Wine Dataset - SAP.IO Challenge

Mudit Goyal

11/7/2017

Introduction

I was given a dataset to work on as part of a coding challenge for SAP.iO recruitment! It's a list of ~6000 wines, both white and red. I had to split it up into two seperate ones because R is not the best with memory allocation and my computer simply could not run it. This problem can be alleviated in the future by putting it in an AWS instance and use their cloud computers to be able to better run these models.

I'm going to work with two models primarily for this - - The traditional K-nearest neighbors - randomForest. ### Pre-process + Exploring the data

The first thing I did is go into the CSV file itself is to split the red and white wines into two different datasets.

We will start with the red wine dataset.

```
# The below lines are to set up R so it uses all of my
# computer's cores in order to run the models much quicker.
library(doParallel)
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
registerDoParallel(cores = detectCores() - 1)

# Set seed is useful for creating simluations
set.seed(10)

# Loading all the required libraries for my analysis
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
## Warning in as.POSIXlt.POSIXct(Sys.time()): unknown timezone 'zone/tz/2017c.
## 1.0/zoneinfo/America/Los_Angeles'
```

```
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
library(kknn)
```

```
##
## Attaching package: 'kknn'
```

```
## The following object is masked from 'package:caret':
##
##     contr.dummy
```

```
library(randomForest)
```

```
## randomForest 4.6-12
```

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

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(kernlab)
```

```
##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
# Reading the data
df <- read.csv("red.csv")

