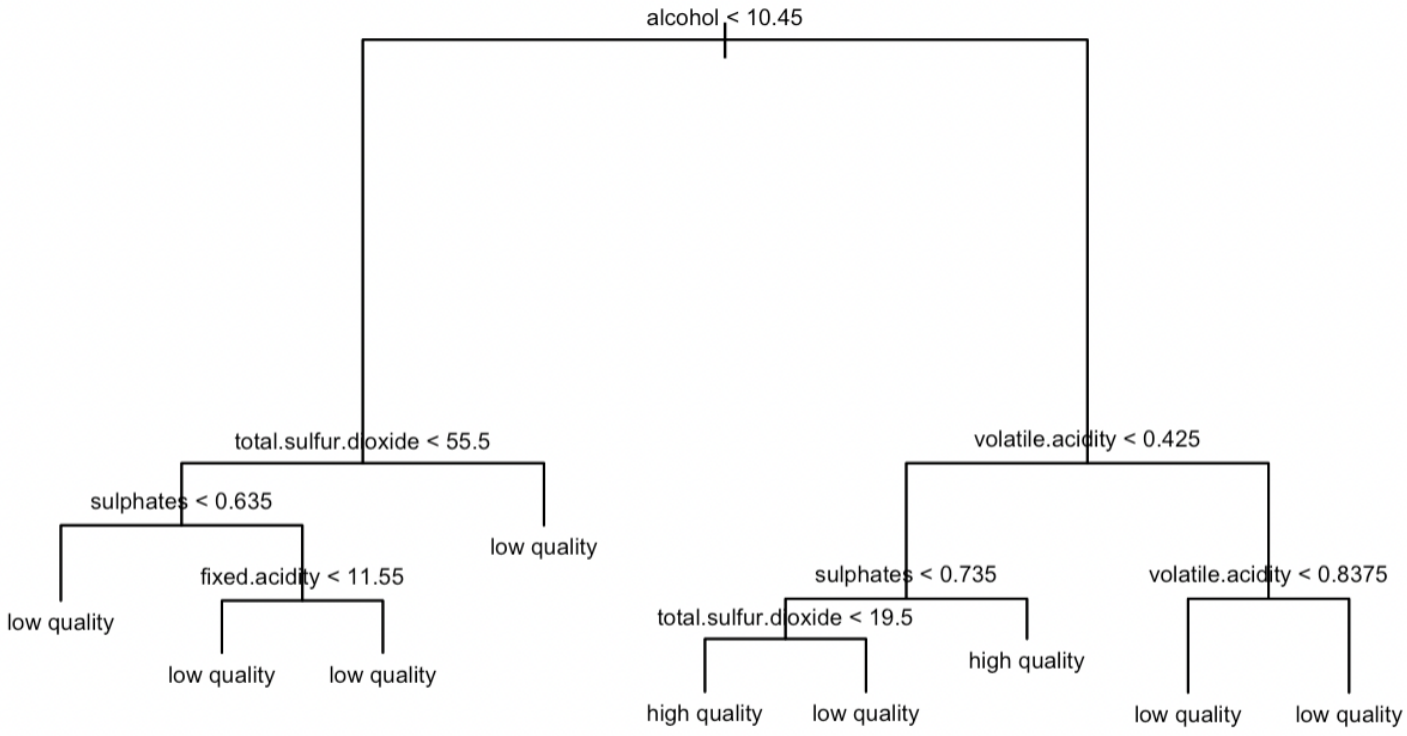


Figure 7: Pruned Decision Tree

Bagging and Random Forest

After using the ensemble method bagging the model produced an MSE of .391 using all variables at the splitting of each tree. Based on the mean decrease accuracy plot the results indicate that across all of the trees considered in the random forest, the two most important variables were alcohol and sulphates. Based on the mean decrease gini plot the two most important variables were alcohol and volatile acidity.

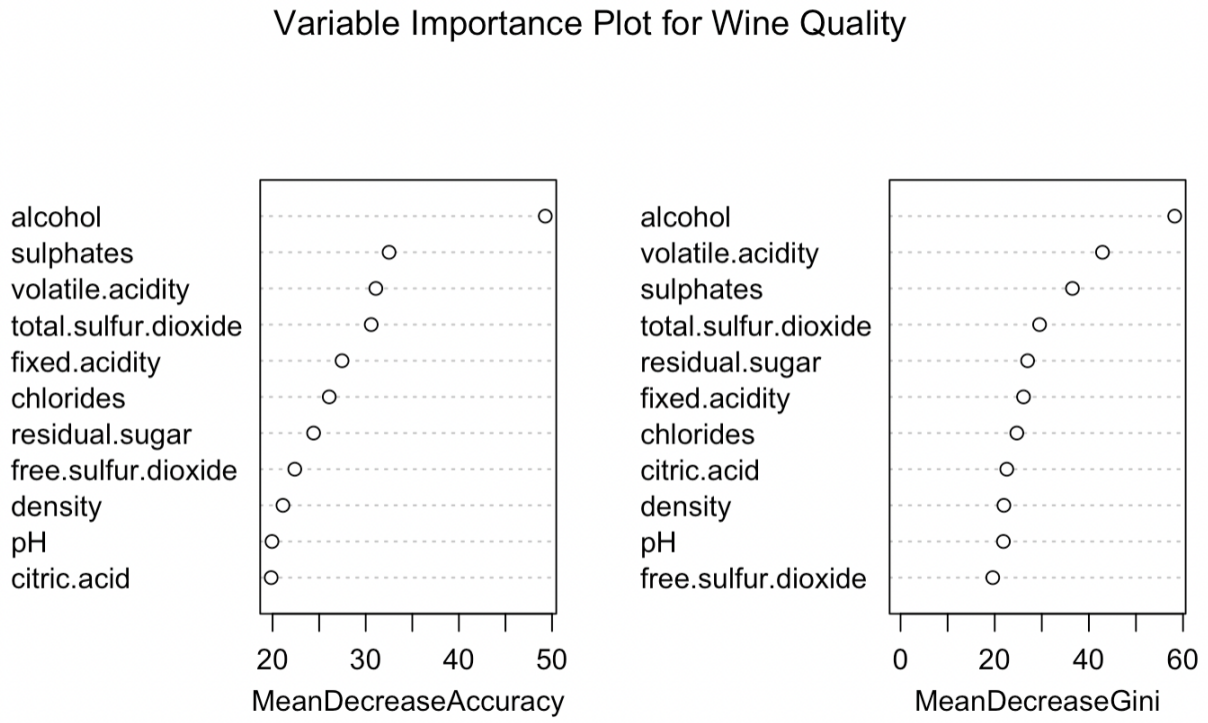


Figure 8: Variable Importance Plot for Wine Quality

Boosting

Boosting was then performed to see which variables influence wine quality the most. The most important variables were again alcohol and volatile acidity. The MSE was .394 which indicates that boosting was slightly worse than the bagging and random forest model. In conclusion, the most important variables that influence wine quality were alcohol, sulphates, and volatile acidity.

Discussion

In this context, the `x` value from the PCA can be interpreted as the original data matrix. The shape of the `x` matrix in PCA is 1599x11, representing the 1,599 rows and 11 variables that had PCA performed on them. Since this value represents the data matrix, this is what is plotted along the first two principal components plot in Figure 3. In this principal component space, not much structure exists in the data, but the points are more spread out than in Figure 4's plot with the two original features. Figure 4 contained much more overlap than the plot along the first two principal components. The concentration of points in the original variables plot is much more significant than in the principal components plot. Again, it is important to keep in mind the disparities in counts for each quality category; the groups ranged from just 10 observations to 681 in a single category. These disparities could also make it difficult to more easily distinguish between groups. PCA did maximize the variance present in the data (especially when compared against the original variables plot), but distinct clusters were still not found despite this analysis.

The `rotation` matrix represents the location of the wine in principal component space. In particular, it provides the principal component loadings. When matrix-multiplied by the `x` data matrix, the coordinates of the wine in principal component space are derived. Based on the proportion of variance explained plot (Figure 1), the first column of the `rotation` matrix accounts for just over 25% of the variance explained, while the second column of the matrix accounts for just under 20% of the variance. Furthermore, the cumulative proportion of variance explained plot (Figure 2) showed that the first 6 principal components (or, the first 6 columns of the `rotation` matrix) accounted for over 80% of the total variance. These results certify that the variance was maximized for the data.

After clustering the dataset with K-means, it became evident that the sizes of the clusters were not evenly distributed due to the scarcity of high and low-quality wine ratings compared to average ratings. To compare and interpret the clusters, examining the means of the variables within each cluster was done. In particular focusing on Cluster 5, which had a size of 25, and Cluster 2, which had a size of 310 provided underlying observations on the clusters.

Cluster 5 stood out with significantly high levels of fixed acidity, volatile acidity, free sulfur dioxide, and total sulfur dioxide. The cluster also exhibited slightly above-average levels of density and below-average levels of pH and alcohol. This suggests that Cluster 5 represents wines with relatively lower acidity levels. The notably high levels of chlorides and sulphates within this cluster may contribute to a specific taste profile or indicate a distinct wine style. The slightly above-average density suggests a more concentrated nature of the wines compared to the other clusters.

On the other hand, Cluster 2 stood out with high levels of fixed acidity, citric acid, density, and low levels of volatile acidity, free sulfur dioxide, total sulfur dioxide, and pH. The cluster exhibited an average level of residual sugar and slightly below-average levels of alcohol. This suggests that Cluster 2 represents wines with high levels of fixed acidity and citric acid. The

slightly above-average level of sulphates and slightly below-average level of alcohol could contribute to the overall flavor and mouthfeel characteristics of the wines within this cluster. Overall, the clusters differentiate based on properties such as acidity, density, and alcohol content, providing valuable insights into how these clusters are formed and how wine characteristics vary within them.

Using decision trees, bagging, and boosting methods is beneficial for predicting wine quality based on chemical attributes due to their ability to handle complex relationships and capture non-linear patterns. Decision trees split the data based on chemical attributes, identifying important variables and interactions. Bagging and boosting methods can accurately predict wine quality by considering the intricate relationships between chemical attributes and their impact. Unsupervised methods provide a holistic view, highlighting natural groupings without considering quality labels. In contrast, supervised methods focus on specific relationships between chemical properties and quality ratings, providing targeted insights for prediction. The most important variables in the supervised models were alcohol, volatile acidity, and sulphates. Overall, unsupervised methods explore underlying patterns, while supervised methods offer predictive power by identifying key chemical properties indicative of wine quality.

Conclusion

This study aimed to explore and predict red wine quality based on its thermochemical properties. The important variables identified in the unsupervised models differed from those in the supervised models. The supervised models had a good performance overall, with the best model achieving an accuracy of 83.54%. However, it is important to consider that the dataset focused solely on the subjective metric of "quality" and the scientific aspects of wine. Including additional features such as the brand and price of the wine could provide further insights into the relationship between quality and product factors. Comparing these results with a similar analysis on the white wine dataset would also be valuable. Overall, the study successfully predicted red wine quality, identified important determinants of quality, and highlighted the influence of alcohol, volatile acidity, and sulphates on the assessment of wine quality.

Bibliography

[1] Priyadarsini, M. (2017). Wine Quality Selection. Retrieved May 20, 2023 from https://www.kaggle.com/datasets/maitree/wine-quality-selection?resource=download

Appendix Attached Below:

Written HW 4 - Wine Quality

Malia Cortez, Ava Delanty

2023-06-05

Load in Data

```
red <- read.csv('../data/winequality-red.csv',sep=',')
white <-  read.csv('../data/winequality-white.csv',sep=',')
```

Exploratory Analysis

The predictor variable is quality

```
unique(red$quality)
```

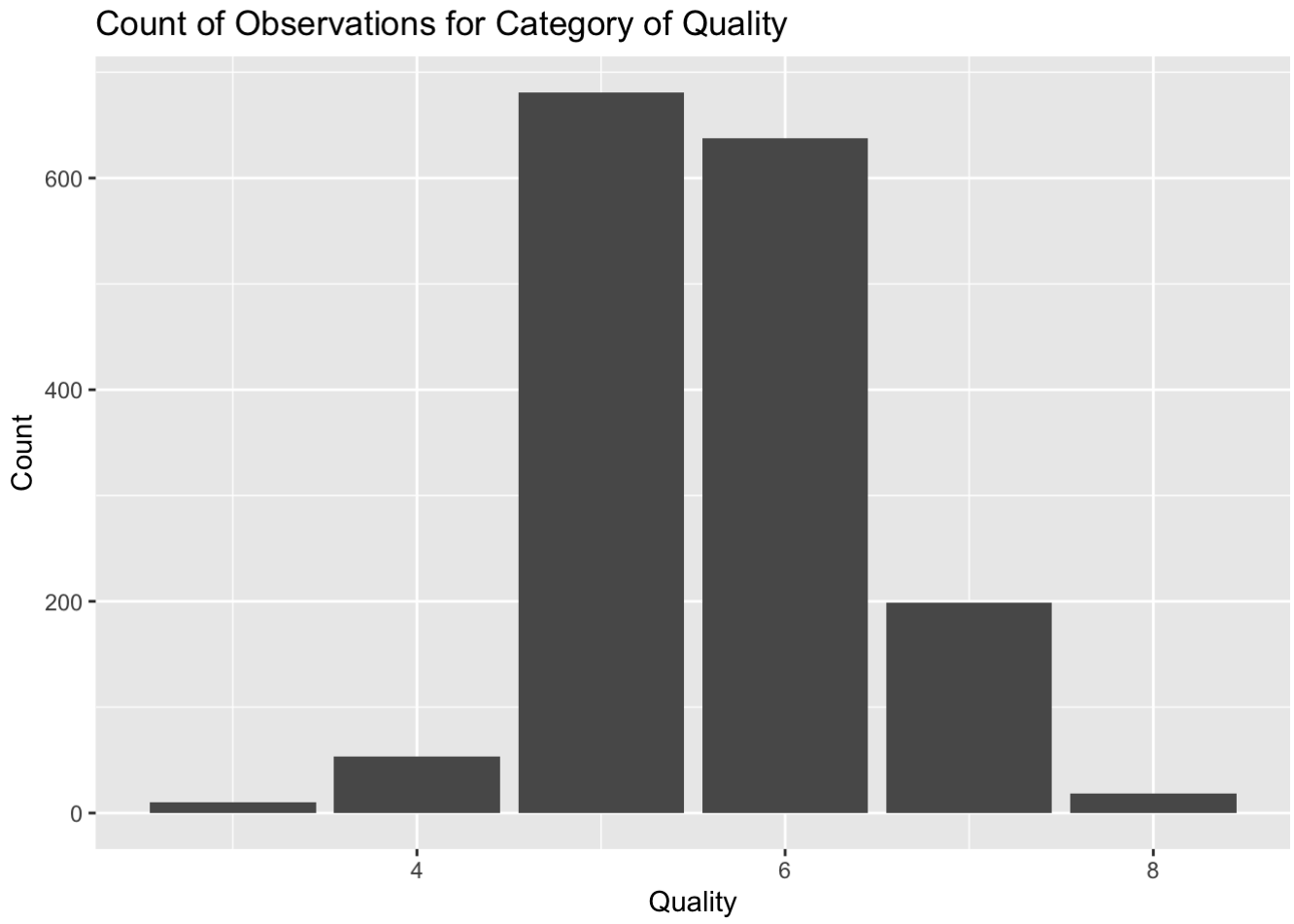
```
## [1] 5 6 7 4 8 3
```

Note: Let's assign 3-4 as low quality wine, 5-6 as average quality wine, and 7-8 as high quality wine for supervised learning models

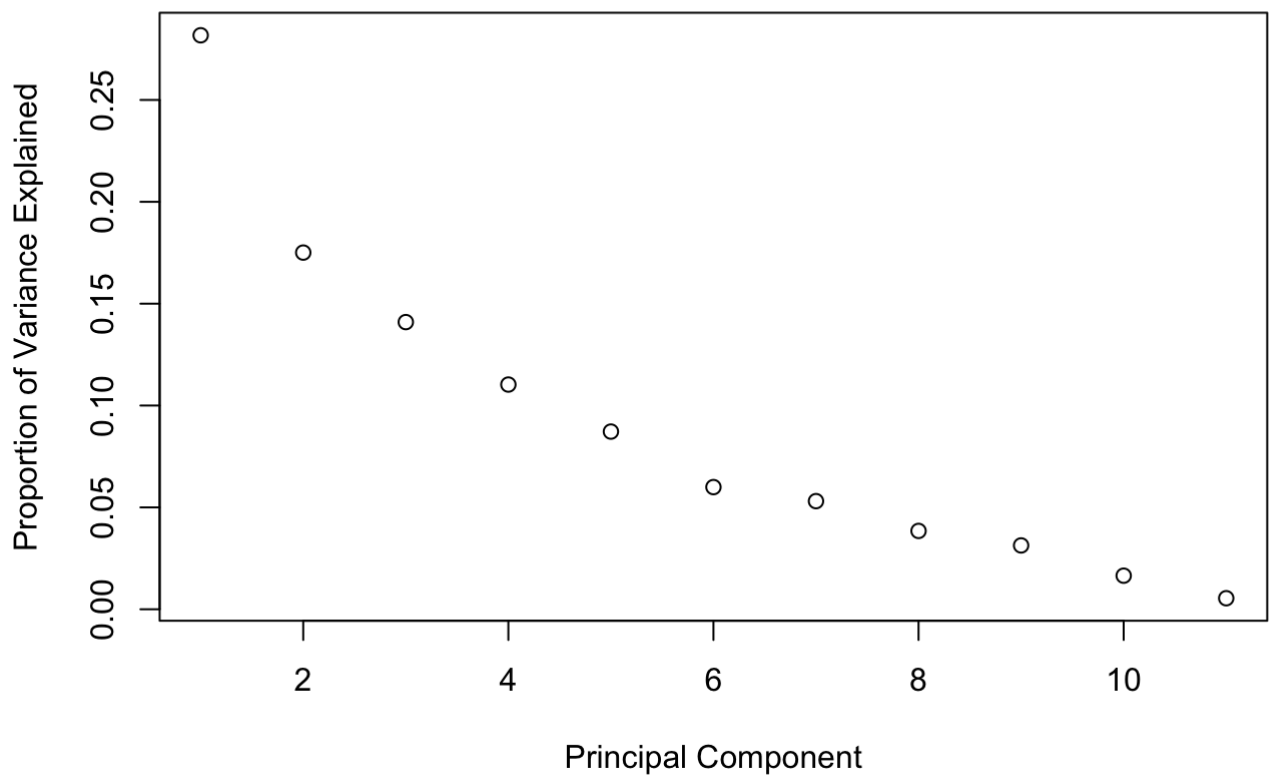
Unsupervised Models

PCA:

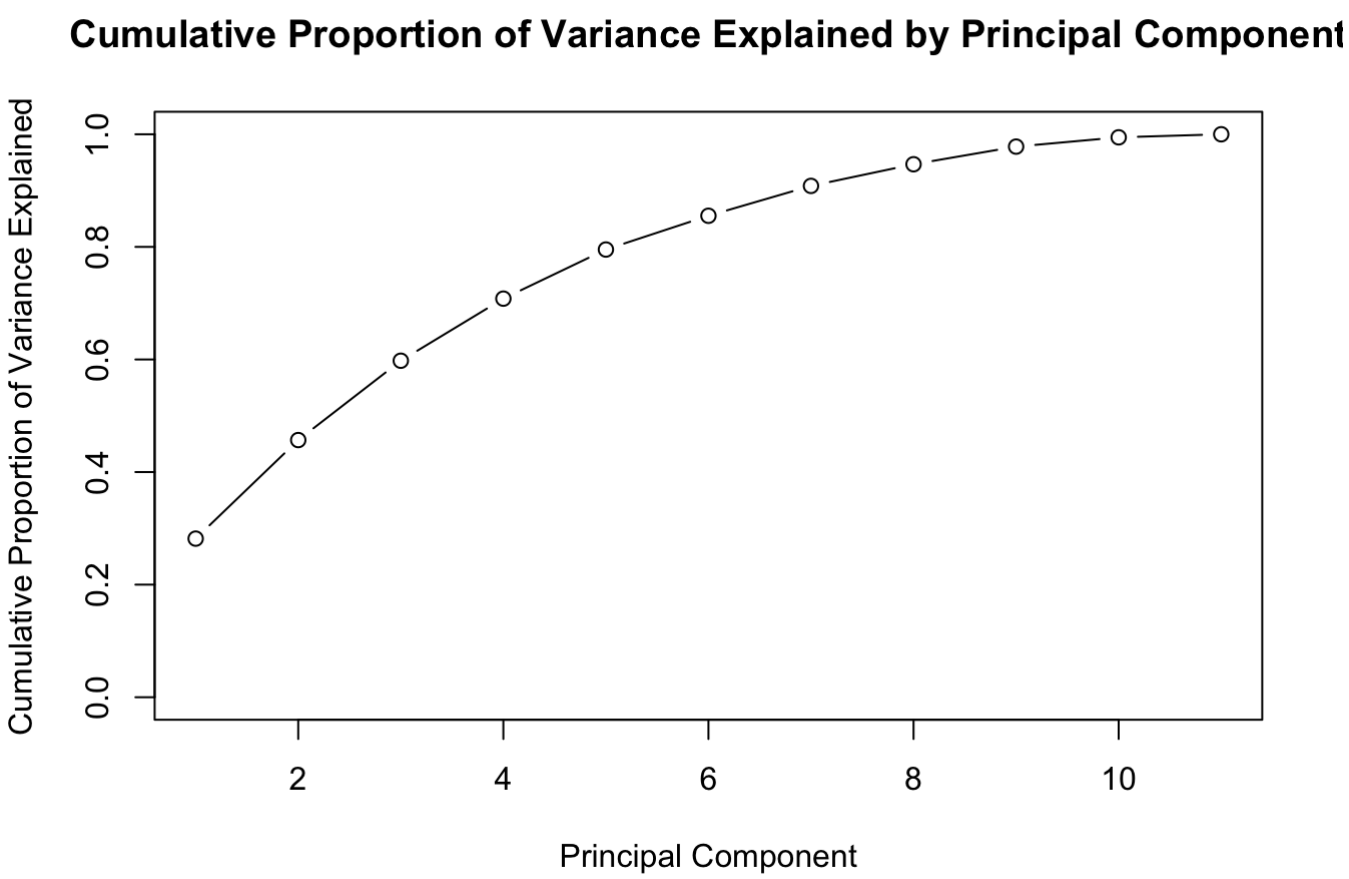
```
# red wine X input
# get counts for each quality category
quality.counts <- red %>%
  group_by(quality) %>%
  count()

ggplot(quality.counts, mapping = aes(quality, n)) +
  geom_col() +
  labs(x = "Quality", y = "Count", title = "Count of Observations for Category of Qualit
y")
```




```
X <- red %>% select(-quality)

# use PCA with scaled data
pr.out <- prcomp(X, scale = TRUE)

# get percent variance explained by principal components
pve <- pr.out$sdev^2 / sum(pr.out$sdev^2)

plot(pve, ylab = "Proportion of Variance Explained", xlab = "Principal Component", main
= "Variance by PC in Red Wine Data")
```




```
# plot cumulative variance explained by principal components
# we can see that the first 6 principal components account for ~80% of the variance expl
ained
plot(cumsum(pve), xlab = "Principal Component",
    ylab = "Cumulative Proportion of Variance Explained",
    ylim = c(0, 1), type = "b", main = "Cumulative Proportion of Variance Explained by P
rincipal Component")
```




```
# get most weighted variables
## for the first PC, fixed acidity, citric acid, pH, and density are all highly weighted
## pH is negatively weighted, which we can see when we remove the absolute value
pr.out$rotation[,1] %>% sort(decreasing=TRUE)
```

```
##        fixed.acidity          citric.acid              density
##           0.48931422           0.46363166           0.39535301
##            sulphates            chlorides       residual.sugar
##           0.24292133           0.21224658           0.14610715
## total.sulfur.dioxide  free.sulfur.dioxide              alcohol
##           0.02357485          -0.03615752          -0.11323206
##     volatile.acidity                    pH
##          -0.23858436          -0.43851962
```

