Vivino Marketing Strategy Project

Avadhoot Jadhav
Ta-Chin (Meek) Ho
Alefiya Lunawadawala
Mark Nevelle Antony

Stuart School of Business, Illinois Institute of Technology
MAX 503-01: Marketing Research and Engineering
Dr. Ghazale Haddadian

April 22, 2024

Vivino Marketing Strategy Report

In this report, we detail our strategies and models using R Studio for Vivino, a company that specializes in selling French "Bordeaux" red wine. Our primary objective was to streamline their data collection and analysis process, and then predict the "good" quality wine based on its physicochemical properties. We will develop two classifier models, the Naive Bayes Model and the Random Forest Model, for comparison. Additionally, we will determine the top three discriminating physicochemical properties of wine quality, which will be helpful for sales tracking, inventory management, product promotion, and quality control evaluation.

- **Upload the data :**

```
getwd()

setwd("C:/Users/Smush/Desktop")

vivino.df <- read.csv("vivino.csv")
```

- **Find the dimensions of this data set**:

```
dim(vivino.df)

str(vivino.df)
```

In the “vivino” data frame, we have 1599 observations of 13 variables.

- **Summarize the data and showcase the structure of this dataset.**

```
summary(vivino.df)
```

```
fixed.acidity   volatile.acidity  citric.acid    residual.sugar     chlorides
 Min.   : 4.60   Min.   :0.1200   Min.   :0.000   Min.   : 0.900   Min.   :0.01200
 1st Qu.: 7.10   1st Qu.:0.3900   1st Qu.:0.090   1st Qu.: 1.900   1st Qu.:0.07000
 Median : 7.90   Median :0.5200   Median :0.260   Median : 2.200   Median :0.07900
 Mean   : 8.32   Mean   :0.5278   Mean   :0.271   Mean   : 2.539   Mean   :0.08747
 3rd Qu.: 9.20   3rd Qu.:0.6400   3rd Qu.:0.420   3rd Qu.: 2.600   3rd Qu.:0.09000
 Max.   :15.90   Max.   :1.5800   Max.   :1.000   Max.   :15.500   Max.   :0.61100
 free.sulfur.dioxide total.sulfur.dioxide    density             pH          sulphates
 Min.   : 1.00       Min.   :  6.00       Min.   :0.9901   Min.   :2.740   Min.   :0.3300
```

```
1st Qu.: 7.00       1st Qu.: 22.00       1st Qu.:0.9956   1st Qu.:3.210   1st Qu.:0.5500
Median :14.00       Median : 38.00       Median :0.9968   Median :3.310   Median :0.6200
Mean   :15.87       Mean   : 46.47       Mean   :0.9967   Mean   :3.311   Mean   :0.6581
3rd Qu.:21.00       3rd Qu.: 62.00       3rd Qu.:0.9978   3rd Qu.:3.400   3rd Qu.:0.7300
Max.   :72.00       Max.   :289.00       Max.   :1.0037   Max.   :4.010   Max.   :2.0000
    alcohol          quality      qualityLabel
Min.   : 8.40   Min.   :3.000   0:1382
1st Qu.: 9.50   1st Qu.:5.000   1: 217
Median :10.20   Median :6.000
Mean   :10.42   Mean   :5.636
3rd Qu.:11.10   3rd Qu.:6.000
Max.   :14.90   Max.   :8.000
```

- **Investigate the correlation between the features using corrplot. Based on your results, Which dimensions are relatively highly correlated?**