# Changing NA's to 0's.
df[is.na(df)] <- 0
str(df)
```

```
## 'data.frame':    1599 obs. of  14 variables:
##  $ fixed.acidity       : num  7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5 ...
##  $ volatile.acidity    : num  0.7 0 0.76 0.28 0.7 0.66 0.6 0.65 0.58 0.5 ...
##  $ citric.acid         : num  0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...
##  $ astringency.rating  : num  0.81 0.86 0.85 1.14 0.81 0.8 0.85 0.79 0.83 0.8 ...
##  $ residual.sugar      : num  1.9 2.6 2.3 0 0 1.8 1.6 1.2 2 6.1 ...
##  $ chlorides           : num  0.076 0.098 0.092 0.075 0.076 0.075 0.069 0.065 0.073 0.071 ...
##  $ free.sulfur.dioxide : num  11 25 15 17 11 13 15 15 9 17 ...
##  $ total.sulfur.dioxide: num  34 67 54 60 34 40 59 21 18 102 ...
##  $ density             : num  0.998 0.997 0.997 0.998 0.998 ...
##  $ pH                  : num  3.51 3.2 3.26 3.16 3.51 3.51 3.3 3.39 3.36 3.35 ...
##  $ sulphates           : num  0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.47 0.57 0.8 ...
##  $ alcohol             : num  9.4 9.8 9.8 9.8 9.4 9.4 9.4 10 9.5 10.5 ...
##  $ vintage             : num  2001 2003 2006 2003 2004 ...
##  $ quality             : int  5 5 5 6 5 5 5 7 7 5 ...
```

Running str(df) displays the internal structure of the red wine dataset. It shows that there are 1599 samples and 14 different variables.

I know want to check for any class imbalances. I'm going to run this following function to check for any class imbalances. We'll need enough samples of these classes to be able to effectively split the data into useable training and test sets and perform a cross-valdation.

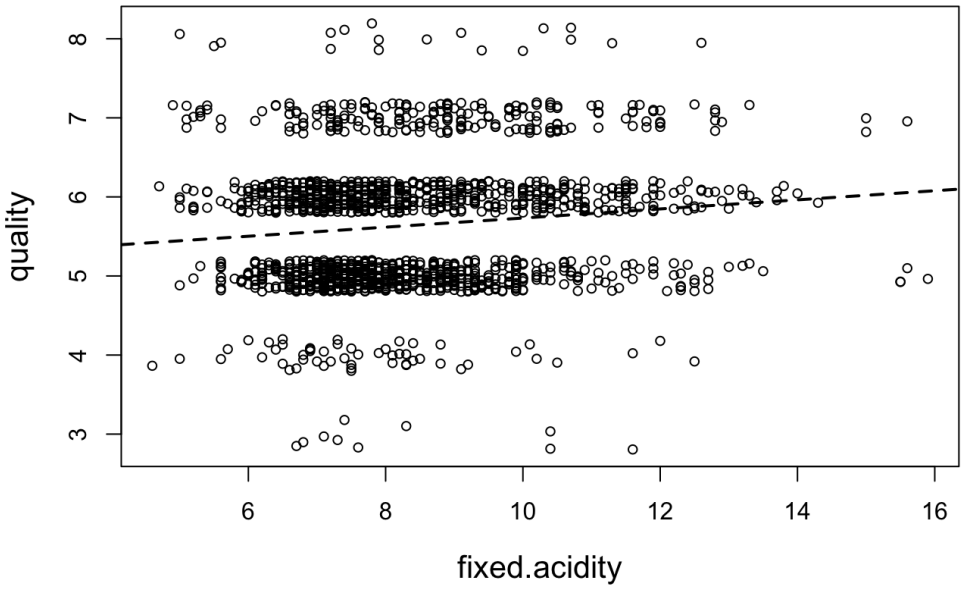
```
table(df$quality)
```

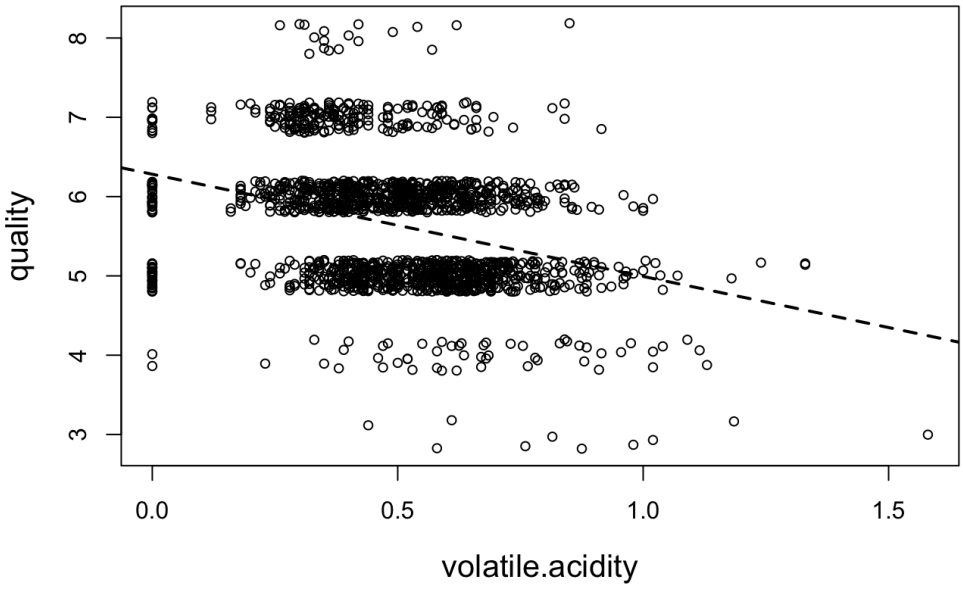
```
##
##   3   4   5   6   7   8
##  10  53 681 638 199  18
```

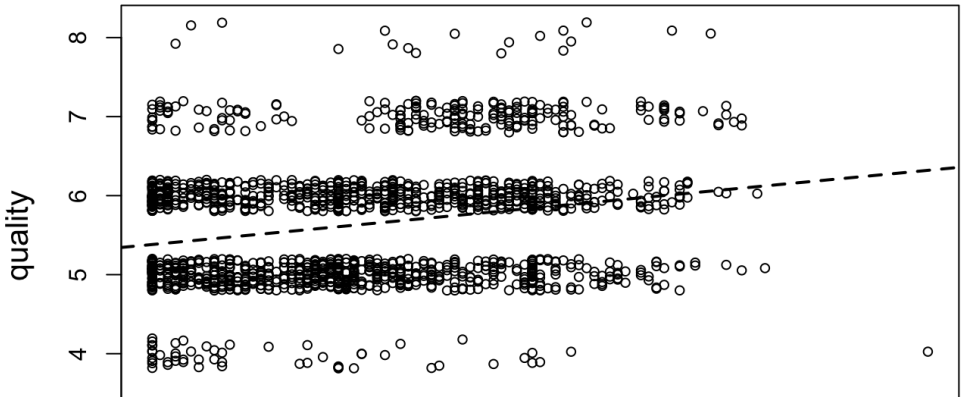
