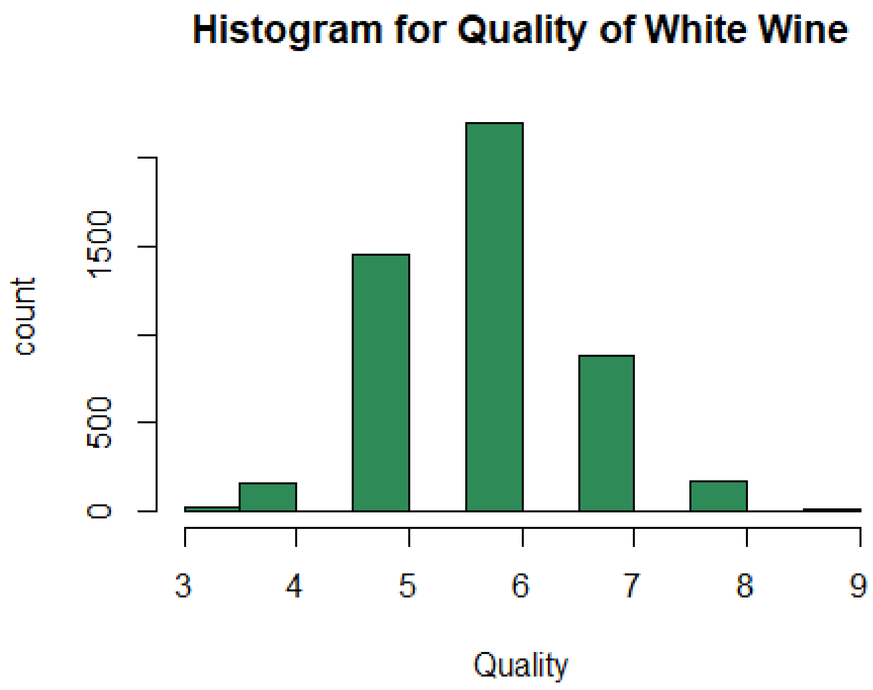
```r
rm(list = ls())

library(ISLR)
library(class)
library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

library(leaps)
library(corrplot)

## corrplot 0.84 loaded

library(car)

## Loading required package: carData

require(e1071)

## Loading required package: e1071

library(bootstrap)
library(rpart)
library(gbm)

## Loading required package: survival

##
## Attaching package: 'survival'

## The following object is masked from 'package:caret':
##
##     cluster

## Loading required package: splines

## Loading required package: parallel

## Loaded gbm 2.1.3

library(ggplot2)
library(ggcorrplot)

white_wine = read.csv2("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-qu
ality/winequality-white.csv")

summary(white_wine)

##  fixed.acidity  volatile.acidity  citric.acid   residual.sugar
##  6.8    : 308   0.28   : 263     0.3    : 307   1.2    : 187
##  6.6    : 290   0.24   : 253     0.28   : 282   1.4    : 184
##  6.4    : 280   0.26   : 240     0.32   : 257   1.6    : 165
##  6.9    : 241   0.25   : 231     0.34   : 225   1.3    : 147
##  6.7    : 236   0.22   : 229     0.29   : 223   1.1    : 146
##  7      : 232   0.27   : 218     0.26   : 219   1.5    : 142
##  (Other):3311   (Other):3464     (Other):3385   (Other):3927
##    chlorides    free.sulfur.dioxide total.sulfur.dioxide    density
##  0.044   : 201   29      : 160        111     :  69         0.992  :  64
##  0.036   : 200   31      : 132        113     :  61         0.9928 :  61
##  0.042   : 184   26      : 129        117     :  57         0.9932 :  53
##  0.04    : 182   35      : 129        118     :  55         0.993  :  52
##  0.046   : 181   34      : 128        114     :  54         0.9934 :  50
##  0.048   : 174   36      : 127        122     :  54         0.9938 :  49
##  (Other):3776   (Other):4093        (Other):4548         (Other):4569
##        pH          sulphates        alcohol         quality
##  3.14    : 172   0.5     : 249   9.4     : 229   Min.   :3.000
##  3.16    : 164   0.46    : 225   9.5     : 228   1st Qu.:5.000
##  3.22    : 146   0.44    : 216   9.2     : 199   Median :6.000
##  3.19    : 145   0.38    : 214   9       : 185   Mean   :5.878
##  3.18    : 138   0.42    : 181   10      : 162   3rd Qu.:6.000
##  3.2     : 137   0.48    : 179   10.5    : 160   Max.   :9.000
##  (Other):3996   (Other):3634   (Other):3735
```

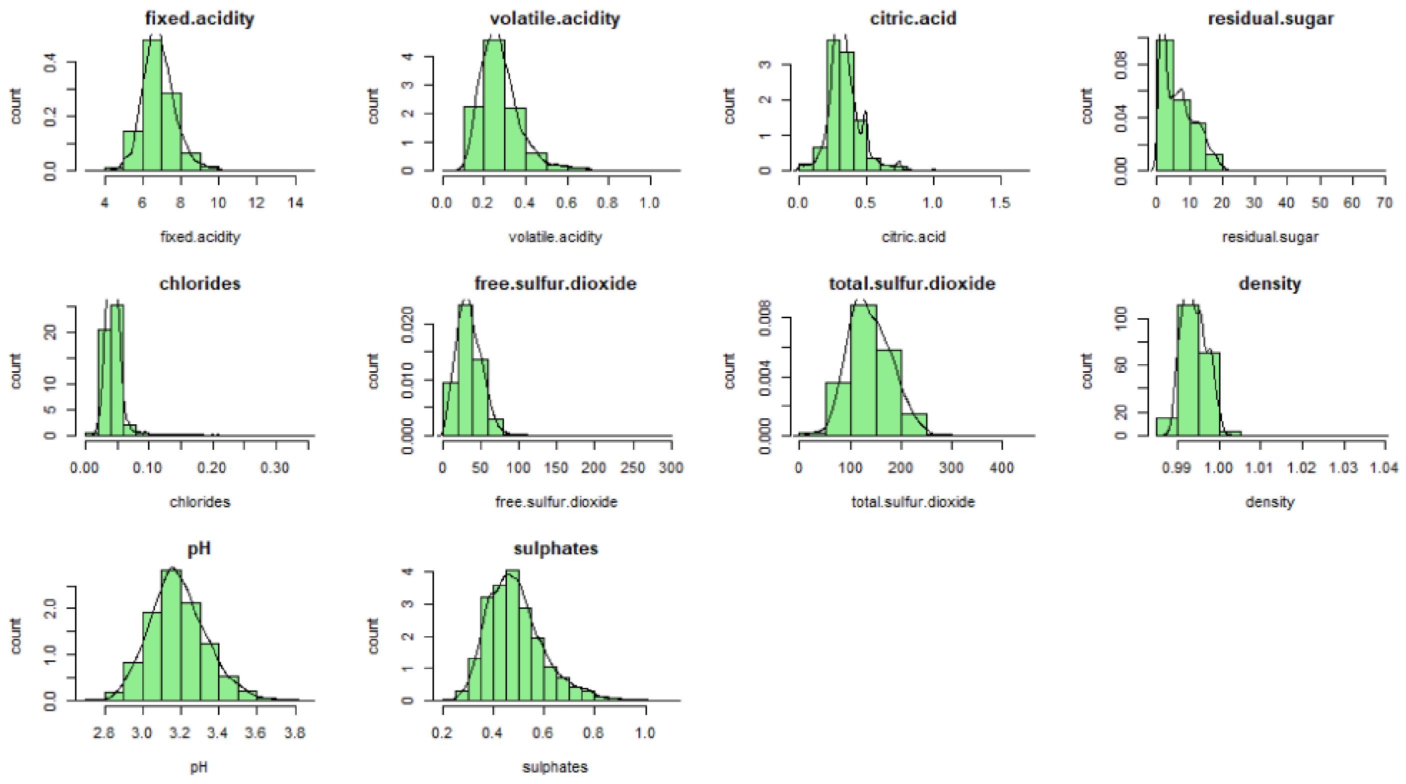
The white wine dataset contain 12 variables and 4898 observations. The 12 variables are: fixed.acidity, volatile.acidity, citric.acid, residual.sugar, chlorides, free.sulfur.dioxide, total.sulfur.dioxide, density, pH, sulphates, alcohol, quality.

```
white_wine = data.frame(lapply(white_wine, function(x) as.numeric(as.character(x))))

hist(white_wine$quality, main = "Histogram for Quality of White Wine",
     xlab ="Quality", ylab = "count", col ="seagreen")
```