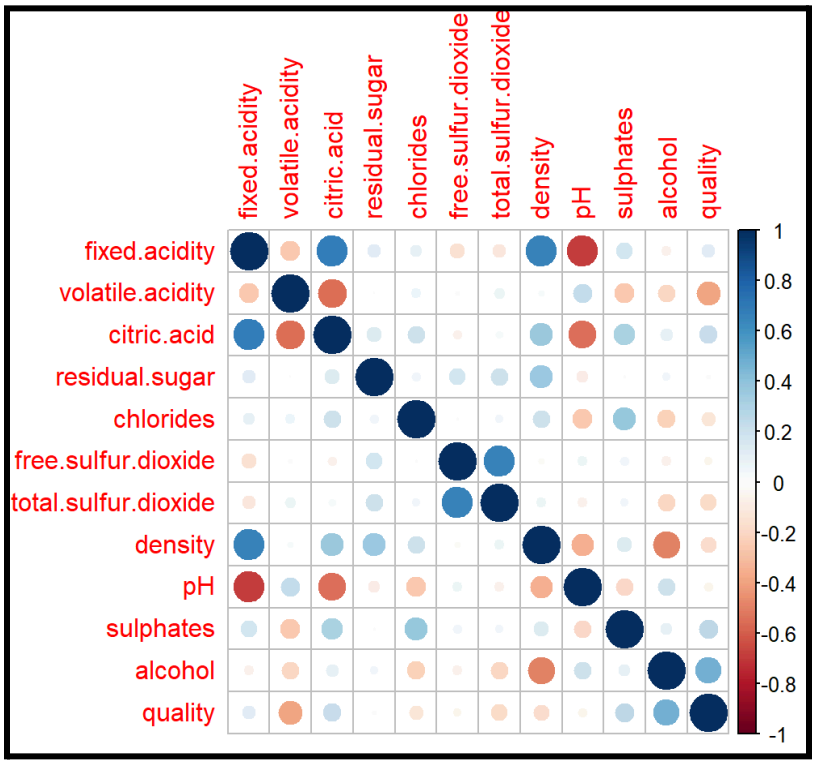
corrplot(cor(vivino.df), method = "circle")

Based on the corrplot, there are 3 dimensions which are relatively highly corrrelated :

