Capstone Own Project

José Andrés Sumano

2024-09-10

####Executive Summary: According to the instructions provided, I developed two machine learning algorithms to predict wine quality based on 11 independent variables, which included alcohol, sulfates, pH, density, among others. The two models performed were decision tree and random forest. I also performed hyperparameter tuning to test better tuning parameters for each model, resulting in four final models. It is important to note that before developing the models, I performed an extensive visual analysis of the data and studied correlation between variables. I compared the performance metrics of the 4 four models (decision tree, tuned decision tree, random forest, and tuned random forest) to check which model performed best. The model which performs best is the random forest model. However, the tuned random forest model is pretty close in performance. Regarding variable importante, alcohol is the most determinant variable, followed by sulphates.

###Methods and Analysis

###The task was to develop models which predicted wine quality using the independent variables contained in the data set. First, I analyzed my data set, checking structure, type of variables, presence of missing values, etc. Afterwards, I visually inspected the data using univariate plots and bivariate plots. Later, I checked the correlation between variables using correlation heatmaps and other functions. Before developing the models, I developed a new variable called quality_class which allowed me to treat quality as a factor. Now, I divided my data intro training and test set. I used the training set for training the models and the test set to test my models. I applied two different models: decision tree and random forest. I performed hyperparameter tuning on each of the models to try different sets of parameters. I show the results and graphs of all four models (for example, visualizing the decision tree or checking variable importante in random forest). Finally, I compared the performance metrics of the four models (results are shown in table format and also in a graph).

###Results###

####The results show that the best performing model is the random forest model (with a F1 score of 0.7946), followed by the . tuned random forest model (with a F1 score of 0.7928). Regarding variable importante, the most determinant variable (according to the random forest model) is alcohol, followed by sulphates.

###Conclusion###

###The best performing model is the random forest model. The most determinant variables for predicting wine quality are alcohol and sulphates. Perhaps, other machine learning algorithms like XGBoost or Adaboost could provide better results. However, the performance of the model is satisfactory with an Accuracy of 0.8079, a Precision of 0.7963, a Recall of 0.7892, and a F1 Score of 0.7928.

```
#This code allows me to load my data frame and start inspecting the structure,
#dimensions and characteristics of the data.

options(warn=-1)
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 ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(ggplot2)
data <- read.csv("winequality-red.csv", sep = ";")
wine <- data
head(data)
```

```
##   fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
## 1           7.4             0.70        0.00            1.9     0.076
## 2           7.8             0.88        0.00            2.6     0.098
## 3           7.8             0.76        0.04            2.3     0.092
## 4          11.2             0.28        0.56            1.9     0.075
## 5           7.4             0.70        0.00            1.9     0.076
## 6           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
```

```
tail(data)
```

```
##      fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
## 1594           6.8            0.620        0.08            1.9     0.068
## 1595           6.2            0.600        0.08            2.0     0.090
## 1596           5.9            0.550        0.10            2.2     0.062
## 1597           6.3            0.510        0.13            2.3     0.076
## 1598           5.9            0.645        0.12            2.0     0.075
## 1599           6.0            0.310        0.47            3.6     0.067
##      free.sulfur.dioxide total.sulfur.dioxide density   pH sulphates alcohol
## 1594                  28                   38 0.99651 3.42      0.82     9.5
## 1595                  32                   44 0.99490 3.45      0.58    10.5
## 1596                  39                   51 0.99512 3.52      0.76    11.2
## 1597                  29                   40 0.99574 3.42      0.75    11.0
## 1598                  32                   44 0.99547 3.57      0.71    10.2
## 1599                  18                   42 0.99549 3.39      0.66    11.0
##      quality
## 1594       6
## 1595       5
## 1596       6
```

```
## 1597       6
## 1598       5
## 1599       6
```

```
dim(data)
```

```
## [1] 1599   12
```

```
str(data)
```

```
## 'data.frame':    1599 obs. of  12 variables:
##  $ 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 ...
```

```
summary(data)
```

```
##  fixed.acidity   volatile.acidity  citric.acid    residual.sugar
##  Min.   : 4.60   Min.   :0.1200   Min.   :0.000   Min.   : 0.900
##  1st Qu.: 7.10   1st Qu.:0.3900   1st Qu.:0.090   1st Qu.: 1.900
##  Median : 7.90   Median :0.5200   Median :0.260   Median : 2.200
##  Mean   : 8.32   Mean   :0.5278   Mean   :0.271   Mean   : 2.539
##  3rd Qu.: 9.20   3rd Qu.:0.6400   3rd Qu.:0.420   3rd Qu.: 2.600
##  Max.   :15.90   Max.   :1.5800   Max.   :1.000   Max.   :15.500
##    chlorides       free.sulfur.dioxide total.sulfur.dioxide    density
##  Min.   :0.01200   Min.   : 1.00       Min.   :  6.00       Min.   :0.9901
##  1st Qu.:0.07000   1st Qu.: 7.00       1st Qu.: 22.00       1st Qu.:0.9956
##  Median :0.07900   Median :14.00       Median : 38.00       Median :0.9968
##  Mean   :0.08747   Mean   :15.87       Mean   : 46.47       Mean   :0.9967
##  3rd Qu.:0.09000   3rd Qu.:21.00       3rd Qu.: 62.00       3rd Qu.:0.9978
##  Max.   :0.61100   Max.   :72.00       Max.   :289.00       Max.   :1.0037
##        pH          sulphates         alcohol         quality
##  Min.   :2.740   Min.   :0.3300   Min.   : 8.40   Min.   :3.000
##  1st Qu.:3.210   1st Qu.:0.5500   1st Qu.: 9.50   1st Qu.:5.000
##  Median :3.310   Median :0.6200   Median :10.20   Median :6.000
##  Mean   :3.311   Mean   :0.6581   Mean   :10.42   Mean   :5.636
##  3rd Qu.:3.400   3rd Qu.:0.7300   3rd Qu.:11.10   3rd Qu.:6.000
##  Max.   :4.010   Max.   :2.0000   Max.   :14.90   Max.   :8.000
```

```
if (!require(caret)) install.packages("caret", dependencies = TRUE)
```

