# Post02 - Stat 133, What makes a wine good?

*Mudit Goyal*

*11/30/2017*

## Introduction

Machine Learning is an extremely hot topic in the world right now, and everyone in the industry (at least if you want to become a Software Engineer of if you do study CS), looks for at least basic knowledge of Machine Learning. For my second post, I'm going to do a quick and dirty introduction to ML! We're going to work with a relatively small data set (wines!) since large data-sets require a large amount of computing power, which you may not have. I received this challenge initially as part of an interview process for SAP and decided to expand on it and actually see if any insight can be made.

Check out this source for more information about why ML is important. (Ref 1)

Another cool document for you guys to read is this piece from business insider (Ref 2)

## Background

Disclaimer: I did this mainly with the framework that I learned in my Intro to ML Class (Industrial Engineering 142.) They taught us how to code in R and also go through the steps of pre-processing, running regressions, builing the correlation plot, modeling itself, and then interpreting the confusion matrix.

You can find more information about IEOR 142 here (Ref 3).

Given that we are so close to CA wine country I'm interested in understanding what makes a wine 'good'. To answer this question, we have obtained the attached dataset with a collection of descriptive and physicochemical characteristics of more than 1300 different red wines along with a 'quality rating'. The quality rating represents the median score from three wine experts from 0-10 where 0 is terrible and 10 is excellent.

I'm going to work with the basic first model you'd learn primarily for this
- The traditional K-nearest neighbors

## Pre-processing all the data.

Before you can do anything with machine learning, you MUST make sure that your data set is clean. If it's not clean, your results really will not mean anything, and you will most definitely have wasted your time doing your modeling. There's a few ways you can go about cleaning the data. Luckily, the wine dataset (Ref 4)we are using here is very, very clean; however, this will not always be the case. There are a ton of empty columns, and if you remember the reading we had to do earlier about data cleaning (Ref 5), you know that it's bad practice to have empty cells. There's a few things you can do; we're going to make every empty cell in the dataset into a 0.

## Let's begin the analysis.

Since R is not as good as python when it comes to machine learning, we need to set up what's called "parallel processing", meaning your computer will use all of it's available cores to run the model, instead of just one. If you're curious, try running the code without that chunk, it'll

take much, much longer to run, it might not even run. This below chunk is to do that and to also read all of the libraries we'll be using in this post.

```
# 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. You'll find descriptions
# of these packages as you read through this post.
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(kknn)
```

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

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

```
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Now we're actually going to be reading the data using read.csv. We're going to be reading our red wine dataset. The line of code str(df) tells us what the structure of the dataset is. This is important to know for you so that you can figure out how many variables you're working with, in addition to knowing what type they are. In this case, each column is numerical, and our response variable is the quality column! Let's do it.

You can download the dataset here (Ref 7). Save it to a directory and read the file accordingly.

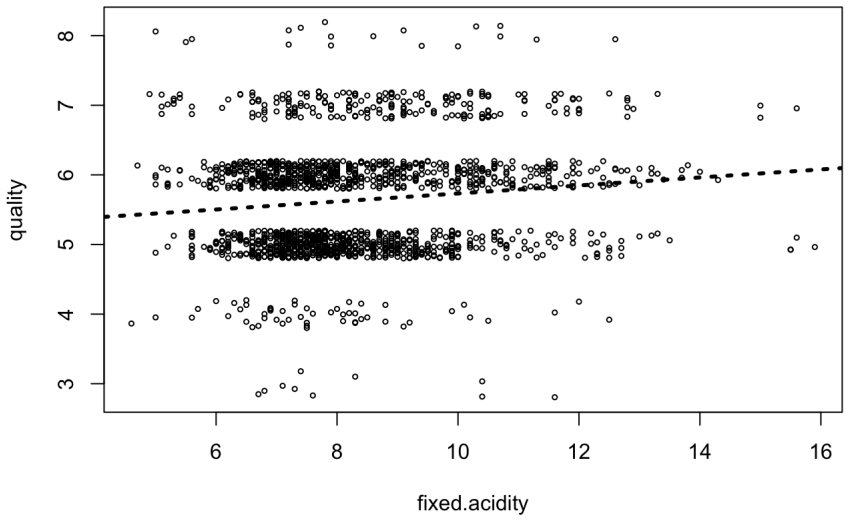
```
# Using the read.csv function to read the data
df <- read.csv("red.csv")

