Wine Project

22 December 2020

Introduction

This data set contains 12 attributes of wine plus a quality, which I treated as both an attribute and an outcome. This started as actually two data sets, one each for red wine and white wine. I combined the two sets and there are about 6500 observations. This is a good size to work with. In the combined data set I created a binary variable for red and white wines. There were 4900 observations of white wine and about 1600 or red wine. I wanted to simulate a data set that classified a tumor as a benign or malign. I assumed most tumors are benign and the challenge is to correctly identify the malign tumor. Therefore I assigned 0 to white wine, simulating a negative on a cancer screening and 1 to red wine for a positive. However, I wanted to use models to see if I could accurately predict the quality of wine, which is a continuous variable. Also, I felt like wine characteristics are a little more interesting to work with than obscure attributes of cells.

I use several models and frequently switch back between predicting the quality on a continuous scale (e.g. linear regression) and the classifier of red or white (e.g. logistic regression). K nearest neighbor and support vector machine were used as classifier. Finally, a decision tree and random forest where used as both. I tried to show how models could be used in different ways.

Import libraries

Import and observe data sets

```
#import and view red wine data set
red <- read.csv('https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequal
ity-red.csv', sep=';')
print(paste('The class of the red wine data set is:', class(red)))
```

```
## [1] "The class of the red wine data set is: data.frame"
```

```
print('The dimensions of the red wine data set are:')
```

```
## [1] "The dimensions of the red wine data set are:"
```

```
          dim(red)
```

```
## [1] 1599   12
```

```
head(red)
```

```
##   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
```

```
print(paste('Total missing values in the red wine data set is:', sum(is.na(red))))
```

```
## [1] "Total missing values in the red wine data set is: 0"
```

```
#import and view white wine data set
white<- read.csv('https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequa
lity-white.csv', sep=';')
print(paste('The class of the red wine data set is:', class(white)))
```

```
## [1] "The class of the red wine data set is: data.frame"
```

```
print('The dimensions of the red wine data set are:')
```

```
## [1] "The dimensions of the red wine data set are:"
```

```
           dim(white)
```

```
## [1] 4898   12
```

```
head(white)
```

```
##   fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
## 1           7.0             0.27        0.36           20.7     0.045
## 2           6.3             0.30        0.34            1.6     0.049
## 3           8.1             0.28        0.40            6.9     0.050
## 4           7.2             0.23        0.32            8.5     0.058
## 5           7.2             0.23        0.32            8.5     0.058
## 6           8.1             0.28        0.40            6.9     0.050
##   free.sulfur.dioxide total.sulfur.dioxide density   pH sulphates alcohol
## 1                  45                  170  1.0010 3.00      0.45     8.8
## 2                  14                  132  0.9940 3.30      0.49     9.5
## 3                  30                   97  0.9951 3.26      0.44    10.1
## 4                  47                  186  0.9956 3.19      0.40     9.9
## 5                  47                  186  0.9956 3.19      0.40     9.9
## 6                  30                   97  0.9951 3.26      0.44    10.1
##   quality
## 1       6
## 2       6
## 3       6
## 4       6
## 5       6
## 6       6
```

```
print(paste('Total missing values in the red wine data set is:', sum(is.na(white))))
```

```
## [1] "Total missing values in the red wine data set is: 0"
```

```
# observe the quality ratings of data set
print('The range of values for quality in the red wine data set are as follow:')
```

```
## [1] "The range of values for quality in the red wine data set are as follow:"
```

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

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

```
print('The range of values for quality in the white wine data set are as follow:')
```

```
## [1] "The range of values for quality in the white wine data set are as follow:"
```

```
sort(unique(white$quality))
```

```
## [1] 3 4 5 6 7 8 9
```

Check for outliers

The grubbs test identifies several outliers. Actually most attributes in both the red and white sets have an outlier. However, I don't have any reason to believe these are erroneious points that don't below in the data set. Therefore, I am not doing to remove any of the outliers. However, I do will plot each of these as a histogram for further analysis.

```
# check for outliers
for (i in 1:11) {
  print(paste("Outlier test for red wine", colnames(red)[i]))
  print(grubbs.test(red[,i]), type = 11)
  print(paste("Outlier test for white wine", colnames(red)[i]))
  print(grubbs.test(white[,i]), type = 11)
}
```

```
## [1] "Outlier test for red wine fixed.acidity"
##
##  Grubbs test for one outlier
##
## data:  red[, i]
## G = 4.35379, U = 0.98813, p-value = 0.01011
## alternative hypothesis: highest value 15.9 is an outlier
##
## [1] "Outlier test for white wine fixed.acidity"
##
##  Grubbs test for one outlier
##
## data:  white[, i]
## G = 8.70422, U = 0.98453, p-value < 2.2e-16
## alternative hypothesis: highest value 14.2 is an outlier
##
## [1] "Outlier test for red wine volatile.acidity"
##
##  Grubbs test for one outlier
##
## data:  red[, i]
## G = 5.87614, U = 0.97838, p-value = 2.779e-06
## alternative hypothesis: highest value 1.58 is an outlier
##
## [1] "Outlier test for white wine volatile.acidity"
##
##  Grubbs test for one outlier
##
## data:  white[, i]
## G = 8.15281, U = 0.98642, p-value = 1.088e-12
## alternative hypothesis: highest value 1.1 is an outlier
##
## [1] "Outlier test for red wine citric.acid"
##
##  Grubbs test for one outlier
##
## data:  red[, i]
## G = 3.74240, U = 0.99123, p-value = 0.1413
## alternative hypothesis: highest value 1 is an outlier
##
## [1] "Outlier test for white wine citric.acid"
##
##  Grubbs test for one outlier
##
## data:  white[, i]
## G = 10.95530, U = 0.97549, p-value < 2.2e-16
## alternative hypothesis: highest value 1.66 is an outlier
##
## [1] "Outlier test for red wine residual.sugar"
##
##  Grubbs test for one outlier
##
## data:  red[, i]
```

```
## G = 9.19281, U = 0.94708, p-value < 2.2e-16
## alternative hypothesis: highest value 15.5 is an outlier
##
## [1] "Outlier test for white wine residual.sugar"
##
##  Grubbs test for one outlier
##
## data:  white[, i]
## G = 11.71292, U = 0.97198, p-value < 2.2e-16
## alternative hypothesis: highest value 65.8 is an outlier
##
## [1] "Outlier test for red wine chlorides"
##
##  Grubbs test for one outlier
##
## data:  red[, i]
## G = 11.12355, U = 0.92252, p-value < 2.2e-16
## alternative hypothesis: highest value 0.611 is an outlier
##
## [1] "Outlier test for white wine chlorides"
##
##  Grubbs test for one outlier
##
## data:  white[, i]
## G = 13.74167, U = 0.96143, p-value < 2.2e-16
## alternative hypothesis: highest value 0.346 is an outlier
##
## [1] "Outlier test for red wine free.sulfur.dioxide"
##
##  Grubbs test for one outlier
##
## data:  red[, i]
## G = 5.36561, U = 0.98197, p-value = 5.659e-05
## alternative hypothesis: highest value 72 is an outlier
##
## [1] "Outlier test for white wine free.sulfur.dioxide"
##
##  Grubbs test for one outlier
##
## data:  white[, i]
## G = 14.91679, U = 0.95455, p-value < 2.2e-16
## alternative hypothesis: highest value 289 is an outlier
##
## [1] "Outlier test for red wine total.sulfur.dioxide"
##
##  Grubbs test for one outlier
##
## data:  red[, i]
## G = 7.37285, U = 0.96596, p-value = 8.326e-11
## alternative hypothesis: highest value 289 is an outlier
##
## [1] "Outlier test for white wine total.sulfur.dioxide"
##
##  Grubbs test for one outlier
```

```
##
## data:  white[, i]
## G = 7.09772, U = 0.98971, p-value = 2.727e-09
## alternative hypothesis: highest value 440 is an outlier
##
## [1] "Outlier test for red wine density"
##
##  Grubbs test for one outlier
##
## data:  red[, i]
## G = 3.67890, U = 0.99153, p-value = 0.182
## alternative hypothesis: highest value 1.00369 is an outlier
##
## [1] "Outlier test for white wine density"
##
##  Grubbs test for one outlier
##
## data:  white[, i]
## G = 15.02976, U = 0.95386, p-value < 2.2e-16
## alternative hypothesis: highest value 1.03898 is an outlier
##
## [1] "Outlier test for red wine pH"
##
##  Grubbs test for one outlier
##
## data:  red[, i]
## G = 4.52687, U = 0.98717, p-value = 0.004481
## alternative hypothesis: highest value 4.01 is an outlier
##
## [1] "Outlier test for white wine pH"
##
##  Grubbs test for one outlier
##
## data:  white[, i]
## G = 4.18365, U = 0.99643, p-value = 0.06917
## alternative hypothesis: highest value 3.82 is an outlier
##
## [1] "Outlier test for red wine sulphates"
##
##  Grubbs test for one outlier
##
## data:  red[, i]
## G = 7.91620, U = 0.96076, p-value = 1.065e-12
## alternative hypothesis: highest value 2 is an outlier
##
## [1] "Outlier test for white wine sulphates"
##
##  Grubbs test for one outlier
##
## data:  white[, i]
## G = 5.17107, U = 0.99454, p-value = 0.0005496
## alternative hypothesis: highest value 1.08 is an outlier
##
## [1] "Outlier test for red wine alcohol"
```

```
##
##  Grubbs test for one outlier
##
## data:  red[, i]
## G = 4.20114, U = 0.98895, p-value = 0.02022
## alternative hypothesis: highest value 14.9 is an outlier
##
## [1] "Outlier test for white wine alcohol"
##
##  Grubbs test for one outlier
##
## data:  white[, i]
## G = 2.99502, U = 0.99817, p-value = 1
## alternative hypothesis: highest value 14.2 is an outlier
```

Classify and combine the data sets

As mentioned in the introduction, I assigned white wine observations a zero (simulating a negative test) and red wine observations a 1 (simulating a positive test). Then I combined the data sets and randomly shuffled the observations.

```
# In order to set up a classification predictor assign a binary variable to each data set
# Part of this project is to simulate predicting benign vs. malign results
# Because there are more whites, I simulate white as benign and assigned it to 0; red is malign
 so it is assigned a 1
white_binary <- white %>% mutate(class = 0)
red_binary <- red %>% mutate(class = 1)

# Combine the data sets, randomly shuffle the rows and reset the index
set.seed(12)
rw_comb <- rbind(white_binary, red_binary)[sample(nrow(white_binary) + nrow( red_binary)),] %>%
          mutate(class = as.factor(class))
row.names(rw_comb) <- NULL
print('The dimensions of the combined data set are: ')
```

```
## [1] "The dimensions of the combined data set are: "
```

```
dim(rw_comb)
```

```
## [1] 6497   13
```

