THE *“WINE QUALITY”* DATASET

Why we chose it
We were fascinated by the idea of being able to objectively predict something as subjective as the Portuguese *Vinho Verde* wine quality, especially without having any significant knowledge in the field, obtaining results comparable to the ones of an expert.

DATA DESCRIPTION

For privacy and logistics issues, the dataset only contains physiochemical variables and a sensory variable, which are all numerical:

```
str(df)
```

```
## '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 0.045 0.049 0.044 ...
##  $ free.sulfur.dioxide : num  45 14 30 47 47 30 30 45 14 28 ...
##  $ total.sulfur.dioxide: num  170 132 97 186 186 97 136 170 132 129 ...
##  $ density             : num  1.001 0.994 0.995 0.996 0.996 ...
##  $ pH                  : num  3 3.3 3.26 3.19 3.19 3.26 3.18 3 3.3 3.22 ...
##  $ sulphates           : num  0.45 0.49 0.44 0.4 0.4 0.44 0.47 0.45 0.49 0.45 ...
##  $ alcohol             : num  8.8 9.5 10.1 9.9 9.9 10.1 9.6 8.8 9.5 11 ...
##  $ quality             : int  6 6 6 6 6 6 6 6 6 6 ...
```

The most significant insight we can derive by looking at the data is that only the response is discrete, whilst all the predictors are continuous.

Some summarizing statistics about the data:

```
summary(df)
```

```
##  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
```

Having seen that the variables differ significantly both in range magnitude and values, we will later scale them to avoid them contributing unequally to the analysis and creating a bias.

Data Cleaning

Before exploring and visualizing the data, we proceed to clean it, so as to obtain a clearer analysis.
Removing Duplicates
Otherwise, the model would come out distorted (it would be more biased): the wines occurring more than once would influence the model more than they should.

```
df<-unique(df)
dim(df)
```

```
## [1] 3961   12
```

Since the instances went from 4898 to 3961 after removing the duplicates, we can deduce there were 937 duplicates.
We were able to remove such duplicate rows because the instances had no ID; therefore, two wines with the same characteristics are likely to be the same wine, rather than being two distinct wines with the same characteristics.

Checking for `NULL` values

To make sure the model works properly, we must remove the `NULL` values: otherwise, in fact, missing data can cause biased estimates and invalid conclusions.

```
is.null(df)
```

```
## [1] FALSE
```

The `is.null()` function checks for `NULL` values: if it returns `FALSE`, it indicates that there are no missing values.

Checking for Unary Values

Defining a function to detect non-unary columns in the dataframe:

```
Nonunary <- function(x) length(unique(x))>1
```

Applying it to our dataframe:

```
df[sapply(df,Nonunary)]
```

```
dim(df)
```

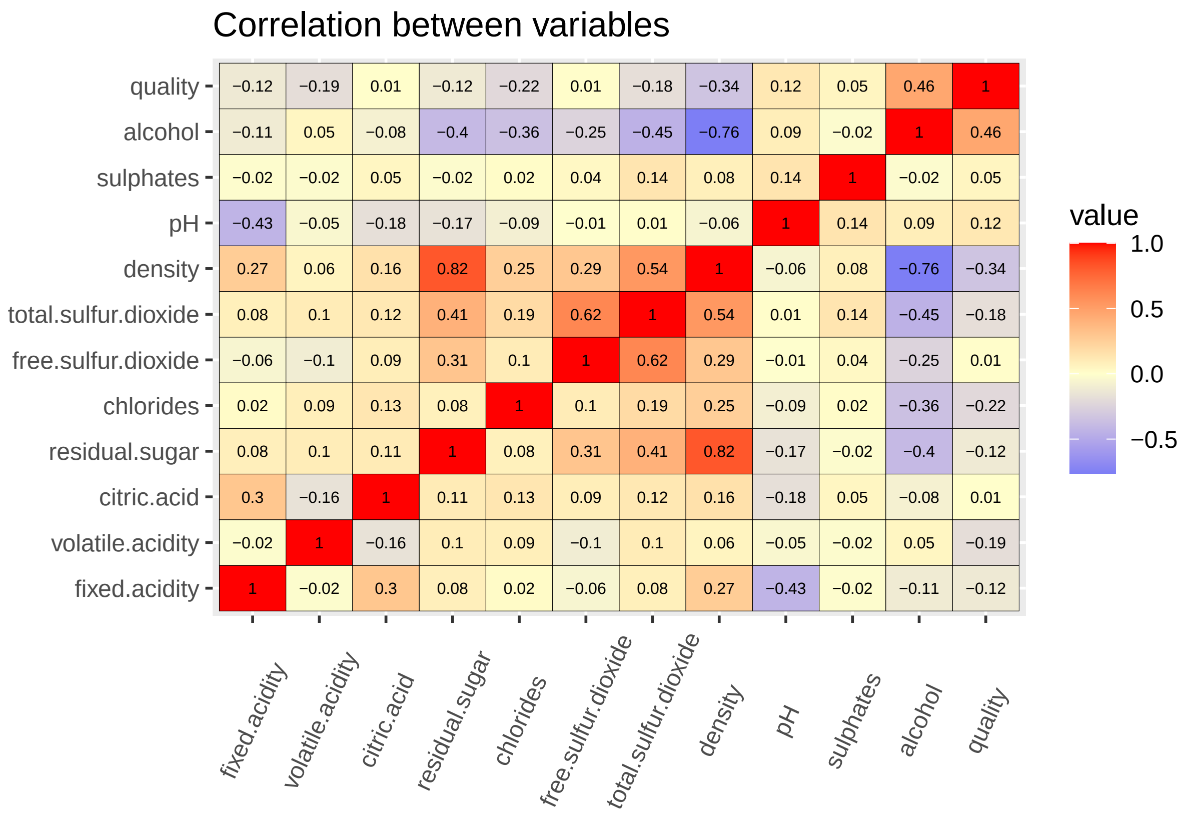
```
## [1] 3961   12
```

The function returns a filtered dataframe with the same number of attributes as before, so there are no unary columns.

Data Exploration and Visualization

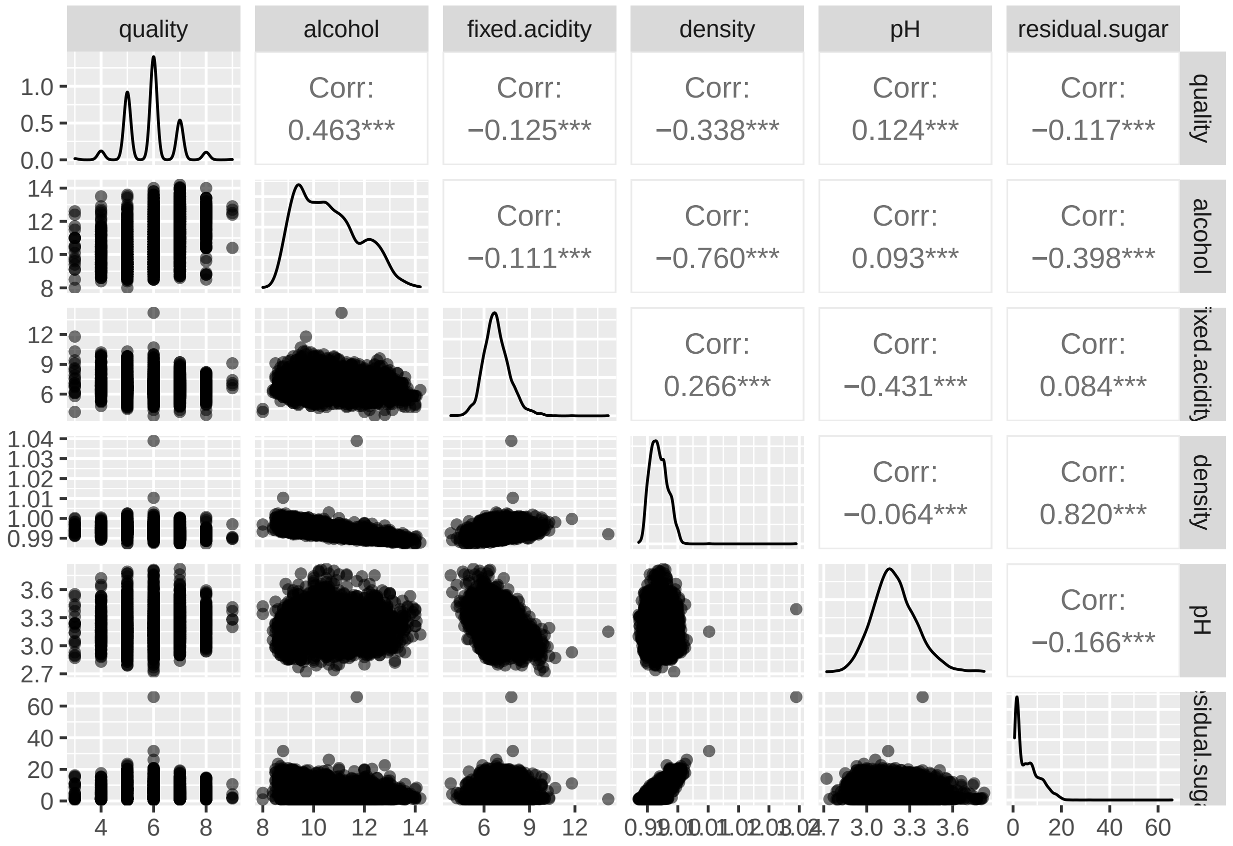
Correlation matrix between all variables and response

