# winequalityx.R

2020-02-18

```
library(GGally)
```

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

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
library(gridExtra)
```

```
set.seed(10)
white.url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv"
white.raw <- read.csv(white.url, header = TRUE, sep = ";")
white <- white.raw
str(white)
```

```
## 'data.frame':    4898 obs. of  12 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 ...
##  $ residual.sugar      : num  20.7 1.6 6.9 8.5 8.5 6.9 7 20.7 1.6 1.5 ...
##  $ chlorides           : num  0.045 0.049 0.05 0.058 0.058 0.05 0.045 0.04
5 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 ...
##  $ quality             : int  6 6 6 6 6 6 6 6 6 6 ...
```

```
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
```

```
table(white$quality)
```

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

```
dim(white)
```

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

```
names(white)
```

```
##  [1] "fixed.acidity"        "volatile.acidity"     "citric.acid"
##  [4] "residual.sugar"       "chlorides"            "free.sulfur.dioxide"
##  [7] "total.sulfur.dioxide" "density"              "pH"
## [10] "sulphates"            "alcohol"              "quality"
```

```
str(white)
```

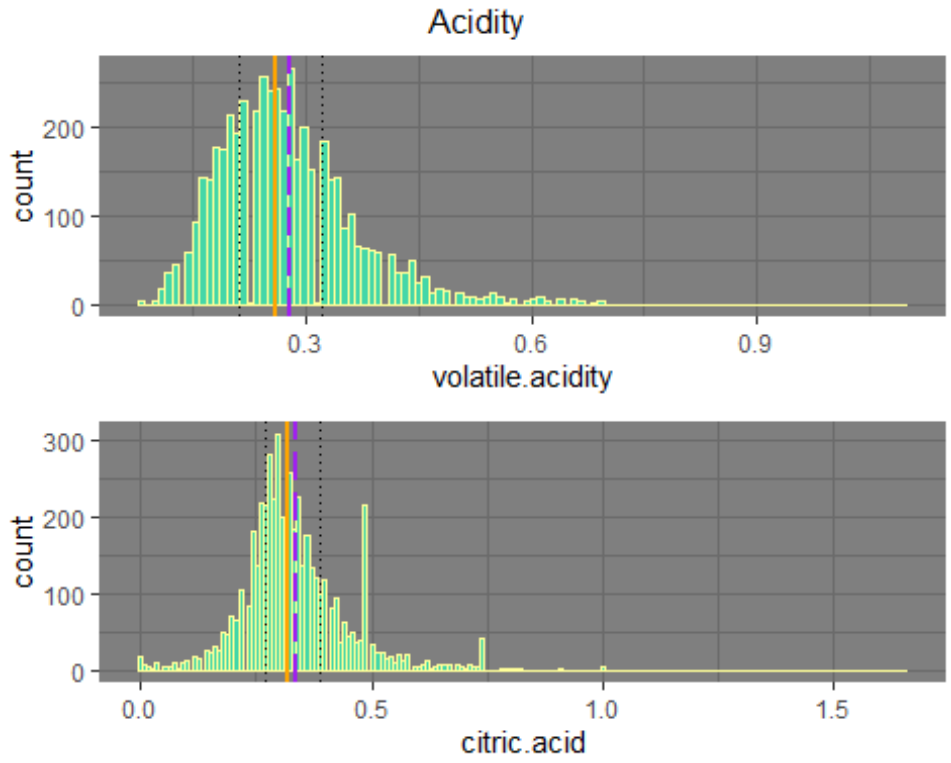
```
## 'data.frame':    4898 obs. of  12 variables:
##  $ fixed.acidity       : num  7 6.3 8.1 7.2 7.2 8.1 6.2 7 6.3 8.1 ...
##  $ volatile.acidity    : num  0.27 0.3 0.28 0.23 0.23 0.28 0.32 0.27 0.3 0
.22 ...
##  $ citric.acid         : num  0.36 0.34 0.4 0.32 0.32 0.4 0.16 0.36 0.34 0
.43 ...
##  $ residual.sugar      : num  20.7 1.6 6.9 8.5 8.5 6.9 7 20.7 1.6 1.5 ...
##  $ chlorides           : num  0.045 0.049 0.05 0.058 0.058 0.05 0.045 0.04
5 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 ...
##  $ quality             : int  6 6 6 6 6 6 6 6 6 6 ...
```

```r
summary(white)
```

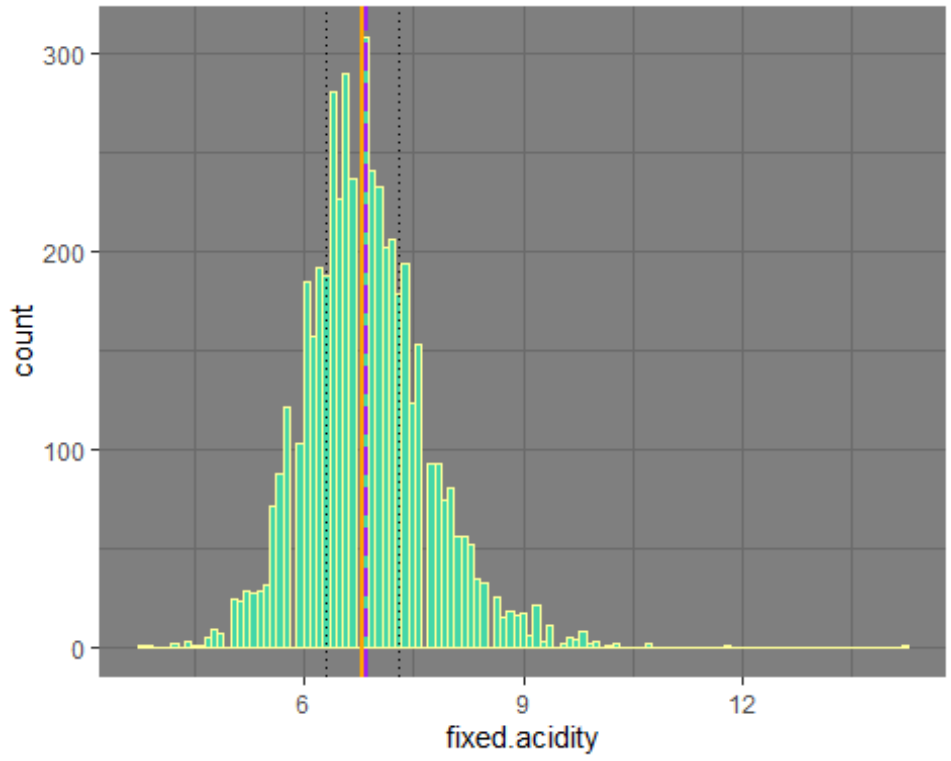
```
##  fixed.acidity    volatile.acidity  citric.acid     residual.sugar
##  Min.   : 3.800   Min.   :0.0800   Min.   :0.0000   Min.   : 0.600
##  1st Qu.: 6.300   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1.700
##  Median : 6.800   Median :0.2600   Median :0.3200   Median : 5.200
##  Mean   : 6.855   Mean   :0.2782   Mean   :0.3342   Mean   : 6.391
##  3rd Qu.: 7.300   3rd Qu.:0.3200   3rd Qu.:0.3900   3rd Qu.: 9.900
##  Max.   :14.200   Max.   :1.1000   Max.   :1.6600   Max.   :65.800
##    chlorides       free.sulfur.dioxide total.sulfur.dioxide    density
##  Min.   :0.00900   Min.   :  2.00      Min.   :  9.0        Min.   :0.9871
##  1st Qu.:0.03600   1st Qu.: 23.00      1st Qu.:108.0        1st Qu.:0.9917
##  Median :0.04300   Median : 34.00      Median :134.0        Median :0.9937
##  Mean   :0.04577   Mean   : 35.31      Mean   :138.4        Mean   :0.9940
##  3rd Qu.:0.05000   3rd Qu.: 46.00      3rd Qu.:167.0        3rd Qu.:0.9961
##  Max.   :0.34600   Max.   :289.00      Max.   :440.0        Max.   :1.0390
##        pH          sulphates         alcohol         quality
##  Min.   :2.720   Min.   :0.2200   Min.   : 8.00   Min.   :3.000
##  1st Qu.:3.090   1st Qu.:0.4100   1st Qu.: 9.50   1st Qu.:5.000
##  Median :3.180   Median :0.4700   Median :10.40   Median :6.000
##  Mean   :3.188   Mean   :0.4898   Mean   :10.51   Mean   :5.878
##  3rd Qu.:3.280   3rd Qu.:0.5500   3rd Qu.:11.40   3rd Qu.:6.000
##  Max.   :3.820   Max.   :1.0800   Max.   :14.20   Max.   :9.000
```

```r
# There is a big range for sulfur.dioxide (both Free and Total) across the sa
mples.
# The sample consists of  4898 White wine.
# The alcohol content varies from 8.00 to 14.20 for the samples in dataset.
# The quality of the samples range from 3 to 9 with 6 being the median.
# The range for fixed acidity is quite high with minimum being 3.8 and maximu
m being 14.2
# pH value varies from 2.720 to 3.820 with a median being 3.180
# Following function will be used for creating histogram matrix as well as
#individual histogram
# Plotting some variables together will make visualisation and explination ea
sier.
# Binwidths are selected by trails, the one which made histogram visually goo
d and
# made explanation easy.
p1 <- ggplot(data = white) + theme_dark()
his.matrix <- function(abc, bin) {
  binwidth <- bin
  ggplot(data = white, aes_string(x = abc )) +
    geom_histogram( fill ='#44D7A8', colour = '#FFFF99', binwidth = binwidth)
+
    geom_vline(aes(xintercept = mean(white[,abc] )) , color = 'purple',
               linetype = 'longdash', size = .75 ) +
    geom_vline(aes(xintercept = median(white[,abc] )), color = 'orange',
               size = .8) +
    geom_vline(aes(xintercept = quantile(white[,abc], 0.25 )),
```

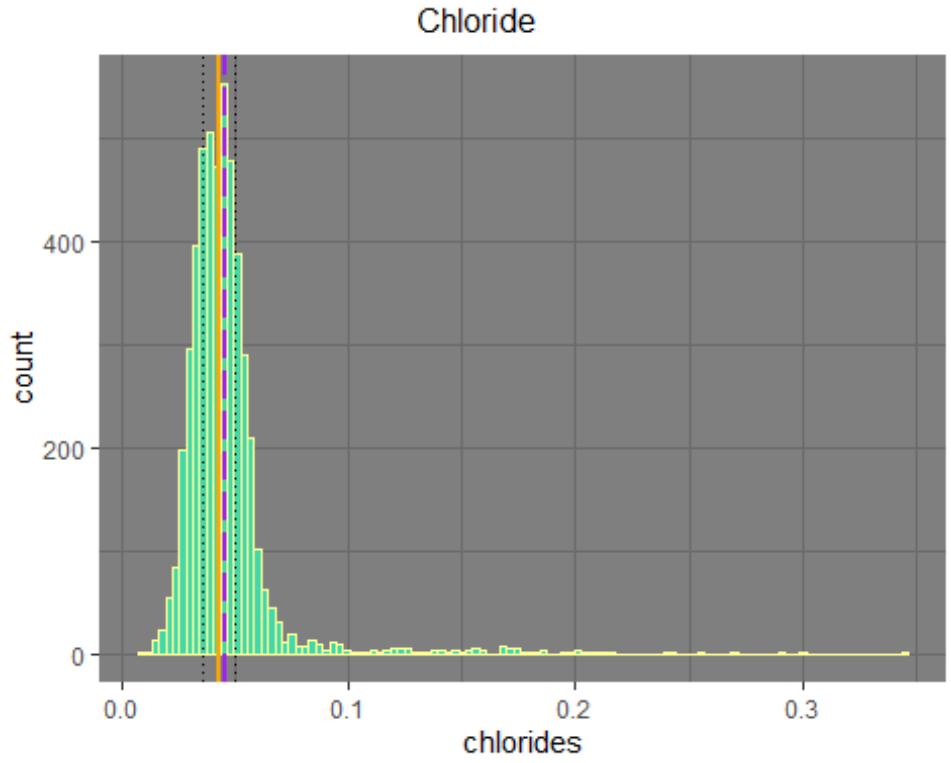
```
                linetype = 'dotted', size=0.5) +
    geom_vline( aes(xintercept=quantile(white[,abc], 0.75 )),
                linetype = 'dotted', size=0.5)+
    theme_dark()
  }