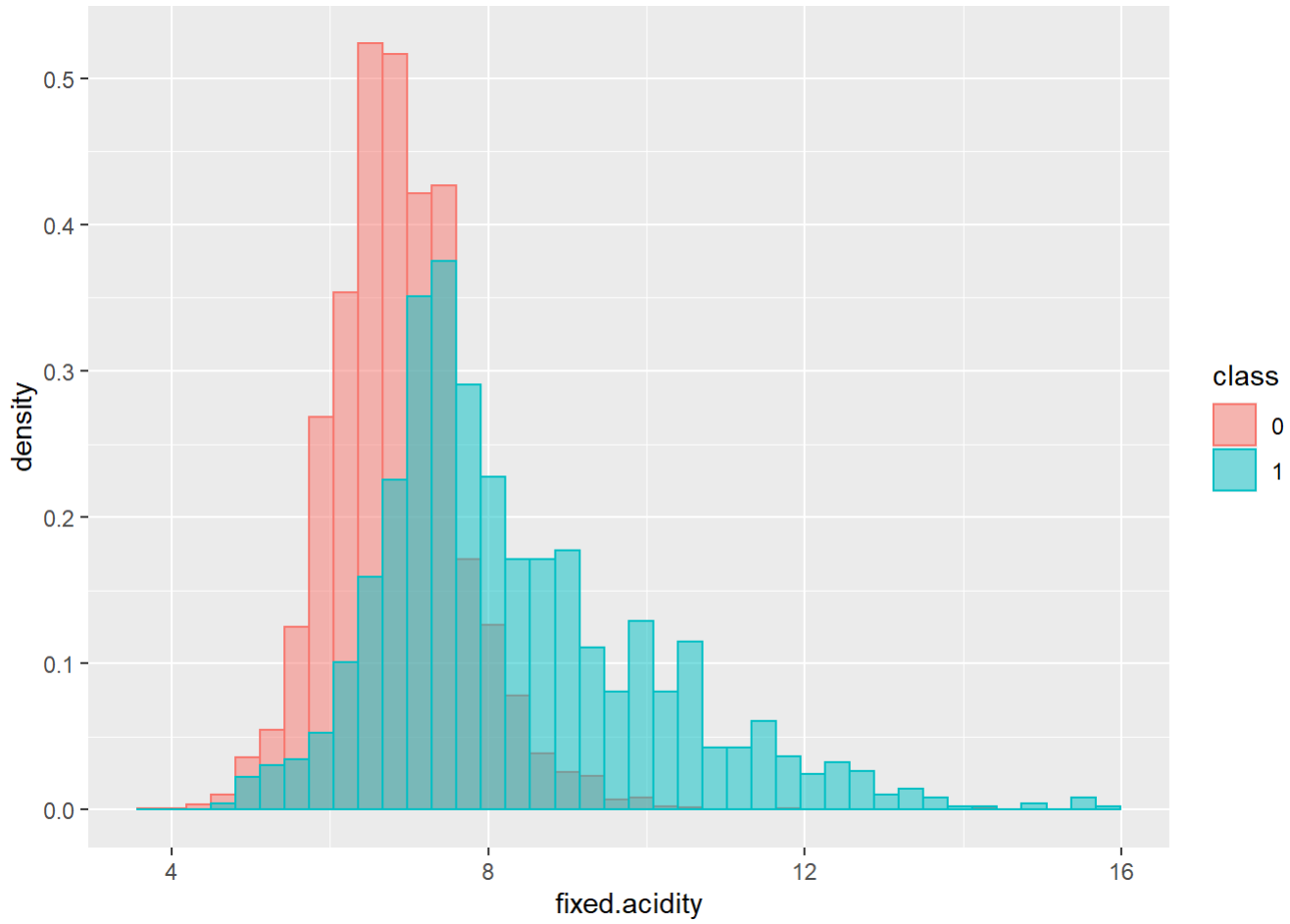
```
head(rw_comb)
```

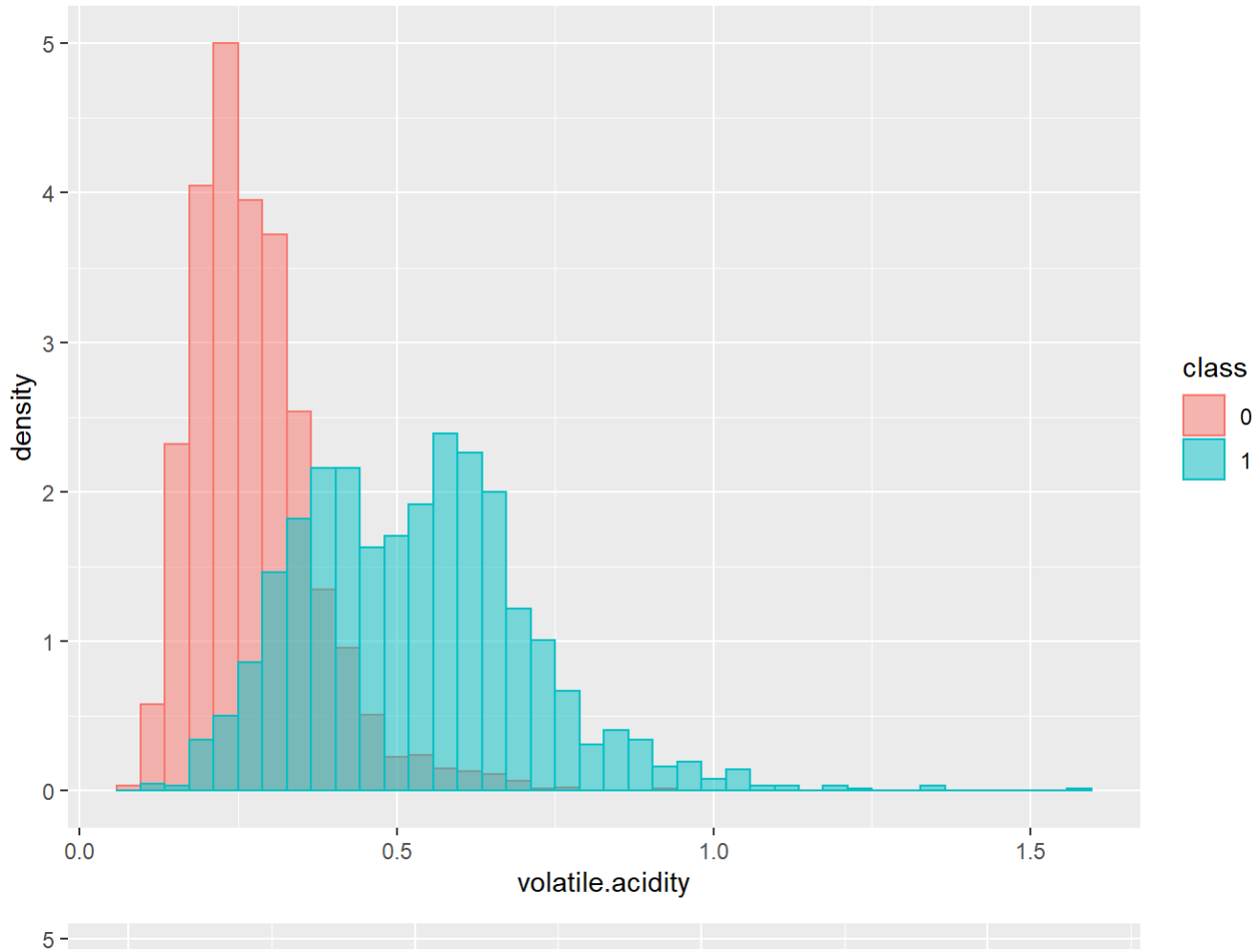
```
##   fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
## 1           6.3             0.25        0.53            1.8     0.021
## 2           6.6             0.27        0.25            1.2     0.033
## 3           6.6             0.38        0.28            2.8     0.043
## 4           8.0             0.22        0.42           14.6     0.044
## 5           6.6             0.26        0.46            6.9     0.047
## 6           8.5             0.28        0.34           13.8     0.041
##   free.sulfur.dioxide total.sulfur.dioxide  density   pH sulphates alcohol
## 1                  41                  101 0.989315 3.19      0.31    13.0
## 2                  36                  111 0.989180 3.16      0.37    12.4
## 3                  17                   67 0.989240 3.21      0.47    13.2
## 4                  45                  163 1.000300 3.21      0.69     8.6
## 5                  59                  183 0.995940 3.20      0.45     9.3
## 6                  32                  161 0.998100 3.13      0.40     9.9
##   quality class
## 1       6     0
## 2       6     0
## 3       6     0
## 4       7     0
## 5       5     0
## 6       6     0
```

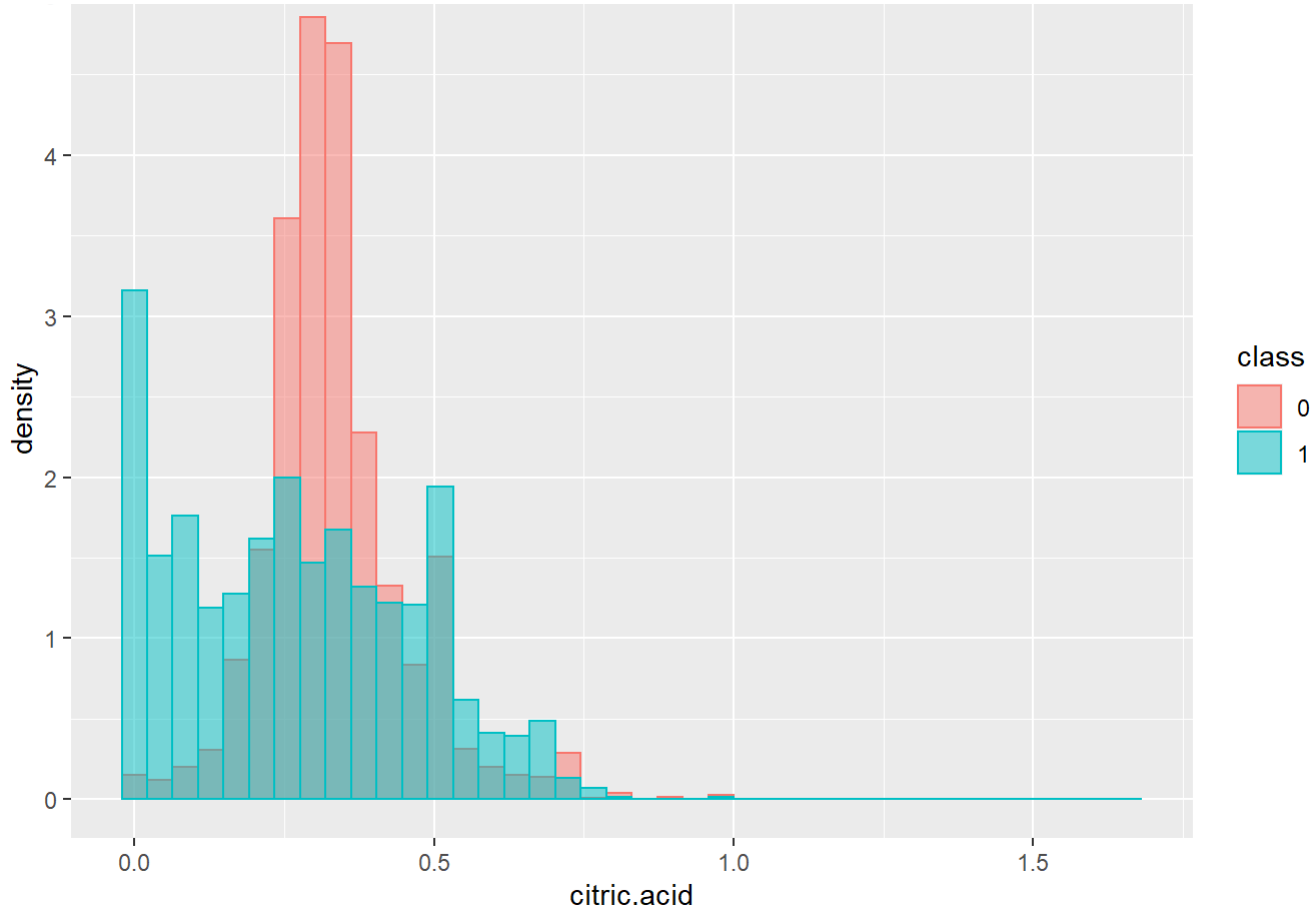
Overlaying histograms

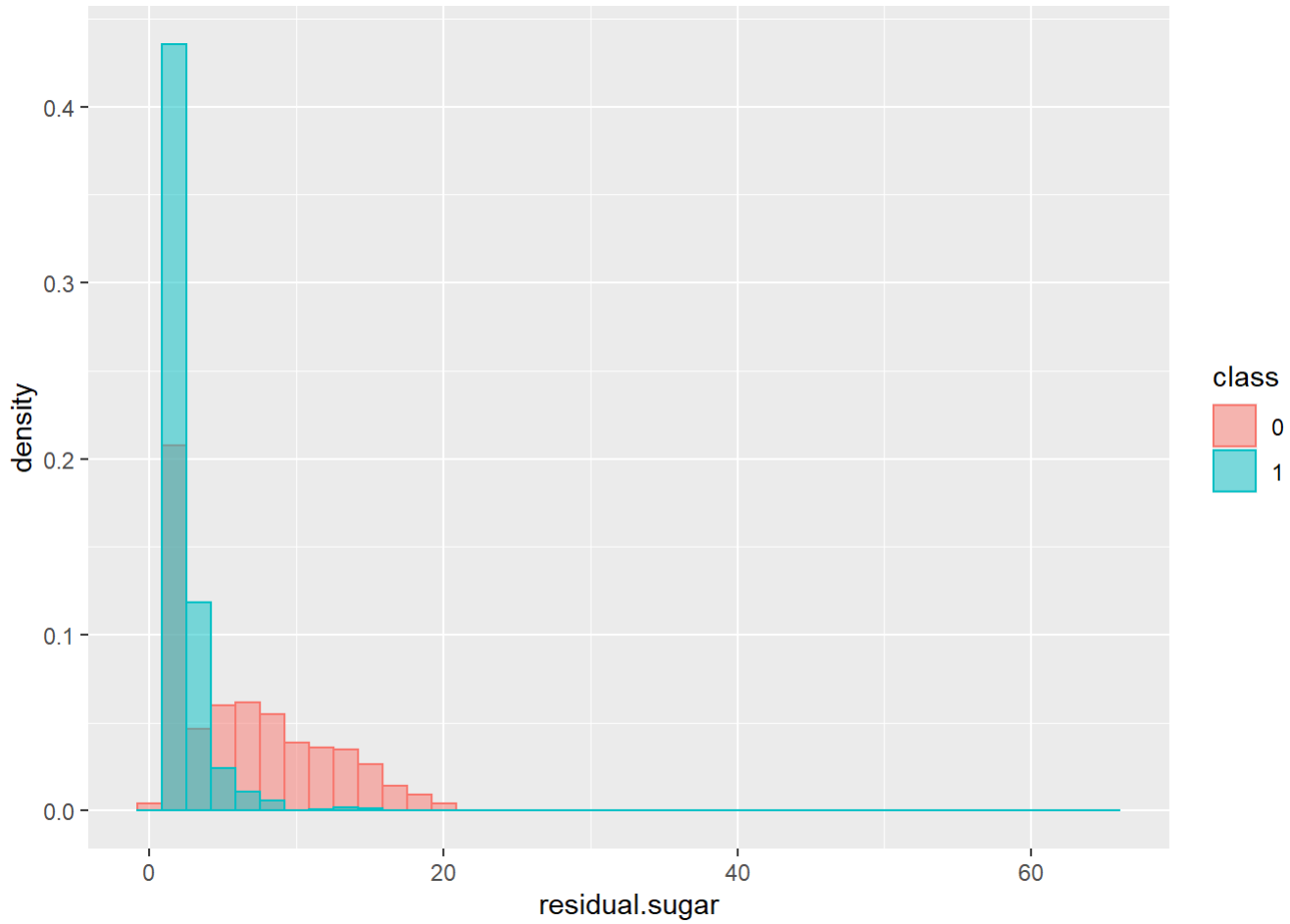
Here are the overlaying histograms. One part that is a little confusing is the red fill actually represents white wines. Also, there are about three times the amount of white wines as red wines, so I used density instead of count on the y-axis. This allows comparison of the distribution. I want to see how the distributions differ without being distracted by white having more counts than red. Some attributes such as total sulfur dioxide appear to be different distributions while others like alcohol have a lot of overlap. I suppose that makes sense. Both red and white wine contain alcohol that vary from around 8% to 15%. Likewise our grubbs test told us us 14.9 is an outlier for red wine with a p-value of .02. However, it is definitely possible for a red wine to have a 14.9% alcohol content, certainly higher than average, but no reason to attribute that value to an erroneous data point. Similarly, chlorides is an outlier for red wine. However, if you removed the highest value, the next highest value would also be an outlier. This would continue over several data points because this is right skewed distribution.