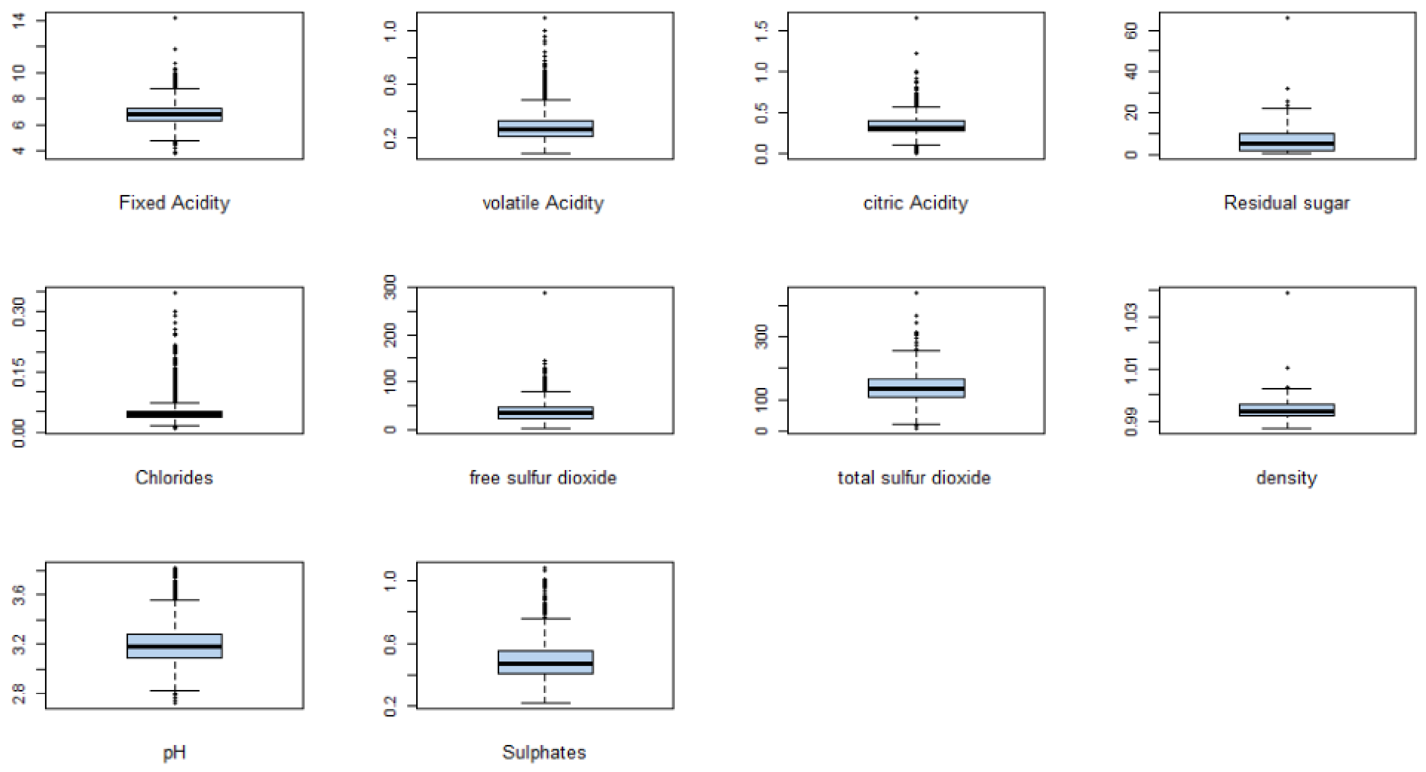
Quality ranges from 3 to 9 for white wine. It has most values concentrated in the categories 5, 6 and 7. Only a small proportion is in the categories 3,4,8 and 9

```r
####################################################################################
# Histogram for all predictors
####################################################################################

par(mfrow = c(3,4))
hist(white_wine$fixed.acidity, main = "fixed.acidity", prob = TRUE, xlab = "fixed.acidity
", ylab = "count", col = "lightgreen")
lines(density(white_wine$fixed.acidity), lwd = 1.5, col = "black")

hist(white_wine$volatile.acidity, main = "volatile.acidity", prob = TRUE, xlab = "volatil
e.acidity", ylab = "count", col = "lightgreen")
lines(density(white_wine$volatile.acidity), lwd = 1.5, col = "black")

hist(white_wine$citric.acid, main = "citric.acid", prob = TRUE,   xlab = "citric.acid", y
lab = "count", col = "lightgreen")
lines(density(white_wine$citric.acid), lwd = 1.5, col = "black")

hist(white_wine$residual.sugar, main = "residual.sugar", prob = TRUE, xlab = "residual.su
gar", ylab = "count", col = "lightgreen")
lines(density(white_wine$residual.sugar), lwd = 1.5, col = "black")

hist(white_wine$chlorides, main = "chlorides", prob = TRUE,       xlab = "chlorides", yla
b = "count", col = "lightgreen")
lines(density(white_wine$chlorides), lwd = 1.5, col = "black")

hist(white_wine$free.sulfur.dioxide, main = "free.sulfur.dioxide", prob = TRUE, xlab = "f
ree.sulfur.dioxide", ylab = "count", col = "lightgreen")
lines(density(white_wine$free.sulfur.dioxide), lwd = 1.5, col = "black")

hist(white_wine$total.sulfur.dioxide, main = "total.sulfur.dioxide", prob = TRUE, xlab =
"total.sulfur.dioxide", ylab = "count", col = "lightgreen")
lines(density(white_wine$total.sulfur.dioxide), lwd = 1.5, col = "black")

hist(white_wine$density, main = "density", prob = TRUE, xlab = "density", ylab = "count",
col = "lightgreen")
lines(density(white_wine$density), lwd = 1.5, col = "black")

hist(white_wine$pH, main = "pH", prob = TRUE, xlab = "pH", ylab = "count", col = "lightgr
een")
lines(density(white_wine$pH), lwd = 1.5, col = "black")

hist(white_wine$sulphates, main = "sulphates", prob = TRUE, xlab = "sulphates", ylab = "c
ount", col = "lightgreen")
lines(density(white_wine$sulphates), lwd = 1.5, col = "black")

hist(white_wine$alcohol, main = "alcohol", prob = TRUE, xlab = "alcohol", ylab = "count",
col = "lightgreen")
lines(density(white_wine$alcohol), lwd = 1.5, col = "black")
```



