Wine Type and Quality Prediction With Machine Learning

HarvardX PH125 Data Science Capstone Project Report

Victor Ivamoto

January, 2020

Contents

Preface

This report is part of the capstone project of HarvardX's Data Science Professional Certificate[1] program.

The R Markdown code used to generate this report and the PDF version are available on GitHub.

The HTML version is available on RPubs.

[1] https://www.edx.org/professional-certificate/harvardx-data-science

Chapter 1

Introduction

HarvardX's Data Science Professional Certificate program covers several steps in a data science project, such as data wrangling, data exploration and visualization, probability and statistics, R language, Rmarkdown, and machine learning. This capstone project briefly applies each of these concepts in a real world case study.

The School of Information and Computer Science at the University of California Irvine (UCI) maintains a machine learning repository used by the machine learning community for analysis of algorithms.

This project uses the wine quality data set of white and red wines of vinho verde. The literal translation for vinho verde is "green wine", which means "young wine" with wine being released three to six months after the grapes are harvested.[1]

Vinho verde is a reference to the wine produced in the northern region of Portugal and follows the regulations set by the *Comissão de Viticultura da Região dos Vinhos Verdes* ("Wine Commission of the Vinho Verde Region").

1.1 Dataset description

Two datasets were created[2], using red and white wine samples.

The inputs are physicochemical information, such as PH values, and the output is based on sensory data which is the median of at least 3 evaluations made by wine experts. Each expert graded the wine quality between 0 (very bad) and 10 (excellent).

The dataset providers alleges that due to privacy and logistic issues, only physicochemical and sensory variables are available.

There's no information about how the dataset was created, such as the distribution of grape types, wine brands and whether the same experts graded all wines. This information is

[1]https://en.wikipedia.org/wiki/Vinho_Verde

[2]https://archive.ics.uci.edu/ml/datasets/Wine+Quality

important to identify possible bias that may distort the analysis results. For example, the physicochemical properties may vary among different wines of the same type, affecting the distribution in the dataset.

The classes are ordered and not balanced. There are munch more normal wines than excellent or poor ones.

Input variables based on physicochemical tests:

1. fixed acidity
2. volatile acidity
3. citric acid
4. residual sugar
5. chlorides
6. free sulfur dioxide
7. total sulfur dioxide
8. density
9. pH
10. sulphates
11. alcohol

Output variable based on sensory data:

12. quality (score between 0 and 10)

1.2 Key Steps

The dataset authors suggests the prediction of wine quality based on the properties. In this project we predict quality of red wines only, and join both datasets and predict the type of wine, red or white, using the same inputs.

For this simple task we apply several data science and machine learning techniques.

The Methods and Analysis section explains the process and techniques used, such as data cleaning, data exploration and visualization, any insights gained, and the modeling approach.

The Results section presents the modeling results and discusses the model performance.

A brief summary of the report, its limitations, and future work are in the Conclusion section.

Note: all charts and tables in this document were created in R Markdown. In many cases, the code is hidden to facilitate the reading and save space. If you want to see the code, download the R Markdown code on GitHub.

Chapter 2

Methods and Analysis

Predicting wine quality and identifying the wine type, red or white, are classification problems. This is because the desired outcomes are grouped in classes or categories. Quality has 10 classes, numbered from 0 to 9, and wine type has two classes, red and white. The methodology used in this document apply machine learning classification techniques.

2.1 Model Evaluation

The evaluation of machine learning algorithms consists in comparing the predicted value with the actual outcome. The confusion matrix displays the predicted and observed values in a table and provides additional statistics summary. There are other metrics that go beyond the scope of this document.

2.1.1 Confusion Matrix

The confusion matrix is a simple table with the cross tabulation of the predicted values with the actual observed values.

	Actual Positive	Actual Negative
Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)

All values are in absolute numbers of observations and predictions, so for example the *True Positive* is the number of predicted values that are exactly the same as the actual values.

The meaning of the values in the table are:

True Positive (TP): Predicted *positive* for an actual *positive* value.

True Negative (TN): Predicted *negative* for an actual *negative* value.

False Positive (FN) or Type 1 Error: Predicted *positive* for an actual *negative* value.

False Negative (FN) or Type 2 Error: Predicted *negative* for an actual *positive* value.

Some useful statistic metrics can be calculated from the confusion matrix.

Accuracy: the proportion of correct predictions for both *positive* and *negative* outcomes, i.e. the ability to correctly predict a *positive* and *negative*. High accuracy with a large difference in the number of positives and negatives becomes less meaningful, since the algorithm loses the ability to predict the less common class. In this case, other metrics complements the analysis.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Sensitivity: the proportion of *positive* values when they are actually *positive*, i.e. the ability to predict *positive* values.

$$Sensitivity = \frac{TP}{TP + FN}$$

Specificity: is the probability of a predicted *negative* value conditioned to a *negative* outcome.

$$Specificity = Pr(\hat{Y} = Negative | Y = Negative)$$

In other words, specificity is the proportion of *negative* values when they are actually *negative*, i.e. the ability to predict *negative* values.

$$Specificity = \frac{TN}{TN + FP}$$

Prevalence: how often the *positive* value appears in the sample. Low prevalence may lead to statistically incorrect conclusions.

$$Prevalence = \frac{TP + FN}{TP + FP + TN + FN}$$

Precision: is the probability of an actual *positive* occurs conditioned to a predicted *positive* result.

$$Precision = Pr(Y = Positive | \hat{Y} = Positive)$$

Precision can be written as the proportion of *positive* values that are actually *positive*.

$$Precision = \frac{TP}{TP + FP}$$

Recall: is the same as sensitivity and is the probability of a predicted *positive* value conditioned to an actual *positive* value.

$$Recall = Sensitivity = Pr(\hat{Y} = Positive | Y = Positive)$$

$$Recall = \frac{TP}{TP + FN}$$

2.1.2 F1 Score

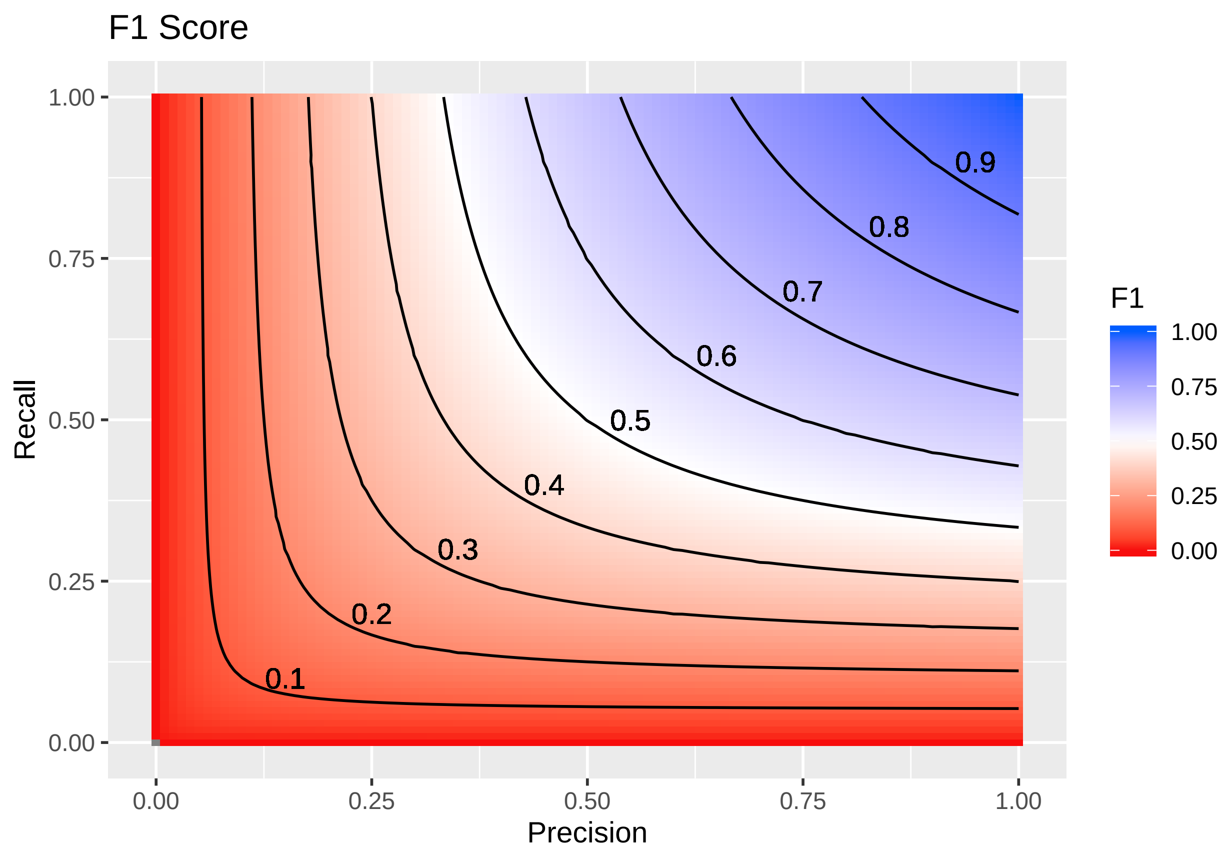
The F1 score is a measure of accuracy for classification problems with binary outcome. It is computed as the harmonic mean between precision and recall.

$$F1\ Score = \frac{1}{\frac{1}{2}\left(\frac{1}{recall} + \frac{1}{precision}\right)}$$

The formula can be simplified to:

$$F1\ Score = 2 \times \frac{precision \times recall}{precision + recall}$$

Since precision and recall vary from 0 to 1, we can make a plot of both values with the F1 score variation in color gradient. The black lines indicate the same F1 score values for different precision and recall combinations.



2.1.3 ROC and AUC

The Receiver Operator Characteristic (ROC) curve is the plot of True Positive Rate (TPR) and False Positive Rate (FPR) at different classification thresholds. It is commonly used in the evaluation of predictions of classification outcomes in machine learning.

$$TPR = Sensitivity = \frac{TP}{TP + FN}$$

$$FPR = 1 - Specificity = \frac{FP}{TN + FP}$$

The area under the ROC curve (AUC) measures the probability that a model will rank a positive value higher than a negative one.

AUC ranges from 0 to 1. A model with 100% wrong predictions has an AUC of 0, a model with 100% right predictions has AUC of 1 and a model with random prediction has an AUC of 0.5.

A rough guide for classifying the accuracy of a diagnostic test is the traditional academic point system[1]:

AUC	Classification
0.90 - 1	excellent (A)
0.80 - 0.90	good (B)
0.70 - 0.80	fair (C)
0.60 - 0.70	poor (D)
0.50 - 0.60	fail (F)

2.2 Data Preparation

In this section, we download and import the datasets in R. In the UCI repository there is one file for each wine type and one file with the dataset description, totaling three files.

The `type` column is added to each dataset to define the type of wine we want to predict with the `factor` variable type. This is variable used as the outcome that will be predicted.

Then, both datasets are combined in the `wine` dataset.

```r
# Set number of significant digits
options(digits = 3)

# The 'load_lib' function installs and loads
# a vector of libraries
load_lib <- function(libs) {
  sapply(libs, function(lib) {

    # Load the package. If it doesn't exists, install and load.
    if(!require(lib, character.only = TRUE)) {

      # Install the package
      install.packages(lib)

      # Load the package
      library(lib, character.only = TRUE)
      }
})}

# Load the libraries used in this section
```

[1] http://gim.unmc.edu/dxtests/roc3.htm

```r
libs <- c("tidyverse", "icesTAF", "readr",
          "lubridate", "caret")

load_lib(libs)

# Download the datasets from UCI repository
if(!dir.exists("data")) mkdir("data")
if(!file.exists("data/winequality-red.csv"))
  download.file("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/
if(!file.exists("data/winequality-white.csv"))
  download.file("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/
if(!file.exists("data/winequality.names"))
  download.file("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/

# Import the datasets.
# 'red' is the red wine dataset
# 'white' is the white wine dataset.
red   <- read_delim("data/winequality-red.csv",
                    delim = ";",
                    locale = locale(decimal_mark = ".",
                                    grouping_mark = ","),
                    col_names = TRUE)
white <- read_delim("data/winequality-white.csv",
                    delim = ";",
                    locale = locale(decimal_mark = ".",
                                    grouping_mark = ","),
                    col_names = TRUE)

# Set column names
cnames <- c("fixed_acidity", "volatile_acidity", "citric_acid",
            "residual_sugar", "chlorides", "free_sulfur_dioxide",
            "total_sulfur_dioxide", "density", "pH",
            "sulphates", "alcohol", "quality")

# Columns used for prediction are all columns
# except 'quality'.
xcol <- c("fixed_acidity", "volatile_acidity", "citric_acid",
          "residual_sugar", "chlorides", "free_sulfur_dioxide",
          "total_sulfur_dioxide", "density", "pH",
          "sulphates", "alcohol")

colnames(red)   <- cnames
colnames(white) <- cnames
```

```r
# Add the column 'type' to define the type of wine
red   <- mutate(red,   type = "red")
white <- mutate(white, type = "white")

# Join 'red' and 'white' datasets
wine <- rbind(red, white)
wine <- mutate(wine,
               quality = as.factor(quality),
               type = as.factor(type))
levels(wine$quality) <- paste0("Q", levels(wine$quality))
```

Now we split the dataset in two parts, one for training and one for testing. The model building and tuning is done in the training set, and then we use the test set to predict new values and evaluate the results. In this project the model won't be evaluated in another dataset.

We create a pair of training and testing sets for each prediction problem. `train_set` and `test_set` will be used for identifying wine type, and `train_set_red` and `test_set_red` will be used for red wine quality.

```r
# Test set will be 10% of the entire dataset
set.seed(2020, sample.kind = "Rounding")
test_index <- createDataPartition(y = wine$type,
                                  times = 1,
                                  p = 0.1,
                                  list = FALSE)

# Train and test sets for wine type
train_set <- wine[-test_index,]
test_set  <- wine[test_index,]

# Train and test sets for red wine quality
train_set_red <- train_set[which(train_set$type == "red"),]
test_set_red  <- test_set[which(test_set$type == "red"),]

train_set_red$quality <- factor(train_set_red$quality)
test_set_red$quality  <- factor(test_set_red$quality)
```

2.3 Data Exploration and Visualization

This section checks the dataset structure, look for possible problems in the data and provides a clear understanding of the data. The information acquired in this section may be used during the modeling phase.

Although UCI provides the dataset ready for analysis, it's good practice to check it for possible issues we may encounter in the future.

Let's start counting the number of empty values, NAs, in the entire dataset, i.e. in both the training and testing sets. Empty values may cause calculation errors that can be avoided if we identify them in advance.

```
sum(is.na(wine))
```

```
## [1] 0
```

Next, we check the predictors that have very few unique values relative to the number of samples or a large difference in the frequency of the most common and the second most common values. Predictors with these characteristics provide little value in the analysis and may be discarded. No variable has near zero variance.

```
# Identification of near zero variance predictors
nearZeroVar(train_set[, xcol], saveMetrics = TRUE)
```

	freqRatio	percentUnique	zeroVar	nzv
fixed_acidity	1.11	1.78	FALSE	FALSE
volatile_acidity	1.06	3.15	FALSE	FALSE
citric_acid	1.11	1.52	FALSE	FALSE
residual_sugar	1.07	5.29	FALSE	FALSE
chlorides	1.05	3.57	FALSE	FALSE
free_sulfur_dioxide	1.03	2.24	FALSE	FALSE
total_sulfur_dioxide	1.16	4.72	FALSE	FALSE
density	1.02	16.64	FALSE	FALSE
pH	1.06	1.85	FALSE	FALSE
sulphates	1.13	1.88	FALSE	FALSE
alcohol	1.10	1.81	FALSE	FALSE

The dataset structure confirms the information provided by UCI and provides additional information. All variables are numeric except `type` which we added as a factor with two levels. There are 5847 observations.

```
# Compactly Display the Structure of an Arbitrary R Object
str(train_set)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    5847 obs. of  13 variables:
##  $ fixed_acidity       : num  7.4 7.8 7.4 7.9 7.3 7.8 7.5 6.7 7.5 5.6 ...
##  $ volatile_acidity    : num  0.7 0.76 0.66 0.6 0.65 0.58 0.5 0.58 0.5 0.615 ...
##  $ citric_acid         : num  0 0.04 0 0.06 0 0.02 0.36 0.08 0.36 0 ...
##  $ residual_sugar      : num  1.9 2.3 1.8 1.6 1.2 2 6.1 1.8 6.1 1.6 ...
##  $ chlorides           : num  0.076 0.092 0.075 0.069 0.065 0.073 0.071 0.097 0.071 0
##  $ free_sulfur_dioxide : num  11 15 13 15 15 9 17 15 17 16 ...
##  $ total_sulfur_dioxide: num  34 54 40 59 21 18 102 65 102 59 ...
##  $ density             : num  0.998 0.997 0.998 0.996 0.995 ...
##  $ pH                  : num  3.51 3.26 3.51 3.3 3.39 3.36 3.35 3.28 3.35 3.58 ...
##  $ sulphates           : num  0.56 0.65 0.56 0.46 0.47 0.57 0.8 0.54 0.8 0.52 ...
##  $ alcohol             : num  9.4 9.8 9.4 9.4 10 9.5 10.5 9.2 10.5 9.9 ...
##  $ quality             : Factor w/ 7 levels "Q3","Q4","Q5",..: 3 3 3 3 5 5 3 3 3 3 ..
##  $ type                : Factor w/ 2 levels "red","white": 1 1 1 1 1 1 1 1 1 1 ...
```

The statistics summary provides a clue of each variable distribution. Median close to the mean may be an indication of normal distribution. The variables `free_sulfur_dioxide`, `total_sulfur_dioxide`, `density`, `pH`, `sulphates` and `alcohol` seems to follow this rule that we will confirm later.

The relative numbers of `red` and `white` in the `type` variable give an idea of prevalence.