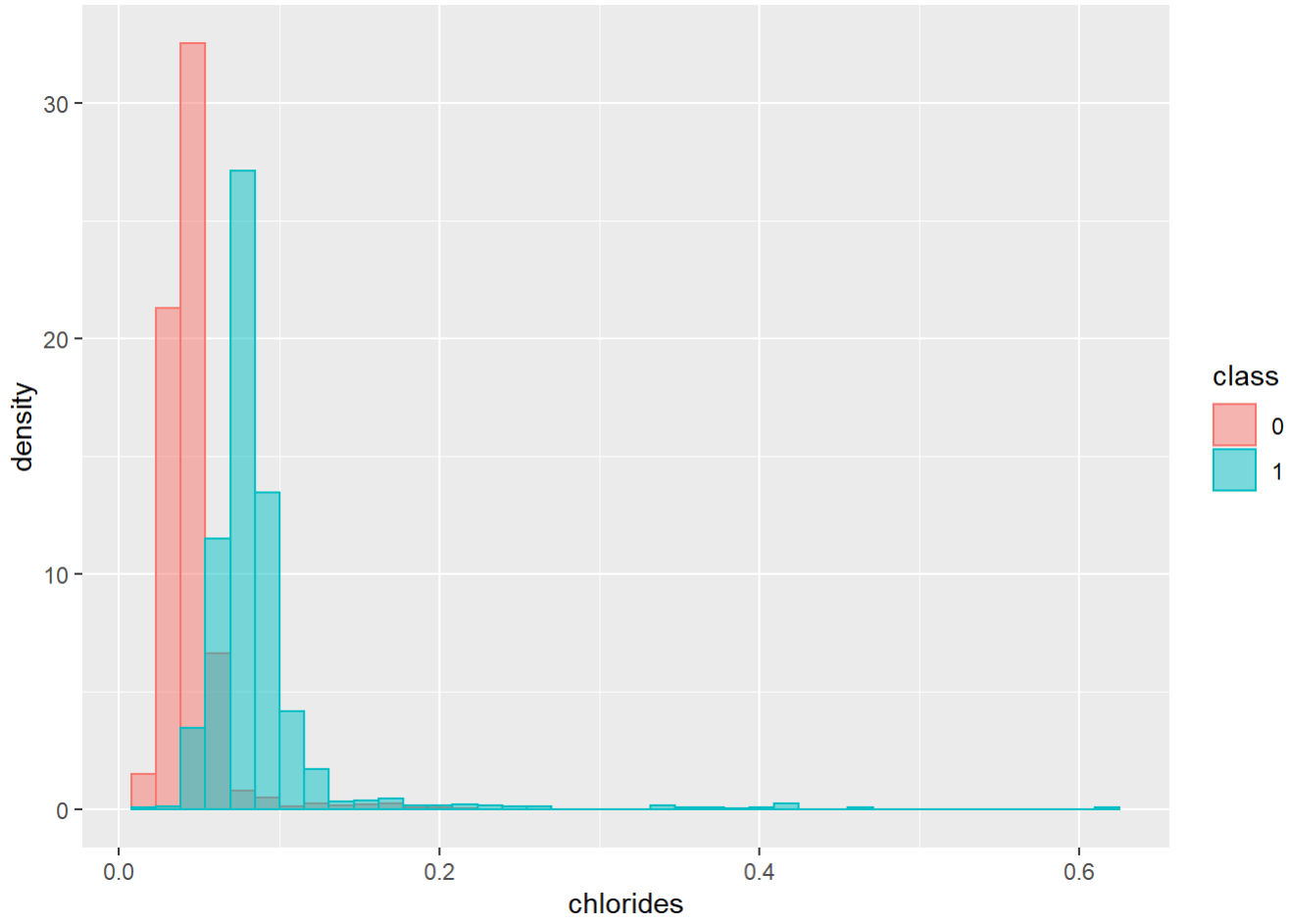
```
# overlaying histograms
for (i in 1:11) {
  print(ggplot(rw_comb, aes(x=rw_comb[,i], y=..density.., color = class, fill=class)) +
    geom_histogram(bins=40, alpha=.5, position='identity')+
    labs(x=colnames(rw_comb)[i]))
}
```