# We don't want any empty cells in the data, so we will
# change all of the NA values to 0.
df[is.na(df)] <- 0
str(df)
```

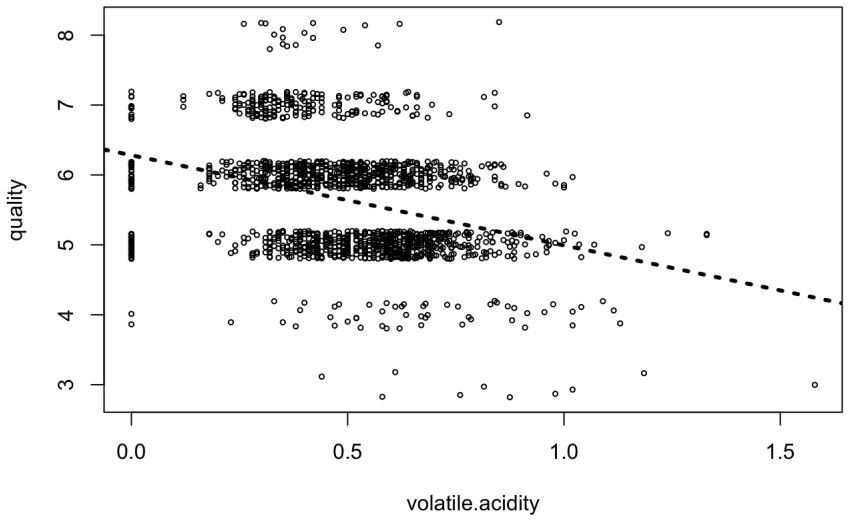
```
## 'data.frame':    1599 obs. of  14 variables:
##  $ fixed.acidity       : num  7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5 ...
##  $ volatile.acidity    : num  0.7 0 0.76 0.28 0.7 0.66 0.6 0.65 0.58 0.5 ...
##  $ citric.acid         : num  0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...
##  $ astringency.rating  : 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 0.56 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 ...
```

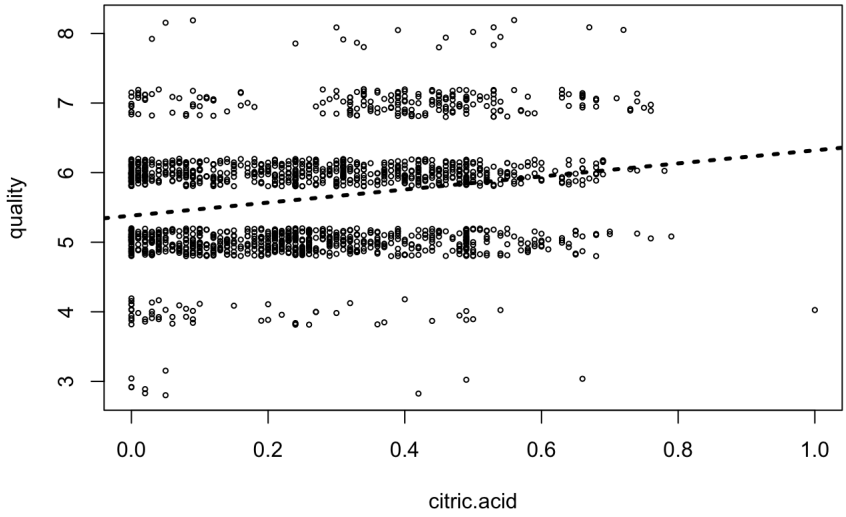
There are 1599 samples and 14 different variables.

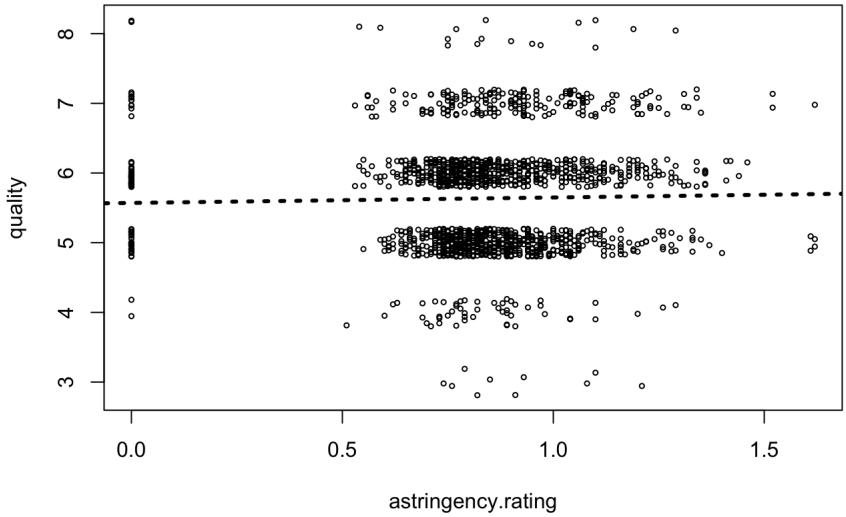
We are now going to visualize the data using plots for each of the predictor variables. We want to see how our response variable, quality, correlates with all of the other variables which contribute to it.

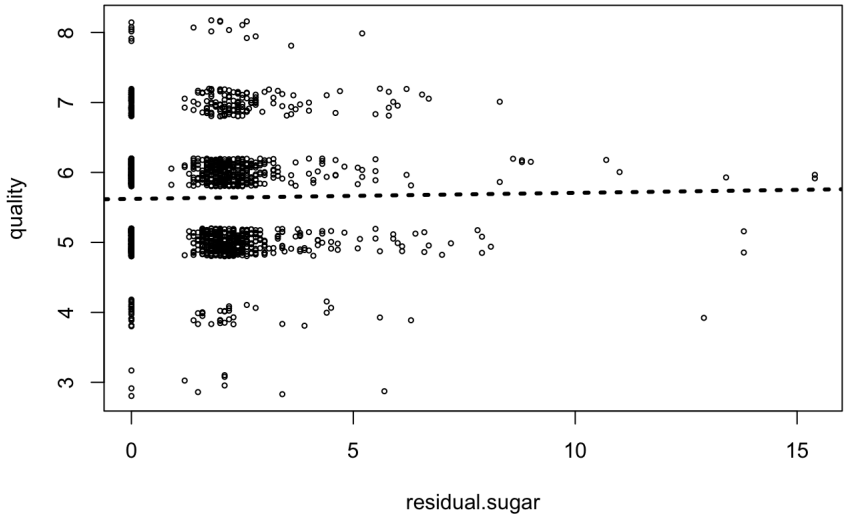
```
for (i in c(1:12)) {
    plot(df[, i], jitter(df[, "quality"]), xlab = names(df)[i],
         ylab = "quality", cex = 0.5, cex.lab = 1)