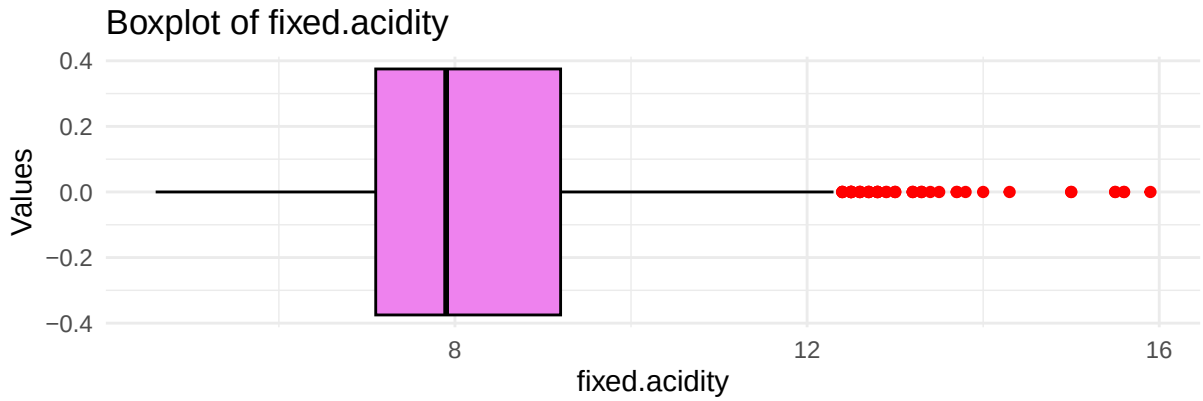
```
## Loading required package: caret
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
```

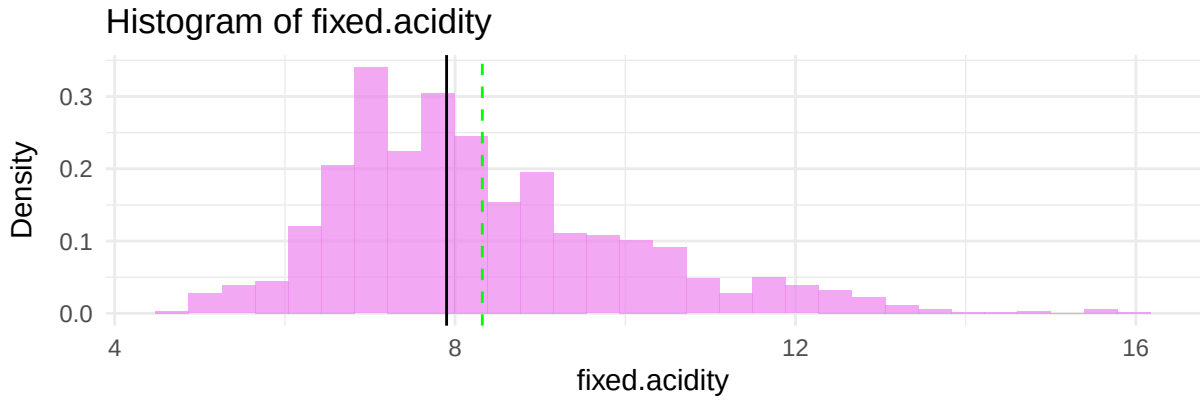
```
##     lift
```

```
library(caret)

#This code allows me to visually inspect the data, specially through histograms
#and boxplots.

install.packages("patchwork")
```