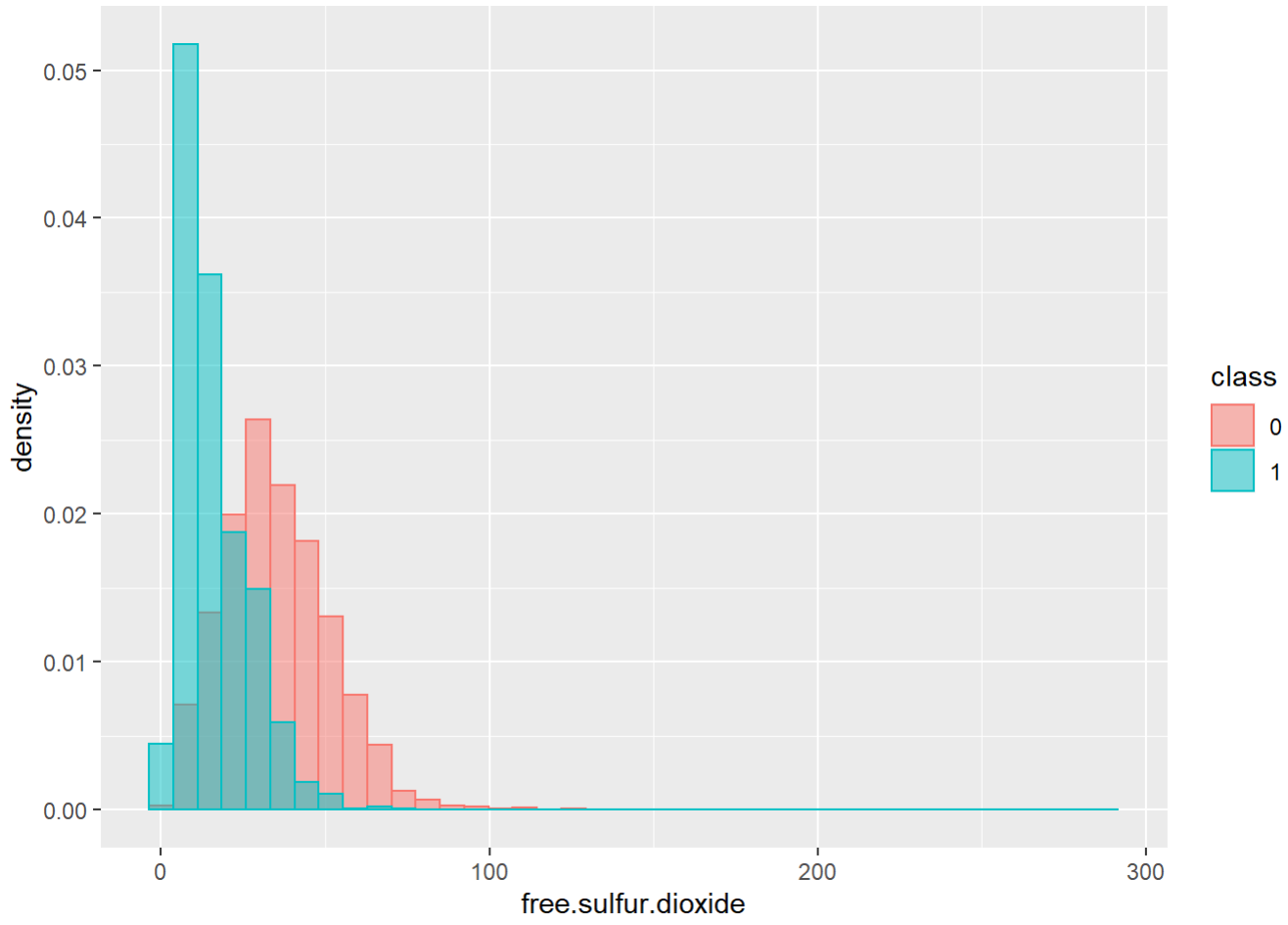

density
0.0
0.1
0.2
0.3
0.4
0.5
4
8
12
16
fixed.acidity
class
0
1

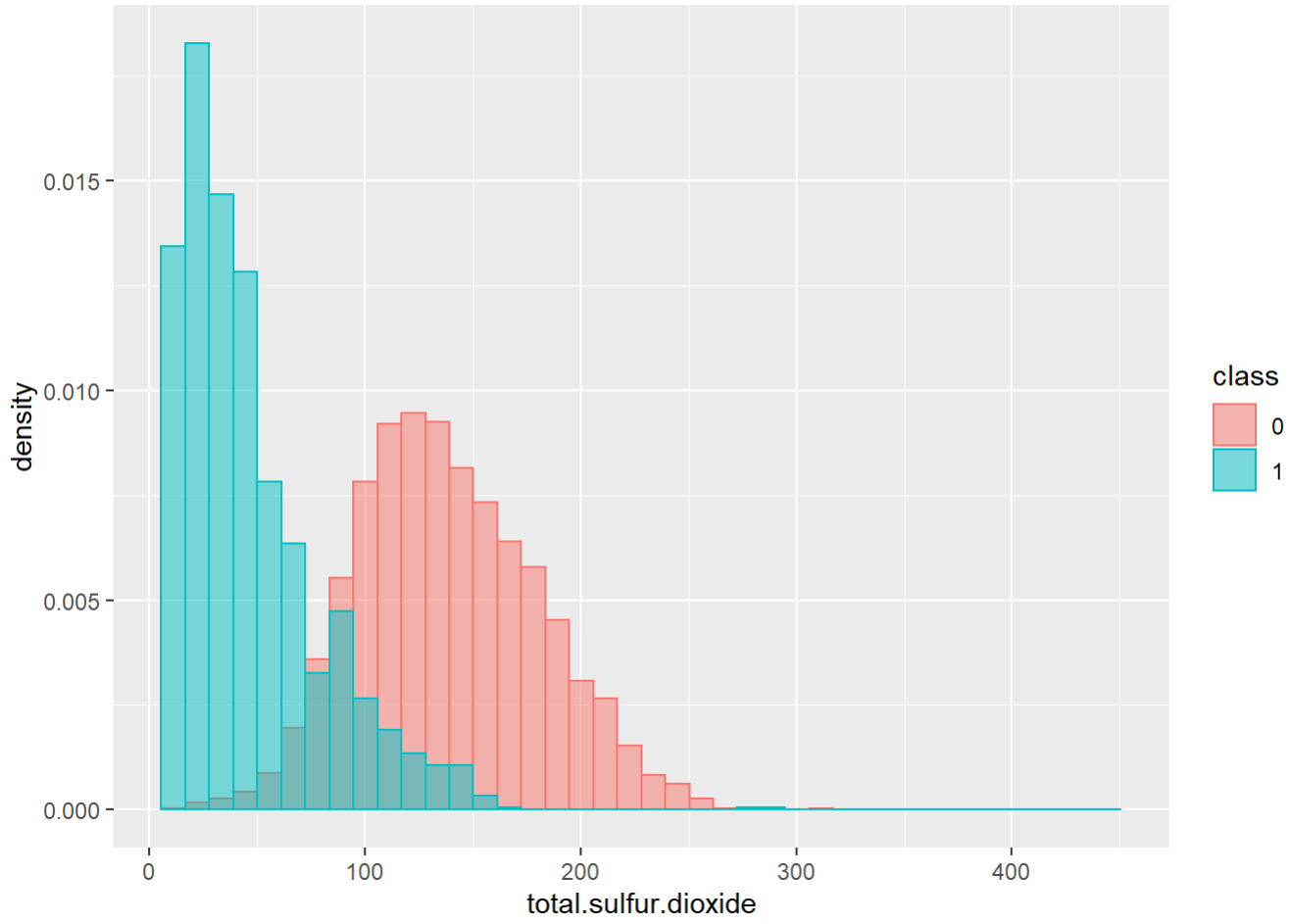

density
0
1
2
3
4
5
0.0
0.5
1.0
1.5
volatile.acidity
class
0
1

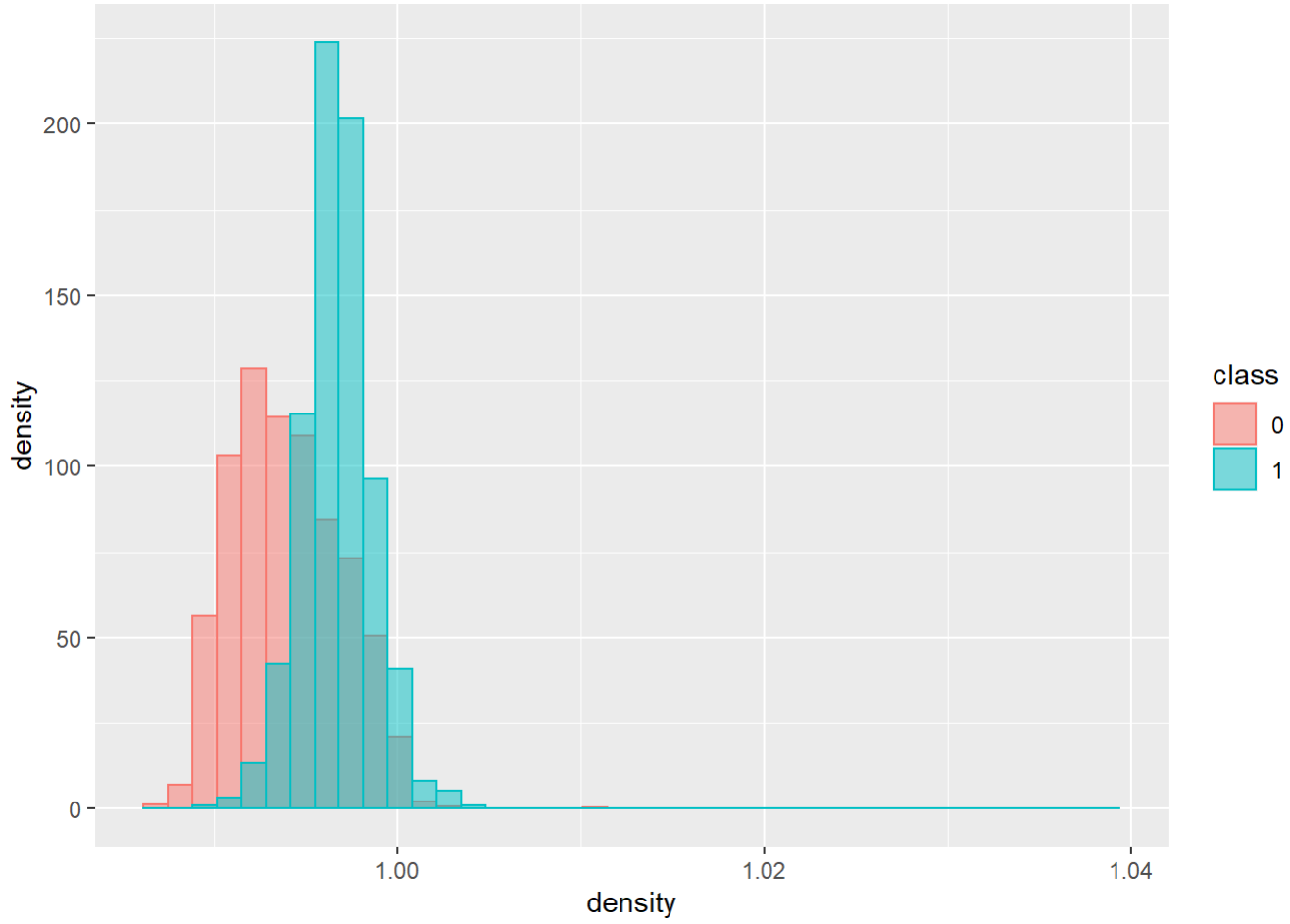

density
4
3
2
1
0
0.0
0.5
1.0
1.5
citric.acid
class
0
1

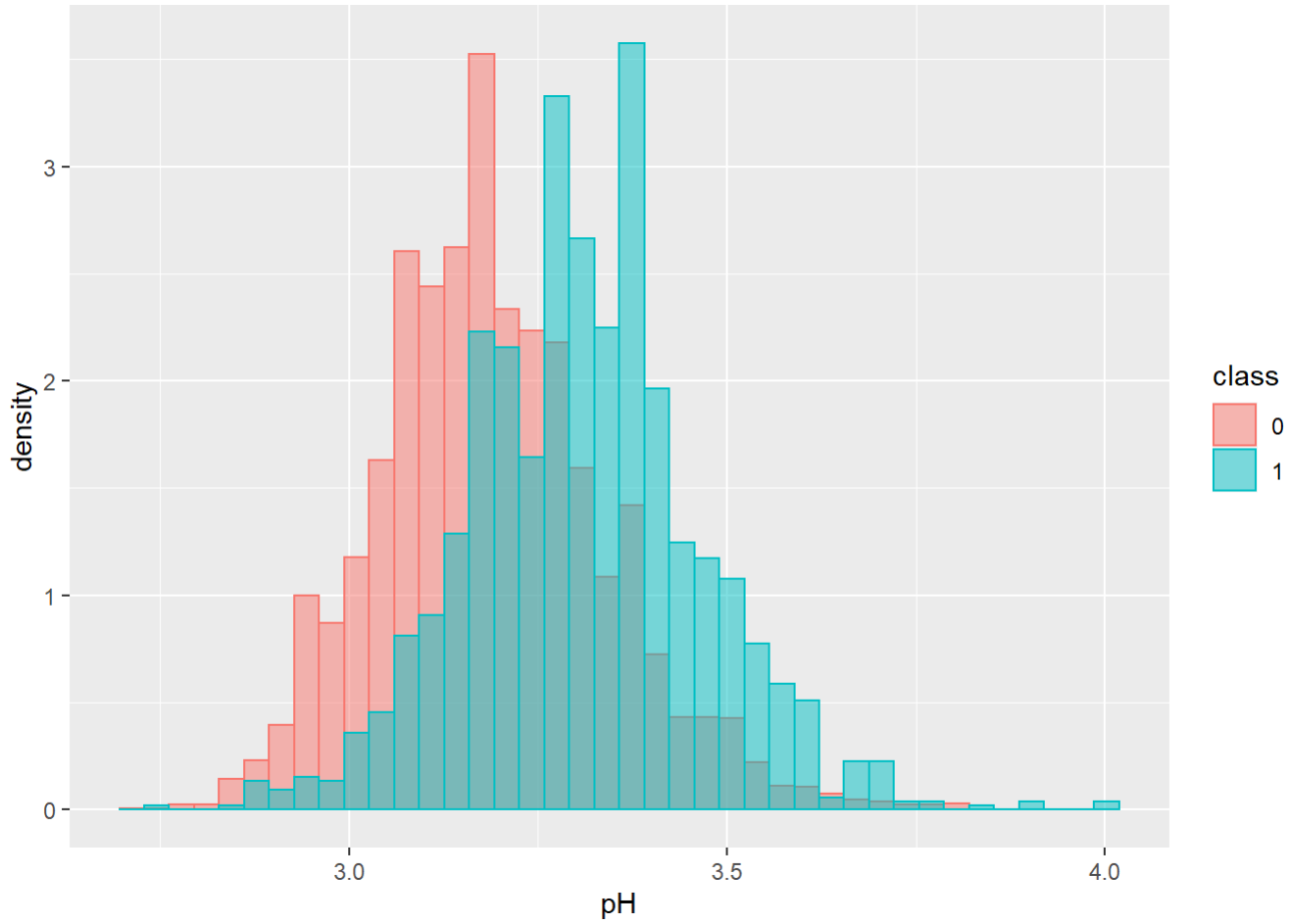

density
0.4
0.3
0.2
0.1
0.0
0
20
40
60
residual.sugar
class
0
1

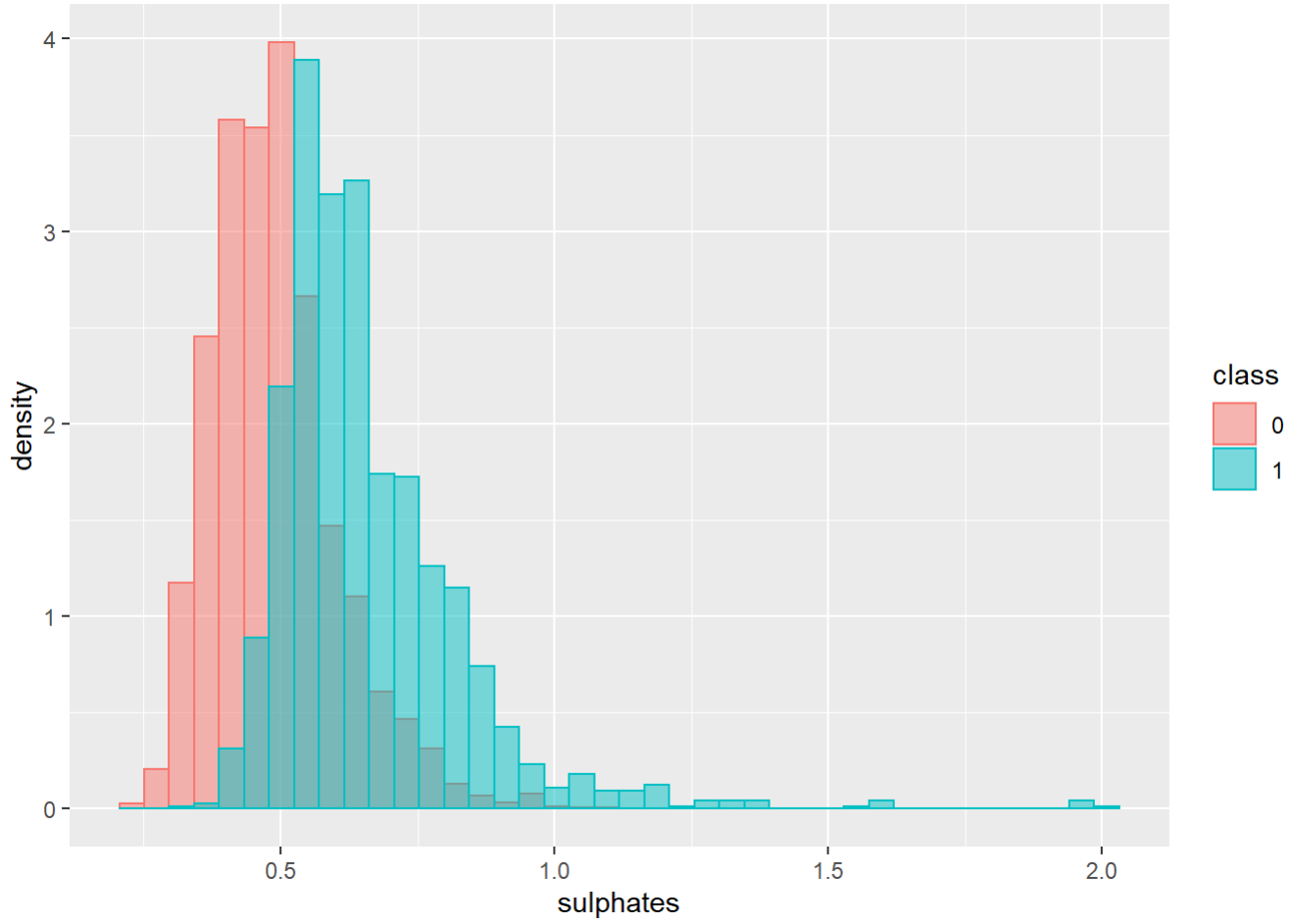

density
chlorides
class
0
1
30
20
10
0
0.0
0.2
0.4
0.6

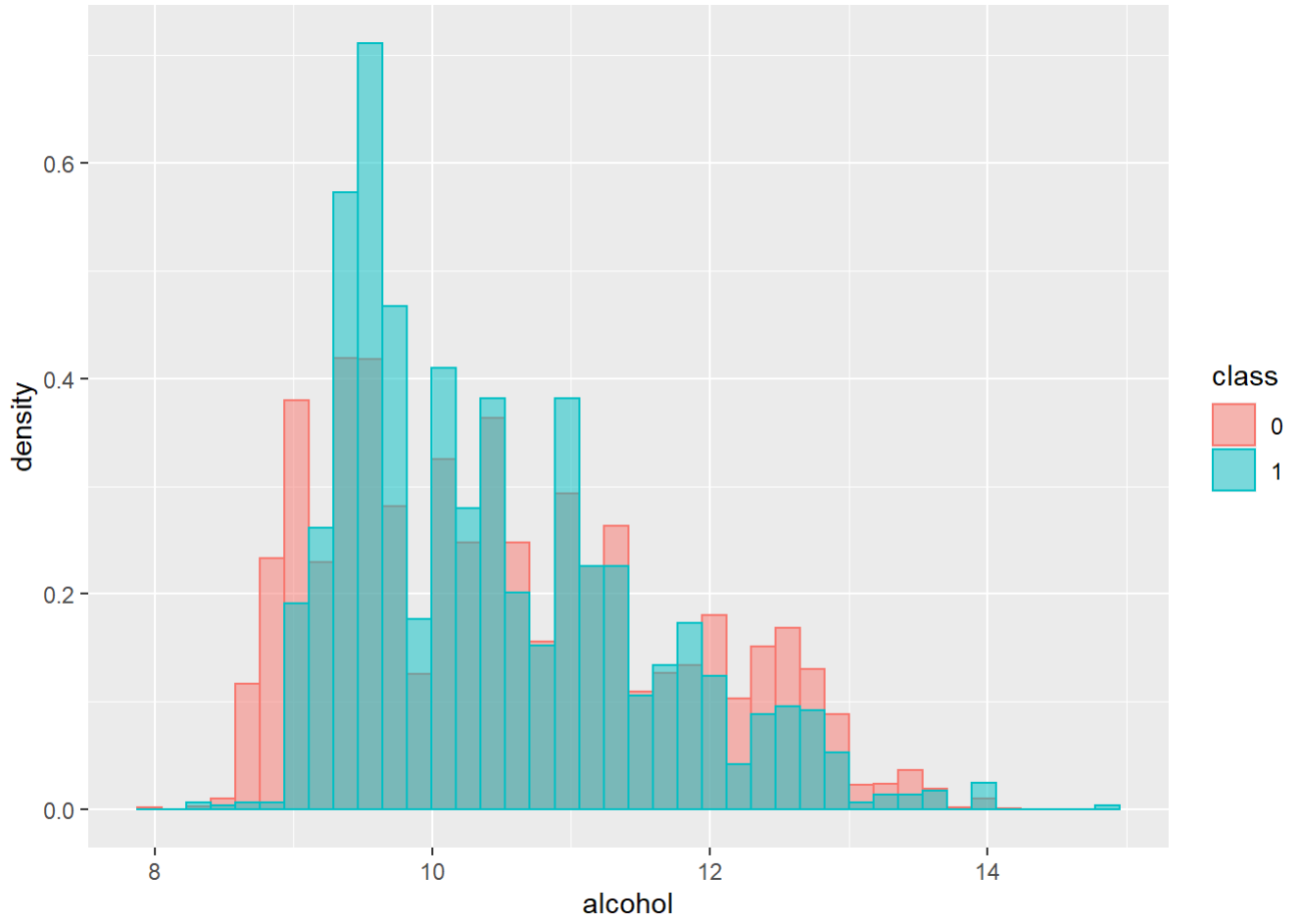

density
free.sulfur.dioxide
class
0
1
0.05
0.04
0.03
0.02
0.01
0.00
0
100
200
300


density
0.015
0.010
0.005
0.000
0
100
200
300
400
total.sulfur.dioxide
class
0
1


density
200
150
100
50
0
1.00
1.02
1.04
density
class
0
1


density
pH
class
0
1
3.0
3.5
4.0
0
1
2
3


density
sulphates
class
0
1
0.5
1.0
1.5
2.0
0
1
2
3
4



Scale and Parition the data

I scaled all the predictors and left the quality attribute unscaled because it is an outcome. Simiraliy I did not scale the binary variable identifying the red and white wines.

Then I randomly partitioned the data set, 80% to the training set and 20% to the test set.

```
# scale the data
rw_comb_scale <- cbind(scale(rw_comb[,1:11]), rw_comb[,12:13])

# In case I needed to reference the original red and wine data sets, I scaled these too, but I d
idn't wind up using them.
#red_scale <- cbind(scale(red[,1:11]), red[,12])
#white_scale <- cbind(scale(white[,1:11]), white[,12])

# partition the data into the training set and test set
set.seed(12)
test_idx <- createDataPartition(rw_comb_scale$class, times=1, p=.2, list=FALSE)
rw_comb_test <- rw_comb_scale[test_idx,]
rw_comb_train <- rw_comb_scale[-test_idx,]
```

Establish a baseline RMSE

I took the mean and median of the training set and used these values as the prediction value for both the training and test sets. Recall, the smaller the RMSE the better. As expected the mean as a predictor yields a slighter better RMSE than the median. Oddly, there is a slightly better RMSE on the test set for the median than the training set. The median is not ordinarily used as the model. I basically just wanted to show that the mean is gives a slightly better RMSE than the median. However, if different models

```
#take the average of the training set
avg <- mean(rw_comb_train$quality)
print(avg)
```

```
## [1] 5.82259
```

```
med <- median(rw_comb_train$quality)
print(med)
```

```
## [1] 6
```

```
#compute the RMSE of on the average
rmse_train_avg <- sqrt(mean((rw_comb_train$quality - avg)^2))
rmse_test_avg <- sqrt(mean((rw_comb_test$quality - avg)^2))

rmse_train_med <- sqrt(mean((rw_comb_train$quality - med)^2))
rmse_test_med <- sqrt(mean((rw_comb_test$quality - med)^2))

#build a table to compare RMSE
rmse_results <- data_frame(method = "Just the average on training set", RMSE = rmse_train_avg)
```

```
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
rmse_results <- bind_rows(rmse_results, data_frame(method = "Just the average on test set", RMSE
= rmse_test_avg))
rmse_results <- bind_rows(rmse_results, data_frame(method = "Just the median on training set", R
MSE = rmse_train_med))
rmse_results <- bind_rows(rmse_results, data_frame(method = "Just the median on test set", RMSE
 = rmse_test_med))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average on training set	0.8692055
Just the average on test set	0.8889809
Just the median on training set	0.8871260
Just the median on test set	0.9106211

Linear regression

I ran a simple linear regression model. The RMSE on the test set went down to .736, and this is slightly higher than the training set which is what we would expect. There is only p-value that is over .05, which is citric acid. It has a high p-value of .536, which you cannot reject the null hypothesis that the coefficient is not zero. It is easier to explain a small p-value, which means we can strongly reject the null hypothesis that the coefficient is zero. However, the citric accid coefficient is only -.008, which is the smallest coefficient in the model. Therefore, I don't think there is negative impact.

However, what is trouble about this model is the low R-squared. That tells me linear regression may not be the best model.

