Wine Quality & Type Analysis and Modeling in R

Quyen Di Sabino

2025-05-16

A. Describes the dataset and variables:

Link to data https://archive.ics.uci.edu/ml/datasets/Wine+Quality

1. type – Type of wine (e.g., red or white).
2. fixed.acidity – Tartaric acid content.
3. volatile.acidity – Acetic acid content; too much can make the wine taste vinegary.
4. citric.acid – Can add freshness and flavor.
5. residual.sugar – Sugar remaining after fermentation.
6. chlorides – Salt content.
7. free.sulfur.dioxide – Free form of SO2; prevents microbial growth and oxidation.
8. total.sulfur.dioxide – Sum of free and bound forms; high values can affect taste and health.
9. density – Density of the wine; related to sugar and alcohol content.
10. pH – Acidity level.
11. sulphates – Wine preservative; contributes to SO2 levels.
12. alcohol – Alcohol percentage by volume.
13. quality – Wine quality score (often rated 0–9, based on sensory data).

```
# Read in data

red <- read.csv("winequality-red.csv", header = TRUE, sep = ";")
white <- read.csv("winequality-white.csv", header = TRUE, sep = ";")
wines <- rbind(
  data.frame(type = "red", red),
  data.frame(type = "white", white)
)

str(wines)
```

```
## 'data.frame':    6497 obs. of  13 variables:
##  $ type                : chr  "red" "red" "red" "red" ...
##  $ fixed.acidity       : num  7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5 ...
##  $ volatile.acidity    : num  0.7 0.88 0.76 0.28 0.7 0.66 0.6 0.65 0.58 0.5 ...
##  $ citric.acid         : num  0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...
##  $ residual.sugar      : num  1.9 2.6 2.3 1.9 1.9 1.8 1.6 1.2 2 6.1 ...
##  $ chlorides           : num  0.076 0.098 0.092 0.075 0.076 0.075 0.069 0.065 0.073 0.071 ...
##  $ free.sulfur.dioxide : num  11 25 15 17 11 13 15 15 9 17 ...
##  $ total.sulfur.dioxide: num  34 67 54 60 34 40 59 21 18 102 ...
##  $ density             : num  0.998 0.997 0.997 0.998 0.998 ...
##  $ pH                  : num  3.51 3.2 3.26 3.16 3.51 3.51 3.3 3.39 3.36 3.35 ...
##  $ sulphates           : num  0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.47 0.57 0.8 ...
```

```
##  $ alcohol             : num  9.4 9.8 9.8 9.8 9.4 9.4 9.4 10 9.5 10.5 ...
##  $ quality             : int  5 5 5 6 5 5 5 7 7 5 ...
```

```
head(wines)
```

```
##   type fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
## 1  red           7.4             0.70        0.00            1.9     0.076
## 2  red           7.8             0.88        0.00            2.6     0.098
## 3  red           7.8             0.76        0.04            2.3     0.092
## 4  red          11.2             0.28        0.56            1.9     0.075
## 5  red           7.4             0.70        0.00            1.9     0.076
## 6  red           7.4             0.66        0.00            1.8     0.075
##   free.sulfur.dioxide total.sulfur.dioxide density   pH sulphates alcohol
## 1                  11                   34  0.9978 3.51      0.56     9.4
## 2                  25                   67  0.9968 3.20      0.68     9.8
## 3                  15                   54  0.9970 3.26      0.65     9.8
## 4                  17                   60  0.9980 3.16      0.58     9.8
## 5                  11                   34  0.9978 3.51      0.56     9.4
## 6                  13                   40  0.9978 3.51      0.56     9.4
##   quality
## 1       5
## 2       5
## 3       5
## 4       6
## 5       5
## 6       5
```

```
table(wines$type)
```

```
##
##   red white
##  1599  4898
```

B. ANALYSIS TASKS

B.1. Install If Needed and Load Necessary Libraries

```
if(!require(corrplot))      install.packages("corrplot")
```

```
## Loading required package: corrplot
```

```
## corrplot 0.95 loaded
```

```
if(!require(ggcorrplot))    install.packages("ggcorrplot")
```

```
## Loading required package: ggcorrplot
```

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

```
if(!require(GGally))        install.packages("GGally")
```

```
## Loading required package: GGally
```

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

```
if(!require(xgboost))       install.packages("xgboost")
```

```
## Loading required package: xgboost
```

```
if(!require(Rtsne))         install.packages("Rtsne")
```

```
## Loading required package: Rtsne
```

```
if(!require(umap))          install.packages("umap")
```

```
## Loading required package: umap
```

```
if(!require(HandTill2001))  install.packages("HandTill2001")
```

```
## Loading required package: HandTill2001
```

```
if(!require(ranger))        install.packages("ranger")
```

```
## Loading required package: ranger
```

```
if(!require(rpart.plot))    install.packages("rpart.plot")
```

```
## Loading required package: rpart.plot
```

```
## Loading required package: rpart
```

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v lubridate 1.9.3     v tibble    3.2.1
## v purrr     1.0.2     v tidyr     1.3.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x dplyr::slice()  masks xgboost::slice()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(e1071)        # for naiveBayes()
library(purrr)        # for map_dbl()
library(ggplot2)      # for ggplot()
library(corrplot)     # for corrplot()
library(GGally)       # for ggpairs()
library(caret)        # for createDataPartition(), trainControl()
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(e1071)        # for svm()
library(randomForest) # for randomForest()
```

```
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
## The following object is masked from 'package:ranger':
##
##     importance
##
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(xgboost)      # for xgb
library(pROC)         # for roc() and auc()
```

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following object is masked from 'package:HandTill2001':
##
##     auc
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
library(Metrics)      # for rmse(), mae(), R2()
```

```
##
## Attaching package: 'Metrics'
##
## The following object is masked from 'package:pROC':
##
##     auc
##
## The following objects are masked from 'package:caret':
##
##     precision, recall
##
## The following object is masked from 'package:HandTill2001':
##
##     auc
```

```r
library(rpart)        # for rpart()
library(nnet)         # for multinom(), multinomial logistic regression
library(tidyr)        # for pivot_longer()
library(Rtsne)        # for t-SNE, Rtsne()
library(umap)         # for UMAP, umap()
library(cluster)      # for silhouette()
library(HandTill2001) # for multcap(), multiclass AUC
library(ranger)       # for ranger()
library(rpart.plot)   # for rpart.plot
library(MASS)         # for polr()
```

```
##
## Attaching package: 'MASS'
##
## The following object is masked from 'package:dplyr':
##
##     select
```

B.2. Exploratory Data Analysis (EDA)

- Summary Statistics: Used summary() to understand distributions, central tendencies, and missing data.
- Class-wise Analysis: Compared red vs. white wines using grouped summaries and visualizations (ggplot2).
- Outlier Detection: Used boxplots and z-scores to identify extreme values for features like residual.sugar, sulphates

```r
# Check for missing values
colSums(is.na(wines))
```

```
##                 type        fixed.acidity     volatile.acidity
##                    0                    0                    0
##          citric.acid       residual.sugar            chlorides
##                    0                    0                    0
##  free.sulfur.dioxide total.sulfur.dioxide              density
##                    0                    0                    0
```

```
##                   pH            sulphates              alcohol
##                    0                    0                    0
##              quality
##                    0
```

```
# Summary Statistics
summary(wines)
```

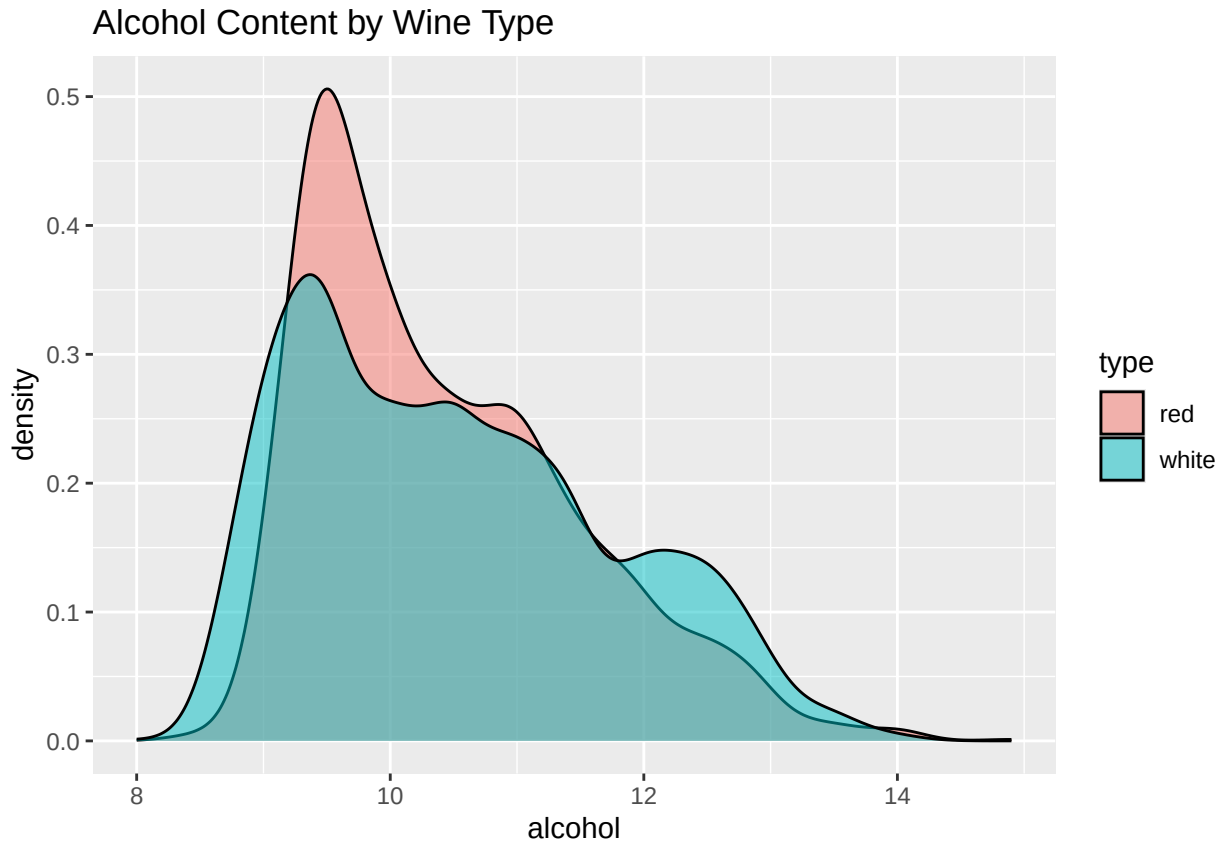
```
##      type           fixed.acidity    volatile.acidity  citric.acid
##  Length:6497        Min.   : 3.800   Min.   :0.0800   Min.   :0.0000
##  Class :character   1st Qu.: 6.400   1st Qu.:0.2300   1st Qu.:0.2500
##  Mode  :character   Median : 7.000   Median :0.2900   Median :0.3100
##                     Mean   : 7.215   Mean   :0.3397   Mean   :0.3186
##                     3rd Qu.: 7.700   3rd Qu.:0.4000   3rd Qu.:0.3900
##                     Max.   :15.900   Max.   :1.5800   Max.   :1.6600
##  residual.sugar     chlorides       free.sulfur.dioxide total.sulfur.dioxide
##  Min.   : 0.600   Min.   :0.00900   Min.   :  1.00      Min.   :  6.0
##  1st Qu.: 1.800   1st Qu.:0.03800   1st Qu.: 17.00      1st Qu.: 77.0
##  Median : 3.000   Median :0.04700   Median : 29.00      Median :118.0
##  Mean   : 5.443   Mean   :0.05603   Mean   : 30.53      Mean   :115.7
##  3rd Qu.: 8.100   3rd Qu.:0.06500   3rd Qu.: 41.00      3rd Qu.:156.0
##  Max.   :65.800   Max.   :0.61100   Max.   :289.00      Max.   :440.0
##     density             pH          sulphates         alcohol
##  Min.   :0.9871   Min.   :2.720   Min.   :0.2200   Min.   : 8.00
##  1st Qu.:0.9923   1st Qu.:3.110   1st Qu.:0.4300   1st Qu.: 9.50
##  Median :0.9949   Median :3.210   Median :0.5100   Median :10.30
##  Mean   :0.9947   Mean   :3.219   Mean   :0.5313   Mean   :10.49
##  3rd Qu.:0.9970   3rd Qu.:3.320   3rd Qu.:0.6000   3rd Qu.:11.30
##  Max.   :1.0390   Max.   :4.010   Max.   :2.0000   Max.   :14.90
##     quality
##  Min.   :3.000
##  1st Qu.:5.000
##  Median :6.000
##  Mean   :5.818
##  3rd Qu.:6.000
##  Max.   :9.000
```

```
# Grouped by wine type
wines %>%
  group_by(type) %>%
  summarize(across(where(is.numeric),
                   list(mean = mean, sd = sd),
                   .names = "{.col}_{.fn}")
            )
```