#grid of histograms of acid variables using his.matrix function and binwidth
# of .009
library(ggplot2)
variable_a <- c("volatile.acidity","citric.acid")
plots_a <- lapply(variable_a, his.matrix, bin=.009)
do.call(grid.arrange, args = c(plots_a, ncol = 1, top = 'Acidity'))
```



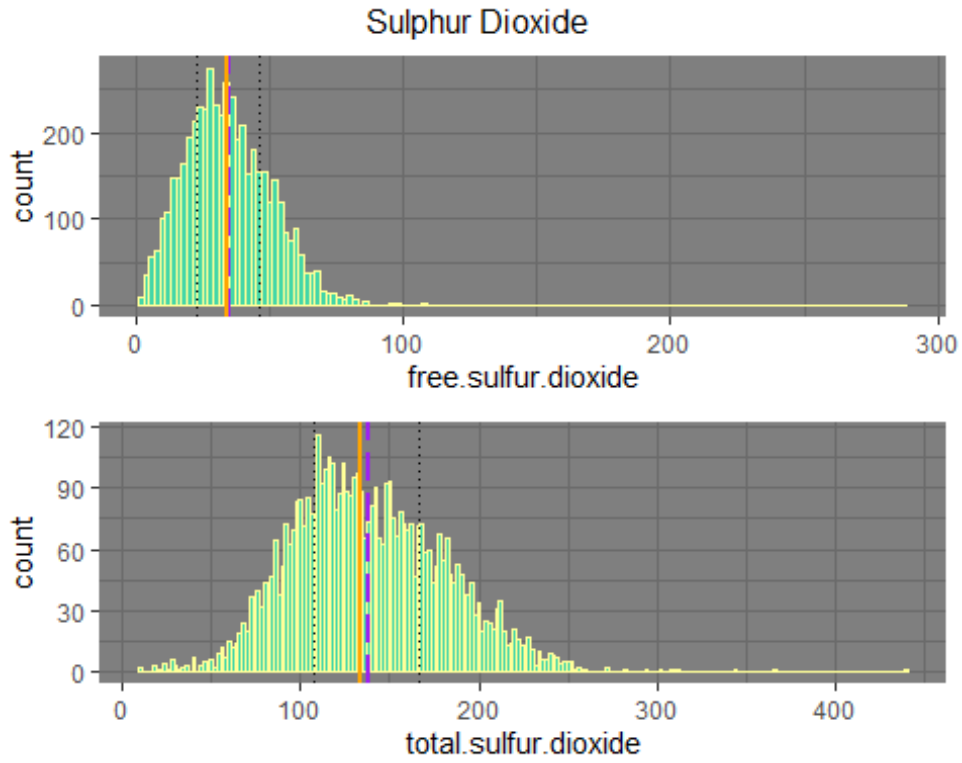

```
# *Purple-Dashed line indicates the mean, orange indicates the median,\
# while the dotted lines are for quartiles.*
#   *1 dm^3 = 1 Liter*
#   Both Citric and Acetic acid(indicated by volatile acid) are measured in g
m/dm^3
# and have an almost normal distribution. Citric acid have an unusual peak at
# 0.49 gm/dm^3, this may be because of persence of many wines from one paticu
lar
# winemaker or because of any regulations.
variable_f <- c("fixed.acidity")
plots_f <- lapply(variable_f, his.matrix, bin=.09)
do.call(grid.arrange, args = c(plots_f, ncol = 1))
```



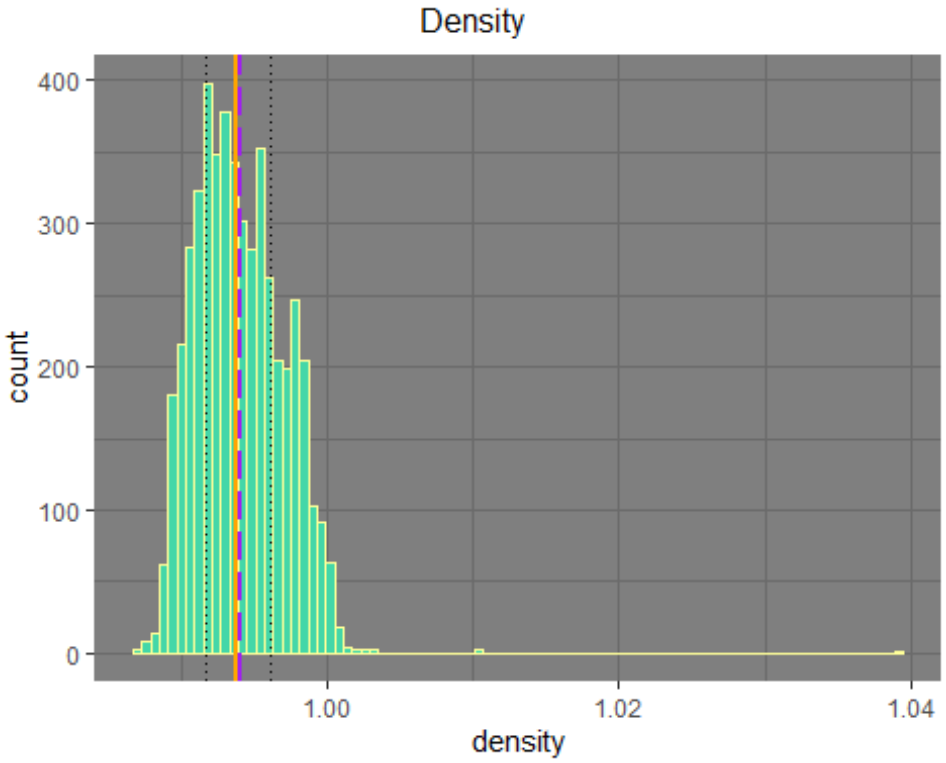

```
variable_d <- c("chlorides" )
plots_d <- lapply(variable_d, his.matrix, bin = .003)
do.call(grid.arrange, args = c(plots_d, ncol = 1, top = 'Chloride'))
```



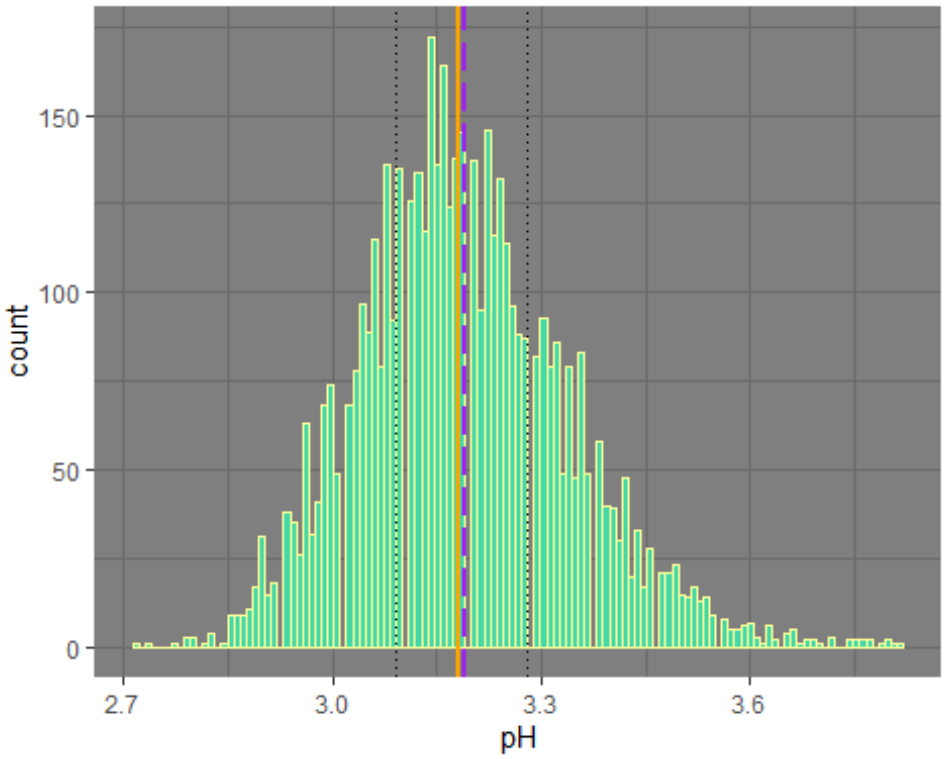
```
# The chloride data looks very lepokurtic as most data lies in 0.1 range.
# Infact 3rd Quartile at 0.05 indicates 75% data lies between 0.009 an 0.05
# (0.009 being min. value). Although data goes upto .34, very very few value
lies
# above .1.
#grid of sulphur variables
variable_b <- c("free.sulfur.dioxide","total.sulfur.dioxide" )
plots_b <- lapply(variable_b, his.matrix, bin = 2)
do.call(grid.arrange, args = c(plots_b, ncol = 1, top = 'Sulphur Dioxide'))
```



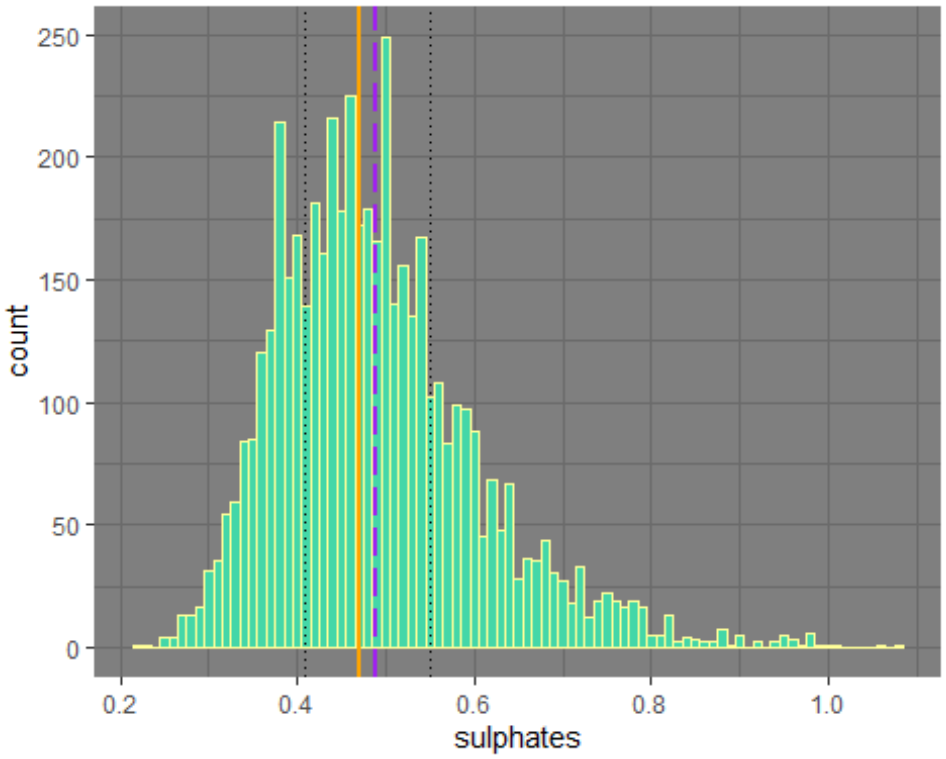