```
# linear regression to predict quality
lm_model <- lm(quality~., data=rw_comb_train)
summary(lm_model)
```

```
##
## Call:
## lm(formula = quality ~ ., data = rw_comb_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.7581 -0.4674 -0.0363  0.4654  3.0112
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)           5.741641   0.018468 310.899  < 2e-16 ***
## fixed.acidity         0.096166   0.022786   4.220 2.48e-05 ***
## volatile.acidity     -0.246236   0.014807 -16.630  < 2e-16 ***
## citric.acid          -0.009042   0.012852  -0.704  0.48173
## residual.sugar        0.286834   0.031203   9.193  < 2e-16 ***
## chlorides            -0.033512   0.012693  -2.640  0.00831 **
## free.sulfur.dioxide   0.092248   0.015204   6.067 1.39e-09 ***
## total.sulfur.dioxide -0.093614   0.020373  -4.595 4.43e-06 ***
## density              -0.281625   0.047068  -5.983 2.33e-09 ***
## pH                    0.067756   0.016209   4.180 2.96e-05 ***
## sulphates             0.099499   0.012673   7.851 4.98e-15 ***
## alcohol               0.274085   0.023622  11.603  < 2e-16 ***
## class1                0.351202   0.063144   5.562 2.80e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7296 on 5184 degrees of freedom
## Multiple R-squared:  0.2971, Adjusted R-squared:  0.2955
## F-statistic: 182.6 on 12 and 5184 DF,  p-value: < 2.2e-16
```

```
y_hat_train <- predict(lm_model)

y_hat_test <- predict(lm_model, rw_comb_test)


#calculate the RMSE on the linear regression model and at to rmse_results table
rmse_train <- sqrt(mean((rw_comb_train$quality - y_hat_train)^2))
rmse_test <- sqrt(mean((rw_comb_test$quality - y_hat_test)^2))
rmse_results <- bind_rows(rmse_results, data_frame(method = "Linear regression on training set",
RMSE = rmse_train))
rmse_results <- bind_rows(rmse_results, data_frame(method = "Linear regression on test set", RMS
E = rmse_test))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average on training set	0.8692055
Just the average on test set	0.8889809
Just the median on training set	0.8871260
Just the median on test set	0.9106211
Linear regression on training set	0.7287282
Linear regression on test set	0.7478543

Gaining a better understanding of data and lm model

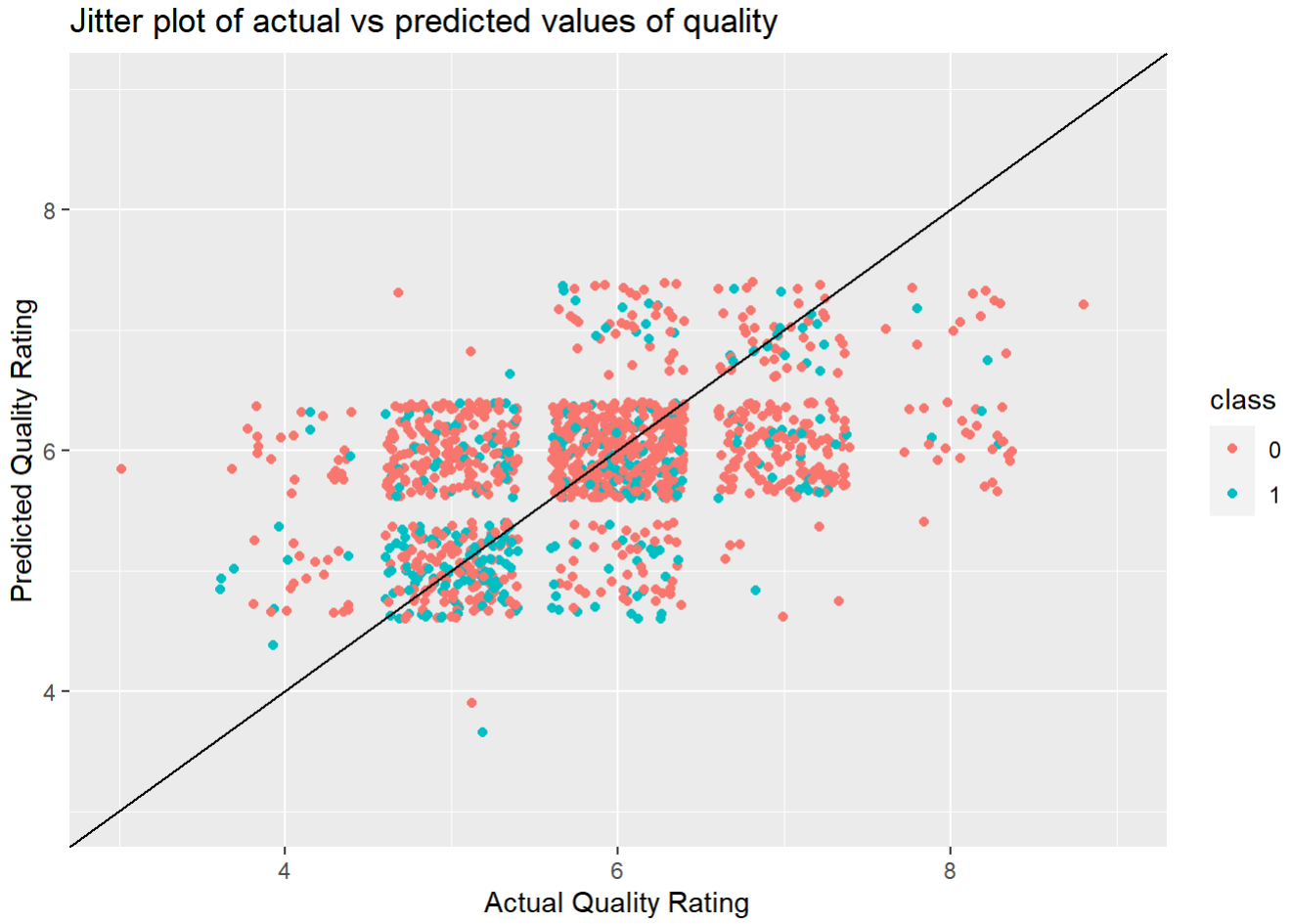
Normally, when predicting a continuous outcome with models such as linear regression, the goal is not to measure an exact prediction. Rather the goal is to minimize the error. However, the quality rating only has 7 possible outcomes. So I rounded the prediction value to the nearest integer and conducted an accuracy test of how many predictions were exactly right. I got about 53% accuracy on the test set. Then I ran a jitter plot. I noticed that most predicted values are either 5 or 6, some 7s and very few of the other 4 values. So most of the predicted values are within +or- 1 of the actual value.

```
# what percentage of the rating can be predicted exactly from the rounded y_hat
train_accuracy <- mean(round(y_hat_train) == rw_comb_train$quality)
test_accuracy <- mean(round(y_hat_test) == rw_comb_test$quality)
accuracy_results <- data_frame(method = "Linear regression on training set", Accuracy = train_ac
curacy)
accuracy_results <- bind_rows(accuracy_results, data_frame(method = "Linear regression on test s
et", Accuracy=test_accuracy))
accuracy_results
```

```
## # A tibble: 2 x 2
##   method                             Accuracy
##   <chr>                                 <dbl>
## 1 Linear regression on training set    0.534
## 2 Linear regression on test set        0.528
```

```
#add the prediction to the the column
rw_comb_test_pred <- rw_comb_test %>% mutate(y_hat = y_hat_test)

#review the plot of predicted (y-axis) as a function of actual (x-axis) quality ratings
ggplot(rw_comb_test_pred, aes(quality, round(y_hat_test), col=class)) + geom_jitter()+xlim(3,9)+
ylim(3,9)+
  labs(x="Actual Quality Rating", y="Predicted Quality Rating", title='Jitter plot of actual vs
 predicted values of quality')+
  geom_abline()
```



Accuracy within range of + or - 1

When I give the accuracy a range of + or - 1, it goes up to 95%. There is actually no reason to round the accuracy to the nearest integer, however when I ran this part of the model without rounding, I only got a 90% accuracy. This is of marginal value since over 90% of the quality ratings are in the 5-7 range, so if you just guessed 6 +or- 1 for every observation you would get about 93%.

```
# what about trying to predict the quality within + or - 1
train_acc_range <- mean( (round(y_hat_train) <= rw_comb_train$quality+1) & (round(y_hat_train) >
= rw_comb_train$quality - 1))
test_acc_range <- mean( (round(y_hat_test) <= rw_comb_test$quality+1) & (round(y_hat_test) >= rw
_comb_test$quality - 1))
accuracy_results <- bind_rows(accuracy_results, data_frame(method = "Linear regresssion on train
ing set with +- 1 range",
                                                           Accuracy = train_acc_range))
accuracy_results <- bind_rows(accuracy_results, data_frame(method = "Linear regression on test s
et with +-1 range",
                                                           Accuracy = test_acc_range))
accuracy_results %>% knitr::kable()
```

method	Accuracy
Linear regression on training set	0.5337695
Linear regression on test set	0.5284615
Linear regresssion on training set with +- 1 range	0.9549740
Linear regression on test set with +-1 range	0.9469231

Variable Selection

Variable selection is an important aspect of linear regression. There are several methods for variable selection, such as elastic net, ridge, and lasso. In this case, I took a simplistic approach and removed the two variables with p-values over .05, citric acid and chlorides. This is not guaranteed to give us a better prediction, but it is worth a try. In this case, removing these two attributes, citric acid and chlorides, did not signficantly change the RMSE nor the accuracy. They actually got slightly worse.

```
# variable selection
var_sel_model <- lm(quality~., data=rw_comb_train[, -c(3,5)])
summary(var_sel_model)
```

```
##
## Call:
## lm(formula = quality ~ ., data = rw_comb_train[, -c(3, 5)])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.7370 -0.4701 -0.0348  0.4626  3.0110
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)            5.74559    0.01833 313.450  < 2e-16 ***
## fixed.acidity          0.09986    0.02208   4.523 6.22e-06 ***
## volatile.acidity      -0.24602    0.01402 -17.554  < 2e-16 ***
## residual.sugar         0.29936    0.03091   9.684  < 2e-16 ***
## free.sulfur.dioxide    0.09125    0.01521   6.000 2.11e-09 ***
## total.sulfur.dioxide -0.09432    0.02029  -4.648 3.44e-06 ***
## density               -0.29869    0.04671  -6.395 1.75e-10 ***
## pH                     0.07723    0.01588   4.865 1.18e-06 ***
## sulphates              0.09207    0.01241   7.417 1.40e-13 ***
## alcohol                0.27441    0.02346  11.695  < 2e-16 ***
## class1                 0.33490    0.06243   5.364 8.49e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7301 on 5186 degrees of freedom
## Multiple R-squared:  0.296,  Adjusted R-squared:  0.2946
## F-statistic:   218 on 10 and 5186 DF,  p-value: < 2.2e-16
```

```
y_hat_train_var_sel <- predict(var_sel_model)

y_hat_test_var_sel <- predict(var_sel_model, rw_comb_test)

#add variable selection rmse to rmse_results table
rmse_train_Var_sel <- sqrt(mean((rw_comb_train$quality - y_hat_train_var_sel)^2))
rmse_test_var_sel <- sqrt(mean((rw_comb_test$quality - y_hat_test_var_sel)^2))

rmse_results <- bind_rows(rmse_results, data_frame(method = "Linear regression after variable se
lection on the training set",
                                                   RMSE = rmse_train_Var_sel))
rmse_results <- bind_rows(rmse_results, data_frame(method = "Linear regression after variable se
lection on the test set",
                                                   RMSE = rmse_test_var_sel))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average on training set	0.8692055
Just the average on test set	0.8889809
Just the median on training set	0.8871260
Just the median on test set	0.9106211

method	RMSE
Linear regression on training set	0.7287282
Linear regression on test set	0.7478543
Linear regression after variable selection on the training set	0.7293082
Linear regression after variable selection on the test set	0.7473045

```
rw_comb_test_pred_Var_sel <- rw_comb_test %>% mutate(y_hat = y_hat_test_var_sel)