In order to see how they relate to each other and to the response. Correlation values range from -1 to 1; a negative value for correlation means the variables are oppositely correlated, whilst a positive one means they are linearly correlated. Generally, a correlation smaller than 0.2 in absolute value is considered weak because, as we get closer to zero, correlation is less and less present.



Correlation plot between the 5 most correlated variables and the response variable (quality)

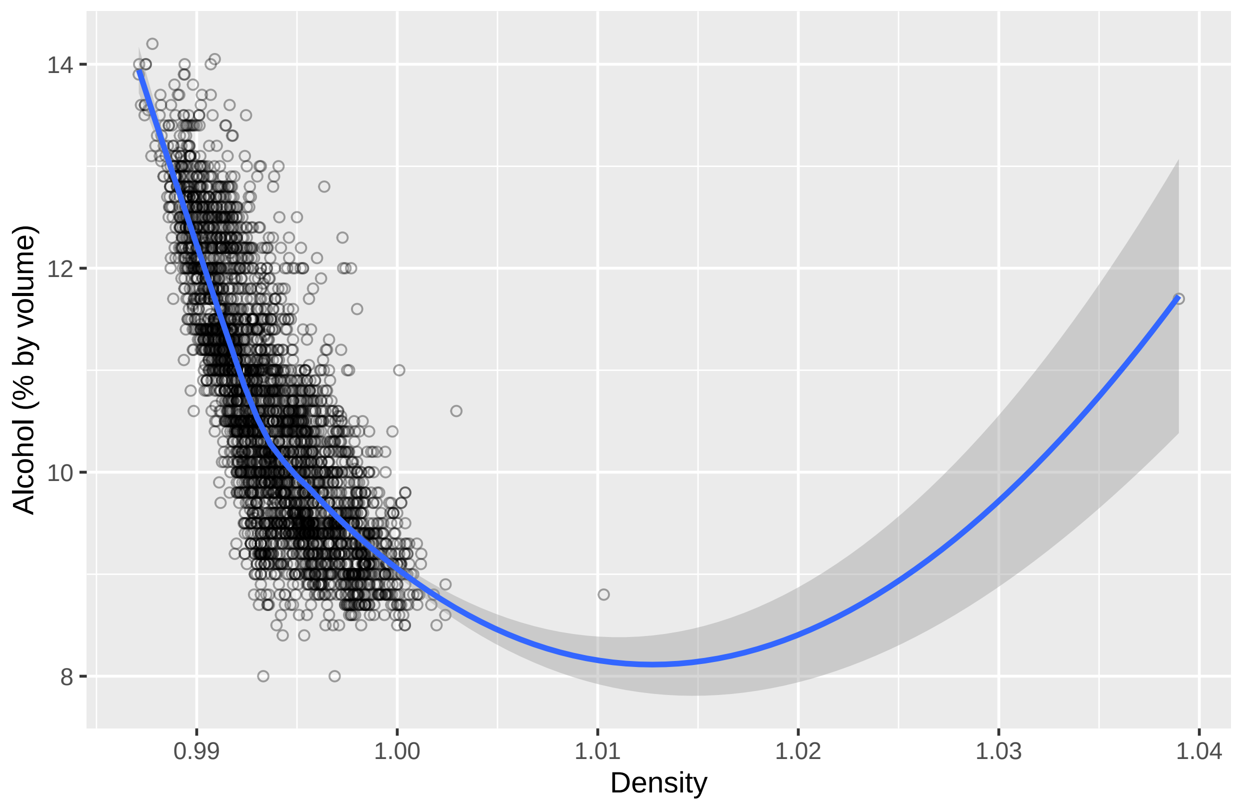
```
varsubset<-c('quality', 'alcohol','fixed.acidity',
             'density', 'pH','residual.sugar')
```



The variables correlated the most with the response `quality` appear to be `alcohol` (positively) and `density` (negatively).

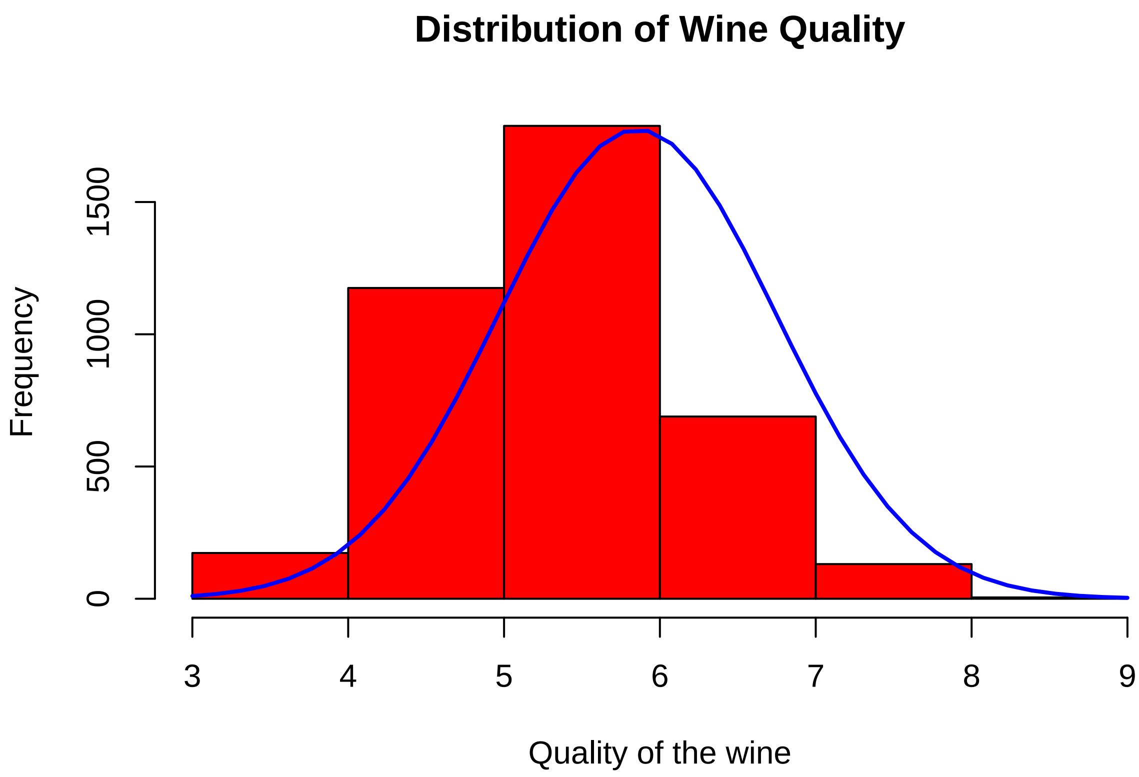
Based on the previous results, we wanted to further investigate the relationship between `alcohol` and `density`, the two variables correlated the most with the response:

Relationship of Alcohol Vs. Density



The plot shows a negative linear relation between `alcohol` and `density`. Out interpretation of this is that, since alcohol is less dense than water, a larger quantity of alcohol implies less density of the liquid overall. Thus, alcohol is negatively correlated with density.