    abline(lm(df[, "quality"] ~ df[ ,i]), lty = 3, lwd = 3)
}
```

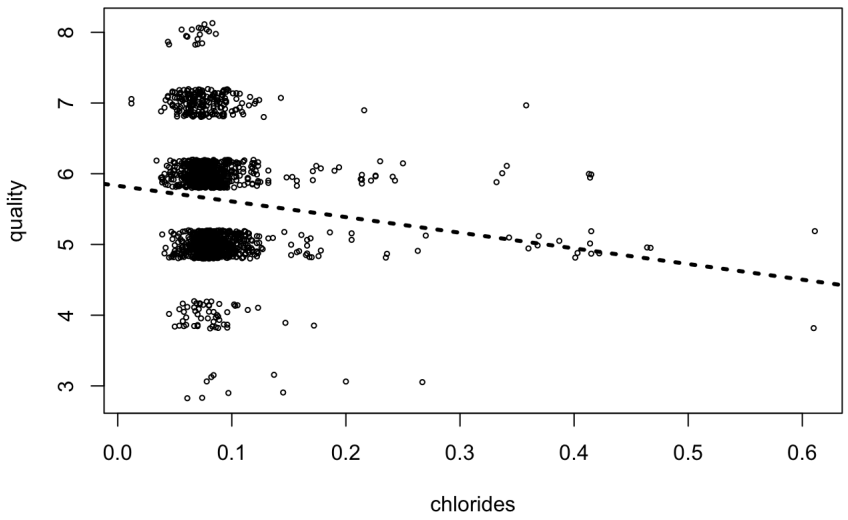


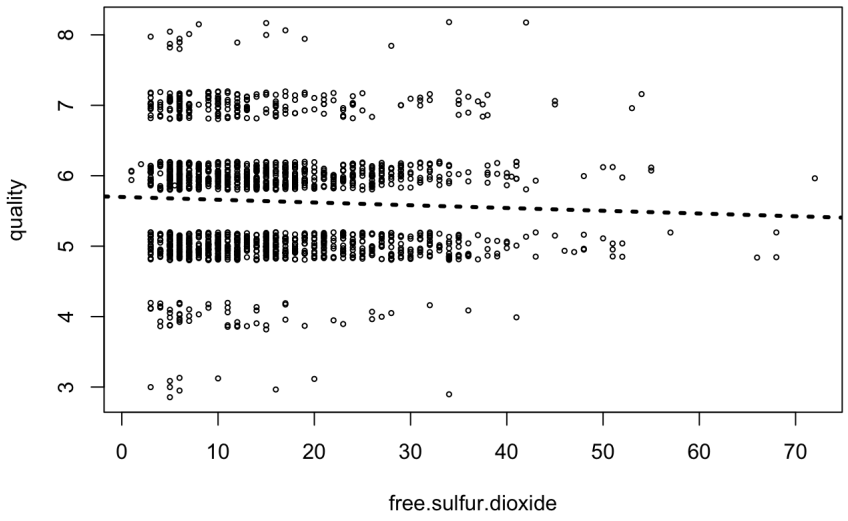



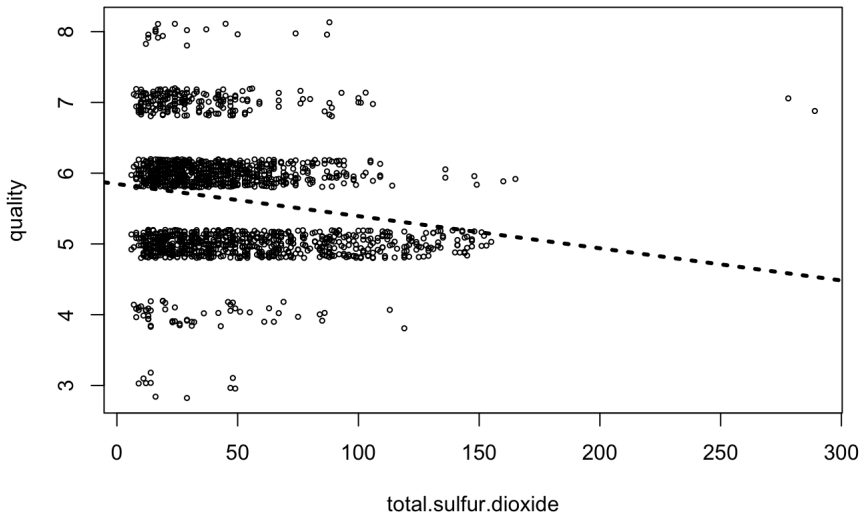

quality
3
4
5
6
7
8
0.0
0.2
0.4
0.6
0.8
1.0
citric.acid

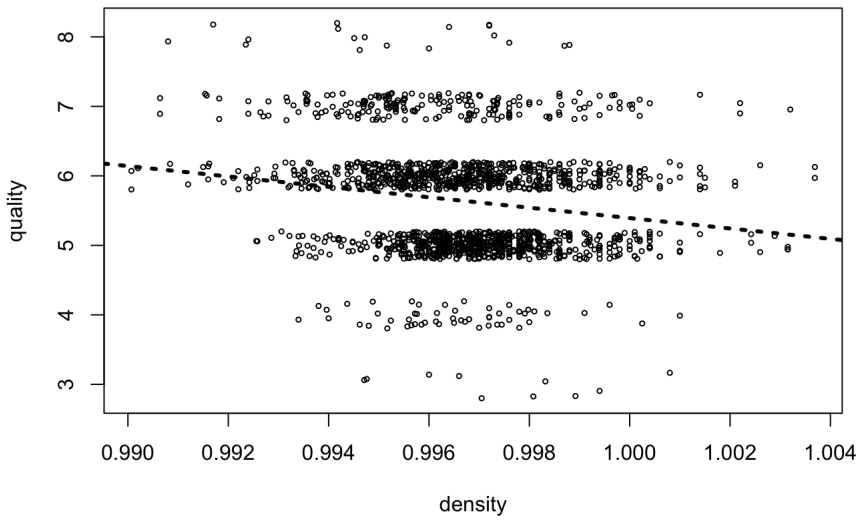

quality
3
4
5
6
7
8
0.0
0.5
1.0
1.5
astringency.rating

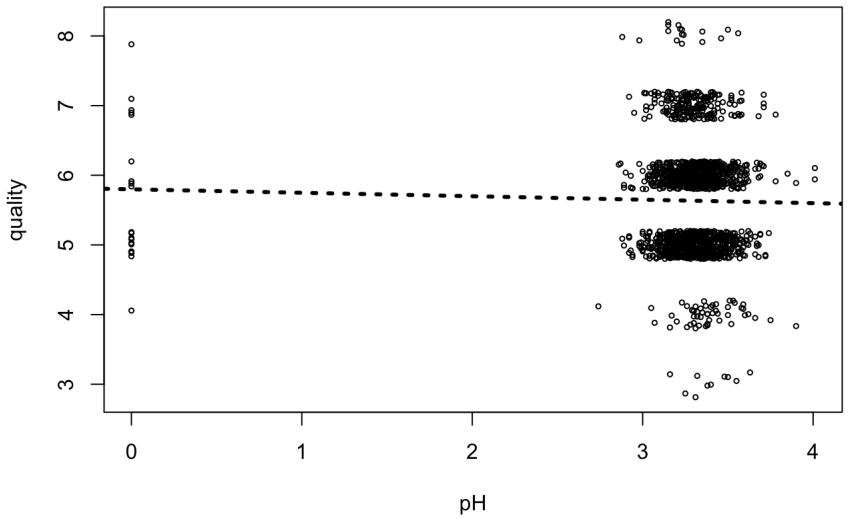

quality
3
4
5
6
7
8
0
5
10
15
residual.sugar

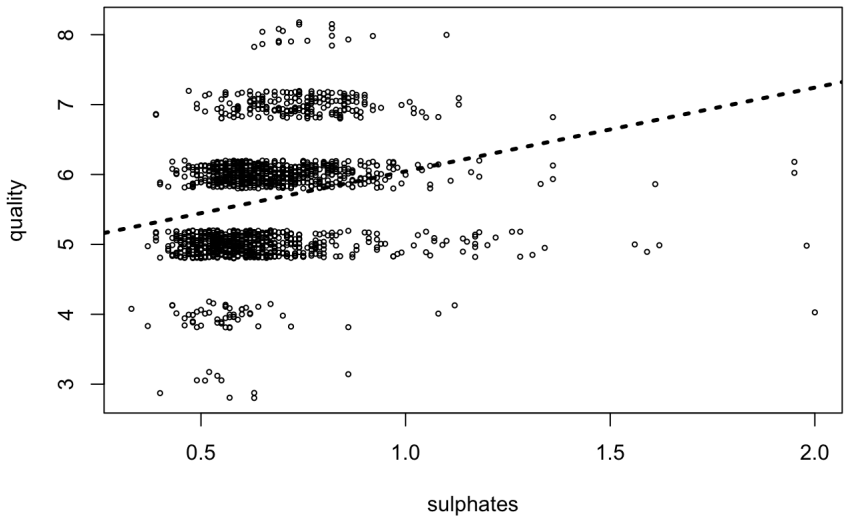

quality
chlorides
0.0
0.1
0.2
0.3
0.4
0.5
0.6