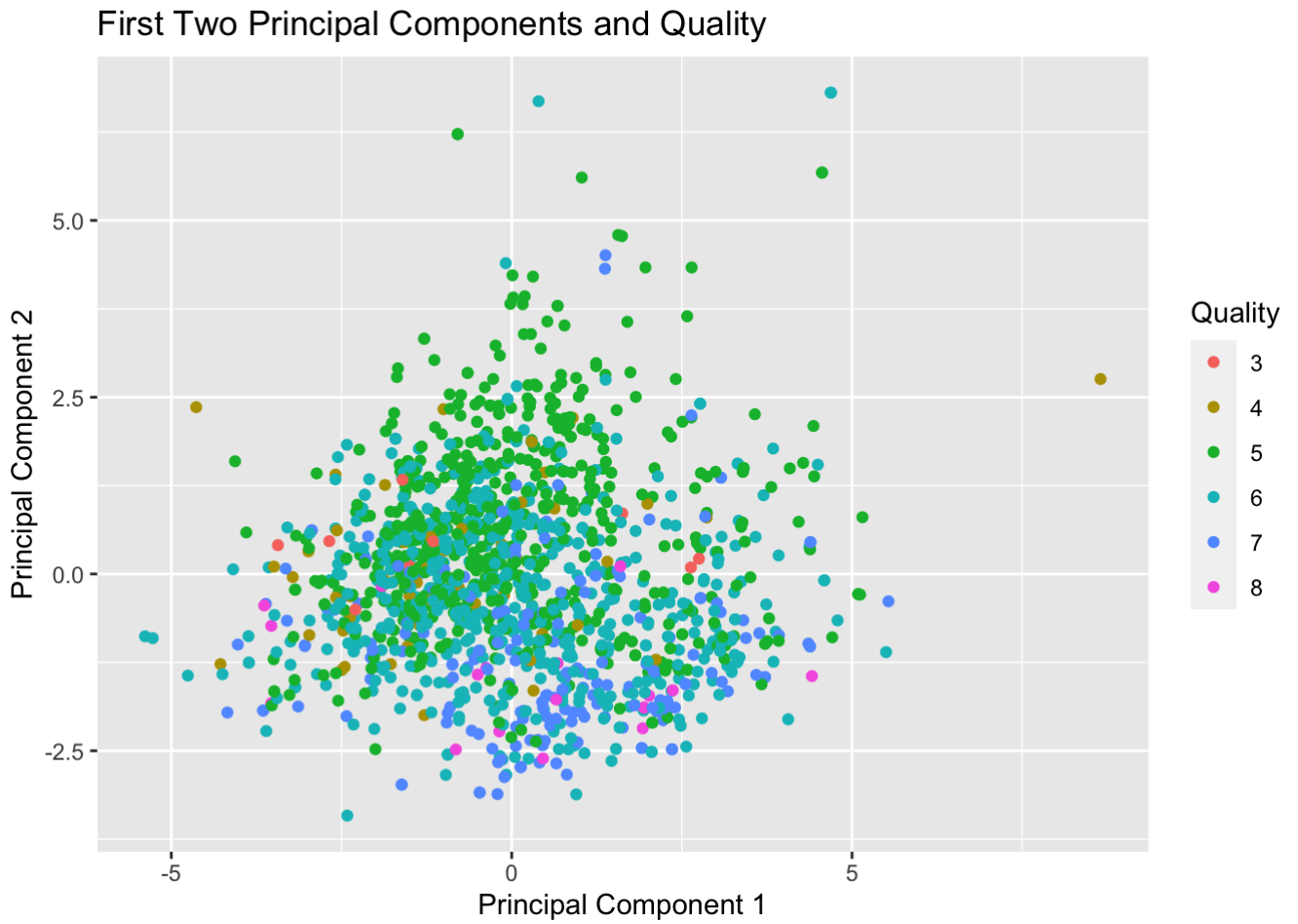
```
pr.out$rotation[,1] %>% abs() %>% sort(decreasing=TRUE)
```

```
##        fixed.acidity          citric.acid                   pH
##           0.48931422           0.46363166           0.43851962
##              density            sulphates     volatile.acidity
##           0.39535301           0.24292133           0.23858436
##            chlorides       residual.sugar              alcohol
##           0.21224658           0.14610715           0.11323206
##  free.sulfur.dioxide total.sulfur.dioxide
##           0.03615752           0.02357485
```

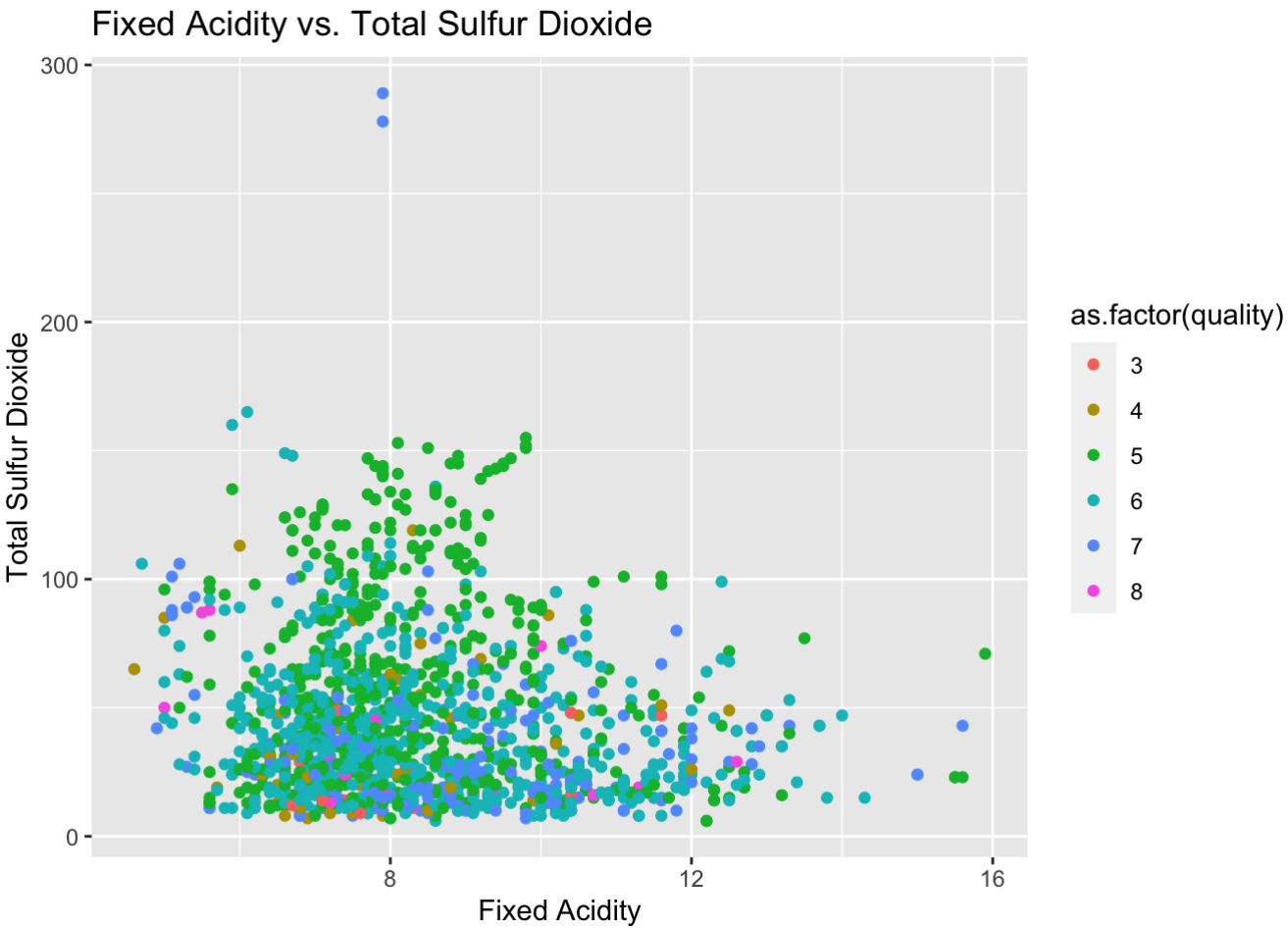
```
## second PC
## total sulfur dioxide, free sulfur dioxide, volatile acidity, residual sugar all highl
y weighted
pr.out$rotation[,2] %>% abs() %>% sort(decreasing = TRUE)
```

```
## total.sulfur.dioxide  free.sulfur.dioxide              alcohol
##          0.569486959          0.513566812          0.386180959
##     volatile.acidity       residual.sugar              density
##          0.274930480          0.272080238          0.233575490
##          citric.acid            chlorides        fixed.acidity
##          0.151791356          0.148051555          0.110502738
##            sulphates                   pH
##          0.037553916          0.006710793
```

```
# plot along first two principal components
# very slight positive relationship, but clusters are ambiguous
toplot <- data.frame(pr.out$x)
quality <- red$quality
levels(quality) <- c(levels(quality))
ggplot(toplot, mapping = aes(toplot[,1], toplot[,2], col = as.factor(quality))) +
  geom_point() +
  labs(x = "Principal Component 1",  y = "Principal Component 2", title = "First Two Pri
ncipal Components and Quality", color = "Quality")
```




```
# plot most important original variables
# most important for PC1: fixed acidity
# most important for PC2: total sulfur dioxide
# little to no relationship
ggplot(data = red, mapping = aes(x = fixed.acidity, y = total.sulfur.dioxide, color = a
s.factor(quality))) +
  geom_point() +
  labs(x = "Fixed Acidity", y = "Total Sulfur Dioxide", title = "Fixed Acidity vs. Total
Sulfur Dioxide")
```

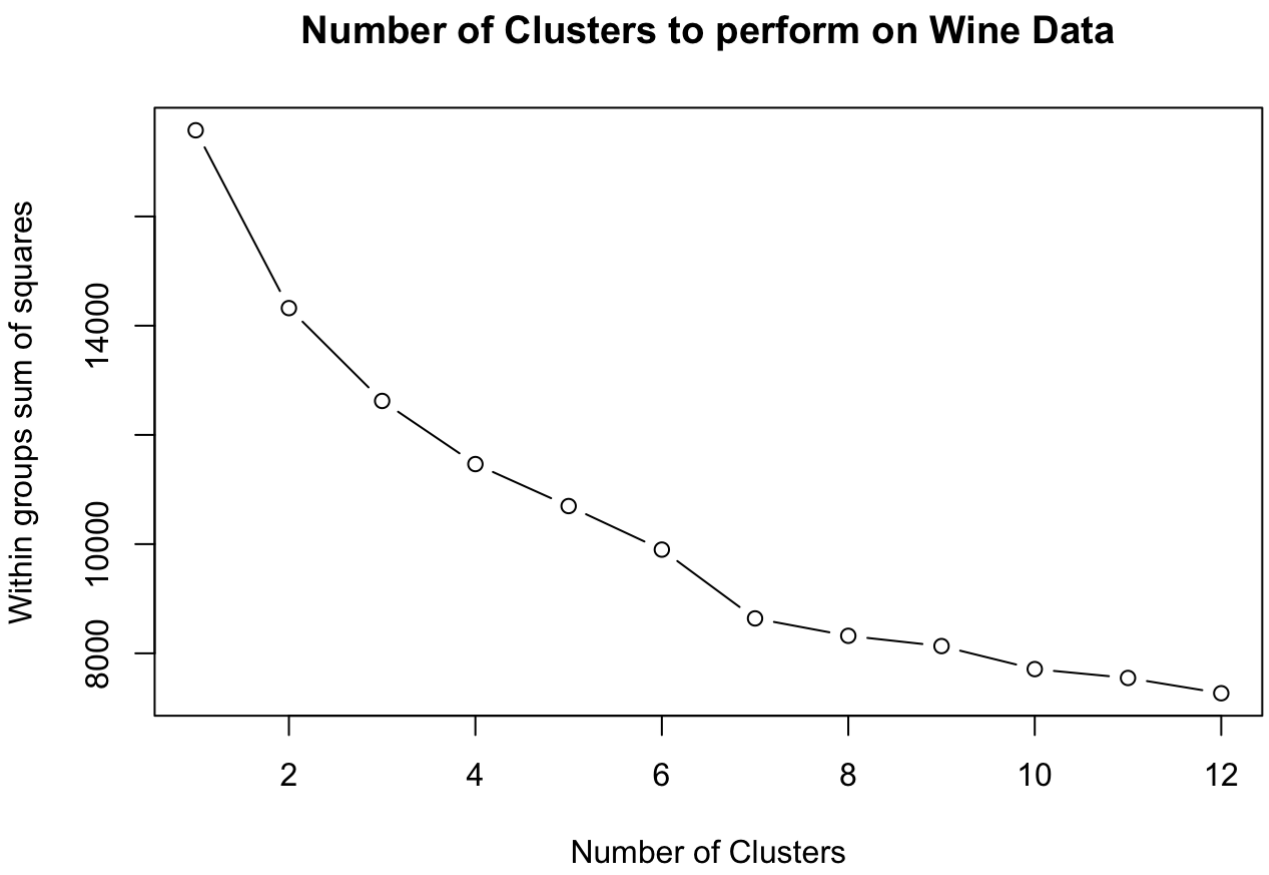


K-means clustering: Red Wine omitting NAs and deselecting quality predictor variable and scaling the data frame

```
red_data = red %>% na.omit() %>% select(-c(quality))
scaled <- scale(red_data)
```

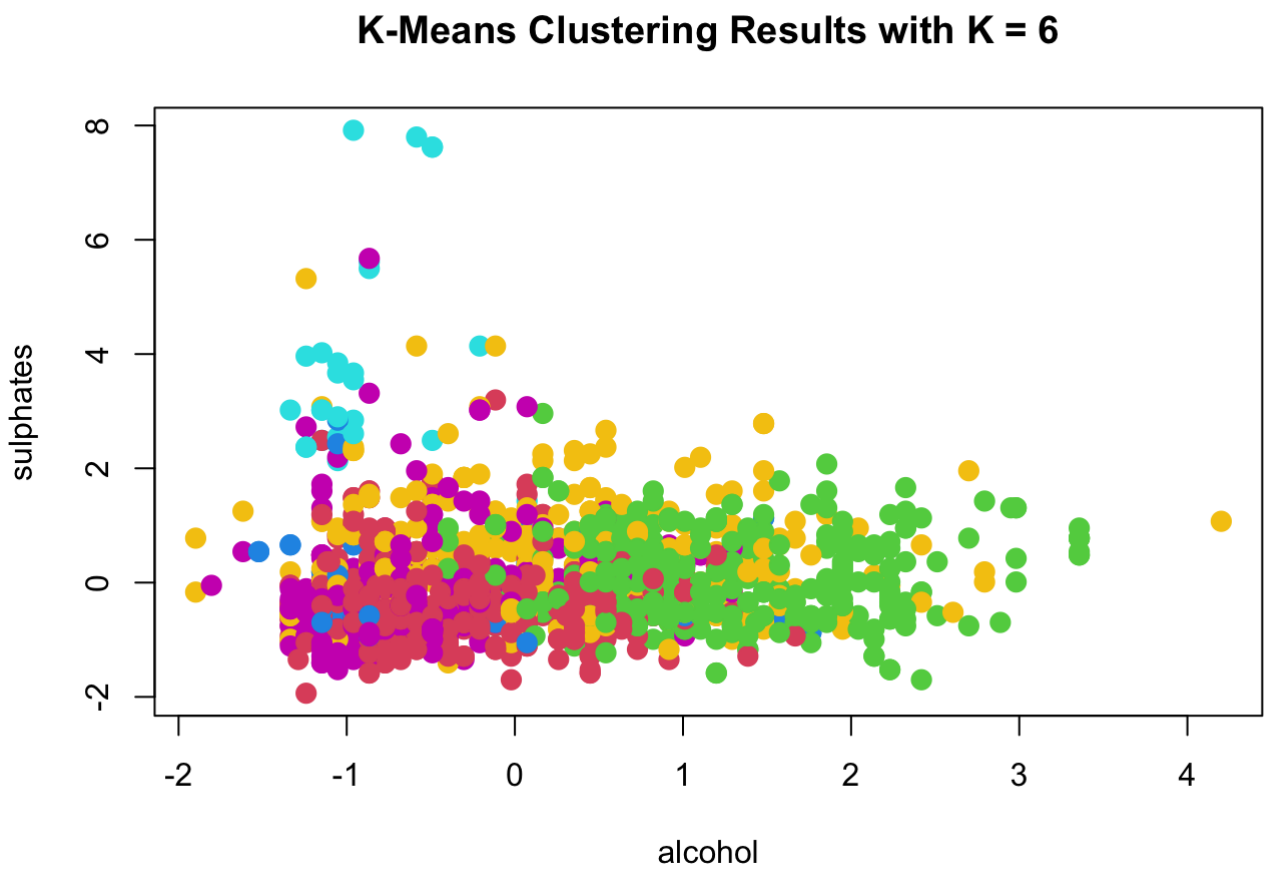
Check to see how many clusters we should perform:

```
wss<-(nrow(scaled)-1)*sum(apply(scaled,2,var))
for(i in 1:12) wss[i]<-sum(kmeans(scaled,centers=i)$withinss)
plot(1:12,wss,type='b',xlab="Number of Clusters",ylab='Within groups sum of squares', ma
in = "Number of Clusters to perform on Wine Data")
```



Clustering with the variables alcohol and sulphates:

```
set.seed(2)
km.out <- kmeans(scaled, 6, nstart = 20)
plot(scaled[,11],scaled[,10],col = (km.out$cluster + 1),
    main = "K-Means Clustering Results with K = 6",
    xlab = "alcohol", ylab = "sulphates", pch = 20, cex = 2)
```



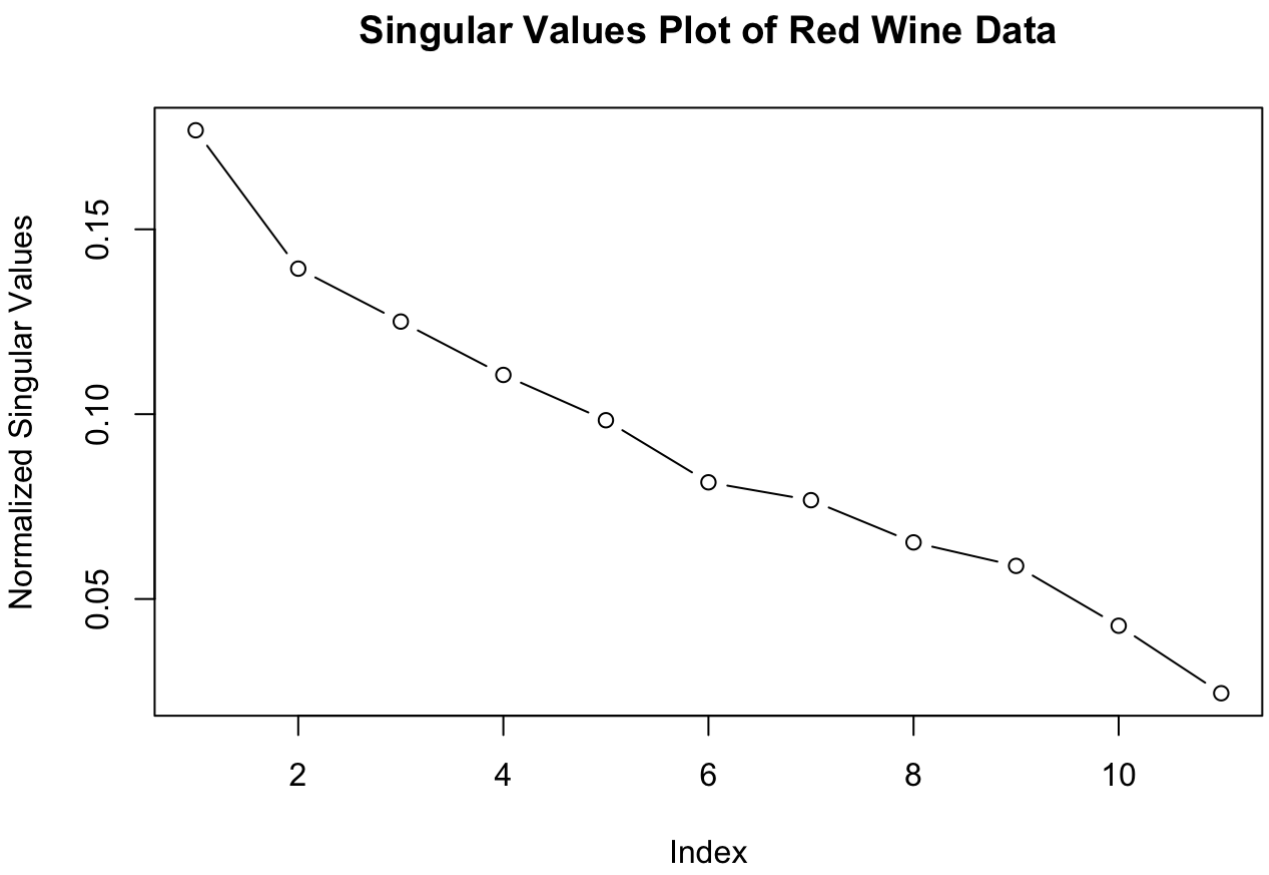

```
km.out$tot.withinss
```

```
## [1] 9358.224
```

The total within group sum of squares is 9358.224

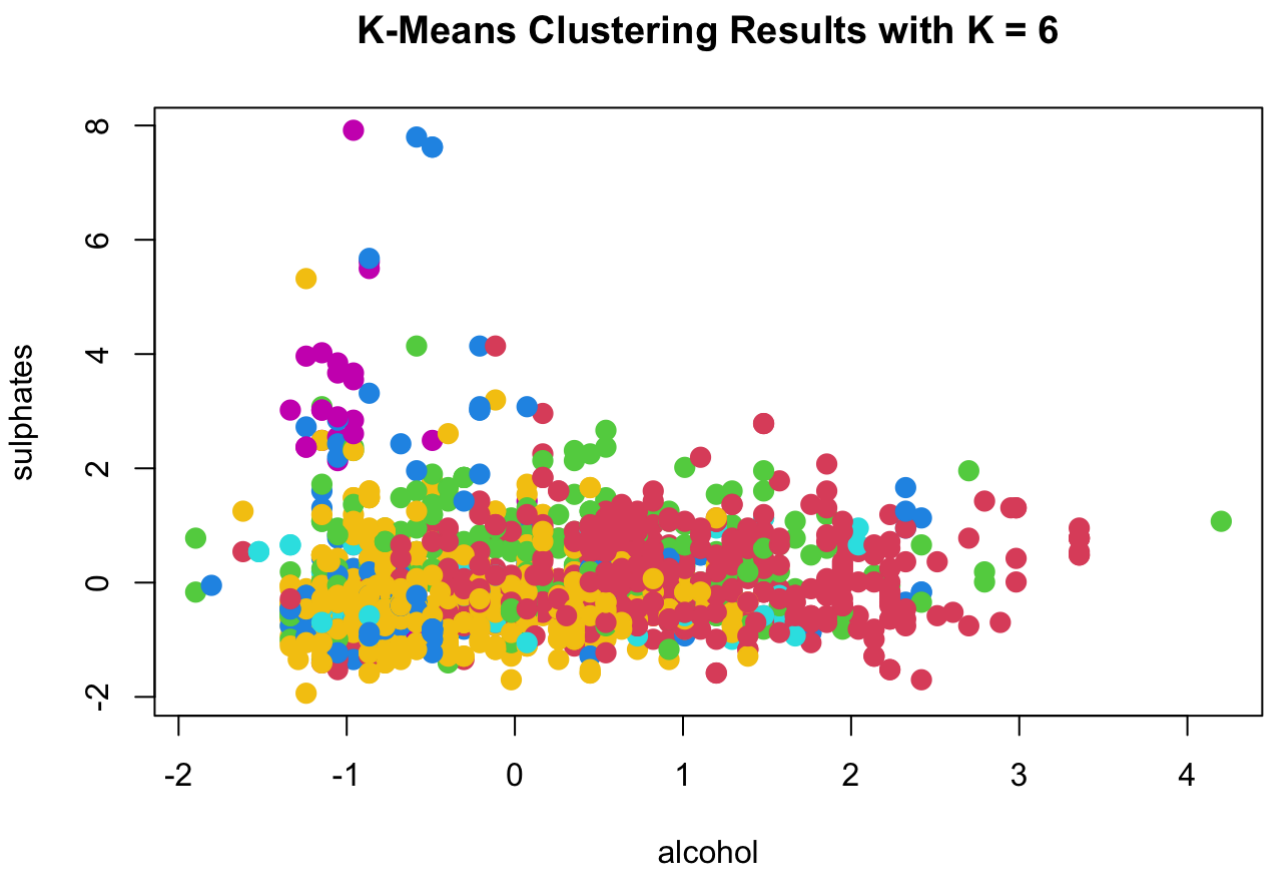
Plotting the proportion of variance explained using the d vector.

```
S <- svd(scaled)
plot(S$d/sum(S$d), type ="b",ylab = "Normalized Singular Values",
     main = "Singular Values Plot of Red Wine Data")
```



Perform k-means on the transformed locations of the data in the singular vector space:

```
set.seed(1000)
sv.km <- kmeans(S$u, 6, nstart = 20)
plot(scaled[,11],scaled[,10],col = (sv.km$cluster + 1),
     main = "K-Means Clustering Results with K = 6",
    xlab = "alcohol", ylab = "sulphates", pch = 20, cex = 2)
```




```
sv.km$tot.withinss
```