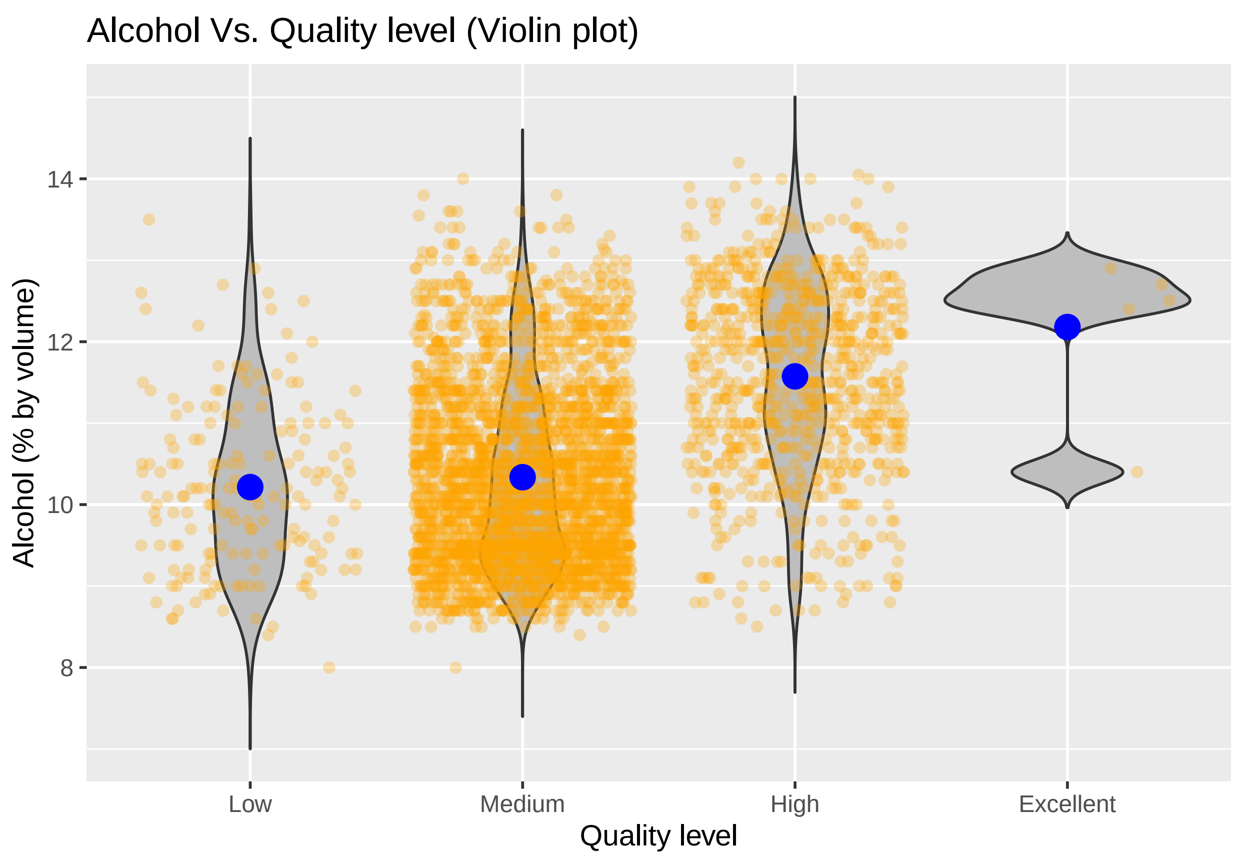
Distribution of the target variable



The frequency distribution appears to resemble a Gaussian distribution, but with a slight skew on the left; hence, there is not really a need to transform it with a log function. Otherwise, we would have had to do so because some of the functions through which we will later fit the models work with a response whose distribution is approximately Gaussian.

Violin Plot of wine `quality` level vs `alcohol` level
Shows the distribution of instances with respect to quality levels, with their respective alcohol percentage by volume on the vertical axis.
We create four tiers of wine-quality levels as a new column of the dataframe:
- *From a quality of 2 to 4: 'Low';*
- *From 4 to 6: 'Medium';*
- *From 6 to 8: 'High';*
- *From 9 on: 'Excellent'.*

```
dfplot<-df
dfplot$quality.levels <- cut(df$quality,c(2,4,6,8,10),
                         labels = c('Low','Medium', 'High','Excellent'))
```



1) Most of the wines in the dataset lie in the two middle quality levels (`Medium` and `High`), but skewed to the left;

2) Alcohol and quality levels do confirm to be positively correlated: in fact, the average alcohol level increases with the quality level;

3) The distribution of the `Excellent` wines appears to be composed of two main quantiles: one higher than all the other quality levels and the other about in the middle. However, the middle one is only composed by one instance; hence, it is more of a special case, rather than a category by itself;

4) Since the alcohol level between the `Low` and `Medium` quality level is similar, the quality difference might be caused by other factors, rather than by the alcohol level.

TASK 1

As mentioned at the beginning, we decided to perform the task of predicting the quality of the wine according to its specifics because we were fascinated by the idea of being able to objectively predict something as subjective as this, especially without having any significant knowledge in the field, obtaining results comparable to the ones of an expert.
Moreover, the dataset seemed to suit itself to this task the most compared to others that could have been done on it.
Finally, such an analysis could potentially have significant commercial and research implications, both for

the *Vinho Verde* company and for its competitors, hence it highly resembles similar tasks we might be asked to perform in our future job.

Modelling

We will be predicting the target variable `Quality` as continuous, even though its observations are discrete.

Train-test Split:
We divide the original dataset into a training set (used to fit the model) and a test set (used to evaluate its performance):

```
training <- df$quality %>%
  createDataPartition(p = 0.75, list = FALSE)
trainset  <- df[training, ]
xtrain<-trainset[,-12]
ytrain<-trainset[,12]
testset<- df[-training, ]
xtest<-trainset[,-12]
ytest<-trainset[,12]
```

We separated the predictors from the response in order to only scale the predictors.

Scaling the variables:
As anticipated earlier, we now scale the data in order to be able to easily analyze and compare the scores:

```
xtrain <- xtrain %>% scale()
xtest <- xtest %>% scale(center=attr(xtrain, "scaled:center"),
                         scale=attr(xtrain, "scaled:scale"))
```

We scaled both the training and test sets so that the model fit on the training set works the properly also on the test set. They are both scaled based on the training set, in fact.

Baseline Model

Before fitting more complicated models, we start by fitting a simple linear-regression model on the training data only with the variable (`alcohol`) correlated the most with the response. Our aim will then be, from this initial model, to gradually improve the model to reduce the error.