```
# Both Free and Total Sulphur dioxied have symmetric data and both are measur
ed in
# mg/dm^3

#grid of histograms of density variables using his.matrix function and binwid
th
# of .0006
variable_e <- c("density" )
plots_e <- lapply(variable_e, his.matrix, bin = .0006)
do.call(grid.arrange, args = c(plots_e, ncol = 1, top = 'Density'))
```



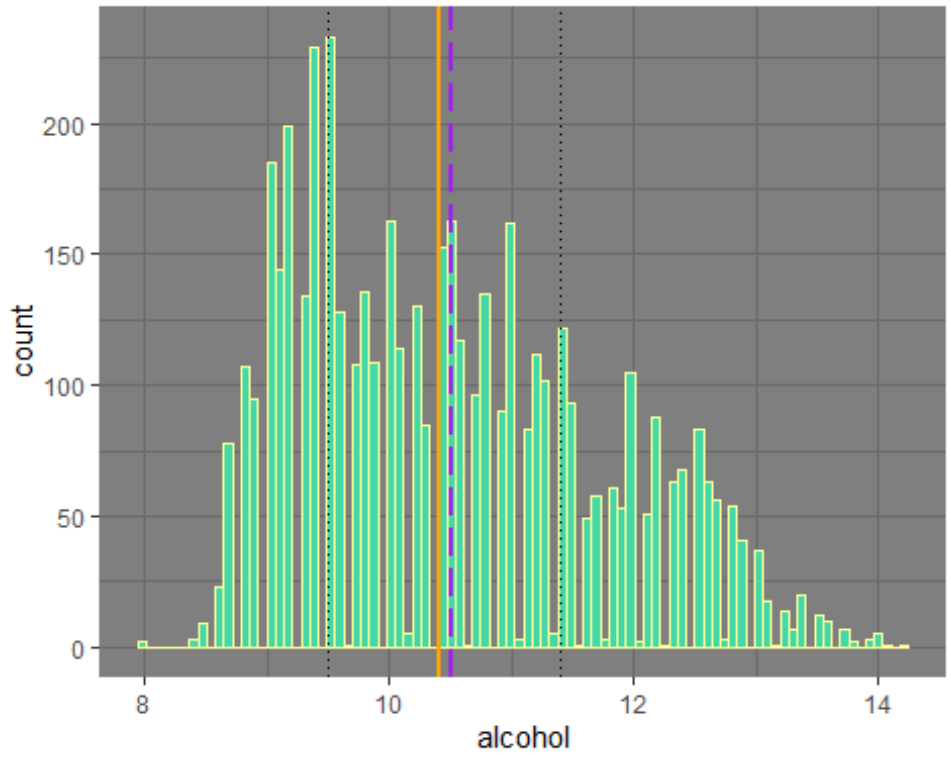

```
# The density data is present in extremely narrow range. As indicated from su
mmary
# and the visualisation above min value is .9871 and 3 Qt is at .9961. This i
s an
# very interesting observation because with almost 4900 observations this sma
ll range
# can help making a generalised statement like wines have density of .99 gm/c
m^3.
# This value can be used for doing mathematical calculations.
#grid of histograms of pH using his.matrix function and binwidth
# of .009
variable_h <- c("pH")
plots_h <- lapply(variable_h, his.matrix, bin = .009)
do.call(grid.arrange, args = c(plots_h, ncol = 1))
```



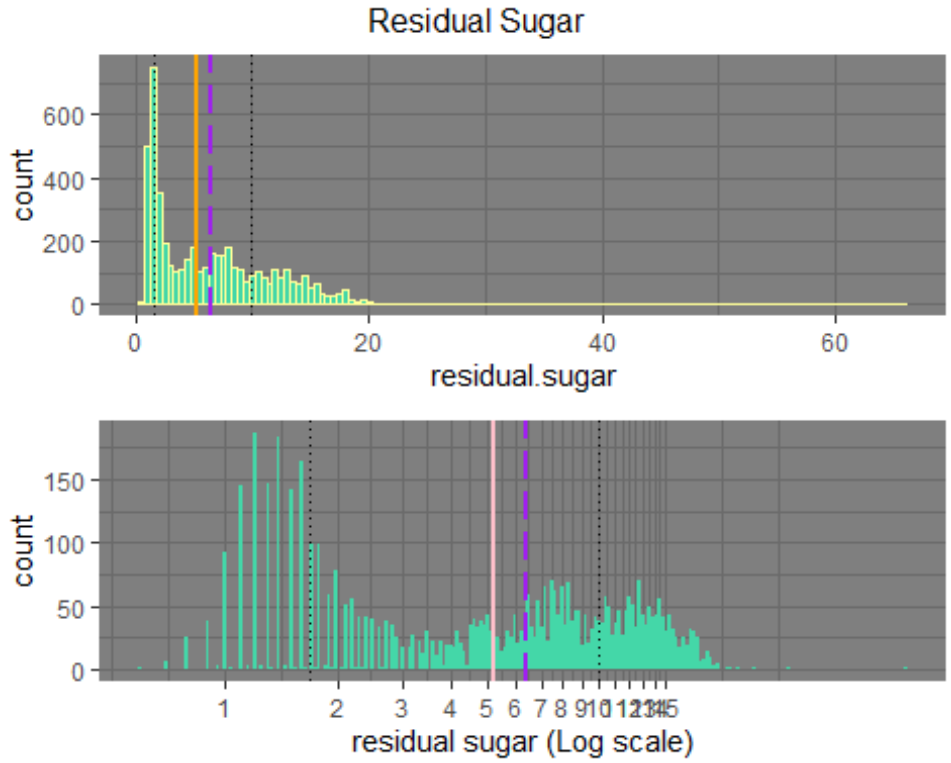

```
# pH range of wine in our datset 2.7 and 3.8, with average of 3.18.
# pH is basically a measure of acidity with 0 being most acidic , 7, neutral
and
# 14 being highly basic.

#grid of histograms of sulphate variables using his.matrix function and binwi
dth
# of .01
variable_c <- c("sulphates")
plots_c <- lapply(variable_c, his.matrix, bin = .01)
do.call(grid.arrange, args = c(plots_c, ncol = 1))
```



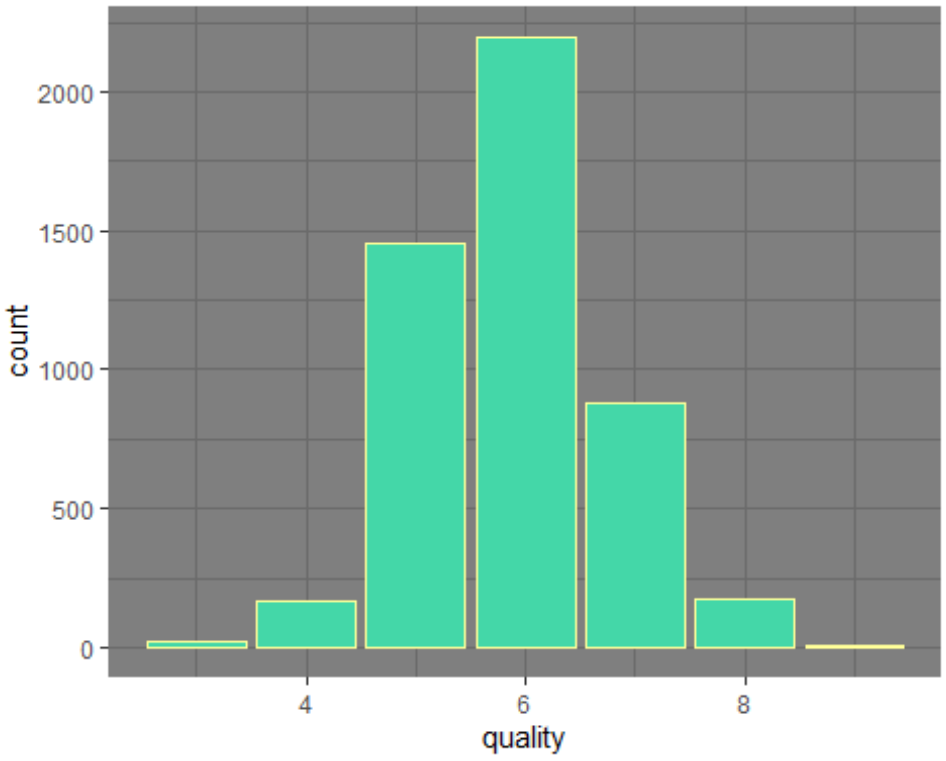

```
#grid of histograms of alcohol variables using his.matrix function and binwid
th
# of .07
variable_i <- c("alcohol")
plots_i <- lapply(variable_i, his.matrix, bin = .07)
do.call(grid.arrange, args = c(plots_i, ncol = 1))
```




```
# Alcohol measured in % volume dosen't show much symmetry and have values bet
ween
# 8 and 14 with 75% lying below 11.4. As visible we have most no. of wines
# with alcohol percent between 9.5 and 9.6.
p10 <- p1+ aes(x = residual.sugar)+
  geom_histogram(fill =I('#44D7A8'), binwidth = .009 )+
  scale_x_log10(breaks = seq(1,15,1))+
  geom_vline(aes(xintercept = median(white$residual.sugar )), color = 'pink',
             size = .8) +
  geom_vline(aes(xintercept = mean(white$residual.sugar)) , color = 'purple',
             linetype = 'longdash', size = .75 )+
  geom_vline(aes(xintercept = quantile(white$residual.sugar, 0.25 )),
             linetype = 'dotted', size=0.5) +
  geom_vline( aes(xintercept=quantile(white$residual.sugar, 0.75 )),
              linetype = 'dotted', size=0.5)+
  xlab('residual sugar (Log scale)')
variable_g <- c("residual.sugar" )
plots_g <- lapply(variable_g, his.matrix, bin = .5)
plots_g[[2]] <- p10
do.call(grid.arrange, args = c(plots_g, ncol = 1, top = "Residual Sugar"))
```