The black line superimposed on the histograms represents the bell-shaped “normal” curve. The data for fixed acidity, pH and alcohol are normal, and the data for all others are non-normal. In this case, the non-normality is driven by the presence of an outlier. We will with box-plots once for more confirmation.

```
########################################################################################
# Box plot to check the outliers
########################################################################################

par(mfrow = c(3,4))
boxplot(white_wine$fixed.acidity, horizontal = FALSE, col="slategray2", pch=19)
mtext("Fixed Acidity", cex=0.8, side=1, line=2)
boxplot(white_wine$volatile.acidity, horizontal = FALSE, col="slategray2", pch=19)
mtext("volatile Acidity", cex=0.8, side=1, line=2)
boxplot(white_wine$citric.acid, horizontal = FALSE, col="slategray2", pch=19)
mtext("citric Acidity", cex=0.8, side=1, line=2)
boxplot(white_wine$residual.sugar, horizontal = FALSE, col="slategray2", pch=19)
mtext("Residual sugar", cex=0.8, side=1, line=2)
boxplot(white_wine$chlorides, horizontal = FALSE, col="slategray2", pch=19)
mtext("Chlorides", cex=0.8, side=1, line=2)
boxplot(white_wine$free.sulfur.dioxide, horizontal = FALSE, col="slategray2", pch=19)
mtext("free sulfur dioxide", cex=0.8, side=1, line=2)
boxplot(white_wine$total.sulfur.dioxide,horizontal = FALSE, col="slategray2", pch=19)
mtext("total sulfur dioxide", cex=0.8, side=1, line=2)
boxplot(white_wine$density,horizontal = FALSE, col="slategray2", pch=19)
mtext("density", cex=0.8, side=1, line=2)
boxplot(white_wine$pH, horizontal = FALSE, col="slategray2", pch=19)
mtext("pH", cex=0.8, side=1, line=2)
boxplot(white_wine$sulphates,horizontal = FALSE, col="slategray2", pch=19)
mtext("Sulphates", cex=0.8, side=1, line=2)
boxplot(white_wine$alcohol,horizontal = FALSE, col="slategray2", pch=19)
mtext("Alcohol", cex=0.8, side=1, line=2)
```



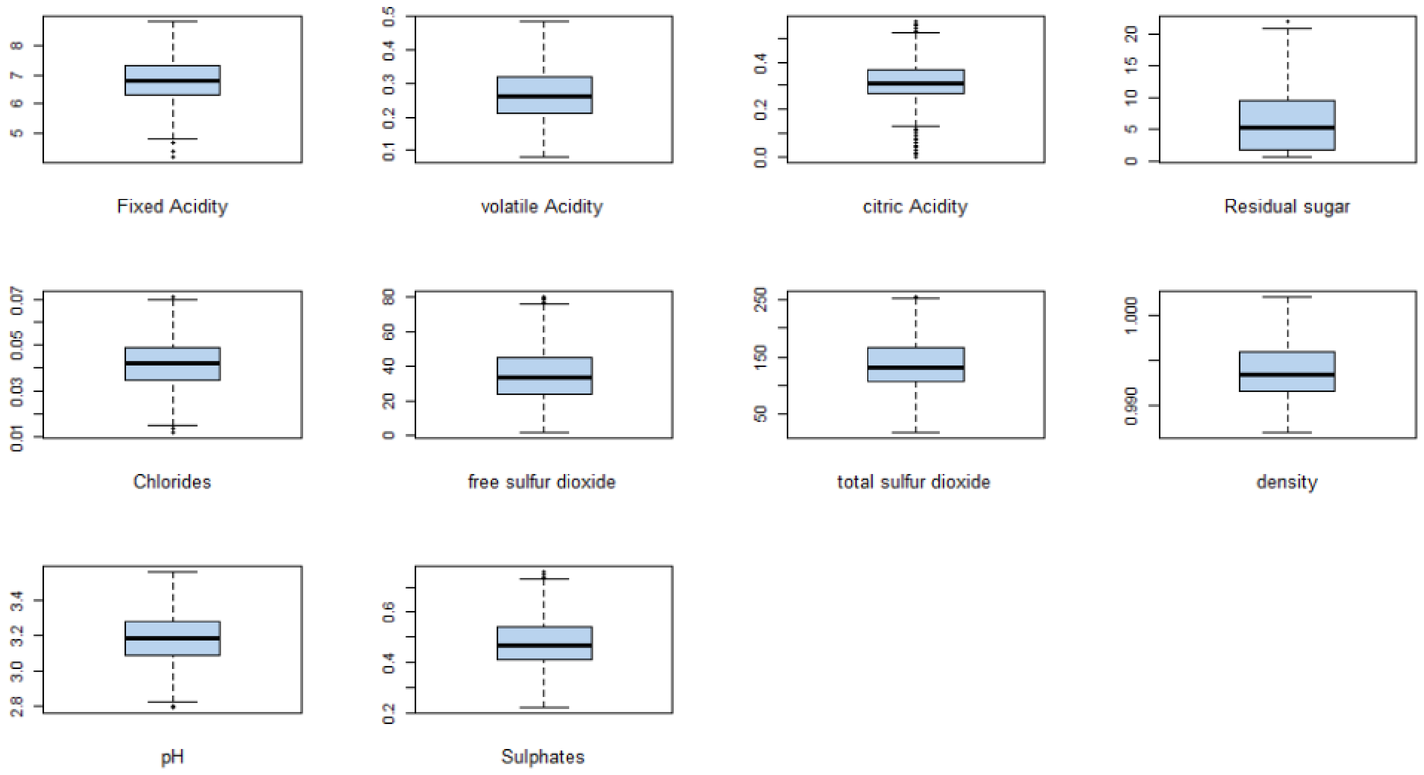
If Q1 and Q3 are the lower and upper quartiles respectively, then one can define an outlier to be any observation outside the range: [Q1-k(Q3-Q1), (Q3+k(Q3-Q1)] For some non-negative constant k. John Tukey proposed that k=1.5 indicates "outlier". As mostly outliers are on the larger side, we considered removal of outlier if it is greater than if it is greater than Q3 + 1.5IQR

```
##################################################################################
# Removing the outliers
##################################################################################

outliers = rep(0,11)

for (i in 1:11){
  t1 <- quantile(white_wine[,i], 0.75)
  t2 <- IQR(white_wine[,i], 0.75)
  outliers[i] <- t1 + 1.5*t2
}
white_wine_index = matrix(0, 4898, 11)
for (i in 1:4898)
  for (j in 1:11){
    if (white_wine[i,j] > outliers[j]) white_wine_index[i,j] = 1
  }
w_index = apply(white_wine_index, 1, sum)
white_wine_data = cbind(w_index, white_wine)
index = rep(0)

j = 1
for (i in 1:4898){
  if (w_index[i] > 0) {index[j]= i
  j = j + 1}
  else j = j
}
```

```r
new_white_wine = white_wine[-index,]
dim(new_white_wine)

## [1] 4074   12
```