Looking at the information above it's clear that there is indeed a large class impabalance. There's almost 1600 samples but there's only about 10 for the 3 class and just **18** for the 8 class. To try and combat the imbalances with the classes, we'll need to merge some of them.

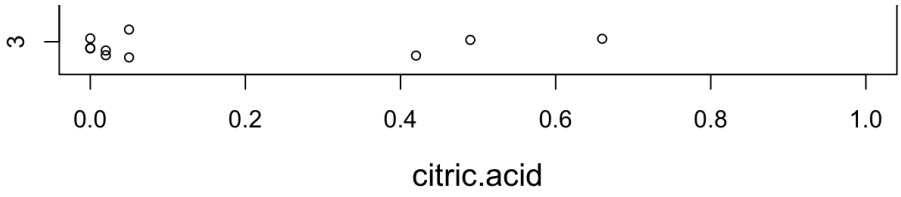
We now going to visualize the data using plots for each of the predictor variables.

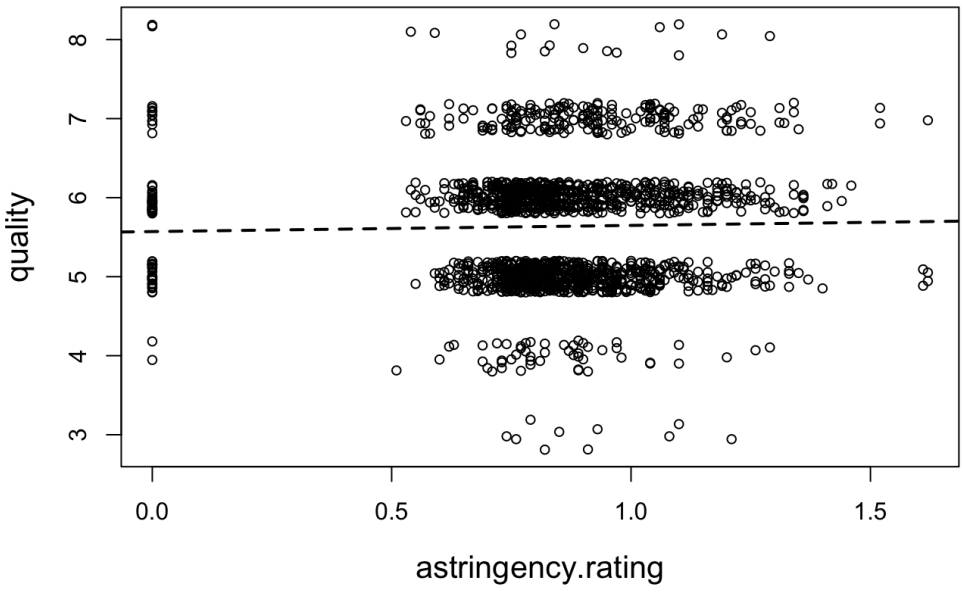
```
# making all of them show in one grid
for (i in c(1:12)) {
    plot(df[, i], jitter(df[, "quality"]), xlab = names(df)[i],
         ylab = "quality", cex = 0.8, cex.lab = 1.3)
    abline(lm(df[, "quality"] ~ df[ ,i]), lty = 2, lwd = 2)
}
```

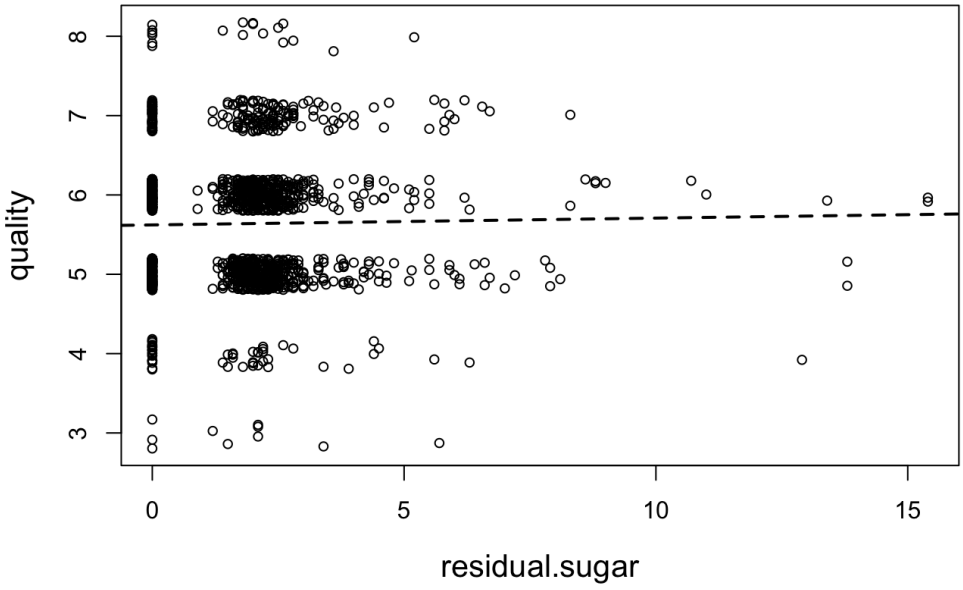

quality
fixed.acidity
3
4
5
6
7
8
6
8
10
12
14
16

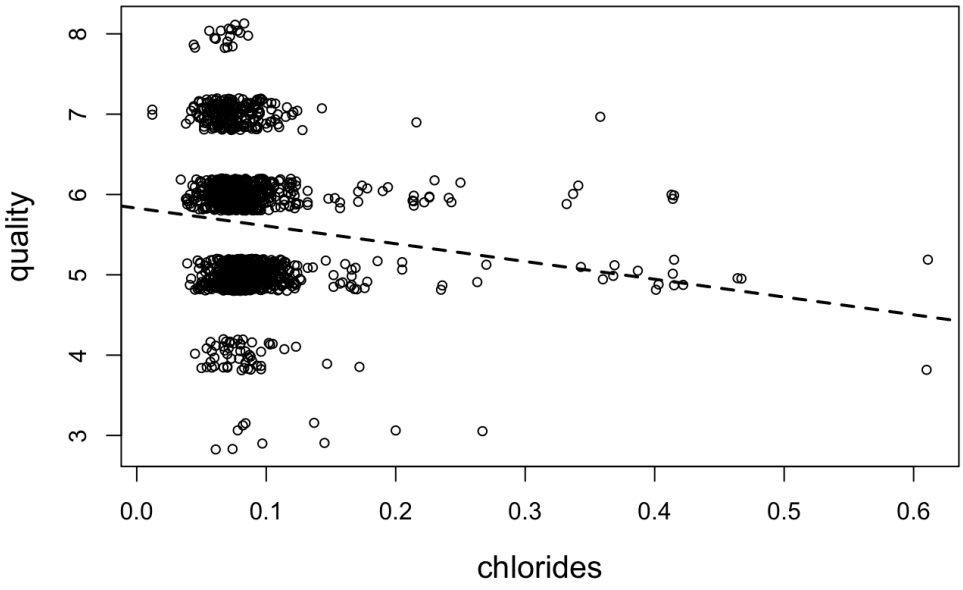

quality
volatile.acidity
3
4
5
6
7
8
0.0
0.5
1.0
1.5

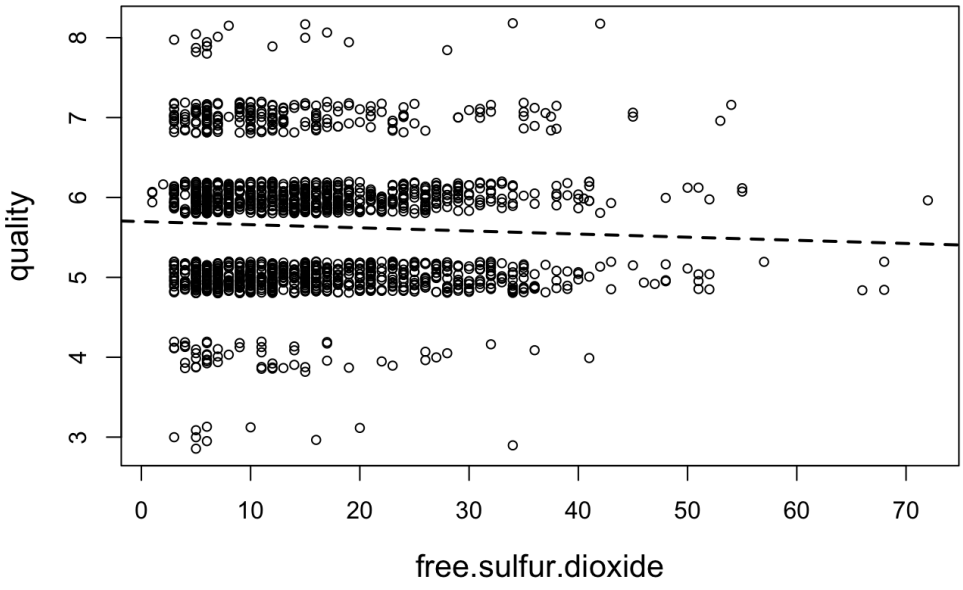

quality
4
5
6
7
8

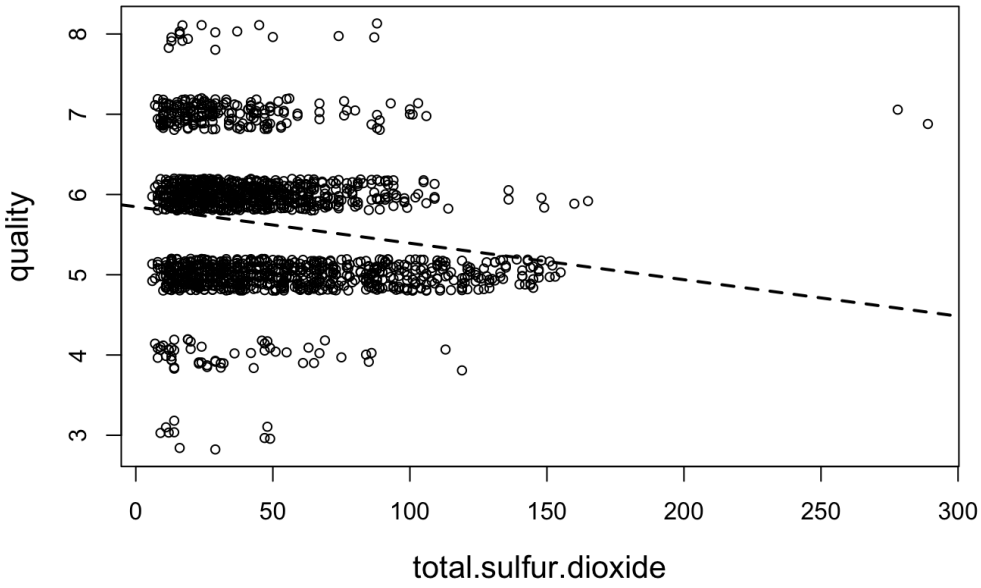

3
0.0
0.2
0.4
0.6
0.8
1.0
citric.acid

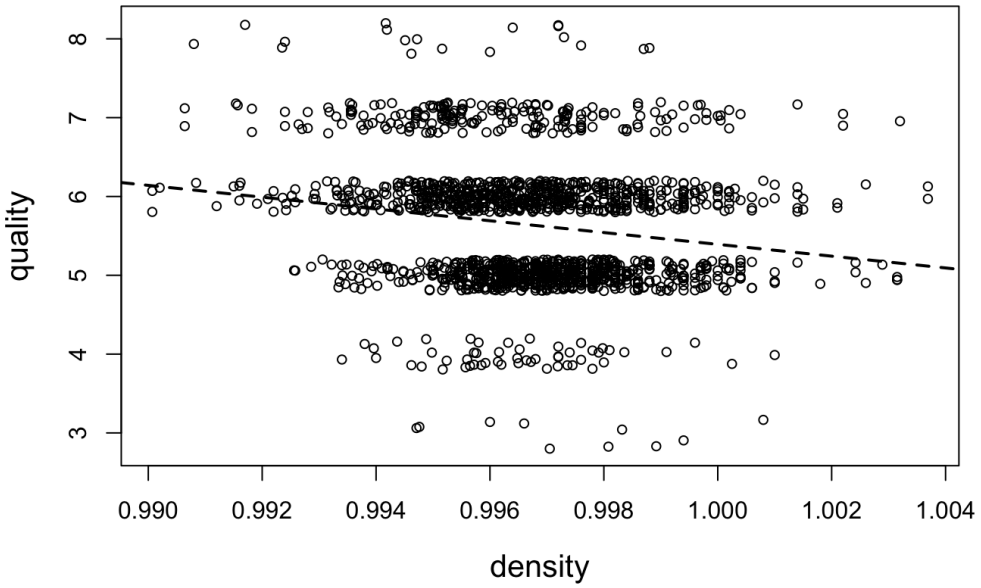

quality
3
4
5
6
7
8
0.0
0.5
1.0
1.5
astringency.rating

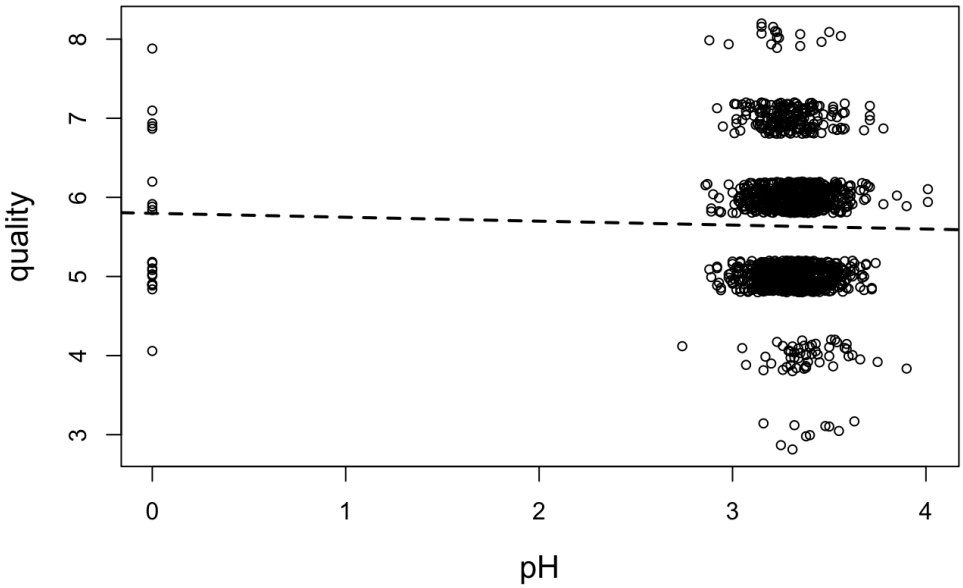

quality
3
4
5
6
7
8
0
5
10
15
residual.sugar

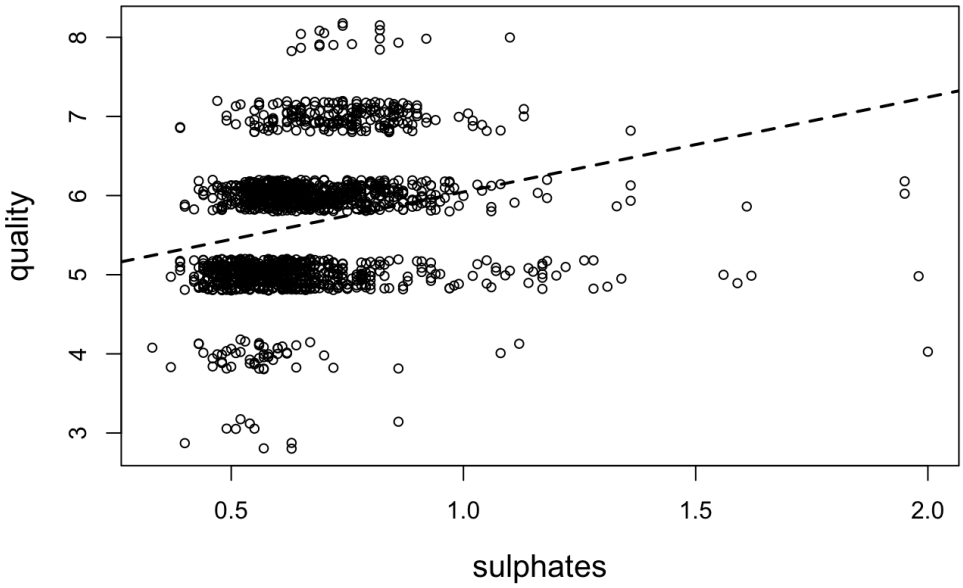

quality
chlorides
3
4
5
6
7
8
0.0
0.1
0.2
0.3
0.4
0.5
0.6

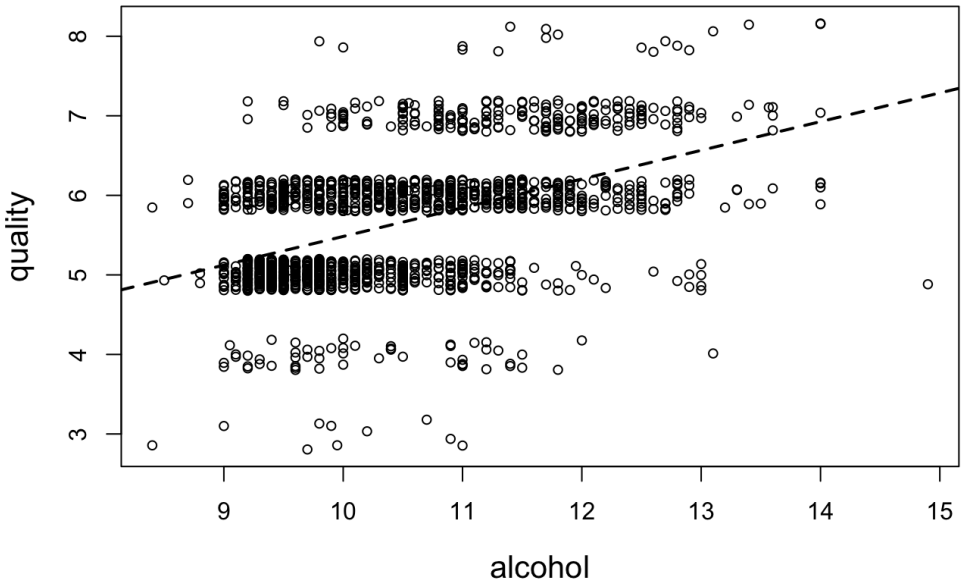

quality
free.sulfur.dioxide
3
4
5
6
7
8
0
10
20
30
40
50
60
70


quality
3
4
5
6
7
8
0
50
100
150
200
250
300
total.sulfur.dioxide


quality
3
4
5
6
7
8
0.990
0.992
0.994
0.996
0.998
1.000
1.002
1.004
density


quality
pH
3
4
5
6
7
8
0
1
2
3
4


quality
sulphates
3
4
5
6
7
8
0.5
1.0
1.5
2.0



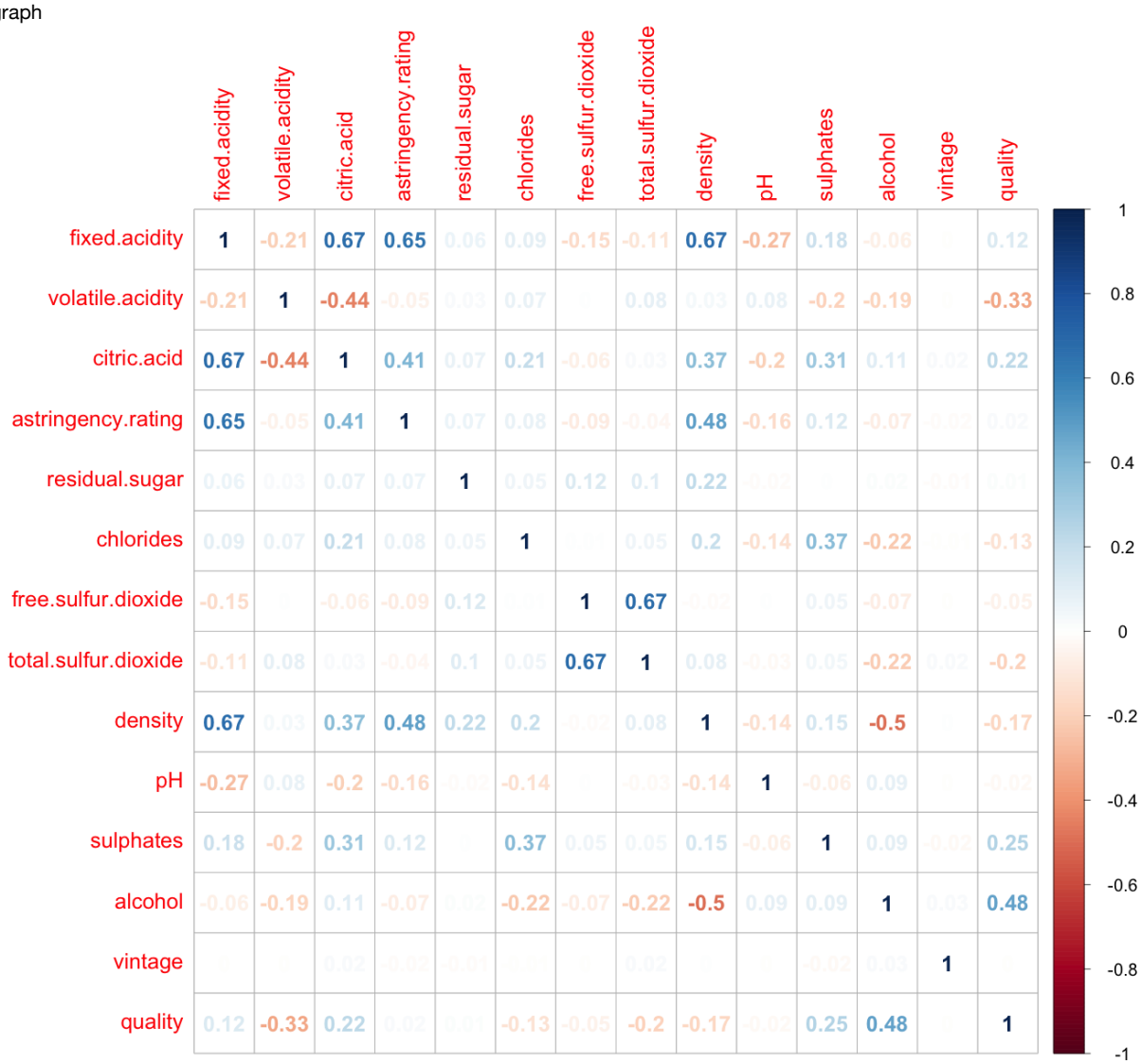
The line on each of these plots displays the linear regression of our response variable **quality** as a function of each of the predictor variables. When looking at each of the plots the first thing that you see are the presence of numerous outliers. For example, there's a very glaring outlier in the total sulfur dioxide plot, as well as in the density plot. We are going to remove this one from the dataset.