# what percentage of the rating can be predicted exactly from the rounded y_hat
train_acc_var_sel <- mean(round(y_hat_train_var_sel) == rw_comb_train$quality)
test_acc_var_sel <- mean(round(y_hat_test_var_sel) == rw_comb_test$quality)
accuracy_results <- bind_rows(accuracy_results, data_frame(method = "Linear regresssion on train
ing set after variable selection",
                                                           Accuracy = train_acc_var_sel))
accuracy_results <- bind_rows(accuracy_results, data_frame(method = "Linear regression on test s
et after variable selection",
                                                           Accuracy = test_acc_var_sel))
# what about trying to predict the quality within + or - 1
train_acc_var_sel_range <- mean((round(y_hat_train_var_sel) <= rw_comb_train$quality+1) &
                                  (round(y_hat_train_var_sel) >= rw_comb_train$quality - 1))
test_acc_var_sel_range <- mean((round(y_hat_test_var_sel) <= rw_comb_test$quality+1) &
                                 (round(y_hat_test_var_sel) >= rw_comb_test$quality - 1))

accuracy_results <- bind_rows(accuracy_results,
                              data_frame(method = "Linear regresssion on training set after vari
able selection with +- 1 range",
                                                           Accuracy = train_acc_var_sel_range))
accuracy_results <- bind_rows(accuracy_results,
                              data_frame(method = "Linear regression on test set after variable
 selection with +-1 range",
                                                           Accuracy = test_acc_var_sel_range))



accuracy_results %>% knitr::kable()
```

method	Accuracy
Linear regression on training set	0.5337695
Linear regression on test set	0.5284615
Linear regresssion on training set with +- 1 range	0.9549740
Linear regression on test set with +-1 range	0.9469231
Linear regresssion on training set after variable selection	0.5312680
Linear regression on test set after variable selection	0.5238462
Linear regresssion on training set after variable selection with +- 1 range	0.9532423

method	Accuracy
Linear regression on test set after variable selection with +-1 range	0.9469231

Classifiers

Logistic Regression

I used logistic regression as the first classifier model to classify red wines and white wines based on the attributes. Logistic regression predicts the chance something will happen. Recall with this wine set, we are using red wine to simulate a positive test for a disease. So I start by rounding. If the model predicts a value of .5 or greater, round up to 1 and call that prediction for a positive result. Otherwise round down to zero the prediction is for a negative result.

However, we can also weigh the cost of a false positive vs. a false negative. If a patient has a false positive, he she can likely get a second opinion and walk away no harm done, save some temporary period of high stress. However, if a patient gets a false negative, he/she probably thinks things are fine while the disease is actually getting worse. So we want to reduce false negatives even at the expense of reducing the overall accuracy. This is called specificity, the models ability to correctly identify people without the disease. I created a tibble that measures specificity, and also includes the false negatives and false positives. The goal is to reduce the false negatives, but the trade off is more false positives.

One thing I like about logistic regression is we can control the threshold that rounds to 1 or stays 0. We don't have to simply round off at .5. For illustrative purposes, I used .1 as the round off point. Anything prediction by the model greater than or equal to .1 is round to 1, and anything less is a zero. We reduce the false negatives by 7, but at the expense of increasing the false positives by 14. Each false negative reduced causes two false positives. Whether is a good trade off depends. The point is to show how logistic regression can be used to find not only the best overall accuracy, but also the best specificity. You could also optimize the sensitivity by setting a threshold closer to 1, and reduce false positives.

```
# logistic regression as a classifier
logreg <- glm(class~., data=rw_comb_train, family=binomial(link = "logit"))
summary(logreg)
```

```
##
## Call:
## glm(formula = class ~ ., family = binomial(link = "logit"), data = rw_comb_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.8376  -0.0506  -0.0128  -0.0004   5.8601
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)           -6.7022     1.3559  -4.943 7.69e-07 ***
## fixed.acidity         -0.4026     0.3810  -1.057   0.2907
## volatile.acidity       1.2437     0.1998   6.224 4.86e-10 ***
## citric.acid           -0.3040     0.1887  -1.611   0.1071
## residual.sugar        -4.8705     0.6371  -7.645 2.10e-14 ***
## chlorides              0.7838     0.1572   4.986 6.16e-07 ***
## free.sulfur.dioxide    1.1773     0.2916   4.037 5.41e-05 ***
## total.sulfur.dioxide  -3.1272     0.3327  -9.401  < 2e-16 ***
## density                5.0965     0.6603   7.719 1.17e-14 ***
## pH                    -0.1355     0.2676  -0.506   0.6126
## sulphates              0.4648     0.2255   2.061   0.0393 *
## alcohol                1.8850     0.3776   4.992 5.97e-07 ***
## quality                0.3754     0.2271   1.653   0.0983 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 5800.0  on 5196  degrees of freedom
## Residual deviance:  325.3  on 5184  degrees of freedom
## AIC: 351.3
##
## Number of Fisher Scoring iterations: 10
```

```
y_hat_train_log <- round(predict(logreg, type = 'response'))
y_hat_test_log <- round(predict(logreg, newdata=rw_comb_test, type = 'response'))

train_log_acc <- mean(y_hat_train_log == rw_comb_train$class)
test_log_acc <- mean(y_hat_test_log == rw_comb_test$class)

#confusion matrix on test set
confusionMatrix(as.factor(y_hat_test_log), as.factor(rw_comb_test$class))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 976  11
##          1   4 309
##
##                Accuracy : 0.9885
##                  95% CI : (0.981, 0.9935)
##     No Information Rate : 0.7538
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9687
##
##  Mcnemar's Test P-Value : 0.1213
##
##             Sensitivity : 0.9959
##             Specificity : 0.9656
##          Pos Pred Value : 0.9889
##          Neg Pred Value : 0.9872
##              Prevalence : 0.7538
##          Detection Rate : 0.7508
##    Detection Prevalence : 0.7592
##       Balanced Accuracy : 0.9808
##
##        'Positive' Class : 0
##
```

```
log_conf_mat <- as.matrix(confusionMatrix(as.factor(y_hat_test_log), as.factor(rw_comb_test$clas
s)))

# Manually calculate the specificity
logreg_spec <- log_conf_mat[2,2] / (log_conf_mat[2,2] + log_conf_mat[1,2])

rw_classifier_results <- data_frame(Method = "Logistic Regression with prediction rounded", Spec
ificity = logreg_spec,
                                    False_Negatives = log_conf_mat[1,2],
                                    False_Positives = log_conf_mat[2,1])

# get the specificity to be 1
y_hat_test_log_spec <- as.numeric(predict(logreg, newdata=rw_comb_test, type = 'response')>=.1)
confusionMatrix(as.factor(y_hat_test_log_spec), as.factor(rw_comb_test$class))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 962   4
##          1  18 316
##
##                Accuracy : 0.9831
##                  95% CI : (0.9745, 0.9894)
##     No Information Rate : 0.7538
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9551
##
##  Mcnemar's Test P-Value : 0.005578
##
##             Sensitivity : 0.9816
##             Specificity : 0.9875
##          Pos Pred Value : 0.9959
##          Neg Pred Value : 0.9461
##              Prevalence : 0.7538
##          Detection Rate : 0.7400
##    Detection Prevalence : 0.7431
##       Balanced Accuracy : 0.9846
##
##        'Positive' Class : 0
##
```

```
log_spec_conf_mat <- as.matrix(confusionMatrix(as.factor(y_hat_test_log_spec), as.factor(rw_comb
_test$class)))

logreg_spec_adjusted <- log_spec_conf_mat[2,2] / (log_spec_conf_mat[2,2] + log_spec_conf_mat[1,2
])
rw_classifier_results <- bind_rows(rw_classifier_results,
                                   data_frame(Method = 'Logistc Regression with prediction adjus
ted for better specification',
                                              Specificity = logreg_spec_adjusted,
                                              False_Negatives = log_spec_conf_mat[1,2],
                                              False_Positives = log_spec_conf_mat[2,1]))
rw_classifier_results %>% knitr::kable()
```

Method	Specificity	False_Negatives	False_Positives
Logistic Regression with prediction rounded	0.965625	11	4
Logistc Regression with prediction adjusted for better specification	0.987500	4	18

Support Vector Machine

SVMs are another common classier. I just ran the basic model here. Vanilla dot is my kernel of choice for linear models, but there are several to choose from. C controls optimized the tradeoff between he decision boundary and penalty for mis-classifying points. I experimented different values of C and didn't see much change so I selected 1. This turns out to be the best overall model overall.

```
#support vector machine
svm_mod <- ksvm(class~., data = rw_comb_train,
                type="C-svc", kernel="vanilladot", C=1)
```

```
##  Setting default kernel parameters
```

```
y_hat_test_svm <- predict(svm_mod, newdata=rw_comb_test)
confusionMatrix(as.factor(y_hat_test_svm), as.factor(rw_comb_test$class))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 977   6
##          1   3 314
##
##                Accuracy : 0.9931
##                  95% CI : (0.9869, 0.9968)
##     No Information Rate : 0.7538
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9813
##
##  Mcnemar's Test P-Value : 0.505
##
##             Sensitivity : 0.9969
##             Specificity : 0.9812
##          Pos Pred Value : 0.9939
##          Neg Pred Value : 0.9905
##              Prevalence : 0.7538
##          Detection Rate : 0.7515
##    Detection Prevalence : 0.7562
##       Balanced Accuracy : 0.9891
##
##        'Positive' Class : 0
##
```

```
svm_conf_mat <- as.matrix(confusionMatrix(as.factor(y_hat_test_svm), as.factor(rw_comb_test$clas
s)))
svm_spec <- svm_conf_mat[2,2]/(svm_conf_mat[1,2] + svm_conf_mat[2,2])

rw_classifier_results <- bind_rows(rw_classifier_results,
                                   data_frame(Method = 'Support Vector Machine',
                                              Specificity = svm_spec,
                                              False_Negatives = svm_conf_mat[1,2],
                                              False_Positives = svm_conf_mat[2,1]))
rw_classifier_results %>% knitr::kable()
```

Method	Specificity	False_Negatives	False_Positives
Logistic Regression with prediction rounded	0.965625	11	4
Logistc Regression with prediction adjusted for better specification	0.987500	4	18
Support Vector Machine	0.981250	6	3

K nearest neighbor

K nearest neighbor is another good classifier model. I set up a for loop to test values from k from 1 to 30. I got the best specificity for values of 1 through 4. This means the same number of false negatives for each of these values of k. So the next step is to run the model again with each of these four values and see which one gives the best overall accuracy, but they turned out to be all the same so I used k=4. While not quite as good as the SVM, this is a very good model.

```
# knn testing for best k
kmax <- 30
specificity <- rep(0,kmax)
for (k in 1:kmax) {
  model <- kknn(formula = class~.,train = rw_comb_train,test=rw_comb_test,
                k=k) # number of neighbors
  y_hat <- fitted(model)
  conf_mat <- as.matrix(confusionMatrix(as.factor(y_hat), as.factor(rw_comb_test$class)))
  specificity[k] <- conf_mat[2,2]/(conf_mat[1,2]+conf_mat[2,2])
}
print("Specificity rates for values of k from 1 to 30")
```

```
## [1] "Specificity rates for values of k from 1 to 30"
```

```
specificity
```

```
##  [1] 0.978125 0.978125 0.978125 0.978125 0.978125 0.978125 0.978125 0.978125
##  [9] 0.978125 0.978125 0.978125 0.978125 0.978125 0.975000 0.975000 0.975000
## [17] 0.975000 0.975000 0.975000 0.975000 0.975000 0.975000 0.975000 0.975000
## [25] 0.975000 0.975000 0.975000 0.975000 0.975000 0.975000
```

```
print('Best specificity rate')
```

```
## [1] "Best specificity rate"
```

```
specificity[which.max(specificity)]
```

```
## [1] 0.978125
```

```
# set up knn model with k = 4
knn_model <- kknn(formula = class~.,train = rw_comb_train,test=rw_comb_test,
              k = 4) # number of neighbors
y_hat_knn <- fitted(knn_model)
confusionMatrix(as.factor(y_hat_knn), as.factor(rw_comb_test$class))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 974   7
##          1   6 313
##
##                Accuracy : 0.99
##                  95% CI : (0.983, 0.9947)
##     No Information Rate : 0.7538
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.973
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9939
##             Specificity : 0.9781
##          Pos Pred Value : 0.9929
##          Neg Pred Value : 0.9812
##              Prevalence : 0.7538
##          Detection Rate : 0.7492
##    Detection Prevalence : 0.7546
##       Balanced Accuracy : 0.9860
##
##        'Positive' Class : 0
##
```

```
knn_conf_mat <- as.matrix(confusionMatrix(as.factor(y_hat_knn), as.factor(rw_comb_test$class)))
knn_spec <- knn_conf_mat[2,2] / (knn_conf_mat[2,2] + knn_conf_mat[1,2])
rw_classifier_results <- bind_rows(rw_classifier_results,
                                   data_frame(Method = 'K nearest neighbor with k=4',
                                              Specificity = knn_spec,
                                              False_Negatives = knn_conf_mat[1,2],
                                              False_Positives = knn_conf_mat[2,1]))
rw_classifier_results %>% knitr::kable()
```

Method	Specificity	False_Negatives	False_Positives
Logistic Regression with prediction rounded	0.965625	11	4
Logistc Regression with prediction adjusted for better specification	0.987500	4	18
Support Vector Machine	0.981250	6	3
K nearest neighbor with k=4	0.978125	7	6

Decision Tree as a classifer

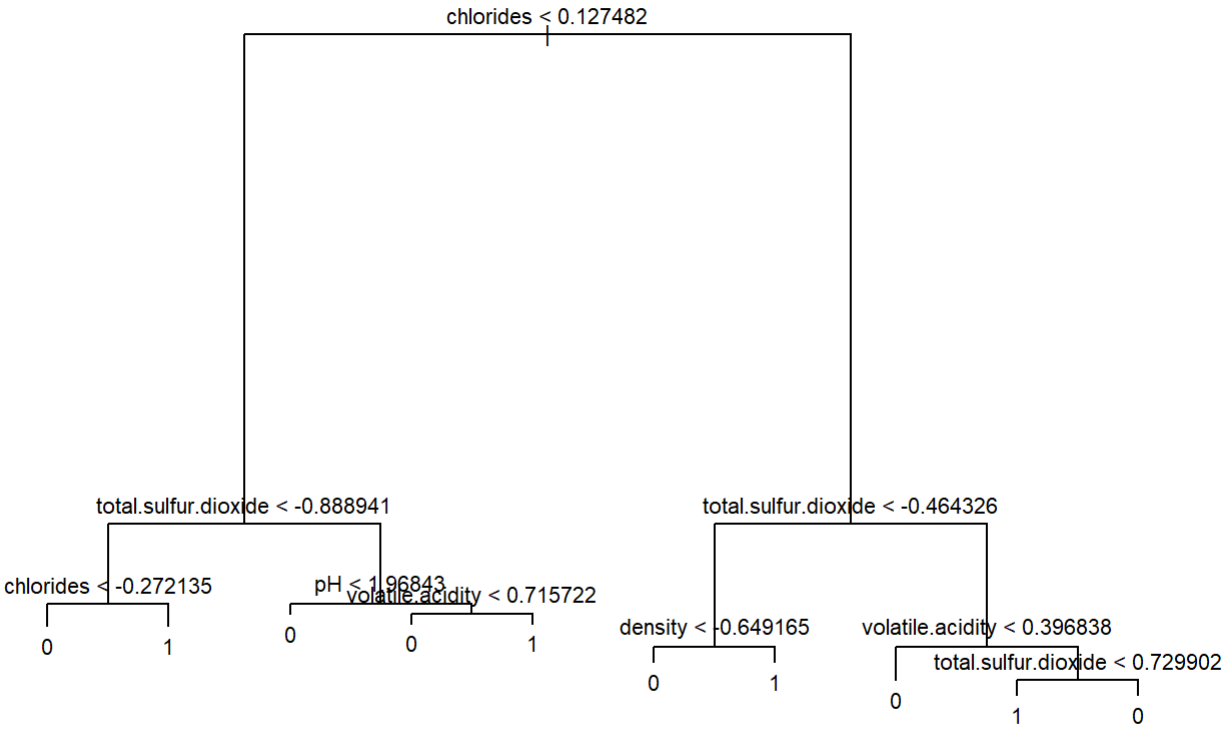
Decision trees are another good classifier, although in this case the tree did not perform as well as the other models. It is interesting to look at the attributes the tree uses for branches. For example, the chlorides attribute is one that removed from the linear regression model because the p-value was above the .05 threshold, yet in the decision tree it the the top branch and arguably the most important variable. Similar to logistic regression the output is a figure between the categorical values, which in this case is a fraction between 0 and 1. Therefore, we do standard rounding or round at specific threshhold to reduce false negative or false positives in the even either is more important than overall accuracy.

It turns out this model is not good. There are 22 false negatives compared to other classifiers that had only 6 or 7. And there are more false positives as well. I adjusted the threshold to .05, and reduced the false negatives from 22 to 15, but at relatively high cost of false positives. While false positives are less important, each false positive represent telling somebody they might have a disease as a result of the test being too conservative.

Another benefit of decision trees is you can prune the tree to a selected number of leaves. I pruned the tree to use only 4 leaves. The model only used two variables, chlorides and total sulfur dioxide, and still came up with the same results as the ordinal tree. While it is not the best model, the results pretty good when considering it ony uses 2 out of 12 predictors.