We will be evaluating such error using the MSE measure because it tells us how close our regression line is to our data points.

```
base <- lm(quality~ alcohol, data = trainset)
summary(base)
```

```
##
## Call:
```

```
## lm(formula = quality ~ alcohol, data = trainset)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.5415 -0.4825  0.0051  0.5175  2.7566
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.23710    0.12937   17.29   <2e-16 ***
## alcohol      0.34162    0.01215   28.12   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8009 on 2969 degrees of freedom
## Multiple R-squared:  0.2103, Adjusted R-squared:  0.2101
## F-statistic: 790.8 on 1 and 2969 DF,  p-value: < 2.2e-16
```

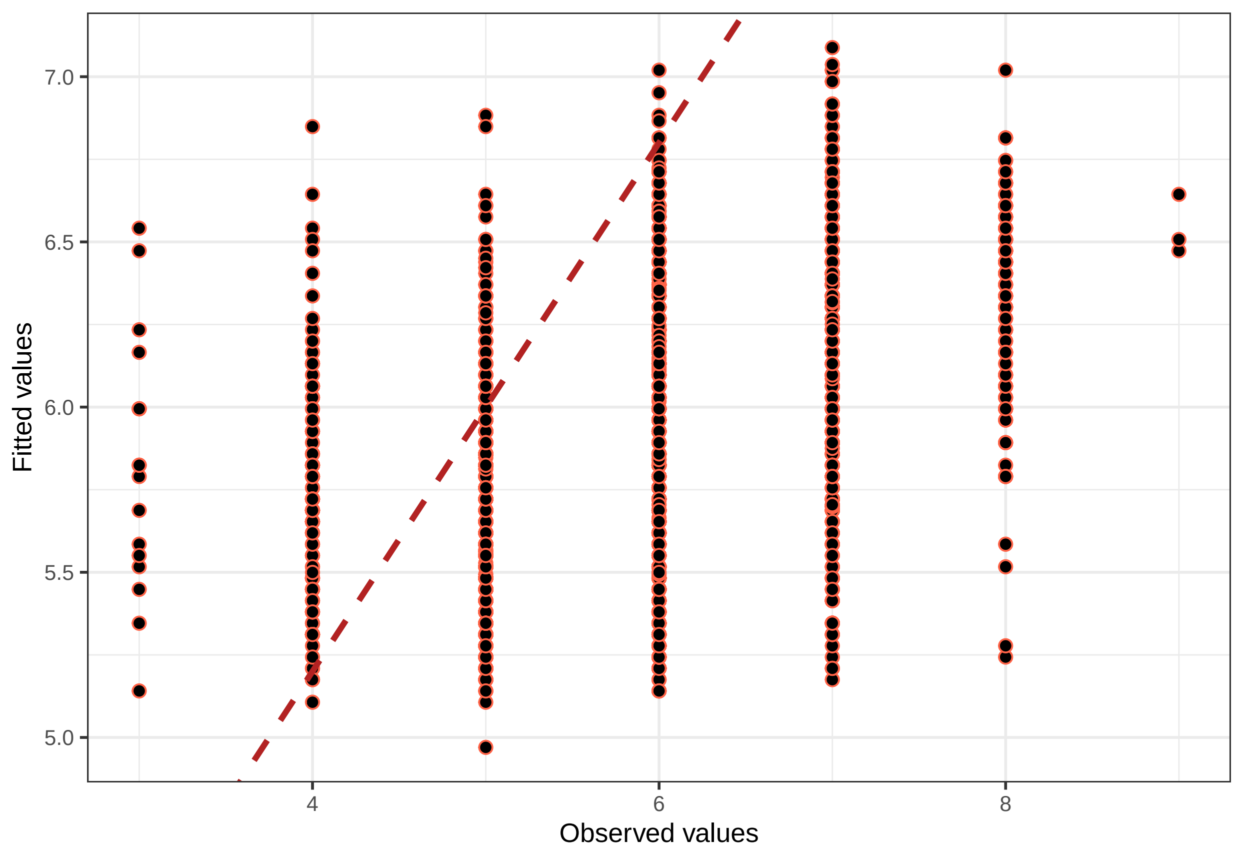
```r
preds<-predict(base,testset[,-12])
mean((testset$quality-preds)^2)
```

```
## [1] 0.5698962
```

MSE = 0.570: terrible, but alcohol seems to be statistically significant (p-value < 0.05), so we will keep it also in the next models.

Baseline: goodness of fit

The following graph shows the fit regression line with respect to the observed values of the response (quality) and to the fitted ones:



What we obtain is clearly a poor fit, but that is mainly because, by using a linear regression, we are trying to predict a continuous outcome for a response that is actually discrete.

Full model (all input variables)
We now try to fit a multiple linear-regression model using all input variables:

```
lr <- lm(quality~., data = trainset)
summary(lr)
```

```
##
## Call:
## lm(formula = quality ~ ., data = trainset)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.2218 -0.4808 -0.0256  0.4856  2.8788
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)           1.324e+02  2.174e+01   6.088 1.29e-09 ***
## fixed.acidity         2.771e-02  2.517e-02   1.101    0.271
## volatile.acidity     -1.684e+00  1.423e-01 -11.831  < 2e-16 ***
## citric.acid           1.838e-01  1.213e-01   1.515    0.130
## residual.sugar        6.636e-02  9.020e-03   7.358 2.41e-13 ***
## chlorides            -5.939e-01  6.884e-01  -0.863    0.388
## free.sulfur.dioxide   5.062e-03  1.088e-03   4.654 3.40e-06 ***
## total.sulfur.dioxide -1.602e-04  4.787e-04  -0.335    0.738
## density              -1.325e+02  2.206e+01  -6.004 2.15e-09 ***
## pH                    6.743e-01  1.303e-01   5.174 2.45e-07 ***
## sulphates             6.544e-01  1.283e-01   5.099 3.64e-07 ***
## alcohol               2.229e-01  2.865e-02   7.778 1.01e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7552 on 2959 degrees of freedom
## Multiple R-squared:  0.3002, Adjusted R-squared:  0.2976
## F-statistic: 115.4 on 11 and 2959 DF,  p-value: < 2.2e-16
```

We chose not to perform an interpretation of the coefficients because, since the model does not fit the data well, we would obtain a conclusion not truly reflecting such data.

```
preds<-predict(lr,testset[,-12])
mean((testset$quality-preds)^2)
```

```
## [1] 0.5206929
```

MSE = 0.521: improved.

Feature Engineering

Having seen that some variables were more significant than others, it seems that there is room for improvement by performing feature engineering. Hence, we proceed to do so on our multiple linear-regression model.
StepAIC
We perform three model selections using AIC as a criterion to select the most relevant predictors.
(We did not use BIC because it yielded the same results, so in the end we opted for the more standard and common AIC).

Forward:
Starting with a null model (only the intercept), the forward AIC adds the best predictor at each step:

```
modelstepaicforward<- step( lr, direction = "forward", trace = F)
summary(modelstepaicforward)
```

```
##
## Call:
## lm(formula = quality ~ fixed.acidity + volatile.acidity + citric.acid +
##     residual.sugar + chlorides + free.sulfur.dioxide + total.sulfur.dioxide +
##     density + pH + sulphates + alcohol, data = trainset)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.2218 -0.4808 -0.0256  0.4856  2.8788
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)           1.324e+02  2.174e+01   6.088 1.29e-09 ***
## fixed.acidity         2.771e-02  2.517e-02   1.101    0.271
## volatile.acidity     -1.684e+00  1.423e-01 -11.831  < 2e-16 ***
## citric.acid           1.838e-01  1.213e-01   1.515    0.130
## residual.sugar        6.636e-02  9.020e-03   7.358 2.41e-13 ***
## chlorides            -5.939e-01  6.884e-01  -0.863    0.388
## free.sulfur.dioxide   5.062e-03  1.088e-03   4.654 3.40e-06 ***
## total.sulfur.dioxide -1.602e-04  4.787e-04  -0.335    0.738
## density              -1.325e+02  2.206e+01  -6.004 2.15e-09 ***
## pH                    6.743e-01  1.303e-01   5.174 2.45e-07 ***
## sulphates             6.544e-01  1.283e-01   5.099 3.64e-07 ***
## alcohol               2.229e-01  2.865e-02   7.778 1.01e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7552 on 2959 degrees of freedom
## Multiple R-squared:  0.3002, Adjusted R-squared:  0.2976
## F-statistic: 115.4 on 11 and 2959 DF,  p-value: < 2.2e-16
```

```r
preds<-predict(modelstepaicforward,testset[,-12])
mean((testset$quality-preds)^2)
```

```
## [1] 0.5206929
```

The forward AIC results in the full model, hence not removing any variables.
MSE = 0.521.