3
4
5
6
7
8

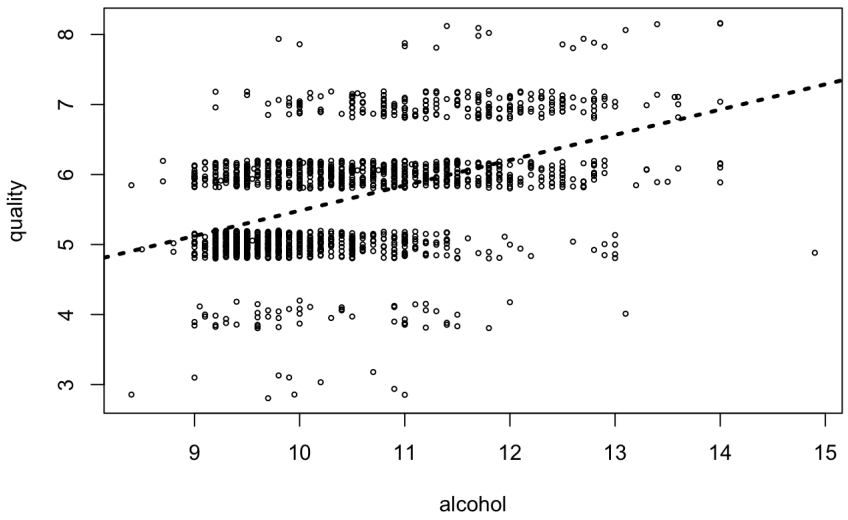

quality
free.sulfur.dioxide
0
10
20
30
40
50
60
70
3
4
5
6
7
8


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


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


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

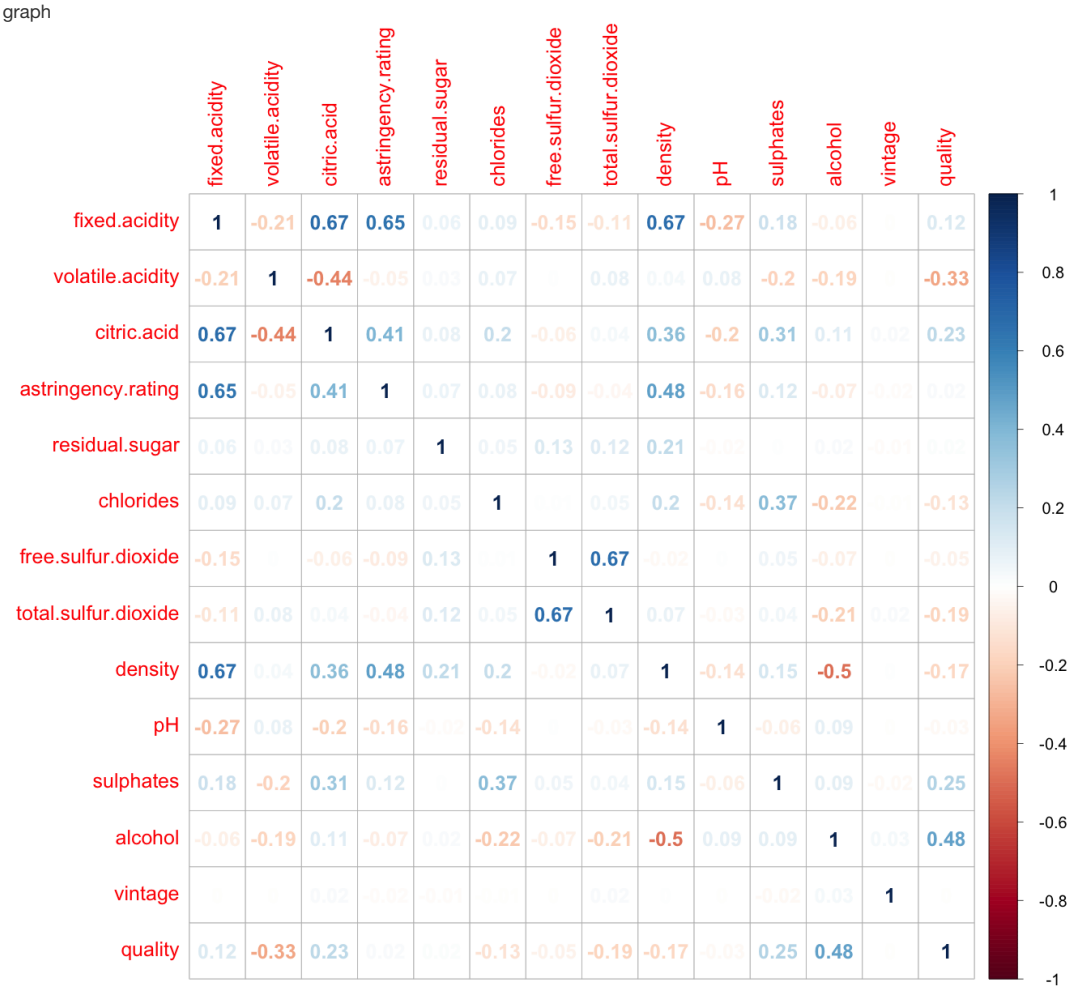
displays the linear regression of our response variable **quality** as a function of each of the predictor variables. We can tell that quality

I created a correlation plot next to further look at the associations between all the variables.The regression line for citric.acid, free.sulfur.dioxide, and sulphates appear to have a very weak relationship to quality, and the other variables don't correlate a whole lot to the response variable either.

To further check out how our response variable is related to the other independent variables, we're going to create what's called a correlation plot. This is probably the more interesting thing you'll see in this because you can really draw a ton of good insights from this.