```
#tree for classification

wine_tree <- tree(class~., data=rw_comb_train)
par(cex = .7)
plot(wine_tree)
text(wine_tree)
```




```
wine_tree$frame
```

```
##                     var    n        dev yval splits.cutleft splits.cutright
## 1             chlorides 5197 5799.99540    0      <0.127482       >0.127482
## 2  total.sulfur.dioxide 3759 1130.18857    0     <-0.888941      >-0.888941
## 4             chlorides  194  268.61112    1     <-0.272135      >-0.272135
## 8                 <leaf>   95   63.93419    0
## 9                 <leaf>   99   55.58624    1
## 5                     pH 3565  336.83762    0       <1.96843        >1.96843
## 10                <leaf> 3490  193.42364    0
## 11     volatile.acidity   75   72.20300    0      <0.715722       >0.715722
## 22                <leaf>   62   10.23805    0
## 23                <leaf>   13    0.00000    1
## 3  total.sulfur.dioxide 1438 1443.03049    1     <-0.464326      >-0.464326
## 6                density 1038  139.71915    1     <-0.649165      >-0.649165
## 12                <leaf>   17   20.59711    0
## 13                <leaf> 1021   15.85610    1
## 7      volatile.acidity  400  495.28053    0      <0.396838       >0.396838
## 14                <leaf>  233   69.66767    0
## 15 total.sulfur.dioxide  167  205.53079    1      <0.729902       >0.729902
## 30                <leaf>  134  105.73476    1
## 31                <leaf>   33    0.00000    0
##         yprob.0      yprob.1
## 1  0.7538964787 0.2461035213
## 2  0.9654163341 0.0345836659
## 4  0.4793814433 0.5206185567
## 8  0.8947368421 0.1052631579
## 9  0.0808080808 0.9191919192
## 5  0.9918653576 0.0081346424
## 10 0.9957020057 0.0042979943
## 11 0.8133333333 0.1866666667
## 22 0.9838709677 0.0161290323
## 23 0.0000000000 1.0000000000
## 3  0.2009735744 0.7990264256
## 6  0.0125240848 0.9874759152
## 12 0.7058823529 0.2941176471
## 13 0.0009794319 0.9990205681
## 7  0.6900000000 0.3100000000
## 14 0.9656652361 0.0343347639
## 15 0.3053892216 0.6946107784
## 30 0.1343283582 0.8656716418
## 31 1.0000000000 0.0000000000
```

```
y_hat_tree <- round(predict(wine_tree, newdata=rw_comb_test))
print('Confusion matrix for tree classifier with prediction rounded to 0 or 1:')
```

```
## [1] "Confusion matrix for tree classifier with prediction rounded to 0 or 1:"
```

```
print(confusionMatrix(as.factor(y_hat_tree[,2]), as.factor(rw_comb_test$class)))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 971  22
##          1   9 298
##
##                Accuracy : 0.9762
##                  95% CI : (0.9663, 0.9837)
##     No Information Rate : 0.7538
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9349
##
##  Mcnemar's Test P-Value : 0.03114
##
##             Sensitivity : 0.9908
##             Specificity : 0.9313
##          Pos Pred Value : 0.9778
##          Neg Pred Value : 0.9707
##              Prevalence : 0.7538
##          Detection Rate : 0.7469
##    Detection Prevalence : 0.7638
##       Balanced Accuracy : 0.9610
##
##        'Positive' Class : 0
##
```

```
tree_conf_mat <- as.matrix(confusionMatrix(as.factor(y_hat_tree[,2]), as.factor(rw_comb_test$cla
ss)))
tree_spec <- tree_conf_mat[2,2] / (tree_conf_mat[2,2] + tree_conf_mat[1,2])
rw_classifier_results <- bind_rows(rw_classifier_results,
                                   data_frame(Method = 'Decision tree classifier',
                                              Specificity = tree_spec,
                                              False_Negatives = tree_conf_mat[1,2],
                                              False_Positives = tree_conf_mat[2,1]))

y_hat_tree_spec <- as.integer(predict(wine_tree, newdata=rw_comb_test)>=.05)
print('Confusion matrix for tree classiier with rounding threshold set to .05 instead of .5 for
 better specificity:')
```

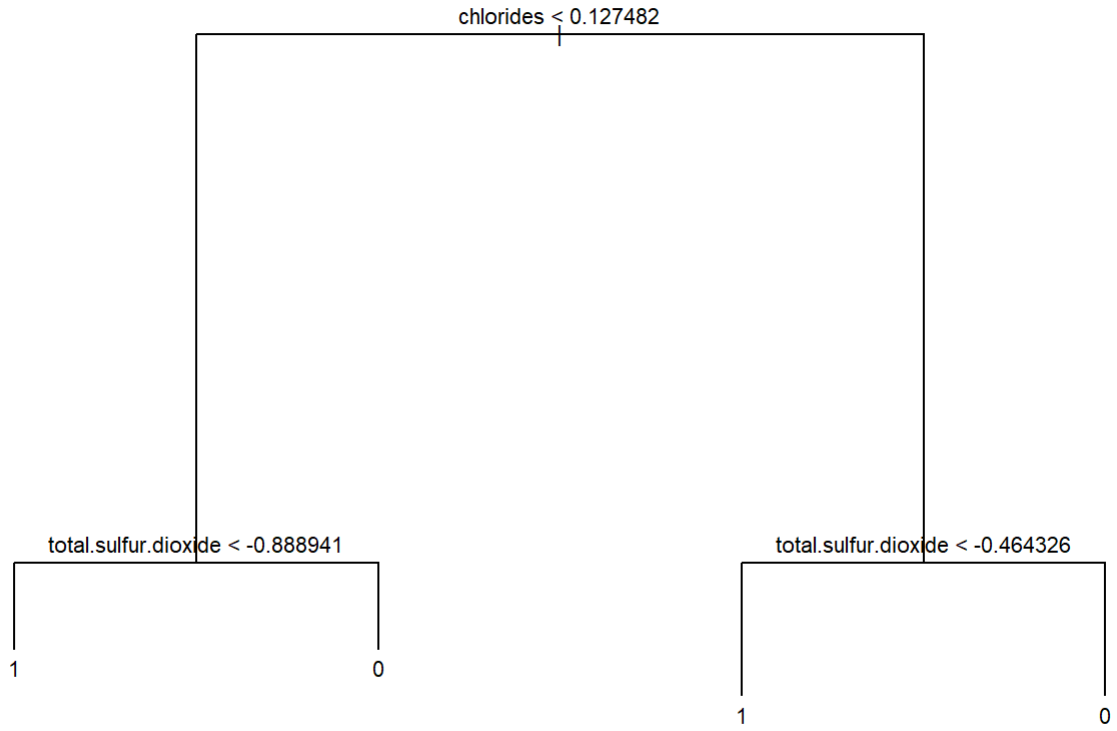
```
## [1] "Confusion matrix for tree classiier with rounding threshold set to .05 instead of .5 for
better specificity:"
```

```
print(confusionMatrix(as.factor(y_hat_tree_spec[1301:2600]), as.factor(rw_comb_test$class)))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 945  15
##          1  35 305
##
##                Accuracy : 0.9615
##                  95% CI : (0.9496, 0.9713)
##     No Information Rate : 0.7538
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.8985
##
##  Mcnemar's Test P-Value : 0.00721
##
##             Sensitivity : 0.9643
##             Specificity : 0.9531
##          Pos Pred Value : 0.9844
##          Neg Pred Value : 0.8971
##              Prevalence : 0.7538
##          Detection Rate : 0.7269
##    Detection Prevalence : 0.7385
##       Balanced Accuracy : 0.9587
##
##        'Positive' Class : 0
##
```

```
tree_conf_mat_adj <- as.matrix(confusionMatrix(as.factor(y_hat_tree_spec[1301:2600]), as.factor
(rw_comb_test$class)))
tree_spec_adj <- tree_conf_mat_adj[2,2] / (tree_conf_mat_adj[2,2] + tree_conf_mat_adj[1,2])
rw_classifier_results <- bind_rows(rw_classifier_results,
                                   data_frame(Method = 'Decision tree classifier adjusted for be
st specificity',
                                              Specificity = tree_spec_adj,
                                              False_Negatives = tree_conf_mat_adj[1,2],
                                              False_Positives = tree_conf_mat_adj[2,1]))


#prune the tree
termnodes <- 4
prune.data <- prune.tree(wine_tree, best = termnodes)
plot(prune.data)
text(prune.data)
```




```
print(prune.data$frame)
```

```
##                   var    n       dev yval splits.cutleft splits.cutright
## 1            chlorides 5197 5799.9954    0      <0.127482       >0.127482
## 2 total.sulfur.dioxide 3759 1130.1886    0     <-0.888941      >-0.888941
## 4               <leaf>  194  268.6111    1
## 5               <leaf> 3565  336.8376    0
## 3 total.sulfur.dioxide 1438 1443.0305    1     <-0.464326      >-0.464326
## 6               <leaf> 1038  139.7191    1
## 7               <leaf>  400  495.2805    0
##       yprob.0     yprob.1
## 1 0.753896479 0.246103521
## 2 0.965416334 0.034583666
## 4 0.479381443 0.520618557
## 5 0.991865358 0.008134642
## 3 0.200973574 0.799026426
## 6 0.012524085 0.987475915
## 7 0.690000000 0.310000000
```

```
y_hat_tree_prune <- round(predict(wine_tree, newdata=rw_comb_test))
print('Confusion matrix for pruned decision tree, only 4 terminal nodes:')
```

```
## [1] "Confusion matrix for pruned decision tree, only 4 terminal nodes:"
```

```
print(confusionMatrix(as.factor(y_hat_tree[,2]), as.factor(rw_comb_test$class)))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 971  22
##          1   9 298
##
##                Accuracy : 0.9762
##                  95% CI : (0.9663, 0.9837)
##     No Information Rate : 0.7538
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9349
##
##  Mcnemar's Test P-Value : 0.03114
##
##             Sensitivity : 0.9908
##             Specificity : 0.9313
##          Pos Pred Value : 0.9778
##          Neg Pred Value : 0.9707
##              Prevalence : 0.7538
##          Detection Rate : 0.7469
##    Detection Prevalence : 0.7638
##       Balanced Accuracy : 0.9610
##
##        'Positive' Class : 0
##
```

```
tree_conf_mat_prune <- as.matrix(confusionMatrix(as.factor(y_hat_tree[,2]), as.factor(rw_comb_te
st$class)))
tree_spec_prune <- tree_conf_mat_prune[2,2] / (tree_conf_mat_prune[2,2] + tree_conf_mat_prune[1,
2])
rw_classifier_results <- bind_rows(rw_classifier_results,
                                   data_frame(Method = 'Decision tree classifier pruned',
                                              Specificity = tree_spec_adj,
                                              False_Negatives = tree_conf_mat_prune[1,2],
                                              False_Positives = tree_conf_mat_prune[2,1]))
rw_classifier_results %>% knitr::kable()
```

Method	Specificity	False_Negatives	False_Positives
Logistic Regression with prediction rounded	0.965625	11	4
Logistc Regression with prediction adjusted for better specification	0.987500	4	18
Support Vector Machine	0.981250	6	3
K nearest neighbor with k=4	0.978125	7	6

Method	Specificity	False_Negatives	False_Positives
Decision tree classifier	0.931250	22	9
Decision tree classifier adjusted for best specificity	0.953125	15	35
Decision tree classifier pruned	0.953125	22	9

Random Forest as a classifier

Random forest is also a good classifier. I often find that it provides a better model that decision trees. It does have a couple of drawbacks. First of all, sometimes you need to use a lot of trees to build the best model, but this can be a very slow process process. I like to use 1000 if it doesn't take a long time to run, although I typically don't see much of change between 500 and 1000. Also, you can't analyze the tree results as well as a decision tree where you can see exactly what the model is doing.

The mtry is the number of variables available for splitting at tree node. I found an mtry of have the number of variables is good. There are 12 predictor variables, so I used an mtyr of 6. You could set up a for loop and tune this to find the best mtry, but it starts to get very costly in terms of run time.

Finally, while you can't see a decision tree, you can still print out the importance of each variable. The higher numbers are better. As you can see,, chlorides and total sulfur dioxide, are the most important, which are the same two variables the decision tree used.

Finally, random forest does give us a better result than decision tree, although it is still not as good as some of the other models.