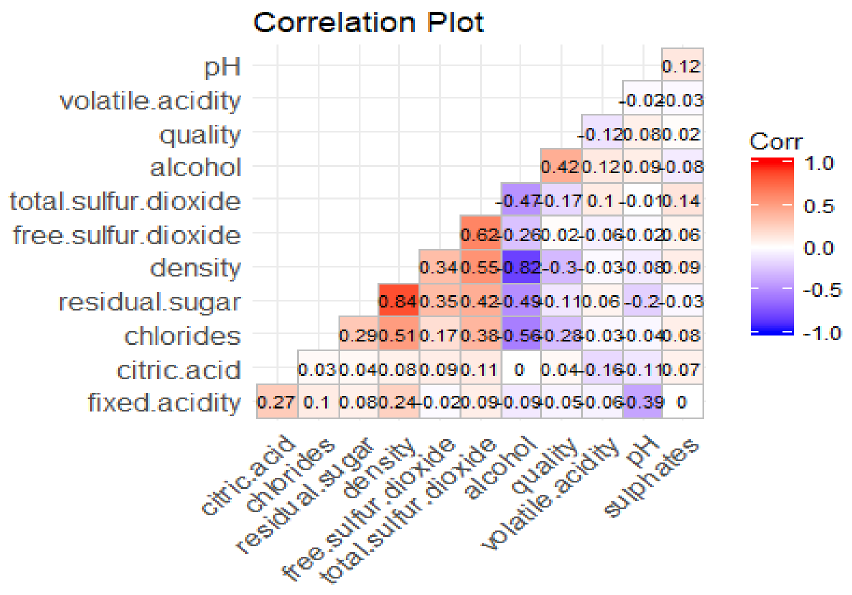
After removing outliers, number of observations for white wine is 4074.

```r
#################################################################################
# Box plot after removing outliers
#################################################################################

par(mfrow = c(3,4))
#boxplot(fixed.acidity)
boxplot(new_white_wine$fixed.acidity, horizontal = FALSE, col="slategray2", pch=19)
mtext("Fixed Acidity", cex=0.8, side=1, line=2)
boxplot(new_white_wine$volatile.acidity, horizontal = FALSE, col="slategray2", pch=19)
mtext("volatile Acidity", cex=0.8, side=1, line=2)
boxplot(new_white_wine$citric.acid, horizontal = FALSE, col="slategray2", pch=19)
mtext("citric Acidity", cex=0.8, side=1, line=2)
boxplot(new_white_wine$residual.sugar, horizontal = FALSE, col="slategray2", pch=19)
mtext("Residual sugar", cex=0.8, side=1, line=2)
boxplot(new_white_wine$chlorides, horizontal = FALSE, col="slategray2", pch=19)
mtext("Chlorides", cex=0.8, side=1, line=2)
boxplot(new_white_wine$free.sulfur.dioxide, horizontal = FALSE, col="slategray2", pch=19)
mtext("free sulfur dioxide", cex=0.8, side=1, line=2)
boxplot(new_white_wine$total.sulfur.dioxide,horizontal = FALSE, col="slategray2", pch=19)
mtext("total sulfur dioxide", cex=0.8, side=1, line=2)
boxplot(new_white_wine$density,horizontal = FALSE, col="slategray2", pch=19)
mtext("density", cex=0.8, side=1, line=2)
boxplot(new_white_wine$pH, horizontal = FALSE, col="slategray2", pch=19)
mtext("pH", cex=0.8, side=1, line=2)
boxplot(new_white_wine$sulphates,horizontal = FALSE, col="slategray2", pch=19)
mtext("Sulphates", cex=0.8, side=1, line=2)
boxplot(new_white_wine$alcohol,horizontal = FALSE, col="slategray2", pch=19)
mtext("Alcohol", cex=0.8, side=1, line=2)
```



We observe the data now without the outliers.

```
#################################################################################
# Correlation and feature selection based on correlation (Manual feature   selection)
#################################################################################
cor_white_wine = cor(new_white_wine)
par(mfrow =c(1,1))
ggcorrplot(cor_white_wine, type = "lower", title = "Correlation Plot",
  show.legend = TRUE, legend.title = "Corr", show.diag = FALSE,
  colors = c("blue", "white", "red"), outline.color = "gray",
  hc.order = TRUE, hc.method = "complete", lab = TRUE,
  lab_col = "black", lab_size = 2.9, p.mat = NULL, sig.level = 0.05,
  insig = c("pch", "blank"), pch = 4, pch.col = "black", pch.cex = 5,
  tl.cex = 12, tl.col = "black", tl.srt = 45)
```



In this collinearity graph, we see that:

- Density is highly positively correlated with residual sugar and negatively correlated with alcohol which might induce selection bias into the model.

- Fixed acidity has a strong positive correlation with citric.acid and a strong negative correlation with pH.

So, we remove density and fixed acidity from the data and proceed with remaining 9 variables

```
fit1 = lm(quality~., data = new_white_wine)
summary(fit1)

##
## Call:
## lm(formula = quality ~ ., data = new_white_wine)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.4014 -0.5109 -0.0429  0.4607  2.8018
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)           2.035e+02  2.747e+01   7.407 1.57e-13 ***
## fixed.acidity         1.590e-01  2.732e-02   5.821 6.30e-09 ***
## volatile.acidity     -1.881e+00  1.617e-01 -11.632  < 2e-16 ***
## citric.acid           4.312e-02  1.378e-01   0.313 0.754279
## residual.sugar        9.894e-02  1.026e-02   9.639  < 2e-16 ***
## chlorides            -3.379e+00  1.454e+00  -2.324 0.020161 *
## free.sulfur.dioxide   4.952e-03  1.050e-03   4.718 2.46e-06 ***
## total.sulfur.dioxide  2.577e-04  4.421e-04   0.583 0.560016
## density              -2.049e+02  2.783e+01  -7.360 2.21e-13 ***
## pH                    9.920e-01  1.292e-01   7.679 2.00e-14 ***
## sulphates             7.468e-01  1.258e-01   5.935 3.19e-09 ***
## alcohol               1.197e-01  3.440e-02   3.480 0.000507 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7439 on 4062 degrees of freedom
## Multiple R-squared:  0.2538, Adjusted R-squared:  0.2518
## F-statistic: 125.6 on 11 and 4062 DF,  p-value: < 2.2e-16

new_white_wine_mfs = new_white_wine[,-c(1,8)]
fit2 = lm(quality~., data = new_white_wine_mfs)
summary(fit2)

##
## Call:
## lm(formula = quality ~ ., data = new_white_wine_mfs)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.3432 -0.5129 -0.0259  0.4512  2.7406
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)           1.3751874  0.3380781   4.068 4.84e-05 ***
## volatile.acidity     -1.9391052  0.1623017 -11.948  < 2e-16 ***
## citric.acid          -0.0256545  0.1344885  -0.191 0.848726    
## residual.sugar        0.0264652  0.0029338   9.021  < 2e-16 ***
## chlorides            -5.4001158  1.4345632  -3.764 0.000169 ***
## free.sulfur.dioxide   0.0062279  0.0010328   6.030 1.78e-09 ***
## total.sulfur.dioxide -0.0005760  0.0004267  -1.350 0.177147    
## pH                    0.3286925  0.0876901   3.748 0.000180 ***
## sulphates             0.4831440  0.1213775   3.981 7.00e-05 ***
## alcohol               0.3527533  0.0135350  26.062  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 0.7487 on 4064 degrees of freedom
## Multiple R-squared:  0.2438, Adjusted R-squared:  0.2421 
## F-statistic: 145.6 on 9 and 4064 DF,  p-value: < 2.2e-16
```

In the above two lm models, it can be observed that citric acid and total sulfur dioxide are not significant.

```
new_white_wine_mfs = new_white_wine[,-c(1,3,7,8)]
fit3 = lm(quality~., data = new_white_wine_mfs)
summary(fit3)
```

```
## 
## Call:
## lm(formula = quality ~ ., data = new_white_wine_mfs)
## 
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -3.3207 -0.5140 -0.0263  0.4476  2.7475 
## 
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)    
## (Intercept)          1.327028   0.331840   3.999 6.47e-05 ***
## volatile.acidity    -1.984721   0.154904 -12.813  < 2e-16 ***
## residual.sugar       0.026024   0.002917   8.922  < 2e-16 ***
## chlorides           -5.732664   1.414629  -4.052 5.16e-05 ***
## free.sulfur.dioxide  0.005404   0.000843   6.411 1.62e-10 ***
## pH                   0.323196   0.086795   3.724 0.000199 ***
## sulphates            0.461791   0.120226   3.841 0.000124 ***
## alcohol              0.357150   0.013021  27.428  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 0.7487 on 4066 degrees of freedom
## Multiple R-squared:  0.2434, Adjusted R-squared:  0.2421 
## F-statistic: 186.9 on 7 and 4066 DF,  p-value: < 2.2e-16
```

```
new_white_wine_mfs = new_white_wine[,-c(1,3,6,7,8,10)]
fit4 = lm(quality~., data = new_white_wine_mfs)
summary(fit4)
```

```
## 
## Call:
## lm(formula = quality ~ ., data = new_white_wine_mfs)
## 
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -3.4992 -0.5038 -0.0415  0.4705  2.8201
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       1.596889   0.331808   4.813 1.54e-06 ***
## volatile.acidity -2.061280   0.155577 -13.249  < 2e-16 ***
## residual.sugar    0.030439   0.002818  10.802  < 2e-16 ***
## chlorides        -5.221333   1.422582  -3.670 0.000245 ***
## pH                0.389536   0.086701   4.493 7.22e-06 ***
## alcohol           0.347593   0.013047  26.642  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7539 on 4068 degrees of freedom
## Multiple R-squared:  0.2325, Adjusted R-squared:  0.2315
## F-statistic: 246.4 on 5 and 4068 DF,  p-value: < 2.2e-16
```