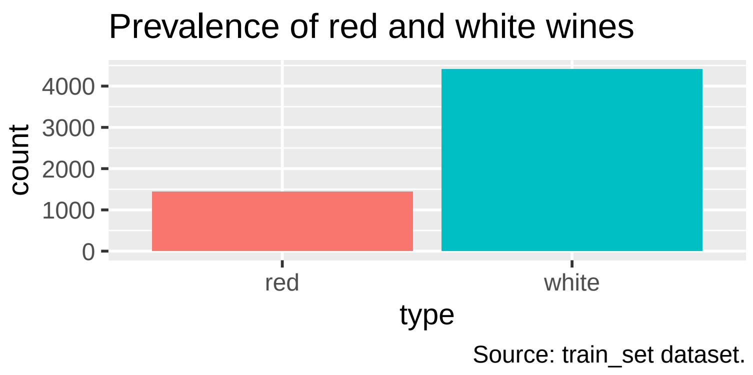
```r
# Statistics summary
summary(train_set)
```

```
##  fixed_acidity   volatile_acidity  citric_acid     residual_sugar
##  Min.   : 3.80   Min.   :0.08     Min.   :0.000   Min.   : 0.6
##  1st Qu.: 6.40   1st Qu.:0.23     1st Qu.:0.250   1st Qu.: 1.8
##  Median : 7.00   Median :0.29     Median :0.310   Median : 3.0
##  Mean   : 7.21   Mean   :0.34     Mean   :0.319   Mean   : 5.5
##  3rd Qu.: 7.70   3rd Qu.:0.40     3rd Qu.:0.390   3rd Qu.: 8.1
##  Max.   :15.90   Max.   :1.58     Max.   :1.660   Max.   :65.8
##
##    chlorides     free_sulfur_dioxide total_sulfur_dioxide    density
##  Min.   :0.009   Min.   :  1.0       Min.   :  6          Min.   :0.987
##  1st Qu.:0.038   1st Qu.: 17.0       1st Qu.: 78          1st Qu.:0.992
##  Median :0.047   Median : 29.0       Median :118          Median :0.995
##  Mean   :0.056   Mean   : 30.5       Mean   :116          Mean   :0.995
##  3rd Qu.:0.065   3rd Qu.: 41.0       3rd Qu.:156          3rd Qu.:0.997
##  Max.   :0.611   Max.   :289.0       Max.   :440          Max.   :1.039
##
##        pH          sulphates         alcohol      quality      type
##  Min.   :2.72   Min.   :0.220   Min.   : 8.0   Q3:  28   red  :1439
##  1st Qu.:3.11   1st Qu.:0.430   1st Qu.: 9.5   Q4: 189   white:4408
##  Median :3.21   Median :0.510   Median :10.3   Q5:1932
##  Mean   :3.22   Mean   :0.531   Mean   :10.5   Q6:2564
##  3rd Qu.:3.32   3rd Qu.:0.600   3rd Qu.:11.3   Q7: 968
##  Max.   :4.01   Max.   :2.000   Max.   :14.9   Q8: 163
##                                                Q9:   3
```

2.3.1 Red and White Prevalence

The goal is to identify red and white wines, so we check the distribution of both types. The summary above provides the figures; now let's check the relative amount.

```r
# Distribution of red and white wines
ggplot(data = train_set) +
  geom_bar(aes(type, fill = type)) +
  labs(title = "Prevalence of red and white wines",
       caption = "Source: train_set dataset.") +
  theme(legend.position = 'none')
```



The prevalence of red wine in this dataset is 24.6% and of white wine is 75.4%.

Let's compare this prevalence against the total production of red and white wines. This dataset was created in 2009[2] and there's no information about the date of each sample, so I'm assuming the most recent information is from 2008.

The commission of vinho verde producers provides annual statistics of production[3] for each wine type.

We import the statistics and calculate the prevalence of red wine production.

The prevalence of the production of red wine is higher than observed in the dataset. There is no information available about production by brand or grape variety.

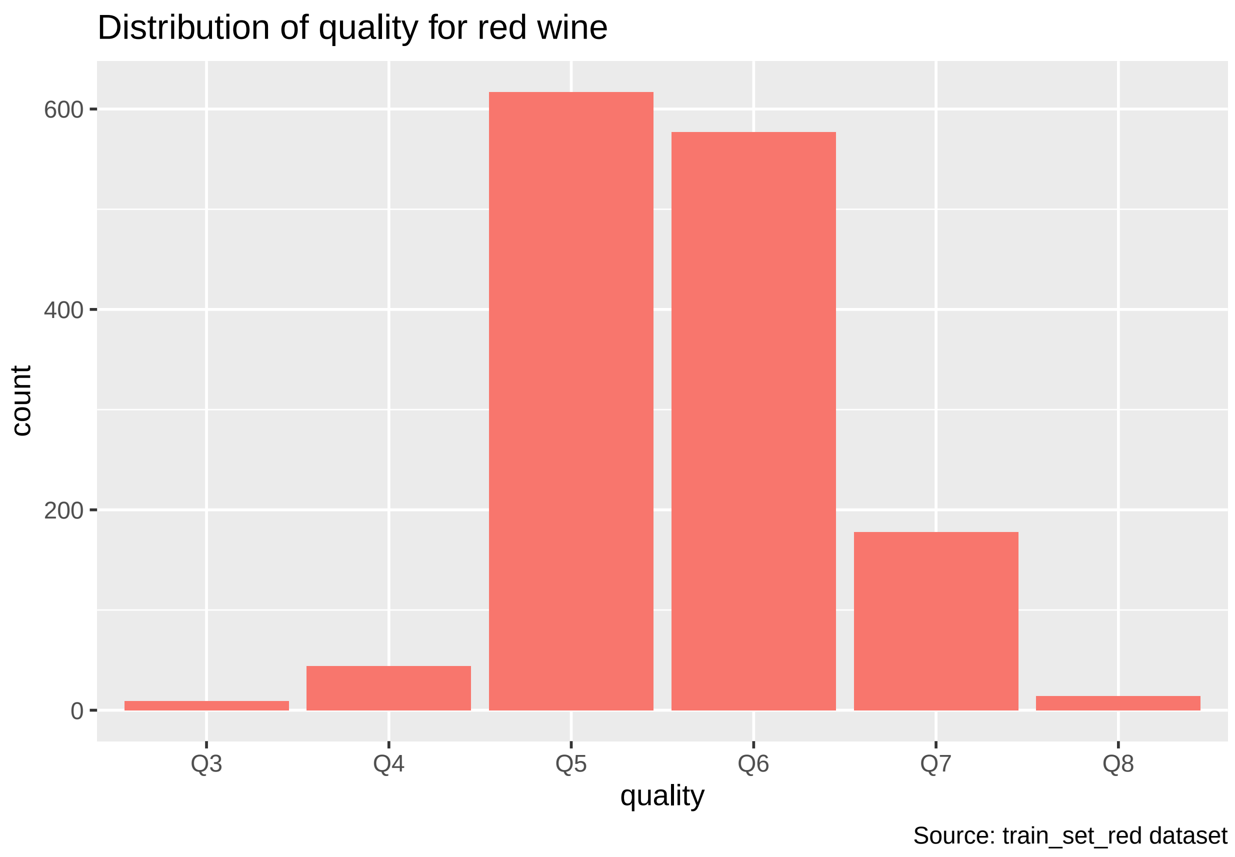
2.3.2 Quality distribution

Before predicting the quality of red wines, we need to understand the quality distribution. Notice that the distribution is highly disproportional: there are many more wines with qualities 5 and 6 than 3,4 and 8.

[2] https://archive.ics.uci.edu/ml/datasets/Wine+Quality
[3] https://portal.vinhoverde.pt/pt/estatisticas

Table 2.4: Vinho Verde Annual Production 1999-2008

Year	White	Red	Red Prevalence (%)
1999/2000	75,322,378	42,430,203	56.33
2000/2001	55,347,997	25,735,143	46.50
2001/2002	85,708,215	40,763,148	47.56
2002/2003	51,552,334	24,586,975	47.69
2003/2004	54,115,085	25,927,194	47.91
2004/2005	61,684,640	29,303,980	47.51
2005/2006	58,653,274	27,813,744	47.42
2006/2007	53,631,404	30,199,897	56.31
2007/2008	45,409,386	18,261,915	40.22



2.3.3 Variable importance

There are 11 variables, or features, that may be used as predictors, but not all of them may be useful in predicting the wine type. We use the R function `filterVarImp` to estimate the importance of each variable. For two class outcomes, such as predicting whether the wine is red or white, the area under the ROC curve is used as a measure of variable importance[4].

[4] https://topepo.github.io/caret/variable-importance.html

Total sulfur dioxide, chlorides and volatile acidity are the most important for predicting wine type, while alcohol, citric acid and residual sugar are the least important.

Table 2.5: Variable Importance for Wine Type

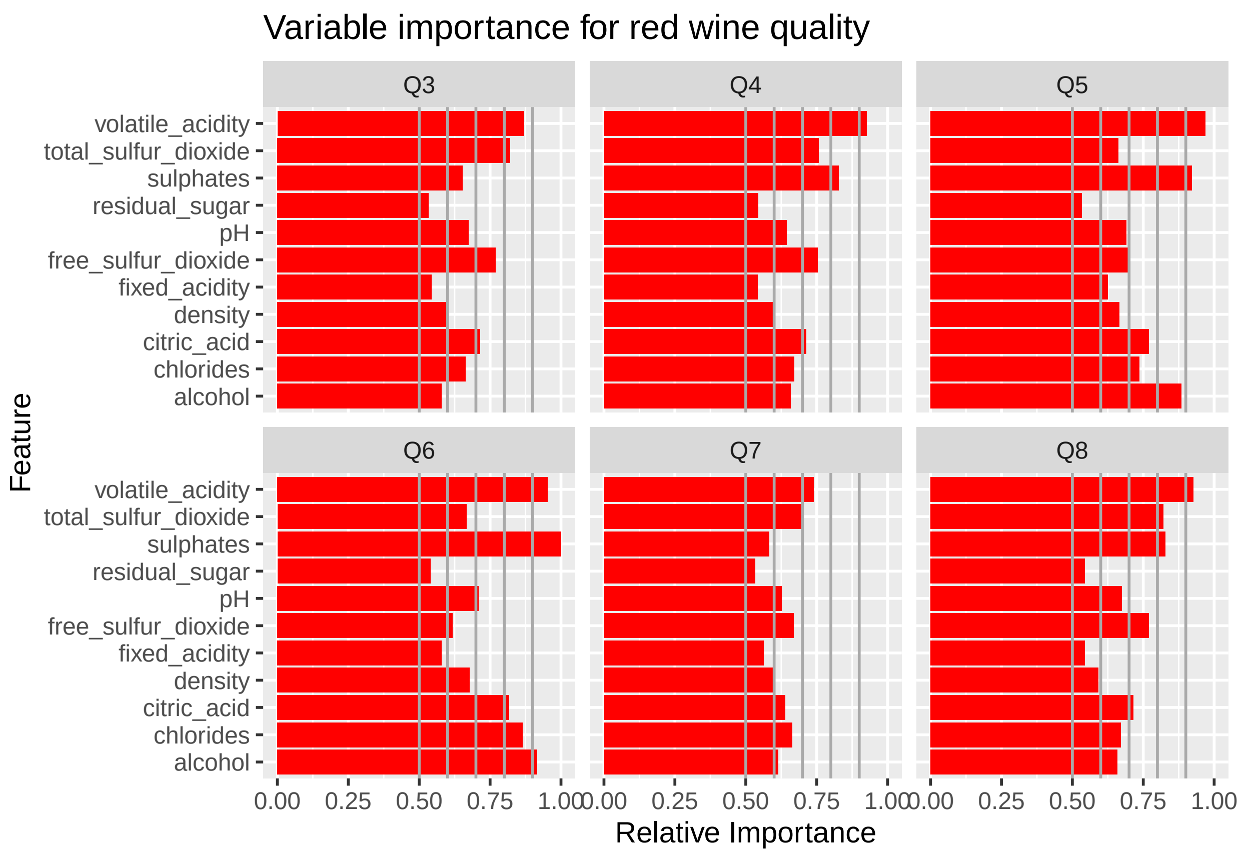
Feature	Red	White
total_sulfur_dioxide	0.953	0.953
chlorides	0.944	0.944
volatile_acidity	0.902	0.902
free_sulfur_dioxide	0.848	0.848
sulphates	0.83	0.83
fixed_acidity	0.782	0.782
density	0.772	0.772
pH	0.728	0.728
residual_sugar	0.674	0.674
citric_acid	0.608	0.608
alcohol	0.513	0.513

For multi-class outcomes, such as predicting the wine quality in a scale of 1 to 9, the problem is decomposed into all pair-wise problems and the area under the curve is calculated for each class pair (i.e. class 1 vs. class 2, class 2 vs. class 3 etc.). For a specific class, the maximum area under the curve across the relevant pair-wise AUC's is used as the variable importance measure.

Volatile acidity, sulphates and alcohol have very high AUC values for red wine quality.

Table 2.6: Variable importance for red Wine quality

Feature	Q3	Q4	Q5	Q6	Q7	Q8
fixed_acidity	0.544	0.543	0.626	0.579	0.564	0.544
volatile_acidity	0.869	0.928	0.969	0.952	0.740	0.928
citric_acid	0.715	0.713	0.771	0.817	0.639	0.715
residual_sugar	0.534	0.544	0.534	0.540	0.534	0.544
chlorides	0.664	0.671	0.737	0.865	0.664	0.671
free_sulfur_dioxide	0.769	0.754	0.702	0.617	0.670	0.769
total_sulfur_dioxide	0.822	0.758	0.662	0.667	0.701	0.822
density	0.598	0.598	0.665	0.679	0.598	0.591
pH	0.674	0.645	0.691	0.710	0.626	0.674
sulphates	0.653	0.827	0.922	1.000	0.584	0.827
alcohol	0.579	0.660	0.885	0.917	0.614	0.660



In practice, the distribution of features isn't so clear, overlapping for different outcomes. In this case, machine learning models use two or more features for prediction. Also, highly correlated features provide low predictive power.

So, let's look at each feature distribution for each wine type and quality of red wines, then we check the correlation among the features.

2.3.4 Data visualization

It's easier to identify the distribution using data visualization. First, we load some packages and define a function used in this section.

```r
# Install and load the libraries used for visualization
# The 'load_lib' function was defined earlier.
load_lib(c("gridExtra", "ggridges", "ggplot2",
           "gtable", "grid", "egg"))


# The 'grid_arrange_shared_legend' function creates a grid of
# plots with one legend for all plots.
# Reference: Baptiste Auguié - 2019
# https://cran.r-project.org/web/packages/egg/vignettes/Ecosystem.html
```

```r
grid_arrange_shared_legend <-
  function(...,
           ncol = length(list(...)),
           nrow = 1,
           position = c("bottom", "right")) {

    plots <- list(...)
    position <- match.arg(position)
    g <-
      ggplotGrob(plots[[1]] + theme(legend.position = position))$grobs
    legend <- g[[which(sapply(g, function(x)
      x$name) == "guide-box")]]
    lheight <- sum(legend$height)
    lwidth <- sum(legend$width)
    gl <- lapply(plots, function(x)
      x + theme(legend.position = "none"))
    gl <- c(gl, ncol = ncol, nrow = nrow)

    combined <- switch(
      position,
      "bottom" = arrangeGrob(
        do.call(arrangeGrob, gl),
        legend,
        ncol = 1,
        heights = unit.c(unit(1, "npc") - lheight, lheight)
      ),
      "right" = arrangeGrob(
        do.call(arrangeGrob, gl),
        legend,
        ncol = 2,
        widths = unit.c(unit(1, "npc") - lwidth, lwidth)
      )
    )

    grid.newpage()
    grid.draw(combined)

    # return gtable invisibly
    invisible(combined)

  }
```

2.3.5 Density plots

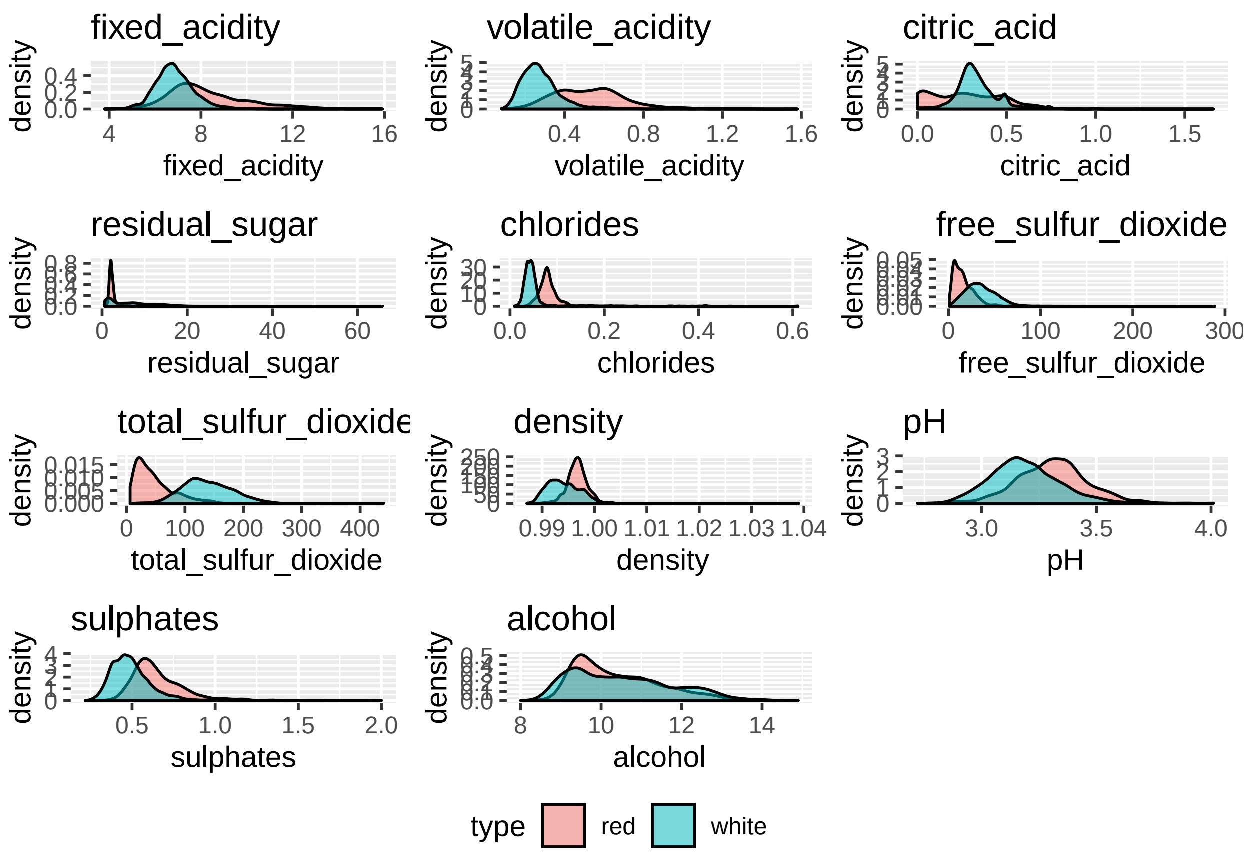
The density plots show each predictor distribution grouped by wine type. Features with little or no overlap make better prediction with higher accuracy, sensitivity and specificity.

To understand this, let's denote Y as the wine type, assuming $Y = 1$ for red wine and $Y = 0$ for white wine, and X as the predictors. We want to find one or more predictors X with high conditional probability $Pr(Y = 1|X = 1)$ and low conditional probability $Pr(Y = 0|X = 1)$.