```
# random forest classification
set.seed(12)
wine_rf <- randomForest(class~., data = rw_comb_train, mtry = 6,
                        ntrees=1000, importance = TRUE)
y_hat_rf <- predict(wine_rf, newdata =  rw_comb_test)
importance(wine_rf)
```

```
##                              0         1 MeanDecreaseAccuracy MeanDecreaseGini
## fixed.acidity        17.639762 19.139263             25.23628        40.335630
## volatile.acidity     30.281391 35.068461             43.52617       178.579053
## citric.acid          17.127359 16.887180             21.54321        15.622056
## residual.sugar        8.136997 23.970431             24.01783        48.284896
## chlorides            54.598671 68.606483             84.07733       645.923227
## free.sulfur.dioxide  20.108979  7.361489             20.99594        53.649766
## total.sulfur.dioxide 63.773311 95.981706            111.98075       776.333403
## density              14.287356 42.125199             43.10212        70.545388
## pH                   21.259607 26.232902             31.35113        30.495000
## sulphates            29.503487 29.462156             34.84588        55.957113
## alcohol              15.196607 13.153447             19.25869        10.414562
## quality               7.705684  3.985811              8.66068         2.313271
```

```
print(confusionMatrix(as.factor(y_hat_rf), as.factor(rw_comb_test$class)) )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 976   9
##          1   4 311
##
##                Accuracy : 0.99
##                  95% CI : (0.983, 0.9947)
##     No Information Rate : 0.7538
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9729
##
##  Mcnemar's Test P-Value : 0.2673
##
##             Sensitivity : 0.9959
##             Specificity : 0.9719
##          Pos Pred Value : 0.9909
##          Neg Pred Value : 0.9873
##              Prevalence : 0.7538
##          Detection Rate : 0.7508
##    Detection Prevalence : 0.7577
##       Balanced Accuracy : 0.9839
##
##        'Positive' Class : 0
##
```

```
rf_conf_mat <- as.matrix(confusionMatrix(as.factor(y_hat_rf), as.factor(rw_comb_test$class))
)

rf_spec <- rf_conf_mat[2,2] / (rf_conf_mat[2,2]+rf_conf_mat[1,2])
rw_classifier_results <- bind_rows(rw_classifier_results,
                                   data_frame(Method = 'Random Forest classifier',
                                              Specificity = rf_spec,
                                              False_Negatives = rf_conf_mat[1,2],
                                              False_Positives = rf_conf_mat[2,1]))
rw_classifier_results %>% knitr::kable()
```

Method	Specificity	False_Negatives	False_Positives
Logistic Regression with prediction rounded	0.965625	11	4
Logistc Regression with prediction adjusted for better specification	0.987500	4	18
Support Vector Machine	0.981250	6	3
K nearest neighbor with k=4	0.978125	7	6
Decision tree classifier	0.931250	22	9
Decision tree classifier adjusted for best specificity	0.953125	15	35

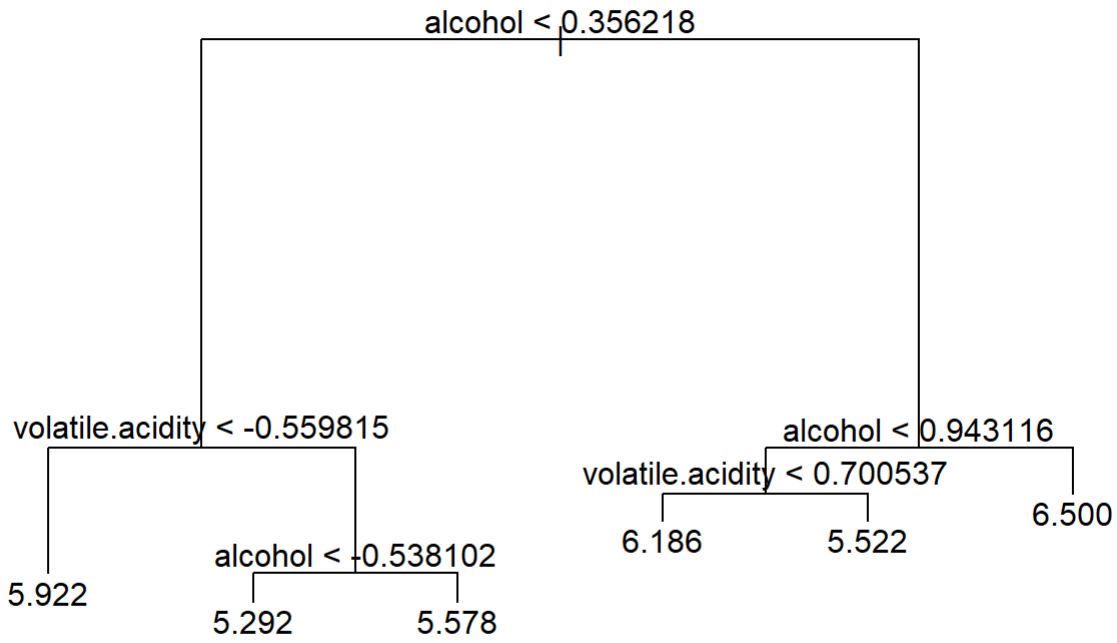
Method	Specificity	False_Negatives	False_Positives
Decision tree classifier pruned	0.953125	22	9
Random Forest classifier	0.971875	9	4

Decision tree as amount predictor

As mentioned the decision tree can also be used as a predictor. In this case neither the accuracy nor the RMSE show that this is best model. The RMSE is better than the mean as the predictor, but worse than the linear regression model. Likewise the accuracy is not as good as any of the linear regression models previously run, although it fairly close, all be in the low 50s%.

It is also interesting to observe that the predictors used here (alcohol, volatile acidity and free sulfur dioxide) are not the same as the predictors used in for the classification tree model.

```
#tree for quality prediction
wine_tree_acc <- tree(quality~., data=rw_comb_train)
plot(wine_tree_acc)
text(wine_tree_acc)
```




```
wine_tree$frame
```

```
##                     var    n        dev yval splits.cutleft splits.cutright
## 1             chlorides 5197 5799.99540    0      <0.127482       >0.127482
## 2  total.sulfur.dioxide 3759 1130.18857    0     <-0.888941      >-0.888941
## 4             chlorides  194  268.61112    1     <-0.272135      >-0.272135
## 8                 <leaf>   95   63.93419    0
## 9                 <leaf>   99   55.58624    1
## 5                     pH 3565  336.83762    0       <1.96843        >1.96843
## 10                <leaf> 3490  193.42364    0
## 11     volatile.acidity   75   72.20300    0      <0.715722       >0.715722
## 22                <leaf>   62   10.23805    0
## 23                <leaf>   13    0.00000    1
## 3  total.sulfur.dioxide 1438 1443.03049    1     <-0.464326      >-0.464326
## 6                density 1038  139.71915    1     <-0.649165      >-0.649165
## 12                <leaf>   17   20.59711    0
## 13                <leaf> 1021   15.85610    1
## 7      volatile.acidity  400  495.28053    0      <0.396838       >0.396838
## 14                <leaf>  233   69.66767    0
## 15 total.sulfur.dioxide  167  205.53079    1      <0.729902       >0.729902
## 30                <leaf>  134  105.73476    1
## 31                <leaf>   33    0.00000    0
##         yprob.0      yprob.1
## 1  0.7538964787 0.2461035213
## 2  0.9654163341 0.0345836659
## 4  0.4793814433 0.5206185567
## 8  0.8947368421 0.1052631579
## 9  0.0808080808 0.9191919192
## 5  0.9918653576 0.0081346424
## 10 0.9957020057 0.0042979943
## 11 0.8133333333 0.1866666667
## 22 0.9838709677 0.0161290323
## 23 0.0000000000 1.0000000000
## 3  0.2009735744 0.7990264256
## 6  0.0125240848 0.9874759152
## 12 0.7058823529 0.2941176471
## 13 0.0009794319 0.9990205681
## 7  0.6900000000 0.3100000000
## 14 0.9656652361 0.0343347639
## 15 0.3053892216 0.6946107784
## 30 0.1343283582 0.8656716418
## 31 1.0000000000 0.0000000000
```

```
y_hat_tree_acc <- round(predict(wine_tree_acc, newdata=rw_comb_test))

tree_acc <- mean(y_hat_tree_acc == rw_comb_test$quality)
tree_rmse <-   sqrt(mean((y_hat_tree_acc - rw_comb_test$quality)^2))

accuracy_results <- bind_rows(accuracy_results, data_frame(method = "Decision tree on test set",
Accuracy = tree_acc))
rmse_results <- bind_rows(rmse_results, data_frame(method="Decision tree on test set", RMSE = tr
ee_rmse))

accuracy_results %>% knitr::kable()
```

method	Accuracy
Linear regression on training set	0.5337695
Linear regression on test set	0.5284615
Linear regresssion on training set with +- 1 range	0.9549740
Linear regression on test set with +-1 range	0.9469231
Linear regresssion on training set after variable selection	0.5312680
Linear regression on test set after variable selection	0.5238462
Linear regresssion on training set after variable selection with +- 1 range	0.9532423
Linear regression on test set after variable selection with +-1 range	0.9469231
Decision tree on test set	0.5376923

```
rmse_results  %>% knitr::kable()
```

method	RMSE
Just the average on training set	0.8692055
Just the average on test set	0.8889809
Just the median on training set	0.8871260
Just the median on test set	0.9106211
Linear regression on training set	0.7287282
Linear regression on test set	0.7478543
Linear regression after variable selection on the training set	0.7293082
Linear regression after variable selection on the test set	0.7473045
Decision tree on test set	0.8250874

Random forest as continuous predictor

Previously, we just say how the decision tree could be use to predict a continuous values, which were then rounded to discrete values, but the model did not perform as well as linear regression. As with the decision tree we see the three most important predictors are alcohol, volatile acidity and free sulfur dioxide.

What is most interesting about this model is the accuracy jumped up to 69% making it far and above the best predictor. Same for RMSE, it dropped to .64 making it the best compared to any of the other models. This is a really good way to finish. I would certainly recommend this model for predicting quality.

```
#random forest for quality prediction
set.seed(12)
wine_rf_qual <- randomForest(quality~., data = rw_comb_train, mtry = 6,
                        ntrees=1000, importance = TRUE)
y_hat_rf_qual <- round(predict(wine_rf_qual, newdata =  rw_comb_test))

importance(wine_rf_qual)
```

```
##                       %IncMSE IncNodePurity
## fixed.acidity         51.62287     207.44781
## volatile.acidity     100.18043     459.80513
## citric.acid           52.36021     255.67426
## residual.sugar        68.39234     275.36137
## chlorides             45.35050     274.00354
## free.sulfur.dioxide   72.36234     326.19241
## total.sulfur.dioxide  65.03719     274.07549
## density               38.35632     362.91350
## pH                    66.46407     252.71831
## sulphates             74.46980     275.02190
## alcohol              109.93319     804.53813
## class                 11.43198       6.94562
```

```
rf_acc <- mean(y_hat_rf_qual == rw_comb_test$quality)
rf_rmse <-   sqrt(mean((y_hat_rf_qual - rw_comb_test$quality)^2))
accuracy_results <- bind_rows(accuracy_results, data_frame(method = "Random forest on test set",
Accuracy = rf_acc))
rmse_results <- bind_rows(rmse_results, data_frame(method="Random forest on test set", RMSE = rf
_rmse))

accuracy_results %>% knitr::kable()
```

method	Accuracy
Linear regression on training set	0.5337695
Linear regression on test set	0.5284615
Linear regresssion on training set with +- 1 range	0.9549740
Linear regression on test set with +-1 range	0.9469231
Linear regresssion on training set after variable selection	0.5312680
Linear regression on test set after variable selection	0.5238462
Linear regresssion on training set after variable selection with +- 1 range	0.9532423
Linear regression on test set after variable selection with +-1 range	0.9469231
Decision tree on test set	0.5376923
Random forest on test set	0.6430769

```
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average on training set	0.8692055
Just the average on test set	0.8889809
Just the median on training set	0.8871260
Just the median on test set	0.9106211
Linear regression on training set	0.7287282
Linear regression on test set	0.7478543
Linear regression after variable selection on the training set	0.7293082
Linear regression after variable selection on the test set	0.7473045
Decision tree on test set	0.8250874
Random forest on test set	0.6838803

Conclusion

I chose this set data set, actually two sets, so I could run a continuous predictor as well as binary classifier. The conclusion is different models performed differently.

For the classification models, the support vector machine did the best followed closely by k nearest neighbors. My assumption is false negatives are more costly than false positives, so specificity is often more important that overall accuracy. This is the case when testing for a disease, which is why I focused on maximizing specificity by minimizing false negatives. Also, while I got a slightly better performance from the SVM, it really came down to a single false negative and 3 false positives in the test set of 1300 observations. A way to further evaluate these two models would be through cross validation. I could also alter the size of the test set. I partitioned 20% of the observations for testing, but I could try other values between 15% and 30%. I set the seed to 12 for the data partition, but other seeds could be used to see if the outputs are truly representative of the data or there is a random effect of a particular seed. However, the seed is not typically used as a tuning parameter. Cross validation should account for most of the random variation. While SVM and KNN yielded the best results, logistic regression, decision trees and random forest are also good models models for classification. In particular, for those latter 3, the threshold can adjusted. When looking at the ensemble, the overall best specificity was actually logistic regression, only 4 false negatives, however the cost in false positives and therefore overall accuracy starts starts to get high.

For the continuous prediction, it is a little bit harder to determine how well the model is performing. Using the mean and median as the predicted value for all observations is a good way to establish a base line. Models such as linear regression should yield a lower RMSE, which is what we got in this analysis. Decision trees and random forest are interesting models because they can be used for both continuous prediction and classification. Here we saw that the decision tree did not perform as well as linear regression, but random forest is by the far the best model. I also tried evaluated the accuracy rate of the prediction which is not common. However, there were only 7 possible outcomes, with the fast majority being the 3 outcomes in the middle. The accuracy results were consistent with the RMSE. Models with lower RMSE had better accuracy rates. It was amazing to see how well random forest could prediction specific outcomes.

References

1. UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/wine+quality (https://archive.ics.uci.edu/ml/datasets/wine+quality)
2. R-Studio Help and Tutorials