We proceed with 5 variable model with Volatile Acidity, Residual Sugar, Chlorides, pH and Alcohol

```
##################################################################################
# create train and test set
##################################################################################
set.seed(789)
train = sample(1:nrow(new_white_wine_mfs), 0.7*nrow(new_white_wine_mfs))
white_train_mfs = new_white_wine_mfs[train,]
white_test_mfs = new_white_wine_mfs[-train,]
white_y_train_mfs = white_train_mfs$quality
white_y_test_mfs = white_test_mfs$quality

##################################################################################
# Multiple Regression for manual Feature selection
##################################################################################
white_lm_mfs = lm(quality~., data = white_train_mfs)
summary(white_lm_mfs)
```

```
##
## Call:
## lm(formula = quality ~ ., data = white_train_mfs)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.4793 -0.4963 -0.0371  0.4740  2.5882
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       1.696508   0.403555   4.204  2.7e-05 ***
## volatile.acidity -1.924029   0.187113 -10.283  < 2e-16 ***
## residual.sugar    0.029886   0.003412   8.760  < 2e-16 ***
## chlorides        -4.119714   1.716081  -2.401  0.01643 *
## pH                0.311130   0.104296   2.983  0.00288 **
## alcohol           0.353686   0.015754  22.451  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 0.7549 on 2845 degrees of freedom
## Multiple R-squared:  0.2271, Adjusted R-squared:  0.2257
## F-statistic: 167.1 on 5 and 2845 DF,  p-value: < 2.2e-16
```

```
white_predict_lm_mfs = predict.lm(white_lm_mfs, newdata = white_test_mfs)
white_error_prediction_lm_mfs = mean((white_predict_lm_mfs - white_y_test_mfs)^2)
white_error_prediction_lm_mfs
```

```
## [1] 0.5659304
```

The accuracy for white wine quality prediction with multiple regression is 43%

```
##################################################################################
# Support Vector Machine for manual Feature selection
##################################################################################
# discretize the data
med = median(new_white_wine_mfs$quality)  # 6
quality =  ifelse(new_white_wine_mfs$quality <= med, "No", "Yes")

# create a new data set
svm_white <- data.frame(new_white_wine_mfs[,-c(6)], quality)

# Divide into test and train
set.seed(1289963)
test_i = sample(1:nrow(svm_white), 1/3*nrow(svm_white))
test = svm_white[test_i, ]
train = svm_white[-test_i, ]

# SVM with a linear kernel
tune.model <- tune(svm, quality~., data = train, kernel = "linear",
                   ranges = list(cost = c(0.001, 0.01, 1, 5, 10, 100)))
tune.model # best performance: 0.2426606, cost= 0.001
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost
##  0.001
##
## - best performance: 0.2315932
```

```
summary(tune.model)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost
##  0.001
##
## - best performance: 0.2315932
##
```

```
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.2315932 0.02731168
## 2 1e-02 0.2315932 0.02731168
## 3 1e+00 0.2315932 0.02731168
## 4 5e+00 0.2315932 0.02731168
## 5 1e+01 0.2315932 0.02731168
## 6 1e+02 0.2315932 0.02731168
```

```
bestmod <- tune.model$best.model
bestmod # Number of Support Vectors:  1324
```

```
##
## Call:
## best.tune(method = svm, train.x = quality ~ ., data = train,
##     ranges = list(cost = c(0.001, 0.01, 1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.001
##       gamma:  0.2
##
## Number of Support Vectors:  1264
```

```
# predict the test data
y_hat <- predict(bestmod, newdata = test)
y_true <- test$quality
accur_lin <- length(which(y_hat == y_true))/length(y_true)
accur_lin #  0.7820324
```

```
## [1] 0.7599411
```

```
table(predict = y_hat, truth = y_true)
```

```
##        truth
## predict   No  Yes
##     No  1032  326
##     Yes    0    0
```

```
# SVM with a radial kernel
tune.model.rad <- tune(svm, quality~., data = train, kernel = "radial",
                   ranges = list(cost = c(0.001, 0.01, 1, 5, 10, 100)))
tune.model.rad # best performance: 0.2036032, cost= 10
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.2047197
```

```
summary(tune.model.rad)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.2047197
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.2315892 0.02986141
## 2 1e-02 0.2315892 0.02986141
## 3 1e+00 0.2080272 0.02628220
## 4 5e+00 0.2050860 0.03196245
## 5 1e+01 0.2047197 0.03184220
## 6 1e+02 0.2117050 0.03236955
```

```
bestmod <- tune.model.rad$best.model
bestmod # Number of Support Vectors:  1255
```

```
##
## Call:
## best.tune(method = svm, train.x = quality ~ ., data = train,
##     ranges = list(cost = c(0.001, 0.01, 1, 5, 10, 100)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##       gamma:  0.2
##
## Number of Support Vectors:  1237
```

```
# predict the test data
y_hat <- predict(bestmod, newdata = test)
y_true <- test$quality
accur_rad <- length(which(y_hat == y_true))/length(y_true)
accur_rad
```