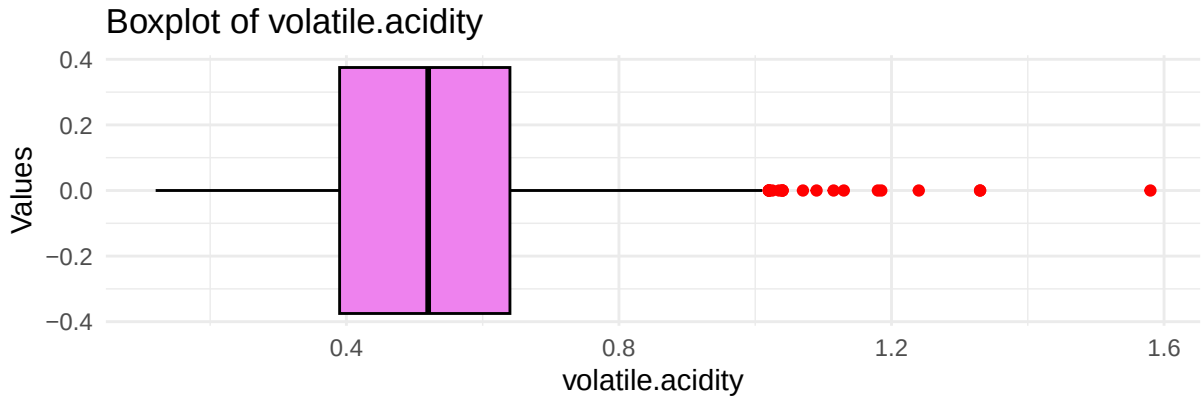
```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.4'
## (as 'lib' is unspecified)
```

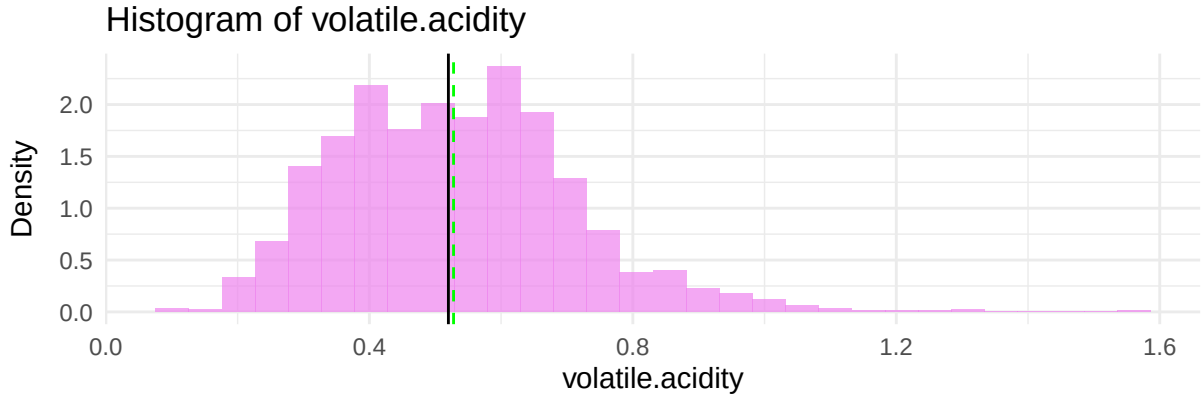
```
histogram_boxplot <- function(data, feature, bins = 30) {

  boxplot <- ggplot(data, aes_string(x = feature)) +
    geom_boxplot(fill = "violet", color = "black", outlier.colour = "red") +
    labs(title = paste("Boxplot of", feature), x = feature, y = "Values") +
    theme_minimal()


  histogram <- ggplot(data, aes_string(x = feature)) +
    geom_histogram(aes(y = ..density..), fill = "violet", bins = bins, alpha = 0.7) +
    geom_vline(aes(xintercept = mean(get(feature))), color = "green", linetype = "dashed") +
    geom_vline(aes(xintercept = median(get(feature))), color = "black", linetype = "solid") +
    labs(title = paste("Histogram of", feature), x = feature, y = "Density") +
    theme_minimal()


  library(patchwork)
  combined_plot <- boxplot / histogram


  print(combined_plot)
}


histogram_boxplot(data, 'fixed.acidity')
```