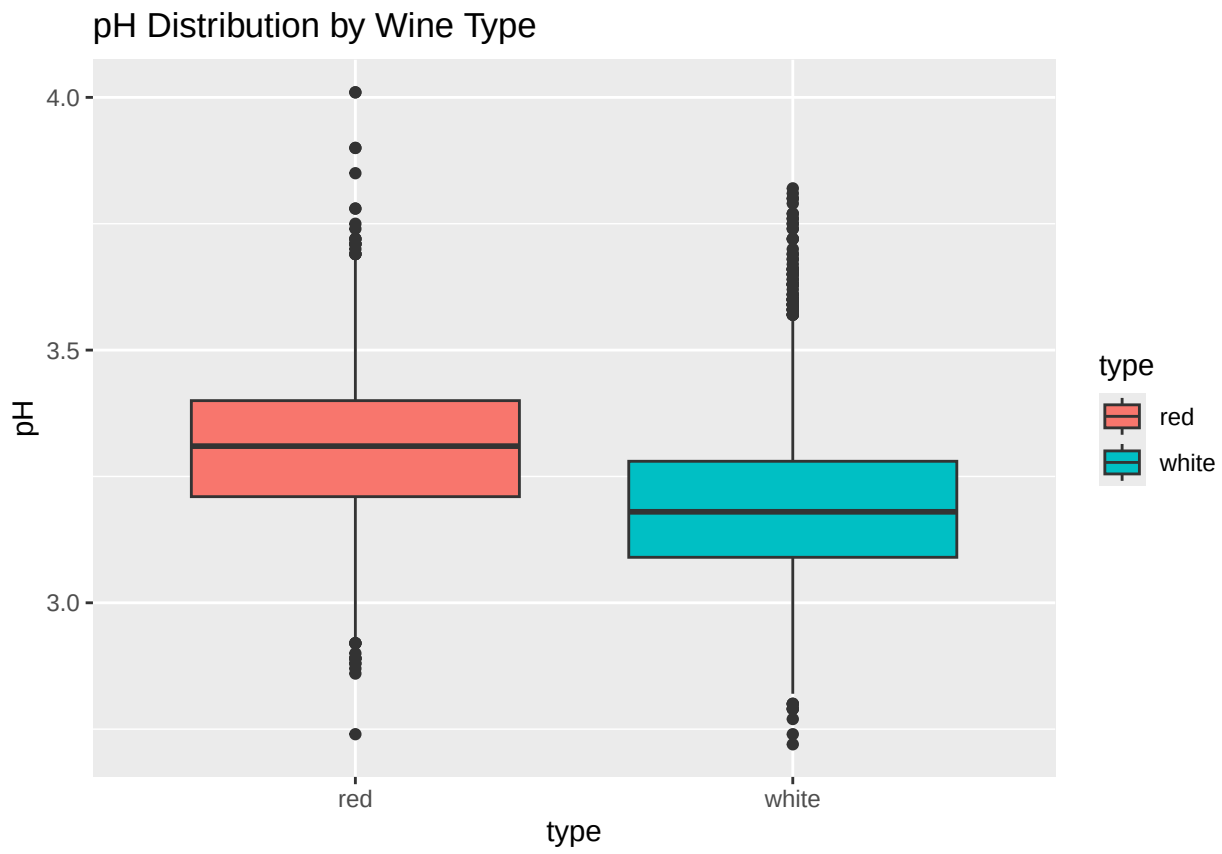
```
## # A tibble: 2 x 25
##   type  fixed.acidity_mean fixed.acidity_sd volatile.acidity_mean
##   <chr>              <dbl>            <dbl>                 <dbl>
## 1 red                 8.32            1.74                 0.528
## 2 white               6.85            0.844                0.278
## # i 21 more variables: volatile.acidity_sd <dbl>, citric.acid_mean <dbl>,
## #   citric.acid_sd <dbl>, residual.sugar_mean <dbl>, residual.sugar_sd <dbl>,
## #   chlorides_mean <dbl>, chlorides_sd <dbl>, free.sulfur.dioxide_mean <dbl>,
```

```
## #   free.sulfur.dioxide_sd <dbl>, total.sulfur.dioxide_mean <dbl>,
## #   total.sulfur.dioxide_sd <dbl>, density_mean <dbl>, density_sd <dbl>,
## #   pH_mean <dbl>, pH_sd <dbl>, sulphates_mean <dbl>, sulphates_sd <dbl>,
## #   alcohol_mean <dbl>, alcohol_sd <dbl>, quality_mean <dbl>, ...
```

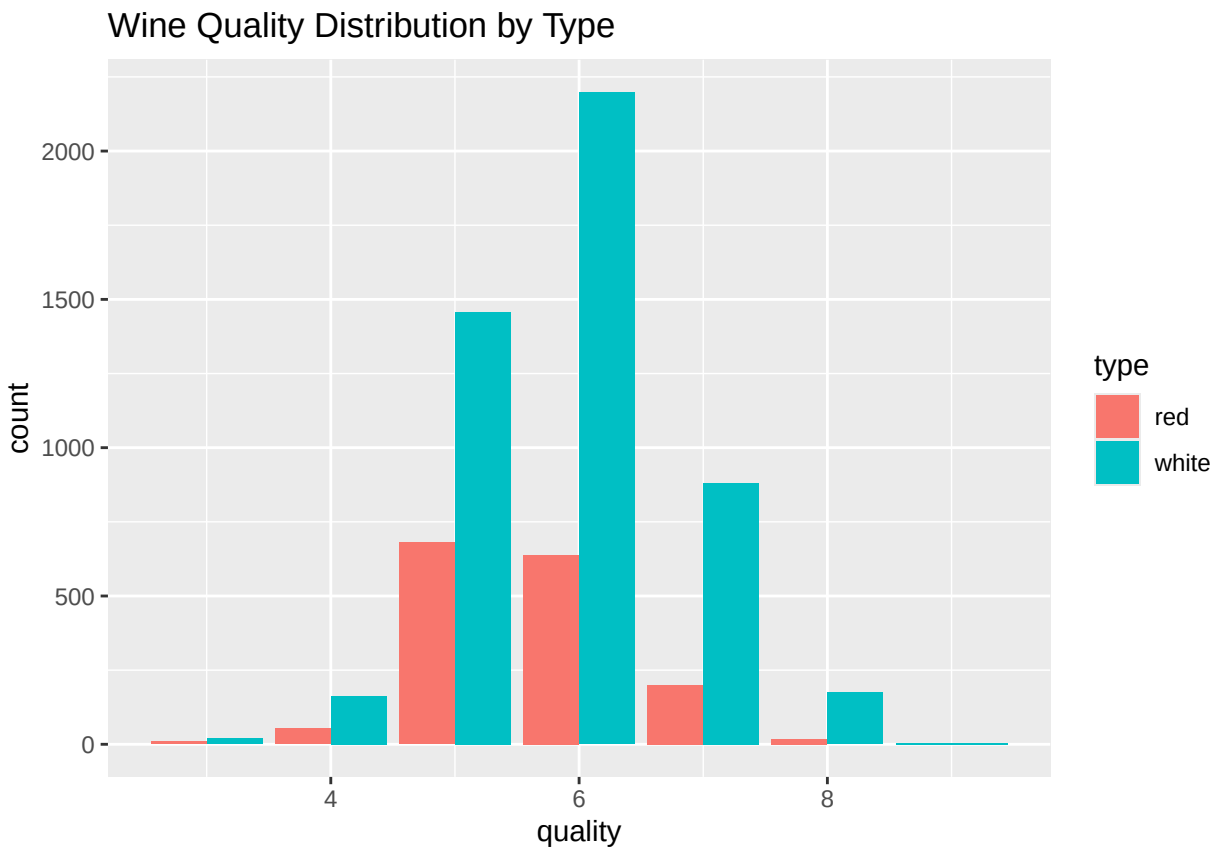
```r
# Numeric features only
wines_numeric <- select_if(wines, is.numeric)

# Alcohol distribution by wine type
ggplot(wines, aes(x = alcohol, fill = type)) +
  geom_density(alpha = 0.5) +
  labs(title = "Alcohol Content by Wine Type")
```



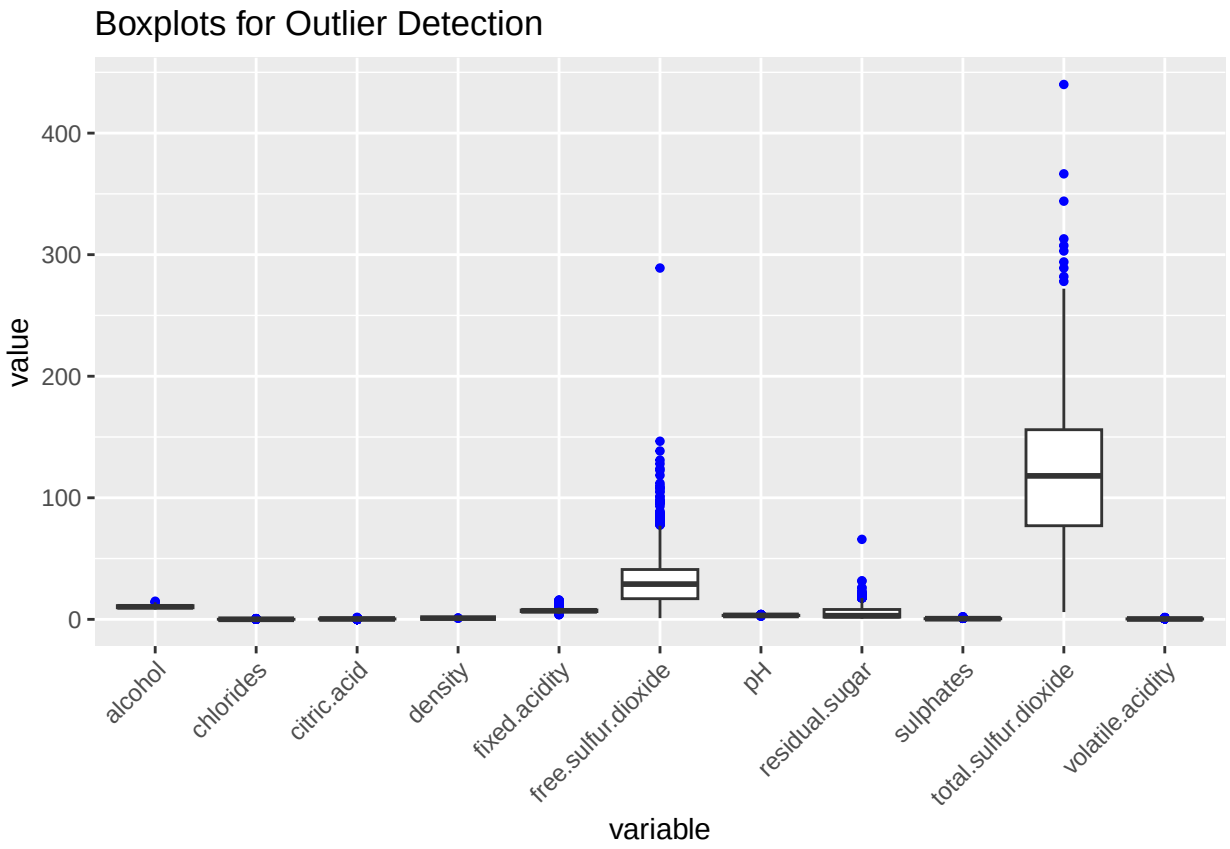

```r
# pH comparison by wine type
ggplot(wines, aes(x = type, y = pH, fill = type)) +
  geom_boxplot() +
  labs(title = "pH Distribution by Wine Type")
```




```r
# Quality distribution by wine type
ggplot(wines, aes(x = quality, fill = type)) +
  geom_bar(position = "dodge") +
  labs(title = "Wine Quality Distribution by Type")
```




```r
# Boxplots for selected features, plot multiple boxplots
wines_long <- wines %>%
  pivot_longer(cols = -c(type, quality), names_to = "variable", values_to = "value")

ggplot(wines_long, aes(x = variable, y = value)) +
  geom_boxplot(outlier.colour = "blue", outlier.size = 1) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = "Boxplots for Outlier Detection")
```