```
max.sug <- which(df$total.sulfur.dioxide == max(df$total.sulfur.dioxide))
df <- df[-max.sug, ]
```

We can see that a few of the regression lines show a very weak association to our response variable. We'll later split into training and test sets and then we can figure out if we want to keep those features or remove them.

```
cor.df <- cor(df)
# Had some trouble displaying the graph, so going to save as .png and
# then show in the R markdown file.
png(height = 1200, width = 1500, pointsize = 25, file = 'red_cor_plot.png')
corrplot(cor.df, method = 'number')
```

Here's our graph



You can see the weak relationships here between quality and citric.acid, free.sulplur dioxide, and also sulphates as shown in the plot. After processing through the data, we can continue on and say that non-linear classification models will be more appropriate than regression.

==========

Building the Model

We **need** to convert our response variable to factor, and then do the split into training and testing sets.

```
df$quality <- as.factor(df$quality)
inTrain <- createDataPartition(df$quality, p = 2/3, list = F)
train.df <- df[inTrain,]
test.df <- df[-inTrain,]
```

We are going to go about this using both k-nearest neighors (KNN), along with randomForest. We will use the caret function which we loaded earlier to tune the model that we can use with the train function. We'll repeat 5 times.

The Caret

I chose to use this library because it really helps to simplify model tuning. We can use the tuneGrid argument, which is a grid of all the hyperparameters we'd want to use to tune the model which we'll then pass into the train function.

Feature Selection

As said above we said that we would decide to use non-linear feature selection methods since there are a few factors that have very weak correlations with our response variable quality. Most feature selection methods would retain all the predictors / excluded 1 at the most - so we are not going to be using feature selection while we train and tune our models.

The Preprocessing

KNN uses distance, so we need to make sure all the predictor variables are standardized. We will use the preProcess argument in the train function for this.

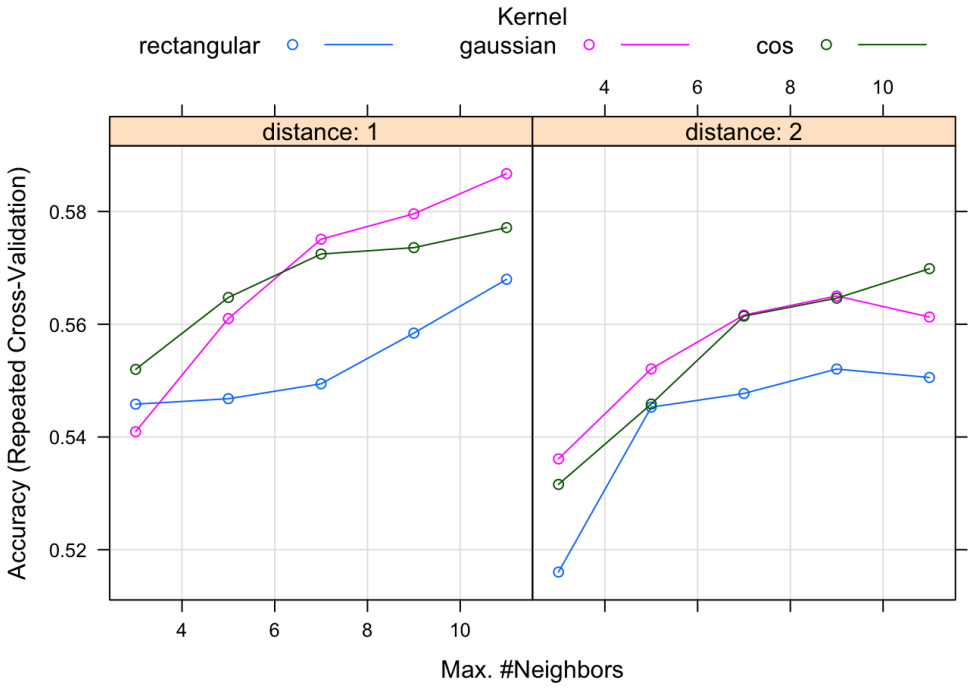
KNN

For KNN, we'll use 5 kmax, 2 distance, and 3 kernel values. For the distance, 1 is the Manhattan distance, and 2 is the Euclidian distance.

```
library(e1071)
t.ctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 5)

knn.grid <- expand.grid(kmax = c(3, 5, 7, 9, 11), distance = c(1, 2),
                        kernel = c("rectangular", "gaussian", "cos"))

knn.train <- train(quality ~ ., data = train.df, method = "knnn",
                   trControl = t.ctrl, tuneGrid = knn.grid,
                   preProcess = c("center", "scale"))
plot(knn.train)
```




```
knn.train$bestTune
```

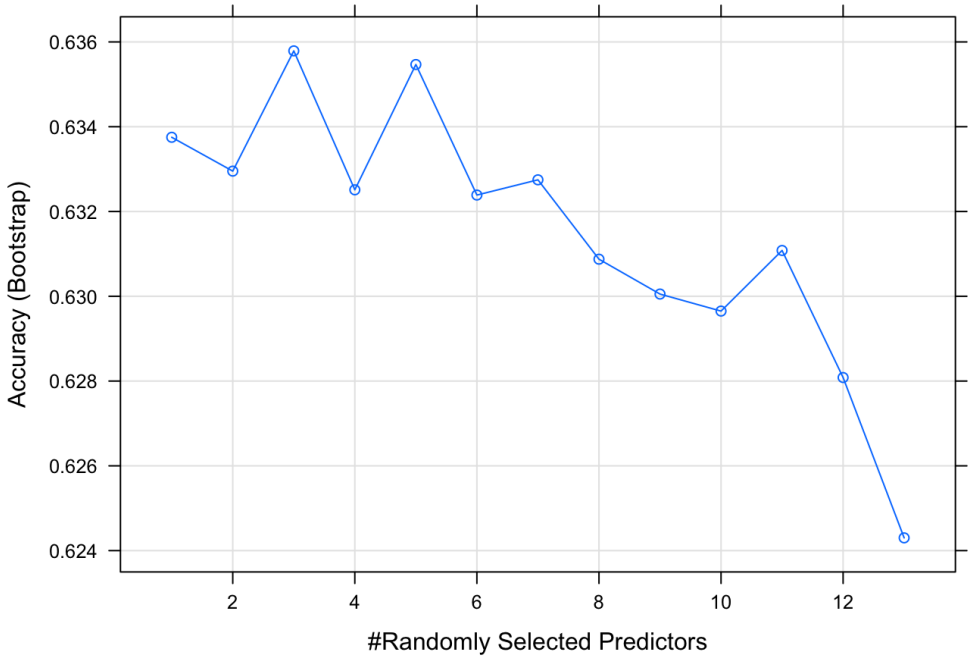
```
##    kmax distance   kernel
## 26   11        1 gaussian
```

The best value for k is 1, after the three repetitions.

The randomForest model.

For Rf, the only parameter that we can mess around with is ***mtry***, which is the number of vars which are randomly sampled at each split. We'll try values of 1 through 13 to pass through the tuneGrid arguement.

```
rf.grid <- expand.grid(mtry = 1:13)
rf.train <- train(quality ~ ., data = train.df, method = "rf",
                  trcontrol = t.ctrl, tuneGrid = rf.grid,
                  preProcess = c("center", "scale"))
plot(rf.train)
```




```
rf.train$bestTune
```

```
##   mtry
## 3    3
```

A mtry of 5 is the best value to use here.

The model selection

```
knn.predict <- predict(knn.train, test.df)
confusionMatrix(knn.predict, test.df$quality)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8
##          3   0   0   0   0   0   0
##          4   0   0   1   0   0   0
##          5   0  11 166  63   8   0
##          6   3   5  56 128  36   0
##          7   0   1   4  21  22   6
##          8   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.5951
##                  95% CI : (0.552, 0.6372)
##     No Information Rate : 0.4275
##     P-Value [Acc > NIR] : 6.267e-15
##
##                   Kappa : 0.3429
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity           0.00000 0.000000   0.7313   0.6038  0.33333   0.0000
## Specificity           1.00000 0.998054   0.7303   0.6865  0.93118   1.0000
## Pos Pred Value            NaN 0.000000   0.6694   0.5614  0.40741      NaN
## Neg Pred Value        0.99435 0.967925   0.7845   0.7228  0.90776   0.9887
## Prevalence            0.00565 0.032015   0.4275   0.3992  0.12429   0.0113
## Detection Rate        0.00000 0.000000   0.3126   0.2411  0.04143   0.0000
## Detection Prevalence  0.00000 0.001883   0.4670   0.4294  0.10169   0.0000
## Balanced Accuracy     0.50000 0.499027   0.7308   0.6451  0.63226   0.5000
```

```
rf.predict <- predict(rf.train, test.df)
confusionMatrix(rf.predict, test.df$quality)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8
##          3   0   0   0   0   0   0
##          4   0   0   0   0   0   0
##          5   1  12 187  54   6   0
##          6   2   5  39 143  30   3
##          7   0   0   1  15  30   3
##          8   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.678
##                  95% CI : (0.6364, 0.7176)
##     No Information Rate : 0.4275
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4741
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity           0.00000  0.00000   0.8238   0.6745  0.45455   0.0000
## Specificity           1.00000  1.00000   0.7599   0.7524  0.95914   1.0000
## Pos Pred Value            NaN      NaN   0.7192   0.6441  0.61224      NaN
## Neg Pred Value        0.99435  0.96798   0.8524   0.7767  0.92531   0.9887
## Prevalence            0.00565  0.03202   0.4275   0.3992  0.12429   0.0113
## Detection Rate        0.00000  0.00000   0.3522   0.2693  0.05650   0.0000
## Detection Prevalence  0.00000  0.00000   0.4896   0.4181  0.09228   0.0000
## Balanced Accuracy     0.50000  0.50000   0.7918   0.7134  0.70684   0.5000
```

For the red wine dataset, the Random Forest Model was the one which performed the best, with an accuracy of almost 70% with a strong Kappa of .5055. The KNN was not better or worse.

==========

Next, the white wine data set