```
# Purple-Dashed line indicates the mean, orange indicates the median,
# while the dotted lines are for quartiles.
#
#
# Residual Sugar, or RS for short, refers to any natural grape sugars that
# is leftover after fermentation ceases. When plotted on log scale showed
# much better visualisation. Half of the wines have RS between 0.5 and
# 5 gm/liter while the other half lies between 5 and 20 gm/liter. The data se
ems to
# have a normal distribution betweem 0 and 5 gm/liter on log scale.
p10 <- p1+ aes(x = quality)+
  geom_bar(fill ='#44D7A8',colour = '#FFFF99')
p10
```




```
# A preliminary look at the dataset reveals that there are no missing values.
#
# First we can look at the correlation matrix of the dataset, to see if any p
redictors are highly correlated with one another. We may have to take out the
se predictors in order to avoid multicollinearity, which can invalidate resul
ts. Having said this, multicollinearity is less of an issue with decision tre
es, and even less so with randomForest, both of which are going to be used in
this analysis.

white.cor <- cor(white[,1:12])
white.cor

##                      fixed.acidity volatile.acidity  citric.acid residual.
sugar
## fixed.acidity           1.00000000      -0.02269729  0.289180698     0.089
02070
## volatile.acidity       -0.02269729       1.00000000 -0.149471811     0.064
28606
## citric.acid             0.28918070      -0.14947181  1.000000000     0.094
21162
## residual.sugar          0.08902070       0.06428606  0.094211624     1.000
00000
## chlorides               0.02308564       0.07051157  0.114364448     0.088
68454
## free.sulfur.dioxide    -0.04939586      -0.09701194  0.094077221     0.299
09835
## total.sulfur.dioxide    0.09106976       0.08926050  0.121130798     0.401
```

```
43931
## density                0.26533101       0.02711385  0.149502571     0.838
96645
## pH                    -0.42585829      -0.03191537 -0.163748211    -0.194
13345
## sulphates             -0.01714299      -0.03572815  0.062330940    -0.026
66437
## alcohol               -0.12088112       0.06771794 -0.075728730    -0.450
63122
## quality               -0.11366283      -0.19472297 -0.009209091    -0.097
57683
##                       chlorides free.sulfur.dioxide total.sulfur.dioxide
## fixed.acidity         0.02308564       -0.0493958591          0.091069756
## volatile.acidity      0.07051157       -0.0970119393          0.089260504
## citric.acid           0.11436445        0.0940772210          0.121130798
## residual.sugar        0.08868454        0.2990983537          0.401439311
## chlorides             1.00000000        0.1013923521          0.198910300
## free.sulfur.dioxide   0.10139235        1.0000000000          0.615500965
## total.sulfur.dioxide  0.19891030        0.6155009650          1.000000000
## density               0.25721132        0.2942104109          0.529881324
## pH                   -0.09043946       -0.0006177961          0.002320972
## sulphates             0.01676288        0.0592172458          0.134562367
## alcohol              -0.36018871       -0.2501039415         -0.448892102
## quality              -0.20993441        0.0081580671         -0.174737218
##                          density            pH   sulphates     alcohol
## fixed.acidity         0.26533101 -0.4258582910 -0.01714299 -0.12088112
## volatile.acidity      0.02711385 -0.0319153683 -0.03572815  0.06771794
## citric.acid           0.14950257 -0.1637482114  0.06233094 -0.07572873
## residual.sugar        0.83896645 -0.1941334540 -0.02666437 -0.45063122
## chlorides             0.25721132 -0.0904394560  0.01676288 -0.36018871
## free.sulfur.dioxide   0.29421041 -0.0006177961  0.05921725 -0.25010394
## total.sulfur.dioxide  0.52988132  0.0023209718  0.13456237 -0.44889210
## density               1.00000000 -0.0935914935  0.07449315 -0.78013762
## pH                   -0.09359149  1.0000000000  0.15595150  0.12143210
## sulphates             0.07449315  0.1559514973  1.00000000 -0.01743277
## alcohol              -0.78013762  0.1214320987 -0.01743277  1.00000000
## quality              -0.30712331  0.0994272457  0.05367788  0.43557472
##                           quality
## fixed.acidity        -0.113662831
## volatile.acidity     -0.194722969
## citric.acid          -0.009209091
## residual.sugar       -0.097576829
## chlorides            -0.209934411
## free.sulfur.dioxide   0.008158067
## total.sulfur.dioxide -0.174737218
## density              -0.307123313
## pH                    0.099427246
## sulphates             0.053677877
## alcohol               0.435574715
## quality               1.000000000
```

```r
# Taking a look at the correlation coefficients r for the predictor variables
, we see that density is strongly correlated with residual.sugar (r=0.84) and
alcohol (r=???0.78), and moderately correlated with total.sulfur.dioxide (r=0
.53). free.sulfur.dioxide and total.sulfur.dioxide are also moderately correl
ated with each other (r=0.62) although this is trivially known because of cou
rse, free sulfur dioxide is incorporated into the total sulfur dioxide.
#
# Aside from that correlations are all very low, including (and especially) q
uality, the response variable, with the predictors.
#
# So, we should actually remove the variables residual.sugar and density, as
well as total.sulfur.dioxide because if its direct relationship with free.sul
fur.dioxide, in order to address problems with multicollinearity. We're going
to withhold removing alcohol, to see the if the initial effect of removing ju
st these three correlated variables is enough to address the issue.
# removing three predictors
white2 <- subset(white, select = -c(4,7,8))
# scaling the 8 numeric attributes
white_sc <- white2
white_sc[,c(1:8)] <- scale(white_sc[,c(1:8)])
# From the new correlation matrix it appears that none of the predictors now
have too high or a correlation with each other, and we can decide that multic
ollinearity is no longer an issue.
#
# From here on out, we're also going to want to convert the quality response
variable into a binary factor so that we can use the predictors to classify t
he observations. We're going to do this by labeling all of the observations t
hat have received an above average (5 out of 10) as "good", and the rest as "
bad", "bad" really meaning "not good". This factor of good and bad goes under
a new column titled label.
#
# We'll remove the quality variable afterwards, since if we use it as an attr
ibute in the predictor, it will skew the results because it is directly corre
lated to the label.

# It's important to note that all of these numeric predictor variables (fixed
.acidity, volatile.acidity, citric.acid, chlorides, free.sulfur.dioxide, pH,
sulphates, alcohol) are not all scaled the same. As such, it's appropriate to
scale them before running any analyses.
# converting quality into a binary factor
for (i in 1:nrow(white_sc)) {
  if (white_sc$quality[i] > 5)
    white_sc$label[i] <- 1
  else
    white_sc$label[i] <- 0
}
white_sc$label <- factor(white_sc$label, levels = c(0, 1), labels = c("bad",
"good"))
# removing the quality variable
white_sc$quality <- NULL
```

```r
# Now we have 8 numeric predictor variables, and one two-level categorical va
riable (label). We're going to apply a few different classification methods i
n order to firstly determine which the best model for predicting is in terms
of the relevant variables, and secondly to find the best classification algor
ithm for this data.
#
# We're going to initialize a matrix to easily compare the quality of the dif
ferent classification methods we're going to utilize going forward, namely de
cision trees (with k-fold cross validation to prune the tree), k-nearest neig
hbor, and randomForest. The 'full randomForest' refers to the model using all
8 predictors whereas the 'small randomForest' refers to a subset of these pre
dictors, the use of which will become clear when discussing decision trees.
# initializing a matrix for records
records <- matrix(NA, nrow = 6, ncol = 3)
colnames(records) <- c("Accuracy Rate", "Error Rate", "AUC")
rownames(records) <- c("tree", "pruned.tree",
                       "k=10 kNN", "k=9 kNN",
                       "full.randomForest", "small.randomForest")
records
```

```
##                    Accuracy Rate Error Rate AUC
## tree                          NA         NA  NA
## pruned.tree                   NA         NA  NA
## k=10 kNN                      NA         NA  NA
## k=9 kNN                       NA         NA  NA
## full.randomForest             NA         NA  NA
## small.randomForest            NA         NA  NA
```

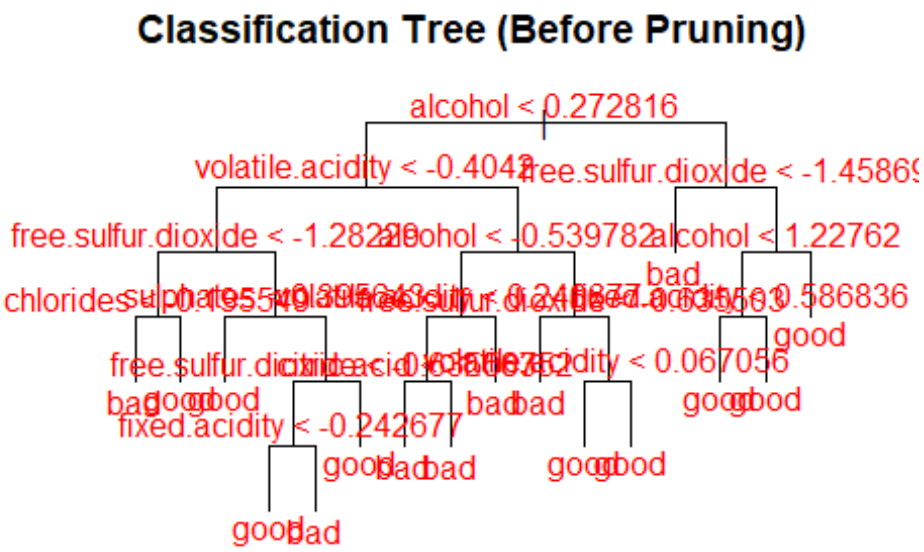
```r
# In order to apply machine learning algorithms to this dataset, we need to s
tratify the dataset into a training set and a test set. The first set will be
used to teach the classification model how to predict, depending on the algor
ithm chosen. We then apply the algorithm to the test set, and see how accurat
e the classification was.
# set.seed(10) is loaded
# using a subset of 1000 obs for the training set
test_indices <- sample(1:nrow(white_sc), 1000)
test <- white_sc[test_indices,]
train <- white_sc[-test_indices,]
# The first method we are going to perform on this dataset, is Decision Trees
. Decision tree is a non-parametric classification method, which uses a set o
f rules to predict that each observation belongs to the most commonly occurri
ng class label of training data.
#
# Of course, we're going to use label as a response variable, and each of the
now 8 remaining numeric attributes as predictors.
library(tree)
# library(tree) is loaded
# predicting the label (good vs bad)
tree <- tree(formula = label ~ ., data = train,
             method = "class",
```

```
             control = tree.control(nobs = nrow(train),
                                    mincut = 5,
                                    minsize = 10,
                                    mindev = .003))
summary(tree)
```

```
##
## Classification tree:
## tree(formula = label ~ ., data = train, control = tree.control(nobs = nrow
(train),
##     mincut = 5, minsize = 10, mindev = 0.003), method = "class")
## Variables actually used in tree construction:
## [1] "alcohol"             "volatile.acidity"    "free.sulfur.dioxide"
## [4] "chlorides"           "sulphates"           "fixed.acidity"
## [7] "citric.acid"
## Number of terminal nodes:  16
## Residual mean deviance:  0.9588 = 3722 / 3882
## Misclassification error rate: 0.2324 = 906 / 3898
```