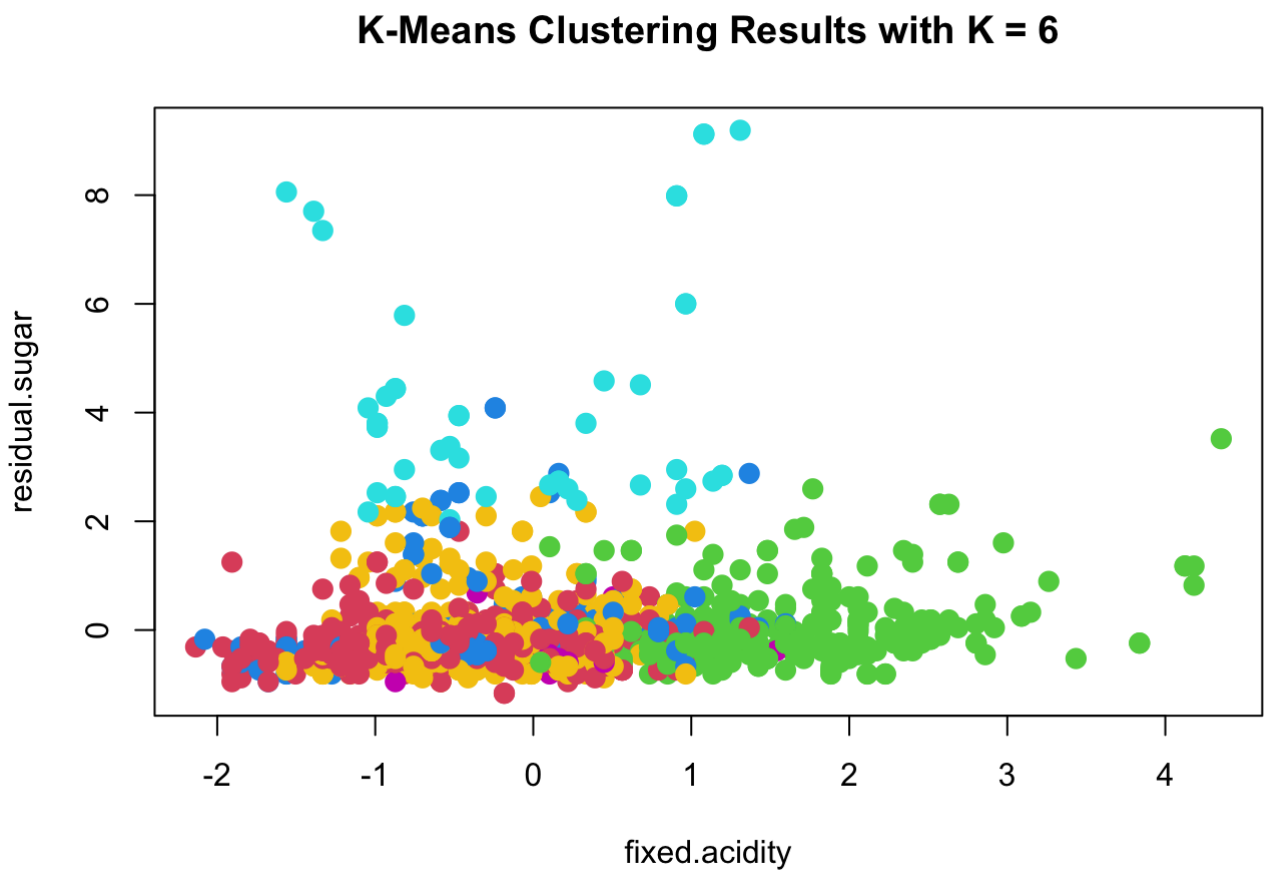
```
## [1] 7.856126
```

```
apply(scaled,2,function(x)tapply(x,sv.km$cluster,mean))
```

```
##   fixed.acidity volatile.acidity citric.acid residual.sugar   chlorides
## 1   -0.51104749      -0.49031825 -0.04737158    -0.25803125 -0.31995491
## 2    1.53613344      -0.62538807  1.23198527     0.03939870 -0.01052949
## 3   -0.23012613       0.40731553  0.01092313     0.03927543 -0.01943719
## 4   -0.08993977      -0.05278099  0.27398308     4.26964348  0.18223482
## 5    0.03237198       0.04121244  1.27835183    -0.41335833  6.52058832
## 6   -0.36399409       0.52749448 -0.71340510    -0.17610847 -0.05540507
##   free.sulfur.dioxide total.sulfur.dioxide     density         pH   sulphates
## 1          0.54820415          -0.07051866 -0.88234062  0.4223005  0.10213339
## 2         -0.45432790          -0.47315437  0.88009358 -0.8135238  0.27944151
## 3          0.47162465           1.71907254  0.10814021 -0.1339636 -0.06917962
## 4          0.99706209           0.56865591  0.89904549 -0.1367555 -0.17632297
## 5         -0.07599521           0.15844828  0.12637977 -1.5306601  2.97953015
## 6         -0.41533163          -0.44113705  0.03674427  0.2680163 -0.31190514
##       alcohol
## 1  0.84218208
## 2  0.15753233
## 3 -0.53012510
## 4 -0.01340705
## 5 -0.92616415
## 6 -0.43389231
```

Clustering with variables fixed acidity and residual sugar:

```
set.seed(1000)
sv.km <- kmeans(S$u, 6, nstart = 20)
plot(scaled[,1],scaled[,4],col = (sv.km$cluster + 1),
     main = "K-Means Clustering Results with K = 6",
    xlab = "fixed.acidity", ylab = "residual.sugar", pch = 20, cex = 2)
```




```
sv.km$tot.withinss
```

```
## [1] 7.856126
```

```
#apply(scaled,2,function(x)tapply(x,sv.km$cluster,mean))
```

Create a confusion matrix between the VD-based clusters and the wine quality:

```
confusion_matrix <- table(sv.km$cluster, red$quality)
confusion_matrix
```

```
##
##      3   4   5   6   7   8
##   1  0   4  81 222  95  10
##   2  3   6  91 140  65   5
##   3  0   6 174  45   6   1
##   4  0   3  19  13  10   1
##   5  1   1  16   6   1   0
##   6  6  33 300 212  22   1
```

```
sv.km
```

```
## K-means clustering with 6 clusters of sizes 412, 310, 232, 46, 25, 574
##
## Cluster means:
##           [,1]          [,2]         [,3]          [,4]         [,5]
## 1 -0.0126001031 -0.0086463289 -0.024857877  0.0011295422  0.002778896
## 2  0.0317674054 -0.0161066211  0.003004939 -0.0113853976  0.002376727
## 3 -0.0004960765  0.0287771497 -0.005950228 -0.0003436318  0.013956899
## 4  0.0157668381  0.0396437274 -0.015234884 -0.0406642106 -0.074723790
## 5  0.0494306883  0.0177166938  0.001598970  0.1391843001 -0.029929572
## 6 -0.0113285805 -0.0006750653  0.019775606  0.0026738164 -0.001627452
##           [,6]          [,7]         [,8]         [,9]        [,10]
## 1 -0.0001856465 -9.886014e-05 -0.003865660  0.011561834 -0.005292541
## 2  0.0013602085  9.024503e-03 -0.010480597  0.001864801 -0.000211138
## 3 -0.0071161866  5.168133e-03  0.003781471 -0.034629340  0.010024045
## 4 -0.0030564100 -3.801315e-02  0.028351717  0.012163319 -0.019811287
## 5 -0.0423467053 -4.511143e-02 -0.029929544  0.006168310  0.007746866
## 6  0.0043641804 -1.880630e-03  0.005937971  0.003447251  0.001111583
##           [,11]
## 1 -0.0036434407
## 2  0.0086873606
## 3  0.0008168287
## 4  0.0144964972
## 5  0.0123242004
## 6 -0.0041052829
##
## Clustering vector:
##    [1] 6 6 6 2 6 6 6 6 6 3 6 3 6 6 3 3 1 5 6 5 1 1 6 3 6 6 6 6 6 6 3 6 3 4 6 6 6
##   [38] 6 6 3 3 6 5 6 6 1 3 2 6 3 6 6 6 3 1 6 2 1 6 6 2 3 6 6 6 6 6 6 1 6 6 3 3 6
##   [75] 2 2 2 6 3 3 6 5 3 5 1 6 3 6 3 6 3 3 3 6 1 3 6 6 6 6 6 6 3 6 3 5 3 3 3 6
##  [112] 3 3 2 6 3 6 6 6 3 3 6 6 6 3 6 6 6 6 6 3 3 3 6 6 6 6 3 3 6 6 1 6 1 3 3 3
##  [149] 6 2 1 5 3 3 3 3 3 3 6 3 6 6 6 4 4 3 3 6 6 5 6 6 6 1 6 6 6 6 6 6 6 5 6 3 3
##  [186] 3 3 6 3 3 3 1 3 6 6 3 6 2 3 6 2 3 6 6 6 2 2 3 3 2 2 3 2 1 6 3 6 6 6 3 1 3
##  [223] 6 6 6 1 5 6 1 6 3 6 6 6 6 6 6 6 6 6 5 2 3 2 2 6 6 3 6 6 2 6 2 3 6 3 6 6 5
##  [260] 2 6 3 6 1 2 2 6 1 6 2 1 2 2 6 4 1 6 2 4 6 2 5 6 6 6 6 2 1 6 2 6 5 2 6 2 2
##  [297] 2 6 6 6 6 2 6 6 3 2 3 2 2 6 2 3 3 3 1 1 1 6 2 6 2 3 6 2 4 4 2 2 2 2 2 2 3
##  [334] 6 6 2 1 1 2 2 2 2 2 2 2 1 1 2 6 6 2 6 6 2 3 1 2 2 2 2 3 6 2 2 2 2 2 2 2 1
##  [371] 6 6 1 3 2 2 2 1 2 1 6 2 6 6 6 6 6 6 6 6 3 2 2 3 2 2 4 2 2 6 4 1 2 2 6 2 2
##  [408] 2 2 2 3 3 3 4 3 3 2 3 2 6 1 1 3 2 3 1 1 6 6 2 2 6 2 2 2 2 1 2 2 6 2 2 2 2
##  [445] 1 6 2 6 6 2 2 5 6 2 1 2 6 6 2 2 2 6 2 3 2 2 2 1 2 1 2 1 2 2 2 6 2 2 6 6 4
##  [482] 2 2 2 2 2 2 2 2 2 6 1 1 3 4 2 6 1 2 3 6 4 4 2 2 2 2 2 2 6 2 2 6 2 2 2 3 2 2
##  [519] 2 1 2 6 3 3 3 2 1 1 1 6 6 2 2 1 6 6 6 6 2 2 6 4 6 2 2 3 6 2 2 2 6 6 6 3 2
##  [556] 2 2 2 2 2 2 3 3 1 2 2 6 6 2 1 2 1 2 2 2 2 2 3 3 2 2 2 2 2 2 6 2 3 1 2 3 3
##  [593] 3 2 6 4 2 2 6 2 6 2 6 2 6 6 2 1 6 1 2 2 6 6 3 2 2 2 2 2 3 3 6 1 6 6 6 6 6
##  [630] 3 6 2 1 3 1 6 3 3 6 1 2 2 2 2 2 4 6 6 1 4 6 3 2 1 6 6 6 2 6 6 6 6 6 2 2 6
##  [667] 6 2 2 2 1 6 3 6 2 2 2 6 1 2 2 6 1 6 3 6 6 6 6 6 6 3 5 3 3 1 6 6 6 2 3 6 6
##  [704] 3 6 6 6 6 6 2 3 3 6 6 3 6 6 6 6 6 6 3 6 3 6 6 6 6 6 1 5 6 6 3 6 6 6 6 3 6
##  [741] 6 3 6 2 3 6 6 3 6 6 6 6 6 6 5 6 6 6 6 6 3 6 6 6 6 6 6 3 3 3 6 3 3 3 6 6 6 6
##  [778] 6 6 3 6 6 3 6 3 2 2 6 6 3 3 3 3 6 2 3 6 1 6 6 3 6 3 6 6 1 1 1 6 6 6 2 2 1
##  [815] 2 2 6 2 6 6 6 1 6 6 6 6 1 6 1 1 6 1 2 2 6 6 1 1 2 6 2 6 2 3 1 6 6 6 6 6 2
##  [852] 2 3 1 1 6 1 1 2 1 3 3 6 3 3 3 1 1 1 1 1 1 1 1 2 1 6 1 6 3 6 1 1 3 6 6 6 1
##  [889] 1 3 1 3 2 3 3 1 1 1 1 6 2 6 6 4 4 3 3 6 1 1 1 4 2 2 1 1 3 4 1 1 2 1 1 4 1
##  [926] 1 1 3 1 1 6 6 1 6 6 1 1 2 1 1 2 2 2 2 1 2 2 1 1 1 1 1 1 2 1 2 2 2 1 6 1 6
```

```
##  [963] 6 1 1 1 1 3 1 6 2 2 2 2 4 1 1 3 1 2 6 6 1 6 2 6 2 1 6 1 6 1 1 1 3 6 1 1 6
## [1000] 1 1 1 1 1 1 1 1 1 1 1 1 1 6 6 6 2 1 1 1 6 2 2 6 1 6 6 1 1 1 6 6 6 6 6 6 2
## [1037] 1 6 4 1 6 6 1 4 1 1 1 1 2 2 1 5 1 1 3 3 2 3 1 2 2 1 1 2 6 6 1 2 2 1 1 4 1
## [1074] 6 4 1 2 4 4 3 2 3 1 1 1 1 2 1 2 2 1 1 1 1 6 2 6 6 1 6 1 1 1 1 1 1 1 1 6 2
## [1111] 6 1 1 6 1 6 6 6 1 1 1 1 1 2 6 1 1 1 3 2 6 1 1 6 1 1 2 2 6 3 1 1 1 1 1 1 6
## [1148] 4 1 2 1 1 6 2 1 6 1 3 1 2 2 1 1 6 6 5 3 1 1 1 6 1 2 1 1 1 4 1 3 1 1 2 1 6
## [1185] 3 1 4 1 3 6 2 6 1 6 6 6 3 6 1 3 6 1 1 4 1 1 1 2 1 1 6 6 6 2 2 1 3 1 6 1 2
## [1222] 2 1 2 2 6 6 6 1 6 1 1 6 2 1 4 6 1 6 6 1 2 1 3 4 6 6 6 1 1 1 6 6 6 6 1 1 3
## [1259] 6 6 5 6 1 6 1 6 6 2 1 1 1 1 1 1 1 3 4 6 3 2 1 1 6 3 1 2 1 1 3 3 6 1 1 6 1
## [1296] 1 1 1 1 6 1 1 2 1 6 3 3 6 3 6 3 1 6 1 1 3 1 2 3 5 3 1 1 1 1 1 1 1 6 3 3 6
## [1333] 6 6 6 1 6 6 6 6 6 6 6 6 6 2 6 6 6 6 6 6 1 6 6 6 6 6 6 1 4 2 6 6 2 3 6 6 6 3 3
## [1370] 6 5 1 5 3 5 3 6 1 6 6 6 6 3 3 3 6 6 6 4 3 1 6 6 6 6 6 6 3 6 6 3 3 1 6 6 1
## [1407] 6 1 1 1 6 1 6 3 2 6 2 1 6 3 6 6 1 6 6 6 1 1 1 1 1 3 6 6 4 4 3 6 6 1 1 3 6
## [1444] 1 1 3 6 6 1 1 1 1 1 3 2 6 1 3 2 1 1 6 6 6 1 1 1 6 1 6 6 1 1 1 4 1 4 1 6 1
## [1481] 6 1 6 1 6 6 6 6 6 6 1 6 1 3 1 6 3 6 6 6 6 6 6 1 1 6 6 1 1 1 1 6 6 1 6 6 1
## [1518] 1 6 6 1 6 1 1 1 6 6 6 1 1 1 6 6 1 1 6 1 1 1 1 6 1 1 2 1 1 6 1 2 1 6 6 1 6
## [1555] 6 1 6 6 4 3 3 3 6 6 6 1 1 6 6 1 1 1 3 1 4 1 1 6 6 1 1 1 1 3 1 1 1 1 1 4 1
## [1592] 1 1 6 1 1 1 1 1
##
## Within cluster sum of squares by cluster:
## [1] 2.0319202 1.5191347 1.4790850 0.4636769 0.1891739 2.1731349
##  (between_SS / total_SS =  28.6 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

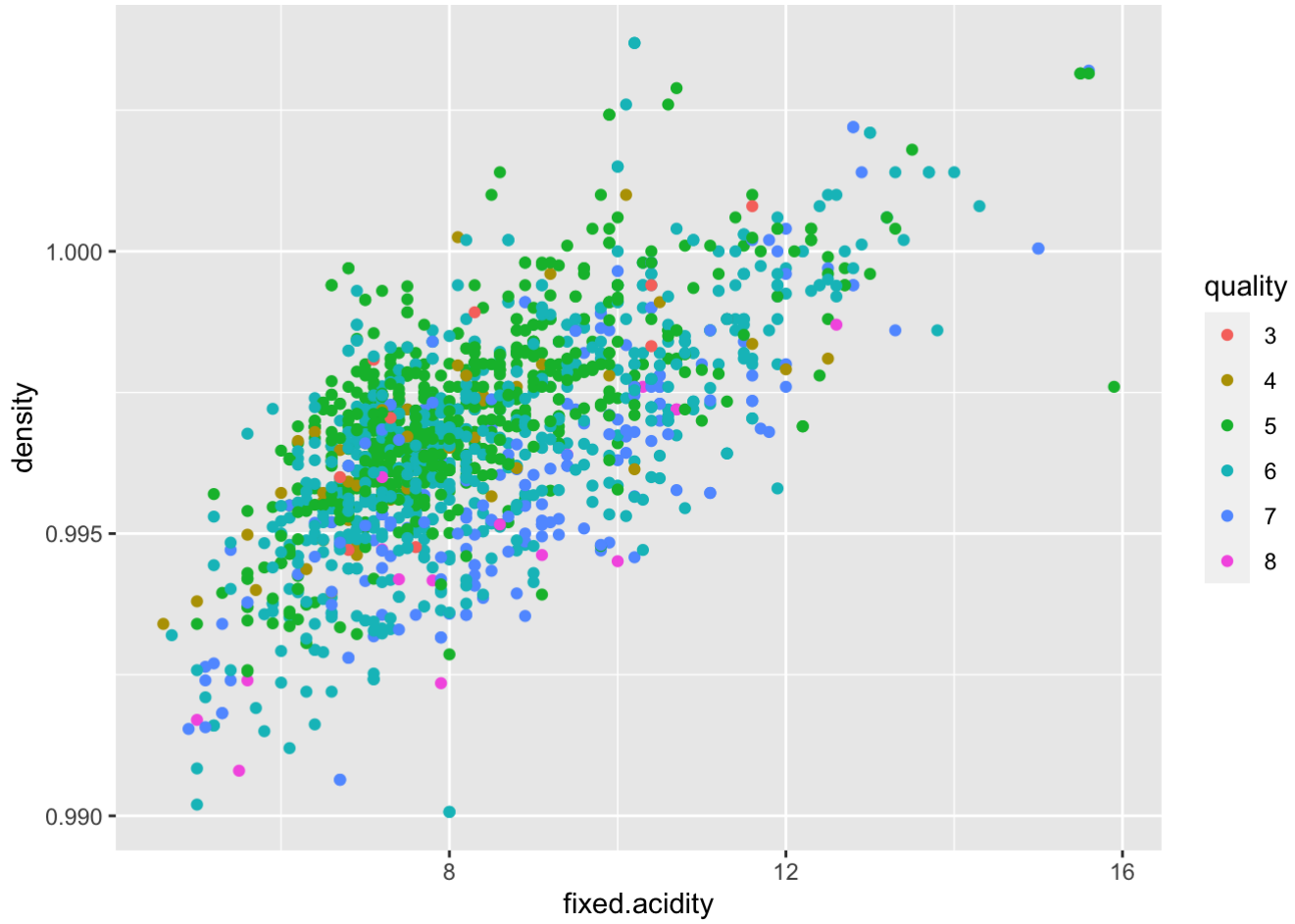
Exploratory analysis using clusters:

```
R = red %>% na.omit()
R$quality <- as.factor(R$quality)
head(R)
```

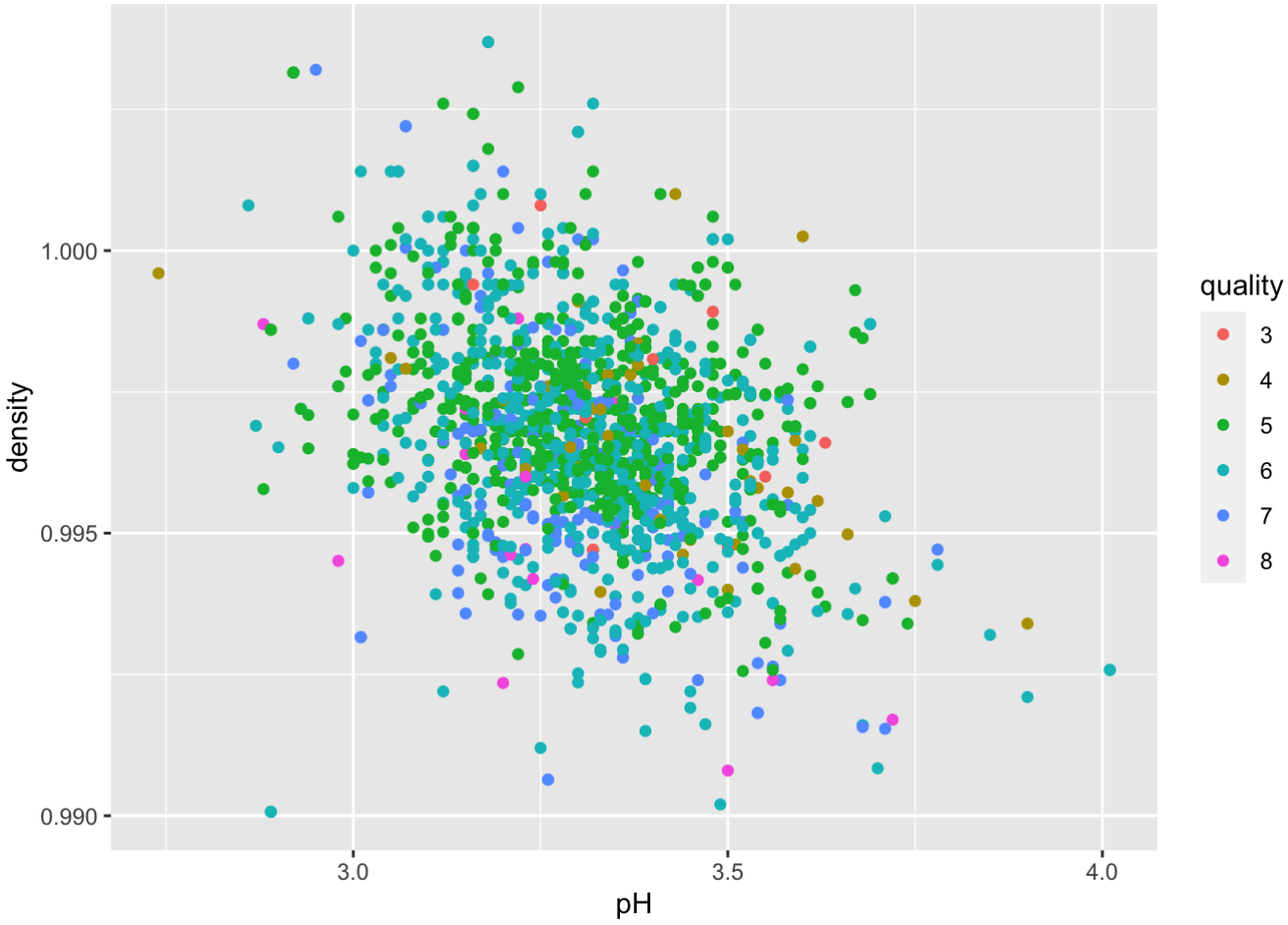
```
##   fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
## 1           7.4             0.70        0.00            1.9     0.076
## 2           7.8             0.88        0.00            2.6     0.098
## 3           7.8             0.76        0.04            2.3     0.092
## 4          11.2             0.28        0.56            1.9     0.075
## 5           7.4             0.70        0.00            1.9     0.076
## 6           7.4             0.66        0.00            1.8     0.075
##   free.sulfur.dioxide total.sulfur.dioxide density   pH sulphates alcohol
## 1                  11                   34  0.9978 3.51      0.56     9.4
## 2                  25                   67  0.9968 3.20      0.68     9.8
## 3                  15                   54  0.9970 3.26      0.65     9.8
## 4                  17                   60  0.9980 3.16      0.58     9.8
## 5                  11                   34  0.9978 3.51      0.56     9.4
## 6                  13                   40  0.9978 3.51      0.56     9.4
##   quality
## 1       5
## 2       5
## 3       5
## 4       6
## 5       5
## 6       5
```

Plotting different variables:

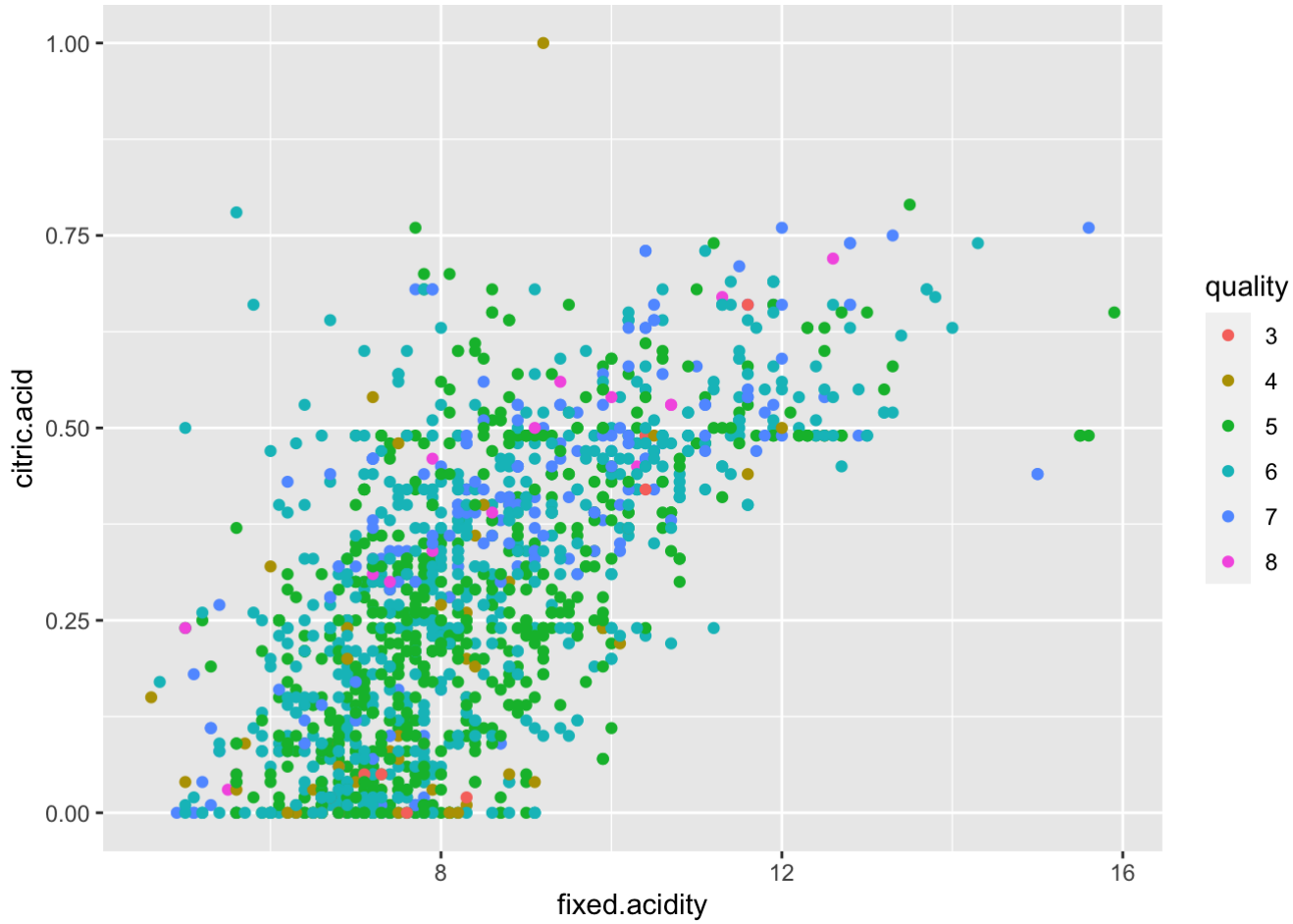
```
ggplot(R, aes(x=fixed.acidity, y=density, color=quality)) + geom_point()
```



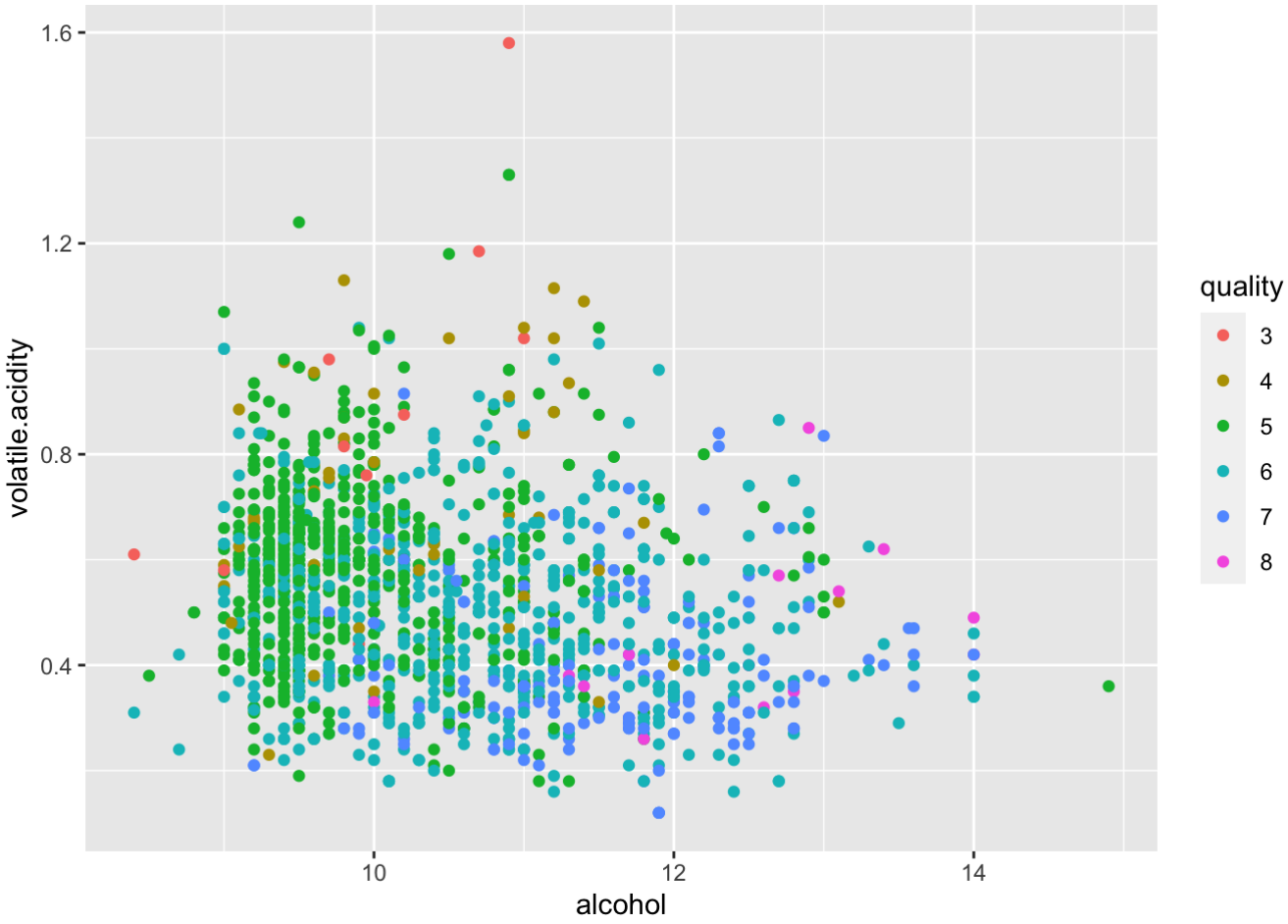
```
ggplot(R, aes(x=pH, y=density, color=quality)) + geom_point()
```



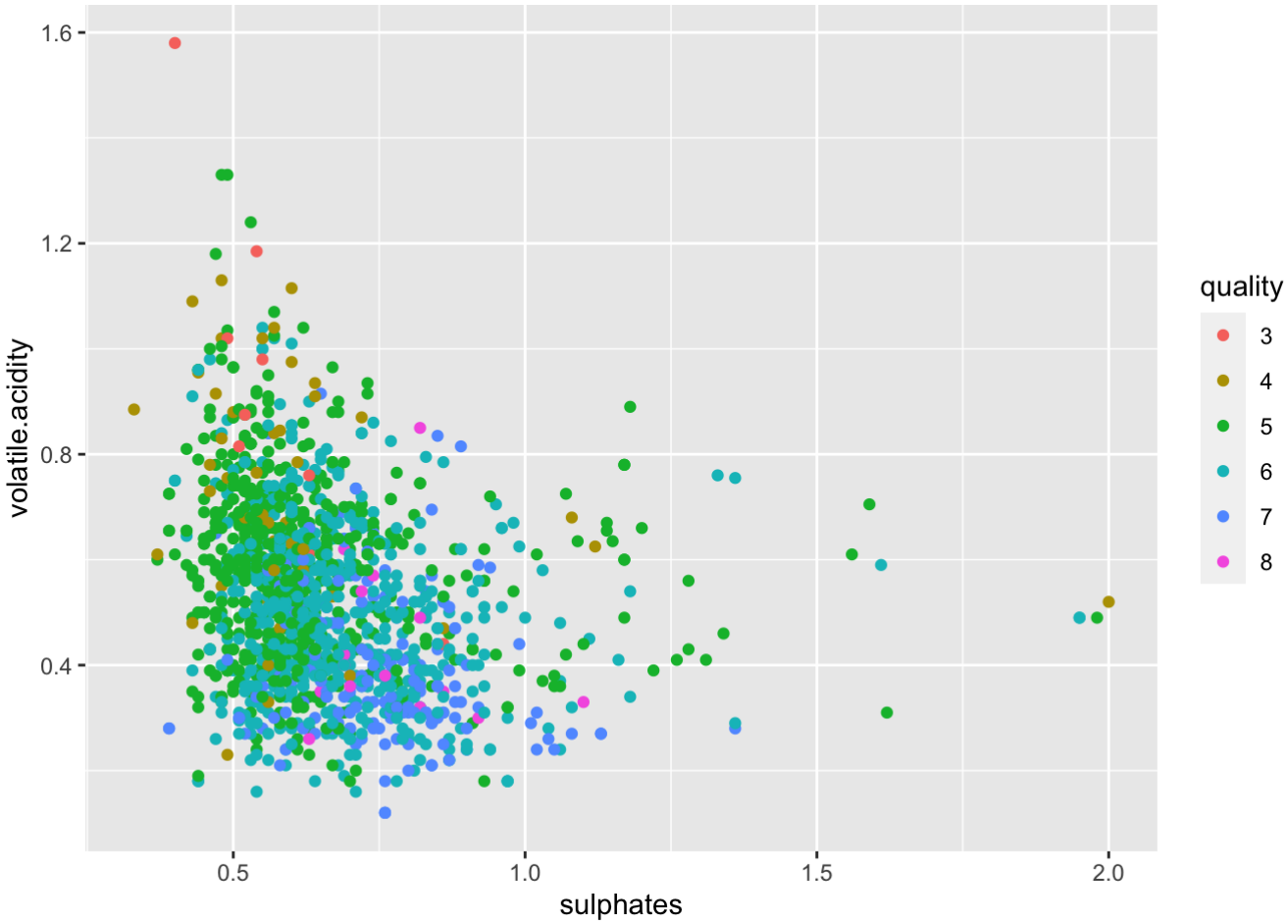

```
ggplot(R, aes(x=fixed.acidity, y=citric.acid, color=quality)) + geom_point()
```




```
ggplot(R, aes(x=alcohol, y=volatile.acidity, color=quality)) + geom_point()
```

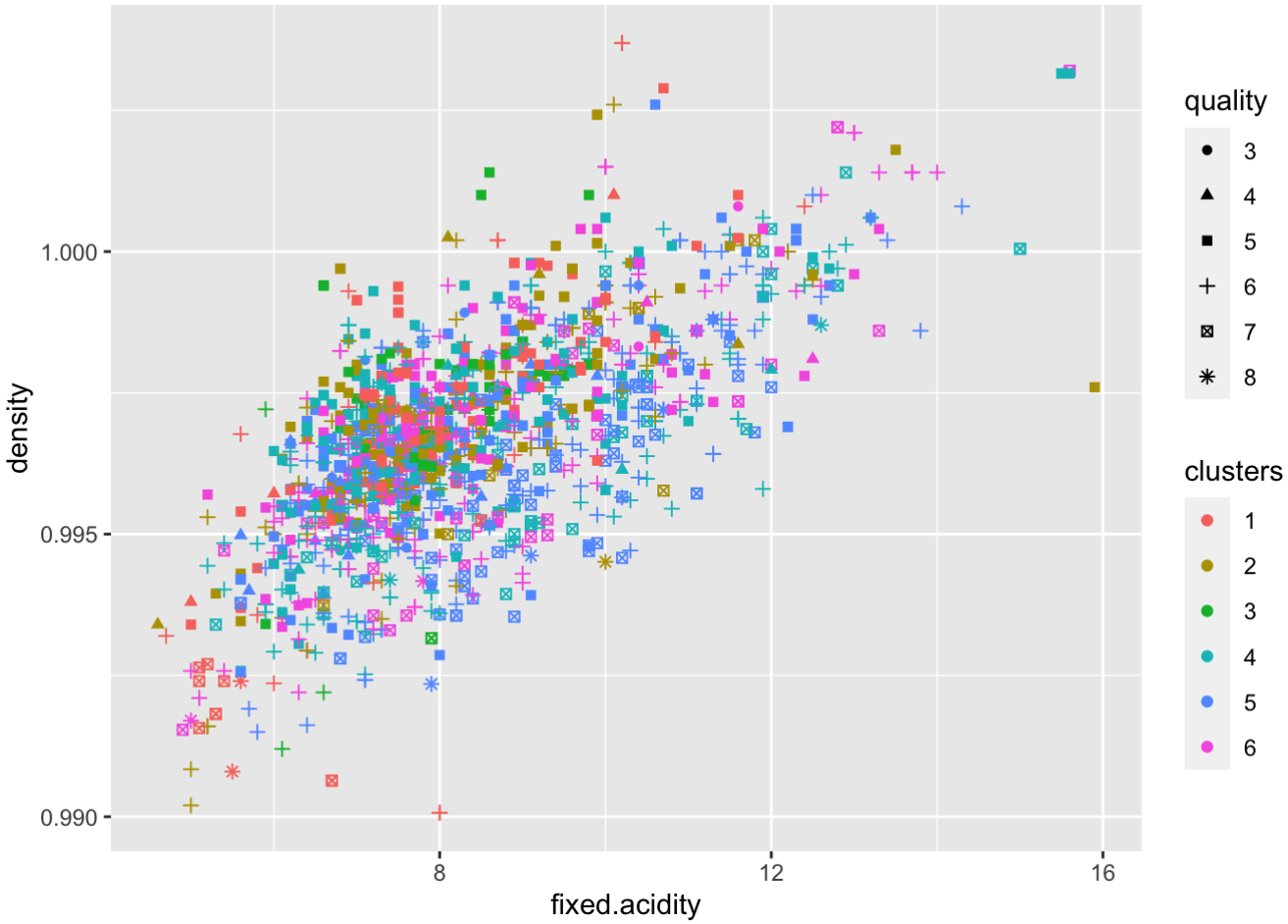



```
ggplot(R, aes(x=sulphates, y=volatile.acidity, color=quality)) + geom_point()
```

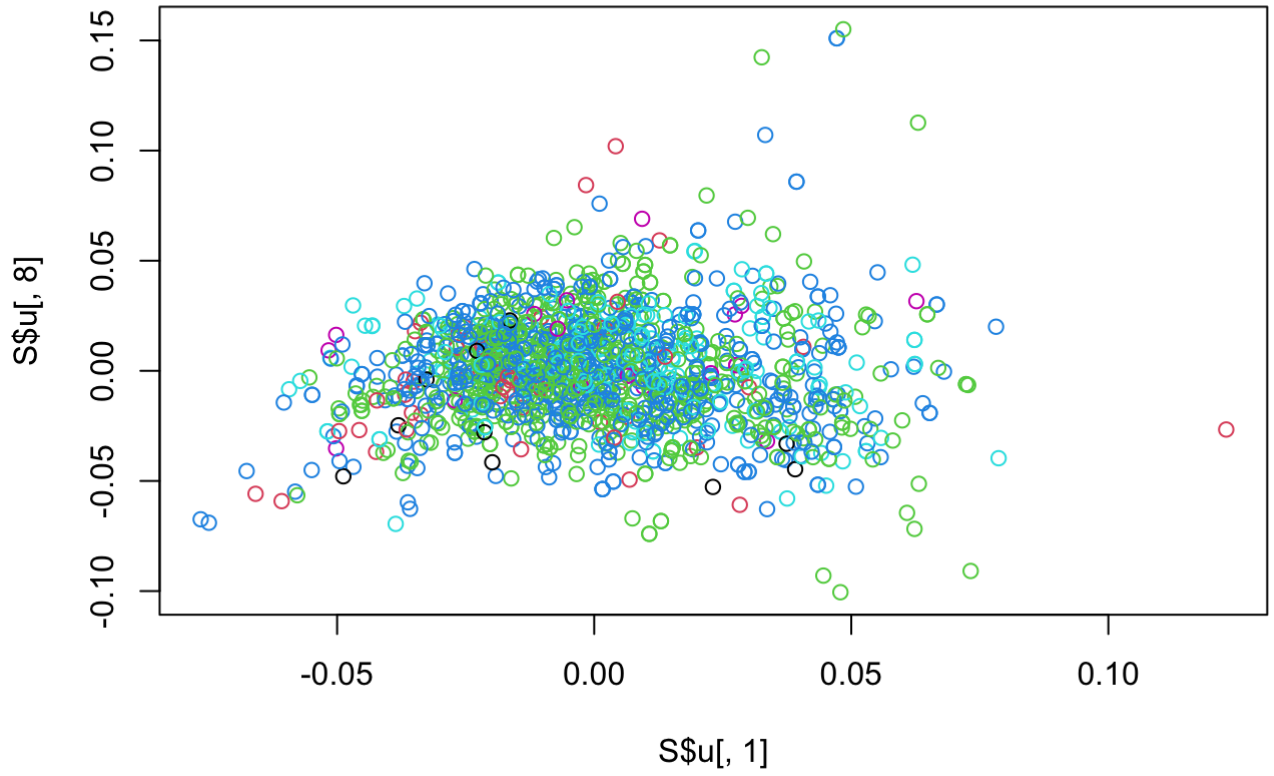


Looking at fixed acidity and density:

```
set.seed(2)
red.km <- kmeans(select(R, -c(quality)), 6, nstart = 20)
R$clusters = as.factor(red.km $cluster)
ggplot(R, aes(x=fixed.acidity, y=density, color=clusters, shape=quality)) + geom_point()
```



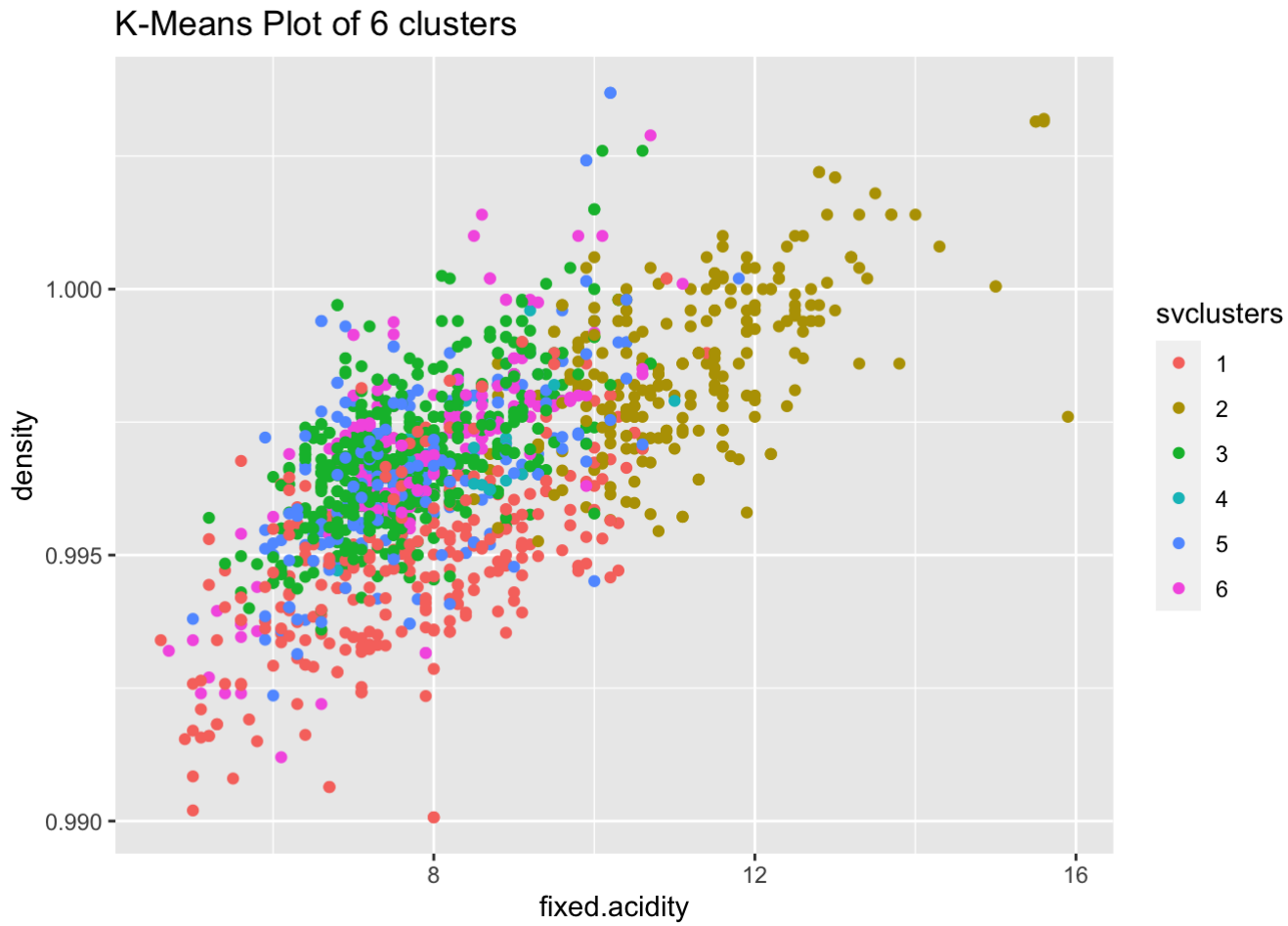