```
library(doParallel)
registerDoParallel(cores = detectCores() - 1)

set.seed(10)
library(caret)
library(corrplot)
library(kknn)
library(randomForest)
library(kernlab)

df1 <- read.csv("white.csv")
# changing NA's to 0's.
df1[is.na(df1)] <- 0
str(df1)
```

```
## 'data.frame':    4898 obs. of  14 variables:
##  $ fixed.acidity       : num  7 6.3 8.1 7.2 7.2 8.1 6.2 7 6.3 8.1 ...
##  $ volatile.acidity    : num  0.27 0.3 0.28 0.23 0.23 0.28 0.32 0.27 0.3 0.22 ...
##  $ citric.acid         : num  0.36 0.34 0.4 0.32 0.32 0.4 0.16 0.36 0.34 0.43 ...
##  $ astringency.rating  : num  0.72 0.66 0.83 0.74 0.74 0.83 0.65 0.72 0.66 0.83 ...
##  $ residual.sugar      : num  0 0 6.9 0 8.5 6.9 7 0 1.6 1.5 ...
##  $ chlorides           : num  0.045 0.049 0.05 0.058 0.058 0.05 0.045 0.045 0.049 0.044 ...
##  $ free.sulfur.dioxide : num  45 14 30 47 47 30 30 45 14 28 ...
##  $ total.sulfur.dioxide: num  170 132 97 186 186 97 136 170 132 129 ...
##  $ density             : num  1.001 0.994 0.995 0.996 0.996 ...
##  $ pH                  : num  3 3.3 3.26 3.19 3.19 3.26 3.18 3 3.3 3.22 ...
##  $ sulphates           : num  0.45 0.49 0.44 0.4 0.4 0.44 0.47 0.45 0.49 0.45 ...
##  $ alcohol             : num  8.8 9.5 10.1 9.9 9.9 10.1 9.6 8.8 9.5 11 ...
##  $ vintage             : num  2004 2004 2006 2004 2007 ...
##  $ quality             : int  6 6 6 6 6 6 6 6 6 6 ...
```