1.fixed acidity & citric acidity

2.density & fixed acidity

3.total.sulfur.dioxide & free.sulfur.dioxide



- **Create a new column to indicate if a wine is good or bad (if quality score > 6 consider it as a "good" wine, otherwise "bad" wine)**

```
vivino.df$qualityLabel <- as.factor(ifelse(vivino.df$quality > 6, 1, 0))
```

```
  [1] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good" "good" "bad"  "bad"  "bad"  "bad"
 [14] "bad"  "bad"  "bad"  "good" "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
 [27] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good" "bad"
 [40] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
 [53] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good" "bad"  "bad"
 [66] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
 [79] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
 [92] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[105] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[118] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good" "bad"
[131] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[144] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[157] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[170] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[183] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[196] "bad"  "bad"  "bad"  "good" "bad"  "good" "bad"  "bad"  "bad"  "bad"  "good" "good" "bad"
[209] "bad"  "good" "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[222] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good" "bad"  "bad"  "bad"
[235] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good" "good" "bad"  "bad"
[248] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good"
[261] "bad"  "bad"  "bad"  "bad"  "bad"  "good" "bad"  "good" "bad"  "bad"  "bad"  "bad"  "bad"
[274] "bad"  "bad"  "bad"  "bad"  "bad"  "good" "good" "bad"  "good" "bad"  "good" "bad"  "bad"
[287] "bad"  "bad"  "good" "bad"  "good" "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[300] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[313] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good" "bad"  "good" "bad"  "bad"  "bad"  "bad"
[326] "bad"  "good" "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good" "good" "bad"  "bad"
[339] "bad"  "good" "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good" "bad"  "bad"  "bad"  "bad"
[352] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good" "good" "bad"  "bad"  "bad"  "bad"  "bad"
[365] "good" "bad"  "good" "bad"  "bad"  "good" "bad"  "bad"  "bad"  "bad"  "bad"  "good" "bad"
[378] "good" "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good"
[391] "good" "bad"  "bad"  "bad"  "bad"  "good" "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[404] "bad"  "bad"  "bad"  "bad"  "good" "bad"  "bad"  "bad"  "bad"  "bad"  "good" "bad"  "bad"
[417] "bad"  "bad"  "bad"  "bad"  "good" "good" "bad"  "good" "bad"  "good" "bad"  "bad"  "bad"
[430] "bad"  "good" "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good" "bad"
[443] "good" "good" "good" "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good" "bad"
[456] "good" "bad"  "bad"  "good" "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[469] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[482] "good" "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good" "bad"  "bad"  "good" "good" "bad"
[495] "bad"  "good" "bad"  "bad"  "good" "bad"  "bad"  "good" "good" "good" "good" "good" "good"
[508] "bad"  "bad"  "good" "bad"  "bad"  "bad"  "good" "good" "bad"  "bad"  "bad"  "bad"  "bad"
[521] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[534] "bad"  "bad"  "bad"  "bad"  "bad"  "good" "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
```

```
[547] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[560] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[573] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good" "good"
[586] "bad"  "good" "bad"  "good" "good" "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[599] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good" "bad"  "bad"  "bad"  "bad"
[612] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[625] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[638] "bad"  "good" "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good" "bad"  "bad"  "good" "bad"
[651] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good" "bad"  "bad"  "bad"  "bad"  "bad"
[664] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[677] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[690] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[703] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[716] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[729] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[742] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[755] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[768] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[781] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[794] "bad"  "bad"  "bad"  "bad"  "good" "bad"  "bad"  "bad"  "bad"  "good" "bad"  "bad"  "good"
[807] "good" "good" "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[820] "bad"  "bad"  "good" "bad"  "bad"  "bad"  "bad"  "good" "bad"  "good" "bad"  "bad"  "bad"
[833] "bad"  "bad"  "bad"  "bad"  "good" "good" "good" "bad"  "good" "bad"  "bad"  "bad"  "bad"
[846] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good" "bad"  "good"
[859] "good" "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[872] "bad"  "bad"  "good" "good" "good" "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[885] "bad"  "bad"  "bad"  "good" "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good"
[898] "bad"  "good" "bad"  "bad"  "good" "good" "good" "good" "bad"  "bad"  "bad"  "bad"  "bad"
[911] "bad"  "bad"  "bad"  "good" "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[924] "bad"  "bad"  "good" "bad"  "bad"  "bad"  "good" "bad"  "bad"  "bad"  "bad"  "bad"  "bad"
[937] "bad"  "bad"  "good" "bad"  "good" "good" "good" "good" "good" "good" "good" "good" "good"
[950] "good" "good" "good" "good" "good" "bad"  "bad"  "bad"  "bad"  "good" "bad"  "bad"  "bad"
[963] "bad"  "bad"  "bad"  "bad"  "good" "bad"  "bad"  "bad"  "bad"  "bad"  "good" "bad"  "good"
[976] "bad"  "bad"  "bad"  "good" "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good" "bad"
[989] "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "bad"  "good" "good" "bad"  "bad"
[ reached getOption("max.print") -- omitted 599 entries ]
```

Predicting Good Wine Quality Based on Physicochemical Properties

This study seeks to improve data collection and analysis to predict wine quality using physicochemical properties. We will apply and compare the Naive Bayes and Random Forest models to assess wine quality. Additionally, we will determine the three most impactful physicochemical properties affecting wine quality.

Dividing data into training and test cases:

```
set.seed(04625)
train.prop   <- 0.65
train.cases  <- sample(nrow(vivino.df), nrow(vivino.df)*train.prop)
train.cases
vivino.df.train <- vivino.df[ train.cases, ]
vivino.df.test  <- vivino.df[-train.cases, ]
```