```
cor_redwines <- cor(df)
png(height = 1200, width = 1500, pointsize = 25, file = 'red_cor_plot.png')
corrplot(cor_redwines, method = 'number')
```

Here's our graph




You can see the weak relationships here between quality, 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, because of all the weak associations shown in the correlation plot.

What other insights can you make from this graph? Check it out. For example, the value between quality and volatile acidity is -0.33. This value

can be interpreted as such: “If a wine has a higher level of volatile acidity, the quality will decrease. The two of these variables have a negative, linear association.” On the flipside, we see that the relationship between alcohol and quality is ~0.5. This can be interpreted like this: “If you add more alcohol, the quality of a wine will increase. There is a strong, positive, linear association between alcohol level and quality of wine.”

Now that we’re able to intepret that data, we’re ready for the K Nearest Neighbor model! You should read this post (Ref 6) if you’re curious in learning a bit more about this particular modeling method.

# Building the Model

We **need** to convert our response variable to factor, and then do the split into training and testing sets. Factors represent a very efficient way to store character values, because each unique character value is stored only once, and the data itself is stored as a vector of integers. It’s essentially normalizing the data. When you’re splitting the data, it’s a good idea to do a 2/3 : 1/3 ratio between the training data and the test data.

```
df$quality <- as.factor(df$quality)