Running str(df) on the wine dataset shows that there are 4898 samples, and 14 different variables.

I know want to check for any class imbalances. I'm going to run this following function to check for any class imbalances. We'll need enough samples of these classes to be able to effectively split the data into useable training and test sets and perform a cross-valdation.

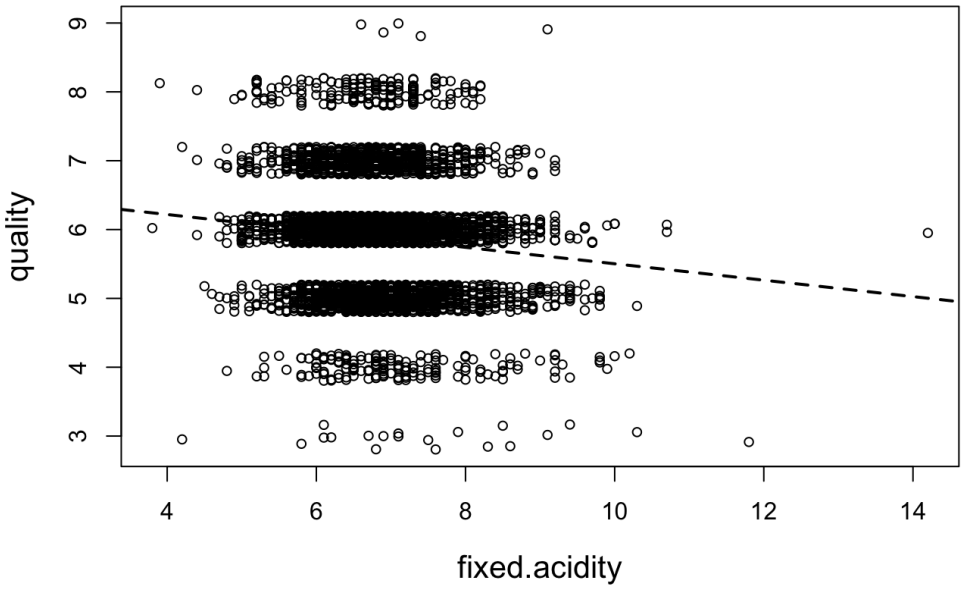
```
table(df1$quality)
```

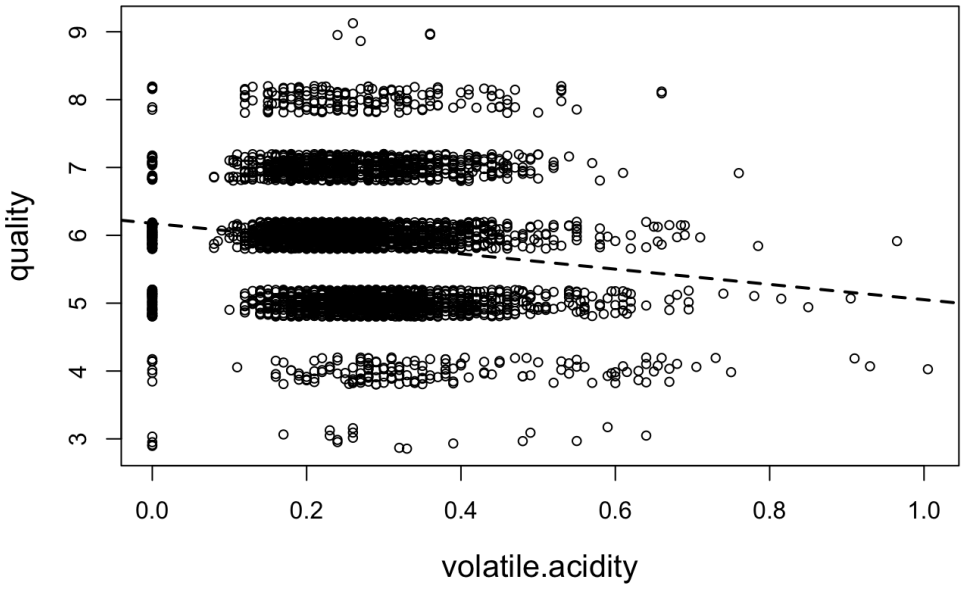
```
##
##    3    4    5    6    7    8    9
##   20  163 1457 2198  880  175    5
```

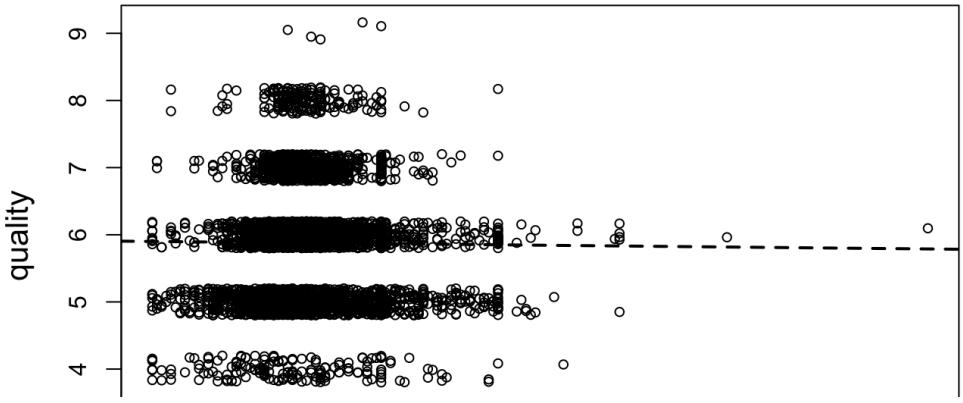
Looking at the information above it's clear that there is indeed a large class impabalance. There's almost 1600 samples but there's only 20 for the 3 class and just **5** for the 9 class. To try and combat the imbalances with the classes, we'll need to merge some of them.

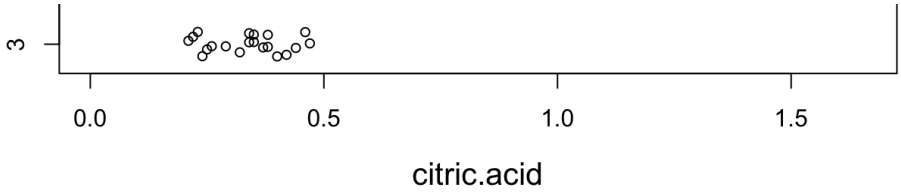
Now going to visualize the data using plots for each of the predictor variables.

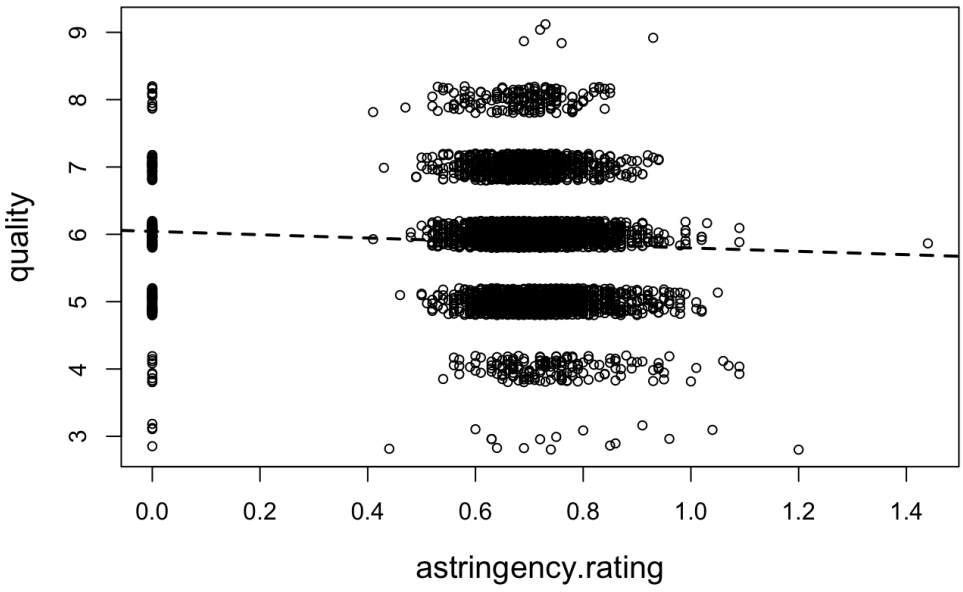
```
# omitting the vintage v quality graph
for (i in c(1:12)) {
    plot(df1[, i], jitter(df1[, "quality"]), xlab = names(df1)[i],
         ylab = "quality", cex = 0.8, cex.lab = 1.3)
    abline(lm(df1[, "quality"] ~ df1[ ,i]), lty = 2, lwd = 2)
}
```

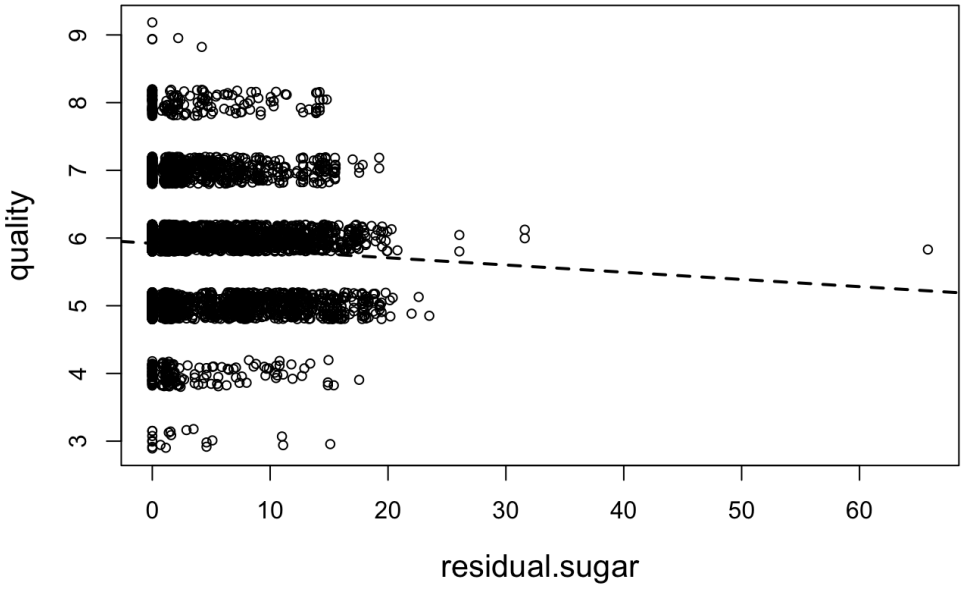

quality
fixed.acidity
3
4
5
6
7
8
9
4
6
8
10
12