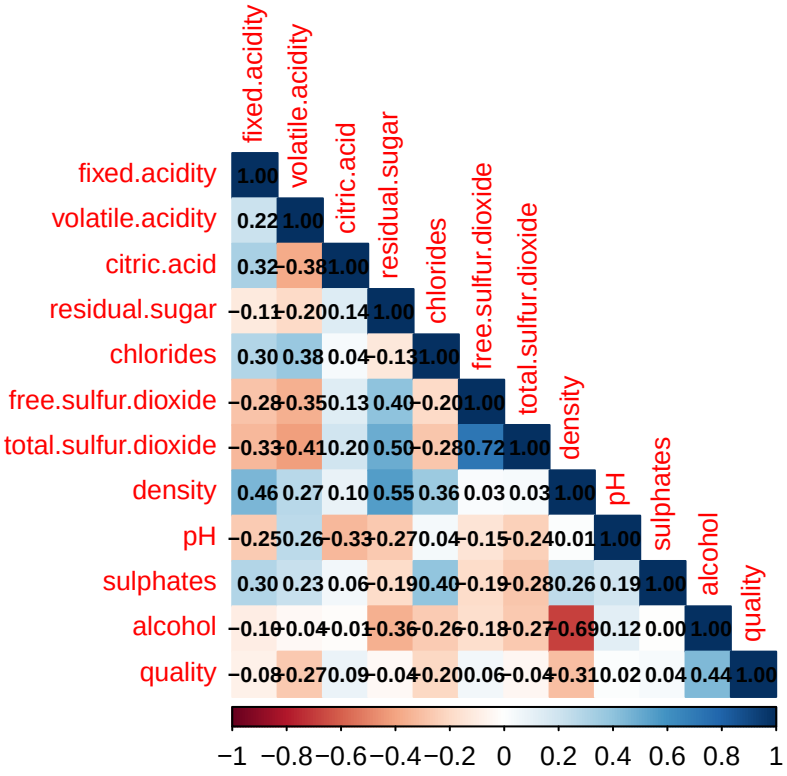

```r
# Z-score based outlier detection
free.sulfur.dioxide_outliers <- wines %>%
  # Convert 1-column matrix from scale() to a numeric vector
  mutate(free.sulfur.dioxide_z = as.numeric(scale(free.sulfur.dioxide))) %>%
  # Filter potential outliers
  filter(abs(free.sulfur.dioxide_z) > 3)

residual.sugar_outliers <- wines %>%
  mutate(residual.sugar_z = scale(residual.sugar)) %>%
  # Filter potential outliers
  filter(abs(residual.sugar_z) > 3)
```

B.3. Feature Relationships

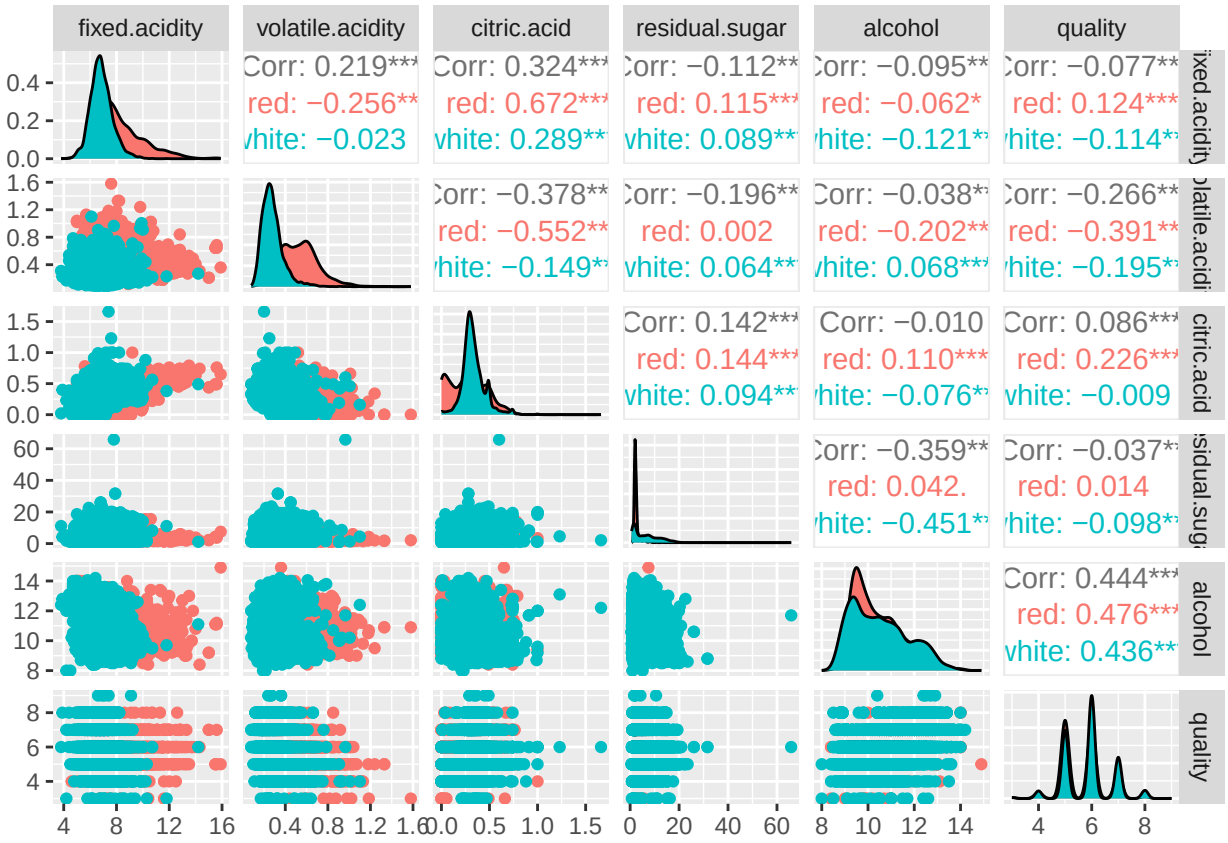
• Correlation Matrix: Visualized feature correlations using corrplot and ggcorrplot. • Pairwise Plots: Used GGally::ggpairs() to explore multivariate relationships, colored by wine type.

```r
# Compute correlation matrix
corr_matrix <- cor(wines_numeric)

# Correlation plot
corrplot(corr_matrix,
         method = "color",
         type = "lower",
         tl.cex = 0.8,          # Text size of the variable names
```

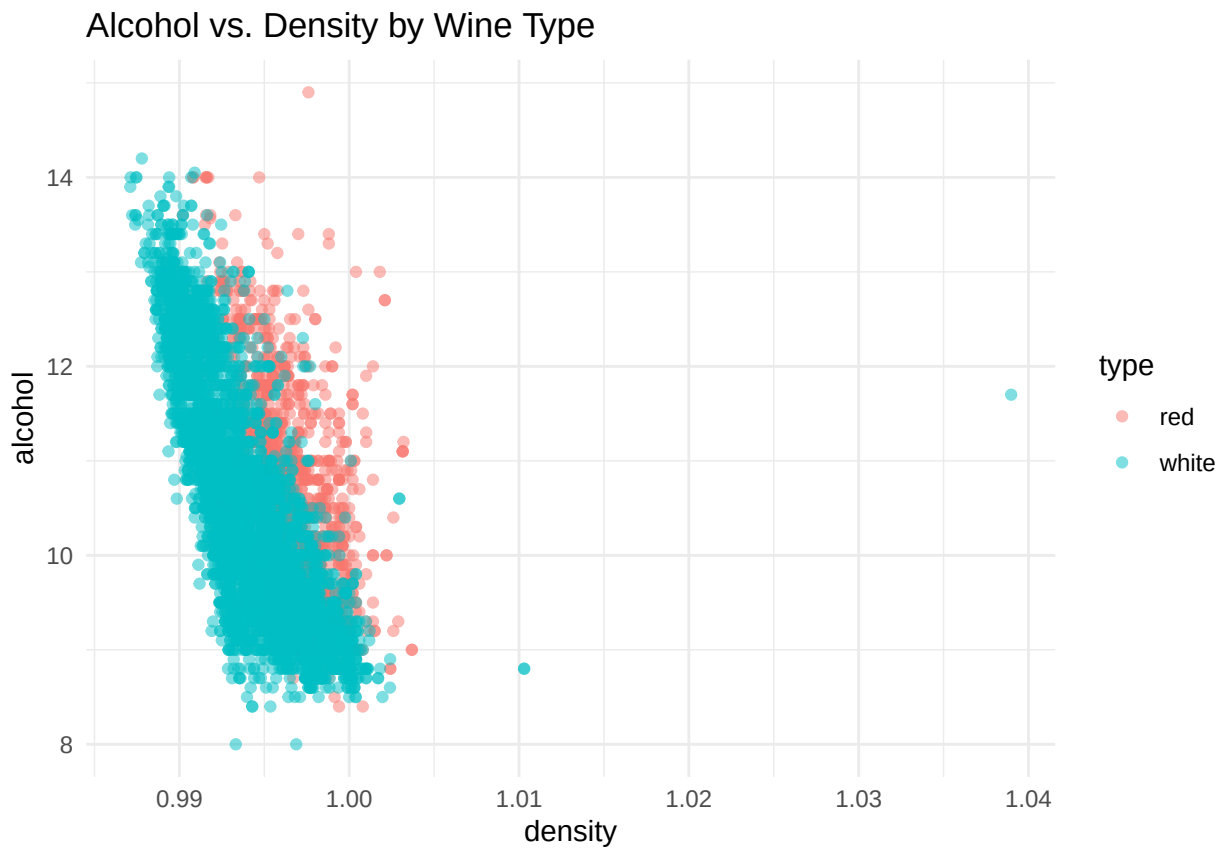
```
        addCoef.col = "black", # Adds correlation values in black
        number.cex = 0.7)      # Adjust the text size of corr values
```