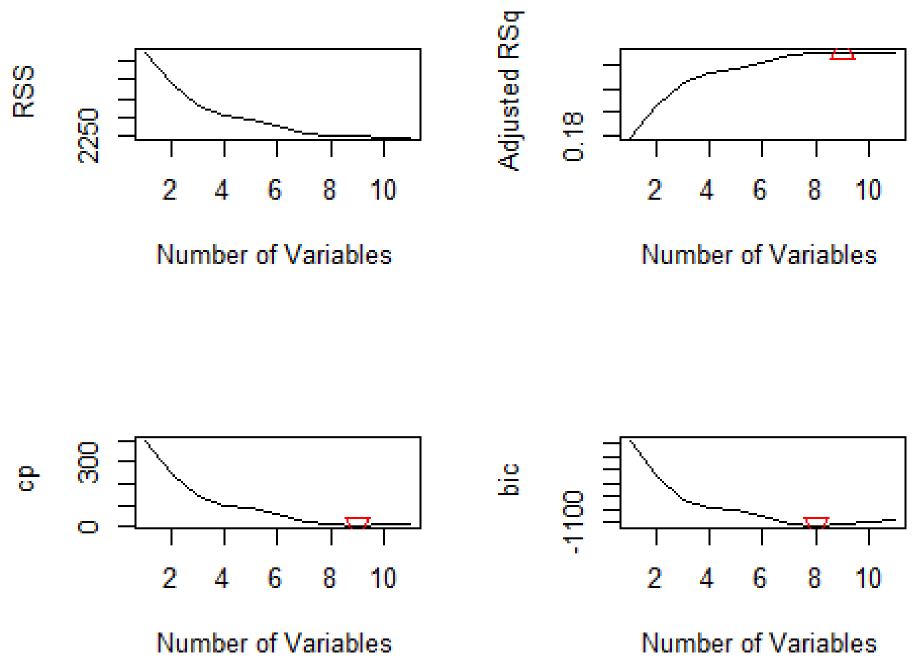
```
## [1] 0.8092784
```

And the accuracy with support vector machine is 81%.

```
table(predict = y_hat, truth = y_true)
```

```
##        truth
## predict  No Yes
##     No  996 223
##     Yes  36 103
```

```
##################################################################################
# Automatic Feature selection (best subset selection)
##################################################################################
regfit.full = regsubsets(quality~.,data = new_white_wine, nvmax = 11)
reg.summary = summary(regfit.full)
#An asterisk indicates that a given variable is included in the corresponding model.

par(mfrow =c(2,2))
plot(reg.summary$rss ,xlab=" Number of Variables ",ylab=" RSS",type="l")
plot(reg.summary$adjr2 ,xlab =" Number of Variables ",ylab=" Adjusted RSq",type="l")
max = which.max (reg.summary$adjr2) #9
points(max, reg.summary$adjr2[max], col ="red",cex =1.5, pch =24)
plot(reg.summary$cp ,xlab=" Number of Variables ",ylab=" cp",type="l")
mincp = which.min (reg.summary$cp) #9
points(mincp, reg.summary$cp[mincp], col ="red",cex =1.5, pch =25)
plot(reg.summary$bic ,xlab =" Number of Variables ",ylab=" bic",type="l")
minbic = which.min (reg.summary$bic) #8
points(minbic, reg.summary$bic[minbic], col ="red",cex =1.5, pch =25)
```



AdjR^2 and Mallow's Cp: 9-variable model

```
max = which.max (reg.summary$adjr2) #9

##################################################################################
# 10-fold CV for model selection
##################################################################################
set.seed (17)

predict.regsubsets = function(object, newdata, id, ...) {
  form = as.formula(object$call[[2]])
  mat = model.matrix(form, newdata)
  coefi = coef(object, id = id)
  mat[, names(coefi)] %*% coefi
}

k=10
set.seed (1)
folds=sample (1:k,nrow(new_white_wine),replace =TRUE)

cv.errors = matrix(NA,10,11)
for (j in 1:k) {
  best.fit = regsubsets(quality~ ., data = new_white_wine[folds != j,], nvmax = 11)
  for (i in 1:11) {
    pred = predict(best.fit, new_white_wine[folds == j, ], id = i)
    cv.errors[j, i] = mean((new_white_wine$quality[folds == j] - pred)^2)
  }
}
rmse.cv = sqrt(apply(cv.errors, 2, mean))
rmse.cv

##  [1] 0.7790783 0.7655760 0.7564845 0.7526546 0.7523505 0.7485425 0.7453616
##  [8] 0.7442417 0.7439880 0.7444069 0.7443416

which.min(rmse.cv)

## [1] 9
```

10-fold CV indicates for 9-variable model
fixed.acidity+volatile.acidity+residual.sugar+chlorides+free.sulfur.dioxide+density+pH+sulphates+alcohol

```
#################################################################################
# Comparing Training error and test error for all models
#################################################################################

###Training and test set for best subset
set.seed(100)
ww_train = sample(1:nrow(new_white_wine), round(0.75*nrow(new_white_wine)))
ww_training_data = new_white_wine[ww_train,]
ww_test_data = new_white_wine[-ww_train,]
Y.train = new_white_wine$quality[ww_train]
Y.test = new_white_wine$quality[-ww_train]

## best subset linear regression
best_subset = regsubsets(quality~.,ww_training_data,nvmax=11)
subset_summary = summary(best_subset)

a = lm(quality~.,ww_training_data)
summary(a)   # 9 significant variables

##
## Call:
## lm(formula = quality ~ ., data = ww_training_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.4529 -0.5103 -0.0466  0.4600  2.4537
##
## Coefficients:
```

```
##                        Estimate Std. Error t value Pr(>|t|)    
## (Intercept)           2.090e+02  3.198e+01   6.535 7.44e-11 ***
## fixed.acidity         1.615e-01  3.164e-02   5.104 3.52e-07 ***
## volatile.acidity     -2.005e+00  1.867e-01 -10.739  < 2e-16 ***
## citric.acid           7.692e-02  1.593e-01   0.483 0.629136    
## residual.sugar        1.038e-01  1.199e-02   8.657  < 2e-16 ***
## chlorides            -3.127e+00  1.677e+00  -1.864 0.062348 .  
## free.sulfur.dioxide   4.440e-03  1.207e-03   3.680 0.000237 ***
## total.sulfur.dioxide  5.177e-04  5.072e-04   1.021 0.307416    
## density              -2.108e+02  3.240e+01  -6.505 9.06e-11 ***
## pH                    1.085e+00  1.499e-01   7.236 5.84e-13 ***
## sulphates             7.724e-01  1.463e-01   5.279 1.39e-07 ***
## alcohol               1.198e-01  3.993e-02   2.999 0.002731 ** 
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7418 on 3044 degrees of freedom
## Multiple R-squared:  0.2575, Adjusted R-squared:  0.2548 
## F-statistic: 95.95 on 11 and 3044 DF,  p-value: < 2.2e-16
```

```
mincp = which.min(subset_summary$cp)
mincp
```

```
## [1] 9
```

```
features_selected = names(coef(best_subset,mincp))
features_selected = features_selected[-1]

#"fixed.acidity","volatile.acidity","residual.sugar","chlorides","free.sulfur.dioxide","d
ensity","pH","sulphates","alcohol"

final_ww_model = new_white_wine[,c("quality",features_selected)]
final_ww_model = final_ww_model[,-c(7)]

##################################################################################
# Create Training and Test set for final White Wine Model
##################################################################################
set.seed(789)
train = sample(1:nrow(final_ww_model), 0.75*nrow(final_ww_model))
white_train_afs = final_ww_model[train,]
white_test_afs  = final_ww_model[-train,]
white_y_train_afs = final_ww_model$quality[train]
white_y_test_afs = final_ww_model$quality[-train]

##################################################################################
# Multiple Regression for Automatic Feature selection (best subset selection)
##################################################################################
white_lm_afs = lm(quality~., data = white_train_afs) #RSE = 0.7446
white_predict_lm_afs = predict.lm(white_lm_afs, newdata = white_test_afs)
summary(white_predict_lm_afs)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   4.665   5.627   5.923   5.943   6.258   7.075
```

```
white_error_prediction_lm_afs = mean((white_predict_lm_afs - white_y_test_afs)^2)
white_error_prediction_lm_afs
```

```
## [1] 0.5771251
```