14

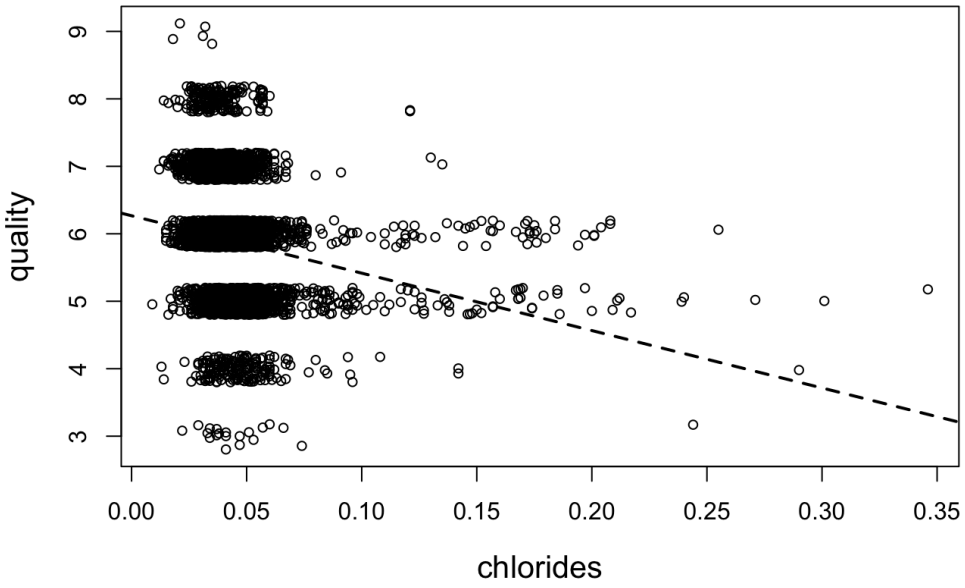

quality
volatile.acidity
3
4
5
6
7
8
9
0.0
0.2
0.4
0.6
0.8
1.0

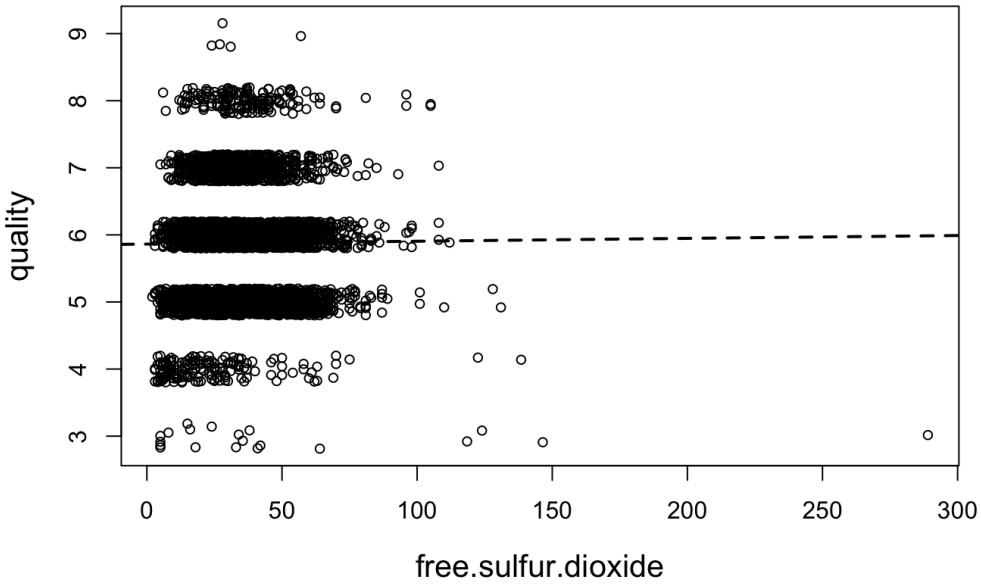

quality
4
5
6
7
8
9

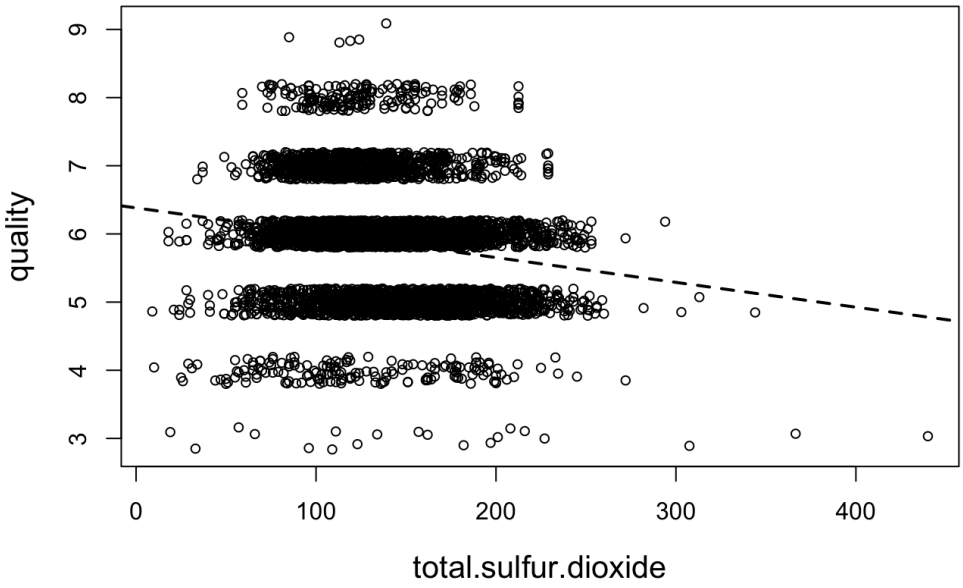

3
0.0
0.5
1.0
1.5
citric.acid

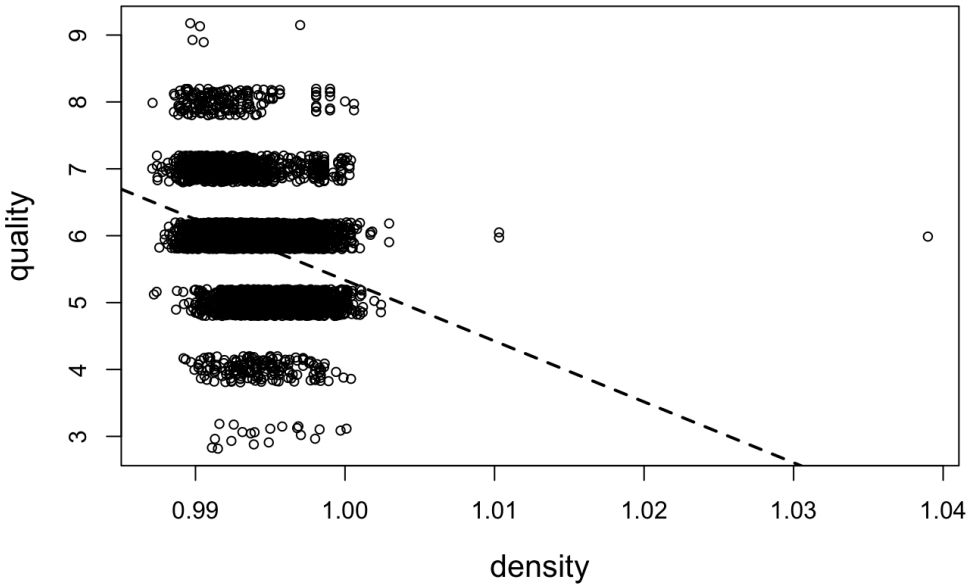

quality
3
4
5
6
7
8
9
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
astringency.rating

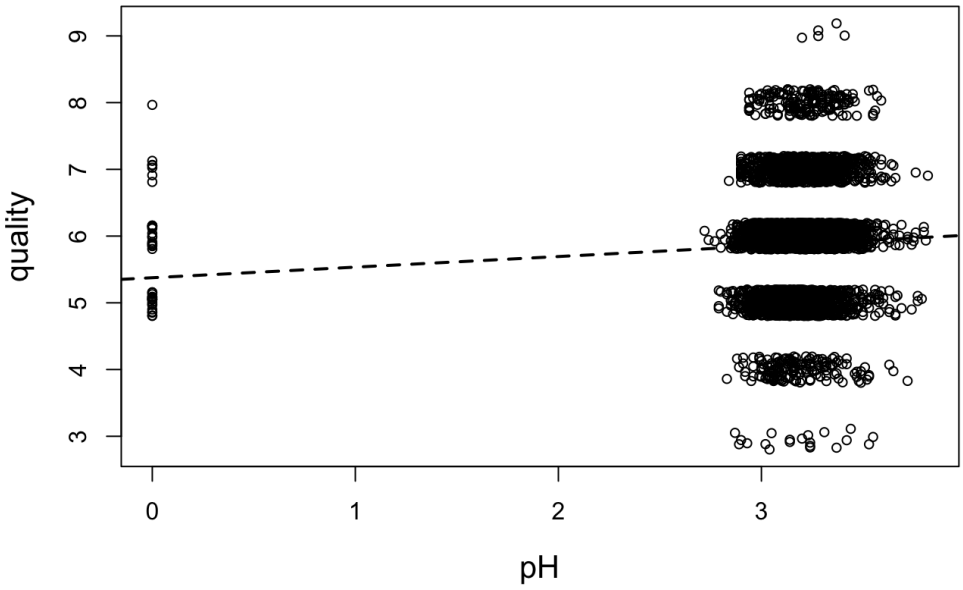

quality
3
4
5
6
7
8
9
0
10
20
30
40
50
60
residual.sugar

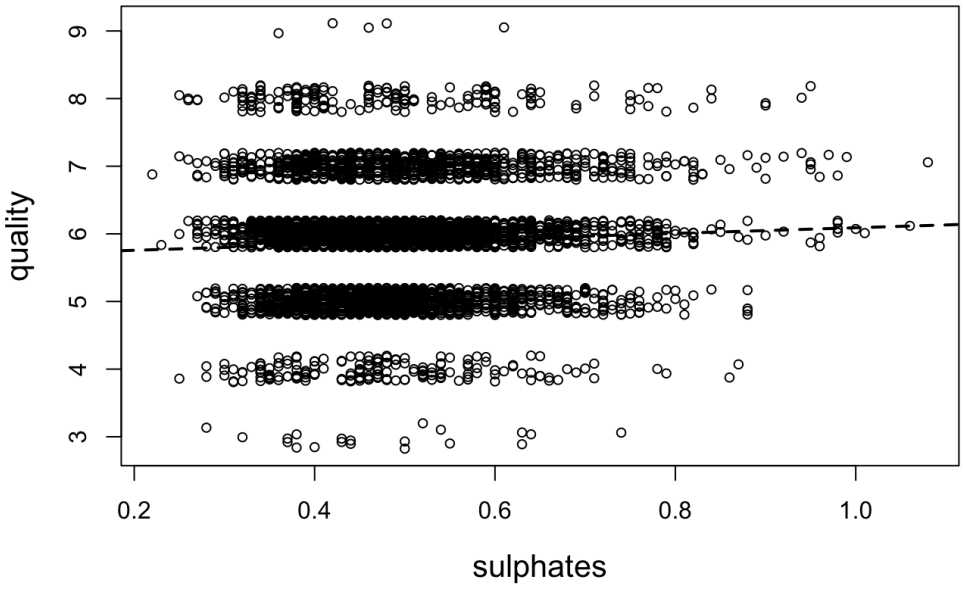

quality
chlorides
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
3
4
5
6
7
8
9

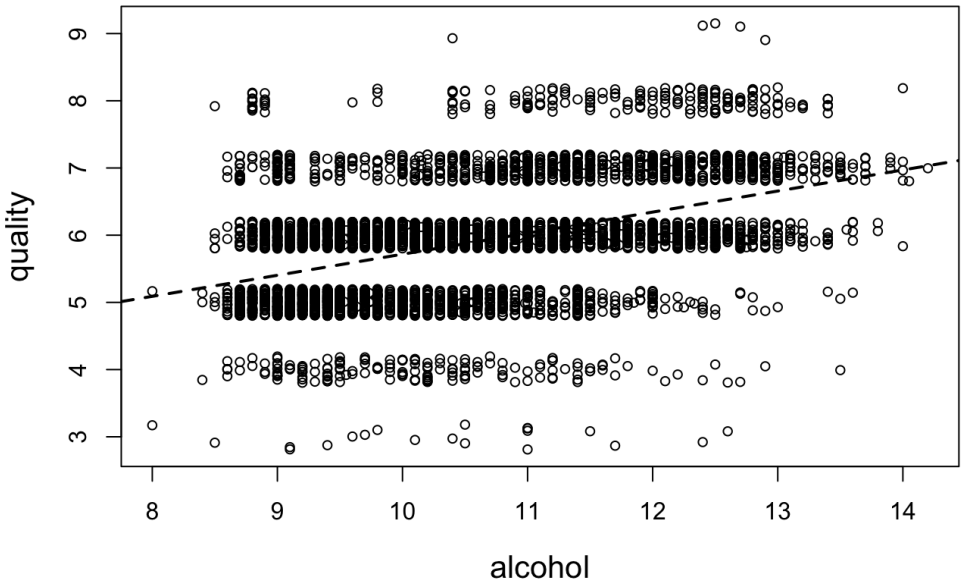

quality
free.sulfur.dioxide
0
50
100
150
200
250
300
3
4
5
6
7
8
9


quality
3
4
5
6
7
8
9
0
100
200
300
400
total.sulfur.dioxide


quality
3
4
5
6
7
8
9
0.99
1.00
1.01
1.02
1.03
1.04
density


quality
pH
3
4
5
6
7
8
9
0
1
2
3


quality
sulphates
3
4
5
6
7
8
9
0.2
0.4
0.6
0.8
1.0



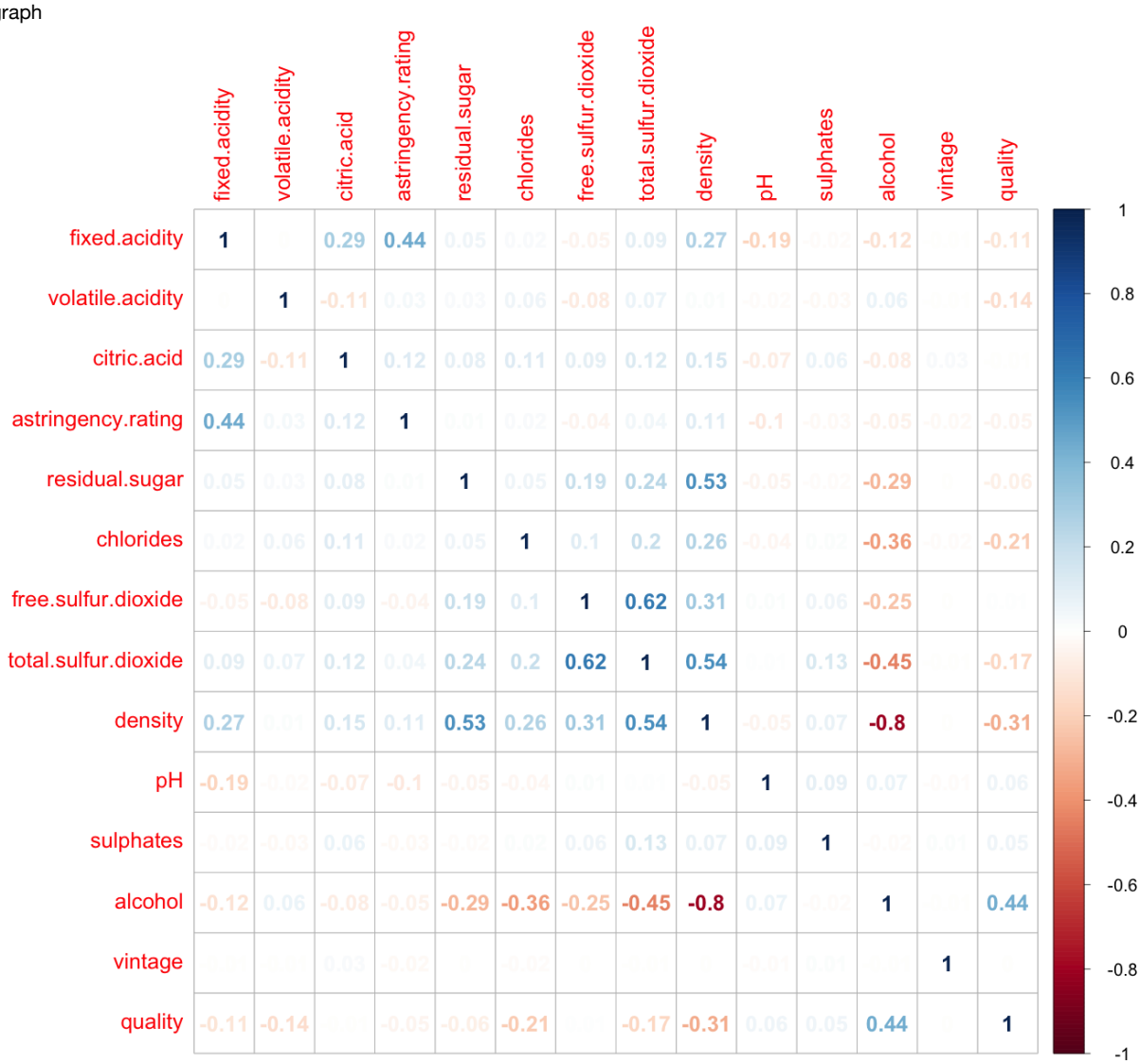
The line on each of these plots displays the linear regression of our response variable **quality** as a function of each of the predictor variables. When looking at each of the plots the first thing that you see are the presence of numerous outliers. For example, there's a very glaring outlier in the residual sugar plot, as well as in the density plot. We are going to remove this one from the dataset.