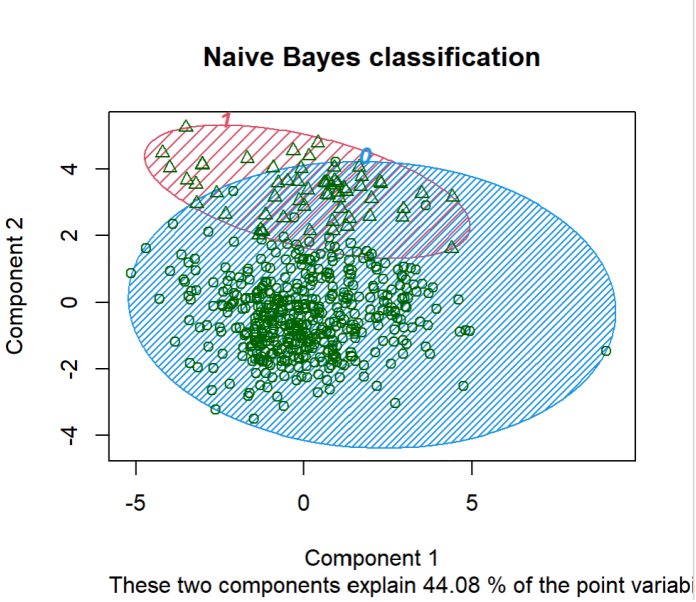
Naive Bayes model

```
library(e1071)
(vivino.nb<-naiveBayes(qualityLabel~.,data = vivino.df.train))

#Predicting Segment Membership
vivino.nb.class<-predict(vivino.nb,vivino.df.test)
vivino.nb.class
prop.table(table(vivino.nb.class))

#Plot the predicted classes
library(cluster)
clusplot(vivino.df.test, vivino.nb.class, color=TRUE, shade=TRUE,
        labels=4,
        main="Naive Bayes classification")

#How well did we predict in the test data?
```

```
mean(vivino.df.test$qualityLabel==vivino.nb.class)
library(mclust)
adjustedRandIndex(vivino.nb.class,vivino.df.test$qualityLabel)

#Confusion Matrix
table(vivino.nb.class,vivino.df.test$qualityLabel)
```

Result :

```
> prop.table(table(vivino.nb.class))
vivino.nb.class
      0         1
0.8910714 0.1089286
> clusplot(vivino.df.test, vivino.nb.class, color=TRUE, shade=TRUE,
       labels=4,
       main="Naive Bayes classification")
```




```
> adjustedRandIndex(vivino.nb.class,vivino.df.test$qualityLabel)
[1] 0.9239226
> table(vivino.nb.class,vivino.df.test$qualityLabel)
```

	0	1
0	492	7
1	0	61

❖ **Predictive modeling.**

Setting Seed: It initializes a random number generator to ensure reproducibility.

Train-Test Split: It randomly selects a proportion (train.prop) of the dataset for training (**65%** in this case) and the remainder for testing.

Purpose: This process allows for the evaluation of model performance on unseen data. The training set is used to build the model, while the test set is used to assess its accuracy and generalization ability.

Usage: It's crucial for machine learning tasks to prevent overfitting and accurately assess model performance before deploying it in real-world scenarios.

❖ **Train the model:**

```
(vivino.nb<-naiveBayes(qualityLabel~.,data = vivino.df.train)):
```

"Trains a Naive Bayes model (vivino_nb) using the dataset vivino.df. The formula [qualityLabel ~ .]specifies that qualityLabel is the target variable, and the model should use all other columns in the dataset as features."

❖ **Model Evaluation:**

Predictions:

```
vivino.nb.class<-predict(vivino.nb,vivino.df.test)

vivino.nb.class

prop.table(table(vivino.nb.class))
```

"This code predicts the classes of instances in the test dataset using a trained Naive Bayes classifier. It helps assess the classifier's performance and its ability to generalize to new data."

Cluster:

```
library(cluster)

clusplot(vivino.df.test, vivino.nb.class, color=TRUE, shade=TRUE,  labels=4,  main="Naive Bayes classification"):
```

“This code uses the clusplot function from the cluster package in R to visualize the clustering results of a Naive Bayes classification performed on a dataset (vivino.df.test). It helps in assessing how well the classifier separates the data into distinct groups based on its features.”

adjustedRandIndex:

```
mean(vivino.df.test$qualityLabel==vivino.nb.class)

adjustedRandIndex(vivino.nb.class,vivino.df.test$qualityLabel)
```

“This code calculates accuracy and the Adjusted Rand Index to evaluate how well a classification model, likely a clustering algorithm, matches predicted labels with true labels.”

Confusion Matrix:

```
table(vivino.nb.class,vivino.df.test$qualityLabel):
```

“This code generates a confusion matrix to assess the performance of a classification model. It compares the predicted class labels (vivino.nb.class) with the actual class labels (vivino.df.test$qualityLabel). The confusion matrix helps in understanding the distribution of correct and incorrect predictions across different classes.”