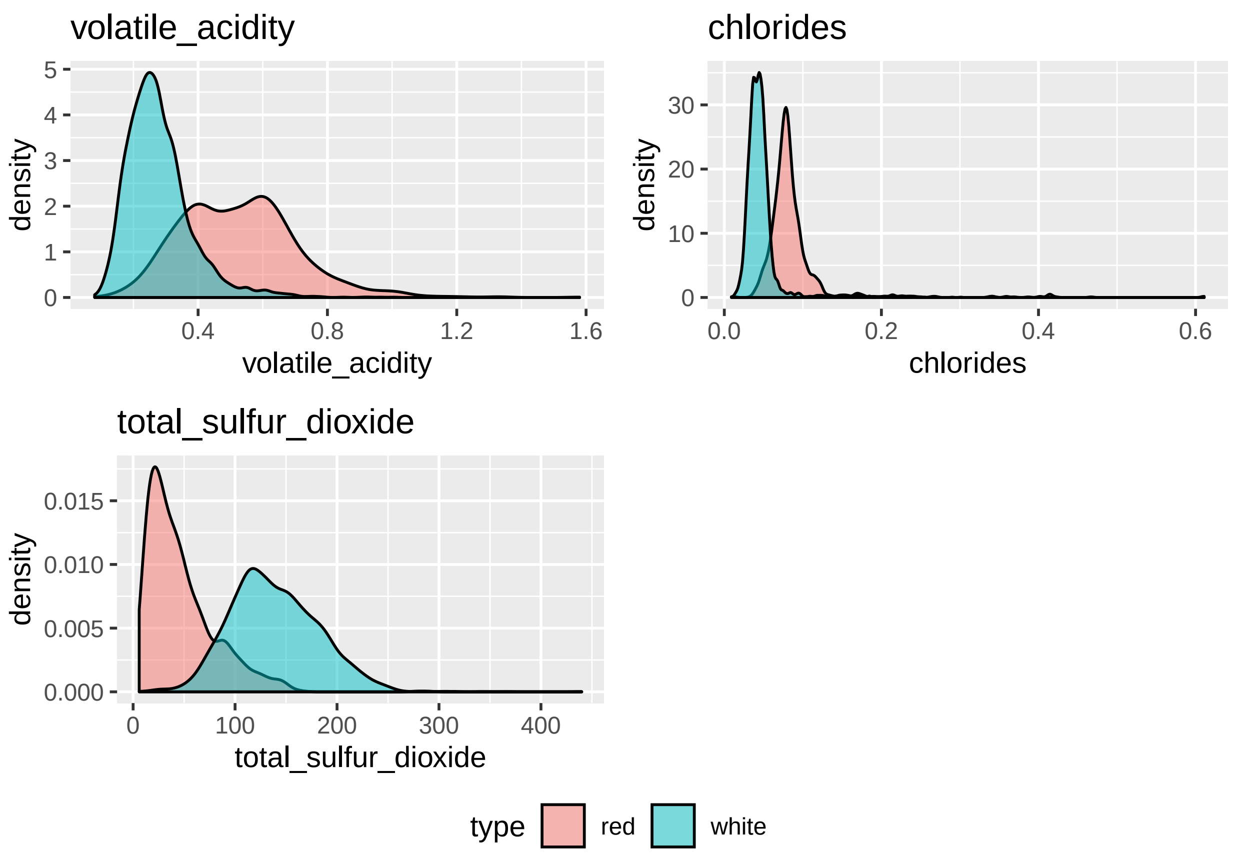
A predictor meets this criteria when there's a clear distinction in the distribution of the observed predictor x for each observed outcome y. For example, if $x < 1$ when $y = 1$ and $x > 1$ when $y = 0$, the prediction just needs to follow this distribution.

The distribution of alcohol is very similar for red and white wines, meaning that alcohol is a bad predictor for wine type. On the other hand, the distribution of volatile acidity, chlorides and total sulfur dioxide is clearly shifted for both types. Sulphates, citric acid and pH have large overlap. The distribution on the density plots confirms the findings of variable importance.

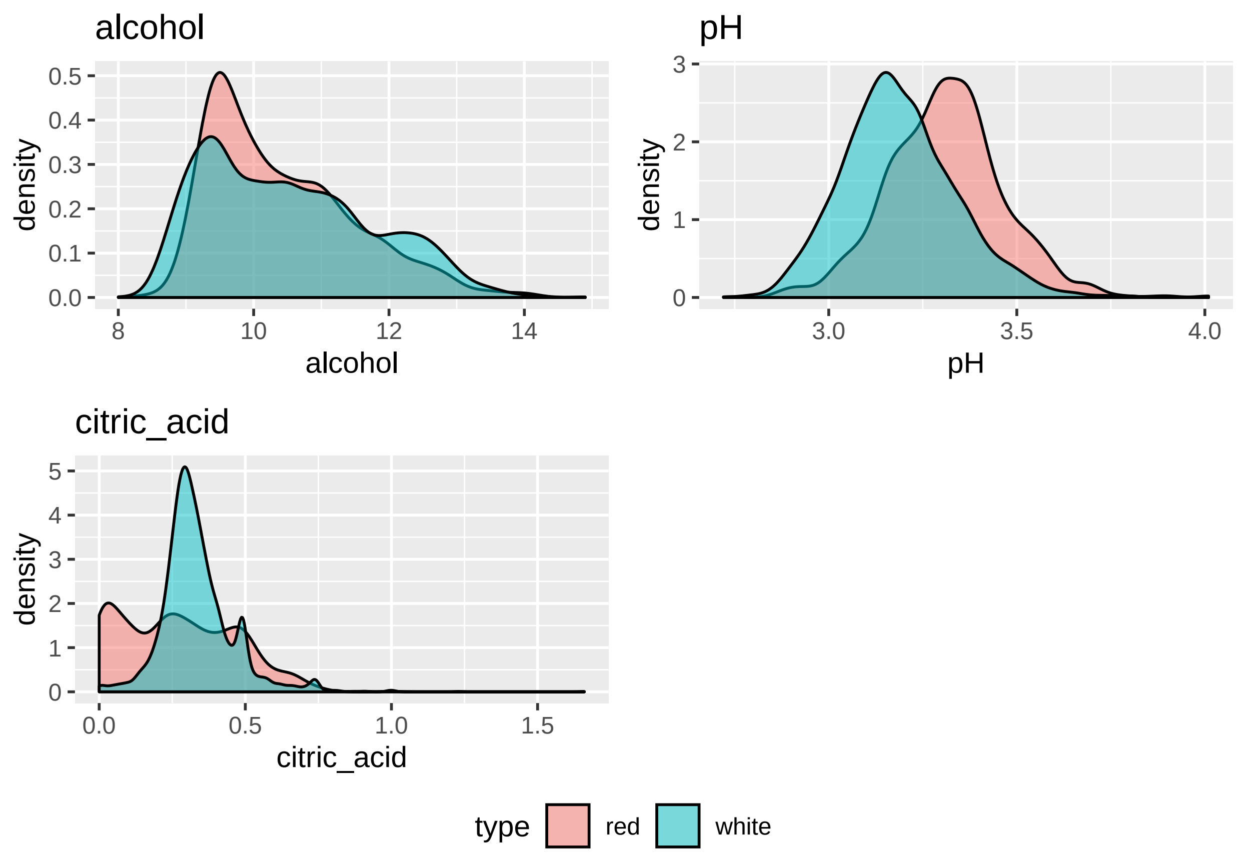
```r
# Density grid
dens_grid <- lapply(xcol, FUN=function(var) {
  # Build the plots
  ggplot(train_set) +
    geom_density(aes_string(x = var, fill = "type"), alpha = 0.5) +
    ggtitle(var)
})
do.call(grid_arrange_shared_legend, args=c(dens_grid, nrow = 4, ncol = 3))
```



We notice that volatile acid, chlorides and total sulfur dioxide have distinct peaks for red and white wines. These predictors may help distinguish both wine types. These three variables distributions are shifted to the left side of the x axis, indicating possible outliers in the right side. The plot above is very small, so let's make a larger plot for them.



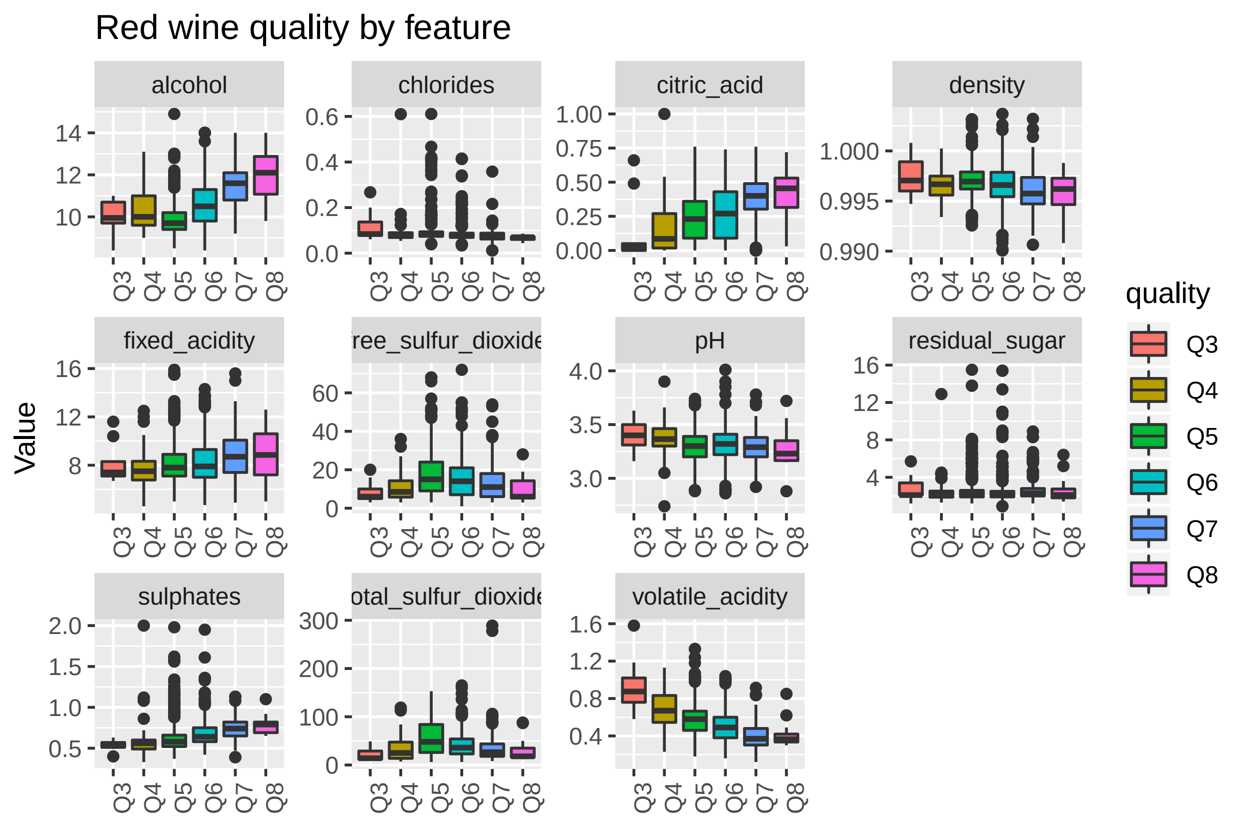
The distribution of alcohol, pH and citric acid overlaps in both wine types.



2.3.6 Box plots

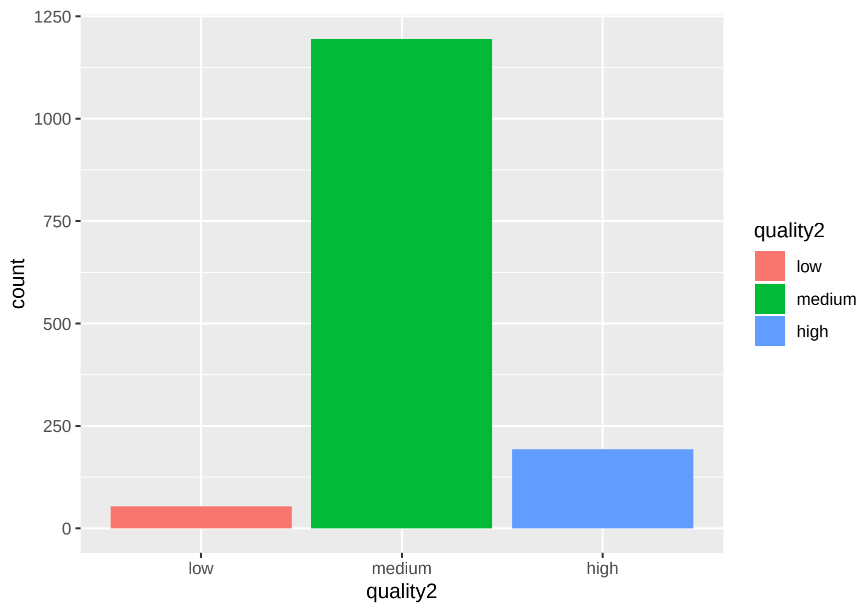
Box plots show a summary of the main values in a distribution: the lower and upper limits, first and third quantiles, the mean and outliers. The distribution overlaps among all quality levels for all predictors. Also, the mean is flat for all quality levels and predictors, except for citric acid and sulphates that increases with quality levels and volatile acidity that decreases.

All features have outliers.



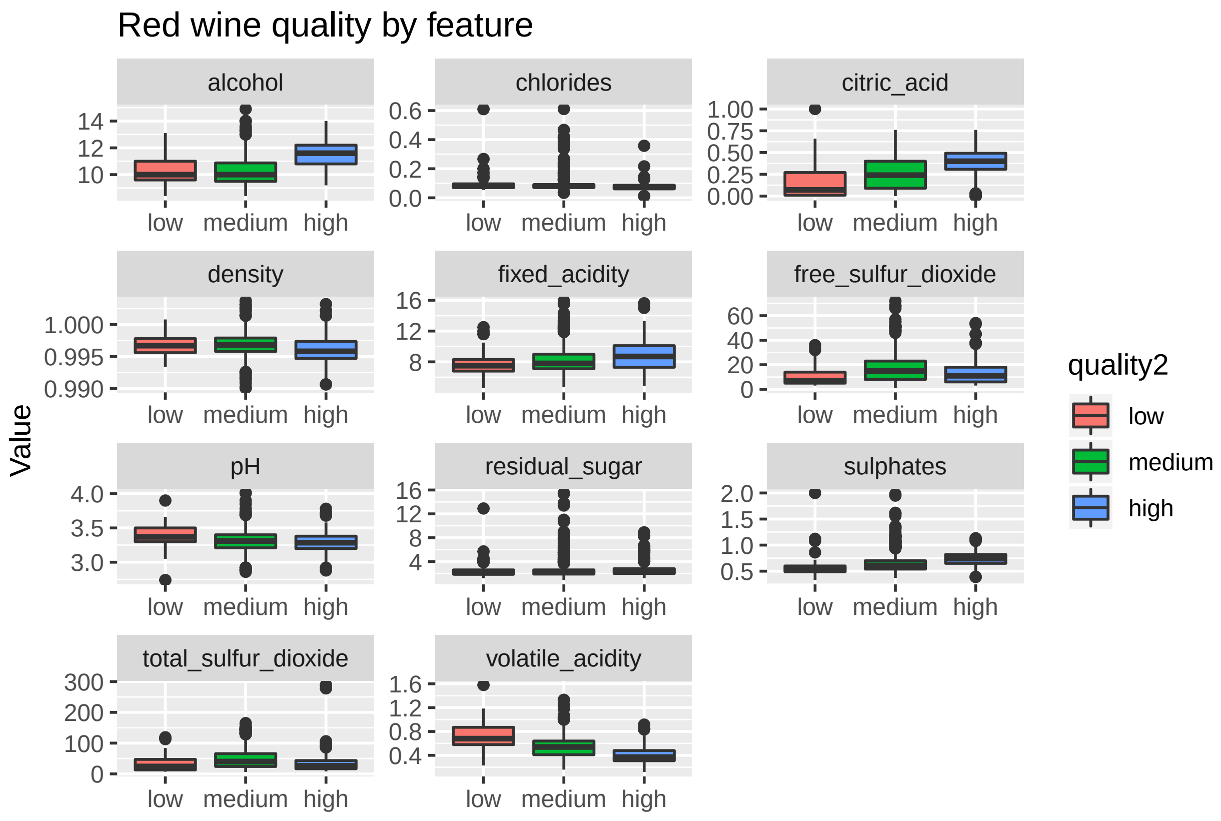
There's no clear distinction of quality levels among the predictors. Maybe grouping the quality values may help. Let's denote `low` quality for levels 3 and 4, `medium` for levels 5 and 6, and `high` for levels 7 and 8.

```r
train_set_red <- train_set_red %>%
  mutate(quality2 = factor(case_when(
    quality %in% c("Q3", "Q4") ~ "low",
    quality %in% c("Q5", "Q6") ~ "medium",
    quality %in% c("Q7", "Q8") ~ "high"),
    levels = c("low", "medium", "high")))

test_set_red <- test_set_red %>%
  mutate(quality2 = factor(case_when(
    quality %in% c("Q3", "Q4") ~ "low",
    quality %in% c("Q5", "Q6") ~ "medium",
    quality %in% c("Q7", "Q8") ~ "high"),
    levels = c("low", "medium", "high")))

train_set_red %>% ggplot(aes(quality2, fill = quality2)) + geom_bar()
```



All distributions remains flat, except alcohol and citric acid that increases with quality and may help distinguish high quality wines.

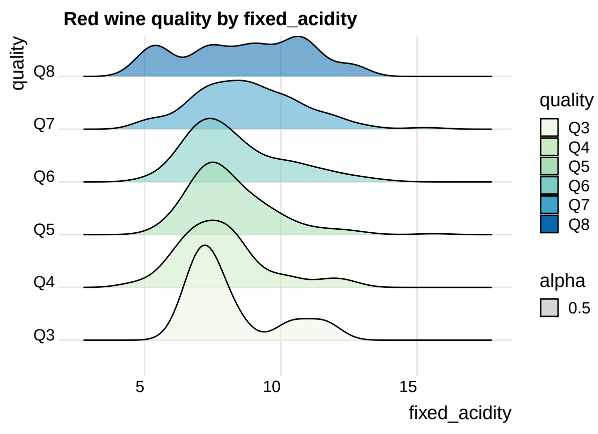
Volatile acidity decrease with quality and may help identify low quality wines.

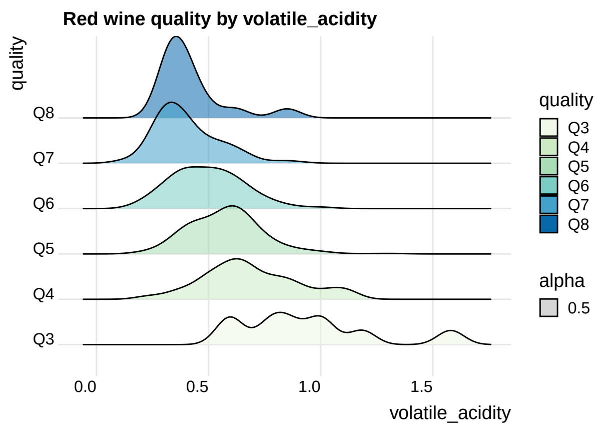


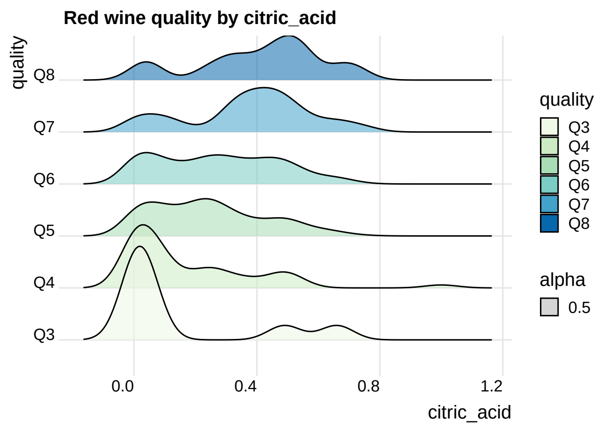
2.3.7 Density ridge plot

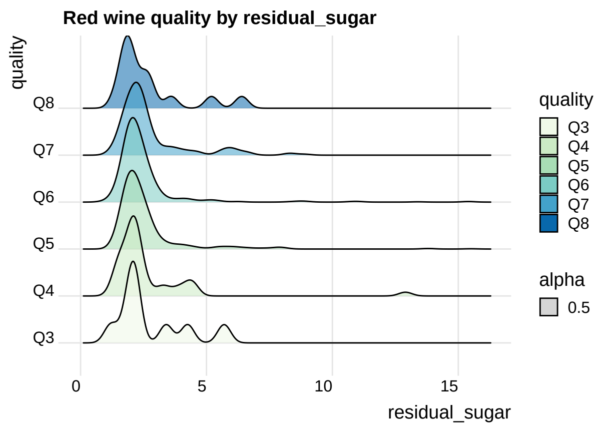
The density plots overlaps the distribution of both wine types. For the visualization of the distribution of features by quality, we use density ridge plots. In every feature the distribution overlap among quality levels, meaning that no single feature alone is able to predict quality.