```
# So we can see from this summary, that in fact 6 out of the 8 predictors wer
e used in constructing this tree: alcohol, volatile.acidity, free.sulfur.diox
ide, sulphates, citric.acid, and fixed.acidity. Now we are actually going to
plot the tree to visualize this.
```

```
plot(tree, type = "uniform")
text(tree, pretty = 0, cex = 1, col = "red")
title("Classification Tree (Before Pruning)")
```




```
# We can see while looking at the tree how often alcohol appears and intuit f
rom that that the amount of alcohol, whether high or low, plays at least some
part in the model's classification of a good wine.
#
# We can build a confusion matrix after using the data to predict on the test
set, and then find the accuracy rate and the error rate.
# a function that returns the accuracy of a confusion matrix
class_acc <- function(conf) {
  sum(diag(conf)) / sum(conf)
}

tree_pred <- predict(tree, test, type = "class")

# confusion matrix
tree_conf <- table(pred = tree_pred, true = test$label)
tree_conf

##       true
## pred   bad good
##   bad  225  118
##   good 133  524

# the class_acc() function is defined locally
tree_acc <- class_acc(tree_conf)
tree_acc

## [1] 0.749
```

```r
# With an accuracy rate of 0.749, this decision tree model is not superb, but
will still classify correctly about 3 out of 4 times.
#
# As an alternative metric to quantify the robustness of this method, we can
use the Receiver Operating Characteristic (ROC) curve and the area underneath
it (AUC). The ROC curve plots the false positive rate against the true positi
ve rate, and the area underneath it falls between either 0.5 or 1, 0.5 being
the worst (random classification), and 1 being the best (perfect classificati
on)
# misclassification error
tree_err <- 1 - tree_acc
tree_err
```

```
## [1] 0.251
```

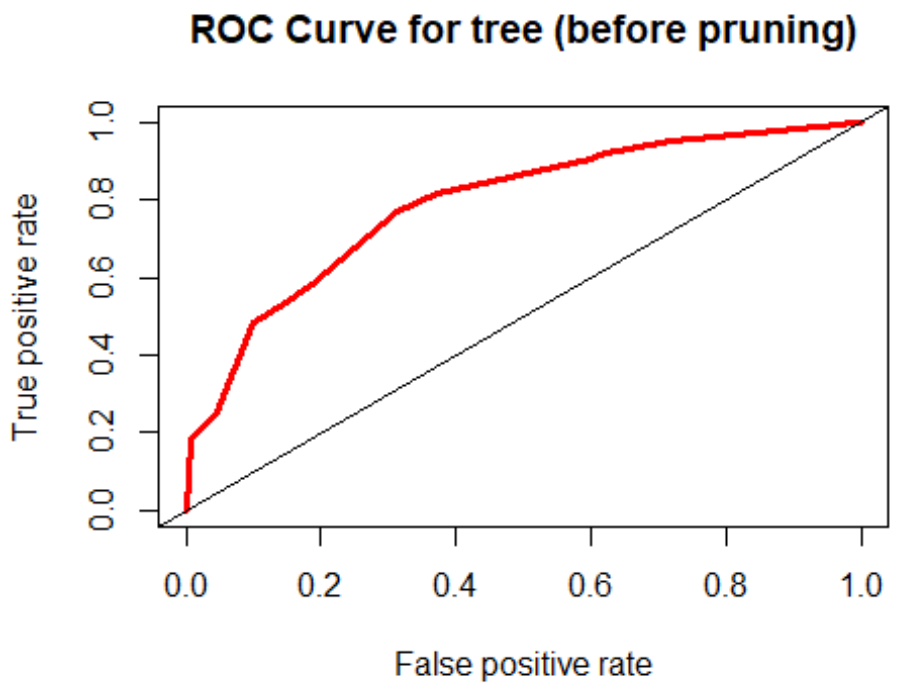
```r
library(ROCR)
```

```
## Loading required package: gplots
```

```
##
## Attaching package: 'gplots'
```

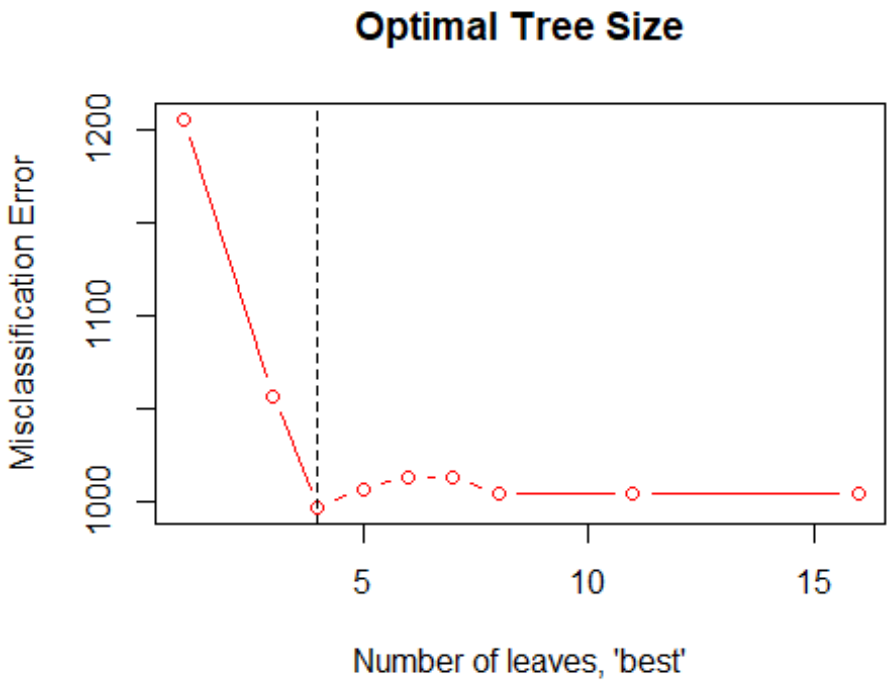
```
## The following object is masked from 'package:stats':
##
##     lowess
```

```r
# library(ROCR) is loaded
# getting matrix of predicted class probabilities
all_tree_probs <- as.data.frame(predict(tree, test, type = "vector"))
tree_probs <- all_tree_probs[,2]

tree_roc_pred <- prediction(tree_probs, test$label)
tree_roc_perf <- performance(tree_roc_pred, "tpr", "fpr")

# Plotting the ROC curve for the decision tree
plot(tree_roc_perf, col = 2, lwd = 3,
     main = "ROC Curve for tree (before pruning)")
abline(0,1)
```

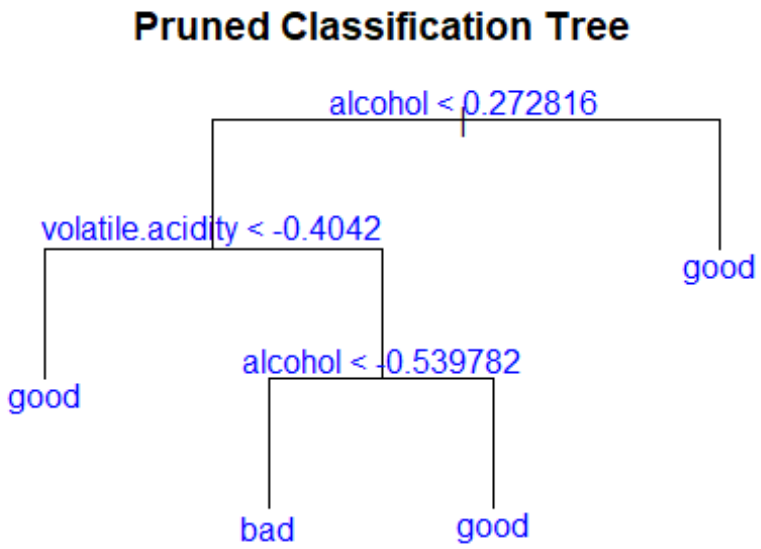



```
# Area under the curve
tree_auc_perf <- performance(tree_roc_pred, "auc")
tree_AUC <- tree_auc_perf@y.values[[1]]
tree_AUC

## [1] 0.7885253

# We see thus that the area under the curve is 0.788 which is slightly closer
to 1 than 0.5. That is to say that it is more good than bad, but hardly so.

# adding to records matrix
records[1, ] <- c(tree_acc, tree_err, tree_AUC)
records

##                    Accuracy Rate Error Rate       AUC
## tree                       0.749      0.251 0.7885253
## pruned.tree                   NA         NA        NA
## k=10 kNN                      NA         NA        NA
## k=9 kNN                       NA         NA        NA
## full.randomForest             NA         NA        NA
## small.randomForest            NA         NA        NA

set.seed(10)

# We can use k-fold cross-validation, which randomly partitions the dataset i
nto folds of similar size, to see if the tree requires any pruning which can
improve the model's accuracy as well as make it more interpretable for us.
```

```r
#
# In k-fold cross validation, we divide the sample into k sub samples, then t
rain the model on k -1 samples, leaving one as a holdout sample. We compute v
alidation error on each of these samples, then average the validation error o
f all of them.
#
# The idea of cross-validation is that it will sample multiple times from the
training set, with different separations. Ultimately, this creates a more rob
ust model i.e. the tree will not be overfit.
#
# Cross validation will help us find the optimal size for the tree (in terms
of number of nodes). We can plot the size against misclassification error to
visualize this as well.


# 10-fold CV (k = 10)
# library(tree) is loaded
cv <- cv.tree(tree, FUN=prune.misclass, K=10)
cv

## $size
## [1] 16 11  8  7  6  5  4  3  1
##
## $dev
## [1] 1004 1004 1004 1013 1013 1007  997 1057 1205
##
## $k
## [1]       -Inf   0.000000   1.333333   6.000000   7.000000  10.000000  40.
000000
## [8]  77.000000 116.000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"

# Best size
best.cv <- cv$size[which.min(cv$dev)]

# plotting misclass error as a function of tree size (k)
plot(cv$size , cv$dev, type="b",
     xlab = "Number of leaves, \'best\'",
     ylab = "Misclassification Error",
     col = "red", main="Optimal Tree Size")
abline(v=best.cv, lty=2)
```




```
best.cv

## [1] 4

# So we see, after running cross-validation, we see that we should prune the
tree so that it has only 7 nodes. With this knowledge we can prune the tree a
nd run the same diagnostics on it that we did on the unpruned model to see if
any improvements are apparent.
tree.pruned <- prune.tree(tree, best = best.cv,
                          method = "misclass")
summary(tree.pruned)

##
## Classification tree:
## snip.tree(tree = tree, nodes = c(10L, 3L, 4L, 11L))
## Variables actually used in tree construction:
## [1] "alcohol"          "volatile.acidity"
## Number of terminal nodes:  4
## Residual mean deviance:  1.054 = 4103 / 3894
## Misclassification error rate: 0.2496 = 973 / 3898

plot(tree.pruned, type = "uniform")
text(tree.pruned, col = "blue")
title("Pruned Classification Tree")
```