Result :

The Naive Bayes model is used for its simplicity, efficiency, and effectiveness in classification tasks, especially when working with large datasets and multiple features. It assumes independence among features, making computations straightforward.The results suggest that a classification model has achieved high accuracy.

The model's predictions show that it's quite confident in its classifications. It assigns about 89% of the instances to bad wine (0) and about 11% to good wine (1).

The Adjusted Rand Index, which measures the similarity between two sets of data, indicates a strong agreement (92.39%) between the model's predictions and the actual quality labels of the wines.

The confusion matrix reveals that the model has correctly identified most instances, with only a few misclassifications. It accurately classified 492 instances of bad wines and 61 instances of good wines , with only 7 instances of bad wines and none of good wines being misclassified.

Overall, these results indicate that the model performs well in predicting wine quality labels based on the features it was trained on.ween 'good' and 'bad' wines.

Random Forest model

```
library(randomForest)
set.seed(98040)
vivino.rf<-randomForest(qualityLabel~.,data=vivino.df.train,
                        ntree=3000)
vivino.rf

#Making Predictions
library(cluster)
(vivino.rf.class<-predict(vivino.rf,vivino.df.test))
clusplot(vivino.df.test, vivino.rf.class, color=TRUE, shade=TRUE,
       labels=4, main="Random Forest")

#Individual prediction Probabilities
(vivino.rf.class.all <- predict(vivino.rf, vivino.df.test, predict.all=TRUE))

#How well did we predict?
mean(vivino.df.test$qualityLabel ==vivino.rf.class)
library(mclust)
adjustedRandIndex(vivino.df.test$qualityLabel , vivino.rf.class)

#Confusion Matrix
table(vivino.df.test$qualityLabel,vivino.rf.class)
```

Result :

```
> vivino.rf<-randomForest(qualityLabel~.,data=vivino.df.train,
                          ntree=3000)
>vivino.rf
```

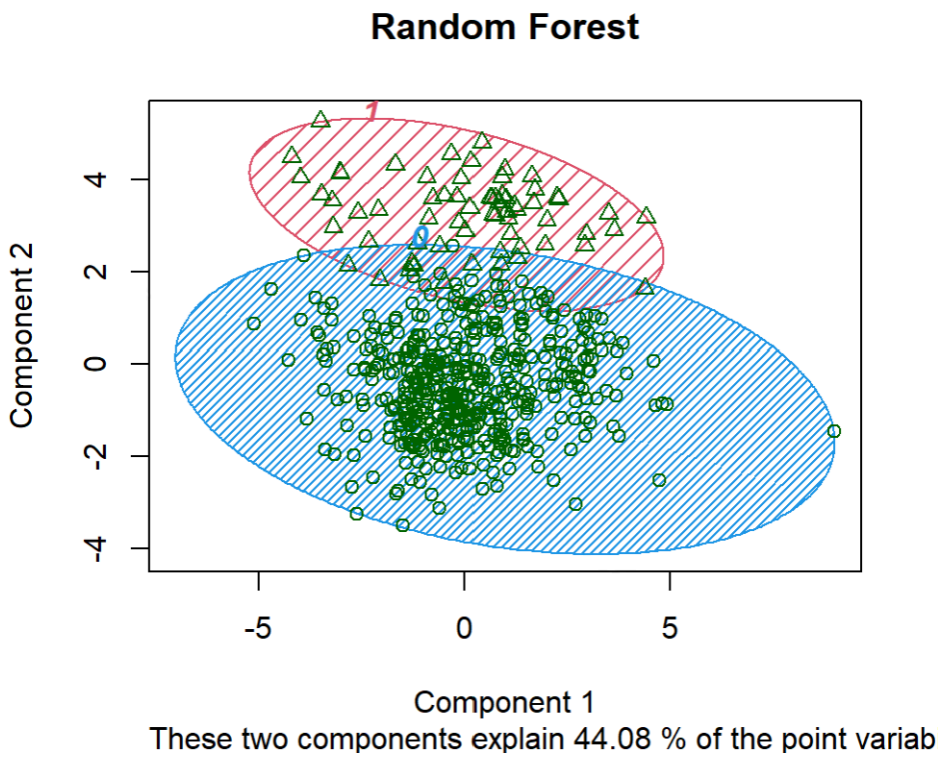
No. of variables tried at each split: 3

OOB estimate of error rate: 0%

Confusion matrix:

	0	1	class.error
0	**890**	**0**	**0**
1	**0**	**149**	**0**

```
> clusplot(vivino.df.test, vivino.rf.class, color=TRUE, shade=TRUE,  labels=4, main="Random
Forest")
```




```
> mean(vivino.df.test$qualityLabel ==vivino.rf.class)
```

[1] 1

```
> library(mclust)
```

```
> adjustedRandIndex(vivino.df.test$qualityLabel , vivino.rf.class)
[1] 1
> table(vivino.df.test$qualityLabel,vivino.rf.class)
  vivino.rf.class
     0   1
 0 492   0
 1   0  68
```

- **Set Seed**: The set.seed(98040) function sets the random number generator's seed to ensure reproducibility of the results. This means the same code will produce the same results each time you run it.
- **Train the model:**

vivino.rf<-randomForest(qualityLabel~.,data=vivino.df.train, ntree=3000) :

"This code trains a Random Forest model in R to predict wine quality labels using all available features from the training dataset and specifies 3000 trees for the forest."

- **Model Evaluation:**

Cluster:

(vivino.rf.class<-predict(vivino.rf,vivino.df.test))

clusplot(vivino.df.test, vivino.rf.class, color=TRUE, shade=TRUE, labels=4, main="Random Forest"):

"This code predicts wine quality labels using a Random Forest model (vivino.rf) on the test dataset (vivino.df.test). Then, it creates a cluster plot (clusplot) to visualize the predicted classes. This helps to understand how well the model separates different quality labels."

Prediction:

(vivino.rf.class.all <- predict(vivino.rf, vivino.df.test, predict.all=TRUE))

“This code predicts wine quality labels for all observations in the test dataset using the trained Random Forest model (vivino.rf). The addition of predict.all=TRUE ensures that predictions for all observations are returned. This is useful for analyzing the predictions comprehensively across the entire test dataset.”

adjustedRandIndex:

mean(vivino.df.test$qualityLabel ==vivino.rf.class)

adjustedRandIndex(vivino.df.test$qualityLabel , vivino.rf.class)

“This code assesses the accuracy and similarity of the Random Forest model's predictions to the true quality labels in the test dataset.”

Confusion Matrix:

table(vivino.df.test$qualityLabel,vivino.rf.class):

“This code creates a confusion matrix to compare the true quality labels (vivino.df.test$qualityLabel) with the predicted labels (vivino.rf.class). It helps to understand how well the model's predictions align with the actual labels, summarizing the classification performance in a concise way.”

Result :

The Random Forest model is chosen for its ability to handle complex datasets and its robustness against overfitting. It works by creating multiple decision trees and combining their predictions for better accuracy. The model's accuracy and adjusted Rand Index both score a perfect 1, indicating flawless agreement between the predicted and actual quality labels in the test dataset.

The confusion matrix reveals that the model has correctly identified most instances, with only a few misclassifications. It accurately classified 492 instances of bad wines and 68 instances of good wines , with none of good / bad wines being misclassified.

Determine the top discriminating factors of wine quality

```
set.seed(98040)
(vivino.imp<-randomForest(qualityLabel~.,data = vivino.df.train,
                          ntree=3000,
                          importance=TRUE))
importance(vivino.imp)
varImpPlot(vivino.imp,main = "Variable importance")

#heatmap for variable importance
library(gplots)
library(RColorBrewer)

#heatmap for variable importance
library(gplots)
library(RColorBrewer)
heatmap.2(t(importance(vivino.imp)[,1:2]),
        col = brewer.pal(5,"Reds"),trace ="none",key = "FALSE",dend="none",
        cexCol = 1.3,cexRow = 1.3,offsetRow = 0.3,offsetCol = 0.3,
        main = "\n\n\n Variable Importance for category")
```