Backward:
Starting with the full model (all predictors), the backward AIC instead removes the worst predictors at each step:

```r
modelstepaicback<- step( lr, direction = "backward", trace = F)
summary(modelstepaicback)
```

```
##
## Call:
## lm(formula = quality ~ volatile.acidity + citric.acid + residual.sugar +
##     free.sulfur.dioxide + density + pH + sulphates + alcohol,
##     data = trainset)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.1964 -0.4850 -0.0219  0.4806  2.5786
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          1.182e+02  1.586e+01   7.454 1.18e-13 ***
## volatile.acidity    -1.732e+00  1.370e-01 -12.645  < 2e-16 ***
## citric.acid          1.823e-01  1.192e-01   1.530    0.126
## residual.sugar       6.095e-02  6.859e-03   8.886  < 2e-16 ***
## free.sulfur.dioxide  4.765e-03  8.671e-04   5.495 4.23e-08 ***
## density             -1.180e+02  1.590e+01  -7.419 1.53e-13 ***
## pH                   5.868e-01  9.910e-02   5.921 3.56e-09 ***
## sulphates            6.247e-01  1.258e-01   4.965 7.27e-07 ***
## alcohol              2.449e-01  2.342e-02  10.457  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7551 on 2962 degrees of freedom
## Multiple R-squared:  0.2996, Adjusted R-squared:  0.2977
## F-statistic: 158.4 on 8 and 2962 DF,  p-value: < 2.2e-16
```

```r
preds<-predict(modelstepaicback,testset[,-12])
mean((testset$quality-preds)^2)
```

```
## [1] 0.523592
```

The backward AIC removes `total sulfur dioxide`, `chlorides` and `fixed acidity`, considering them the least relevant for the prediction.
MSE = 0.524.

Stepwise:
After progressively adding new variables, it checks if all the remaining ones are significant. Otherwise, it removes them and tries again.

```r
modelstepaicboth<- step( lr, direction = "both", trace = F)
summary(modelstepaicboth)
```

```
##
## Call:
## lm(formula = quality ~ volatile.acidity + citric.acid + residual.sugar +
##     free.sulfur.dioxide + density + pH + sulphates + alcohol,
##     data = trainset)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.1964 -0.4850 -0.0219  0.4806  2.5786
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          1.182e+02  1.586e+01   7.454 1.18e-13 ***
## volatile.acidity    -1.732e+00  1.370e-01 -12.645  < 2e-16 ***
## citric.acid          1.823e-01  1.192e-01   1.530    0.126
## residual.sugar       6.095e-02  6.859e-03   8.886  < 2e-16 ***
## free.sulfur.dioxide  4.765e-03  8.671e-04   5.495 4.23e-08 ***
## density             -1.180e+02  1.590e+01  -7.419 1.53e-13 ***
## pH                   5.868e-01  9.910e-02   5.921 3.56e-09 ***
## sulphates            6.247e-01  1.258e-01   4.965 7.27e-07 ***
## alcohol              2.449e-01  2.342e-02  10.457  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7551 on 2962 degrees of freedom
## Multiple R-squared:  0.2996, Adjusted R-squared:  0.2977
## F-statistic: 158.4 on 8 and 2962 DF,  p-value: < 2.2e-16
```

```
preds<-predict(modelstepaicboth,testset[,-12])
mean((testset$quality-preds)^2)
```

```
## [1] 0.523592
```

The stepwise AIC yields the same model as the stepwise one and therefore the same MSE.
MSE = 0.524: it did not improve with respect to the model before performing AIC.

MSE results are essentially the same with all types of steps: this means the linear regression model is limited to having this error.

Further Experiments to Improve the Linear Regression
We remove two of the least statistically significant variables (based on their p-value): `total sulfur dioxide` and `chlorides`.
We originally intended to remove the ones most correlated between them, among the ones least significant for the model (in terms of p-value). However, since of these was `citric.acid`, which was the only one of the four kept by the AIC model selections, we instead decided to remove the second most correlated ones.

```
train.dataint<-trainset[,c(-7,-5)]
test.dataint<-testset[,c(-7,-5)]
```

We only removed two variables to avoid risking that model performs even worse: in fact, if we otherwise removed too many variables, the model would have resulted in a higher bias (and hence smaller variance).

Trying interaction models on the newly obtained subset of variables (later we'll reuse them all):
1) `DensityxAlcohol`, because of their high correlation to the output:

```
modinteraction<-lm(quality~ alcohol+I(density*alcohol)+ residual.sugar+free.sulfur.dioxide+sulphates+pH
summary(modinteraction)
```

```
##
## Call:
## lm(formula = quality ~ alcohol + I(density * alcohol) + residual.sugar +
##     free.sulfur.dioxide + sulphates + pH + density + volatile.acidity +
##     fixed.acidity, data = train.dataint)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.2132 -0.4782 -0.0223  0.4836  3.0817
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)           8.790e+01  5.416e+01   1.623    0.105
## alcohol               4.073e+00  4.265e+00   0.955    0.340
## I(density * alcohol) -3.868e+00  4.290e+00  -0.902    0.367
## residual.sugar        6.460e-02  9.220e-03   7.006 3.02e-12 ***
## free.sulfur.dioxide   4.829e-03  8.715e-04   5.540 3.28e-08 ***
## sulphates             6.462e-01  1.282e-01   5.042 4.87e-07 ***
## pH                    6.646e-01  1.280e-01   5.192 2.22e-07 ***
## density              -8.775e+01  5.459e+01  -1.607    0.108
## volatile.acidity     -1.755e+00  1.370e-01 -12.816  < 2e-16 ***
## fixed.acidity         3.299e-02  2.479e-02   1.331    0.183
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7552 on 2961 degrees of freedom
## Multiple R-squared:  0.2997, Adjusted R-squared:  0.2976
## F-statistic: 140.8 on 9 and 2961 DF,  p-value: < 2.2e-16
```

```
preds<-predict(modinteraction,testset[,-12])
mean((testset$quality-preds)^2)
```

```
## [1] 0.5226331
```

Adjusted R Squared = 0.2976: interaction between `density` and `alcohol` doesn't really impact.
MSE = 0.523.

2) `PhxVolatile Acidity`, because of their high correlation (right after the previous ones):

```
modinteraction2<- lm(quality~ alcohol+ residual.sugar+free.sulfur.dioxide+sulphates+pH+ density+volatil
summary(modinteraction2)
```

```
##
## Call:
## lm(formula = quality ~ alcohol + residual.sugar + free.sulfur.dioxide +
##     sulphates + pH + density + volatile.acidity + I(pH * volatile.acidity),
##     data = train.dataint)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.2289 -0.4846 -0.0229  0.4824  2.4786
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)               1.131e+02  1.544e+01   7.321 3.16e-13 ***
## alcohol                   2.513e-01  2.299e-02  10.931  < 2e-16 ***
## residual.sugar            5.901e-02  6.741e-03   8.753  < 2e-16 ***
## free.sulfur.dioxide       4.858e-03  8.654e-04   5.613 2.17e-08 ***
## sulphates                 6.271e-01  1.259e-01   4.983 6.63e-07 ***
## pH                        4.140e-01  2.492e-01   1.661   0.0967 .
## density                  -1.122e+02  1.543e+01  -7.274 4.46e-13 ***
## volatile.acidity         -3.403e+00  2.763e+00  -1.232   0.2182
## I(pH * volatile.acidity)  5.077e-01  8.598e-01   0.591   0.5549
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 0.7554 on 2962 degrees of freedom
## Multiple R-squared:  0.2991, Adjusted R-squared:  0.2973
## F-statistic:   158 on 8 and 2962 DF,  p-value: < 2.2e-16
```

```
preds<-predict(modinteraction2,testset[,-12])
mean((testset$quality-preds)^2)
```

```
## [1] 0.5256487
```

MSE = 0.526: slightly worse, so this interaction term explains the variance even less than the previous one.

Regularization Models

We do this in order to shrink the coefficient estimates towards zero: this way, we discourage learning a more complex or flexible model, so as to avoid the risk of overfitting.