```
max.sug1 <- which(df1$residual.sugar == max(df1$residual.sugar))
df1 <- df1[-max.sug1, ]
```

Again, there are a few regression lines which show a very weak association. Like before, ee first split into training and test sets and then we can figure out if we want to keep those features or remove them.

```
par(mfrow = c(1,1))
cor.df1 <- cor(df1)
png(height = 1200, width = 1500, pointsize = 25, file = 'white_cor_plot.png')
corrplot(cor.df1, method = 'number')
```

Here's our graph



You can see the weak relationships here between quality, citric acid, residual sugar, free.sulplur dioxide, and also sulphates as shown in the plot. After looking at this data, after processing through the data, we can continue on and say that non-linear classification models will be more appropriate than regression.

==========

Building the Model

We **need** to convert our response variable to factor, and then do the split into training and testing sets.

```
df1$quality <- as.factor(df1$quality)
inTrain1 <- createDataPartition(df1$quality, p = 2/3, list = F)
train.df1 <- df1[inTrain1,]
test.df1 <- df1[-inTrain1,]
```

We are going to go about this using both k-nearest neighors (KNN), along with randomForest. We will use the caret function which we loaded earlier to tune the model that we can use with the train function. We'll repeat 3 times.

The Caret

I chose to use this library because it really helps to simplify model tuning. We can use the tuneGrid argument, which is a grid of all the hyperparameters we'd want to use to tune the model which we'll then pass into the train function.

Feature Selection

As said above we said that we would decide to use non-linear feature selection methods since there are a few factors that have very weak correlations with our response variable quality. Most feature selection methods would retain all the predictors / excluded 1 at the most - so we are not going to be using feature selection while we train and tune our models.

Preprocessing

KNN uses distance, so we need to make sure all the predictor variables are standardized. We will use the preProcess argument in the train function for this.

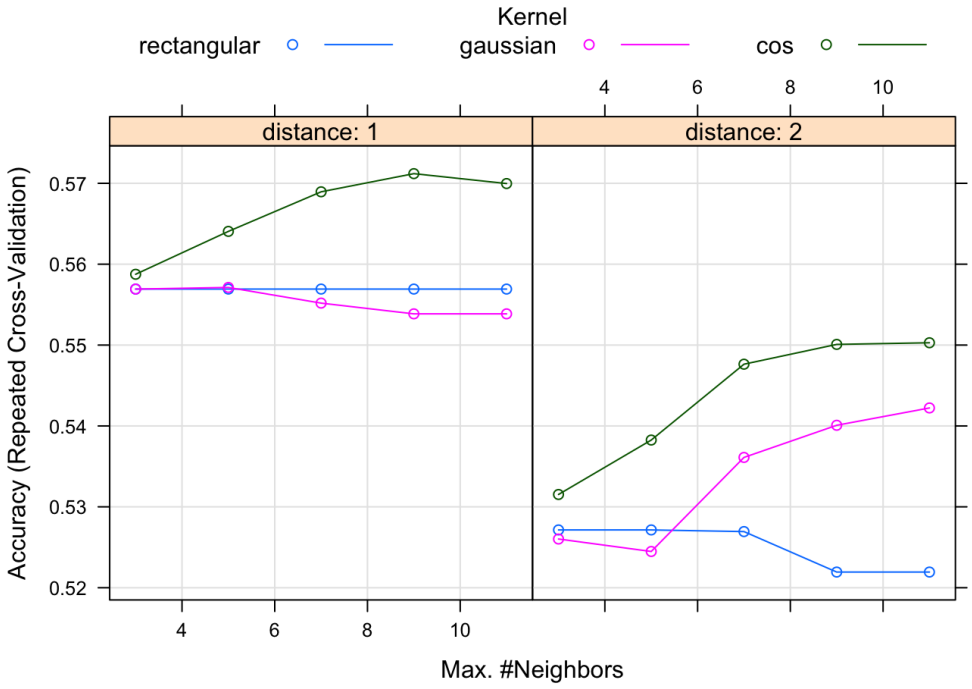
KNN

For KNN, we'll use 5 kmax, 2 distance, and 3 kernel values. For the distance, 1 is the Manhattan distance, and 2 is the Euclidian distance.

```
library(e1071)
t.ctrl1 <- trainControl(method = "repeatedcv", number = 5, repeats = 3)

knn.grid1 <- expand.grid(kmax = c(3, 5, 7, 9, 11), distance = c(1, 2),
                         kernel = c("rectangular", "gaussian", "cos"))

knn.train1 <- train(quality ~ ., data = train.df1, method = "kknn",
                    trControl = t.ctrl1, tuneGrid = knn.grid1,
                    preProcess = c("center", "scale"))
plot(knn.train1)
```




```
knn.train1$bestTune
```

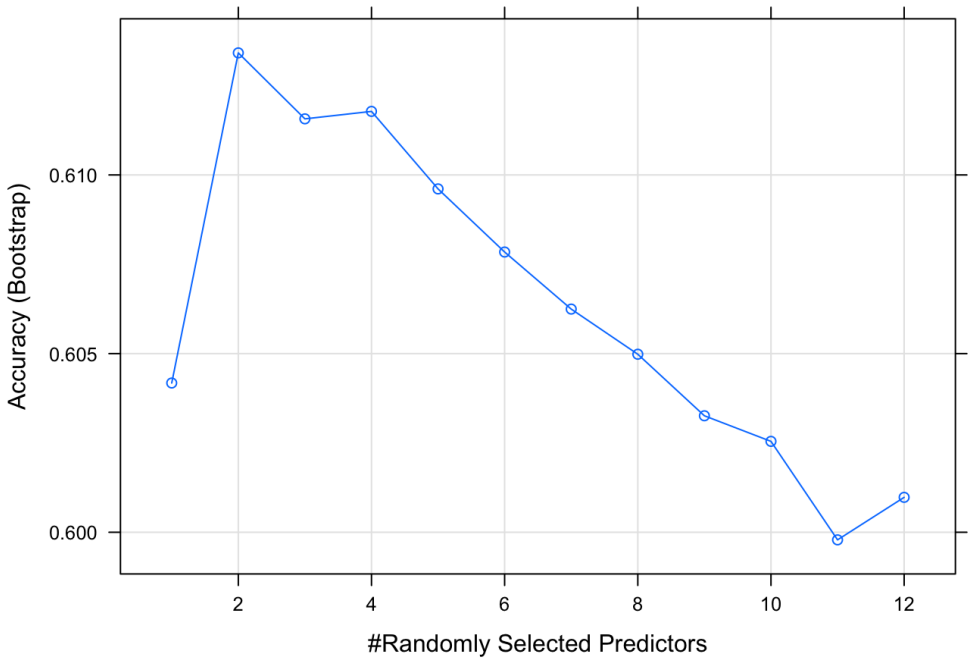
```
##    kmax distance kernel
## 21    9        1    cos
```

The best value for k is 9, after the three repetitions.

The randomForest model.

For this model, it seems that only the mtry (number of variables hyperparameter is of use to us. We'll pass mtry values of 1-12 into the train function's tuneGrid arg.

```
rf.grid1 <- expand.grid(mtry = 1:12)
rf.train1 <- train(quality ~ ., data = train.df1, method = "rf",
                   trcontrol = t.ctrl1, tuneGrid = rf.grid1,
                   preProcess = c("center", "scale"))
plot(rf.train1)
```




```
rf.train1$bestTune
```

```
##   mtry
## 2    2
```

A mtry of 2 is the best value to use here.

The Model Selection

```
knn.predict1 <- predict(knn.train1, test.df1)
confusionMatrix(knn.predict1, test.df1$quality)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8   9
##          3   0   0   1   0   0   0   0
##          4   0  10  11   1   0   0   0
##          5   4  23 293 135  14   1   0
##          6   1  20 158 494 117  18   0
##          7   1   1  21  97 149  24   1
##          8   0   0   1   5  13  15   0
##          9   0   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.5899
##                  95% CI : (0.5656, 0.6139)
##     No Information Rate : 0.4494
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3764
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                       Class: 3 Class: 4 Class: 5 Class: 6 Class: 7
## Sensitivity          0.0000000 0.185185   0.6041   0.6749  0.50853
## Specificity          0.9993839 0.992381   0.8453   0.6499  0.89147
## Pos Pred Value       0.0000000 0.454545   0.6234   0.6114  0.50680
## Neg Pred Value       0.9963145 0.972620   0.8343   0.7101  0.89213
## Prevalence           0.0036832 0.033149   0.2977   0.4494  0.17986
## Detection Rate       0.0000000 0.006139   0.1799   0.3033  0.09147
## Detection Prevalence 0.0006139 0.013505   0.2885   0.4960  0.18048
## Balanced Accuracy    0.4996919 0.588783   0.7247   0.6624  0.70000
##                      Class: 8  Class: 9
## Sensitivity          0.258621 0.0000000
## Specificity          0.987906 1.0000000
## Pos Pred Value       0.441176       NaN
## Neg Pred Value       0.973041 0.9993861
## Prevalence           0.035605 0.0006139
## Detection Rate       0.009208 0.0000000
## Detection Prevalence 0.020872 0.0000000
## Balanced Accuracy    0.623263 0.5000000
```

```
rf.predict1 <- predict(rf.train1, test.df1)
confusionMatrix(rf.predict1, test.df1$quality)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8   9
##          3   0   0   0   0   0   0   0
##          4   0   4   2   0   0   0   0
##          5   4  33 314 101   4   0   0
##          6   2  17 163 589 143  30   0
##          7   0   0   6  41 143  17   1
##          8   0   0   0   1   3  11   0
##          9   0   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.6513
##                  95% CI : (0.6276, 0.6745)
##     No Information Rate : 0.4494
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.449
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.000000 0.074074   0.6474   0.8046  0.48805 0.189655
## Specificity          1.000000 0.998730   0.8759   0.6042  0.95135 0.997454
## Pos Pred Value            NaN 0.666667   0.6886   0.6239  0.68750 0.733333
## Neg Pred Value       0.996317 0.969193   0.8542   0.7912  0.89444 0.970880
## Prevalence           0.003683 0.033149   0.2977   0.4494  0.17986 0.035605
## Detection Rate       0.000000 0.002455   0.1928   0.3616  0.08778 0.006753
## Detection Prevalence 0.000000 0.003683   0.2799   0.5795  0.12769 0.009208
## Balanced Accuracy    0.500000 0.536402   0.7616   0.7044  0.71970 0.593555
##                      Class: 9
## Sensitivity          0.0000000
## Specificity          1.0000000
## Pos Pred Value             NaN
## Neg Pred Value       0.9993861
## Prevalence           0.0006139
## Detection Rate       0.0000000
## Detection Prevalence 0.0000000
## Balanced Accuracy    0.5000000
```

For white wine, the random forest model performed better. We have a 95% CI of (.6276, and .6745), and a Kappa level of 0.4494. KNN did not perform as well. Both did a rather poor job of identifying white wines of the 2 lowest and 2 highest classes.

==========

Finishing up

From our models here, we've learned that it's only accurate to identify very average quality wines, rendering it not very useful. It is quite difficult to conclude that there can be a model that can accurately identify the low and high quality wine.