tr <- createDataPartition(df$quality, p = 2/3, list = F)
train_red <- df[tr,]
test_red <- df[-tr,]
```

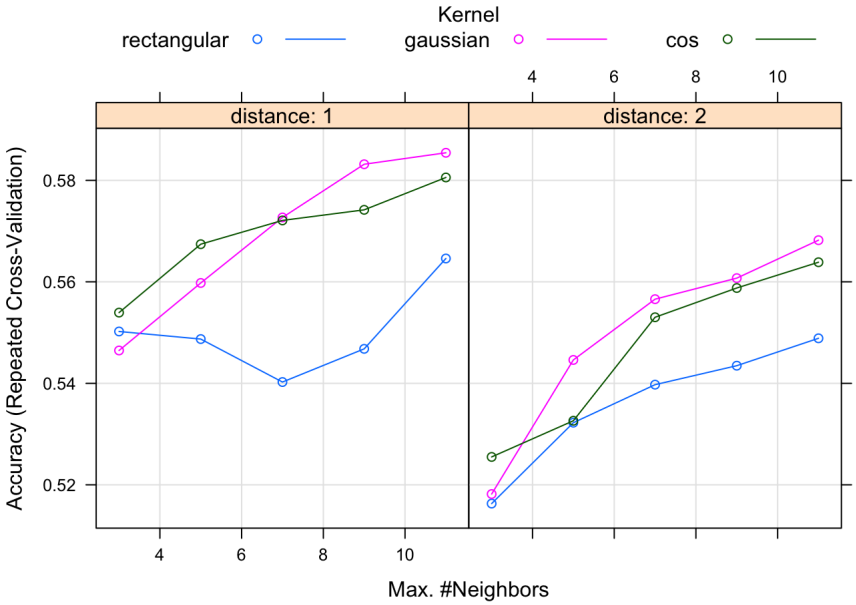
We are going to go about this using both k-nearest neighors (KNN).

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. Caret is a fantastic library and abstracts away a lot of the peculiarities of machine learning. Use this! Work smart, not hard. You really do not need to know a lot of the technicalities of ML to do ML.

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. Again, take the leap of faith here.

For the distance, 1 is the Manhattan distance, and 2 is the Euclidian distance. Picture the Manhattan distance as the perimeter of a triangle, minus the hypotenuse. The euclidean distance is the hypotenuse.