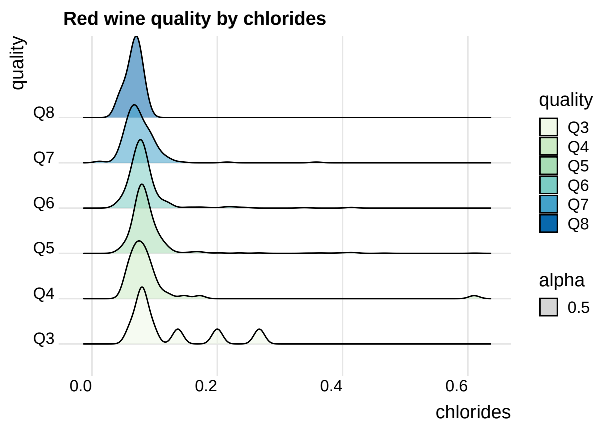
```r
# Density ridge plots
lapply(xcol, FUN=function(var) {

  train_set_red %>%
    ggplot(data = ., aes_string(x = var,
                                y = "quality",
                                fill = "quality",
                                alpha = 0.5)) +
    geom_density_ridges() +
    theme_ridges() +
    theme(axis.text.x = element_text(hjust = 1)) +
    scale_fill_brewer(palette = 4) +
    ggtitle(paste0("Red wine quality by ", var))
})
```

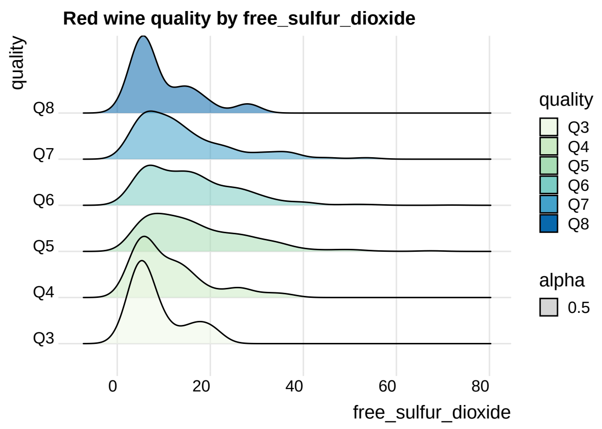


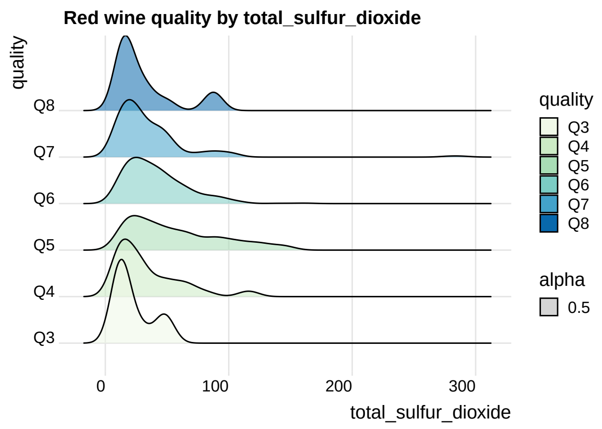


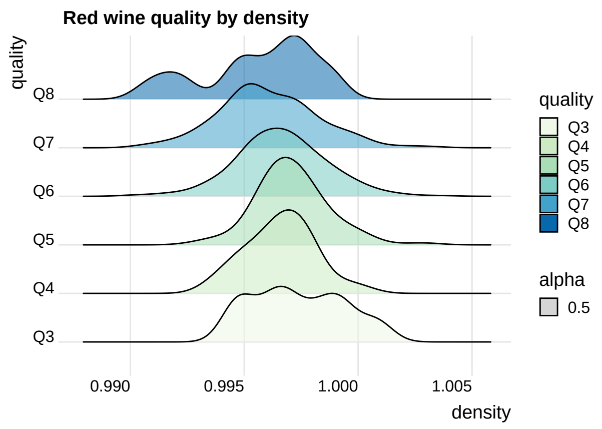


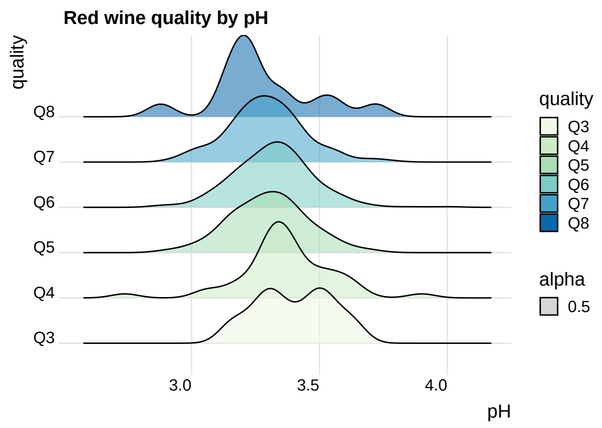


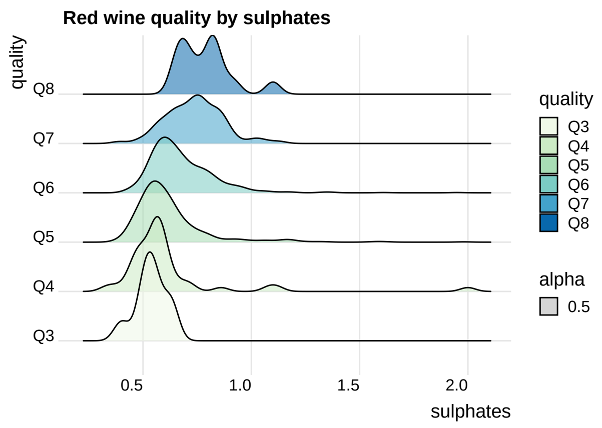

Red wine quality by chlorides
quality
Q8
Q7
Q6
Q5
Q4
Q3
0.0
0.2
0.4
0.6
chlorides
quality
Q3
Q4
Q5
Q6
Q7
Q8
alpha
0.5

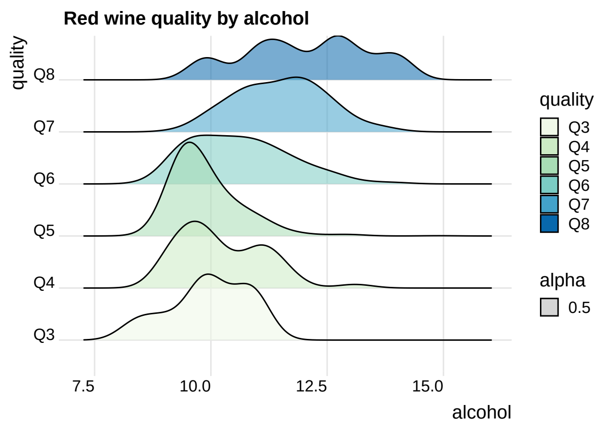

Red wine quality by free_sulfur_dioxide
quality
Q8
Q7
Q6
Q5
Q4
Q3
0
20
40
60
80
free_sulfur_dioxide
quality
Q3
Q4
Q5
Q6
Q7
Q8
alpha
0.5


Red wine quality by total_sulfur_dioxide
quality
Q8
Q7
Q6
Q5
Q4
Q3
0
100
200
300
total_sulfur_dioxide
quality
Q3
Q4
Q5
Q6
Q7
Q8
alpha
0.5


Red wine quality by density
quality
Q8
Q7
Q6
Q5
Q4
Q3
0.990
0.995
1.000
1.005
density
quality
Q3
Q4
Q5
Q6
Q7
Q8
alpha
0.5

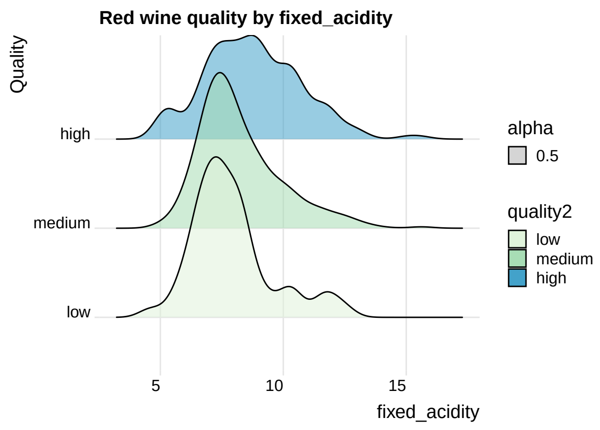

Red wine quality by pH
quality
Q8
Q7
Q6
Q5
Q4
Q3
3.0
3.5
4.0
pH
quality
Q3
Q4
Q5
Q6
Q7
Q8
alpha
0.5

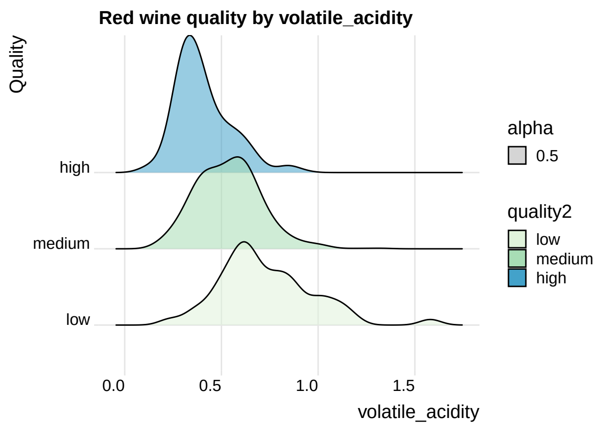

Red wine quality by sulphates
quality
Q8
Q7
Q6
Q5
Q4
Q3
0.5
1.0
1.5
2.0
sulphates
quality
Q3
Q4
Q5
Q6
Q7
Q8
alpha
0.5

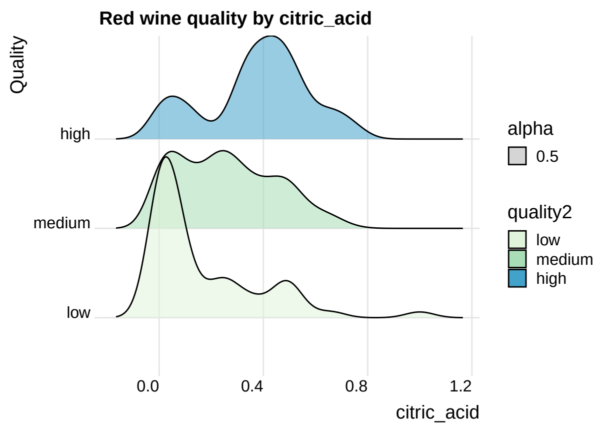


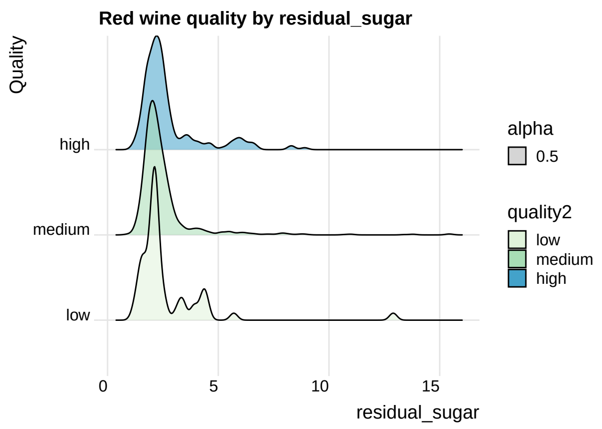
The distribution overlaps for all quality levels and all predictors. Grouping quality levels makes small improvement.

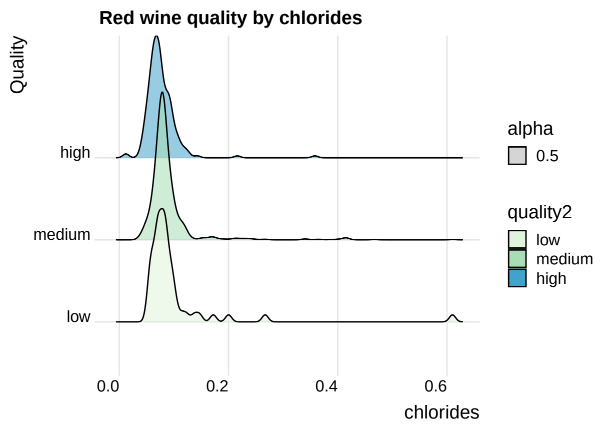
```r
# Density ridge plots
lapply(xcol, FUN=function(var) {

  train_set_red %>%
    ggplot(data = ., aes_string(x = var,
                                y = "quality2",
                                fill = "quality2",
                                alpha = 0.5)) +
    geom_density_ridges() +
    # Format chart
    theme_ridges() +
    scale_fill_brewer(palette = 4) +
    # Format labels
    theme(axis.text.x = element_text(hjust = 1)) +
    ggtitle(paste0("Red wine quality by ", var)) +
    ylab("Quality")
})
```

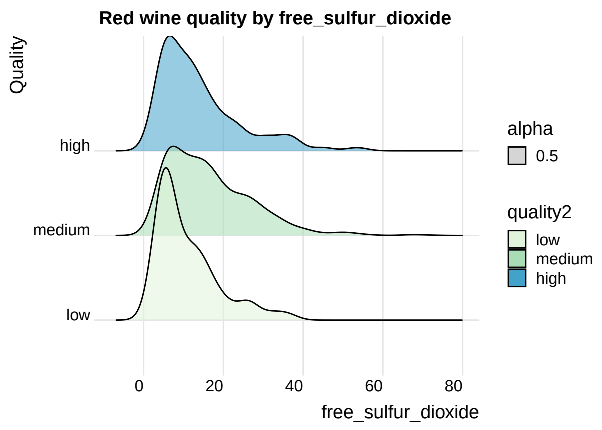


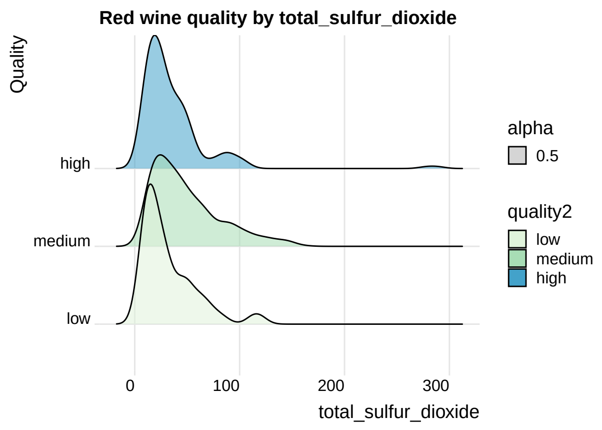


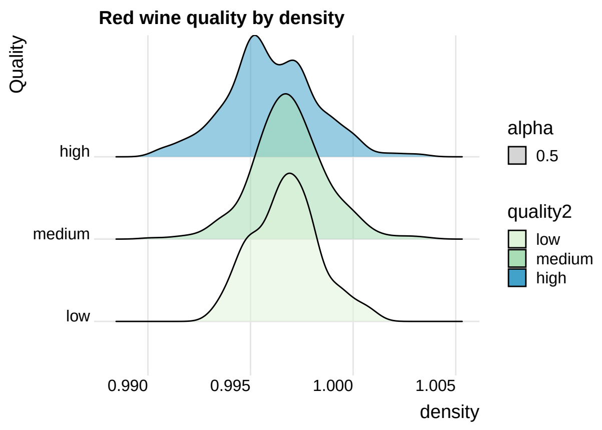

Red wine quality by citric_acid
Quality
high
medium
low
0.0
0.4
0.8
1.2
citric_acid
alpha
0.5
quality2
low
medium
high

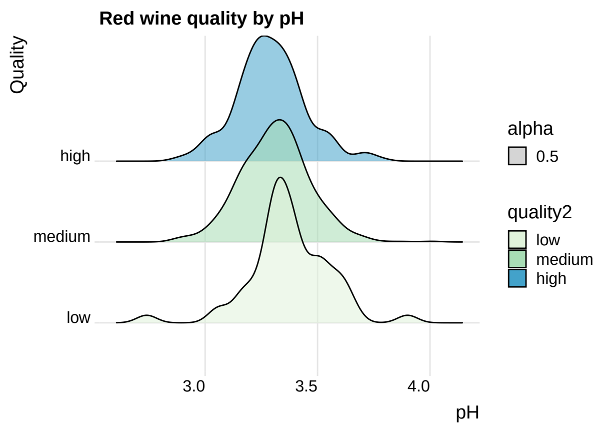

Red wine quality by residual_sugar
Quality
high
medium
low
0
5
10
15
residual_sugar
alpha
0.5
quality2
low
medium
high

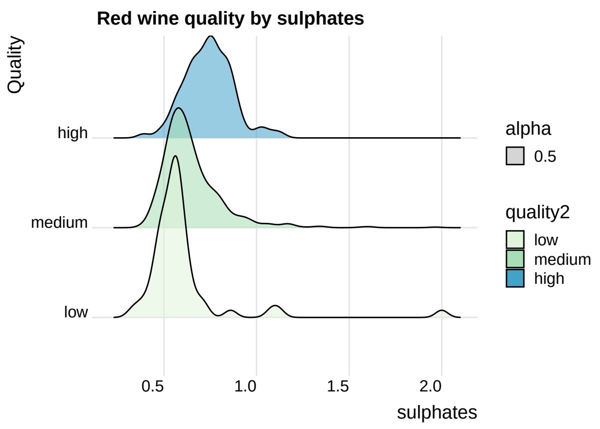

Red wine quality by chlorides
Quality
high
medium
low
0.0
0.2
0.4
0.6
chlorides
alpha
0.5
quality2
low
medium
high

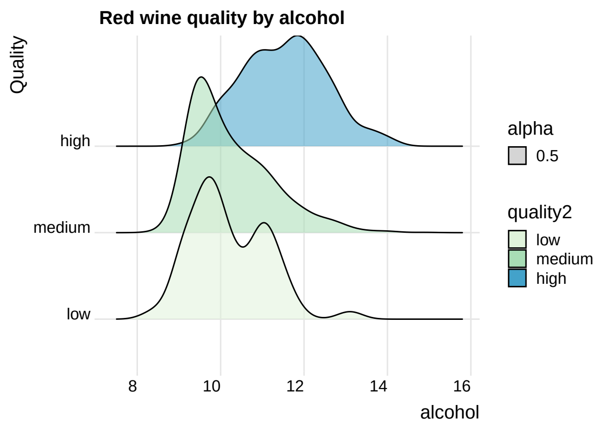

Red wine quality by free_sulfur_dioxide
Quality
high
medium
low
0
20
40
60
80
free_sulfur_dioxide
alpha
0.5
quality2
low
medium
high


Red wine quality by total_sulfur_dioxide
Quality
high
medium
low
0
100
200
300
total_sulfur_dioxide
alpha
0.5
quality2
low
medium
high


Red wine quality by density
Quality
high
medium
low
0.990
0.995
1.000
1.005
density
alpha
0.5
quality2
low
medium
high