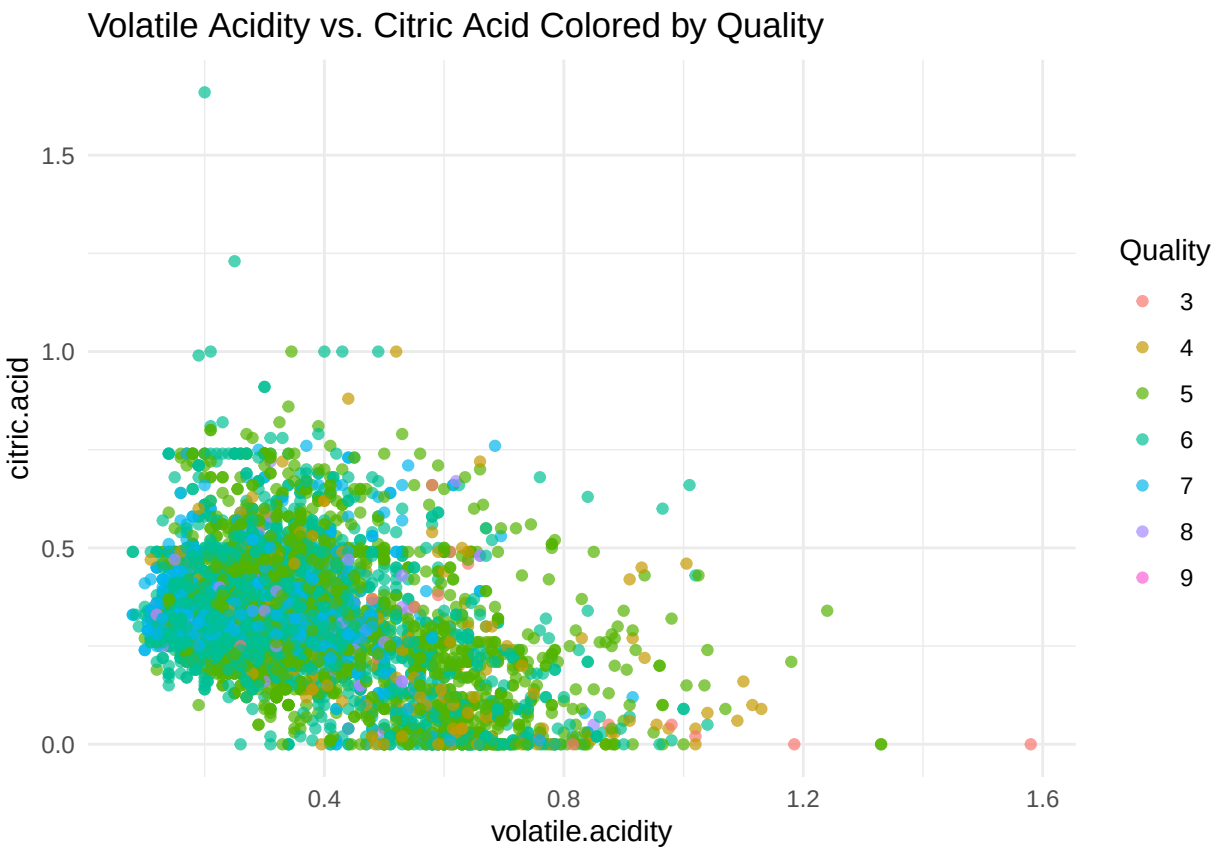

```
# Pairwise plot of a subset of key features (This is powerful but slow for many features)
ggpairs(wines,
        columns = c("fixed.acidity", "volatile.acidity", "citric.acid", "residual.sugar", "alcohol", "q
        mapping = aes(color = type)
        )
```




```r
# Alcohol vs. Density
ggplot(wines, aes(x = density, y = alcohol, color = type)) +
  geom_point(alpha = 0.5) +
  labs(title = "Alcohol vs. Density by Wine Type") +
  theme_minimal()
```




```
# Colored by Quality (instead of type)
ggplot(wines, aes(x = volatile.acidity, y = citric.acid, color = as.factor(quality))) +
  geom_point(alpha = 0.7) +
  labs(title = "Volatile Acidity vs. Citric Acid Colored by Quality",
       color = "Quality") +
  theme_minimal()
```



C. MODELING TASKS

C.I. BINARY CLASSIFICATION

```
#-------------------------------------------------------------------------

# Goal: Predict type (red or white) using physicochemical features.

#-------------------------------------------------------------------------


# Encode 'type' as a binary factor: red = 0, white = 1
wines$type <- factor(ifelse(wines$type == "white", 1, 0), levels = c(0, 1))

# Train/test split
set.seed(1)
index <- createDataPartition(wines$type, p = 0.8, list = FALSE)
train <- wines[index, ]
test <- wines[-index, ]

# Features and labels
train_x <- train %>% dplyr::select(-type)
train_y <- train$type
```

```r
test_x <- test %>% dplyr::select(-type)
test_y <- test$type
```

```r
#-----------------------------------------
# 1. Logistic Binary Classification
#-----------------------------------------

# Fit model
glm_model <- glm(type ~ ., data = train, family = "binomial")

# Predict
glm_prob <- predict(glm_model, test, type = "response")

# Class prediction from probabilities:
glm_class <- ifelse(glm_prob > 0.5, 1, 0)
glm_class <- factor(glm_class, levels = levels(test_y))

# Evaluation
confusionMatrix(glm_class, test_y) # 'Positive' Class : 0
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 315   7
##          1   4 972
##
##                Accuracy : 0.9915
##                  95% CI : (0.9849, 0.9958)
##     No Information Rate : 0.7542
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9772
##
##  Mcnemar's Test P-Value : 0.5465
##
##             Sensitivity : 0.9875
##             Specificity : 0.9928
##          Pos Pred Value : 0.9783
##          Neg Pred Value : 0.9959
##              Prevalence : 0.2458
##          Detection Rate : 0.2427
##    Detection Prevalence : 0.2481
##       Balanced Accuracy : 0.9902
##
##        'Positive' Class : 0
##
```

```r
glm_roc <- roc(test_y, glm_prob)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
glm_auc <- pROC::auc(glm_roc) # getAnywhere(auc) to check which version of auc() is active
glm_auc
```

```
## Area under the curve: 0.9954
```

```r
#---------------------------------------------
# 2. Random Forest Binary Classification
#---------------------------------------------

# Fit model
rf_model <- randomForest(type ~ ., data = train, ntree = 100)

# Predict
rf_prob <- predict(rf_model, test, type = "prob") # colnames(rf_prob) to get column names

# Class prediction from probabilities:
rf_class <- ifelse(rf_prob[, "1"] > 0.5, 1, 0)
rf_class <- factor(rf_class, levels = levels(test_y))

# Evaluation
confusionMatrix(rf_class, test_y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 315   2
##          1   4 977
##
##                Accuracy : 0.9954
##                  95% CI : (0.99, 0.9983)
##     No Information Rate : 0.7542
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9875
##
##  Mcnemar's Test P-Value : 0.6831
##
##             Sensitivity : 0.9875
##             Specificity : 0.9980
##          Pos Pred Value : 0.9937
##          Neg Pred Value : 0.9959
##              Prevalence : 0.2458
##          Detection Rate : 0.2427
##    Detection Prevalence : 0.2442
##       Balanced Accuracy : 0.9927
##
##        'Positive' Class : 0
##
```

```r
rf_roc <- roc(test_y, rf_prob[, "1"])
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

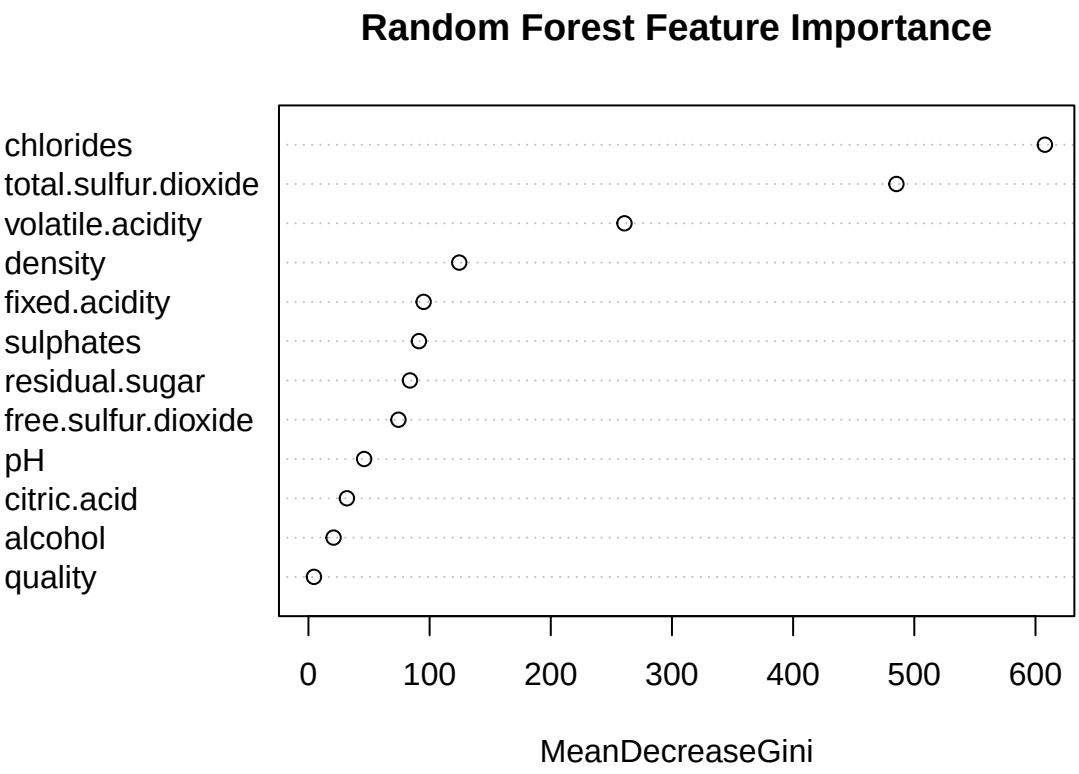
```
rf_auc <- pROC::auc(rf_roc)
rf_auc
```

```
## Area under the curve: 0.9976
```

```
# Feature Importance - Random Forest
# View variable importance
randomForest::importance(rf_model)
```

```
##                       MeanDecreaseGini
## fixed.acidity                95.072399
## volatile.acidity            260.810645
## citric.acid                  31.773228
## residual.sugar               83.791154
## chlorides                   607.759224
## free.sulfur.dioxide          74.281811
## total.sulfur.dioxide        485.216398
## density                     124.478290
## pH                           45.971759
## sulphates                    91.189034
## alcohol                      20.739152
## quality                       4.559322
```

```
# Plot importance
# Higher values indicate greater importance in classification splits.
varImpPlot(rf_model, main = "Random Forest Feature Importance")
```




```r
#-------------------------------------------------------------
# 3. Support Vector Machine (SVM) Binary Classification
#-------------------------------------------------------------

# Fit model
svm_model <- svm(type ~ ., data = train, probability = TRUE)

# Predict
svm_pred <- predict(svm_model, test, probability = TRUE)
svm_prob <- attr(svm_pred, "probabilities")

# Class prediction from probabilities:
svm_class <- ifelse(svm_prob[, "1"] > 0.5, 1, 0)
svm_class <- factor(svm_class, levels = levels(test_y))

# Evaluation
confusionMatrix(svm_class, test_y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 315   1
##          1   4 978
##
##                Accuracy : 0.9961
```

```
##                  95% CI : (0.991, 0.9987)
##     No Information Rate : 0.7542
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9896
##
##  Mcnemar's Test P-Value : 0.3711
##
##             Sensitivity : 0.9875
##             Specificity : 0.9990
##          Pos Pred Value : 0.9968
##          Neg Pred Value : 0.9959
##              Prevalence : 0.2458
##          Detection Rate : 0.2427
##    Detection Prevalence : 0.2435
##       Balanced Accuracy : 0.9932
##
##        'Positive' Class : 0
##
```

```
svm_roc <- roc(test_y, svm_prob[, "1"])
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
svm_auc <- pROC::auc(svm_roc)
svm_auc
```

```
## Area under the curve: 0.9978
```

```
#--------------------------------------
# 4. XGBoost Binary Classification
#--------------------------------------

# XGBoost needs numeric data and numeric labels.
# Prepare data
train_matrix <- xgb.DMatrix(data = as.matrix(train_x), label = as.numeric(as.character(train_y)))
test_matrix  <- xgb.DMatrix(data = as.matrix(test_x),  label = as.numeric(as.character(test_y)))

# Fit model
xgb_model <- xgboost(data = train_matrix,
                     objective = "binary:logistic",
                     nrounds = 100,
                     verbose = 0) # Suppresses training output - no messages while training

# Predict
xgb_prob <- predict(xgb_model, test_matrix)

# Class prediction from probabilities:
xgb_class <- ifelse(xgb_prob > 0.5, 1, 0)
xgb_class <- factor(xgb_class, levels = levels(test_y))
```

```r
# Evaluation
confusionMatrix(xgb_class, test_y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 315   2
##          1   4 977
##
##                Accuracy : 0.9954
##                  95% CI : (0.99, 0.9983)
##     No Information Rate : 0.7542
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9875
##
##  Mcnemar's Test P-Value : 0.6831
##
##             Sensitivity : 0.9875
##             Specificity : 0.9980
##          Pos Pred Value : 0.9937
##          Neg Pred Value : 0.9959
##              Prevalence : 0.2458
##          Detection Rate : 0.2427
##    Detection Prevalence : 0.2442
##       Balanced Accuracy : 0.9927
##
##        'Positive' Class : 0
##
```

```r
xgb_roc <- roc(test_y, xgb_prob)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