```
Rscale <-  R %>% select(-c(clusters,quality)) %>% scale()
S <- svd(Rscale)
plot(S$u[,1], S$u[,8], col=R$quality)
```



Now

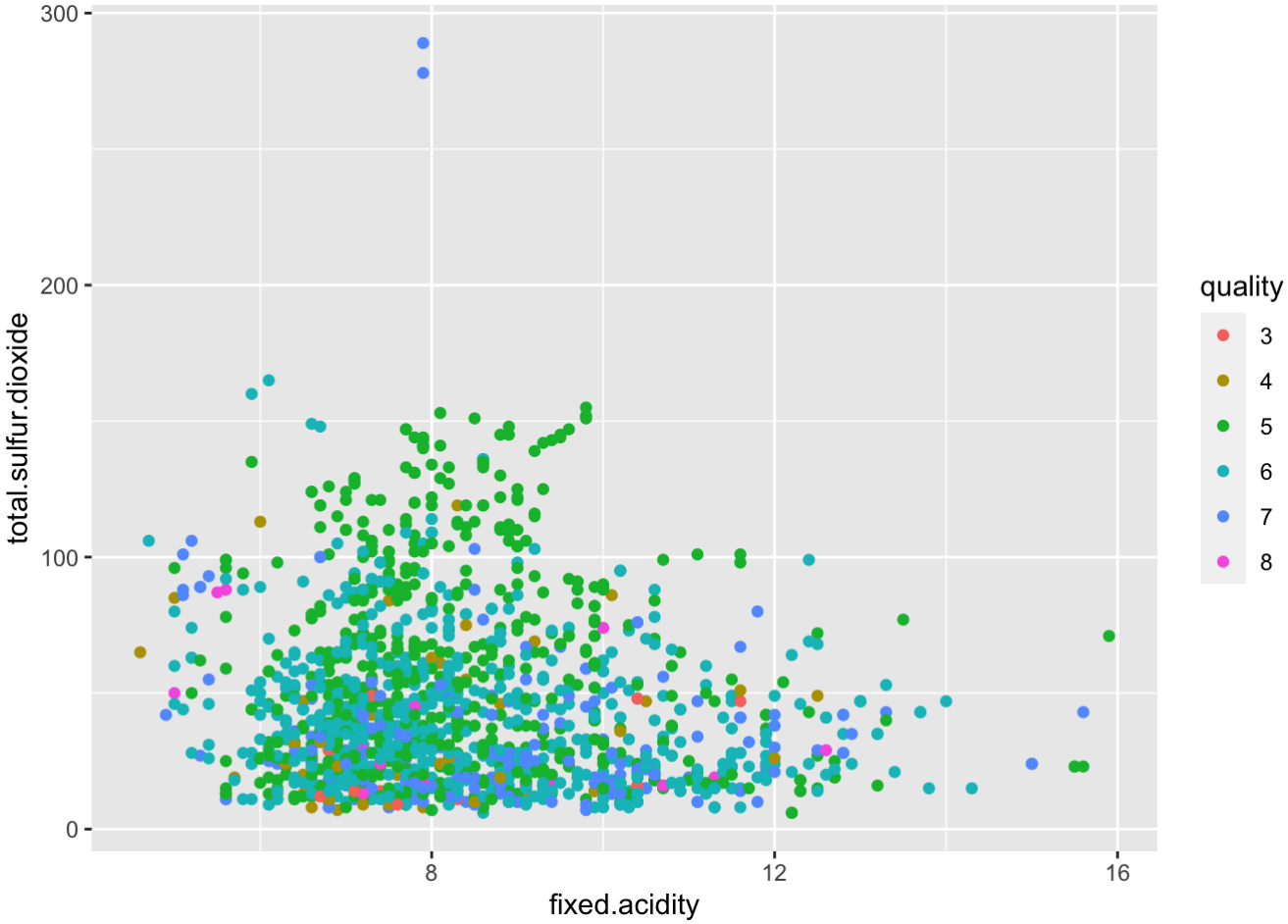
we apply k-means on the locations of the data in singular vector space. We plot the clusters, agnostic to the quality of the wine. This seems closer to the quality seperations:

```
set.seed(2)
sv.km <- kmeans(S$u, 6, nstart = 20)
R$svclusters = as.factor(sv.km$cluster)
ggplot(R, aes(x=fixed.acidity, y=density, color=svclusters)) + geom_point()+
  labs(title = "K-Means Plot of 6 clusters")
```

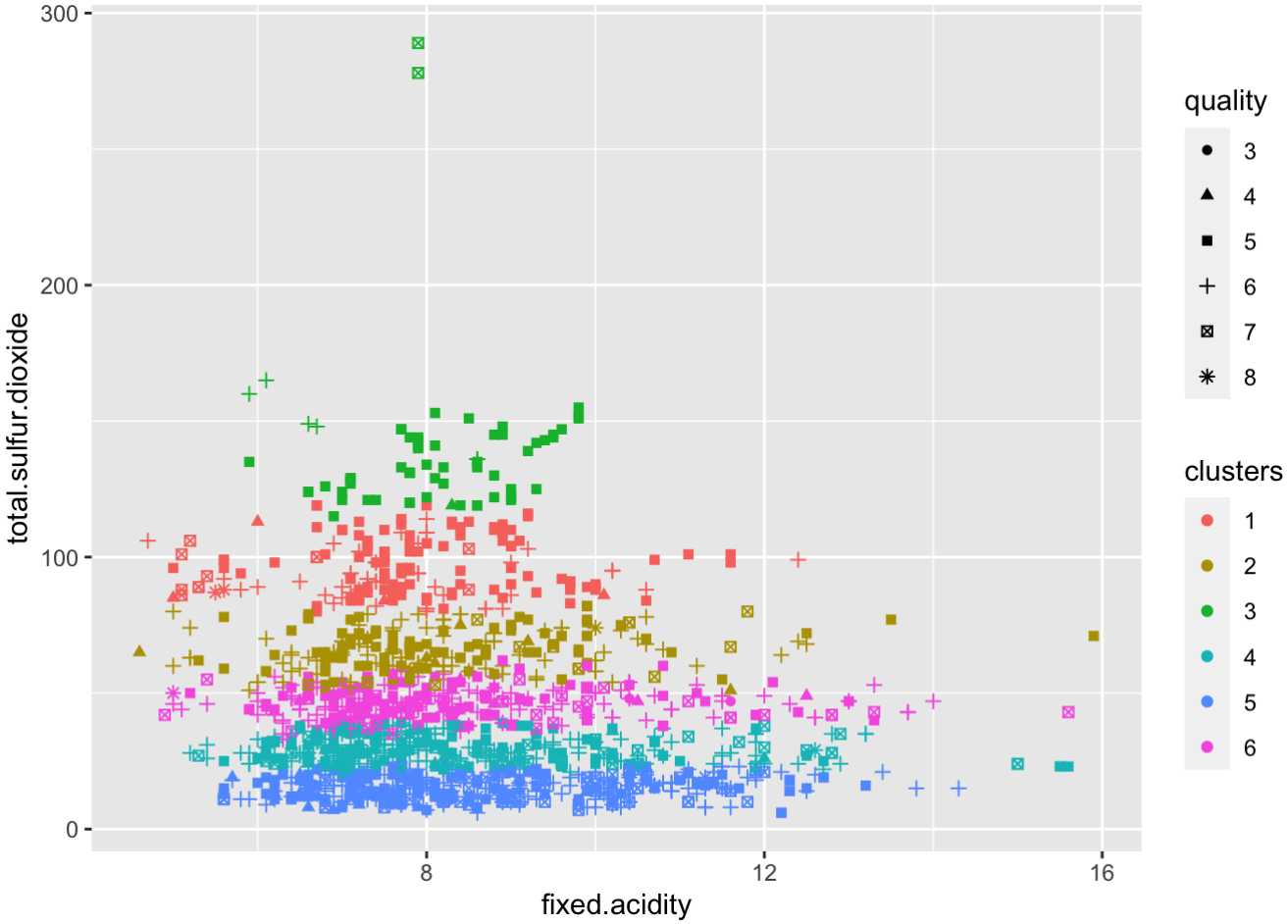


Looking at alcohol and volatile acidity:

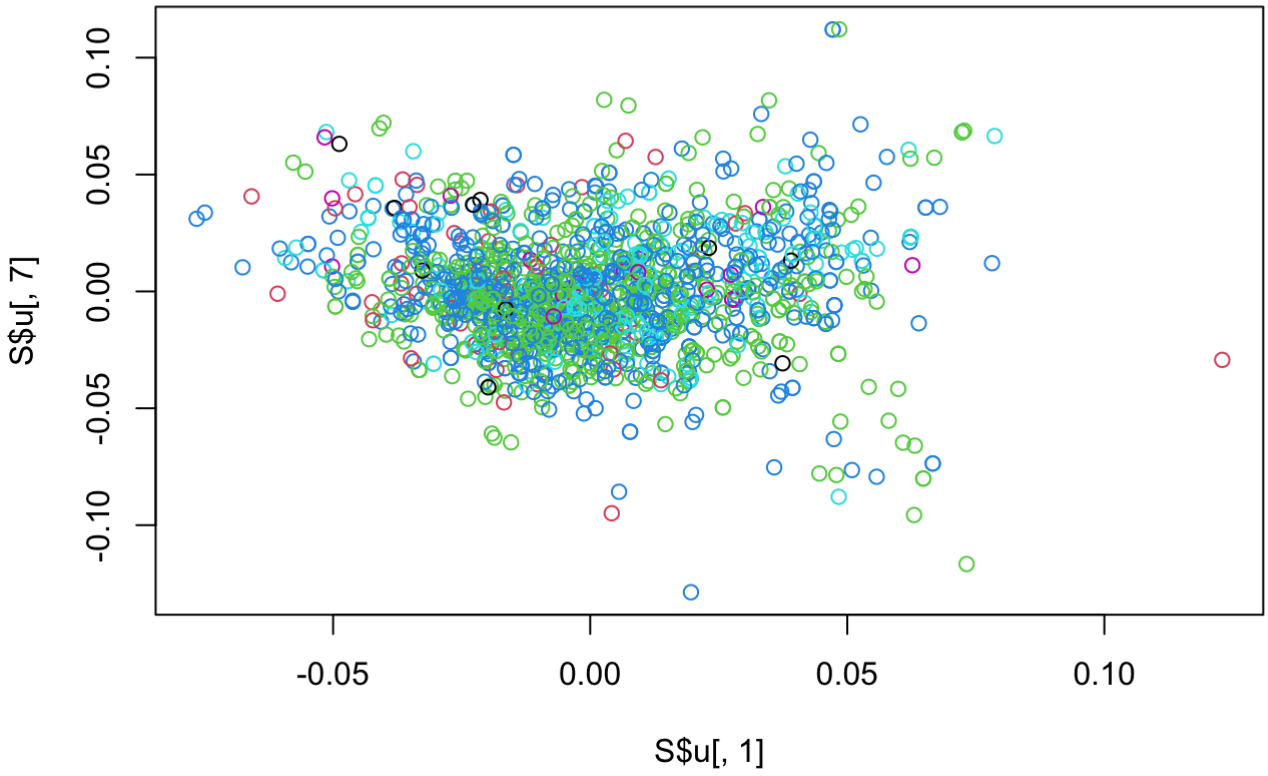
```
R = red %>% na.omit()
R$quality <- as.factor(R$quality)
ggplot(R, aes(x=fixed.acidity, y=total.sulfur.dioxide, color=quality)) + geom_point()
```



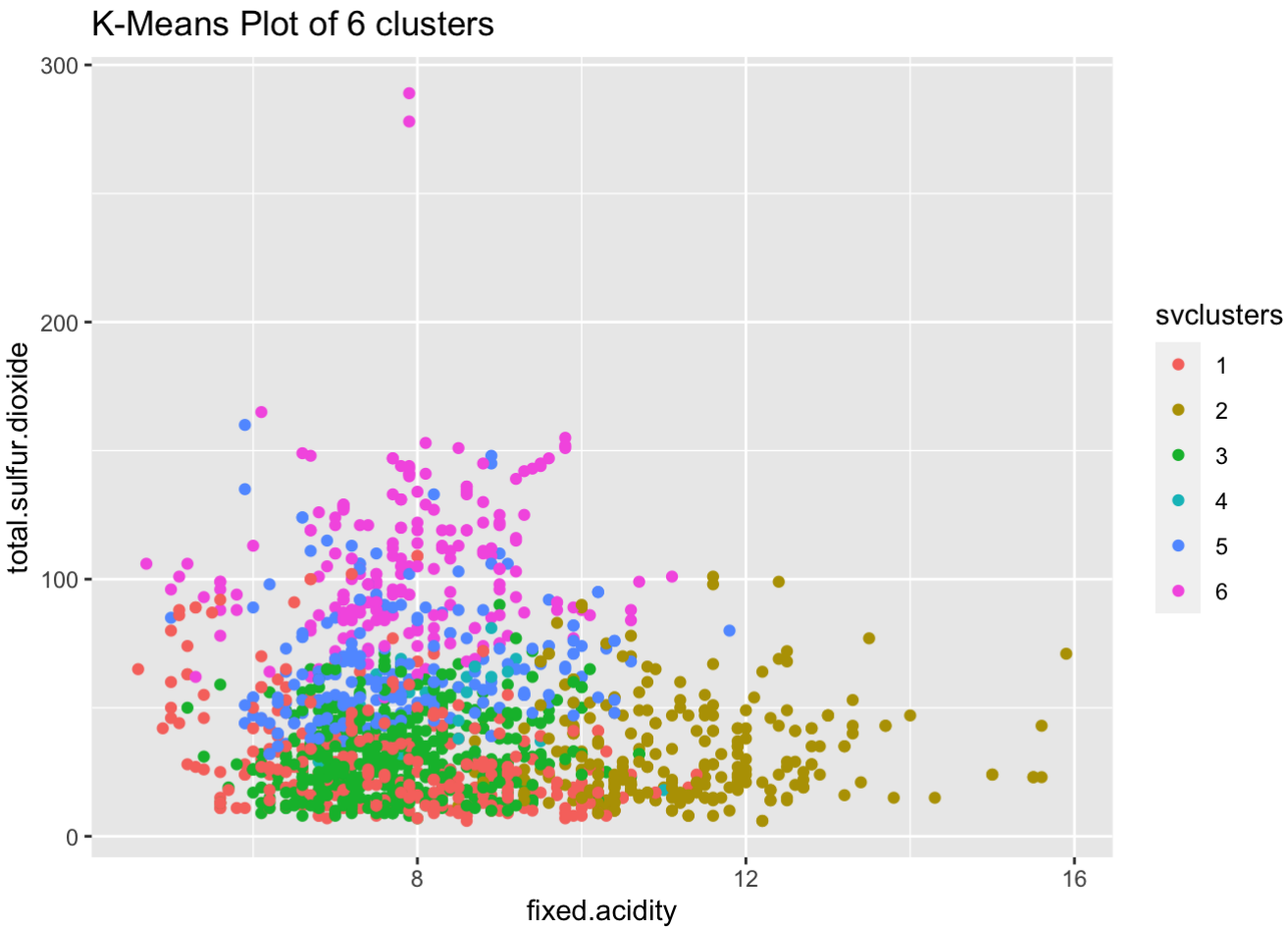

```
set.seed(2)
red.km2 <- kmeans(select(R, -c(quality)), 6, nstart = 20)
R$clusters = as.factor(red.km2 $cluster)
ggplot(R, aes(x=fixed.acidity, y=total.sulfur.dioxide, color=clusters, shape=quality)) +
geom_point()
```




```
Rscale <-  R %>% select(-c(clusters,quality)) %>% scale()
S <- svd(Rscale)
plot(S$u[,1], S$u[,7], col=R$quality)
```

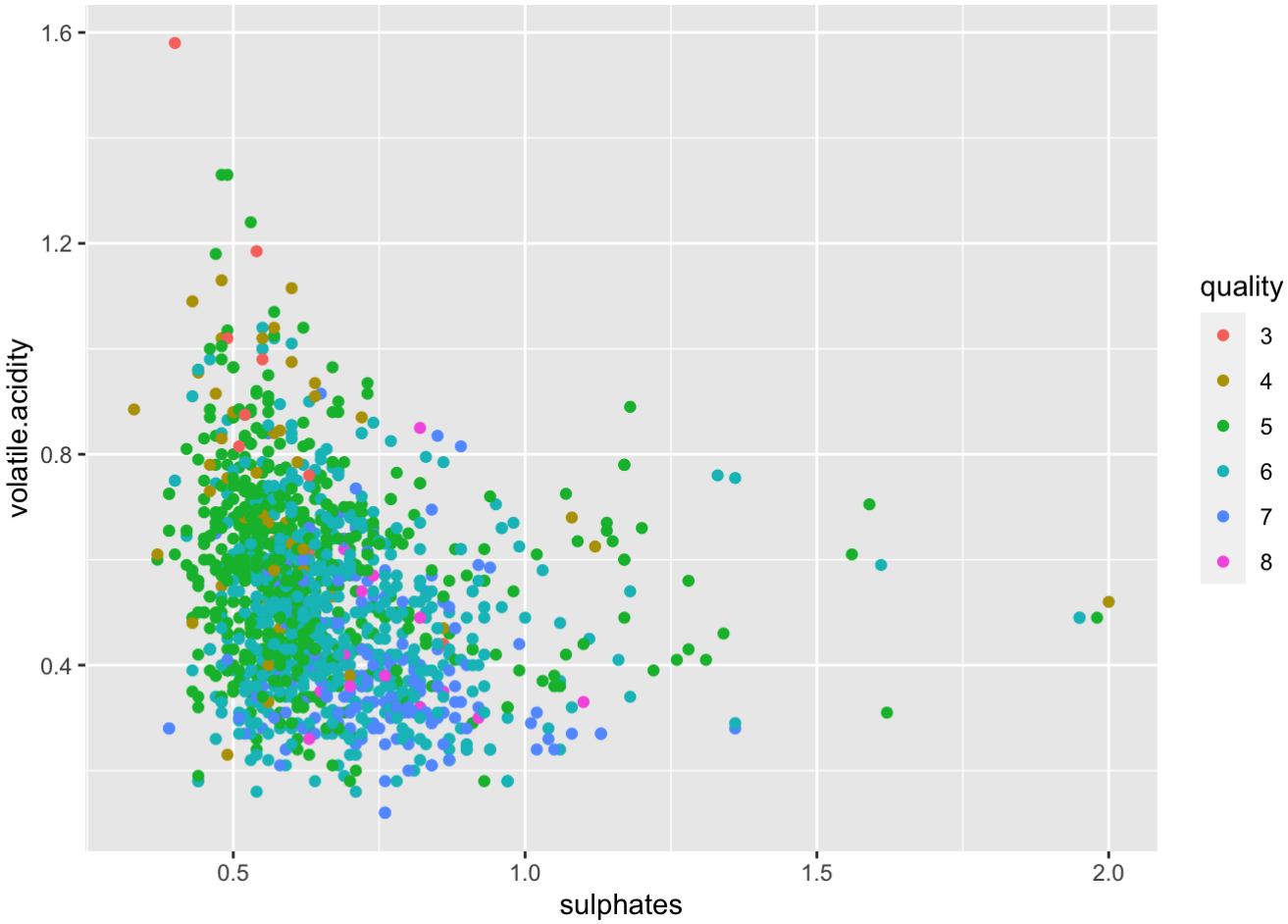



```
set.seed(2)
sv.km <- kmeans(S$u, 6, nstart = 20)
R$svclusters = as.factor(sv.km$cluster)
ggplot(R, aes(x=fixed.acidity, y=total.sulfur.dioxide, color=svclusters)) + geom_point()
+
  labs(title = "K-Means Plot of 6 clusters")
```

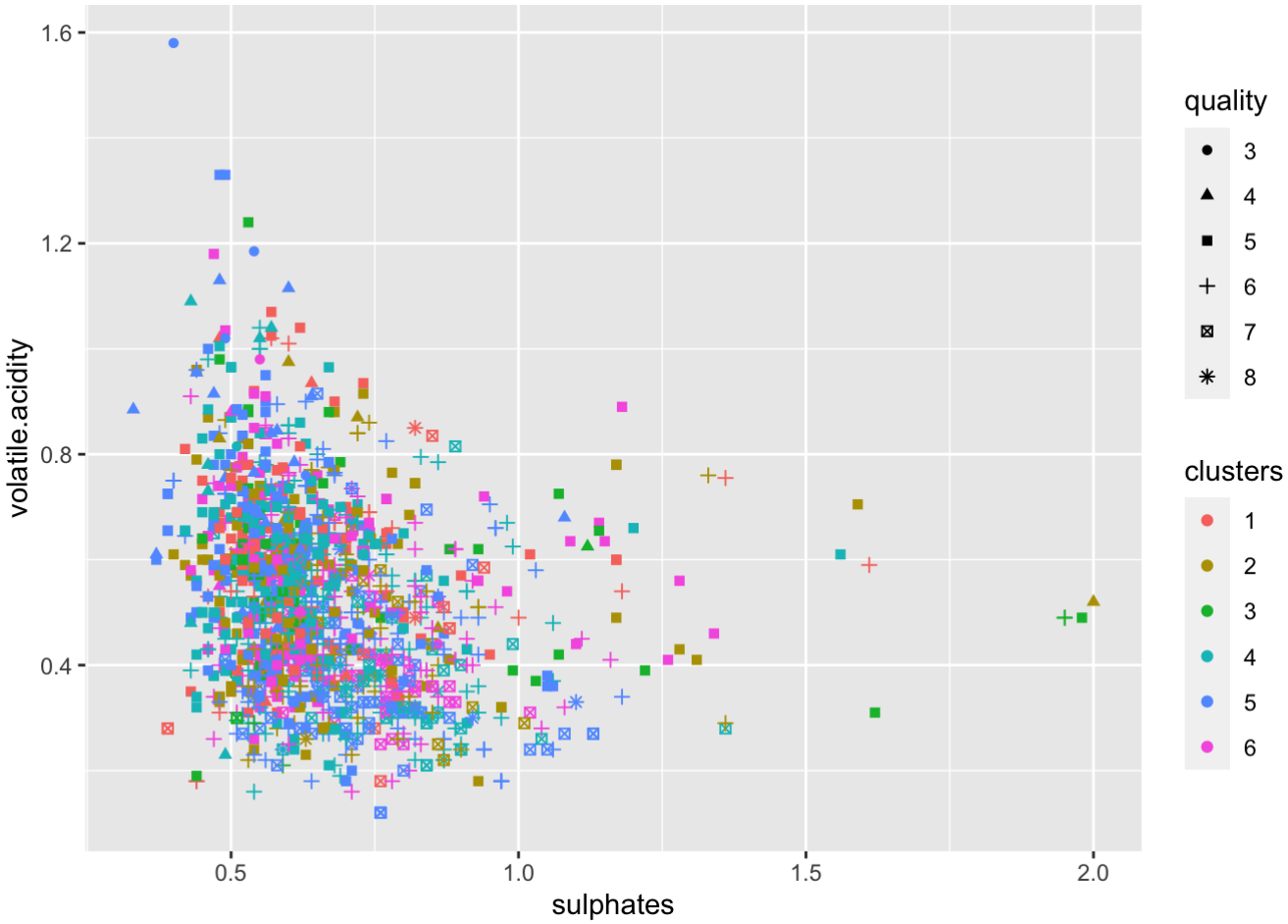


Looking at sulphates and volatile acidity:

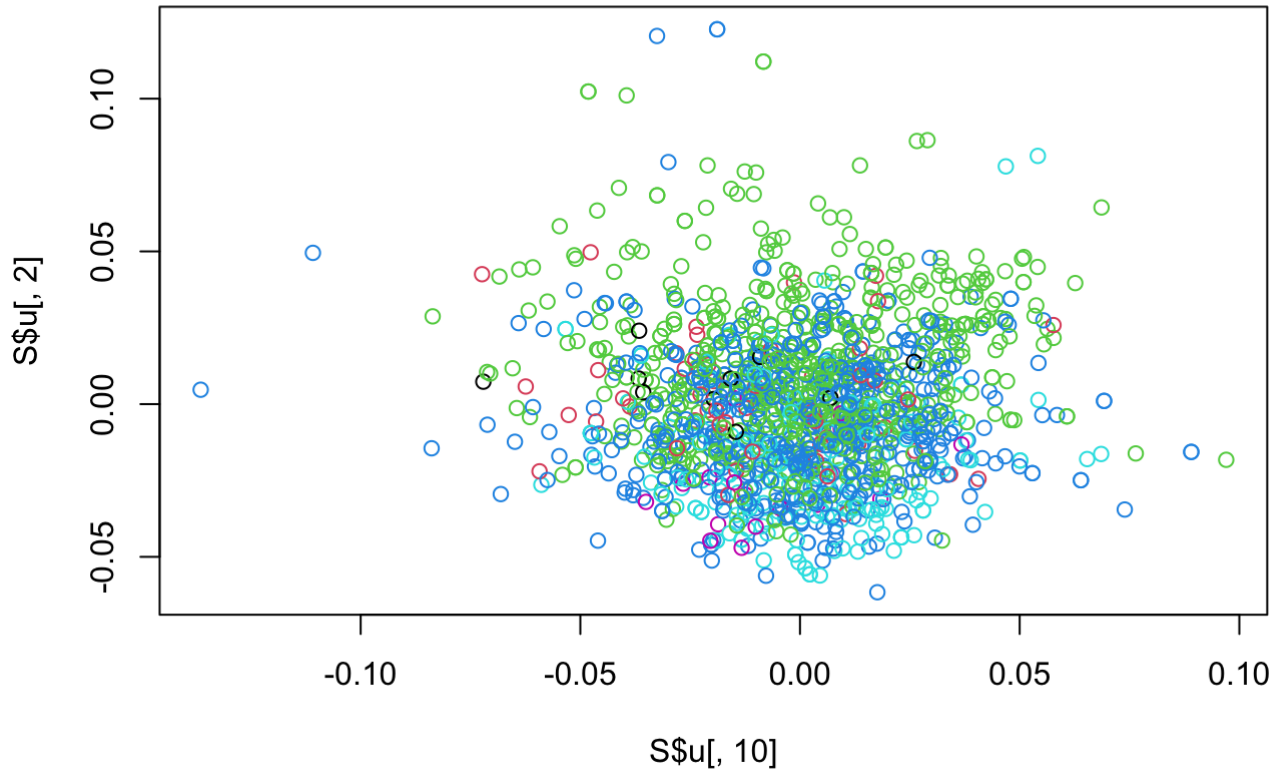
```
R = red %>% na.omit()
R$quality <- as.factor(R$quality)
ggplot(R, aes(x=sulphates, y=volatile.acidity, color=quality)) + geom_point()
```



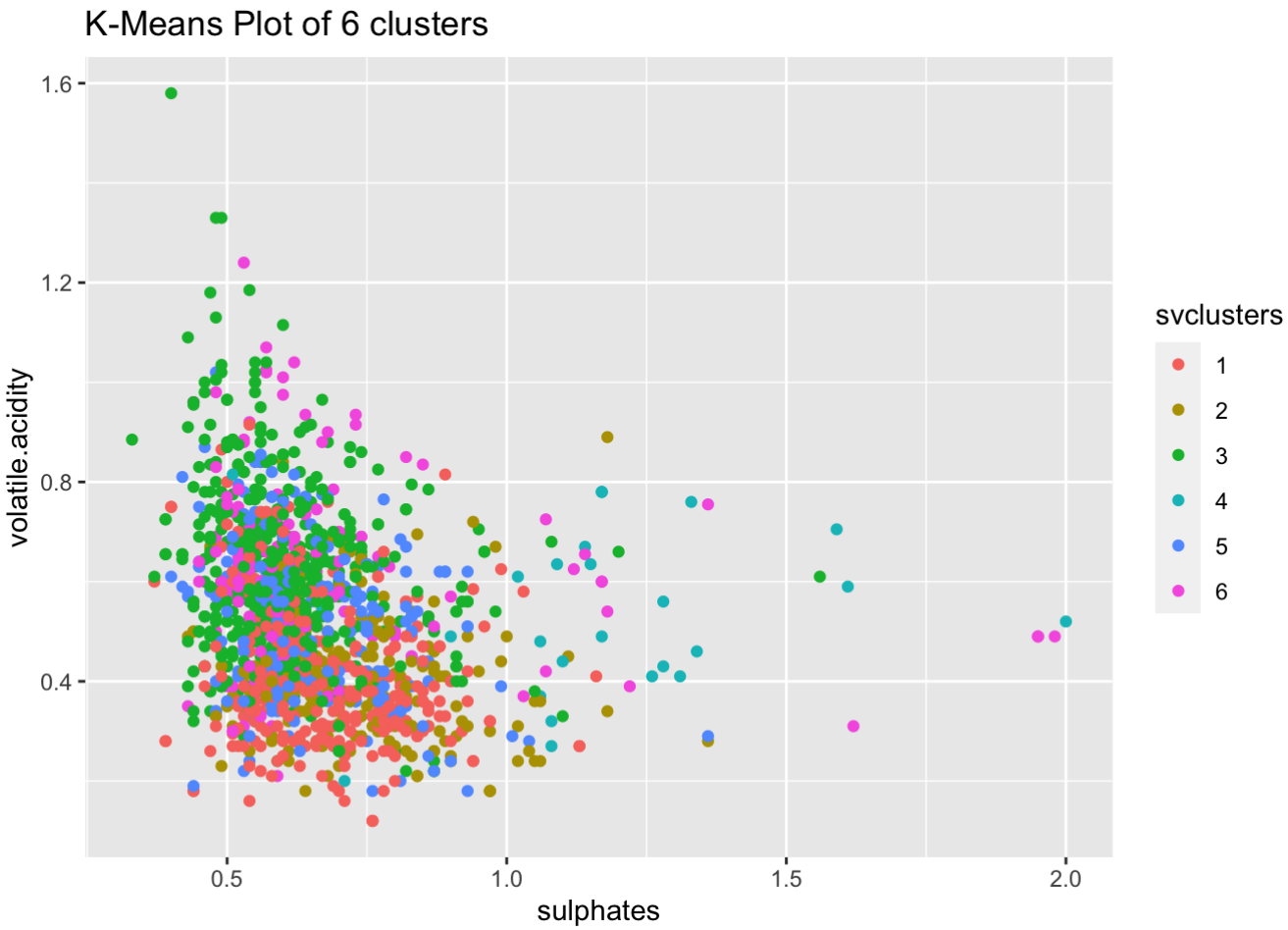