```
# This block is to figure out the control of the data. Need to specify how many times we're running this!
train_ctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 5)
# Creates the grid on which we're drawing the graphs on.
knn_grid <- expand.grid(kmax = c(3, 5, 7, 9, 11), distance = c(1, 2),
                        kernel = c("rectangular", "gaussian", "cos"))
# Need to actually train the data!
knn_train <- train(quality ~ ., data = train_red, method = "knn",
                   trControl = train_ctrl, tuneGrid = knn_grid,
                   preProcess = c("center", "scale"))
# Displaying the plot.
plot(knn_train)
```




You can interpret this graph in many ways. In the simplest form, the x-axis is the number of variables which are used to get the quality amount. You can tell that more neighbors = higher accuracy.

Lastly, we need to actually see how accurate our model is. A quick note on accuracy, the only way you can benchmark your model’s performance is against a previous measure. For example, if we knew that the accuracy was like 30%, and we have an accuracy of 60% here, our model is helpful. If it’s lower, it’s relatively useless. This below information is the confusion matrix.

```
knn.predict <- predict(knn_train, test_red)
confusionMatrix(knn.predict, test_red$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   0  12 167  68   8   1
##          6   3   5  57 127  33   3
##          7   0   0   3  17  24   2
##          8   0   0   0   0   1   0
##
## Overall Statistics
##
##                Accuracy : 0.5989
##                  95% CI : (0.5558, 0.6408)
##     No Information Rate : 0.4275
##     P-Value [Acc > NIR] : 1.57e-15
##
##                   Kappa : 0.3442
##  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   0.7357   0.5991  0.36364 0.000000
## Specificity           1.00000  1.00000   0.7072   0.6834  0.95269 0.998095
## Pos Pred Value            NaN      NaN   0.6523   0.5570  0.52174 0.000000
## Neg Pred Value        0.99435  0.96798   0.7818   0.7195  0.91340 0.988679
## Prevalence            0.00565  0.03202   0.4275   0.3992  0.12429 0.011299
## Detection Rate        0.00000  0.00000   0.3145   0.2392  0.04520 0.000000
## Detection Prevalence  0.00000  0.00000   0.4821   0.4294  0.08663 0.001883
## Balanced Accuracy     0.50000  0.50000   0.7215   0.6412  0.65816 0.499048
```

There's a bunch of different ways that you can go about interpreting this data. The accuracy statistic, which is hovering around 60% or so, which at first you might say is not great. However, this is a very poor approach in figuring out whether or not you can predict and figure out what makes a wine good. This number is only useful if we have a **benchmark** to compare it too. For example, if the benchmark was above 60%, we know our model didn't provide anything. However, if it's like…30% or something, we know that our model goes above and beyond that, so we can make a conclusion there.

All in all, with this dataset, we'd need to do **a lot more work** to get any good results.