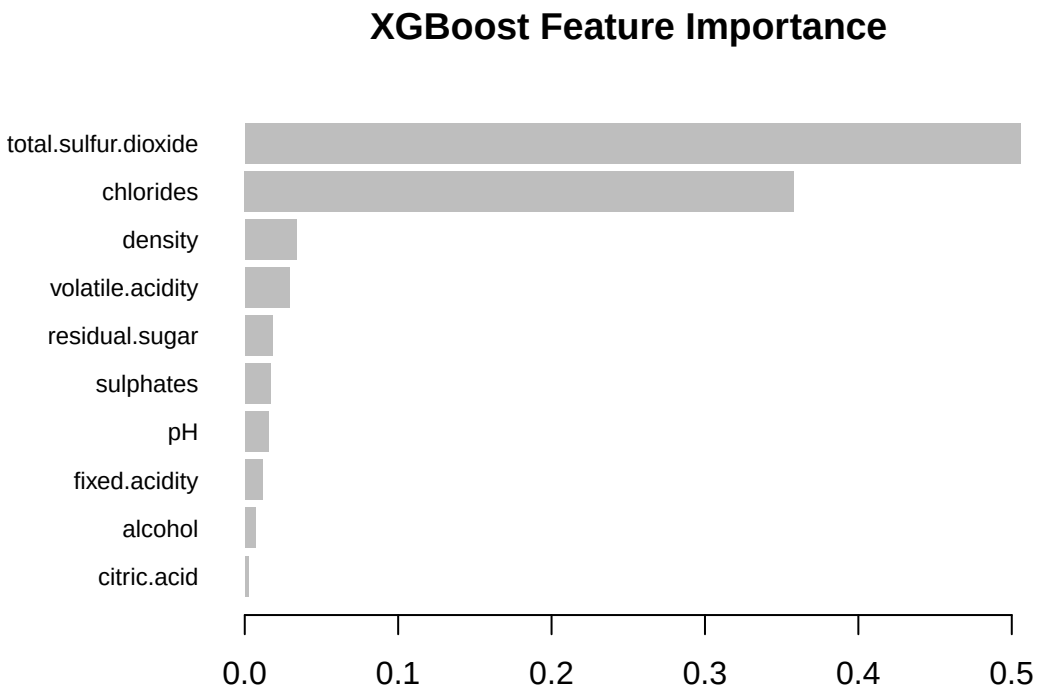
```r
xgb_auc <- pROC::auc(xgb_roc)
xgb_auc
```

```
## Area under the curve: 0.9977
```

```r
# Feature Importance - XGBoost
# Get importance matrix
importance_matrix <- xgb.importance(model = xgb_model)

# Access the raw table
# "weight/frequency" = number of times a feature is used in a tree
# "gain" = average gain when it is used
# "cover" = number of samples affected
print(importance_matrix)
```

```
##                  Feature         Gain       Cover   Frequency
##                   <char>        <num>       <num>       <num>
##  1: total.sulfur.dioxide 0.5055957538 0.296866783 0.151741294
##  2:            chlorides 0.3579968932 0.268109988 0.149253731
##  3:              density 0.0338222350 0.053366565 0.129353234
##  4:     volatile.acidity 0.0292680197 0.117943024 0.101990050
##  5:       residual.sugar 0.0182305684 0.029504715 0.064676617
##  6:            sulphates 0.0168282675 0.093989548 0.083333333
##  7:                   pH 0.0153623697 0.033760968 0.073383085
##  8:        fixed.acidity 0.0114690038 0.008112131 0.044776119
##  9:              alcohol 0.0069604820 0.050839400 0.090796020
## 10:          citric.acid 0.0025265860 0.040001703 0.037313433
## 11:  free.sulfur.dioxide 0.0016851724 0.005683983 0.065920398
## 12:              quality 0.0002546486 0.001821192 0.007462687
```

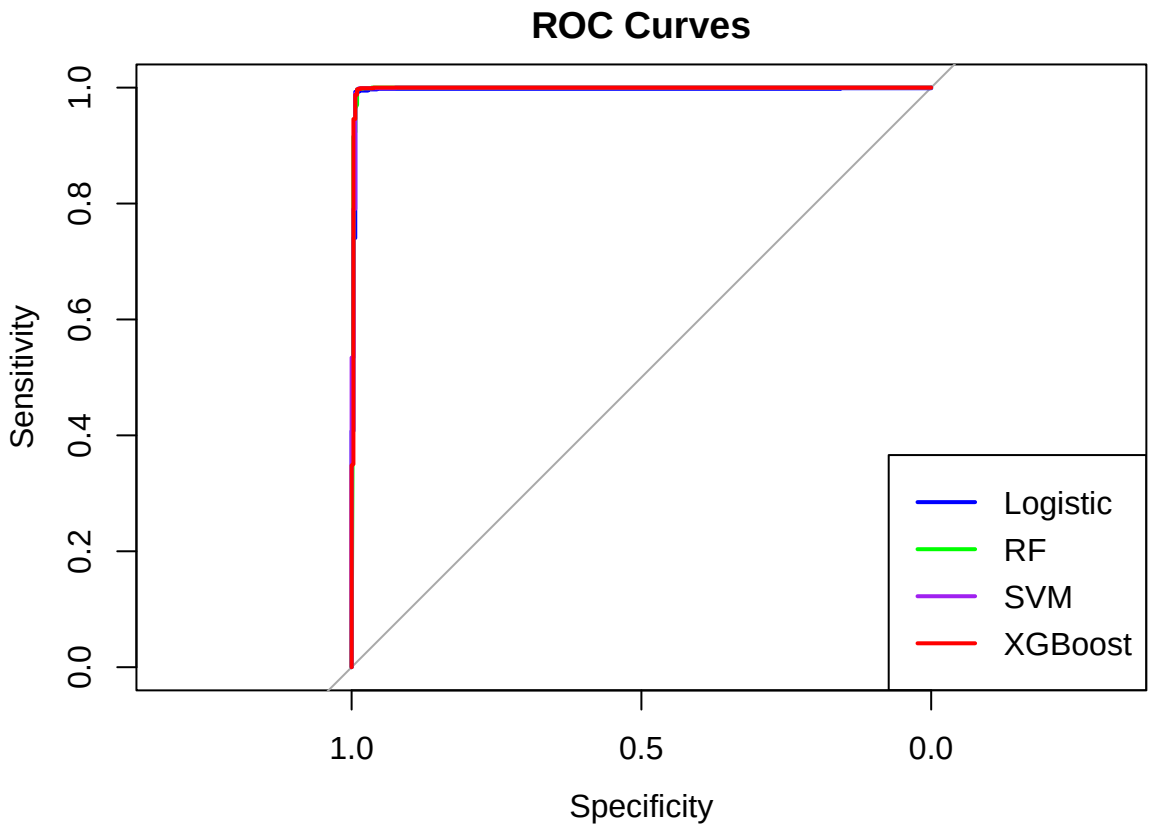
```r
# Plot importance
xgb.plot.importance(importance_matrix, top_n = 10,
                    main = "XGBoost Feature Importance")
```




```r
# Summary
auc_comparison <- tibble(
  Models = c("Logistic", "Random Forest", "Support Vector Machine (SVM)", "XGBoost"),
  AUC = c(glm_auc, rf_auc, svm_auc, xgb_auc)
)
auc_comparison
```

```
## # A tibble: 4 x 2
##   Models                          AUC
##   <chr>                         <dbl>
## 1 Logistic                      0.995
## 2 Random Forest                 0.998
## 3 Support Vector Machine (SVM) 0.998
## 4 XGBoost                       0.998
```

```r
# AUCs Comparison Plot
plot(glm_roc, col = "blue", main = "ROC Curves", lwd = 2)
lines(rf_roc, col = "green", lwd = 2)
lines(svm_roc, col = "purple", lwd = 2)
lines(xgb_roc, col = "red", lwd = 2)
legend("bottomright", legend = c("Logistic", "RF", "SVM", "XGBoost"),
       col = c("blue", "green", "purple", "red"), lwd = 2)
```



C.II. REGRESSION: Predict Wine Quality

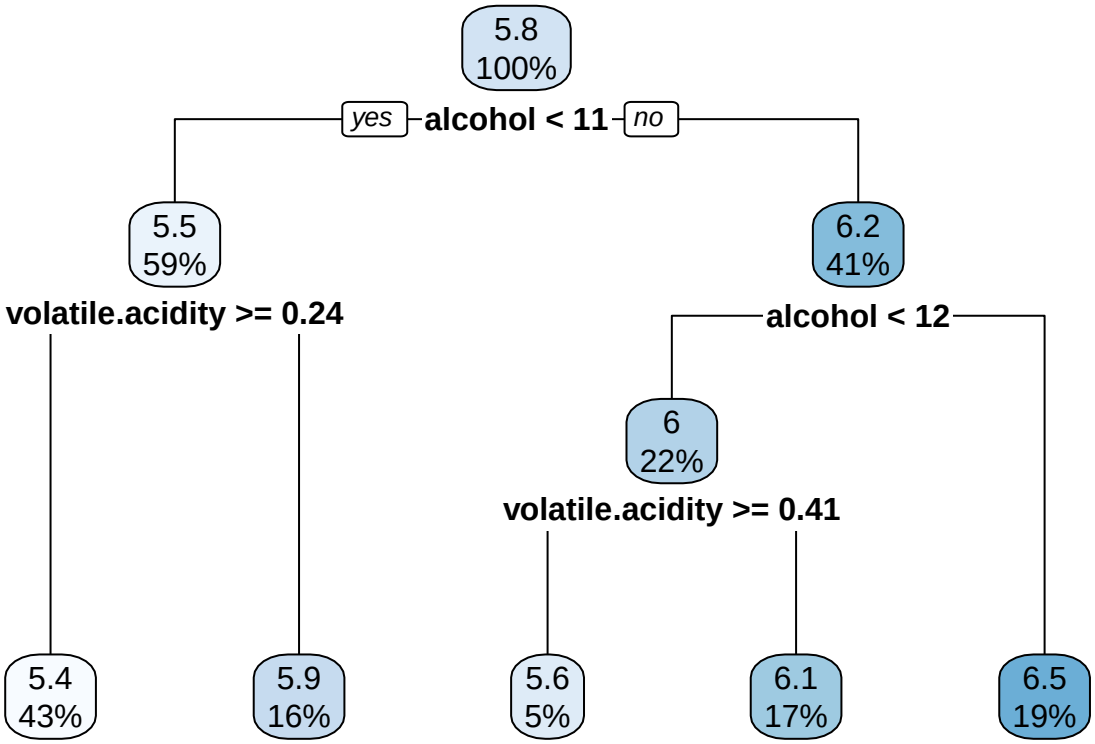
```r
#-------------------------------------------------------

# Goal: Predict quality as a numeric score (0 to 9)

#-------------------------------------------------------
```

```r
# Drop 'type', we don't use 'type' for prediction
wines_reg <- wines %>% dplyr::select(-type)

# Train/test split
set.seed(12)
index <- createDataPartition(wines_reg$quality, p = 0.8, list = FALSE)
train <- wines_reg[index, ]
test <- wines_reg[-index, ]

# Separate features and target
train_x <- train %>% dplyr::select(-quality)
train_y <- train$quality

test_x <- test %>% dplyr::select(-quality)
test_y <- test$quality
```

```r
#------------------------------
# 1. Linear Regression
#------------------------------

# Fit model
lm_model <- lm(quality ~ ., data = train)

# Predict
lm_pred <- predict(lm_model, test)

# Evaluation
lm_rmse <- rmse(test_y, lm_pred)
lm_mae  <- mae(test_y, lm_pred)
lm_r2   <- R2(lm_pred, test_y)

cat("Linear Regression:\n",
    "RMSE:", lm_rmse,
    "\nMAE:", lm_mae,
    "\nR²:", lm_r2, "\n")
```

```
## Linear Regression:
##  RMSE: 0.7344934
## MAE: 0.5739312
## R²: 0.2940003
```

```r
#---------------------------------
# 2. Decision Tree Regression
#---------------------------------

# Fit model
tree_model <- rpart(quality ~ ., data = train, method = "anova")

# Plot the tree
# Each leaf node shows the predicted quality for wines in that segment.
rpart.plot(tree_model)
```




```
# Predict
tree_pred <- predict(tree_model, test)

# Evaluation
tree_rmse <- rmse(test_y, tree_pred)
tree_mae  <- mae(test_y, tree_pred)
tree_r2   <- R2(tree_pred, test_y)

cat("Decision Tree:\n",
    "RMSE:", tree_rmse,
    "\nMAE:", tree_mae,
    "\nR²:", tree_r2, "\n")
```

```
## Decision Tree:
##  RMSE: 0.7598171
## MAE: 0.6138894
## R²: 0.2444273
```

```
#----------------------------------
# 3. Random Forest Regression
#----------------------------------

# Fit model
rf_model_reg <- randomForest(quality ~ ., data = train, ntree = 100)
```