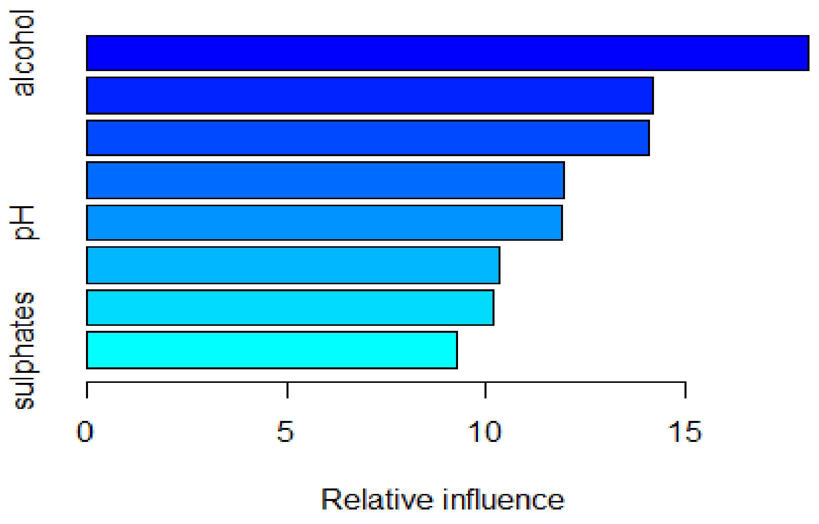
The error in multiple regression is 0.5771251

```
#####################################################################################
# Decision Trees
#####################################################################################

##########Regression Tree################
model.control = rpart.control(minsplit = 5,xval = 10,cp=0)
reg_tree = rpart(quality~.,data = white_train_afs,method = "anova",control = model.contro
l)
pred = predict(reg_tree, white_test_afs)
mean((pred-white_y_test_afs)^2)

## [1] 0.6452813

# Pruning the Regression Tree
min_cp=which.min(reg_tree$cptable[,4])
prune_fit = prune(reg_tree,cp=reg_tree$cptable[min_cp,1])

# Test MSE for pruned tree
yhat = predict(prune_fit, white_test_afs)
reg_tree_err = mean((yhat-white_y_test_afs)^2)
reg_tree_err

## [1] 0.5439929
```

The error for regression tree is 64% before pruning and becomes 55% after pruning. Pruning reduces the complexity of the final classifier, and hence improves predictive accuracy by the reduction of overfitting.

```
#######Classification Tree#############
table(final_ww_model$quality)

##
##    3    4    5    6    7    8    9
##   10   92 1134 1883  802  149    4

# 1 - Bad, 2 - Good
final_ww_model$wine_quality[final_ww_model$quality<=6 ] = 1
final_ww_model$wine_quality[final_ww_model$quality>=7] = 2
final_ww_model$wine_quality = as.factor(final_ww_model$wine_quality)

table(final_ww_model$wine_quality)

##
##    1    2
## 3119  955

# taking density out
Classification_model = final_ww_model[,c("wine_quality",features_selected[-6])]

# Divide the classification model into training and test set
set.seed(100)
classif_train = sample(1:nrow(Classification_model), round(0.75*nrow(Classification_model
)))
classif_training_data = Classification_model[classif_train,]
```

```
classif_test_data = Classification_model[-classif_train,]
Y.train = Classification_model$wine_quality[classif_train]
Y.test = Classification_model$wine_quality[-classif_train]

model.control = rpart.control(minsplit = 5,xval = 10,cp=0)
wine_tree_fit = rpart(wine_quality~., data =classif_training_data,method = "class",contro
l = model.control)

## Predict with test data
table(predict(wine_tree_fit, classif_test_data, type = "class"),Y.test)

##    Y.test
##       1   2
##   1 710  89
##   2  96 123

# Pruning the Classification tree
min_cp=which.min(wine_tree_fit$cptable[,4])
prune_fit = prune(wine_tree_fit,cp=wine_tree_fit$cptable[min_cp,1])

# Prediction with pruned tree
yhat_class = predict(prune_fit, classif_test_data)
class_tree_err = mean((yhat_class-as.numeric(classif_test_data$wine_quality))^2)
class_tree_err

## [1] 0.8045777

###############################################################################
# Boosting for regression tree
##############################################################################
set.seed(1)
par(mfrow =c(1,1))
reg_boost = gbm(quality~.,data=white_train_afs, distribution = "gaussian",
                n.trees = 5000,interaction.depth = 4,shrinkage = 0.1,
                verbose = F)
summary(reg_boost)
```

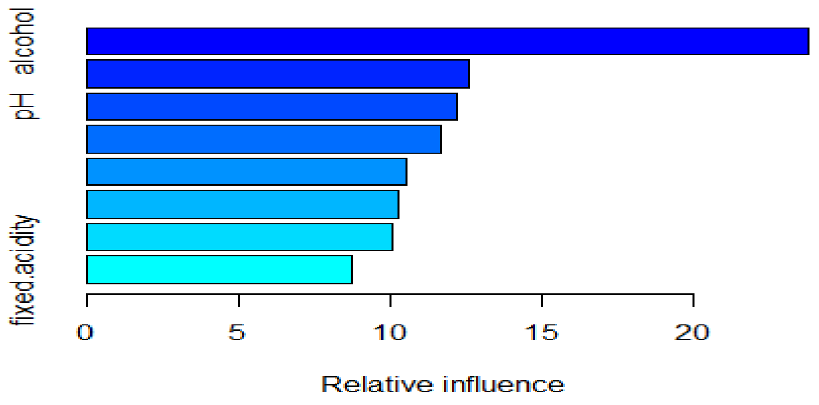


- Good thing about boosting - shows relative importance of the variables towards the response
- Sequentially applies the weak classification algorithm to repeatedly modified versions of the data, thereby producing a powerful model
- In regression, while predicting the quality alcohol, free SO2 and residual sugar are most important.

```
##                                       var   rel.inf
## alcohol                           alcohol 18.081948
## free.sulfur.dioxide free.sulfur.dioxide 14.186929
## residual.sugar             residual.sugar 14.058282
## volatile.acidity         volatile.acidity 11.937945
## pH                                     pH 11.923468
## chlorides                       chlorides 10.343779
## fixed.acidity               fixed.acidity 10.180910
## sulphates                       sulphates  9.286738

yhat.boost = predict(reg_boost, newdata = white_test_afs,n.trees = 5000)
reg_boost_err = mean((yhat.boost-white_y_test_afs)^2)
reg_boost_err

## [1] 0.4458032
```

Error for boosting regression is 44%

```
################################################################################
# Boosting for Classification tree
###############################################################################
set.seed(1)
# 0 - bad,1-good
boost_training = classif_training_data
boost_training$wine_quality = as.numeric(classif_training_data$wine_quality)-1
boost_test = classif_test_data
boost_test$wine_quality = as.numeric(classif_test_data$wine_quality)-1


class_boost = gbm(wine_quality~.,data=boost_training,distribution = "bernoulli",
                  n.trees = 5000,interaction.depth = 4,shrinkage = 0.1)

summary(class_boost)
```



- In classification, while predicting the class of quality alcohol, residual sugar and pH are most important.

- For best wine, the percentage of alcohol and the taste (pH) really matters

```
##                                   var   rel.inf
## alcohol                       alcohol 23.812213
## residual.sugar         residual.sugar 12.629944
## pH                                 pH 12.210719
## volatile.acidity     volatile.acidity 11.694584
## free.sulfur.dioxide free.sulfur.dioxide 10.558240
## chlorides                   chlorides 10.281417
## sulphates                   sulphates 10.052990
## fixed.acidity           fixed.acidity  8.759892
```

```
class_pred = predict(class_boost,newdata = boost_test,n.trees = 5000,type = "response")
class_boost_err = mean((class_pred - boost_test$wine_quality)^2)
class_boost_err

## [1] 0.1207259
```