```
set.seed(2)
red.km2 <- kmeans(select(R, -c(quality)), 6, nstart = 20)
R$clusters = as.factor(red.km2 $cluster)
ggplot(R, aes(x=sulphates, y=volatile.acidity, color=clusters, shape=quality)) + geom_po
int()
```




```
Rscale <-  R %>% select(-c(clusters,quality)) %>% scale()
S <- svd(Rscale)
plot(S$u[,10], S$u[,2], col=R$quality)
```



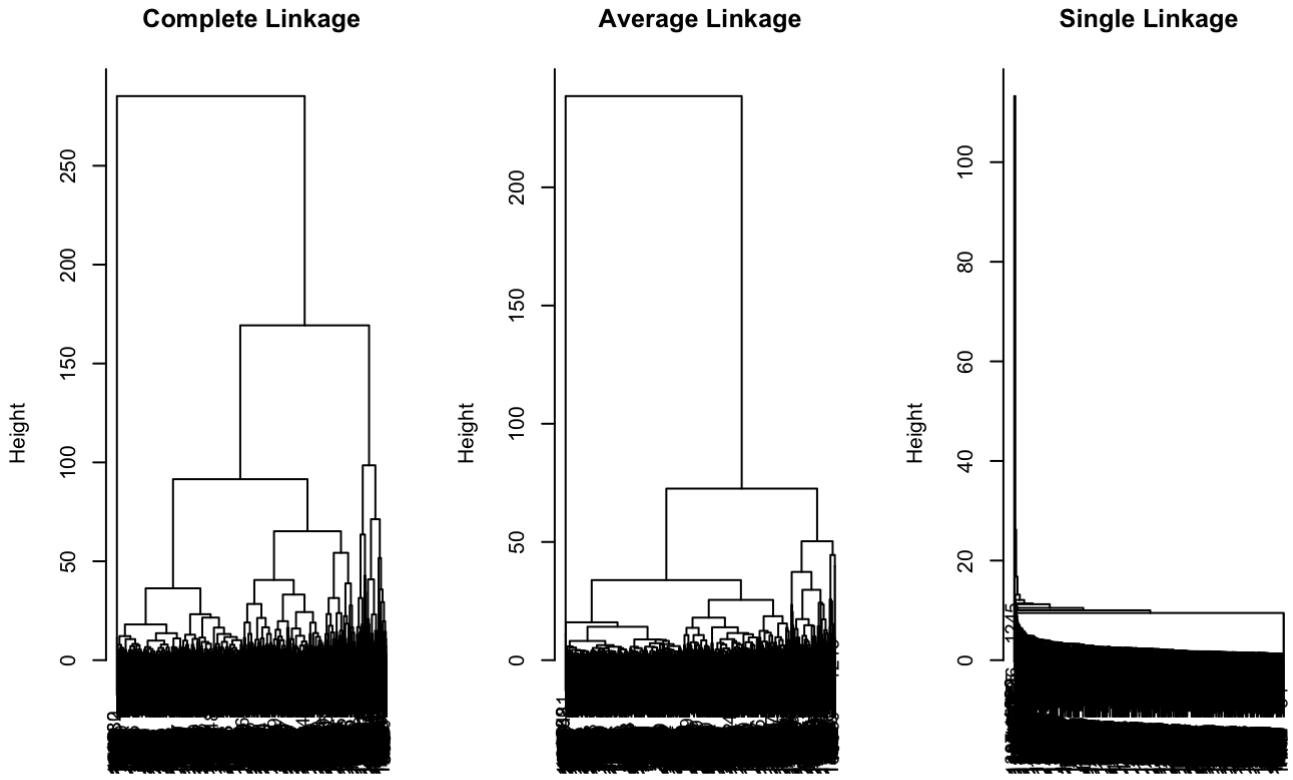

```
set.seed(2)
sv.km <- kmeans(S$u, 6, nstart = 20)
R$svclusters = as.factor(sv.km$cluster)
ggplot(R, aes(x=sulphates, y=volatile.acidity, color=svclusters)) + geom_point()+
  labs(title = "K-Means Plot of 6 clusters")
```



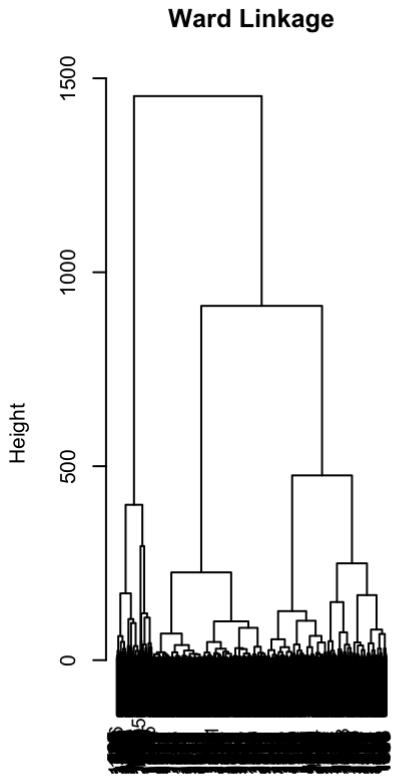
Hierarchical Clustering:

```
hc.complete <- hclust(dist(red_data), method = "complete")
hc.average <- hclust(dist(red_data), method = "average")
hc.single <- hclust(dist(red_data), method = "single")
hc.ward <- hclust(dist(red_data), method = "ward.D2")
```

```
par(mfrow = c(1, 3))
plot(hc.complete, main = "Complete Linkage",
    xlab = "", sub = "", cex = .9)
plot(hc.average, main = "Average Linkage",
    xlab = "", sub = "", cex = .9)
plot(hc.single, main = "Single Linkage",
    xlab = "", sub = "", cex = .9)
```

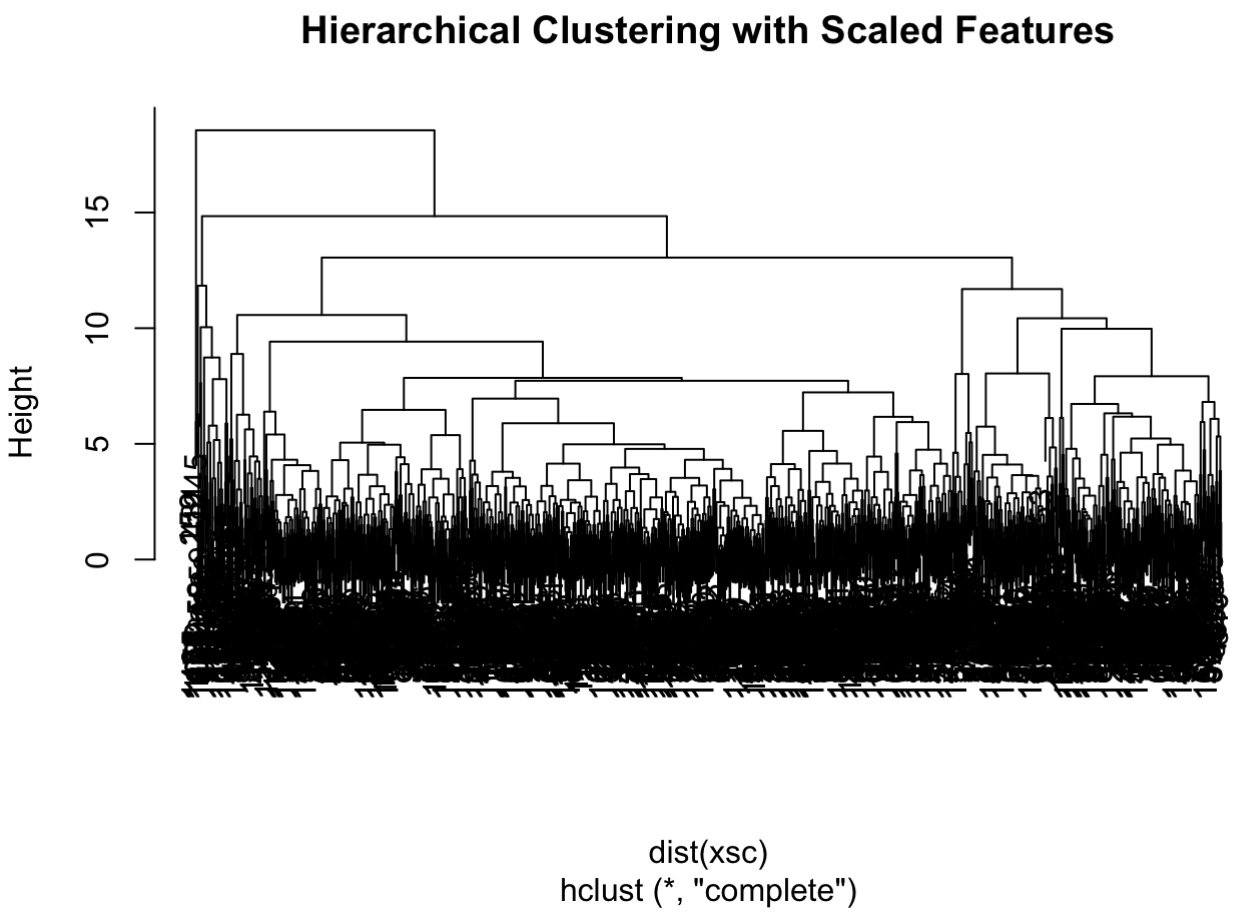



```
plot(hc.ward, main = "Ward Linkage",
    xlab = "", sub = "", cex = .9)
```

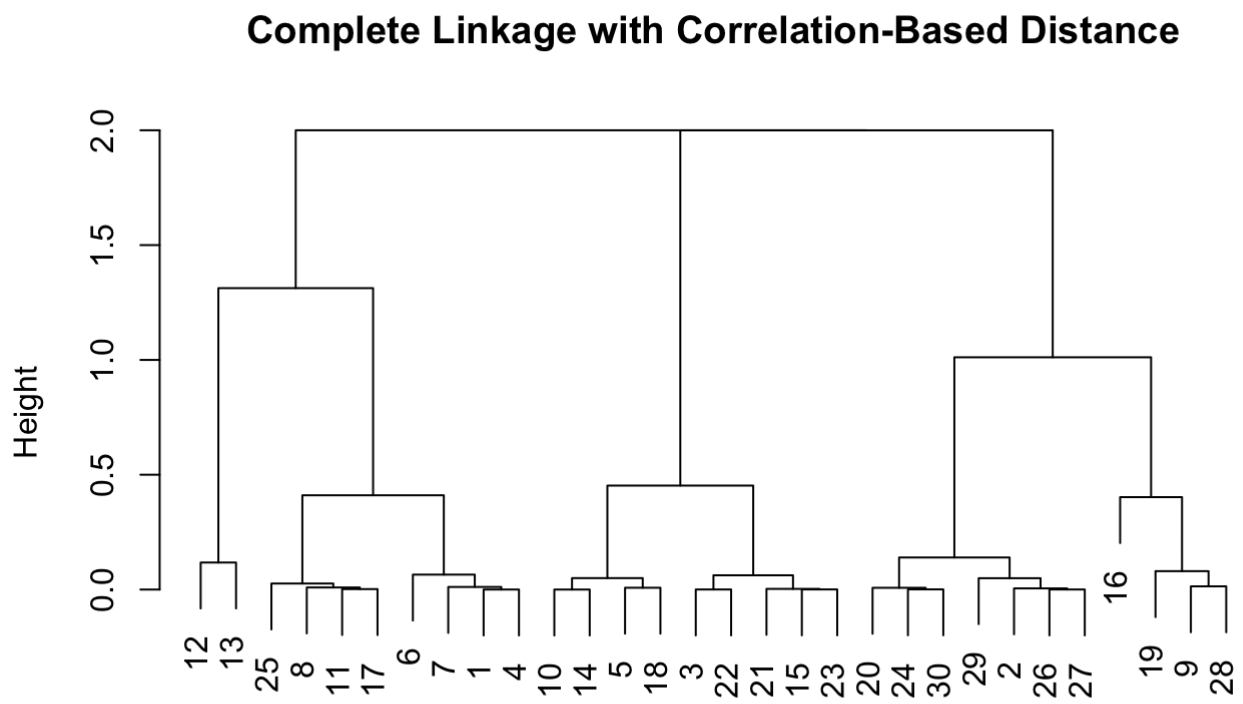


Scaling using complete linkage:

```
xsc <- scale(red_data)
hc.complete2 <- hclust(dist(xsc), method = "complete")
plot(hclust(dist(xsc), method = "complete"),
    main = "Hierarchical Clustering with Scaled Features")
```

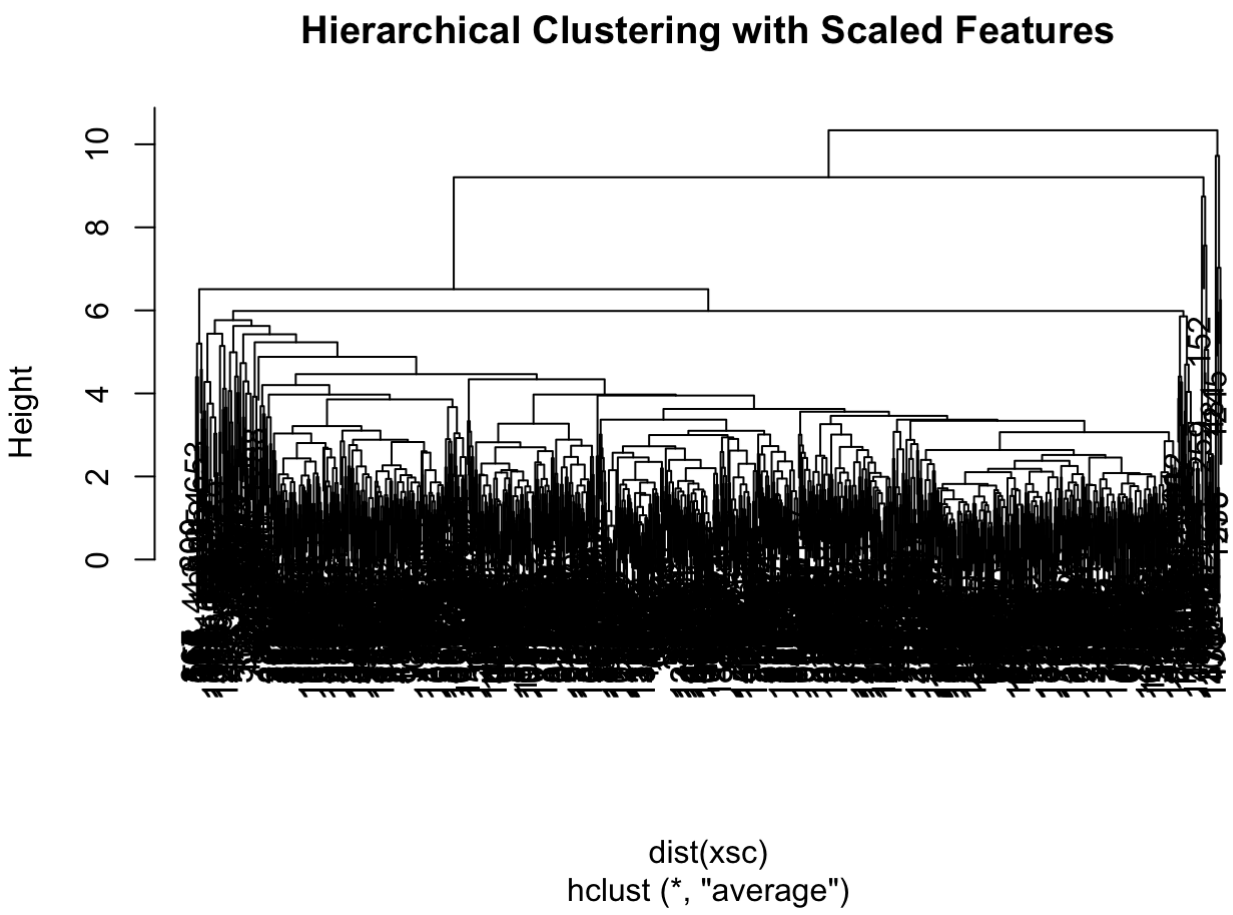



```
xsc  <- matrix(rnorm(30 * 3), ncol = 3)
dd <- as.dist(1 - cor(t(xsc)))
plot(hclust(dd, method = "complete"),
    main = "Complete Linkage with Correlation-Based Distance",
    xlab = "", sub = "")
```

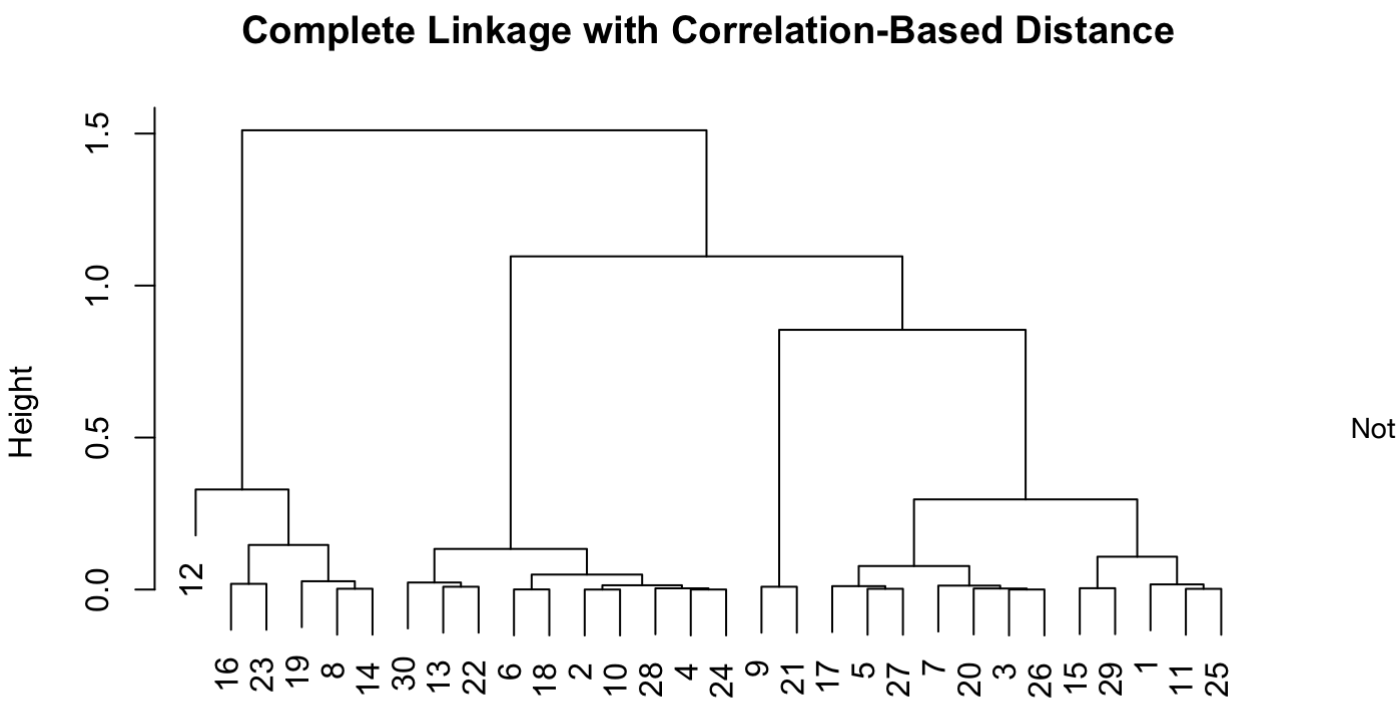


Scaling using average linkage:

```
xsc <- scale(red_data)
hc.complete2 <- hclust(dist(xsc), method = "average")
plot(hclust(dist(xsc), method = "average"),
    main = "Hierarchical Clustering with Scaled Features")
```




```
xsc  <- matrix(rnorm(30 * 3), ncol = 3)
dd <- as.dist(1 - cor(t(xsc)))
plot(hclust(dd, method = "average"),
    main = "Complete Linkage with Correlation-Based Distance",
    xlab = "", sub = "")
```



Not

much to interpret for these plots.

Supervised Model:

Decision Trees: Multi-class

Creating categories for quality :

```
red_data2 <- red%>%
  mutate(quality = factor(quality , levels = c(3,4,5,6,7,8),labels = c("low quality","lo
w quality","average quality","average quality","high quality","high quality")))
red_data2 <- red_data2 %>%
  mutate(quality = relevel(quality, ref = "average quality"))

levels(red_data2$quality) <- c("low quality", "average quality", "high quality")
```