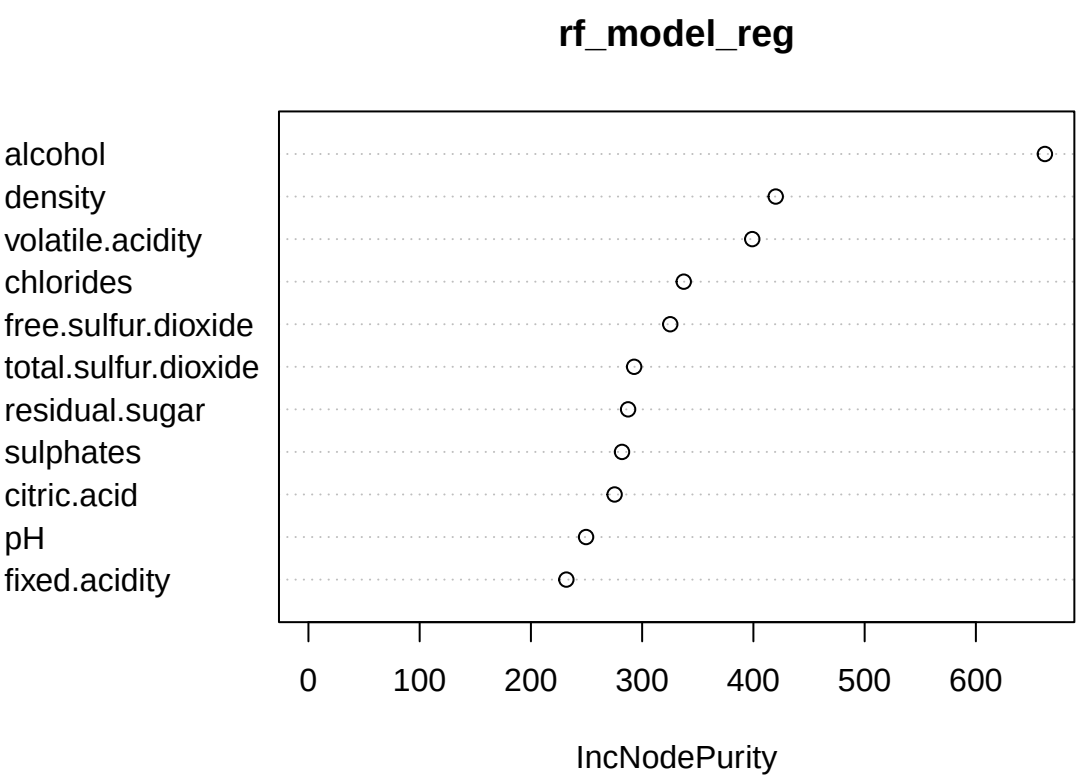
```r
# Predict
rf_pred <- predict(rf_model_reg, test)

# Evaluation
rf_rmse <- rmse(test_y, rf_pred)
rf_mae  <- mae(test_y, rf_pred)
rf_r2   <- R2(rf_pred, test_y)

cat("Random Forest:\n",
    "RMSE:", rf_rmse,
    "\nMAE:", rf_mae,
    "\nR²:", rf_r2, "\n")
```

```
## Random Forest:
##  RMSE: 0.6005224
## MAE: 0.446674
## R²: 0.5365326
```

```r
# Feature importance
randomForest::importance(rf_model_reg)
```

```
##                       IncNodePurity
## fixed.acidity             231.8711
## volatile.acidity          398.9723
## citric.acid               275.2370
## residual.sugar            287.3568
## chlorides                 337.3882
## free.sulfur.dioxide       325.2496
## total.sulfur.dioxide      292.7905
## density                   420.0723
## pH                        249.5884
## sulphates                 281.8847
## alcohol                   662.0371
```

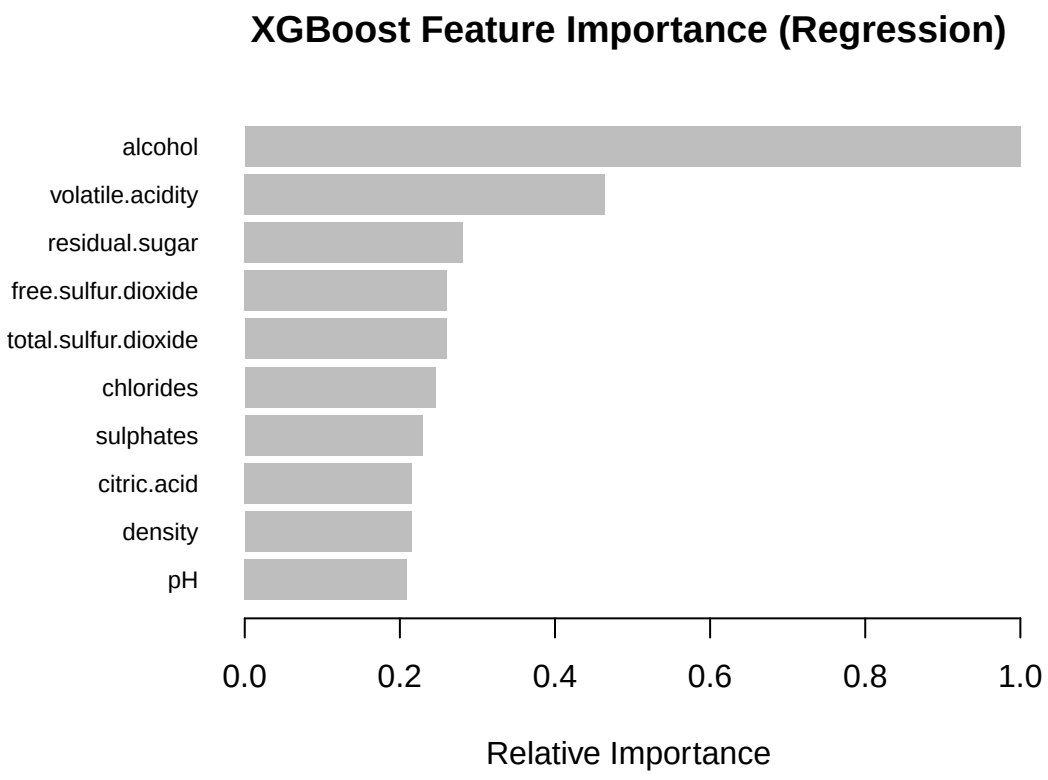
```r
varImpPlot(rf_model_reg)
```




```r
#------------------------------
# 4. XGBoost Regression
#------------------------------

# Prepare matrices
train_matrix <- xgb.DMatrix(data = as.matrix(train_x), label = as.numeric(as.character(train_y)))
test_matrix  <- xgb.DMatrix(data = as.matrix(test_x),  label = as.numeric(as.character(test_y)))

# Fit model
xgb_model_reg <- xgboost(data = train_matrix,
                         objective = "reg:squarederror",
                         nrounds = 100,
                         verbose = 0)

# Predict
xgb_pred <- predict(xgb_model_reg, test_matrix)

# Evaluation
xgb_rmse <- rmse(test_y, xgb_pred)
xgb_mae  <- mae(test_y, xgb_pred)
xgb_r2   <- R2(xgb_pred, test_y)

cat("XGBoost:\n",
    "RMSE:", xgb_rmse,
    "\nMAE:", xgb_mae,
    "\nR²:", xgb_r2, "\n")
```

```
## XGBoost:
##  RMSE: 0.6398854
## MAE: 0.4699056
## R²: 0.4691317
```

```r
# Get Feature Importance from XGBoost
# Extract importance scores
importance <- xgb.importance(model = xgb_model_reg)

# Plot Feature Importance
# Plot top 10 most important features by gain
xgb.plot.importance(importance, top_n = 10,
                    # Scale all feature importance scores relative to the most important feature
                    rel_to_first = TRUE,
                    main = "XGBoost Feature Importance (Regression)",
                    xlab = "Relative Importance")
```




```r
# View full importance table
print(importance)
```

```
##                 Feature       Gain      Cover  Frequency Importance
##                  <char>      <num>      <num>      <num>      <num>
##  1:              alcohol 0.27872340 0.07332851 0.06070568 0.27872340
##  2:     volatile.acidity 0.12948395 0.08798530 0.10913872 0.12948395
##  3:       residual.sugar 0.07839476 0.10295134 0.09160640 0.07839476
```

```
##  4:  free.sulfur.dioxide 0.07262795 0.07683065 0.08152531 0.07262795
##  5: total.sulfur.dioxide 0.07252057 0.13087394 0.09511286 0.07252057
##  6:            chlorides 0.06860041 0.08902700 0.09752356 0.06860041
##  7:            sulphates 0.06380114 0.08116791 0.07670392 0.06380114
##  8:          citric.acid 0.06012006 0.07137003 0.08393601 0.06012006
##  9:              density 0.05986649 0.12425637 0.08875740 0.05986649
## 10:                   pH 0.05823722 0.07852540 0.07801885 0.05823722
## 11:        fixed.acidity 0.05762404 0.08368355 0.13697129 0.05762404
```

```
# View top 10 importance features, sorted by Gain.
# High Gain = large impact per use → most valuable features
importance %>%
  arrange(desc(Gain)) %>%
  head(10)
```

```
##                  Feature       Gain      Cover  Frequency Importance
##                   <char>      <num>      <num>      <num>      <num>
##  1:              alcohol 0.27872340 0.07332851 0.06070568 0.27872340
##  2:     volatile.acidity 0.12948395 0.08798530 0.10913872 0.12948395
##  3:       residual.sugar 0.07839476 0.10295134 0.09160640 0.07839476
##  4:  free.sulfur.dioxide 0.07262795 0.07683065 0.08152531 0.07262795
##  5: total.sulfur.dioxide 0.07252057 0.13087394 0.09511286 0.07252057
##  6:            chlorides 0.06860041 0.08902700 0.09752356 0.06860041
##  7:            sulphates 0.06380114 0.08116791 0.07670392 0.06380114
##  8:          citric.acid 0.06012006 0.07137003 0.08393601 0.06012006
##  9:              density 0.05986649 0.12425637 0.08875740 0.05986649
## 10:                   pH 0.05823722 0.07852540 0.07801885 0.05823722
```

```
# Summary
metrics_comparison <- tibble(
  Model = c("Linear Regression", "Decision Tree", "Random Forest", "XGBoost"),
  RMSE  = c(lm_rmse, tree_rmse, rf_rmse, xgb_rmse),
  MAE   = c(lm_mae, tree_mae, rf_mae, xgb_mae),
  R2    = c(lm_r2, tree_r2, rf_r2, xgb_r2)
)

metrics_comparison
```

```
## # A tibble: 4 x 4
##   Model              RMSE   MAE    R2
##   <chr>             <dbl> <dbl> <dbl>
## 1 Linear Regression 0.734 0.574 0.294
## 2 Decision Tree     0.760 0.614 0.244
## 3 Random Forest     0.601 0.447 0.537
## 4 XGBoost           0.640 0.470 0.469
```