Ridge
Unlike variable selection techniques (the stepwise performed above), the ridge regression will always return p predictors (no variable selection performed).
In order to shrink coefficient estimates towards zero, we will tune the lambda parameter (which controls the amount of shrinkage) through cross validation. As lambda grows, the coefficients get closer to zero, meaning that they will impact less and less the response; a lambda parameter equal to zero results in a linear regression.

```
nlambdas <- 1000
lambdas <- seq(0.001, 2, length.out = nlambdas)
```

```
modridge <- cv.glmnet(x, y,nfolds=folds, alpha=0)
modridge <- train(x, y, method = "glmnet", trControl = ctrl,
                  tuneGrid = expand.grid(alpha = 0,
                                         lambda = lambdas))
preds<-predict(modridge,as.matrix(testmatrix[,-12]))
mean((preds - testmatrix[,12])^2 )
```

```
## [1] 0.523604
```

MSE = 0.524: no improvement.

Lasso
Contrarily to Ridge, LASSO also performs variable selection. This is because, as lambda increases, coefficients not only get closer to zero, but even come to equal it at times; when they do, their respective variable is effectively discarded.

```
lambdas <- 10^seq(2, -3, by = -.1)
```

```
ctrl <- trainControl(method="cv", number=folds)
modlasso <- train(x, y, method = "glmnet", trControl = ctrl,
                     tuneGrid = expand.grid(alpha = 1,
                                            lambda = lambdas))
```

```
modlasso$bestTune$lambda
```

```
## [1] 0.005011872
```

Such a low lambda value means our model chose an extremely low penalty, hence the model resembles the one obtained earlier with a normal linear regression.

```
preds<-predict(modlasso,as.matrix(testmatrix [,-12]))
mean((preds - testmatrix[,12])^2)
```

```
## [1] 0.5231457
```

MSE = 0.523: better than ridge, yet still no clear improvement.
The above explanation of lambda justifies having obtained an MSE similar to the linear regression one.

Elastic net
It is a mix between the two previous types of regularization, determined by the parameter alpha, ranging between 0 (ridge) and 1 (lasso). Once again, such parameter is chosen through cross validation.

```
lambda.grid<-10^seq(2,-2,length=100)
alpha.grid<-seq(0,1,length=10)
grid<-expand.grid(alpha=alpha.grid,lambda=lambda.grid)

Elnet<-train(x, y, method = "glmnet", trControl = ctrl,tuneGrid = grid)
Elnet$bestTune
```

```
##         alpha     lambda
## 109 0.1111111 0.02104904
```

```
preds<-predict(Elnet,as.matrix(testmatrix [,-12]))
mean((preds - testmatrix[,12])^2)
```

```
## [1] 0.5234057
```

The value obtained for the alpha parameter (0.1) indicates the mix between ridge and LASSO tends towards ridge; in fact, an alpha value of 0 means the model obtained is solely a ridge one, whilst a value of 1 consists in a LASSO model.
MSE = 0.523: still no improvement

Since 0.523 is the best MSE we could achieve through a linear model (which, as we said before, is not the ideal model to predict such an output), it appears such model is limited to having this error.

Non-linear Models

Random Forest
We chose a number of trees equal to 1000 due to how the error changed with different numbers of trees:

with 500, we tried and got a worse MSE, whilst with 2000 it did not change much and was only slower to execute.
We expect this model to perform better than the linear regression one because, through its splitting mechanism, it might be better able to capture the discreteness of the response.

```
Rf <- randomForest(
  formula = quality~ .,
  data= trainset,
  importance=TRUE,
  ntree=1000
)
preds<-predict(Rf,testset[-12])
mean((preds - testset[,12])^2)
```

```
## [1] 0.4498982
```

MSE = 0.450: the best so far. Our expectations have been confirmed.

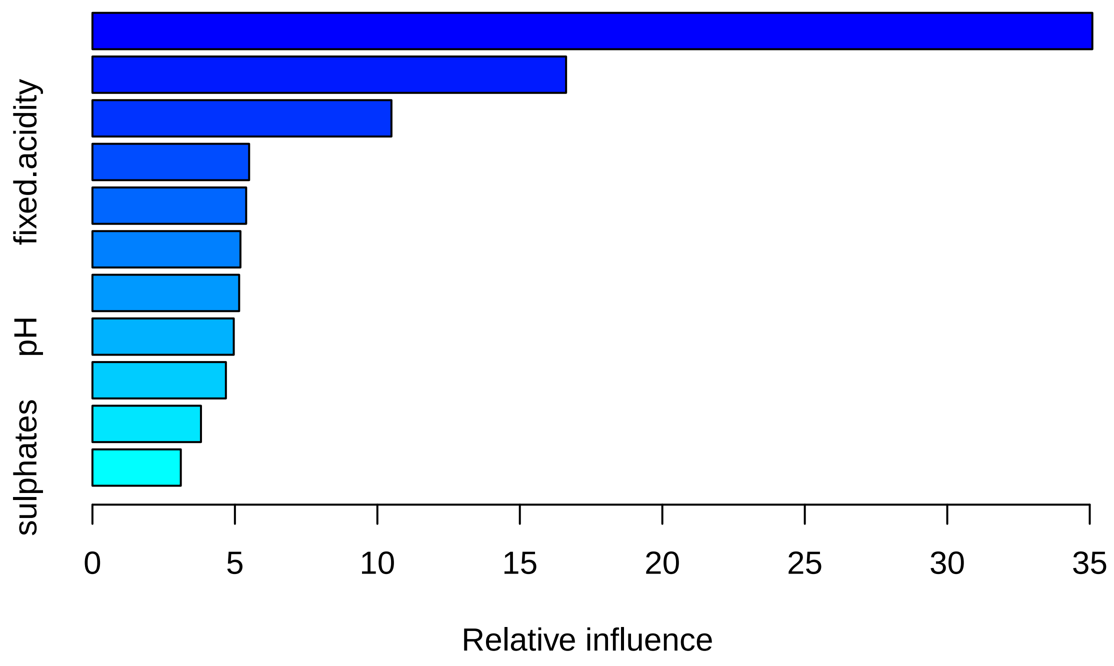
Boosting

Boosting is a type of ensemble learning which, rather than building the trees in a random forest indipendently, builds them one at a time, optimizing at each step in chosing the best tree possible, based on the previously built one.
Gradient Boosting

```
boost = gbm(quality ~.,
                data = trainset,
                distribution='gaussian',
                cv.folds = 5,
                shrinkage = .1,
                n.minobsinnode = 10,
                n.trees = 500 )
```

Here we show the relative influence of all variables, in order:

```
summary(boost)
```



```
##                                         var   rel.inf
## alcohol                             alcohol 35.091557
## free.sulfur.dioxide   free.sulfur.dioxide 16.622328
## volatile.acidity         volatile.acidity 10.493258
## fixed.acidity               fixed.acidity  5.499462
## total.sulfur.dioxide total.sulfur.dioxide  5.395450
## residual.sugar             residual.sugar  5.193579
## chlorides                       chlorides  5.149372
## pH                                     pH  4.960164
## citric.acid                   citric.acid  4.684301
## density                           density  3.809229
## sulphates                       sulphates  3.101297
```

```
predboost<-predict(boost,testset[,-12])
```

```
mean((predboost - testset[,12])^2)
```

```
## [1] 0.4916466
```

MSE = 0.492: random forest performed better.

XG Boost
A popular type of boosting implementation and a more regularized version of gradient boosting, which usually performs very well; this time, however, it probably suffers from the same issues we faced with the linear regression regarding predicting a discrete variable as a continuous one.

```
modxgboost = xgboost(data = trainmatrix[,-12], label = trainmatrix[,12], nrounds = 150,
                 objective = "reg:squarederror", eval_metric = "error")
```

```
err_xg_tr = modxgboost$evaluation_log$train_error
predxgboost<-predict(modxgboost,testmatrix[,-12])
mean((predxgboost - testset[,12])^2)
```

```
## [1] 0.5467072
```

MSE = 0.547: again, random forest performed better.