```
levels(red_data2$quality)
```

```
## [1] "low quality"     "average quality" "high quality"
```

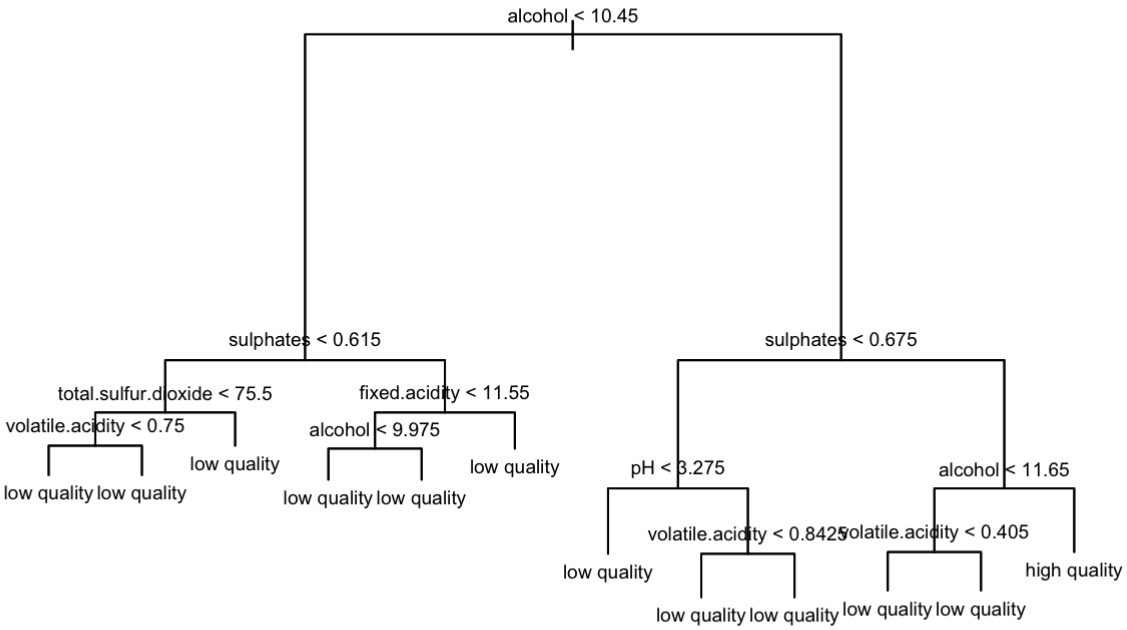
Tree model without training/test set:

```
tree.red <- tree(quality~.-quality,red_data2)
summary(tree.red)
```

```
##
## Classification tree:
## tree(formula = quality ~ . - quality, data = red_data2)
## Variables actually used in tree construction:
## [1] "alcohol"              "sulphates"            "total.sulfur.dioxide"
## [4] "volatile.acidity"     "fixed.acidity"        "pH"
## Number of terminal nodes:  12
## Residual mean deviance:  0.7809 = 1239 / 1587
## Misclassification error rate: 0.1538 = 246 / 1599
```

We get a training error rate of .15 or 15%

```
plot(tree.red)
text(tree.red,cex = 0.6,pretty=0)
```




```
tree.red
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 1599 1782.00 low quality ( 0.824891 0.039400 0.135710 )
##    2) alcohol < 10.45 916  583.40 low quality ( 0.924672 0.043668 0.031659 )
##      4) sulphates < 0.615 517  265.60 low quality ( 0.936170 0.058027 0.005803 )
##        8) total.sulfur.dioxide < 75.5 379  244.30 low quality ( 0.912929 0.079156 0.0
07916 )
##         16) volatile.acidity < 0.75 328  167.60 low quality ( 0.939024 0.051829 0.009
146 ) *
##         17) volatile.acidity > 0.75 51   57.90 low quality ( 0.745098 0.254902 0.0000
00 ) *
##        9) total.sulfur.dioxide > 75.5 138    0.00 low quality ( 1.000000 0.000000 0.0
00000 ) *
##      5) sulphates > 0.615 399  284.40 low quality ( 0.909774 0.025063 0.065163 )
##       10) fixed.acidity < 11.55 363  208.50 low quality ( 0.933884 0.024793 0.041322
)
##         20) alcohol < 9.975 263  104.30 low quality ( 0.958175 0.030418 0.011407 ) *
##         21) alcohol > 9.975 100   84.33 low quality ( 0.870000 0.010000 0.120000 ) *
##       11) fixed.acidity > 11.55 36   52.71 low quality ( 0.666667 0.027778 0.305556 )
*
##    3) alcohol > 10.45 683  989.90 low quality ( 0.691069 0.033675 0.275256 )
##      6) sulphates < 0.675 357  466.20 low quality ( 0.784314 0.064426 0.151261 )
##       12) pH < 3.275 103  129.20 low quality ( 0.679612 0.000000 0.320388 ) *
##       13) pH > 3.275 254  295.10 low quality ( 0.826772 0.090551 0.082677 )
##         26) volatile.acidity < 0.8425 221  224.00 low quality ( 0.855204 0.049774 0.0
95023 ) *
##         27) volatile.acidity > 0.8425 33   43.26 low quality ( 0.636364 0.363636 0.00
0000 ) *
##      7) sulphates > 0.675 326  441.60 low quality ( 0.588957 0.000000 0.411043 )
##       14) alcohol < 11.65 216  259.00 low quality ( 0.712963 0.000000 0.287037 )
##         28) volatile.acidity < 0.405 100  137.60 low quality ( 0.550000 0.000000 0.45
0000 ) *
##         29) volatile.acidity > 0.405 116   96.67 low quality ( 0.853448 0.000000 0.14
6552 ) *
##       15) alcohol > 11.65 110  141.80 high quality ( 0.345455 0.000000 0.654545 ) *
```

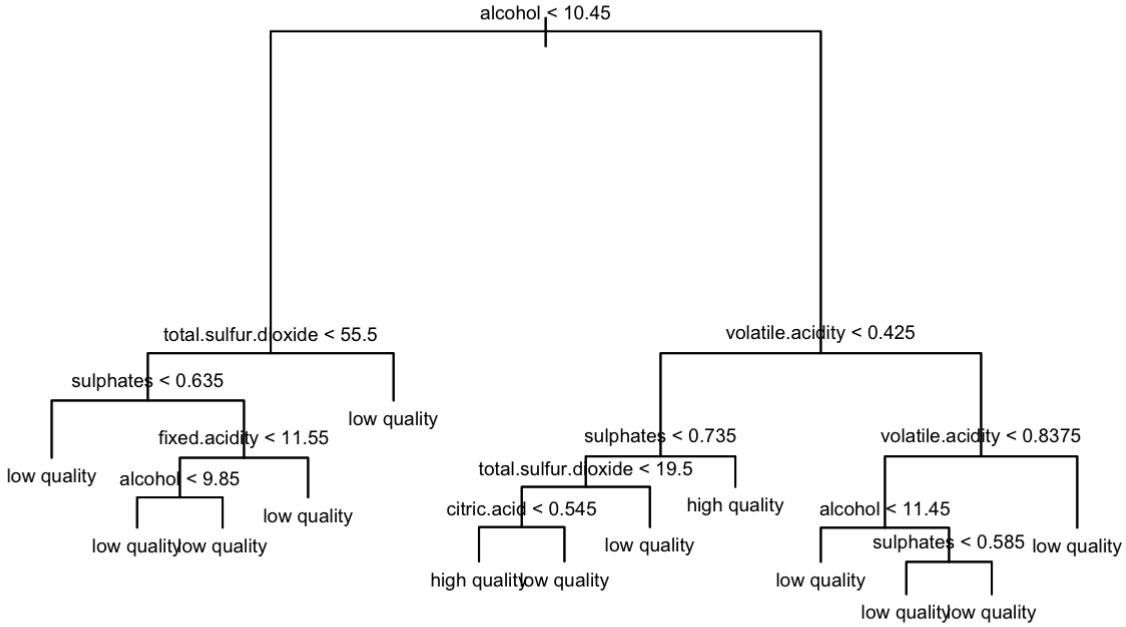
Let's make a train and test set:

```
#training and test data
set.seed(1)
train <- sample(nrow(red_data2), nrow(red_data2)*.7)
red.train <- red_data2[train,]
red.test <- red_data2[-train,]
```

```
tree.red2 <- tree(quality~.-quality,red_data2, subset = train)
summary(tree.red2)
```

```
##
## Classification tree:
## tree(formula = quality ~ . - quality, data = red_data2, subset = train)
## Variables actually used in tree construction:
## [1] "alcohol"              "total.sulfur.dioxide" "sulphates"
## [4] "fixed.acidity"        "volatile.acidity"     "citric.acid"
## Number of terminal nodes:  13
## Residual mean deviance:  0.7536 = 833.4 / 1106
## Misclassification error rate: 0.1492 = 167 / 1119
```

```
plot(tree.red2)
text(tree.red2,cex = 0.6, pretty = 0)
```




```
tree.red2
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 1119 1230.00 low quality ( 0.828418 0.038427 0.133155 )
##    2) alcohol < 10.45 639  398.30 low quality ( 0.926448 0.045383 0.028169 )
##      4) total.sulfur.dioxide < 55.5 396  319.80 low quality ( 0.896465 0.058081 0.045
455 )
##        8) sulphates < 0.635 233  185.30 low quality ( 0.888412 0.094421 0.017167 ) *
##        9) sulphates > 0.635 163  107.50 low quality ( 0.907975 0.006135 0.085890 )
##         18) fixed.acidity < 11.55 143   55.89 low quality ( 0.951049 0.000000 0.04895
1 )
##           36) alcohol < 9.85 89    0.00 low quality ( 1.000000 0.000000 0.000000 ) *
##           37) alcohol > 9.85 54   41.65 low quality ( 0.870370 0.000000 0.129630 ) *
##         19) fixed.acidity > 11.55 20   32.95 low quality ( 0.600000 0.050000 0.350000
) *
##      5) total.sulfur.dioxide > 55.5 243   56.27 low quality ( 0.975309 0.024691 0.000
000 ) *
##    3) alcohol > 10.45 480  680.20 low quality ( 0.697917 0.029167 0.272917 )
##      6) volatile.acidity < 0.425 211  298.90 low quality ( 0.573460 0.004739 0.421801
)
##       12) sulphates < 0.735 121  161.40 low quality ( 0.677686 0.008264 0.314050 )
##         24) total.sulfur.dioxide < 19.5 42   65.68 high quality ( 0.428571 0.023810
0.547619 )
##           48) citric.acid < 0.545 34   49.51 high quality ( 0.294118 0.029412 0.67647
1 ) *
##           49) citric.acid > 0.545 8    0.00 low quality ( 1.000000 0.000000 0.000000
) *
##         25) total.sulfur.dioxide > 19.5 79   76.79 low quality ( 0.810127 0.000000 0.
189873 ) *
##       13) sulphates > 0.735 90  123.20 high quality ( 0.433333 0.000000 0.566667 ) *
##      7) volatile.acidity > 0.425 269  332.70 low quality ( 0.795539 0.048327 0.156134
)
##       14) volatile.acidity < 0.8375 246  261.10 low quality ( 0.817073 0.016260 0.166
667 )
##         28) alcohol < 11.45 153  122.60 low quality ( 0.888889 0.019608 0.091503 ) *
##         29) alcohol > 11.45 93  122.40 low quality ( 0.698925 0.010753 0.290323 )
##           58) sulphates < 0.585 25   16.71 low quality ( 0.920000 0.040000 0.040000 )
*
##           59) sulphates > 0.585 68   90.47 low quality ( 0.617647 0.000000 0.382353 )
*
##       15) volatile.acidity > 0.8375 23   37.99 low quality ( 0.565217 0.391304 0.0434
78 ) *
```

```
set.seed(1)
test <- red_data2$quality[-train]
tree.pred <- predict(tree.red2, red.test,
    type = "class")
table <- table(tree.pred, test)
table
```

```
##                 test
## tree.pred         low quality average quality high quality
##   low quality             368              19           35
##   average quality           0               0            0
##   high quality             24               1           33
```

```
mean(tree.pred == test)
```

```
## [1] 0.8354167
```

```
accuracy_Test <- sum(diag(table)) / sum(table)
print(paste('Accuracy for test', accuracy_Test))
```

```
## [1] "Accuracy for test 0.835416666666667"
```

Pruning the tree:

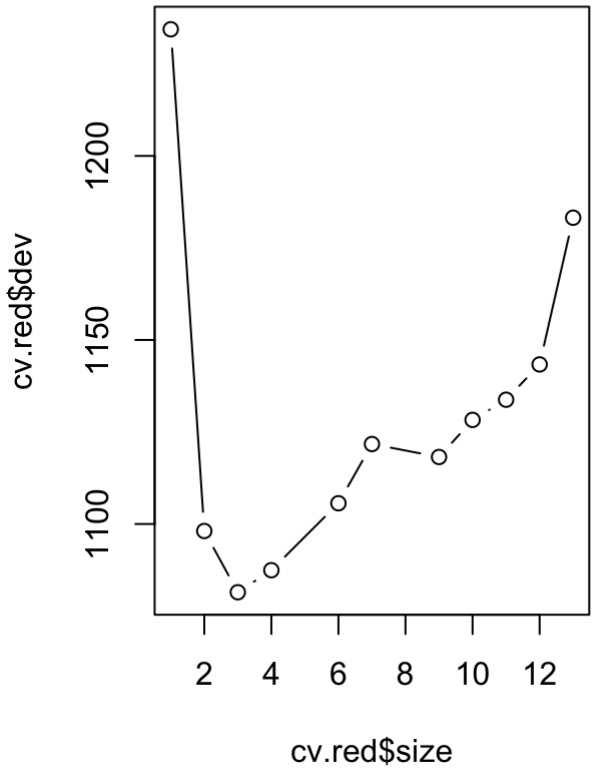
```
set.seed(2)
cv.red <- cv.tree(tree.red2, FUN = prune.tree)
names(cv.red)
```

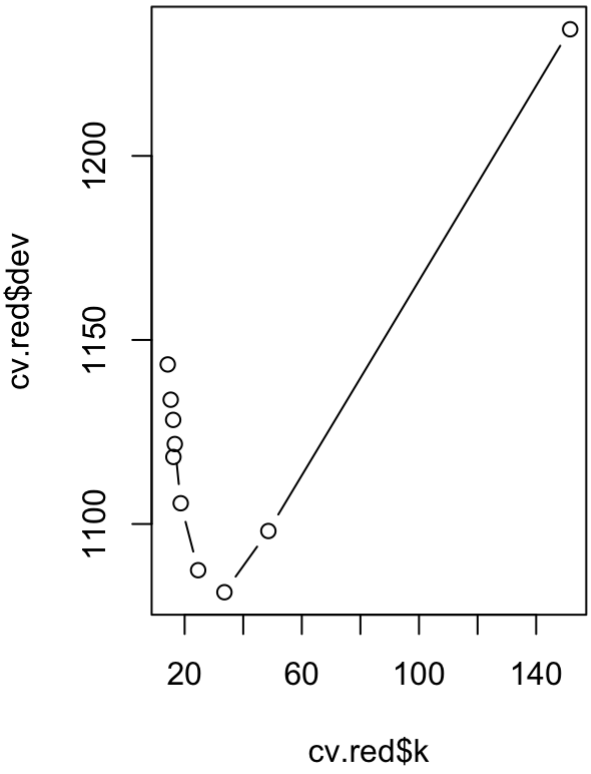
```
## [1] "size"   "dev"    "k"      "method"
```

```
cv.red
```

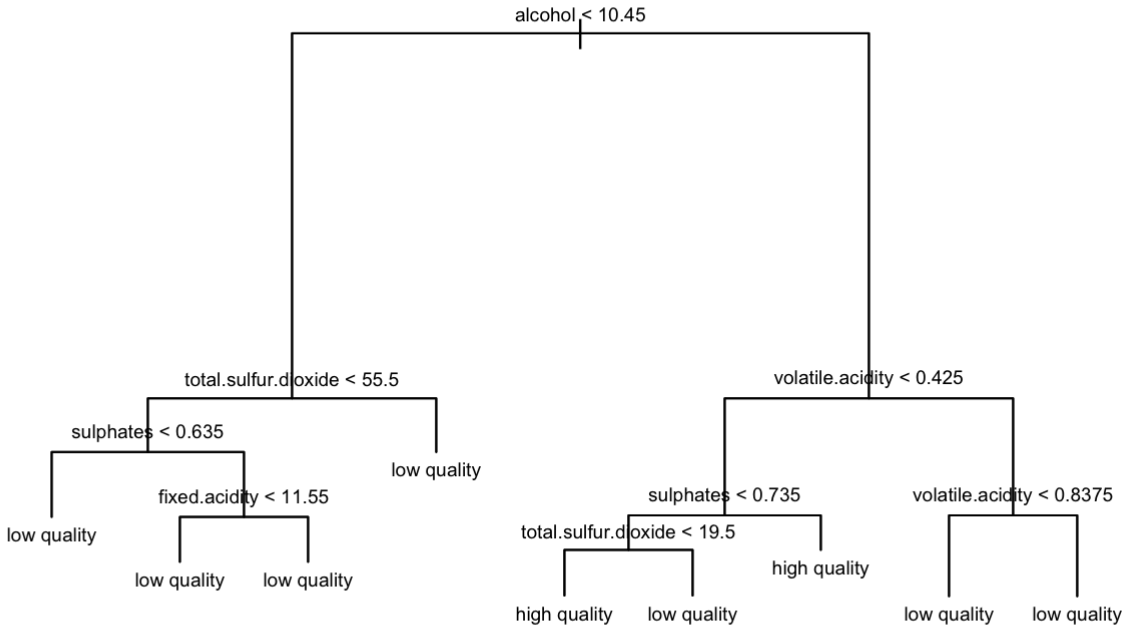
```
## $size
##  [1] 13 12 11 10  9  7  6  4  3  2  1
##
## $dev
##  [1] 1183.209 1143.345 1133.763 1128.280 1118.204 1121.720 1105.637 1087.449
##  [9] 1081.463 1098.097 1234.396
##
## $k
##  [1]      -Inf  14.23503  15.23913  16.08671  16.17005  16.64574  18.65664
##  [8]  24.61351  33.57877  48.58230 151.60429
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
par(mfrow = c(1, 2))
plot(cv.red$size, cv.red$dev, type = "b")
plot(cv.red$k, cv.red$dev, type = "b")
```






```
optimal_K <- which.min(cv.red$dev)
prune.red <- prune.tree(tree.red2, best = optimal_K)
plot(prune.red)
text(prune.red,cex = 0.6, pretty = 0)
```




```
summary(prune.red)
```

```
##
## Classification tree:
## snip.tree(tree = tree.red2, nodes = c(18L, 14L, 24L))
## Variables actually used in tree construction:
## [1] "alcohol"              "total.sulfur.dioxide" "sulphates"
## [4] "fixed.acidity"        "volatile.acidity"
## Number of terminal nodes:  9
## Residual mean deviance:  0.8065 = 895.2 / 1110
## Misclassification error rate: 0.1564 = 175 / 1119
```

```
summary(prune.red)
```

```
##
## Classification tree:
## snip.tree(tree = tree.red2, nodes = c(18L, 14L, 24L))
## Variables actually used in tree construction:
## [1] "alcohol"              "total.sulfur.dioxide" "sulphates"
## [4] "fixed.acidity"        "volatile.acidity"
## Number of terminal nodes:  9
## Residual mean deviance:  0.8065 = 895.2 / 1110
## Misclassification error rate: 0.1564 = 175 / 1119
```

```
tree.pred <- predict(prune.red, red.test,
    type = "class")
table(tree.pred, test)
```

```
##                 test
## tree.pred         low quality average quality high quality
##   low quality             365              19           35
##   average quality           0               0            0
##   high quality             27               1           33
```

```
mean(tree.pred == test)
```

```
## [1] 0.8291667
```

```
accuracy_Test <- sum(diag(table)) / sum(table)
```

Pruning did worse by 1% with a slightly lower accuracy of 82.91%

Bagging and random forest:

```
bag.red <- randomForest(quality ~ .-quality,red_data2, subset = train, mtry = 12, import
ance = TRUE)
```

```
## Warning in randomForest.default(m, y, ...): invalid mtry: reset to within valid
## range
```

```
bag.red
```

```
##
## Call:
##  randomForest(formula = quality ~ . - quality, data = red_data2,      mtry = 12, impo
rtance = TRUE, subset = train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 11
##
##         OOB estimate of  error rate: 13.32%
## Confusion matrix:
##                 low quality average quality high quality class.error
## low quality             890               3           34   0.0399137
## average quality          38               4            1   0.9069767
## high quality             73               0           76   0.4899329
```

```
yhat.bag <- predict(bag.red, newdata = red_data2[-train,])
yhat.bag.num <- as.numeric(yhat.bag)
test.num <- as.numeric(test)

# Check for any non-numeric values in yhat.bag.num or test.num
any(!is.na(yhat.bag.num) & !is.numeric(yhat.bag.num))
```

```
## [1] FALSE
```

```
any(!is.na(test.num) & !is.numeric(test.num))
```

```
## [1] FALSE
```

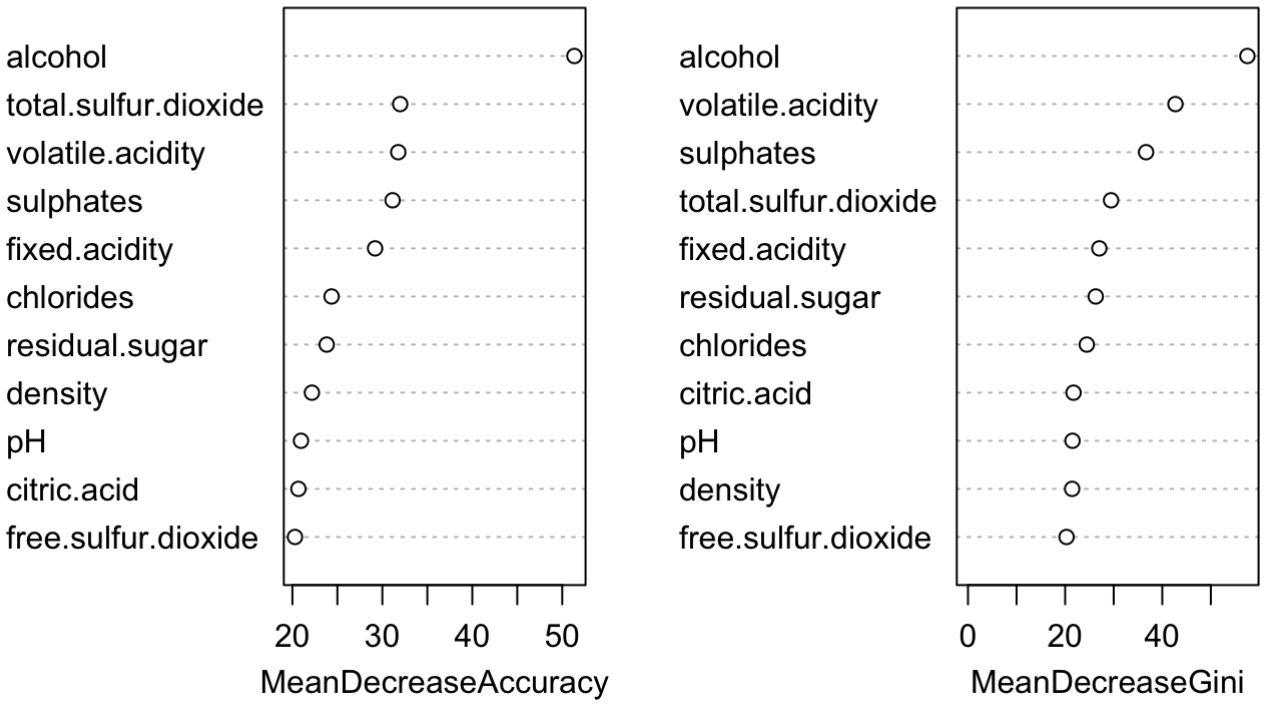
```
# Calculate the mean squared error
mean((yhat.bag.num - test.num)^2)
```