C.III. MULTICLASS CLASSIFICATION: Predict Discrete Quality

```
#------------------------------------------------------------------------

# Goal: Treat quality as a classification problem instead of regression
```

```r
#-------------------------------------------------------------------------------

# Convert quality to factor (multiclass)
wines_class <- wines %>% dplyr::select(-type)
wines_class$quality <- factor(wines_class$quality)

# Train/test split
set.seed(123)
index <- createDataPartition(wines_class$quality, p = 0.8, list = FALSE)
train <- wines_class[index, ]
test <- wines_class[-index, ]

train_x <- train %>% dplyr::select(-quality)
train_y <- train$quality

test_x <- test %>% dplyr::select(-quality)
test_y <- test$quality
```

```r
#-------------------------------------------------------------
# 1. Multinomial Logistic - Multiclass Classification
#-------------------------------------------------------------

# Fit model
multinom_model <- multinom(quality ~ ., data = train)
```

```
## # weights:  91 (72 variable)
## initial  value 10118.732775
## iter  10 value 6864.099466
## iter  20 value 6368.891550
## iter  30 value 6097.524912
## iter  40 value 5657.173575
## iter  50 value 5562.057991
## iter  60 value 5549.668030
## iter  70 value 5544.560943
## iter  80 value 5542.267614
## iter  90 value 5539.365203
## iter 100 value 5534.326544
## final  value 5534.326544
## stopped after 100 iterations
```

```r
# Predict
multinom_prob <- predict(multinom_model, test, type = "prob")
head(multinom_prob)
```

```
##              3         4         5         6           7           8
## 3  0.012104817 0.09279456 0.6967941 0.1898366 0.008185217 0.0002847359
## 17 0.001294040 0.00654225 0.2260417 0.5986899 0.158328110 0.0091037717
## 22 0.001678543 0.02031125 0.4401497 0.4769301 0.058048987 0.0028813632
## 29 0.007646438 0.08732910 0.7139560 0.1834658 0.007302636 0.0002999881
## 31 0.011194619 0.10573916 0.6337049 0.2374162 0.011392388 0.0005527001
## 32 0.015126784 0.04401096 0.4754585 0.4326002 0.031375124 0.0014283484
##               9
```

```
## 3  1.348711e-10
## 17 2.610664e-07
## 22 4.564095e-08
## 29 8.500248e-10
## 31 5.443716e-10
## 32 5.203113e-09
```

```r
multinom_class <- predict(multinom_model, test)

# Rename columns to match class labels
colnames(multinom_prob) <- levels(test_y)
# Confirm the internal match
all.equal(sort(colnames(multinom_prob)), sort(levels(test_y)))
```

```
## [1] TRUE
```

```r
# Evaluation
confusionMatrix(multinom_class, test_y)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8   9
##          3   1   0   0   0   0   0   0
##          4   0   0   0   0   0   0   0
##          5   3  23 236 134  10   5   0
##          6   2  19 186 400 153  24   1
##          7   0   1   4  33  52   9   0
##          8   0   0   0   0   0   0   0
##          9   0   0   1   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.5312
##                  95% CI : (0.5036, 0.5587)
##     No Information Rate : 0.4372
##     P-Value [Acc > NIR] : 6.512e-12
##
##                   Kappa : 0.242
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.166667  0.00000   0.5527   0.7055  0.24186   0.0000
## Specificity          1.000000  1.00000   0.7989   0.4726  0.95656   1.0000
## Pos Pred Value       1.000000      NaN   0.5742   0.5096  0.52525      NaN
## Neg Pred Value       0.996142  0.96685   0.7844   0.6738  0.86394   0.9707
## Prevalence           0.004626  0.03315   0.3292   0.4372  0.16577   0.0293
## Detection Rate       0.000771  0.00000   0.1820   0.3084  0.04009   0.0000
## Detection Prevalence 0.000771  0.00000   0.3169   0.6052  0.07633   0.0000
## Balanced Accuracy    0.583333  0.50000   0.6758   0.5890  0.59921   0.5000
```

```
##                      Class: 9
## Sensitivity          0.000000
## Specificity          0.999228
## Pos Pred Value       0.000000
## Neg Pred Value       0.999228
## Prevalence           0.000771
## Detection Rate       0.000000
## Detection Prevalence 0.000771
## Balanced Accuracy    0.499614
```

```
multinom_mc <- multcap(response = test_y, predicted = multinom_prob)
multinom_auc <- HandTill2001::auc(multinom_mc)
multinom_auc
```

```
## [1] 0.7036493
```

```
#------------------------------------------------------
# 2. Ordinal Logistic - Multiclass Classification
# -----------------------------------------------------

# Ensure quality is an ordered factor
train$quality <- ordered(train$quality)
test$quality  <- ordered(test$quality)

# Fit model
polr_model <- polr(quality ~ ., data = train, method = "logistic")

# Predict
polr_prob <- predict(polr_model, newdata = test, type = "probs")
head(polr_prob)
```

```
##              3         4         5         6          7           8
## 3  0.019015855 0.12775480 0.6589348 0.1768398 0.01573152 0.001683312
## 17 0.001613930 0.01252837 0.2428153 0.5674288 0.15533453 0.019801291
## 22 0.004771257 0.03603581 0.4655084 0.4266686 0.06008480 0.006769931
## 29 0.020483126 0.13603956 0.6607784 0.1664981 0.01460322 0.001560594
## 31 0.014627308 0.10176942 0.6441167 0.2168146 0.02042300 0.002196951
## 32 0.006270566 0.04675629 0.5214225 0.3738021 0.04647388 0.005152240
##               9
## 3  3.985948e-05
## 17 4.777594e-04
## 22 1.611473e-04
## 29 3.694898e-05
## 31 5.204947e-05
## 32 1.224365e-04
```

```
polr_class <- predict(polr_model, newdata = test)

# Rename columns to match class labels
colnames(polr_prob) <- levels(test_y)
# Confirm the internal match
all.equal(sort(colnames(polr_prob)), sort(levels(test_y)))
```

```
## [1] TRUE
```

```
# Evaluation
confusionMatrix(polr_class, test$quality)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8   9
##          3   0   0   0   0   0   0   0
##          4   1   0   1   0   0   0   0
##          5   2  23 225 128  10   5   0
##          6   3  19 196 398 150  20   1
##          7   0   1   4  41  55  13   0
##          8   0   0   1   0   0   0   0
##          9   0   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.5227
##                  95% CI : (0.4952, 0.5502)
##     No Information Rate : 0.4372
##     P-Value [Acc > NIR] : 3.708e-10
##
##                   Kappa : 0.2307
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.000000 0.000000   0.5269   0.7019  0.25581 0.000000
## Specificity          1.000000 0.998405   0.8069   0.4671  0.94547 0.999206
## Pos Pred Value            NaN 0.000000   0.5725   0.5057  0.48246 0.000000
## Neg Pred Value       0.995374 0.966795   0.7765   0.6686  0.86475 0.970679
## Prevalence           0.004626 0.033153   0.3292   0.4372  0.16577 0.029298
## Detection Rate       0.000000 0.000000   0.1735   0.3069  0.04241 0.000000
## Detection Prevalence 0.000000 0.001542   0.3030   0.6068  0.08790 0.000771
## Balanced Accuracy    0.500000 0.499203   0.6669   0.5845  0.60064 0.499603
##                      Class: 9
## Sensitivity          0.000000
## Specificity          1.000000
## Pos Pred Value            NaN
## Neg Pred Value       0.999229
## Prevalence           0.000771
## Detection Rate       0.000000
## Detection Prevalence 0.000000
## Balanced Accuracy    0.500000
```

```
ordinal_mc <- multcap(response = test$quality, predicted = polr_prob)
ordinal_auc <-  HandTill2001::auc(ordinal_mc)
ordinal_auc
```

```
## [1] 0.6862919
```

```r
#---------------------------------------------------------
# 3. Random Forest - Multiclass Classification
#---------------------------------------------------------

# Fit model
rf_model_multi <- randomForest(quality ~ ., data = train, ntree = 100)

# Predict
rf_prob_multi <- predict(rf_model_multi, newdata = test, type = "prob")
head(rf_prob_multi)
```

```
##       3    4    5    6    7 8 9
## 3  0.02 0.02 0.56 0.39 0.01 0 0
## 17 0.00 0.03 0.38 0.44 0.15 0 0
## 22 0.00 0.06 0.41 0.53 0.00 0 0
## 29 0.01 0.00 0.82 0.17 0.00 0 0
## 31 0.00 0.01 0.70 0.28 0.01 0 0
## 32 0.00 0.05 0.41 0.50 0.04 0 0
```

```r
rf_class_multi <- predict(rf_model_multi, test)

# Rename columns to match class labels
colnames(rf_prob_multi) <- levels(test_y)
# Confirm the internal match
all.equal(sort(colnames(rf_prob_multi)), sort(levels(test_y)))
```

```
## [1] TRUE
```

```r
# Evaluation
cm <- confusionMatrix(rf_class_multi, test_y)
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8   9
##          3   0   1   0   0   0   0   0
##          4   0   4   0   0   0   0   0
##          5   3  18 291  79   3   0   1
##          6   3  17 134 450 105  17   0
##          7   0   3   2  37 105   8   0
##          8   0   0   0   1   2  13   0
##          9   0   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.6654
##                  95% CI : (0.639, 0.691)
##     No Information Rate : 0.4372
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4728
```

```
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.000000 0.093023   0.6815   0.7937  0.48837  0.34211
## Specificity          0.999225 1.000000   0.8805   0.6219  0.95379  0.99762
## Pos Pred Value       0.000000 1.000000   0.7367   0.6198  0.67742  0.81250
## Neg Pred Value       0.995370 0.969838   0.8492   0.7951  0.90368  0.98048
## Prevalence           0.004626 0.033153   0.3292   0.4372  0.16577  0.02930
## Detection Rate       0.000000 0.003084   0.2244   0.3470  0.08096  0.01002
## Detection Prevalence 0.000771 0.003084   0.3045   0.5598  0.11951  0.01234
## Balanced Accuracy    0.499613 0.546512   0.7810   0.7078  0.72108  0.66986
##                      Class: 9
## Sensitivity          0.000000
## Specificity          1.000000
## Pos Pred Value            NaN
## Neg Pred Value       0.999229
## Prevalence           0.000771
## Detection Rate       0.000000
## Detection Prevalence 0.000000
## Balanced Accuracy    0.500000
```

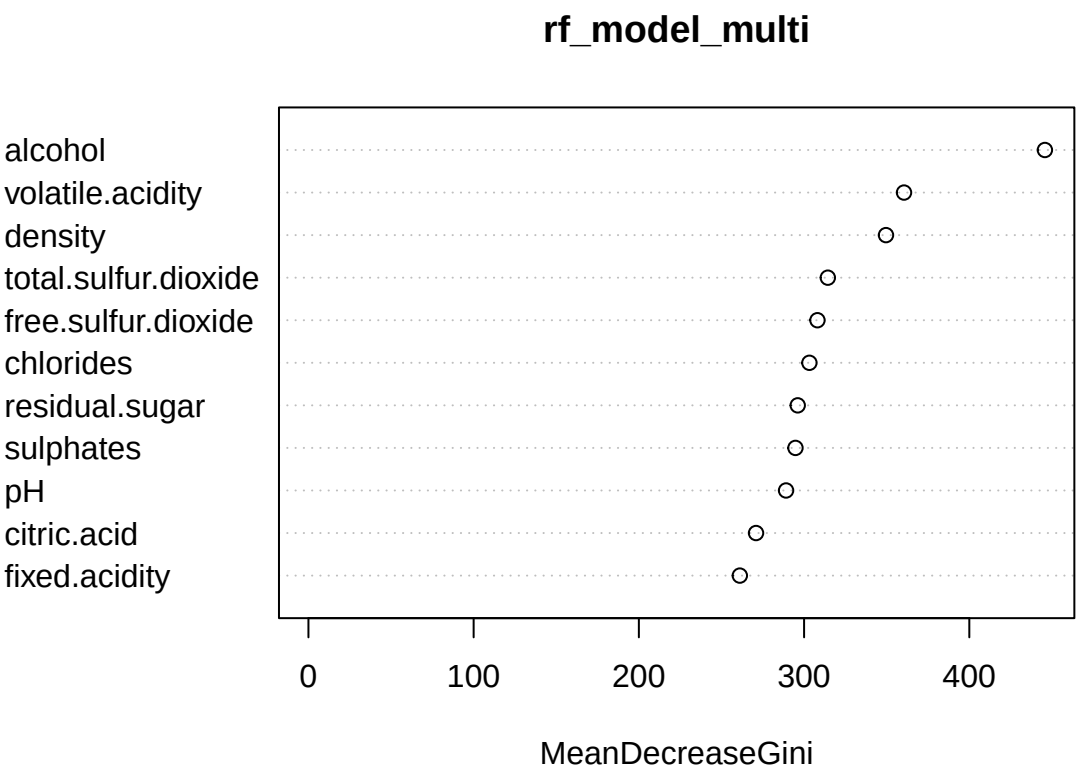
```
rf_mc <- multcap(response = test_y, predicted = rf_prob_multi)
rf_auc_multi <- HandTill2001::auc(rf_mc)
rf_auc_multi
```

```
## [1] 0.7860236
```

```
# Feature importance
randomForest::importance(rf_model_multi)
```

```
##                      MeanDecreaseGini
## fixed.acidity                261.0935
## volatile.acidity             360.4767
## citric.acid                  270.9080
## residual.sugar               296.1364
## chlorides                    303.2027
## free.sulfur.dioxide          308.0782
## total.sulfur.dioxide         314.3615
## density                      349.5564
## pH                           289.0847
## sulphates                    294.7690
## alcohol                      445.7432
```

```
varImpPlot(rf_model_multi)
```