```
# Note that after pruning the tree, the only relevant variables used in tree
construction are: alcohol, volatile.acidity, and free.sulfur.dioxide. And of
course, this tree only has 7 nodes, the best tree size we determined from usi
ng cross-validation.
#
# Now we can apply the same diagnostic methods as before: looking at confusio
n matrix, accuracy/error rate, the ROC curve and the area underneath it, for
the sake of comparison.
pruned_pred <- predict(tree.pruned, test, type = "class")
# confusion matrix
pruned_conf <- table(pred = pruned_pred, true = test$label)
pruned_conf
```

```
##       true
## pred   bad good
##   bad  179   88
##   good 179  554
```

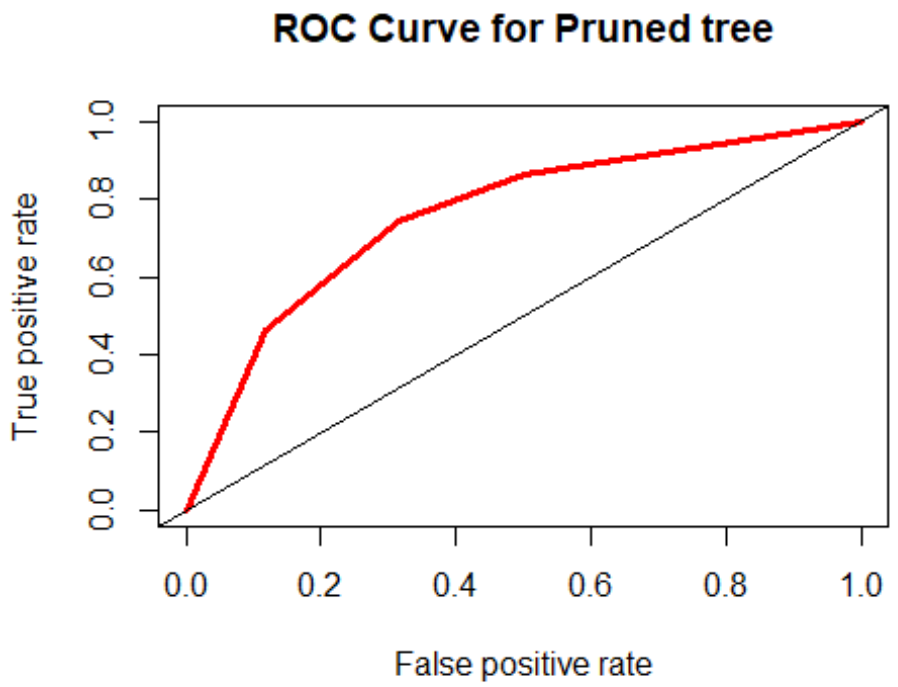
```
pruned_acc <- class_acc(tree_conf)
pruned_acc
```

```
## [1] 0.749
```

```
pruned_err <- 1 - tree_acc
pruned_err
```

```
## [1] 0.251
```

```r
# ROC Curve with tree object
# getting matrix of predicted class probabilities
all_pruned_probs <- as.data.frame(predict(tree.pruned, test, type = "vector")
)
pruned_probs <- all_pruned_probs[,2]

pruned_roc_pred <- prediction(pruned_probs, test$label)
pruned_roc_perf <- performance(pruned_roc_pred, "tpr", "fpr")

# Plotting the ROC curve for the rpart decision tree
plot(pruned_roc_perf, col = 2, lwd = 3,
     main = "ROC Curve for Pruned tree")
abline(0,1)
```



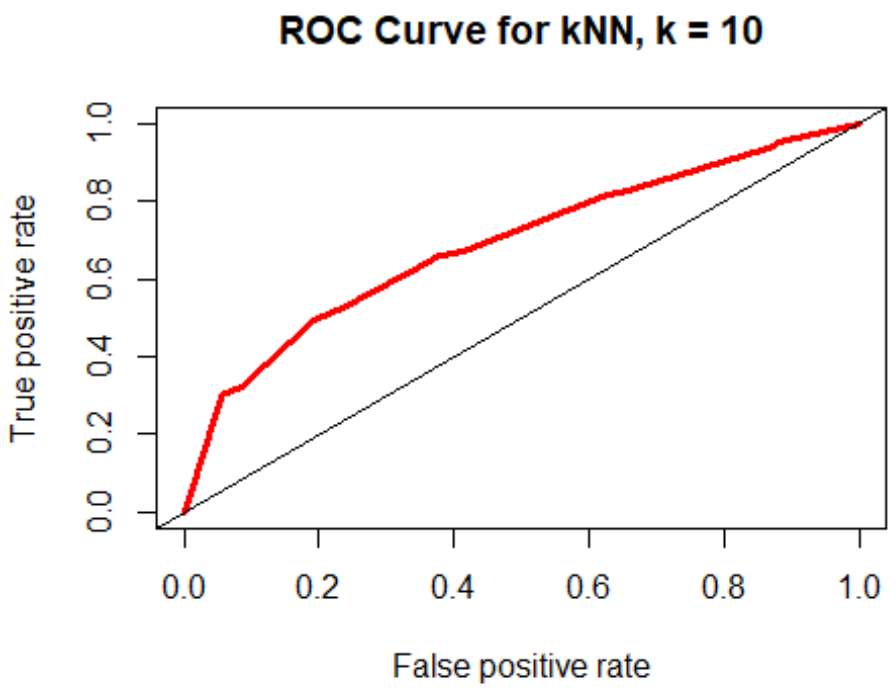

```r
pruned_auc_perf <- performance(pruned_roc_pred, "auc")
pruned_AUC <- pruned_auc_perf@y.values[[1]]
pruned_AUC

## [1] 0.7604509

records[2, ] <- c(pruned_acc, pruned_err, pruned_AUC)
records

##                     Accuracy Rate Error Rate       AUC
## tree                       0.749      0.251 0.7885253
## pruned.tree                0.749      0.251 0.7604509
## k=10 kNN                      NA         NA        NA
```

```
## k=9 kNN                          NA          NA          NA
## full.randomForest                NA          NA          NA
## small.randomForest               NA          NA          NA

# We see that pruning the tree didn't actually really improve the accuracy ra
te of the model at all, although it did condense the number of relevant varia
bles. Initially, seeing that accuracy did not improve might give the impressi
on that pruning was not meaningful, but to the contrary, the fact that we wer
e able to prune the tree without losing any accuracy shows that the sole 3 va
riables we have remaining (alcohol, volatile.acidity, and free.sulfur.dioxide
) are just as good as classifying when using a decision tree as when using al
l 8 predictors.
#
# The original model being rather complex with as many as 6 predictors runs t
he risk of over-fitting, which is to say that the data follows the training d
ata too closely and cannot be well generalized to new data. This is why we ar
e inclined to favor a simpler model such as that we found after pruning with
cross-validation.
# So while the accuracy and error rates are virtually unchanged, the area und
er the curve (AUC) has slightly decreased. It's not a substantial decrease, b
ut one could argue that it has overall made the model worse. Conversely it co
uld be argued that the strength of the model is relatively preserved while re
ducing the number of variables included. This is good because it gives us a b
etter idea of what the important variables are when it comes to classifying t
he wines.
#
# Now we have added both the original and the pruned tree's respective error
rates and AUC's to the records matrix, and we can proceed to the next method
of classification.
set.seed(10)
# We're now going to apply the k-nearest neighbors method of classification,
which is a non-parametric method. k-Nearest neighbors (or kNN) is called a "l
azy learning" technique because it goes through the training set every time i
t predicts a test sample's label. It finds this label by plotting the test sa
mple in the same dimensional space as the training data, then classifies it b
ased on the "k nearest neighbor(s)", i.e. if k = 10, then the label of the 10
nearest neighbors in the training data to the test data observation will be a
pplied to that observation.
#
# Distance is measured in different ways, but by default the knn() function u
tilized Euclidean distance.
#
# This is rather problematic because when calculating distance it's assumed t
hat attributes have the same effect, while this is not generally true. So the
distance metric (Euclidean distance in this case) does not take into account
the attributes' relationships with each other, which can result in misclassif
ication. So already we have determined a shortcoming in the kNN method before
we have even applied it. Although of course, we already dropped the predictor
s that were highly correlated with each other, and what's more we scaled the
```

```
remaining numeric predictors, which goes in a small way to addressing this.
library(caret)

## Loading required package: lattice

library(class)
# using 20 nearest neighbors
knn_pred <- knn(train = train[,-9],
                test = test[,-9],
                cl = train$label,
                k = 10, prob = TRUE)

# confusion matrix
knn_conf <- table(pred = knn_pred, true = test$label)
knn_conf

##       true
## pred   bad good
##   bad  205   91
##   good 153  551

# accuracy
knn_acc <- class_acc(knn_conf)
knn_acc

## [1] 0.756

# So, using 10 nearest neighbors was just a random estimate, and it ended up
with another mediocre accuracy rate (0.757) but we can look at the area under
the ROC curve (AUC) and look at the strength of the test relative to the meth
ods we have tried so far.


# misclassification error
knn_err <- 1 - knn_acc
knn_err

## [1] 0.244

library(dplyr)

##
## Attaching package: 'dplyr'

## The following object is masked from 'package:gridExtra':
##
##     combine

## The following object is masked from 'package:GGally':
##
##     nasa
```

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

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

```
# Creating the ROC curve for knn
# library(dplyr) is loaded
knn_prob <- attr(knn_pred, "prob")
knn_prob <- 2 * ifelse(knn_pred == "-1", 1-knn_prob, knn_prob) - 1
knn_roc_pred <- prediction(predictions = knn_prob, labels = test$label)
knn_roc_perf <- performance(knn_roc_pred,
                            measure = "tpr", x.measure = "fpr")

# Plotting the KNN ROC curve
plot(knn_roc_perf, col = 2, lwd = 3,
     main = "ROC Curve for kNN, k = 10")
abline(0,1)
```



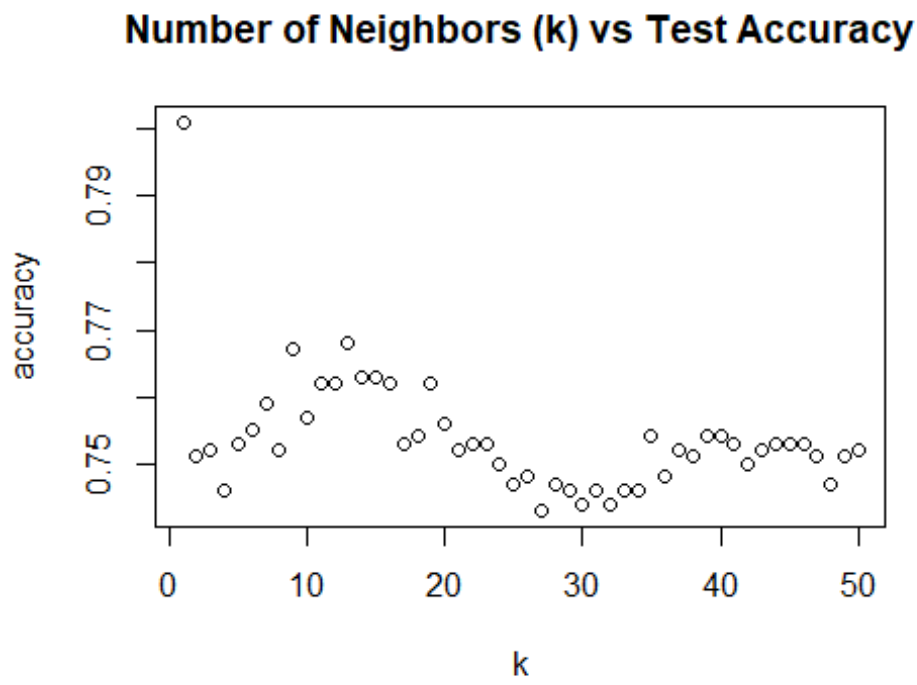