```
## [1] 0.3979167
```

```
importance(bag.red)
```

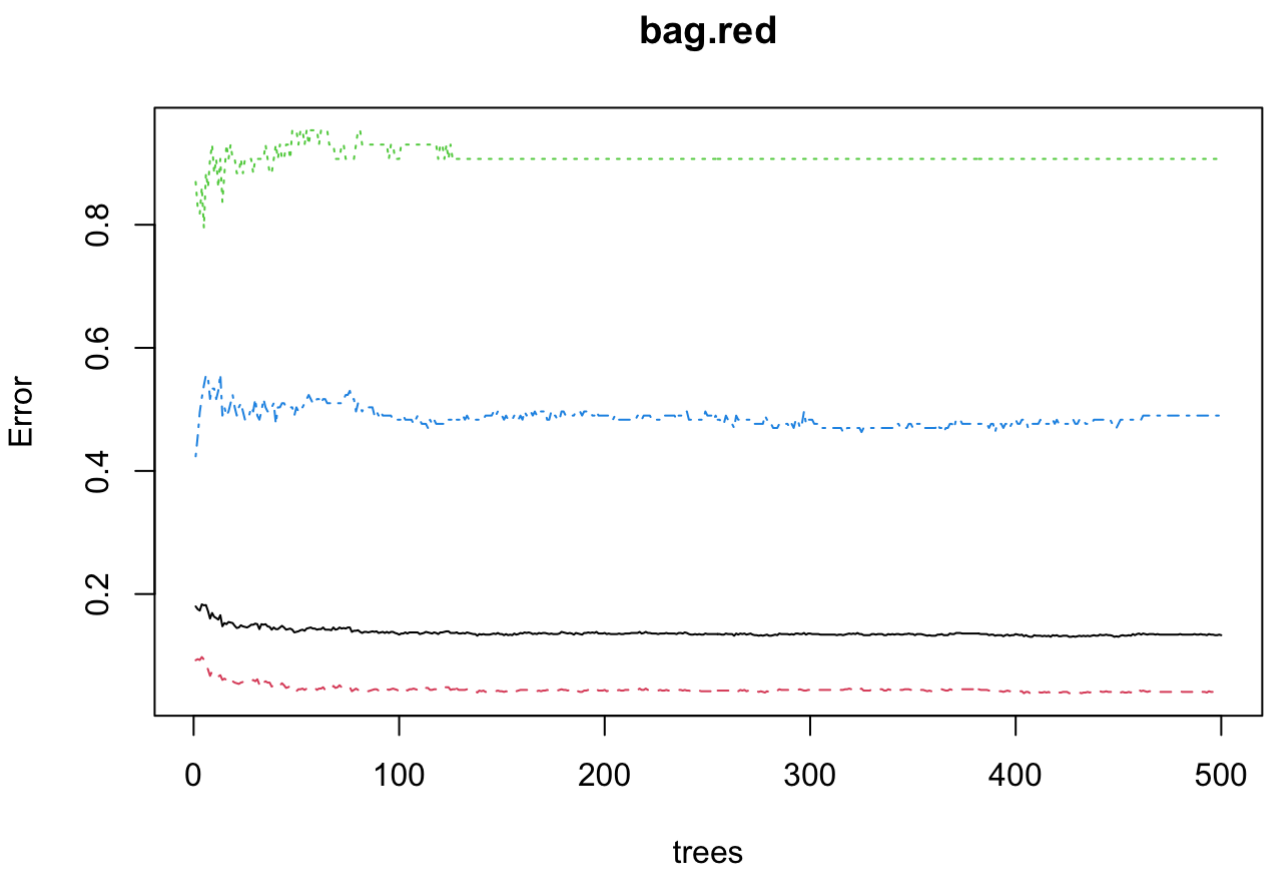
```
##                       low quality average quality high quality
## fixed.acidity           25.92005      -4.4067796    15.233878
## volatile.acidity        12.25224      21.3798235    32.011018
## citric.acid             16.56905      -1.9445302    12.299338
## residual.sugar          19.23403      -0.6396272    16.971530
## chlorides               18.98632      -4.6758356    18.602950
## free.sulfur.dioxide     18.58586       0.4800584     6.351773
## total.sulfur.dioxide    24.15458       8.3333199    24.063339
## density                 18.47160      -6.3499100    13.429298
## pH                      17.40513      -0.7920192    12.214389
## sulphates               12.69013       2.7936621    36.807276
## alcohol                 26.15110       6.7789477    64.657521
##                       MeanDecreaseAccuracy MeanDecreaseGini
## fixed.acidity                     29.19257         27.03487
## volatile.acidity                  31.76846         42.71128
## citric.acid                       20.66578         21.73244
## residual.sugar                    23.81497         26.29912
## chlorides                         24.35355         24.46054
## free.sulfur.dioxide               20.28710         20.28934
## total.sulfur.dioxide              31.98078         29.45702
## density                           22.16989         21.44372
## pH                                20.95521         21.51584
## sulphates                         31.13905         36.66708
## alcohol                           51.34979         57.52321
```

```
varImpPlot(bag.red,main = "Variable Importance Plot for Wine Quality")
```



Alcohol, Sulphates, and volatile.acidity are the important variables.

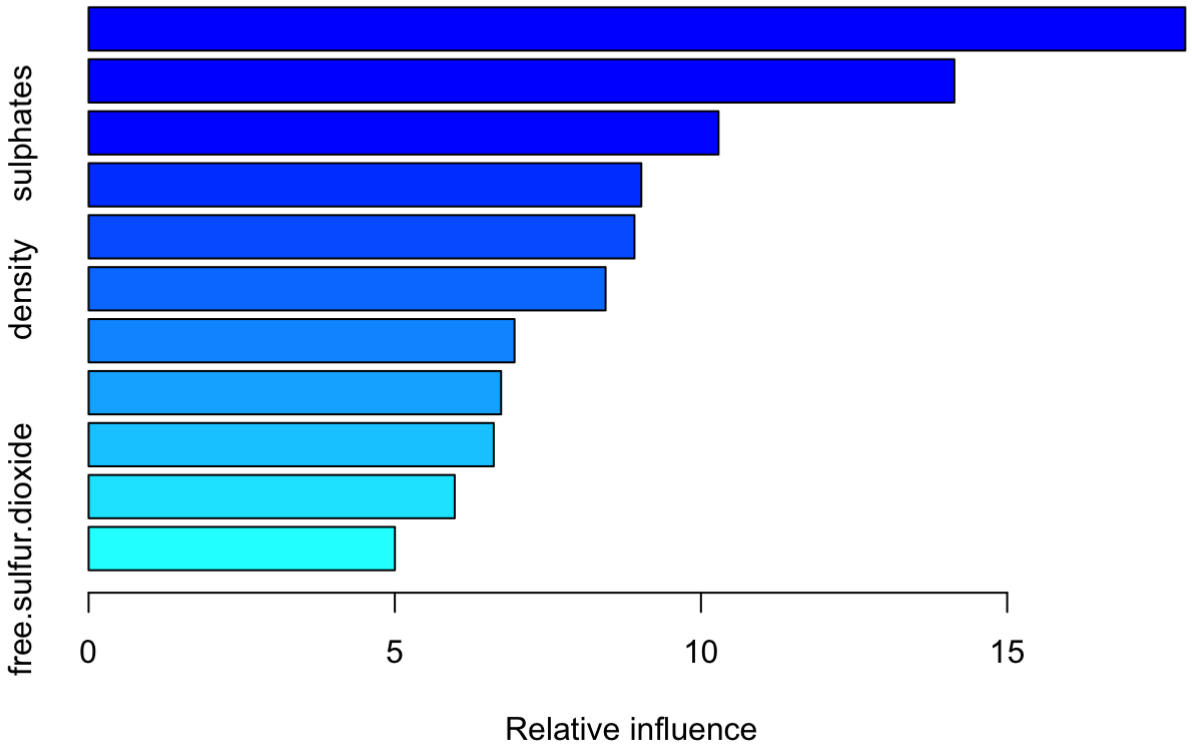
```
plot(bag.red)
```



Boosting:

```
#training and test data
set.seed(1)
train <- sample(nrow(red), nrow(red)*.7)
red.train <- red[train,]
red.test <- red[-train,]
```

```
set.seed(1)
boost.red <- gbm(quality ~ .-quality, data = red[train, ],
    distribution = "gaussian", n.trees = 1000,
    interaction.depth = 4)
summary(boost.red)
```



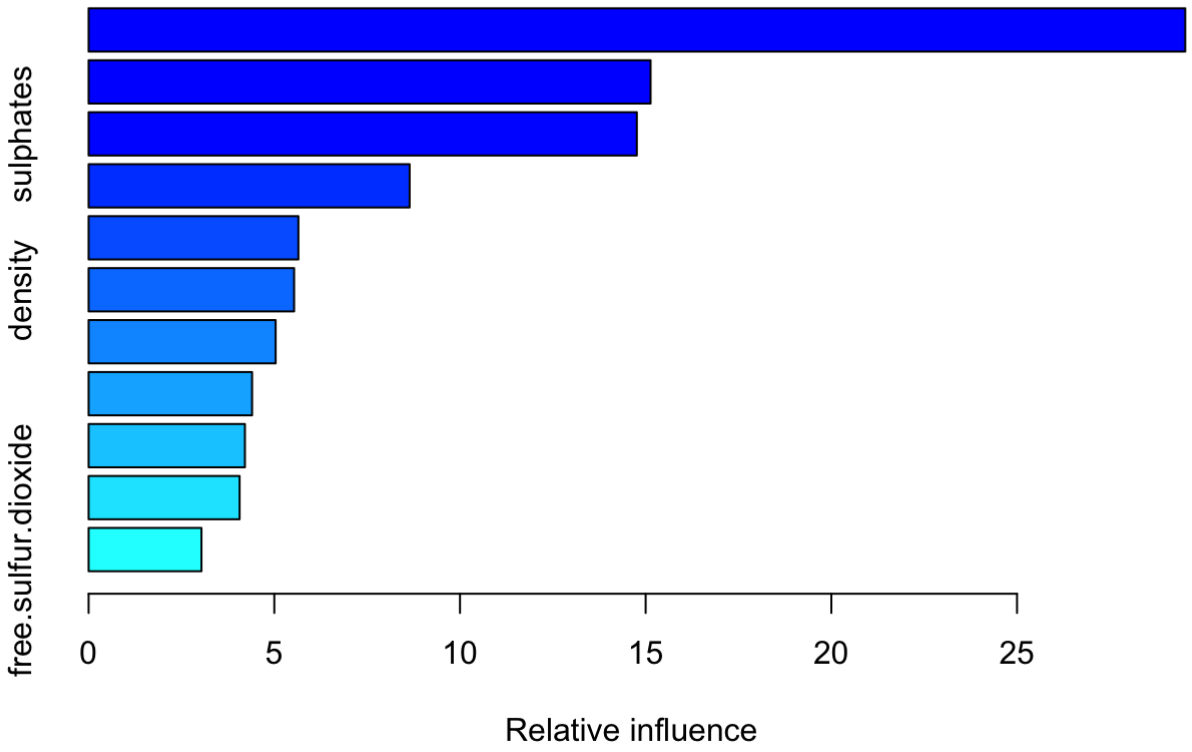

```
##                                     var   rel.inf
## alcohol                         alcohol 17.907982
## volatile.acidity       volatile.acidity 14.137570
## sulphates                     sulphates 10.286493
## total.sulfur.dioxide total.sulfur.dioxide  9.025763
## chlorides                     chlorides  8.913768
## density                         density  8.442271
## citric.acid                 citric.acid  6.955181
## pH                                   pH  6.735526
## fixed.acidity             fixed.acidity  6.615733
## residual.sugar           residual.sugar  5.978221
## free.sulfur.dioxide free.sulfur.dioxide  5.001492
```

```
testred <- red[-train,"quality"]
yhat.boost <- predict(boost.red,
    newdata = red[-train, ], n.trees = 1000)
mean((yhat.boost - testred)^2)
```

```
## [1] 0.4207397
```

Changing shrinkage:

```
set.seed(1)
boost.red <- gbm(quality ~ .-quality, data = red[train, ],
    distribution = "gaussian", n.trees = 1000,
    interaction.depth = 4, shrinkage = .01,verbose = F)
summary(boost.red)
```




```
##                                     var   rel.inf
## alcohol                         alcohol 29.529839
## volatile.acidity       volatile.acidity 15.132436
## sulphates                     sulphates 14.762407
## total.sulfur.dioxide total.sulfur.dioxide  8.643360
## chlorides                     chlorides  5.648560
## density                         density  5.533317
## citric.acid                 citric.acid  5.036021
## pH                                   pH  4.401509
## fixed.acidity             fixed.acidity  4.208562
## residual.sugar           residual.sugar  4.065567
## free.sulfur.dioxide free.sulfur.dioxide  3.038423
```

```
yhat.boost <- predict(boost.red,
    newdata = red[-train, ], n.trees = 1000)
mean((yhat.boost - testred)^2)
```

```
## [1] 0.3945877
```

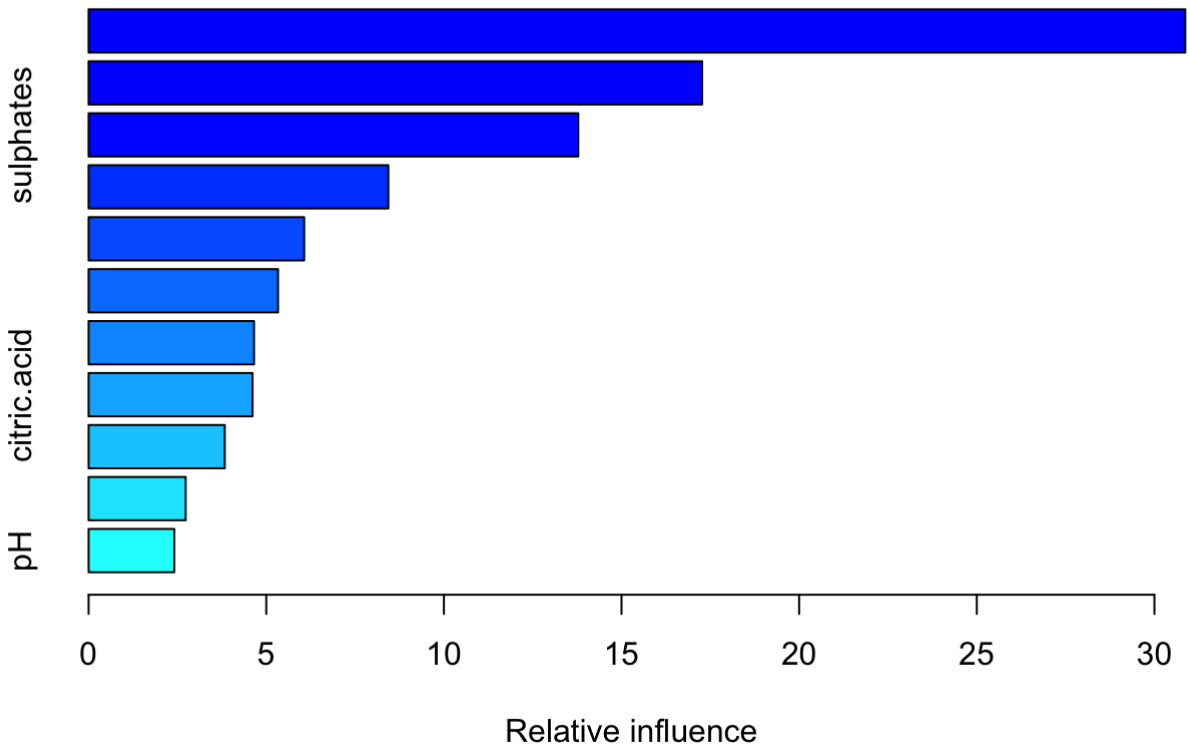
Notes on models: Importance: The “alcohol” content of the wine is a critical factor in determining its quality. Wines with higher alcohol content tend to be associated with higher quality or vice versa.

Discrimination: The “alcohol” variable effectively differentiates between different quality levels of wines. Wines with lower alcohol content are more likely to be classified as low quality, while wines with higher alcohol content are more likely to be classified as high quality.

Correlation: The “alcohol” content is strongly correlated with other important predictors or aspects related to wine quality. Its high vrel.inf value suggests that it captures a significant portion of the variability in the response variable.

Changing number of trees:

```
set.seed(1)
boost.red <- gbm(quality ~ .-quality, data = red[train, ],
    distribution = "gaussian", n.trees = 100,
    interaction.depth = 4)
summary(boost.red)
```



```
##                                       var   rel.inf
## alcohol                           alcohol 30.867596
## volatile.acidity         volatile.acidity 17.269827
## sulphates                       sulphates 13.782498
## total.sulfur.dioxide total.sulfur.dioxide  8.435883
## chlorides                       chlorides  6.067589
## density                           density  5.331114
## fixed.acidity               fixed.acidity  4.655485
## citric.acid                   citric.acid  4.614514
## residual.sugar             residual.sugar  3.832249
## free.sulfur.dioxide   free.sulfur.dioxide  2.731922
## pH                                    pH  2.411323
```

```
yhat.boost <- predict(boost.red,
    newdata = red[-train, ], n.trees = 100)
mean((yhat.boost - testred)^2)
```

```
## [1] 0.396579
```

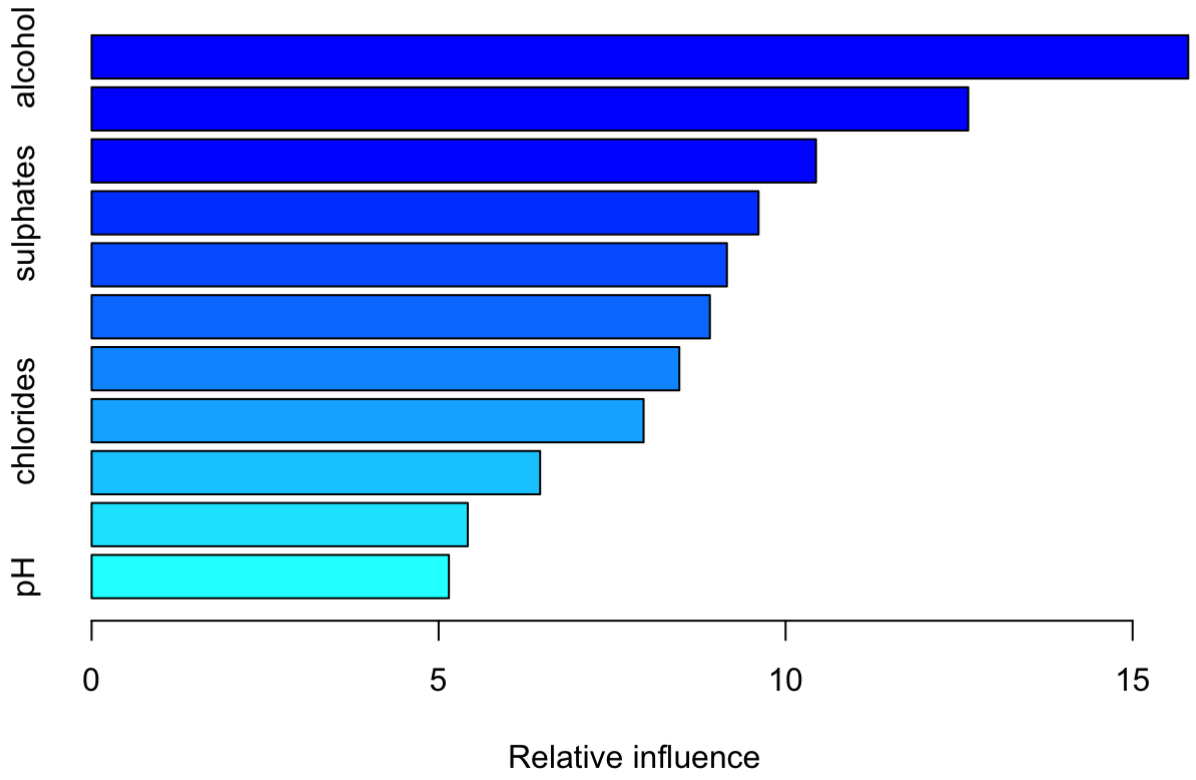
By lowering the number of trees to 100, it lowered the MSE significantly but the other boosting model performed better. Changing shrinkage value improved the model with 1000 tree iterations.

Boosting performed slightly worse than bagging and random forest

Extra code that was not used in final report but worth noting:

Boosting from the 3 classes of quality and not as numeric:

```
set.seed(1)
boost.red <- gbm(quality ~ .-quality, data = red_data2[train, ],
    distribution = "gaussian", n.trees = 1000,
    interaction.depth = 4)
summary(boost.red)
```




```
##                                       var   rel.inf
## alcohol                           alcohol 15.801807
## volatile.acidity         volatile.acidity 12.631802
## fixed.acidity               fixed.acidity 10.438198
## sulphates                       sulphates  9.610511
## total.sulfur.dioxide total.sulfur.dioxide  9.154742
## citric.acid                   citric.acid  8.909303
## density                           density  8.468875
## chlorides                       chlorides  7.953861
## residual.sugar             residual.sugar  6.462526
## free.sulfur.dioxide   free.sulfur.dioxide  5.420195
## pH                                    pH  5.148181
```

Again we can see that alcohol and volatile acidity are the two most important variables.

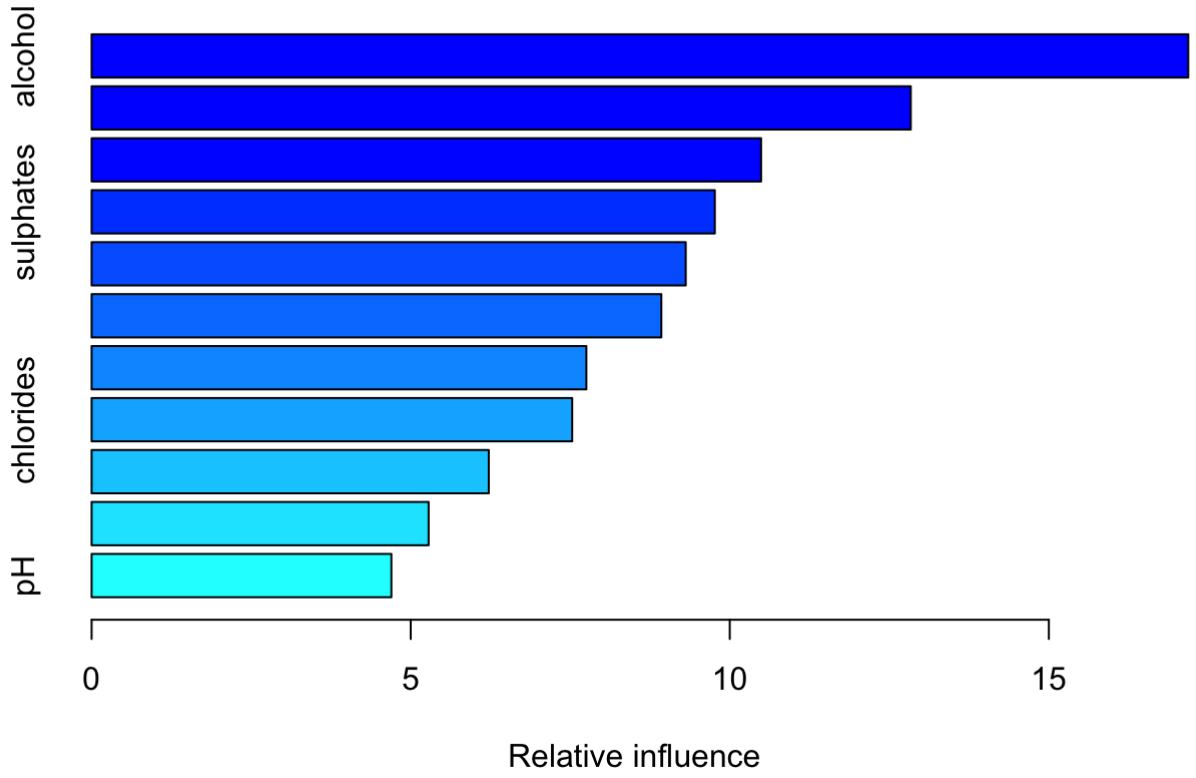
Test MSE:

```
yhat.boost <- predict(boost.red,
    newdata = red_data2[-train, ], n.trees = 1000)
mean((yhat.boost - test.num)^2)
```

```
## [1] 0.3773565
```

Changing shrinkage:

```
set.seed(1)
boost.red <- gbm(quality ~ .-quality, data = red_data2[train, ],
    distribution = "gaussian", n.trees = 500,
    interaction.depth = 4)
summary(boost.red)
```




```
##                                   var   rel.inf
## alcohol                       alcohol 17.184376
## volatile.acidity     volatile.acidity 12.836519
## fixed.acidity           fixed.acidity 10.491141
## sulphates                   sulphates  9.766140
## total.sulfur.dioxide total.sulfur.dioxide  9.310044
## citric.acid               citric.acid  8.927618
## density                       density  7.752475
## chlorides                   chlorides  7.530528
## residual.sugar         residual.sugar  6.223357
## free.sulfur.dioxide free.sulfur.dioxide  5.280633
## pH                                 pH  4.697170
```

```
yhat.boost <- predict(boost.red,
    newdata = red_data2[-train, ], n.trees = 500)
mean((yhat.boost - test.num)^2)
```

```
## [1] 0.3585649
```

It performed slightly better when changing the number of trees.

Boosting performed better than bagging and random forest, but the distribution doesn't align correctly with categorical quality data.