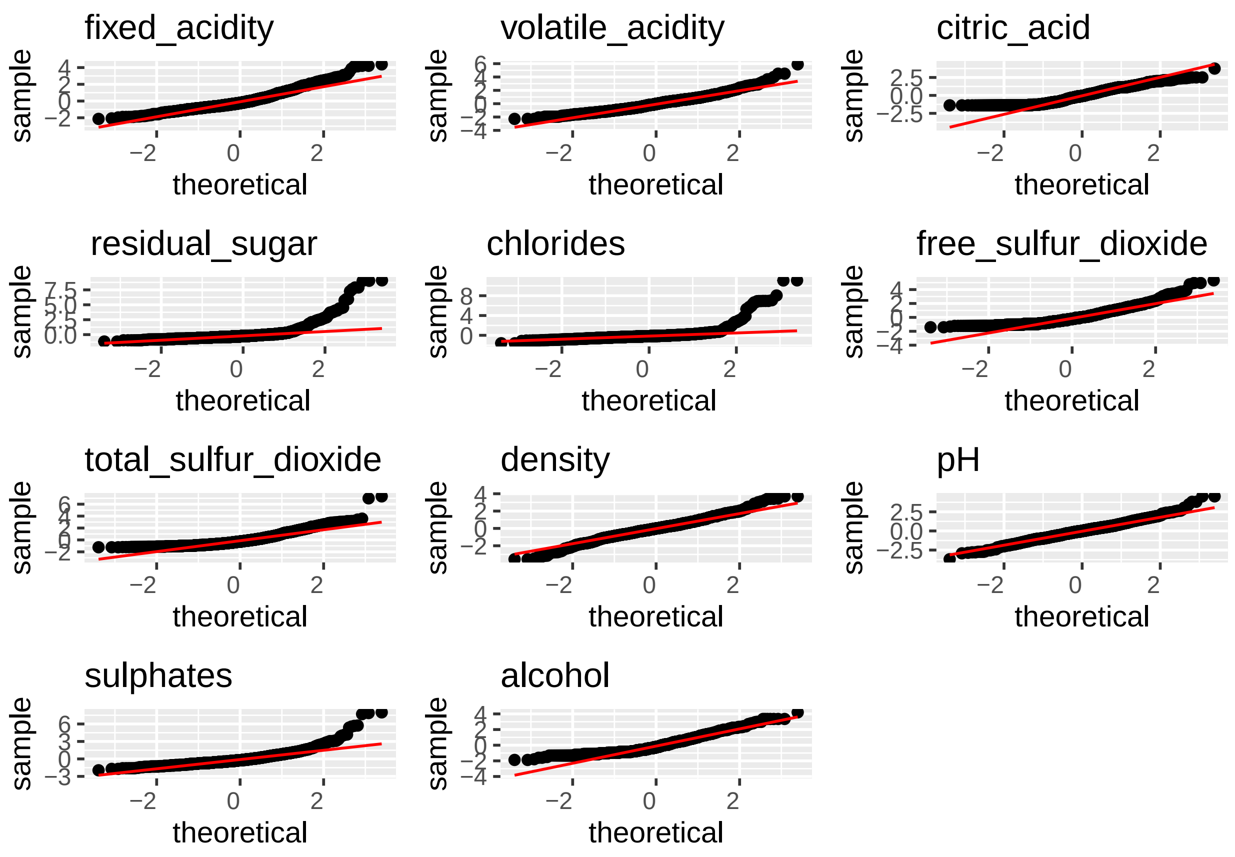






2.3.8 QQ plot

The QQ plots compare the feature distribution with the normal distribution. All variables are close to the normal distribution within two standard deviations of the mean. Some machine learning algorithms, such as LDA and QDA, assume the predictors are normally distributed.



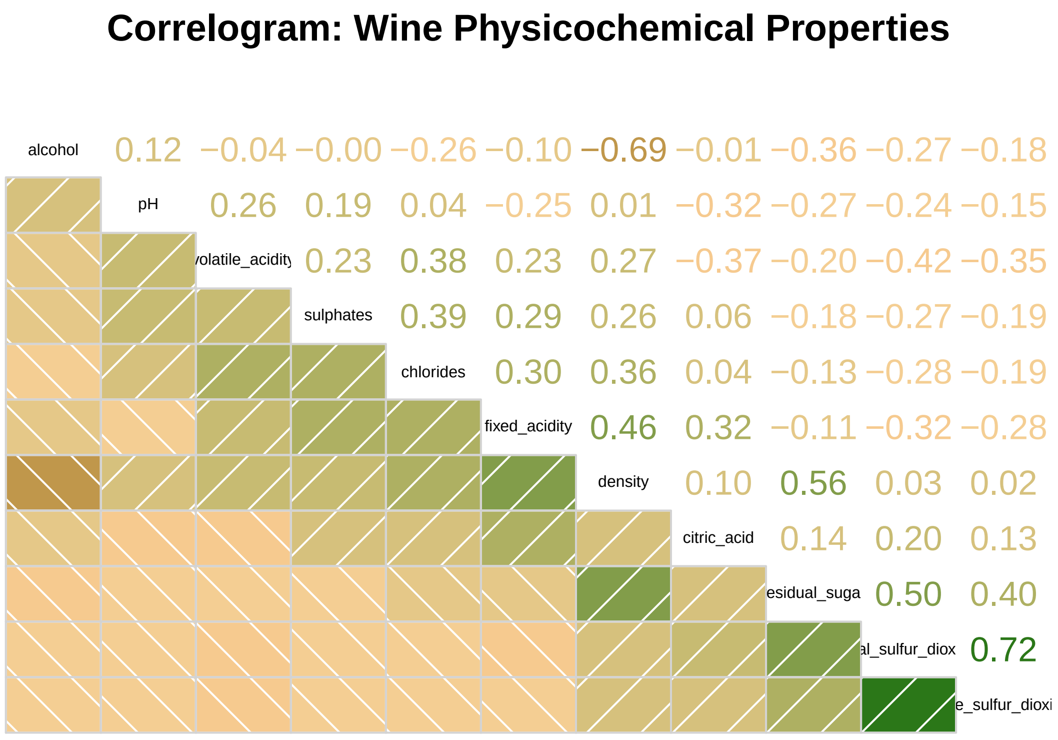
2.3.9 Correlation

The exploration done so far shows that volatile acid, chlorides and total sulfur dioxide are good candidates as wine type predictors. The prediction may be lower than expected if any two of these variables are highly correlated.

The correlogram shows the correlation of all predictors in pairs. The diagonal contains the variable names. The correlation of two variables are in the intersection of the variables rows and columns. The color intensity of the boxes and numbers indicate the correlation degree: strong colors indicate high correlation, whereas light colors indicate low correlation.

In this analysis, we assume that low correlation is between -0.5 and 0.5, and high correlation otherwise. Most colors in the correlogram are light and the correlations are low.

```
# Load the "corrgram" package to draw a correlogram
load_lib("corrgram")

# Draw a correlogram
corrgram(train_set[,xcol], order=TRUE,
         lower.panel = panel.shade,
         upper.panel = panel.cor,
         text.panel  = panel.txt,
         main = "Correlogram: Wine Physicochemical Properties",
         col.regions=colorRampPalette(c("darkgoldenrod4", "burlywood1",
                                        "darkkhaki", "darkgreen")))
```



The table below summarizes the pairs of variables with high correlation.

The correlations of volatile acid, chlorides and total sulfur dioxide are low.

Table 2.7: Correlation Matrix - Selected Features

Feature	Volatile acidity	Chlorides	Total sulfur dioxide
Volatile acidity	1.000	0.378	-0.416
Chlorides	0.378	1.000	-0.276
Total sulfur dioxide	-0.416	-0.276	1.000

2.4 Modeling

During data exploration we learned that total sulfur dioxide, chlorides and volatile acidity are good candidates for wine type prediction. The AUC is very high, the distributions have low overlapping areas and the correlations are low.

On the other hand, the quality prediction of red wines may be challenging. Although some features with high AUC have low correlations, all features present large distribution overlapping areas. Besides this, the prevalence of wines with qualities 3, 4 and 8 is very low.

In this section we discuss several modeling approaches to predict red and white wines and wine quality.

2.4.1 Simple Model

The simplest model is just assuming that all wines are red or white. Since there's more white than red wines, the model predicts all wines are white. The accuracy at 75.4 is higher than 50%, the specificity is 1 and sensitivity 0. The model is good at predicting white wines, but it isn't very useful at predicting red wines. A better model should have higher sensitivity, although it may lower specificity and accuracy.

2.4.2 Random Model

Another model is to use the sample probabilities of each wine type as a predictor. We know that the prevalence of red and white wines is 24.6% and 75.4% respectively, so we just randomly assign red or white using these proportions. The problem with this approach is that we don't know the actual distribution in the population, and the actual distribution may fluctuate over time. Any machine learning model should do better than both models.

2.4.3 Single Predictor

Since the distributions of total sulfur dioxide, chlorides and volatile acidity stratified by wine type have small overlapping areas, we can find a cutoff value that maximizes the conditional probability.

$$Pr(Y = k|X = x)$$

Then, we combine the results in a single prediction, using majority of votes.

2.4.4 Linear Regression

We want to build a model that for any predictor X, the algorithm predicts the class `k`, or wine type, with the largest conditional probability expressed by this equation:

$$Pr(Y = k|X_1 = x_1, ..., X_p = x_p)$$

Where Y is the outcome and p is the number of predictors.

Since we selected three predictors during data exploration, our model becomes:

$$Pr(Y = k|X_1 = x_1, X_2 = x_2, X_3 = x_3) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

where x_1 is total sulfur dioxide, x_2 is chlorides and x_3 is volatile acidity.

The difference with the previous model is that linear regression builds a function that best fits the data.

2.4.5 K-Nearest Neighbors (KNN)

The k-nearest neighbors algorithm estimates the conditional probability:

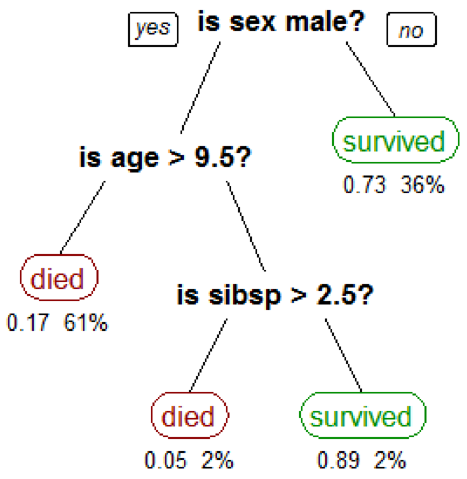
$$p(x_1, .., x_p) = Pr(Y = k|X_1 = x_1, .., X_p = x_p)$$

The algorithm calculates the euclidean distance of all predictors, then for any point $(x_1, .., x_p)$ in the multidimensional space that we want to predict, the algorithm determines the distance to k points. The k nearest points is refereed as neighborhood.

For $k = 1$ the algorithm finds the distance to a single neighbor. For k equals to the number of samples, the algorithm uses all points. Hence, k is a tuning parameter that can be calculated running the algorithm for several values of k and picking the result with highest accuracy or AUC.

2.4.6 Classification and Regression Trees

A tree is basically a flow chart of *yes* or *no* questions, such as in the picture below.



(Source: Wikimedia Commons)

Classification trees are easy to read and understand, and can model human decision process. However, they don't have the best accuracy, they are hard to train and they are unstable in changes in the data.

2.4.7 Random Forest

Random forests improve prediction performance over classification trees by averaging multiple decision trees. The algorithm creates several random subsets of the original data, in this case the training set, and calculates the classification trees, then the final result is the average of all trees.

The name random forest derives from the random process of splitting the data and creating many trees, or a forest.

2.4.8 Linear and Quadratic Discriminant Analysis

Linear discriminant analysis, or LDA, and quadratic discriminant analysis, or QDA, assumes that all predictors are normally distributed and have the same standard deviations[5].

The decision boundary of LDA is linear, while QDA is an extension of LDA with quadratic boundaries.

Both are intended for classification problems where the output variable is categorical. LDA supports both binary and multi-class classification.

Both LDA and QDA can be derived from simple probabilistic models which model the class conditional distribution of the data $P(X|y=k)$ for each class k. Predictions can then be obtained by using Bayes' rule[6]:

$$Pr(y=k|X) = \frac{Pr(X|y=k)Pr(y=k)}{Pr(X)} = \frac{Pr(X|y=k)Pr(y=k)}{\sum_l Pr(X|y=l) \cdot Pr(y=l)}$$

and we select the class k which maximizes this conditional probability.

LDA and QDA work better with few predictors.

2.4.9 Cross Validation

The original dataset is partitioned in the training set used to train the model, and the test set used to predict the values with the trained model. Cross validation partitions the training

[5] https://machinelearningmastery.com/linear-discriminant-analysis-for-machine-learning/

[6] https://scikit-learn.org/stable/modules/lda_qda.html

set in the same way and performs the training and prediction several times. Then, the result with the best RMSE, accuracy, AUC or the chosen metric is selected. This process can be used in conjunction to tuning parameters, for example picking the best k number of neighbors in knn.

2.4.10 Ensemble

Combining the results of several predictions may improve prediction accuracy. This method is called ensemble and consists in selecting the most common class for a given set of observations.

For example, suppose the predictions of wine type for three models as illustrated in the table below. The ensemble is just the majority of votes of the combined models.

Model 1	Model 2	Model 3	Ensemble
red	red	red	red
red	white	red	red
white	red	white	white

Generally, the final result provides better accuracy than the best model.

2.4.11 Clustering

Clustering is a methodology used for unknown outcomes and belongs to the machine learning category of unsupervised learning. The predictors are grouped in clusters that may overlap. Although clustering is beyond the scope of this project, we'll apply this technique.

One common clustering method is `k-means`, that groups the predictors in k clusters. Each observation is assigned to the cluster that has the closest center. Then, the centers are redefined with the observations of each cluster using the column means. The process is repeated until the centers converge. This approach is similar to `knn`.

The number of clusters k is a tuning parameter, and may be estimated before running the algorithm.

Chapter 3

Results

This section presents the modeling results and discusses the model performance. For the most part of this section, the models are applied to predicting wine type, allowing the comparison of different models. The prediction of red wine quality is performed in the Predicting Quality section.

Formula used in this section.

```r
# Formula used in predictions
fml <- as.formula(paste("type", "~",
                        paste(xcol, collapse=' + ')))
```

3.1 Single Predictor

Volatile acidity, chlorides and total sulfur dioxide have the lowest overlapping area in the density plots of wine type. In this model, we calculate accuracy for several cutoff values in the distribution range, and pick the parameters with highest accuracy.

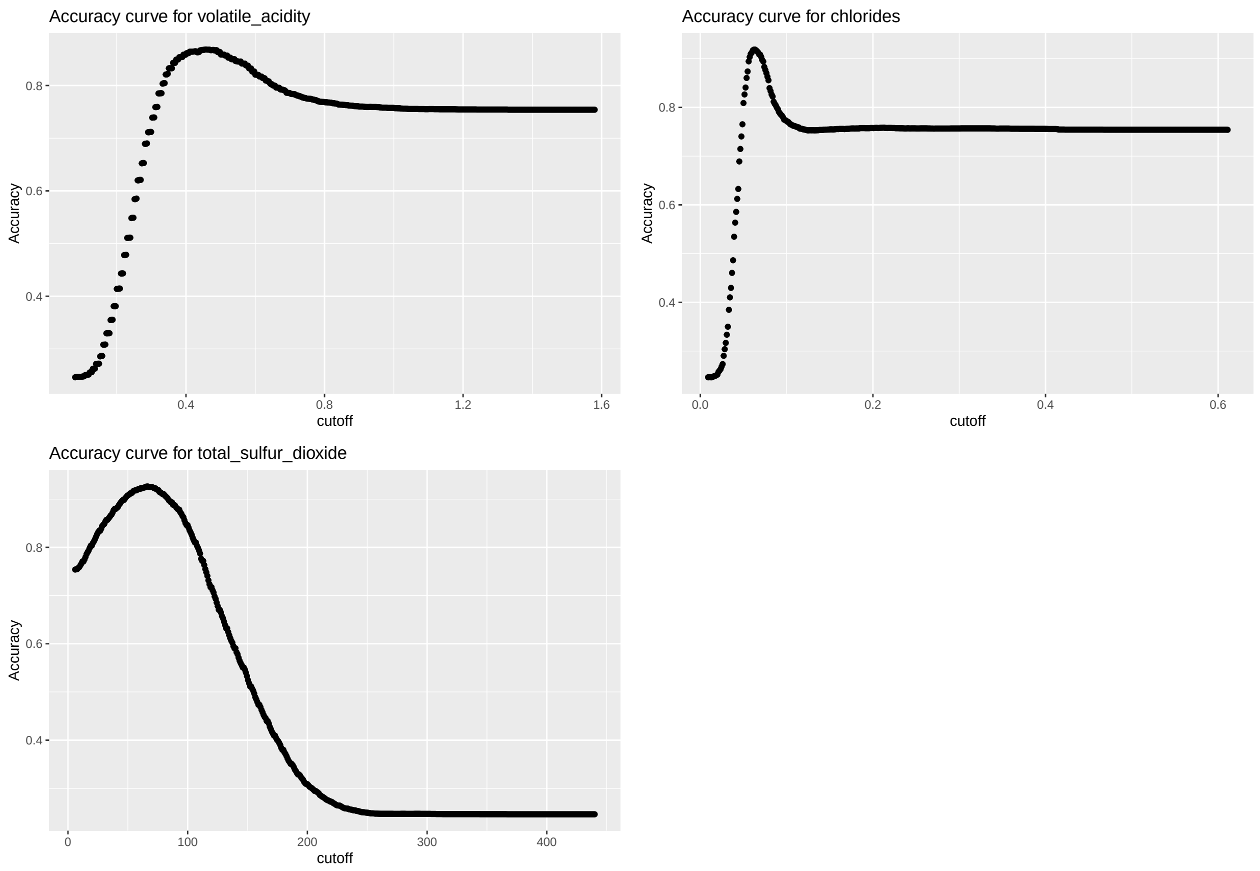
This process is repeated for each of these three predictors, then the result is combined in a single prediction. The final combination has superior performance over any individual predictor.

```r
# Create a list with variable names and cutoff decision rule.
# If the predicted value is lower than the cutoff value, the first color
# is chosen, otherwise the second. To understand this, look at the
# density plots in data visualization.
type_var <- list( c("white", "red"), c("white", "red"), c("red", "white"))
names(type_var) <- c("volatile_acidity", "chlorides", "total_sulfur_dioxide")

# Create an empty results table. The first row
# contains NAs and will be removed after the predictions.
```

```r
type_results <<- data.frame(Feature = NA,
                            Accuracy = NA,
                            Sensitivity = NA,
                            Specificity = NA,
                            stringsAsFactors = FALSE)

# Prediction function
preds <- sapply(1:length(type_var), function(x){

    # Get the variable name
    var <- names(type_var[x])

    # Cutoff value is the distribution range divided by 500
    cutoff <- seq(min(train_set[,var]),
                  max(train_set[,var]),
                  length.out = 500)

    # Calculate accuracy
    acc <- map_dbl(cutoff, function(y){
      type <- ifelse(train_set[,var] < y, type_var[[x]][1],
                     type_var[[x]][2]) %>%
        factor(levels = levels(train_set$type))

      # Accuracy
      mean(type == train_set$type)
      })

    # Build the accuracy vs cutoff curve
    acc_plot <- data.frame(cutoff = cutoff, Accuracy = acc) %>%
      ggplot(aes(x = cutoff, y = Accuracy)) +
        geom_point() +
        ggtitle(paste0("Accuracy curve for ", var))

    # Print the plot
    print(acc_plot)

    # Predict new values in the test set
    # The model uses the cutoff value with the best accuracy.
    max_cutoff <- cutoff[which.max(acc)]
    y_hat <- ifelse(test_set[,var] < max_cutoff,
                    type_var[[x]][1], type_var[[x]][2]) %>%
      factor(levels = levels(test_set$type))

    # Calculate accuracy, specificity and sensitivity
    acc <- max(acc)
    sens <- sensitivity(y_hat, test_set$type)
    spec <- specificity(y_hat, test_set$type)

    # Update results table
    type_results <<- rbind(type_results,
                           data.frame(Feature = names(type_var[x]),
                                      Accuracy = acc,
                                      Sensitivity = sens,
                                      Specificity = spec,
                                      stringsAsFactors = FALSE))

    # The prediction will be used in the ensemble
    return(y_hat)
})

# Remove first row with NA
type_results <- type_results[2:nrow(type_results),]

# Combine the results using majority of votes
y_hat_ens <-as_factor(data.frame(preds) %>%
                        mutate(x = as.numeric(preds[,1] == "red") +
                                 as.numeric(preds[,2] == "red") +
                                 as.numeric(preds[,3]  == "red"),
                               y_hat = ifelse(x >=2, "red", "white")) %>%
                        pull(y_hat))

# Update results table
type_results <<- rbind(type_results,
                       data.frame(Feature = "Ensemble",
                                  Accuracy = mean(y_hat_ens == test_set$type),
                                  Sensitivity = sensitivity(y_hat_ens, test_set$type),
                                  Specificity = specificity(y_hat_ens, test_set$type),
                                  stringsAsFactors = FALSE))
# Show the results table
as_hux(type_results,
       add_colnames = TRUE) %>%
  # Format header
  set_bold(row = 1, col = everywhere, value = TRUE)        %>%
  set_top_border(row = 1, col = everywhere, value = 1)     %>%
  set_bottom_border(row = c(1,5), col = everywhere, value = 1)  %>%
  # Format cells
  set_align(row = 1:4, col = 2, value = 'right')               %>%
  # Format numbers
  set_number_format(row = everywhere, col = 2:4, value = 3)      %>%
  # Format table
  set_caption("Superior Performance for Combined Predictions")  %>%
  set_position(value = "center")
```

Table 3.1: Superior Performance for Combined Predictions

Feature	Accuracy	Sensitivity	Specificity
volatile_acidity	0.868	0.638	0.947
chlorides	0.919	0.894	0.947
total_sulfur_dioxide	0.926	0.794	0.971
Ensemble	0.969	0.894	0.994



3.2 Linear Regression

Linear regression approximates the multidimensional space of predictors X and outcomes Y into a function. The outcome, which is wine type, is categorical, so we need to convert to number before building the function. Here, we assign 1 to red wine and 0 to white wine.