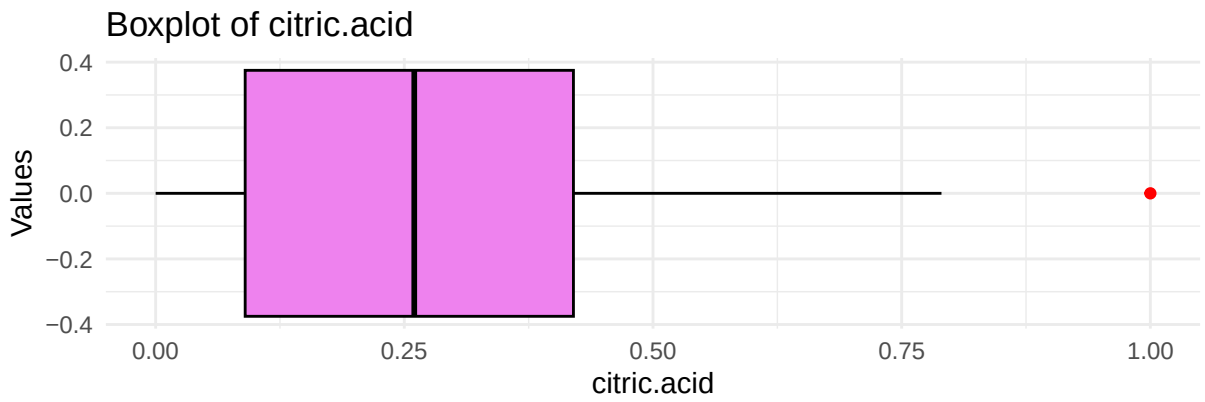


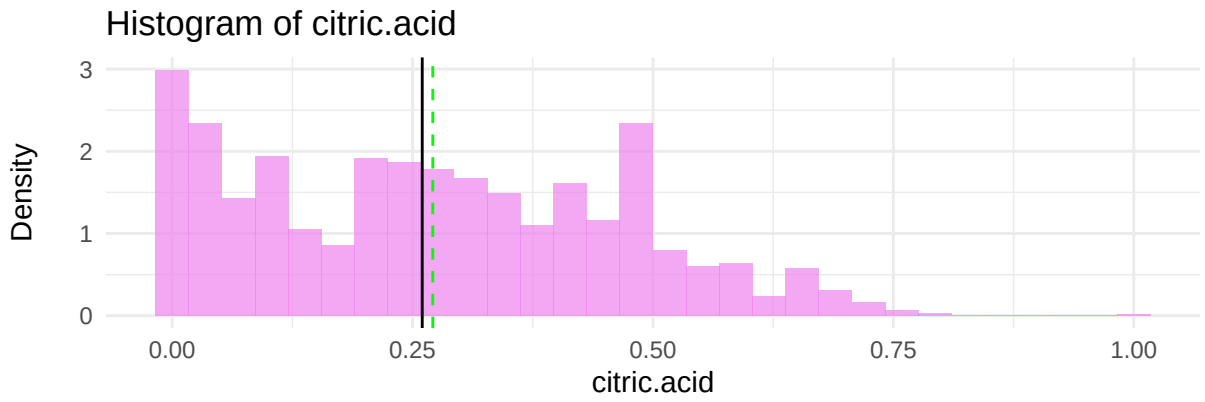

```
histogram_boxplot(data, 'volatile.acidity')
```