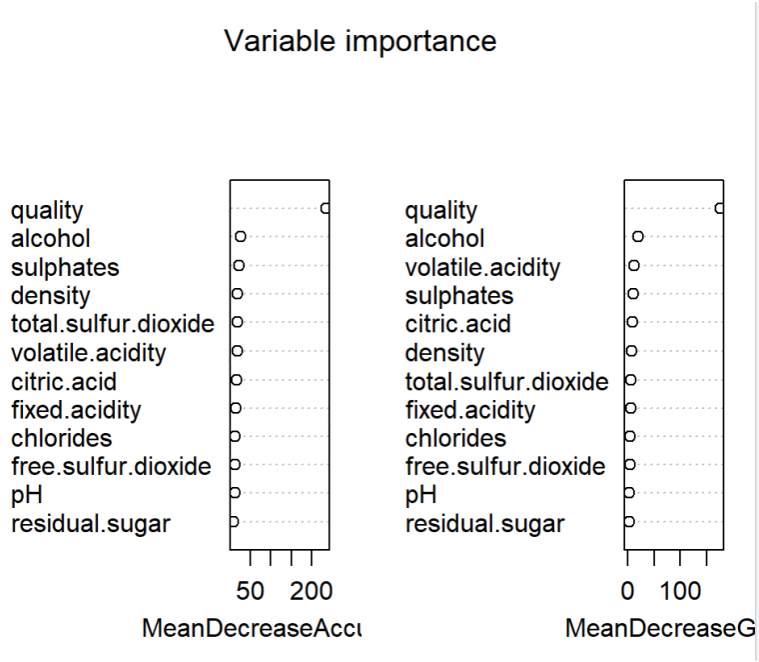
Result :

```
>
> importance(vivino.imp)
```

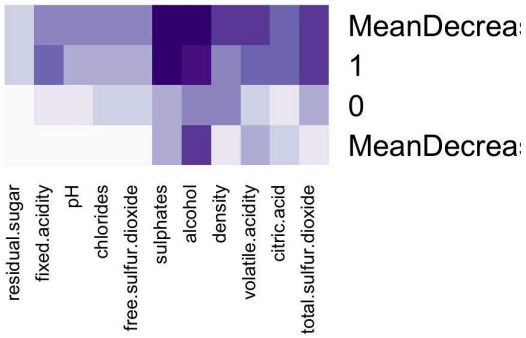
	0	1	MeanDecreaseAccuracy	MeanDecreaseGini
fixed.acidity	**8.007780**	**16.208316**	**16.46247**	**4.742803**
volatile.acidity	**9.077953**	**18.030490**	**19.07690**	**12.131762**
citric.acid	**8.729609**	**15.360602**	**16.72617**	**8.129588**
residual.sugar	**6.897749**	**8.598875**	**10.65798**	**2.562962**
chlorides	**10.694945**	**10.916354**	**14.55973**	**4.148112**
free.sulfur.dioxide	**10.568532**	**10.553219**	**14.52701**	**3.339483**

total.sulfur.dioxide	**11.263311**	**18.054084**	**19.13629**	**5.487624**
density	**15.310059**	**13.517161**	**19.99896**	**6.572632**
pH	**7.293006**	**12.791194**	**13.69738**	**2.942963**
sulphates	**11.200866**	**22.726632**	**23.89871**	**10.323609**
alcohol	**15.686214**	**22.616247**	**26.61553**	**18.779362**
quality	**182.045280**	**272.774833**	**234.32422**	**176.093390**

```
> varImpPlot(vivino.imp,main = "Variable importance")
```




```
>
> heatmap.2(t(importance(vivino.imp)[,1:2]),
        col = brewer.pal(5,"Reds"),trace ="none",key = "FALSE",dend="none",
        cexCol = 1.3,cexRow = 1.3,offsetRow = 0.3,offsetCol = 0.3,
        main = "\n\n\n Variable Importance for category")
```



❖ **Model Evaluation**

```
set.seed(98040)
(vivino.imp<-randomForest(qualityLabel~.,data = vivino.df.train,
                          ntree=3000,
                          importance=TRUE))
importance(vivino.imp)
varImpPlot(vivino.imp,main = "Variable importance")
```