PCA

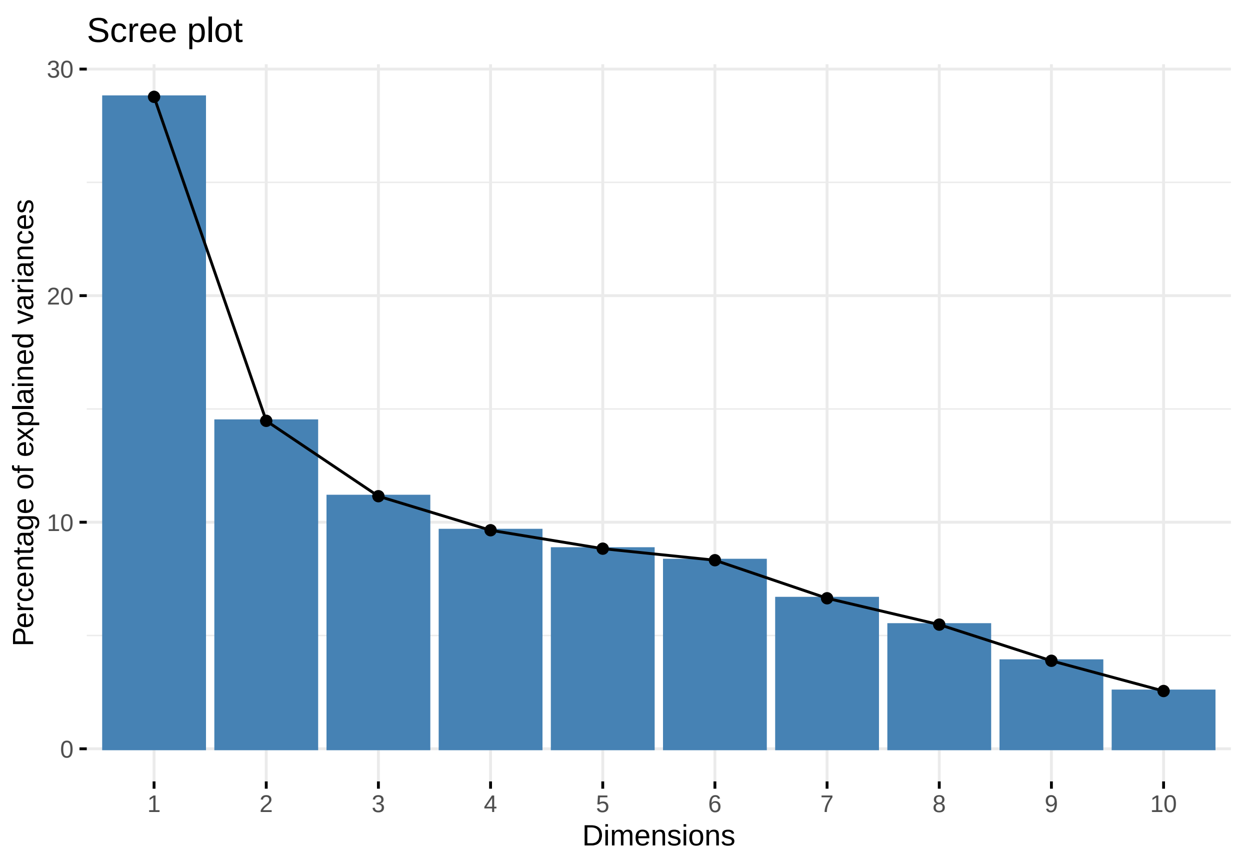
Visualizing the number of components needed:

```
pca <- princomp(trainset[,-12],
                cor = TRUE,
                score = TRUE)
```

```
summary(pca)
```

```
## Importance of components:
##                           Comp.1    Comp.2    Comp.3     Comp.4     Comp.5
## Standard deviation     1.7791016 1.2618355 1.1074003 1.03006330 0.98571606
## Proportion of Variance 0.2877457 0.1447481 0.1114850 0.09645731 0.08833056
## Cumulative Proportion  0.2877457 0.4324938 0.5439788 0.64043611 0.72876667
##                            Comp.6    Comp.7     Comp.8     Comp.9    Comp.10
## Standard deviation     0.95685025 0.8548937 0.77650663 0.65368660 0.52964480
## Proportion of Variance 0.08323295 0.0664403 0.05481478 0.03884602 0.02550215
## Cumulative Proportion  0.81199962 0.8784399 0.93325469 0.97210071 0.99760286
##                            Comp.11
## Standard deviation     0.162384017
## Proportion of Variance 0.002397143
## Cumulative Proportion  1.000000000
```

```
fviz_eig(pca)
```



By using the elbow method, it is clear that 4 components are more than enough to explain most of the variance: 64%. However, we expect this reduction not to work too well, because the number of dimensions we are trying to reduce is already quite low compared to the datasets PCA is usually used onto.

Principal component regression

```
pcrmodel <- pcr(quality~., data = trainset, scale = TRUE, validation = "CV")
pcrpred <- predict(pcrmodel, testset[,-12], ncomp = 4)
mean((pcrpred - testset[,12])^2)
```

```
## [1] 0.6002491
```

MSE = 0.600: the worst overall. We believe this is due to both a loss of information that the PCA causes and to the dataset not being suited to such method because of its already low number of dimensions.

Comparing the Results

```
MSE[order(MSE$MSE),]
```

```
##                  Model        MSE
## 12     RandomForest 0.4498982
## 13         Boosting 0.4916466
## 2   Multivariate LR 0.5206929
## 3   Step_forwardAIC 0.5206929
## 6   Interaction LR1 0.5226331
## 9             Lasso 0.5231457
## 10      Elastic Net 0.5234057
## 4  Step_backwardAIC 0.5235920
## 5      Step_bothAIC 0.5235920
## 11            Ridge 0.5236040
## 7   Interaction LR2 0.5256487
## 14          XGBoost 0.5467072
## 1          baseline 0.5698962
## 8               PCR 0.6002491
```

Conclusions

Premised that predicting the quality of wines was not a regression task but most likely a multiclass classification task, the best model among those we fit seems to be the Random Forest with MSE =0.456, together with gradient boosting with 0.497.
Random forest performs the best because it is the simplest and most flexible one; therefore, through its splits, it adapts better to the discreteness of our response, compared to a regression line model. Regularization methods, instead, do not seem to impact that much on the performance of the model.
Finally, PCR performs the worst because, we believe, it loses some of the variability and information of the dataset. Moreover, being the dataset already composed of few dimensions, it does not help as much as it usually would.

TASK 2 BAD/GOOD WINES

Building onto the reasons for which we performed the first task, we decided to emulate the process a wine expert might go through of classifying a wine's quality as "Good" (sufficient, 6 or above on a scale from 1 to 10) or "Bad" (insufficient, below 6). What is more, such a classification could help in identifying which variables to focus on to improve one's wine and obtain a "good" one.
Therefore, if predicted wine quality is higher than 6, we will classify it as a "good" wine; otherwise, as a "bad" one.