The predicted values are also numeric, so we need to convert back to categories.

Again, we use only the three main features to predict wine type: total sulfur dioxide, chlorides and volatile acidity.

```
# Train the linear regression model
fit_lm <- train_set %>%
  # Convert the outcome to numeric
  mutate(type = ifelse(type == "red", 1, 0)) %>%
  # Fit the model
  lm(type ~ total_sulfur_dioxide + chlorides + volatile_acidity, data = .)

# Predict
p_hat_lm <- predict(fit_lm, newdata = test_set)

# Convert the predicted value to factor
y_hat_lm <- factor(ifelse(p_hat_lm > 0.5, "red", "white"))

# Evaluate the results
caret::confusionMatrix(y_hat_lm, test_set$type)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction red white
##      red   151     7
##      white   9   483
##
##                Accuracy : 0.975
##                  95% CI : (0.96, 0.986)
##     No Information Rate : 0.754
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.933
##
##  Mcnemar's Test P-Value : 0.803
##
##             Sensitivity : 0.944
##             Specificity : 0.986
##          Pos Pred Value : 0.956
##          Neg Pred Value : 0.982
##              Prevalence : 0.246
##          Detection Rate : 0.232
##    Detection Prevalence : 0.243
##       Balanced Accuracy : 0.965
```

```
##
##        'Positive' Class : red
##
```

3.3 KNN

Now we use all features to predict wine type.

```r
# Train
fit_knn <- knn3(formula = fml, data = train_set, k = 5)

# Predict
y_knn <- predict(object = fit_knn,
                 newdata = test_set,
                 type ="class")

# Compare the results: confusion matrix
caret::confusionMatrix(data = y_knn,
                       reference = test_set$type,
                       positive = "red")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction red white
##      red   145    15
##      white  15   475
##
##                Accuracy : 0.954
##                  95% CI : (0.935, 0.969)
##     No Information Rate : 0.754
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.876
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.906
##             Specificity : 0.969
##          Pos Pred Value : 0.906
##          Neg Pred Value : 0.969
##              Prevalence : 0.246
##          Detection Rate : 0.223
##    Detection Prevalence : 0.246
##       Balanced Accuracy : 0.938
##
##        'Positive' Class : red
##
```

```
# F1 score
F_meas(data = y_knn, reference = test_set$type)
```

```
## [1] 0.906
```

3.4 Regression tree - rpart

Regression tree uses all features to predict wine type.

```
# The "rpart" package trains regression trees and
# "rpart.plot" plots the tree
load_lib(c("rpart", "rpart.plot"))
```

```
# Train the model
fit_rpart <- rpart::rpart(formula = fml,
                          method = "class",
                          data = train_set)
# Predict
y_rpart <- predict(object = fit_rpart,
                   newdata = test_set,
                   type = "class")

# Compare the results: confusion matrix
caret::confusionMatrix(data = y_rpart,
                       reference = test_set$type,
                       positive = "red")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction red white
##      red   154     2
##      white   6   488
##
```

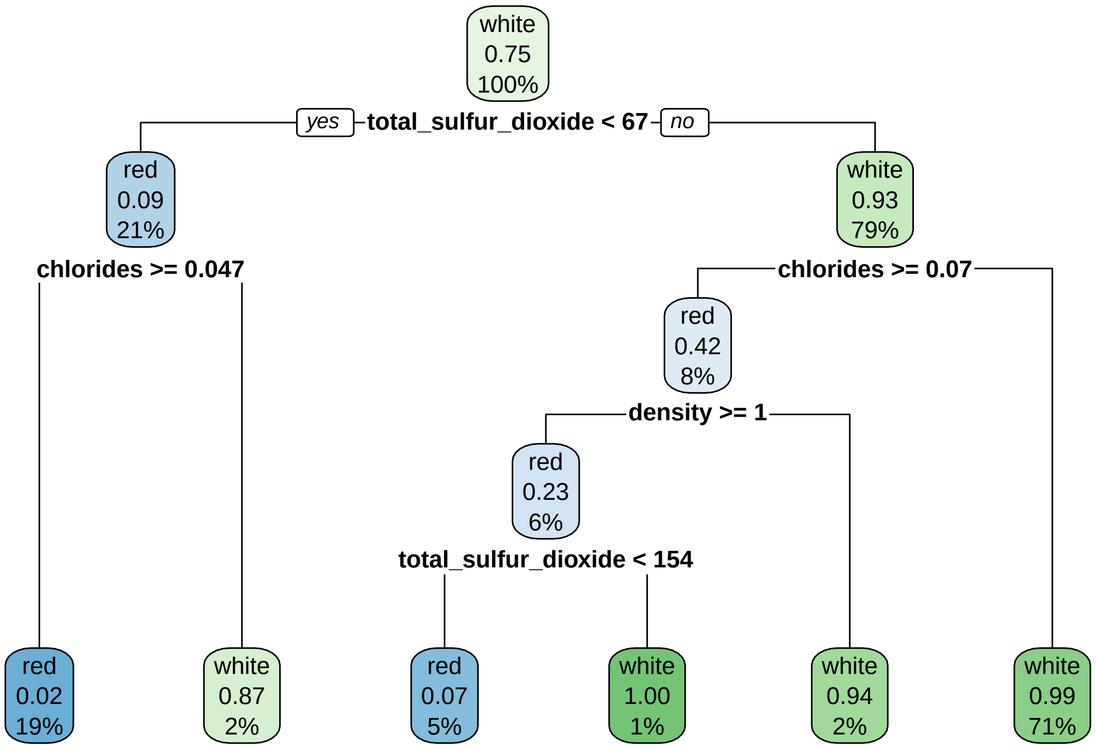
```
##                Accuracy : 0.988
##                  95% CI : (0.976, 0.995)
##     No Information Rate : 0.754
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.967
##
##  Mcnemar's Test P-Value : 0.289
##
##             Sensitivity : 0.963
##             Specificity : 0.996
##          Pos Pred Value : 0.987
##          Neg Pred Value : 0.988
##              Prevalence : 0.246
##          Detection Rate : 0.237
##    Detection Prevalence : 0.240
##       Balanced Accuracy : 0.979
##
##        'Positive' Class : red
##
```

The resulting tree represents the rule to predict wine type. The intensity of a node's color is proportional to the value predicted at the node.

Each node shows:

- the predicted class (red or white),
- the predicted probability of that class,
- the percentage of observations in the node.

```
# Plot the result
rpart.plot(fit_rpart)
```



```
# F1 score
F_meas(data = y_rpart, reference = test_set$type)
```

```
## [1] 0.975
```

Chlorides, total sulfur dioxide and volatile acidity are the three most important variables. Alcohol and citric acid have no importance at all.

```
# Variable importance
caret::varImp(fit_rpart)
```

```
##                       Overall
## chlorides                1773
## density                   259
## fixed_acidity             186
## free_sulfur_dioxide       548
## pH                         39
## residual_sugar            139
## sulphates                 615
## total_sulfur_dioxide     1595
## volatile_acidity        1235
## citric_acid                0
## alcohol                    0
```

3.5 Random Forest

Random forest achieves very high sensitivity and specificity.

```r
# The "randomForest" package trains classification and regression
# with Random Forest
load_lib("randomForest")
```

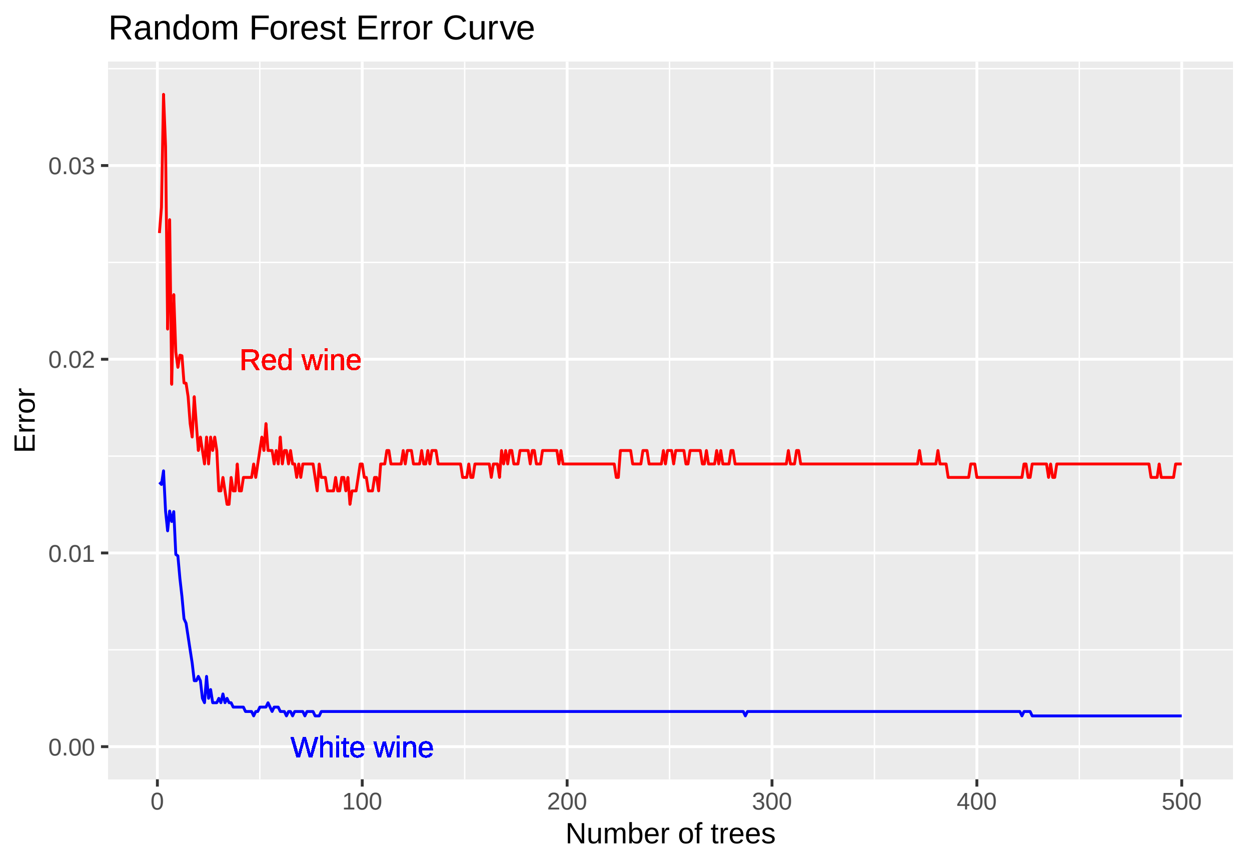
```r
# Train the model
fit_rf <- randomForest(formula = fml, data = train_set)

# Predict
y_rf <- predict(object = fit_rf, newdata = test_set)

# Compare the results: confusion matrix
caret::confusionMatrix(data = y_rf,
                       reference = test_set$type,
                       positive = "red")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction red white
##      red   158     0
##      white   2   490
##
##                Accuracy : 0.997
##                  95% CI : (0.989, 1)
##     No Information Rate : 0.754
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.992
##
##  Mcnemar's Test P-Value : 0.48
##
##             Sensitivity : 0.988
##             Specificity : 1.000
##          Pos Pred Value : 1.000
##          Neg Pred Value : 0.996
##              Prevalence : 0.246
##          Detection Rate : 0.243
##    Detection Prevalence : 0.243
##       Balanced Accuracy : 0.994
##
##        'Positive' Class : red
##
```

```
# F1 score
F_meas(data = y_rf, reference = test_set$type)
```

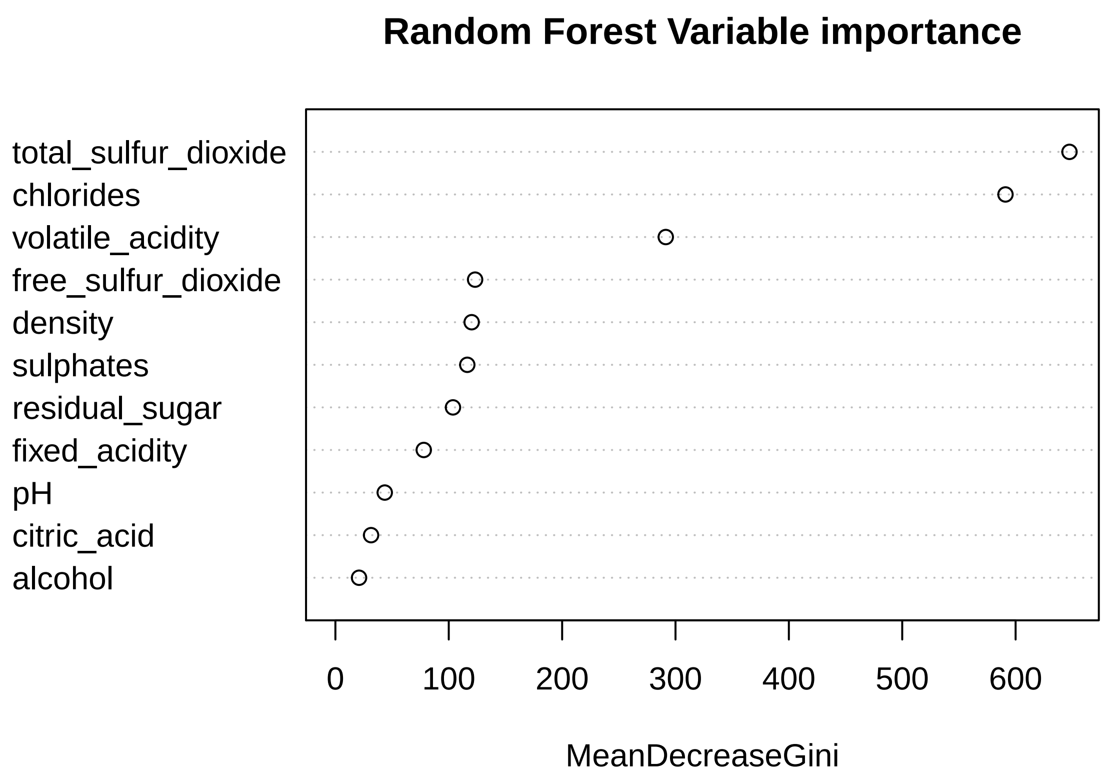
```
## [1] 0.994
```

The error decreases as the number of trees grows, stabilizing around 50 trees. The error for red wines is higher than white wines maybe because of the lower prevalence.



Total sulfur dioxide, chlorides and volatile acidity are the three most important variables for random forest prediction. Alcohol, citric acid and pH have low predictive power.

```r
# Variable importance plot
varImpPlot(fit_rf, main = "Random Forest Variable importance")
```



3.6 Linear Discriminant Analysis - LDA

LDA uses all features to predict wine type.

```r
load_lib("MASS")
```

```r
# Train the model
fit_lda <- lda(formula = fml, data = train_set)

# Predict
y_lda <- predict(object = fit_lda, newdata = test_set)

# Compare the results: confusion matrix
caret::confusionMatrix(data = y_lda[[1]],
                       reference = test_set$type,
                       positive = "red")
```

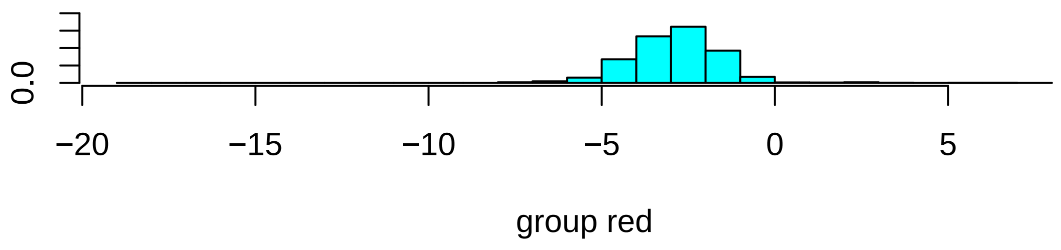
```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction red white
##      red    158     0
##      white    2   490
##
##                Accuracy : 0.997
##                  95% CI : (0.989, 1)
##     No Information Rate : 0.754
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.992
##
##  Mcnemar's Test P-Value : 0.48
##
##             Sensitivity : 0.988
##             Specificity : 1.000
##          Pos Pred Value : 1.000
##          Neg Pred Value : 0.996
##              Prevalence : 0.246
##          Detection Rate : 0.243
##    Detection Prevalence : 0.243
##       Balanced Accuracy : 0.994
##
##        'Positive' Class : red
##
```

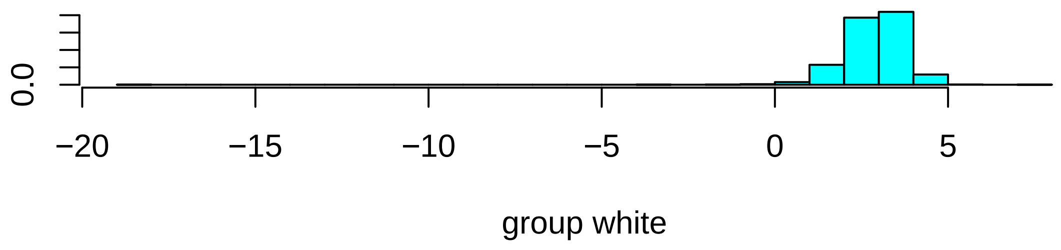
```r
# F1 score
F_meas(data = y_lda[[1]], reference = test_set$type)
```

```
## [1] 0.994
```