Process:

This code trains a Random Forest model (vivino.imp) to predict wine quality labels using all features from the training dataset (vivino.df.train). Here's a step-by-step explanation:

1. set.seed(98040): Sets the random seed to ensure reproducibility of the results.
2. randomForest(...): Trains the Random Forest model with the following specifications:
    - qualityLabel~.: Predicts wine quality labels using all available features.
    - data = vivino.df.train: Uses the training dataset.
    - ntree=3000: Specifies 3000 trees in the forest.
    - importance=TRUE: Computes variable importance.
3. importance(vivino.imp): Computes the importance of variables in the trained Random Forest model.
4. varImpPlot(vivino.imp, main = "Variable importance"): Plots the variable importance, showing the contribution of each feature to the model's predictive performance.

Purpose:

The purpose of this code is to train a Random Forest model to predict wine quality labels and to identify which features are most influential in making these predictions.

❖ **Heatmap**

```
#heatmap for variable importance
library(gplots)
library(RColorBrewer)
heatmap.2(t(importance(vivino.imp)[,1:2]),
        col = brewer.pal(5,"Reds"),trace ="none",key = "FALSE",dend="none",
        cexCol = 1.3,cexRow = 1.3,offsetRow = 0.3,offsetCol = 0.3,
        main = "\n\n\n Variable Importance for category")
```

Process:

heatmap.2(...): Generates the heatmap with the following specifications:

t(importance(vivino.imp)[,1:2]): Transposes and selects the first two columns of the variable importance matrix.

col = brewer.pal(5,"Reds"): Sets the color palette for the heatmap using the Reds color scheme from Brewer palettes.

trace = "none": Suppresses the display of trace lines.

key = "FALSE": Hides the color key.

dend = "none": Suppresses dendrograms.

cexCol, cexRow: Adjusts the size of column and row labels.

offsetRow, offsetCol: Offsets the positions of row and column labels.

main = "\n\n\n Variable Importance for category": Sets the main title of the heatmap.

Results

Top 3 Features:

The top 3 discriminating factors of wine quality based on importance in the Random Forest model are **alcohol, sulphates, and volatile.acidity**. These features have the highest importance values, indicating they contribute most to the model's decision-making process.

Conclusion

The Random Forest model uses multiple decision trees and achieved perfect accuracy (100%) in classifying wines as 'bad' or 'good', The confusion matrix shows that it correctly classified all samples as either 'bad' (492) or 'good' (68) quality wines with no mistakes.

The Naive Bayes model is simple and efficient, with 92.39% accuracy.It accurately classified 492 instances of bad wines and 61 instances of good wines , making only 7 mistakes.. Both models have classified all the samples with great accuracy.

Based on the result of most of the wine (492) belong in “bad” quality, we provide following solution :

Quality Improvement: Wineries can analyze factors like grapes, fermentation, and storage to improve wine quality.

Market Segmentation: Retailers can discount or promote 'bad' wines differently to manage consumer expectations.

Customer Education: Educate consumers about wine faults to reduce dissatisfaction.

Recycling or Repurposing: Repurpose 'bad' wines for cooking or other products.

The top 3 discriminating factors of wine quality based on importance in the Random Forest model are **alcohol, sulphates, and volatile.acidity.** These features have the highest importance values, indicating they contribute most to the model's decision-making process.Vivino should focus on the top 3 discriminating factor of wine quality to improve their product and grow their business. Therefore, we provide following solution :

Quality Verification Service: Offer a premium service where Vivino verifies the alcohol content, sulphate levels, and overall volatile.acidity of wines before they are listed on the platform. This would ensure that users can trust the authenticity and quality of the wines.

Collaborations with Wineries: Partner with wineries that produce high-quality wines with optimal alcohol content and sulphate levels. Vivino could feature these wines on the platform and offer exclusive deals to users to attract more customers and driving sales.

Customized Wine Clubs: Create customized wine clubs where members receive monthly shipments of wines selected based on their preferences for alcohol content, sulphate levels, and overall volatile.acidity.