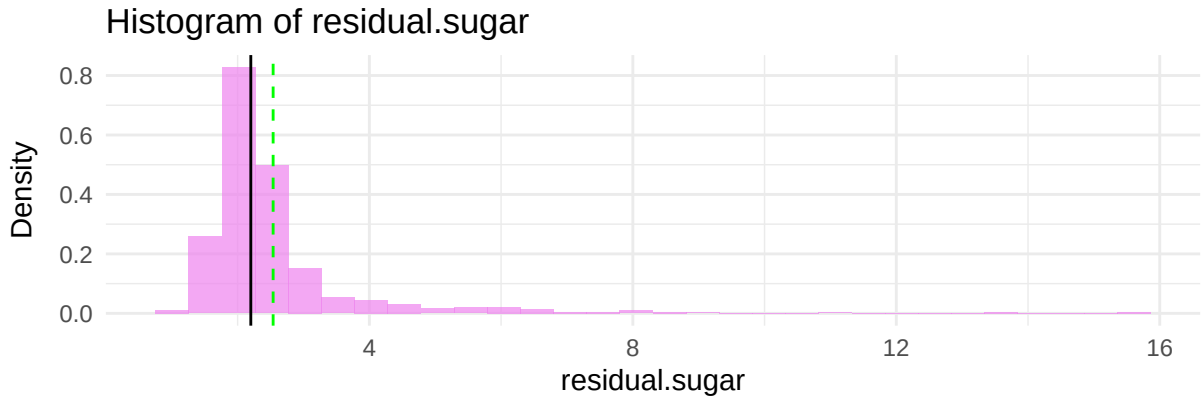


```
histogram_boxplot(data, 'citric.acid')
```

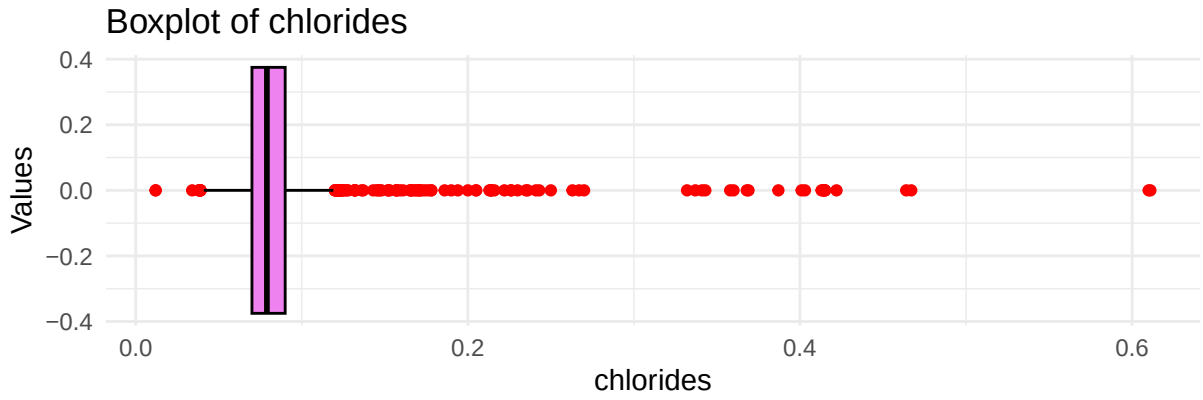


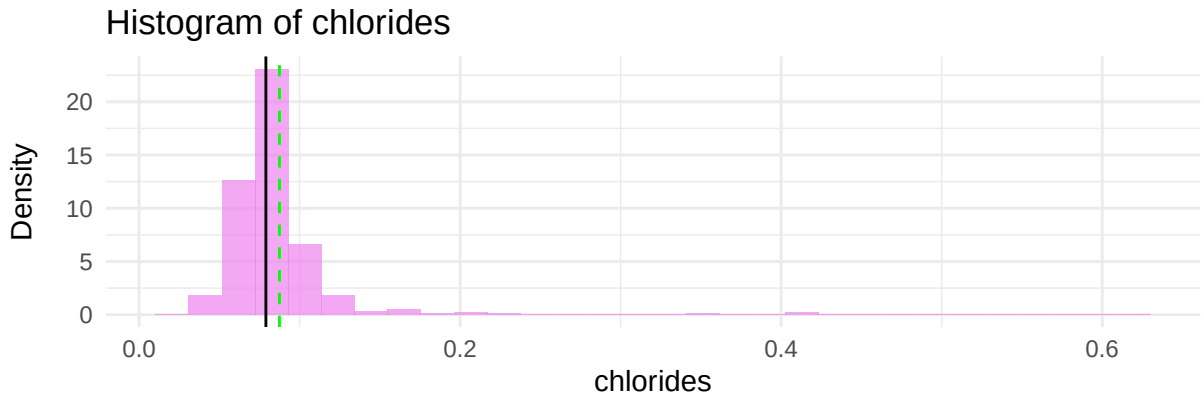



```
histogram_boxplot(data, 'residual.sugar')
```