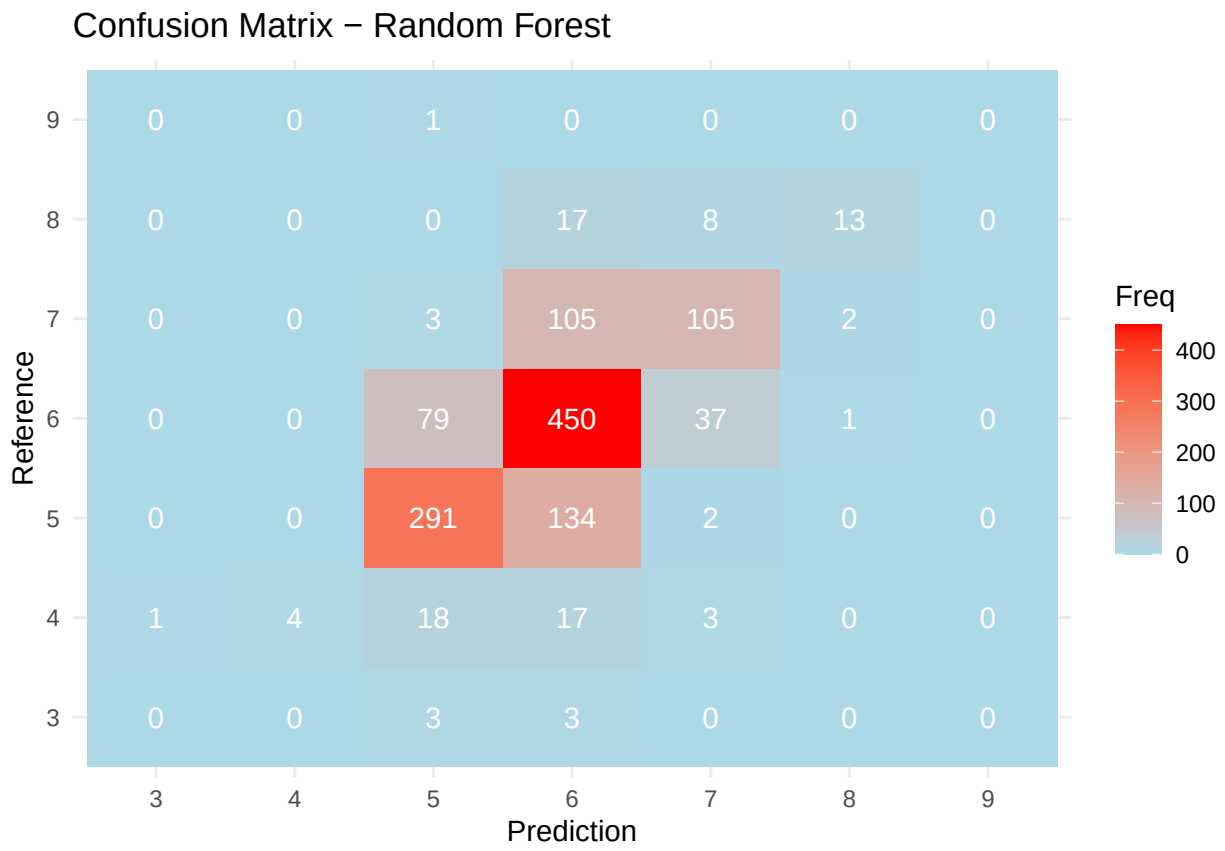



```
# Plot Confusion Matrix
# Example using RF results
cm_table <- as.data.frame(cm$table)
cm_table
```

```
##    Prediction Reference Freq
## 1           3         3    0
## 2           4         3    0
## 3           5         3    3
## 4           6         3    3
## 5           7         3    0
## 6           8         3    0
## 7           9         3    0
## 8           3         4    1
## 9           4         4    4
## 10          5         4   18
## 11          6         4   17
## 12          7         4    3
## 13          8         4    0
## 14          9         4    0
## 15          3         5    0
## 16          4         5    0
## 17          5         5  291
## 18          6         5  134
## 19          7         5    2
## 20          8         5    0
```

```
## 21          9         5    0
## 22          3         6    0
## 23          4         6    0
## 24          5         6   79
## 25          6         6  450
## 26          7         6   37
## 27          8         6    1
## 28          9         6    0
## 29          3         7    0
## 30          4         7    0
## 31          5         7    3
## 32          6         7  105
## 33          7         7  105
## 34          8         7    2
## 35          9         7    0
## 36          3         8    0
## 37          4         8    0
## 38          5         8    0
## 39          6         8   17
## 40          7         8    8
## 41          8         8   13
## 42          9         8    0
## 43          3         9    0
## 44          4         9    0
## 45          5         9    1
## 46          6         9    0
## 47          7         9    0
## 48          8         9    0
## 49          9         9    0
```

```r
ggplot(cm_table, aes(Prediction, Reference, fill = Freq)) +
  geom_tile() +
  geom_text(aes(label = Freq), color = "white") +
  scale_fill_gradient(low = "lightblue", high = "red") +
  theme_minimal() +
  labs(title = "Confusion Matrix - Random Forest")
```




```
#-----------------------------------------------------------------
# 4. ranger - Multiclass Classification
# ranger is a fast and efficient Random Forest implementation
#-----------------------------------------------------------------

# Fit model
ranger_model <- ranger(
  formula = quality ~ .,
  data = train,
  probability = TRUE,
  num.trees = 100,
  importance = "impurity"  # optional: gives Gini importance, aka Mean Decrease in Impurity for each fe
)

# Predict
ranger_pred <- predict(ranger_model, data = test)
ranger_pred
```

```
## Ranger prediction
##
## Type:                             Probability estimation
## Sample size:                      1297
## Number of independent variables:  11
```

```r
# Extract predicted probabilities
ranger_prob <- ranger_pred$predictions  # matrix of probabilities
head(ranger_prob)
```

```
##               3           4         5         6           7         8 9
## [1,] 0.004444444 0.015496032 0.7051429 0.2624167 0.012500000 0.0000000 0
## [2,] 0.002222222 0.009916667 0.3262778 0.4629563 0.179301587 0.0193254 0
## [3,] 0.000000000 0.043130952 0.3700952 0.5639960 0.022777778 0.0000000 0
## [4,] 0.003666667 0.012634921 0.7707698 0.2095952 0.003333333 0.0000000 0
## [5,] 0.010111111 0.029000000 0.6433929 0.3048294 0.012666667 0.0000000 0
## [6,] 0.002111111 0.015571429 0.4725992 0.4430357 0.066682540 0.0000000 0
```

```r
# Get class predictions
# Or if probability = FALSE, predict() will return predicted classes directly.
# Converts the index to the class label
ranger_class <- colnames(ranger_prob)[apply(ranger_prob, 1, which.max)]
ranger_class <- factor(ranger_class, levels = levels(test_y))

# Rename columns to match class labels
colnames(ranger_prob) <- levels(test_y)
# Confirm the internal match
all.equal(sort(colnames(ranger_prob)), sort(levels(test_y)))
```

```
## [1] TRUE
```

```r
# Evaluation
cm <- confusionMatrix(ranger_class, test_y)
cm$table
```

```
##           Reference
## Prediction   3   4   5   6   7   8   9
##          3   0   0   0   0   0   0   0
##          4   0   1   0   0   0   0   0
##          5   3  20 293  89   5   0   1
##          6   3  21 128 442 107  15   0
##          7   0   1   6  35 102  12   0
##          8   0   0   0   1   1  11   0
##          9   0   0   0   0   0   0   0
```

```r
ranger_mc <- multcap(response = test_y, predicted = ranger_prob)
ranger_auc <- HandTill2001::auc(ranger_mc)
ranger_auc
```

```
## [1] 0.7666276
```

```r
#------------------------------------------
# 5. XGBoost - Multiclass Classification
#------------------------------------------

# XGBoost expects:
# Label: integers from 0 to num_class - 1
```

```r
# train_matrix and test_matrix are xgb.DMatrix objects

# Convert factor to numeric (0-based indexing)
# XGBoost labels must start at 0 - not 1 or 3 or 5.

# Original factor with labels
original_labels <- levels(test_y)

train_label <- as.numeric(train_y) - 1
test_label  <- as.numeric(test_y) - 1

# Model matrix for features
train_matrix <- xgb.DMatrix(data = as.matrix(train_x), label = train_label)
test_matrix  <- xgb.DMatrix(data = as.matrix(test_x), label = test_label)

num_class = length(levels(wines_class$quality))

# Train model
xgb_model_multi <- xgboost(data = train_matrix,
                           objective = "multi:softprob", # always used for multiclass probability outpu
                           num_class = num_class,
                           nrounds = 100,
                           verbose = 0)

# Get raw predictions, predicted probabilities
xgb_prob_multi <- predict(xgb_model_multi, test_matrix)

# Reshape into matrix: rows = samples, columns = class probabilities
xgb_prob_multi <- matrix(xgb_prob_multi, ncol = num_class, byrow = TRUE)

# Due to floating-point precision errors, especially when reshaping the output
# Normalize each row to sum to 1 for multcap() to work
xgb_prob_multi <- xgb_prob_multi / rowSums(xgb_prob_multi)

# Predicted classes,
# Or use objective = "multi:softmax", predict() will return predicted classes directly.
# Predicted class index
pred_class_index <- max.col(xgb_prob_multi) - 1  # 0-based

# Map back to original labels
xgb_class_multi <- original_labels[pred_class_index + 1]
xgb_class_multi <- factor(xgb_class_multi, levels = original_labels)

# Label the columns with the original factor levels
colnames(xgb_prob_multi) <- levels(train_y)
# Confirm the internal match
all.equal(sort(colnames(xgb_prob_multi)), sort(levels(train_y)))
```

```
## [1] TRUE
```

```r
# Evaluation
cm <- confusionMatrix(xgb_class_multi, test_y)
cm$table
```

```
##           Reference
## Prediction   3   4   5   6   7   8   9
##          3   0   0   1   0   0   0   0
##          4   0   6   6   0   1   0   0
##          5   3  20 294  93   6   0   0
##          6   3  17 122 426  90  15   1
##          7   0   0   4  47 114  10   0
##          8   0   0   0   1   4  13   0
##          9   0   0   0   0   0   0   0
```

```
# Compute Multiclass AUC
# Create multcap object and compute AUC
xgb_mc <- multcap(response = test_y, predicted = xgb_prob_multi)
xgb_auc_multi <-HandTill2001::auc(xgb_mc)
xgb_auc_multi
```

```
## [1] 0.779817
```

```
# Summary
auc_comparison_multi <- tibble(
  Model    = c("Multinomial Logistic", "Ordinal Logistic", "Random Forest", "Random Forest - ranger", "
  AUC      = c(multinom_auc,ordinal_auc, rf_auc_multi, ranger_auc, xgb_auc_multi),
)

auc_comparison_multi
```

```
## # A tibble: 5 x 2
##   Model                    AUC
##   <chr>                  <dbl>
## 1 Multinomial Logistic   0.704
## 2 Ordinal Logistic       0.686
## 3 Random Forest          0.786
## 4 Random Forest - ranger 0.767
## 5 XGBoost                0.780
```