```
# Area under the knn curve
knn_auc_perf <- performance(knn_roc_pred, measure = "auc")
knn_AUC <- knn_auc_perf@y.values[[1]]
knn_AUC
```

```
## [1] 0.6893241
```

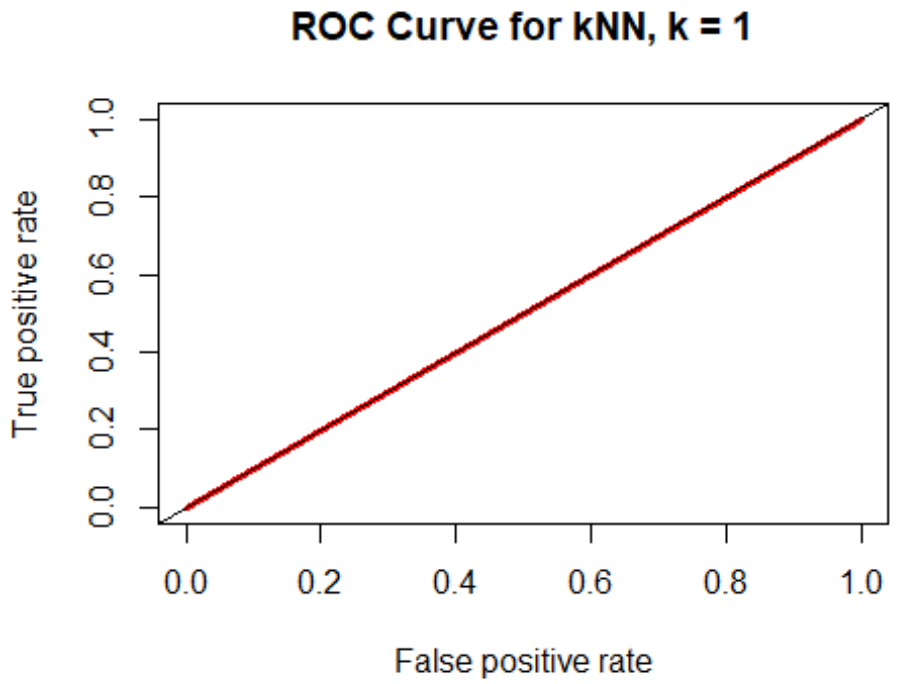
```r
# So with an AUC of 0.689, this test is not very good. We can look at differe
nt values for k and try to find the best one to use and then compare the resu
lts from that with these.
records[3, ] <- c(knn_acc, knn_err, knn_AUC)
records
```

```
##                     Accuracy Rate Error Rate       AUC
## tree                        0.749      0.251 0.7885253
## pruned.tree                 0.749      0.251 0.7604509
## k=10 kNN                    0.756      0.244 0.6893241
## k=9 kNN                        NA         NA        NA
## full.randomForest              NA         NA        NA
## small.randomForest             NA         NA        NA
```

```r
set.seed(10)
# library 'class' is loaded
range <- 1:50
knn_accs <- rep(0, length(range))

# Determining the best k for k-nearest neighbors classification
for (k in range) {
  knn_pred <- knn(train = train[,-9],
                  test = test[,-9],
                  cl = train$label,
                  k = k, prob = TRUE)
  knn_conf <- table(pred = knn_pred, true = test$label)
  knn_accs[k] <- class_acc(knn_conf)
}

# plotting k vs accuracy
plot(range, knn_accs, xlab = "k", ylab = "accuracy",
     main = "Number of Neighbors (k) vs Test Accuracy")
```




```
# Because accuracy seems to follow a slight negative trend but overall there
are huge jumps in accuracy when incrementing only by 1. We know well that usi
ng k=1 will result in a very low bias and high variance, and this also means
that we are fitting too closely to the training dataset and therefore, overfi
tting. This makes for a bad model that cannot be well generalized to new data
.

# Here is the ROC curve demonstrating this.
worst_knn_pred <- knn(train = train[,-9],
                      test = test[,-9],
                      cl = train$label,
                      k = 1,
                      prob = TRUE)
worst_knn_prob <- attr(worst_knn_pred, "prob")
worst_knn_prob <- 2*ifelse(worst_knn_pred == "-1", 1-worst_knn_prob, worst_kn
n_prob)-1
worst_knn_roc_pred <- prediction(predictions = worst_knn_prob,
                                 labels = test$label)
worst_knn_roc_perf <- performance(worst_knn_roc_pred, measure = "tpr", x.meas
ure = "fpr")

# Plotting the KNN ROC curve
plot(worst_knn_roc_perf, col = 2, lwd = 3,
     main = "ROC Curve for kNN, k = 1")
abline(0,1)
```




```
# library(knn) is loaded
# So we think better not to opt for k=1 and rather choose some k like 9, whic
h is still decently accurate, and probably less biased.
new_knn_pred <- knn(train = train[,-9],
                    test = test[,-9],
                    cl = train$label,
                    k = 9,
                    prob = TRUE)

# confusion matrix
new_knn_conf <- table(true = test$label, pred = new_knn_pred)
new_knn_conf
```

```
##       pred
## true   bad good
##   bad  210  148
##   good  83  559
```

```
# accuracy rate
new_knn_acc <- class_acc(new_knn_conf)
new_knn_acc
```

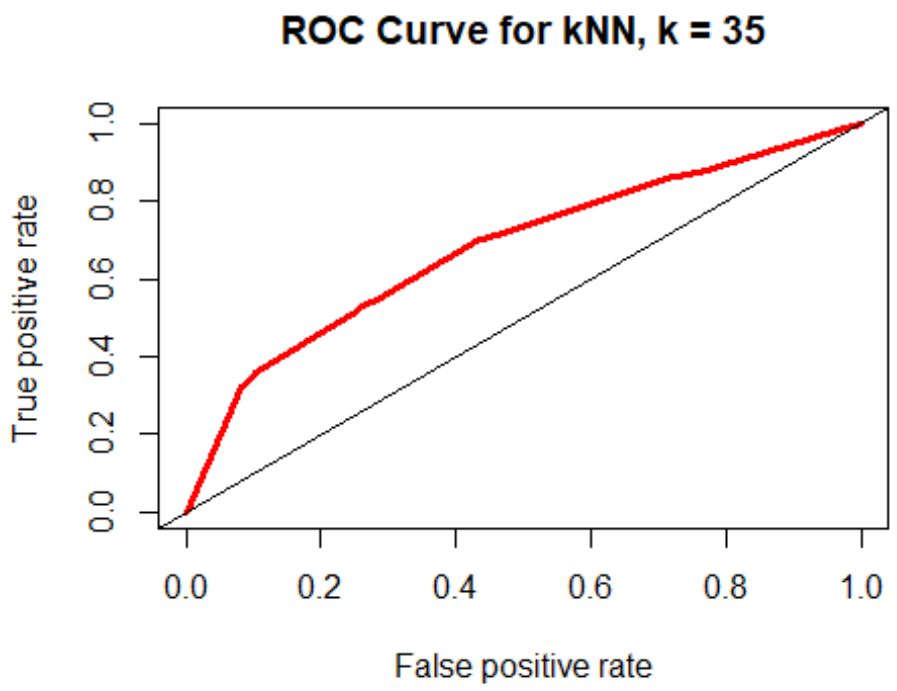
```
## [1] 0.769
```

```
# Using k=9 gives a slight increase in test accuracy relative to k=10, althou
gh it is not very significant. Now let's look at the ROC curve and the AUC to
make our final comparison, both with the k=10 model, and the decision trees.
```

```r
# misclassification error rate
new_knn_err <- 1 - new_knn_acc
new_knn_err
```

```
## [1] 0.231
```

```r
# Creating the ROC curve for knn
# library(dplyr) is loaded
new_knn_prob <- attr(new_knn_pred, "prob")
new_knn_prob <- 2*ifelse(new_knn_pred == "-1", 1-new_knn_prob, new_knn_prob)-
1
new_knn_roc_pred <- prediction(predictions = new_knn_prob,
                               labels = test$label)
new_knn_roc_perf <- performance(new_knn_roc_pred, measure = "tpr", x.measure
= "fpr")

# Plotting the KNN ROC curve
plot(new_knn_roc_perf, col = 2, lwd = 3,
     main = "ROC Curve for kNN, k = 35")
abline(0,1)
```




```r
# Area under the knn curve
new_knn_auc_perf <- performance(new_knn_roc_pred, measure = "auc")
new_knn_AUC <- new_knn_auc_perf@y.values[[1]]
new_knn_AUC
```

```
## [1] 0.679615
```

```
records[4,] <- c(new_knn_acc, new_knn_err, new_knn_AUC)
records
```

```
##                     Accuracy Rate Error Rate       AUC
## tree                        0.749      0.251 0.7885253
## pruned.tree                 0.749      0.251 0.7604509
## k=10 kNN                    0.756      0.244 0.6893241
## k=9 kNN                     0.769      0.231 0.6796150
## full.randomForest              NA         NA        NA
## small.randomForest             NA         NA        NA
```

```
### Random Forest
# RandomForest is similar to the decision tree method in that it builds trees
, hence the name 'random Forest'. This is an ensemble learning method which c
reates a multitude of decision trees, and outputting the class that occurs mo
st frequently among them. The advantage that randomForest has over decision t
rees is the element of randomness which guards against the pitfall of overfit
ting that decision trees run into on their own.
library(randomForest)
```

```
## randomForest 4.6-14
```

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

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

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:gridExtra':
##
##     combine
```

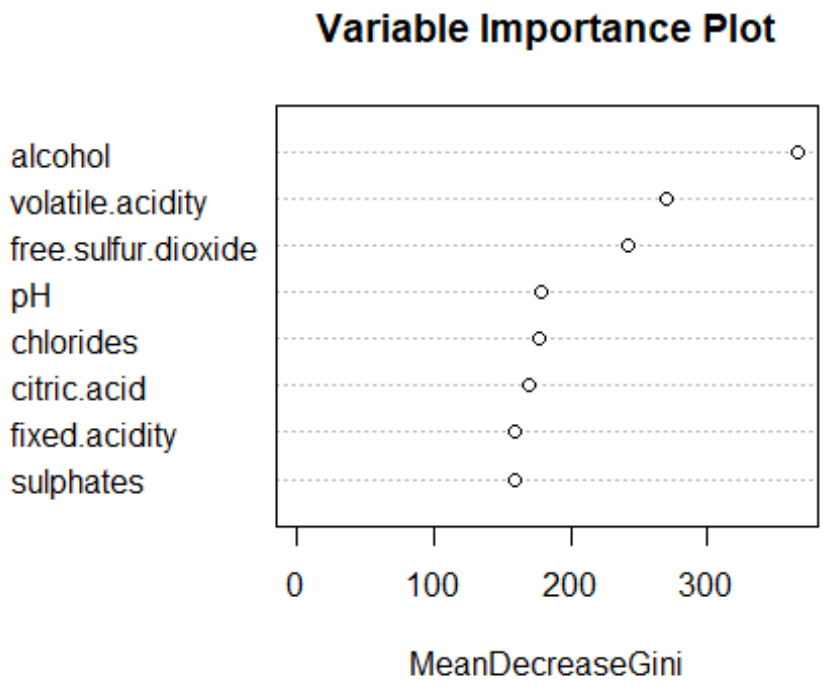
```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
# using all 8 predictor attributes, on the training set
rf <- randomForest(formula = label ~ .,
                   data = train,
                   mtry = 8)

print(rf)
```

```
##
## Call:
##  randomForest(formula = label ~ ., data = train, mtry = 8)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 8
```