Error for boosting classification is 12%

```
###############################################################################
# Support Vector Machine
##############################################################################

# Divide the classification model into training and test set
set.seed(100)
classif_train = sample(1:nrow(Classification_model), round(0.75*nrow(Classification_model
)))
classif_training_data = Classification_model[classif_train,]
classif_test_data = Classification_model[-classif_train,]
Y.train = Classification_model$wine_quality[classif_train]
Y.test = Classification_model$wine_quality[-classif_train]

# SVM with a linear kernel
# we use tune() function to perform cross validation - by default 10 fold CV
tune.model <- tune(svm, wine_quality~., data = classif_training_data, kernel = "linear",
                   ranges = list(cost = c(0.01, 0.1, 1, 5, 10, 100)))
tune.model # cost = 0.01 which is the lowest cross validation error , best performance =
0.2431265

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.2431265

summary(tune.model)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.2431265
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-02 0.2431265 0.01160039
## 2 1e-01 0.2431265 0.01160039
## 3 1e+00 0.2431265 0.01160039
## 4 5e+00 0.2431265 0.01160039
## 5 1e+01 0.2431265 0.01160039
## 6 1e+02 0.2431265 0.01160039
```

```
bestmod <- tune.model$best.model
bestmod
```

```
##
## Call:
## best.tune(method = svm, train.x = wine_quality ~ ., data = classif_training_data,
##     ranges = list(cost = c(0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.125
##
## Number of Support Vectors:  1511
```

```
# Number of Support Vectors:  1511
# predict the test data
y_hat <- predict(bestmod, newdata = classif_test_data)
y_true <- Y.test
accur_lin <- length(which(y_hat == y_true))/length(y_true)
accur_lin #  0.7917485
```

```
## [1] 0.7917485
```

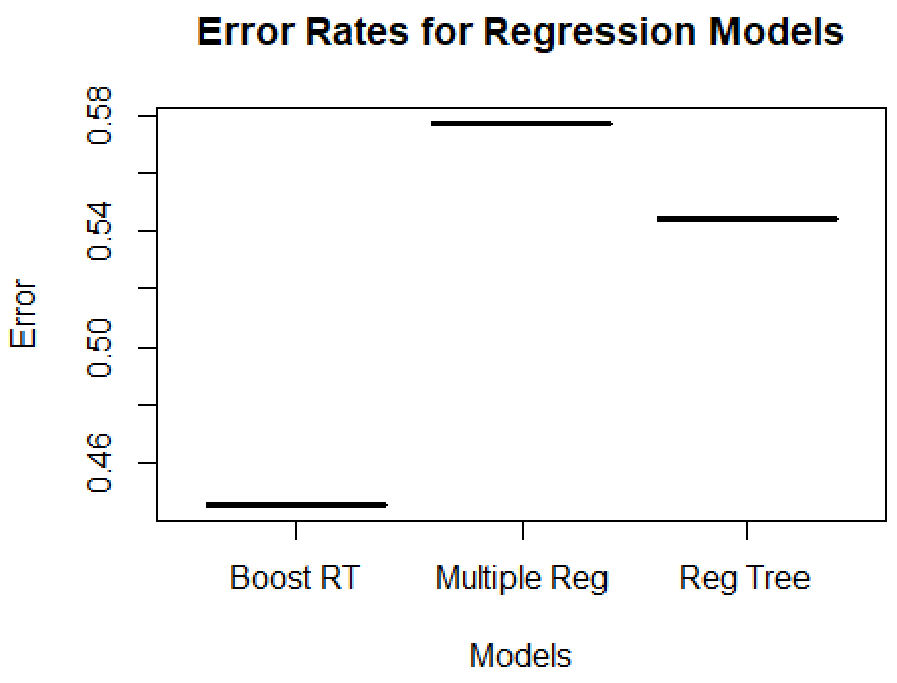
```
table(predict = y_hat, truth = y_true)
```

```
##        truth
## predict   1   2
##       1 806 212
##       2   0   0
```

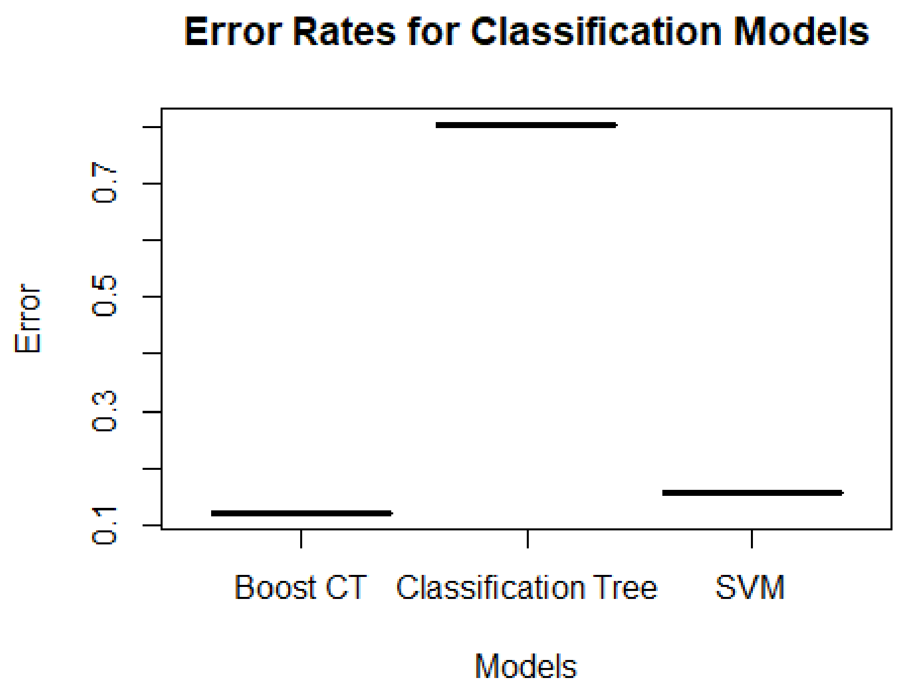
```
##################################################################################
# SVM with a radial kernel
##################################################################################
# we use tune() function to perform cross validation - by default 10 fold CV
```

```
tune.model.rad <- tune(svm, wine_quality~., data = classif_training_data, kernel = "radia
l",ranges = list(cost = c(0.01, 0.1, 1, 5, 10, 100)))

tune.model.rad # cost = 10 which is the lowest cross validation error , best performance
= 0.1960077

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.1960077

summary(tune.model.rad)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.1960077
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-02 0.2431137 0.01470219
## 2 1e-01 0.2254452 0.01730346
## 3 1e+00 0.2051688 0.01924360
## 4 5e+00 0.1966656 0.02225291
## 5 1e+01 0.1960077 0.01859208
## 6 1e+02 0.1979856 0.01674587

bestmod <- tune.model.rad$best.model
bestmod

##
## Call:
## best.tune(method = svm, train.x = wine_quality ~ ., data = classif_training_data,
##     ranges = list(cost = c(0.01, 0.1, 1, 5, 10, 100)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##       gamma:  0.125
##
## Number of Support Vectors:  1429
```

```
# Number of Support Vectors:  1429
# predict the test data
y_hat <- predict(bestmod, newdata = classif_test_data)
y_true <- Y.test
accur_rad <- length(which(y_hat == y_true))/length(y_true)
accur_rad #  0.8447937

## [1] 0.8447937

svm_err = 1 - accur_rad

###############################################################################
# Error and Accuracy Plotting
###############################################################################

reg_error = c(white_error_prediction_lm_afs,reg_tree_err,reg_boost_err)
reg_models = c("Multiple Reg", 'Reg Tree', "Boost RT")
reg_err_tb = data.frame(reg_models,reg_error)
plot(reg_err_tb$reg_models,reg_err_tb$reg_error,main = 'Error Rates for Regression Models
', xlab = "Models", ylab = "Error")
```



```
class_error = c(class_tree_err,class_boost_err,svm_err)
class_models = c("Classification Tree","Boost CT", "SVM")
class_err_tb = data.frame(class_models,class_error)
plot(class_err_tb$class_models,class_err_tb$class_error,main = 'Error Rates for Classific
ation Models', xlab = "Models", ylab = "Error")
```



In both regression and classification, boosting gives less error.

Disadvantage of SVM: Speed and size, both in training and testing data that can increase the computational cost

Accuracy of SVM – 84% whereas for model selected manually it was 81%