LDA is able to create clear distinct groups for red and white wines.

```r
# Plot the result
plot(fit_lda)
```





3.7 Quadratic Discriminant Analysis - QDA

QDA uses all features to predict wine type.

```r
load_lib(c("MASS", "scales"))
```

```r
# Train the model
fit_qda <- qda(formula = fml, data = train_set)

# Predict
y_qda <- predict(object = fit_qda, newdata = test_set)

# Compare the results: confusion matrix
caret::confusionMatrix(data = y_qda[[1]],
                       reference = test_set$type,
                       positive = "red")
```

```
## Confusion Matrix and Statistics
```

```
##
##            Reference
## Prediction red white
##      red   158     4
##      white   2   486
##
##                Accuracy : 0.991
##                  95% CI : (0.98, 0.997)
##     No Information Rate : 0.754
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.975
##
##  Mcnemar's Test P-Value : 0.683
##
##             Sensitivity : 0.988
##             Specificity : 0.992
##          Pos Pred Value : 0.975
##          Neg Pred Value : 0.996
##              Prevalence : 0.246
##          Detection Rate : 0.243
##    Detection Prevalence : 0.249
##       Balanced Accuracy : 0.990
##
##        'Positive' Class : red
##
```

```r
# F1 score
F_meas(data = y_qda[[1]], reference = test_set$type)
```

```
## [1] 0.981
```

3.8 Prediction results

Random forest and LDA have best performance. Although the single predictor model and linear regression models used just three features as predictors, they have better performance than knn that used all features.

```
##               Model Accuracy Sensitivity Specificity
## 1  Single predictor    96.9%       89.4%       99.4%
## 2 Linear Regression    97.5%       94.4%       98.6%
## 3               Knn    95.4%       90.6%       96.9%
## 4  Regression trees     98.8%         96.2%        99.6%
## 5      Random forest     99.7%         98.8%       100.0%
## 6                LDA     99.7%         98.8%       100.0%
## 7                QDA     99.1%         98.8%        99.2%
```

3.9 Cross Validation and Ensemble

All models used up to now run on a single partition of the training set. The result is near perfect for identifying wine type. Now we apply cross validation on 11 classification algorithms and then combine the results.

```r
# Now we are going to do several things:
# 1. train 10 classification models,
# 2. make the predictions for each model
# 3. calculate some statistics and store in the 'results' table
# 4. plot the ROC and precision-recall curves
# Then, we're going to plot the values in the 'results' table
# and make the ensemble of all models together.

# Load the packages used in this section
# Package "pROC" creates ROC and precision-recall plots
load_lib(c("pROC", "plotROC"))

# Several machine learning libraries
load_lib(c("e1071", "dplyr", "fastAdaboost", "gam",
           "gbm", "import", "kernlab", "kknn", "klaR",
           "MASS", "mboost", "mgcv", "monmlp", "naivebayes", "nnet", "plyr",
           "ranger", "randomForest", "Rborist", "RSNNS", "wsrf"))


# Define models
models <- c("glm", "lda", "naive_bayes", "svmLinear", "rpart",
            "knn", "gamLoess", "multinom", "qda", "rf", "adaboost")

# We run cross validation in 10 folds, training with 90% of the data.
# We save the prediction to calculate the ROC and precision-recall curves
# and we use twoClassSummary to compute the sensitivity, specificity and
# area under the ROC curve
control <- trainControl(method = "cv", number = 10, p = .9,
                        summaryFunction = twoClassSummary,
                        classProbs = TRUE,
                        savePredictions = TRUE)
```

```
control <- trainControl(method = "cv", number = 10, p = .9,
                        classProbs = TRUE,
                        savePredictions = TRUE)

# Create 'results' table. The first row
# contains NAs and will be removed after
# the training
results <- tibble(Model = NA,
                  Accuracy = NA,
                  Sensitivity = NA,
                  Specificity = NA,
                  F1_Score = NA,
                  AUC = NA)
#-------------------------------
# Start parallel processing
#-------------------------------
# The 'train' function in the 'caret' package allows the use of
# parallel processing. Here we enable this before training the models.
# See this link for details:
# http://topepo.github.io/caret/parallel-processing.html
cores <- 4    # Number of CPU cores to use
# Load 'doParallel' package for parallel processing
load_lib("doParallel")
cl <- makePSOCKcluster(cores)
registerDoParallel(cl)
```

3.9.1 Train the models

Here, we train the models, make predictions, calculate stats and store in 'results' table. We use standard tuning parameters for all models except knn and random forest.

```
set.seed(1234, sample.kind = "Rounding")
# Formula used in predictions
fml <- as.formula(paste("type", "~",
                        paste(xcol, collapse=' + ')))

# Run predictions
preds <- sapply(models, function(model){

  if (model == "knn") {
    # knn use custom tuning parameters
    grid <- data.frame(k = seq(3, 50, 2))
```

```r
  fit <- caret::train(form = fml,
                      method = model,
                      data = train_set,
                      trControl = control,
                      tuneGrid = grid)
} else if (model == "rf") {
  # Random forest use custom tuning parameters
  grid <- data.frame(mtry = c(1, 2, 3, 4, 5, 10, 25, 50, 100))

  fit <- caret::train(form = fml,
                      method = "rf",
                      data = train_set,
                      trControl = control,
                      ntree = 150,
                      tuneGrid = grid,
                      nSamp = 5000)
} else {
  # Other models use standard parameters (no tuning)
  fit <- caret::train(form = fml,
                      method = model,
                      data = train_set,
                      trControl = control)
}

# Predictions
pred <- predict(object = fit, newdata = test_set)

# Accuracy
acc <- mean(pred == test_set$type)

# Sensitivity
sen <- sensitivity(data = pred,
                   reference = test_set$type,
                   positive = "red")
# Specificity
spe <- specificity(data = pred,
                   reference = test_set$type,
                   positive = "red")

# F1 score
f1 <- F_meas(data = factor(pred), reference = test_set$type)

# AUC
auc_val <- auc(fit$pred$obs, fit$pred$red)
```

```
  # Store stats in 'results' table
  results <<- rbind(results,
                    tibble(
                      Model = model,
                      Accuracy = acc,
                      Sensitivity = sen,
                      Specificity = spe,
                      AUC = auc_val,
                      F1_Score = f1))

  # The predictions will be used for ensemble
  return(pred)
})

# Remove the first row of 'results' that contains NAs
results <- results[2:(nrow(results)),]
```

Next, we combine the best results of all models in a single ensemble. The next table summarizes the results.

```
#-------------------------------
# Combine all models
#-------------------------------
# Use votes method to ensemble the predictions
votes <- rowMeans(preds == "red")
y_hat <- factor(ifelse(votes > 0.5, "red", "white"))

# Update the 'results' table
results <<- rbind(results,
                  tibble(
                    Model = "Ensemble",
                    Accuracy = mean(y_hat == test_set$type),
                    Sensitivity = sensitivity(y_hat, test_set$type),
                    Specificity = specificity(y_hat, test_set$type),
                    AUC = auc(y_hat, as.numeric(test_set$type)),
                    F1_Score = F_meas(y_hat, test_set$type)))

as_hux(results,
    add_colnames = TRUE) %>%
  # Format header row
  set_bold(row = 1, everywhere, value = TRUE)          %>%
  set_top_border(row = 1, everywhere, value = 1)       %>%
  set_bottom_border(row = c(1,13), everywhere, value = 1)    %>%
  # Format numbers
```

```r
  set_number_format(row = -1, col = 2:6, value = 3)  %>%
  # Format alignment
  set_align(row = everywhere, col = 1,   value = 'left')  %>%
  set_align(row = everywhere, col = 2:6, value = 'right') %>%
  # Title
  set_caption('Model Performance With Cross Validation') %>%
  set_position(value = "center")
```

Table 3.2: Model Performance With Cross Validation

Model	Accuracy	Sensitivity	Specificity	F1_Score	AUC
glm	0.997	0.988	1.000	0.994	0.996
lda	0.997	0.988	1.000	0.994	0.996
naive_bayes	0.991	0.981	0.994	0.981	0.992
svmLinear	0.997	0.988	1.000	0.994	0.996
rpart	0.965	0.975	0.961	0.931	0.934
knn	0.954	0.906	0.969	0.906	0.967
gamLoess	0.995	0.988	0.998	0.991	0.997
multinom	0.992	0.988	0.994	0.984	0.995
qda	0.991	0.988	0.992	0.981	0.993
rf	0.997	0.988	1.000	0.994	0.999
adaboost	0.994	0.975	1.000	0.987	0.889
Ensemble	0.997	0.988	1.000	0.994	0.998

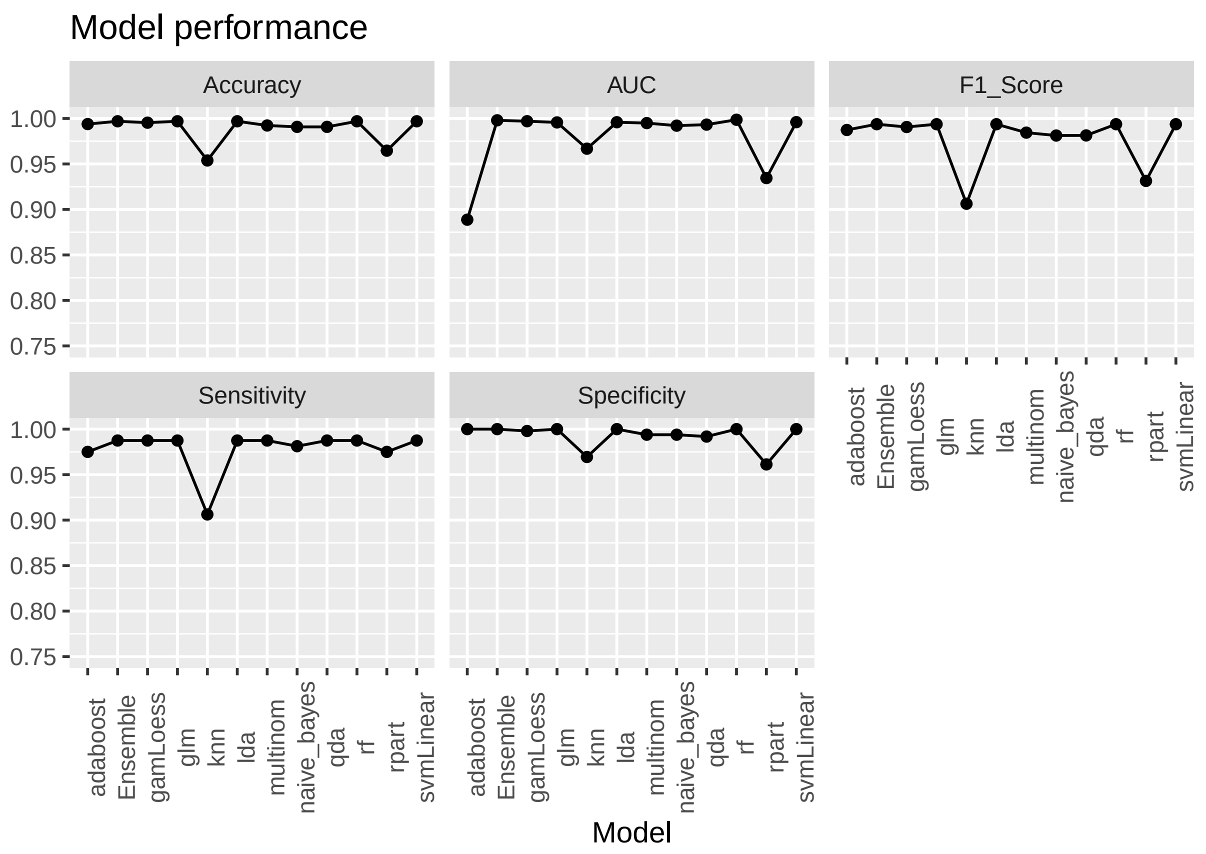
Generalized linear model (glm) has the best overall performance, better even than the ensemble.

```r
hux(Accuracy  = results[which.max(results$Accuracy),1]$Model,
    Sensitivity = results[which.max(results$Sensitivity),1]$Model,
    Specificity = results[which.max(results$Specificity),1]$Model,
    F_1    = results[which.max(results$F1_Score),1]$Model,
    AUC   = results[which.max(results$AUC),1]$Model,
    add_colnames = TRUE) %>%
    # Format header row
    set_bold(row = 1, col = everywhere, value = TRUE)         %>%
    set_top_border(row = 1, col = everywhere, value = 1)      %>%
    set_bottom_border(row = c(1,2), col = everywhere, value = 1)  %>%
    # Format table
    set_width(value = 0.6)                                    %>%
    set_caption("Best model")                                 %>%
    set_position(value = "center")
```

Table 3.3: Best model

Accuracy	Sensitivity	Specificity	F_1	AUC
glm	glm	glm	glm	rf

```r
#--------------------------------
# Plot the 'results' table
#--------------------------------
# We create a grid with all plots together.
# Each plot is simple Model vs Stats
results %>%
  # Convert columns to lines
  pivot_longer(cols = 2:6, names_to = "Metric", values_drop_na = TRUE) %>%
  ggplot(aes(x = Model, y = value, group = 1)) +
      geom_line() +
      geom_point() +
      # Y axis scale
      ylim(0.75, 1) +
      # Format labels
      ggtitle("Model performance") +
      ylab("") +
      theme(legend.position="none" ,
            axis.text.x = element_text(angle = 90)) +
      # Arrange in grid
    facet_wrap(~Metric)
```



```r
results
```

```
## # A tibble: 12 x 6
##    Model       Accuracy Sensitivity Specificity F1_Score   AUC
##    <chr>          <dbl>       <dbl>       <dbl>    <dbl> <dbl>
##  1 glm            0.997       0.988       1        0.994 0.996
##  2 lda            0.997       0.988       1        0.994 0.996
##  3 naive_bayes    0.991       0.981       0.994    0.981 0.992
##  4 svmLinear      0.997       0.988       1        0.994 0.996
##  5 rpart          0.965       0.975       0.961    0.931 0.934
##  6 knn            0.954       0.906       0.969    0.906 0.967
##  7 gamLoess       0.995       0.988       0.998    0.991 0.997
##  8 multinom       0.992       0.988       0.994    0.984 0.995
##  9 qda            0.991       0.988       0.992    0.981 0.993
## 10 rf             0.997       0.988       1        0.994 0.999
## 11 adaboost       0.994       0.975       1        0.987 0.889
## 12 Ensemble       0.997       0.988       1        0.994 0.998
```

3.10 Predicting Quality

The prevalence of quality levels 3, 4 and 8 is very low in the dataset. There are just 5 samples of quality level 9 in the red and white datasets combined, and no sample in the train_set_red dataset.

We previously combined the levels into the `low`, `medium` and `high` quality categories in the train_set_red dataset. In this section, we use cross validation with ensemble to predict the combined levels.

The models glm, gamLoess, qda and adaboost don't work in this case.

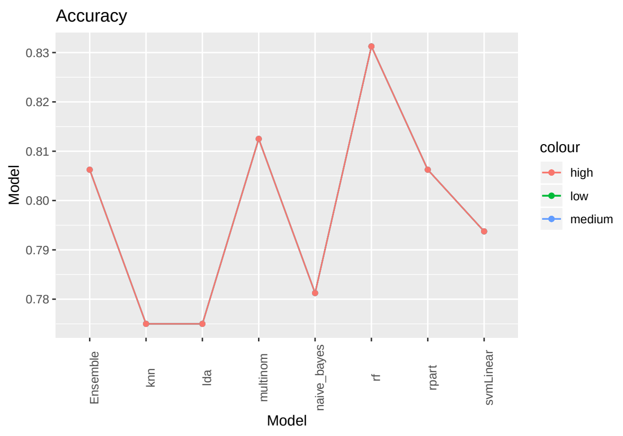
```r
set.seed(1234, sample.kind = "Rounding")
# Formula used in predictions
fml_qual <- as.formula(paste("quality2", "~",
                             paste(xcol, collapse=' + ')))

# Define models
#"glm",gamLoess, qda, adaboost
models <- c( "lda", "naive_bayes", "svmLinear", "rpart",
            "knn", "multinom", "rf")

# Create 'results' table. The first row
# contains NAs and will be removed after
# the training
quality_results <- tibble(Model = NA,
                          Quality = NA,
                          Accuracy = NA,
                          Sensitivity = NA,
                          Specificity = NA,
                          F1_Score = NA)

preds_qual <- sapply(models, function(model){

  print(model)
  if (model == "knn") {
    # knn use custom tuning parameters
    grid <- data.frame(k = seq(3, 50, 2))
    fit <- caret::train(form = fml_qual,
                        method = model,
                        data = train_set_red,
                        trControl = control,
                        tuneGrid = grid)
  } else if (model == "rf") {
    # Random forest use custom tuning parameters
    grid <- data.frame(mtry = c(1, 2, 3, 4, 5, 10, 25, 50, 100))

    fit <- caret::train(form = fml_qual,
                        method = "rf",
                        data = train_set_red,
                        trControl = control,
                        ntree = 150,
                        tuneGrid = grid,
                        nSamp = 5000)
  } else {
    # Other models use standard parameters (no tuning)
    fit <- caret::train(form = fml_qual,
                        method = model,
                        data = train_set_red,
                        trControl = control)
  }

  # Predictions
  pred <- predict(object = fit, newdata = test_set_red)

  # Accuracy
  acc <- mean(pred == test_set_red$quality2)

  # Sensitivity
  sen <- caret::confusionMatrix(pred,
                                test_set_red$quality2)$byClass[,"Sensitivity"]
  # Specificity
  spe <- caret::confusionMatrix(pred,
                                test_set_red$quality2)$byClass[,"Specificity"]

  # F1 score
  f1 <- caret::confusionMatrix(pred,
                               test_set_red$quality2)$byClass[,"F1"]

  # Store stats in 'results' table
  quality_results <<- rbind(quality_results,
                            tibble(Model = model,
                                   Quality = levels(test_set_red$quality2),
                                   Accuracy = acc,
                                   Sensitivity = sen,
                                   Specificity = spe,
                                   F1_Score = f1))

  # The predictions will be used for ensemble
```

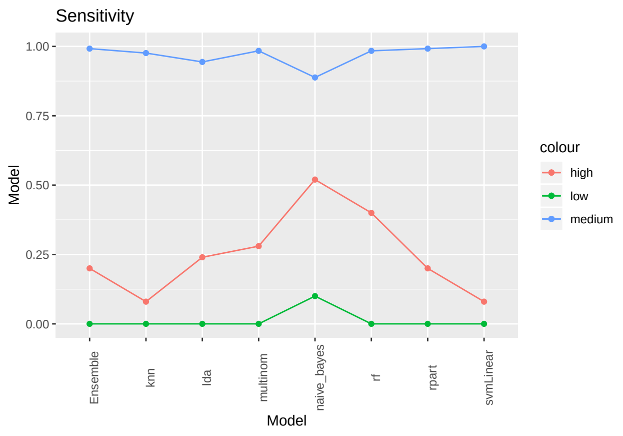
```
  return(pred)
})

# Remove the first row of 'results' that contains NAs
quality_results <- quality_results[2:(nrow(quality_results)),]
```

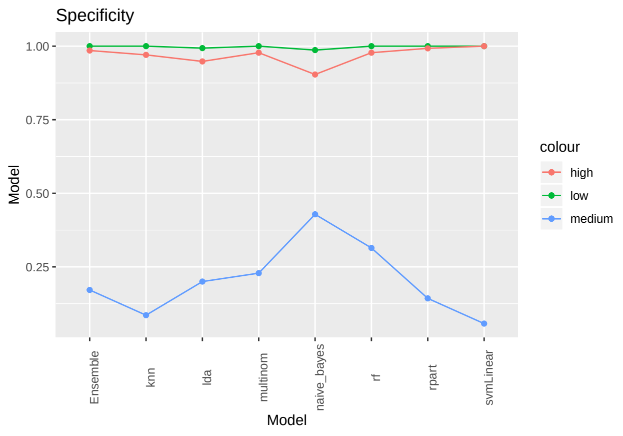
Next, we combine the best results of all models in a single ensemble. The next table summarizes the results.