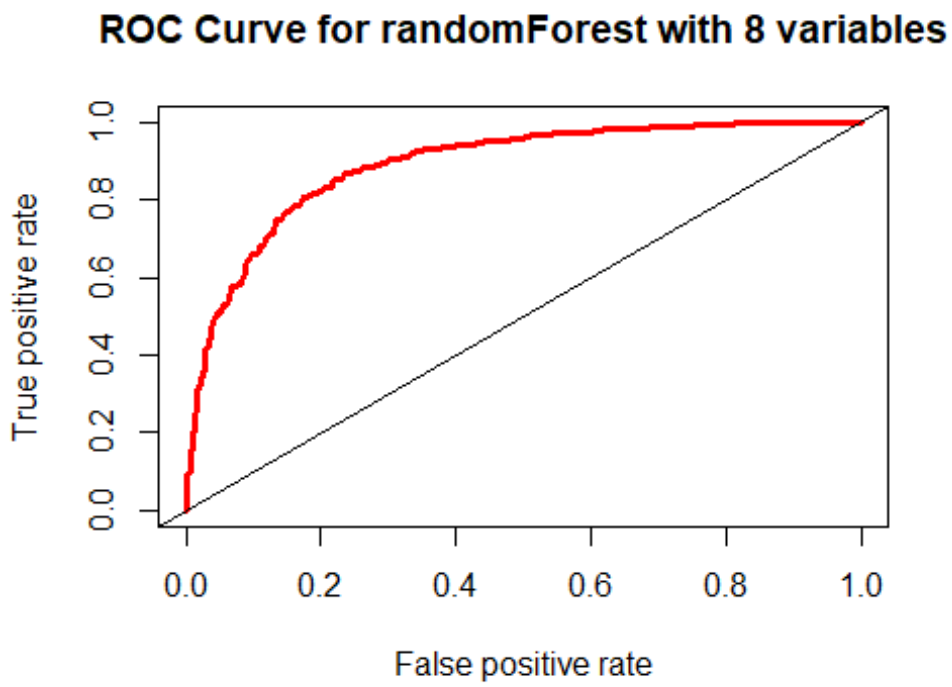
```
##
##         OOB estimate of  error rate: 16.6%
## Confusion matrix:
##      bad good class.error
## bad  896  386  0.30109204
## good 261 2355  0.09977064
```

```
varImpPlot(rf, main = "Variable Importance Plot")
```




```
# meanDecreaseGini refers to the "mean decrease in node impurity". Impurity i
s a way that the optimal condition of a tree is determined, and this plot sho
ws how each variable individually affects the weighted impurity of the tree i
tself.
#
# randomForest used all 8 of the predictor variables. This variable importanc
e plot shows how 'important' each variable was in determining the classificat
ion. We can see that, consistent with the pruned decision tree, that alcohol,
volatile.acidity, and free.sulfur.dioxide are the three most important predic
tors.
# predicting on the test set
rf_pred <- predict(rf, test, type = "class")

# Confusion Matrix
rf_conf <- table(true = test$label, pred =  rf_pred)
rf_conf
```

```
##       pred
## true   bad good
##   bad  250  108
##   good  63  579

rf_acc <- class_acc(rf_conf)
rf_acc

## [1] 0.829

# With an accuracy rate of 0.829, this randomForest model is looking pretty g
ood so far, and it already is more accurate than any method we've tried thus
far.
#
# Let's take a look at the ROC curve and the area underneath it.
rf_err <- 1 - rf_acc
rf_err

## [1] 0.171

# Building the ROC Curve
rf_pred <- as.data.frame(predict(rf, newdata = test, type = 'prob'))
rf_pred_probs <- rf_pred[,2]
rf_roc_pred <- prediction(rf_pred_probs, test$label)
rf_perf <- performance(rf_roc_pred, measure = "tpr",
                       x.measure = "fpr")

# Plotting the curve
plot(rf_perf, col = 2, lwd = 3,
     main = "ROC Curve for randomForest with 8 variables")
abline(0,1)
```




```
# Area under the curve
rf_perf2 <- performance(rf_roc_pred, measure = "auc")
rf_AUC <- rf_perf2@y.values[[1]]
rf_AUC

## [1] 0.8899367

# The area under the ROC curve for randomForest is 0.888, which is also a str
ong AUC for a classification model.
#
# So we see actually that randomForest stands head and shoulders above the ot
her two methods, decision tree and k-nearest neighbors. This is seen in the f
act that the accuracy rate, as well as the AUC, are the highest. Judging from
this, we can assume that randomForest would be the most likely to correctly c
lassify a wine based off of the attributes and data given.
#
# Recall that earlier we determined in the decision tree that the relevant va
riables were: alcohol, volatile.acidity, and free.sulfur.dioxide. While this
randomForest model was pretty effective in utilizing all of the 8 predictors,
we can take a look at a model using only these 3 as well for the sake of comp
arison.
#
# We have established by now that simpler models have a reduced bias and comp
lexity, but higher variance and a higher chance of underfitting, whereas comp
lex models (such as the full model) have the opposite issue. The good thing a
bout randomForest is that it inherently accounts for this "Bias-Variance" tra
deoff by introducing randomness with bagging (bootstrap aggregating).
```

```r
#
# The question here is whether or not making the model simpler is worthwhile,
but we can build the simple model and compare their metrics to find out.
records[5, ] <- c(rf_acc, rf_err, rf_AUC)
records
```

```
##                     Accuracy Rate Error Rate       AUC
## tree                       0.749      0.251 0.7885253
## pruned.tree                0.749      0.251 0.7604509
## k=10 kNN                   0.756      0.244 0.6893241
## k=9 kNN                    0.769      0.231 0.6796150
## full.randomForest          0.829      0.171 0.8899367
## small.randomForest            NA         NA        NA
```

```r
rf2 <- randomForest(formula = label ~ alcohol + volatile.acidity + free.sulfu
r.dioxide,
                    data = train,
                    mtry = 3)
# predicting on the test set
rf_pred2 <- predict(rf2, test, type = "class")

# Confusion Matrix
rf_conf2 <- table(test$label, rf_pred2)
rf_conf2
```

```
##       rf_pred2
##        bad good
##   bad  234  124
##   good  70  572
```

```r
rf_acc2 <- class_acc(rf_conf2)
rf_acc2
```

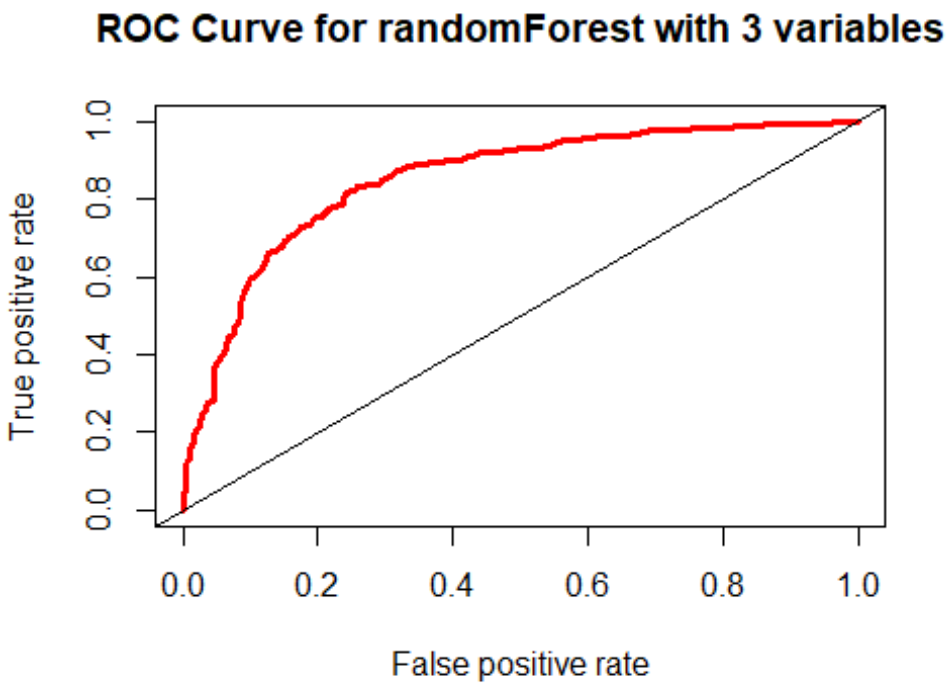
```
## [1] 0.806
```

```r
rf_err2 <- 1 - rf_acc2
rf_err2
```

```
## [1] 0.194
```

```r
# Building the ROC Curve
rf_pred2 <- as.data.frame(predict(rf2, test, type = 'prob'))
rf_pred_probs2 <- rf_pred2[,2]
rf_roc_pred2 <- prediction(rf_pred_probs2, test$label)
rf_perf2 <- performance(rf_roc_pred2,
                        measure = "tpr",
                        x.measure = "fpr")

# Plotting the curve
plot(rf_perf2, col = 2, lwd = 3,
```

```
     main = "ROC Curve for randomForest with 3 variables")
abline(0,1)
```




```
# Area under the curve
rf_perf22 <- performance(rf_roc_pred2, measure = "auc")
rf_AUC2 <- rf_perf22@y.values[[1]]
rf_AUC2
```

```
## [1] 0.8517639
```

```
records[6,] <- c(rf_acc2, rf_err2, rf_AUC2)
records
```

```
##                     Accuracy Rate Error Rate       AUC
## tree                        0.749      0.251 0.7885253
## pruned.tree                 0.749      0.251 0.7604509
## k=10 kNN                    0.756      0.244 0.6893241
## k=9 kNN                     0.769      0.231 0.6796150
## full.randomForest           0.829      0.171 0.8899367
## small.randomForest          0.806      0.194 0.8517639
```

```
# The accuracy rate has actually decreased, as well as the area under the cur
ve, but not significantly. We're managed to actually preserve the strength of
the model, both in relation to the tree and knn methods, but also relative to
the original application of randomForest with all of the predictors.
#
# As such, we can opt to utilize this much smaller model for classification i
nstead if we are concerned about complexity and bias. Having said that, becau
```

```
se of the randomization introduced in the randomForest, it is inherently more
robust so subsetting in this manner may even be unnecessary.
#
```