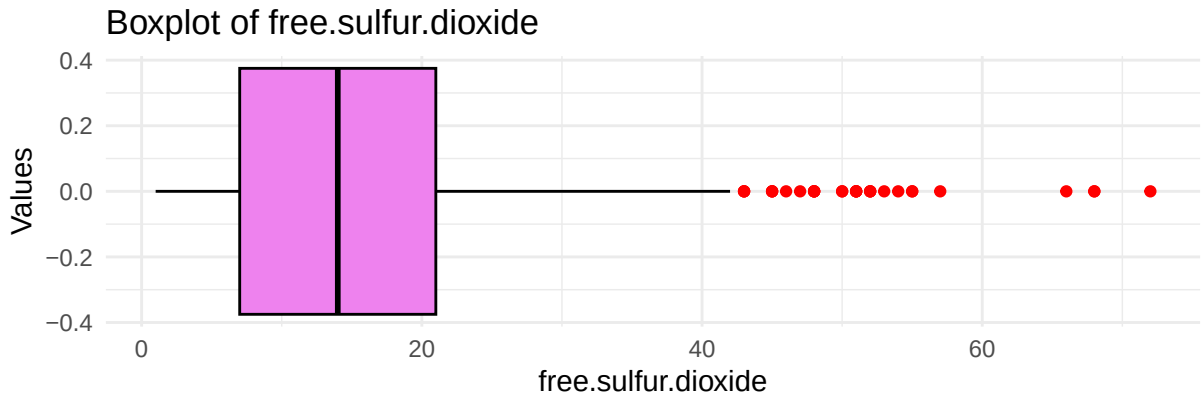


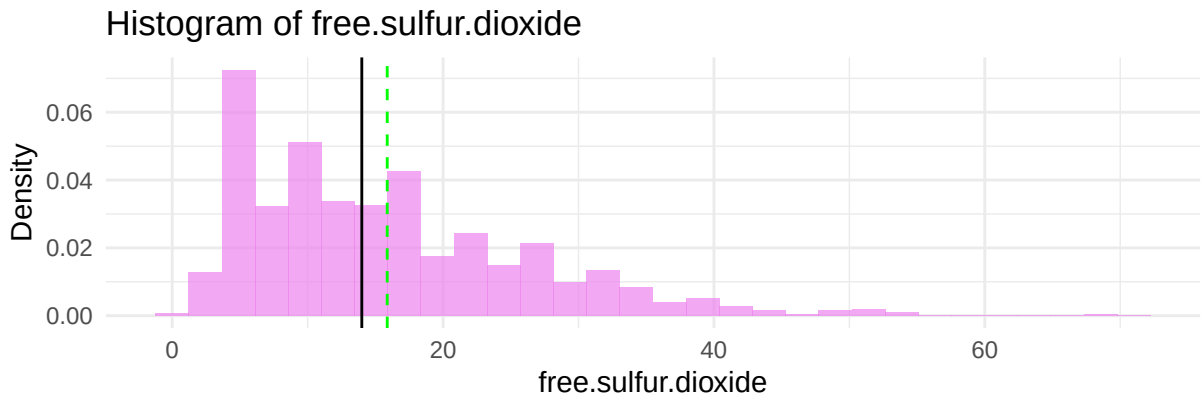

```
histogram_boxplot(data, 'chlorides')
```