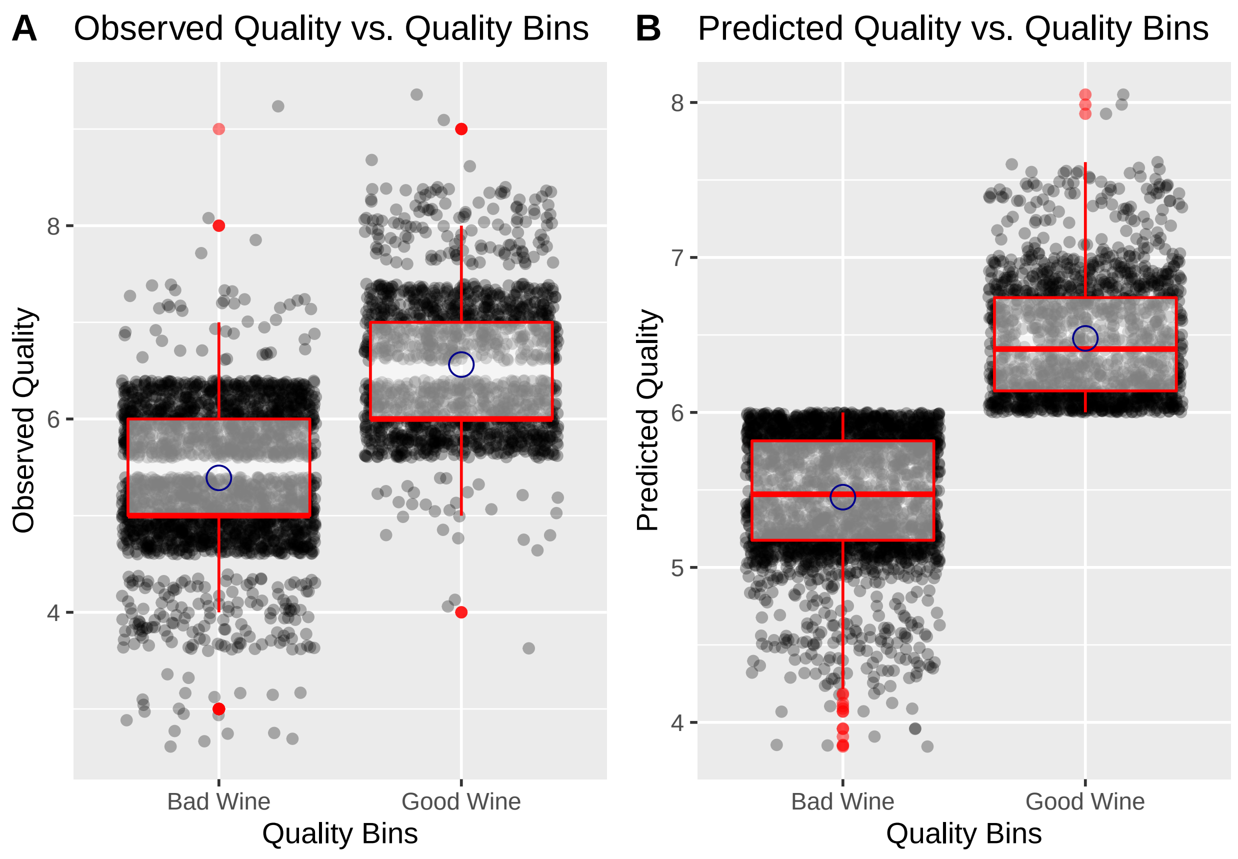
To perform such task, we apply the best model previously obtained on all instances of the data: Random forest.

```
winepred<- ifelse(predict(Rf, df[,-12]) >= 6 , "Good Wine", "Bad Wine")
df$Winequality<- winepred
df[-c(1:5,15:3961),-c(1:11)]
```

```
##    quality Winequality
## 10       6    Bad Wine
## 11       5    Bad Wine
## 12       5    Bad Wine
## 13        5     Bad Wine
## 14        7    Good Wine
## 15        5     Bad Wine
## 16        7    Good Wine
## 17        6     Bad Wine
## 18        8    Good Wine
```

Some wines with quality equal to 6 are classified as bad wines: this is because, despite showing as a 6, they were originally slightly smaller than 6 and are now showing rounded by excess.

Bins Comparison



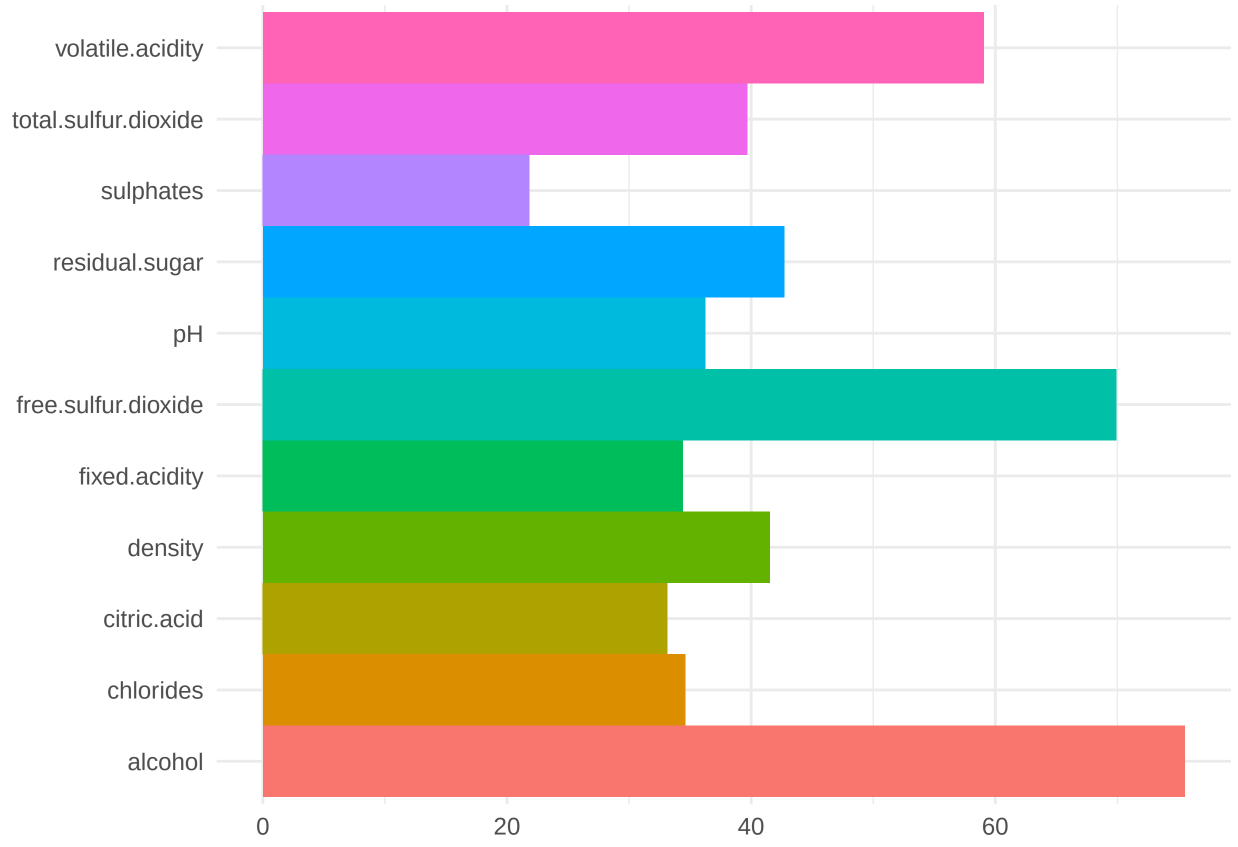
Feature Importance
Looking at the most important variables for predicting wine quality in the Rf model:

```
Featureimportance <- varImp(Rf, conditional=TRUE)

Featureimportance <- Featureimportance %>% tibble::rownames_to_column("var")
Featureimportance$var<- Featureimportance$var %>% as.factor()
```

Whilst in linear regression such variable selection procedure is performed by looking at the p-value and at the correlation with the response, the random forest model provides its own method to do it.

Plotting the bar chart to compare feature importance:



The 5 most important variables for predicting the quality of the wine are (in ranks):
1) `alcohol`;
2) `free.sulfur.dioxide`;
3) `volatile.acidity`;
4) `residual.sugar`;
5) `density`.

Conclusions

Those above are the five most important variables for predicting whether a wine will result in a "good" or a "bad" one, simply based on its specifics. Such an analysis can be valuable as a benchmark to know in which direction to move (and so which variables to focus on) in order to improve one's wine or even simply to produce a "good" one, independently of the exact quality score one aims at. The exact quality marks ranging from 0 to 10, in fact, do not help clearly in giving a general judgement about the wine; by partitioning such marks into two categories ("good" and "bad"), instead, one is better able to judge what a specific mark means in context.