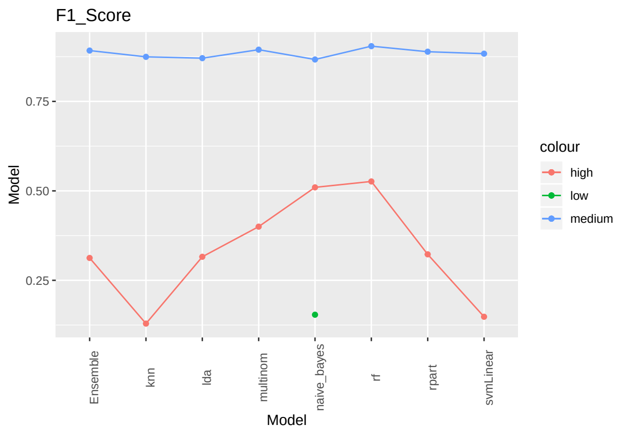
```
#--------------------------------
# Combine all models
#--------------------------------
# Use votes method to ensemble the predictions
votes <- data.frame(low    = rowSums(preds_qual =="low"),
                    medium = rowSums(preds_qual =="medium"),
                    high   = rowSums(preds_qual =="high"))

y_hat <- factor(sapply(1:nrow(votes), function(x)
  colnames(votes[which.max(votes[x,])])))

y_hat <- relevel(y_hat, "medium")

  # Accuracy
  acc <- caret::confusionMatrix(y_hat,
                              test_set_red$quality2)$overall["Accuracy"]

  # Sensitivity
  sen <- caret::confusionMatrix(y_hat,
                              test_set_red$quality2)$byClass[,"Sensitivity"]
  # Specificity
  spe <- caret::confusionMatrix(y_hat,
                              test_set_red$quality2)$byClass[,"Specificity"]

  # F1 score
  f1 <- caret::confusionMatrix(y_hat,
                             test_set_red$quality2)$byClass[,"F1"]


quality_results <<- rbind(quality_results,
                          tibble(Model = "Ensemble",
                                 Quality = levels(test_set_red$quality2),
                                 Accuracy = acc,
                                 Sensitivity = sen,
                                 Specificity = spe,
                                          F1_Score = f1))
```

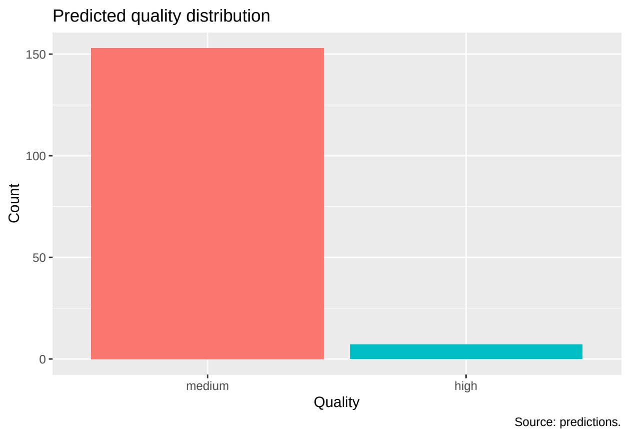
```
#--------------------------------
# Plot the 'results' table
#--------------------------------
# Make a plot for each metric with the 3 quality levels
# The plots are grouped using the chunk options.

# Metric names
metrics <- names(quality_results)[3:6]

res <- sapply(metrics, function(var){

  # Plot stored in 'p'
  p <- quality_results %>%
    # Convert columns to lines
    pivot_longer(cols = 3:6, names_to = "Metric",
                 values_drop_na = TRUE) %>%

    pivot_wider(names_from = Quality) %>%
    filter(Metric == var) %>%

    ggplot(aes(x = Model, group = 1)) +
        # Draw lines
        geom_line(aes(y = low,      col = "low")) +
        geom_line(aes(y = medium,   col = "medium")) +
        geom_line(aes(y = high,     col = "high")) +
        # Draw points
        geom_point(aes(y = low,     col = "low")) +
        geom_point(aes(y = medium, col = "medium")) +
        geom_point(aes(y = high,    col = "high")) +
        # Format labels
        ggtitle(var) +
        ylab("Model") +
        theme(axis.text.x = element_text(angle = 90))

    # Show the plot
    print(p)
})
```

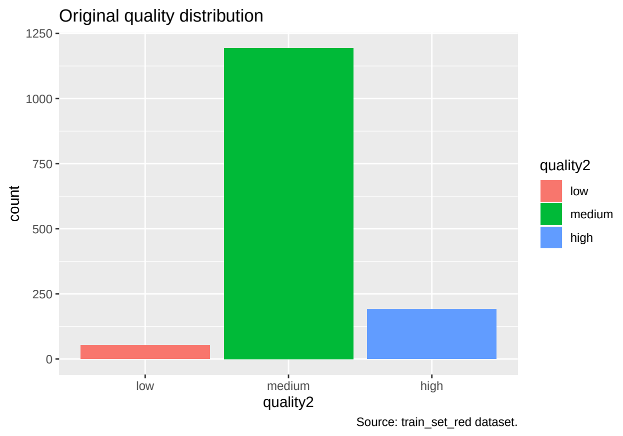












```r
#-------------------------------
# Stop parallel processing used in 'train'
#-------------------------------
stopCluster(cl)
```

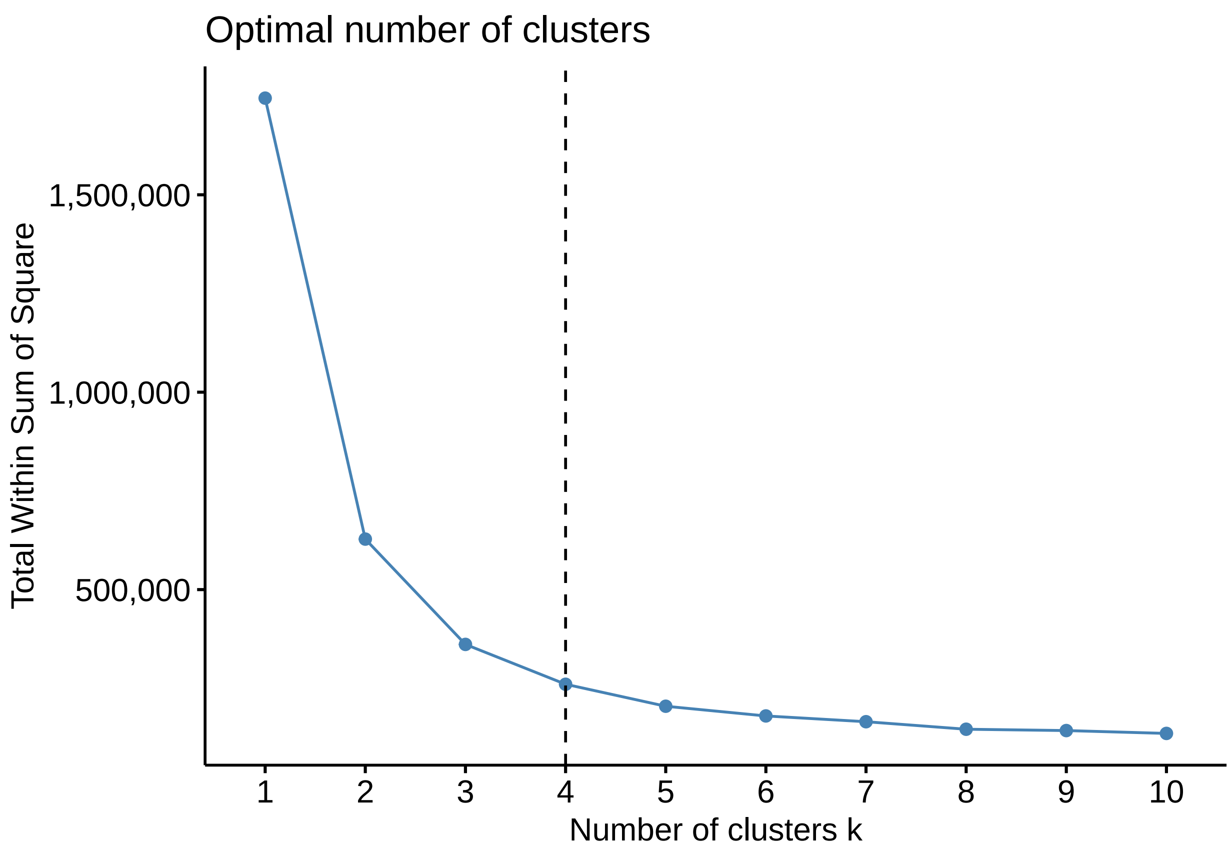
3.11 Clustering

In this section, we use unsupervised learning to identify possible clusters.

Partitioning methods, such as k-means clustering require the users to specify the number of clusters to be generated. The function `fviz_nbclust` from the `factoextra` package determines and visualize the optimal number of clusters using different methods: within cluster sums of squares, average silhouette and gap statistics.

The properties of red and white wines differ, so in this analysis we create clusters just for red wines.

```
#-------------------------------
# k-means
#-------------------------------
# Reference:
# https://www.datanovia.com/en/lessons/k-means-clustering-in-r-algorith-and-practical-

# Load 'factoextra' to visualize clusters
load_lib("factoextra")

# Determine and visualize the optimal number of clusters
# using total within sum of square (method = "wss")
train_set %>% filter(type == "red") %>% .[,xcol] %>%
  fviz_nbclust(x = ., FUNcluster = kmeans, method = "wss") +
  geom_vline(xintercept = 4, linetype =2) +
  scale_y_continuous(labels = comma)
```



The plot shows that the total sum of squares decreases slowly for more than 4 clusters. Selecting a larger value provides little improvement and increases the overlaps between the clusters.

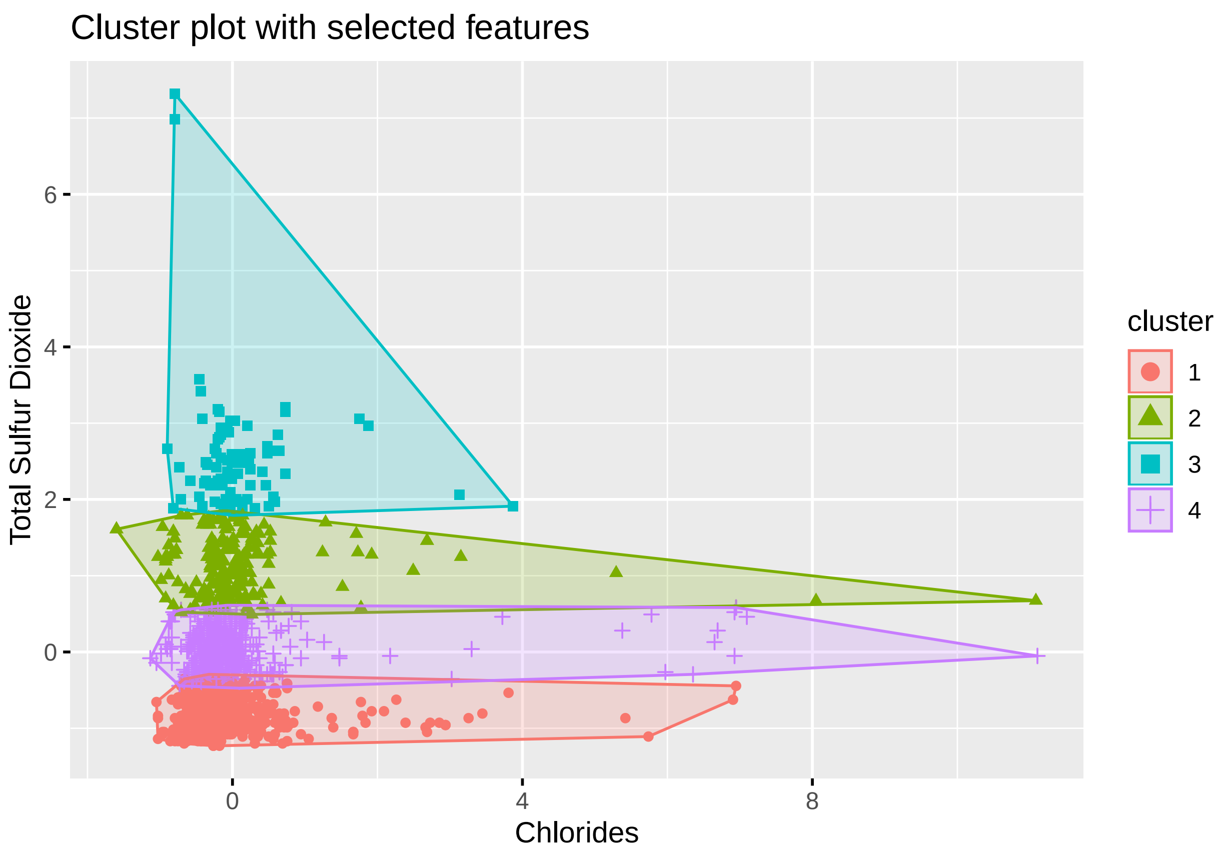
```r
# We use 25 random starts for the clusters
k <- train_set %>% filter(type == "red") %>% .[,xcol] %>%
      kmeans(x = ., centers = 4, nstart = 25)

# Calculate cluster means
cm <- as_hux(data.frame(t(k$centers)), add_rownames = TRUE)
colnames(cm) <- c("Feature", paste("Cluster", 1:4))
cm <- add_colnames(cm)
cm %>%
      # Format header row
      set_bold(row = 1, col = everywhere, value = TRUE)        %>%
      set_top_border(row = 1, col = everywhere, value = 1)     %>%
      set_bottom_border(row = c(1,12), col = everywhere, value = 1)  %>%
      # Format cells
      set_align(row = everywhere, col = 1,   value = 'left')   %>%
      set_align(row = everywhere, col = 2:5, value = 'right')  %>%
      set_number_format(row = 2:nrow(cm),
                        col = everywhere, value = 3)           %>%
      # Format table
      set_width(value = 0.7)                                       %>%
      set_position(value = "center")                               %>%
      set_caption("Cluster Center by Feature")
```

Table 3.4: Cluster Center by Feature

Feature	Cluster 1	Cluster 2	Cluster 3	Cluster 4
fixed_acidity	8.519	8.083	7.958	8.168
volatile_acidity	0.520	0.544	0.545	0.524
citric_acid	0.274	0.279	0.322	0.252
residual_sugar	2.403	2.901	3.465	2.374
chlorides	0.084	0.090	0.090	0.090
free_sulfur_dioxide	8.361	24.646	30.478	19.129
total_sulfur_dioxide	20.703	83.537	130.725	47.413
density	0.997	0.997	0.997	0.997
pH	3.306	3.319	3.234	3.330
sulphates	0.648	0.645	0.688	0.672
alcohol	10.598	10.137	9.853	10.410

The `fviz_cluster` function plots the clusters for the combination of two features. Here, we plot the clusters for total sulfur dioxide and chlorides.

```
# Plot the cluster
train_set %>% filter(type == "red") %>% .[,xcol] %>%
  fviz_cluster(object = k,
               choose.vars = c("chlorides", "total_sulfur_dioxide"),
               geom    = "point",
               repel   = TRUE,
               main    = "Cluster plot with selected features",
               xlab    = "Chlorides",
               ylab    = "Total Sulfur Dioxide")
```


Cluster plot with selected features
Total Sulfur Dioxide
Chlorides
cluster
1
2
3
4
6
4
2
0
0
4
8

Chapter 4

Conclusion

This report uses two datasets of vinho verde wine to predict the wine type, red or white, and the quality based on physicochemical properties. Quality is a subjective measure, given by the average grade of three experts.

Before starting the predictions, the report makes a brief summary of model evaluation, explaining the most common metrics used in categorical problems in machine learning.

In data preparation, the datasets are downloaded and imported in R. In this phase, the training and testing sets are created and they will be used during the model building.

In data exploration and visualization we look for features that may provide good prediction results. The best predictors have low distribution overlapping area and low correlation among them.

Modeling starts explaining very simple models and gradually moves to more complex ones. There's a brief explanation of the models used in this report. Other important machine learning concepts, such as ensemble and cross validation, are also discussed. Clustering is unsupervised method, which is included for illustrative purposes.

The results section presents the modeling results and discusses the model performance. The algorithms are used to predict wine type; there's a section for predicting quality and the last part demonstrates clustering.

The physicochemical properties of wine differ between red and white wines, but the difference is not so evident when evaluating red wine quality. Maybe there are other properties not considered in this dataset that are better indicators for quality.

4.1 Limitations

The machine learning models used in this research aren't able to predict red wine quality with high accuracy, specificity and sensitivity. This is partially explained by the low prevalence of quality levels 3, 4 and 8 and the large distribution overlapping area stratified by quality.

Besides this, the lack of information about how the dataset was created may impact the prediction of quality using the physicochemical properties as predictors. Such examples are the composition of grape varieties in each wine, the mix of experts that evaluated wine quality, or the production year.

References

Rafael A. Irizarry, (2020) - "Introduction to Data Science" - https://rafalab.github.io/dsbook/

Baptiste Auguié, (2019) - "Laying out multiple plots on a page" - https://cran.r-project.org/web/packages/egg/vignettes/Ecosystem.html

Yihui Xie, J. J. Allaire, Garrett Grolemund (2019) - "R Markdown: The Definitive Guide" - https://bookdown.org/yihui/rmarkdown/

Baptiste Auguie, (2017) - "Arranging multiple grobs on a page" https://cran.r-project.org/web/packages/gridExtra/vignettes/arrangeGrob.html

Max Kuhn, (2019) - "The caret Package" https://topepo.github.io/caret/model-training-and-tuning.html

ROC and AUC - https://stackoverflow.com/questions/31138751/roc-curve-from-training-data-in-caret

Xavier Robin et al, (2019), "Display and Analyze ROC Curves" - https://cran.r-project.org/web/packages/pROC/pROC.pdf

Sarang Narkhede. (2018) - "Understanding AUC - ROC Curve" https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

Alboukadel Kassambara, Fabian Mundt, (2019) - "Extract and Visualize the Results of Multivariate Data Analyses" - https://cran.r-project.org/web/packages/factoextra/factoextra.pdf

"K-Means Clustering in R: Algorithm and Practical Examples" https://www.datanovia.com/en/lessons/k-means-clustering-in-r-algorith-and-practical-examples/

Teru Watanabe, (2016), "Quality and Physicochemical Properties of White Wine" - https://rstudio-pubs-static.s3.amazonaws.com/152060_f4b5dad9dbd742a383ac3b8cee4e679c.html

Hadley Wickham, "The Split-Apply-Combine Strategy for Data Analysis" https://vita.had.co.nz/papers/plyr.pdf

https://stackoverflow.com/questions/57381627/add-text-labels-at-the-top-of-density-plots-to-label-the-different-groups

"What is Wine?" https://waterhouse.ucdavis.edu/tags/what-wine?page=1

"Cluster Analysis in R" http://girke.bioinformatics.ucr.edu/GEN242/pages/mydoc/Rclustering.html#1_introduction