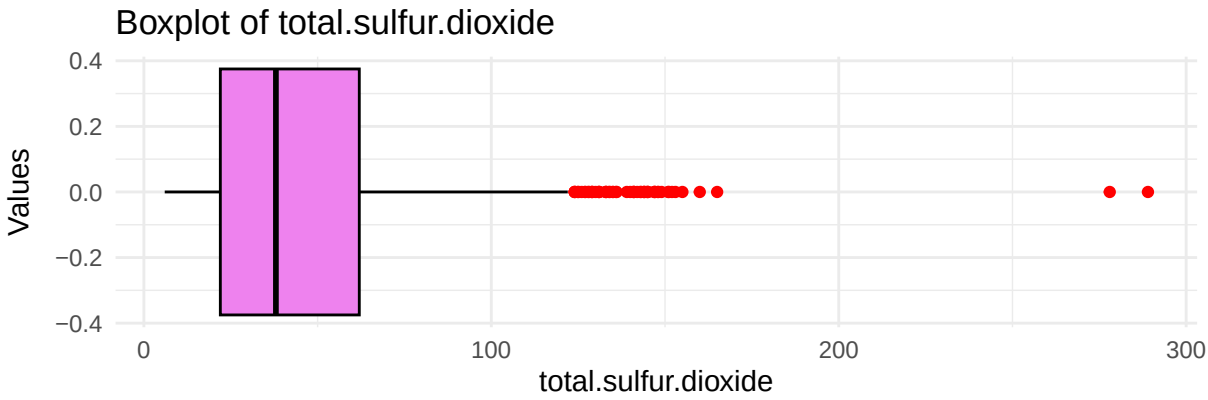


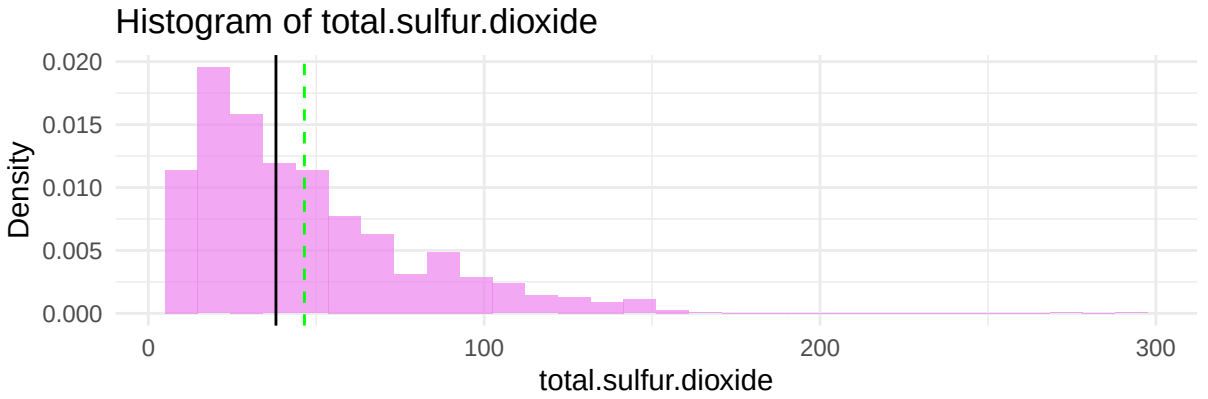
```
histogram_boxplot(data, 'free.sulfur.dioxide')
```



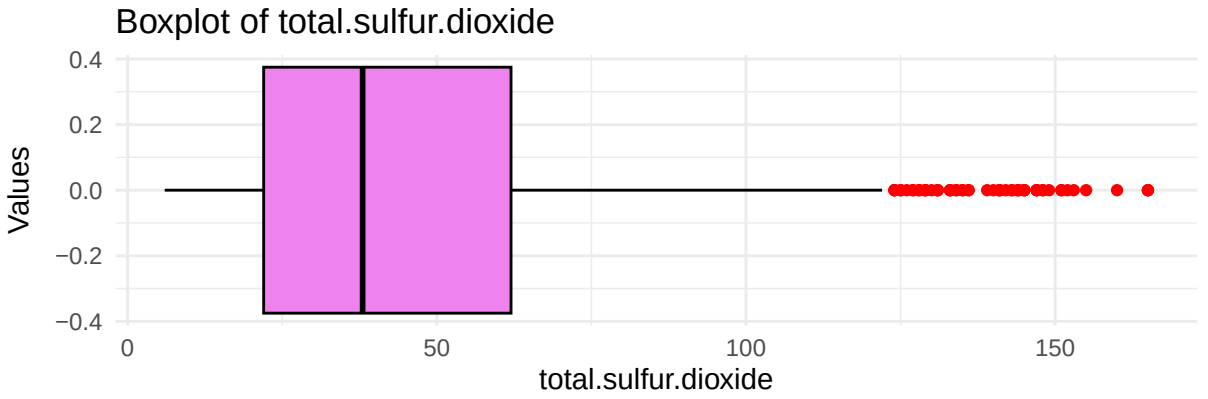


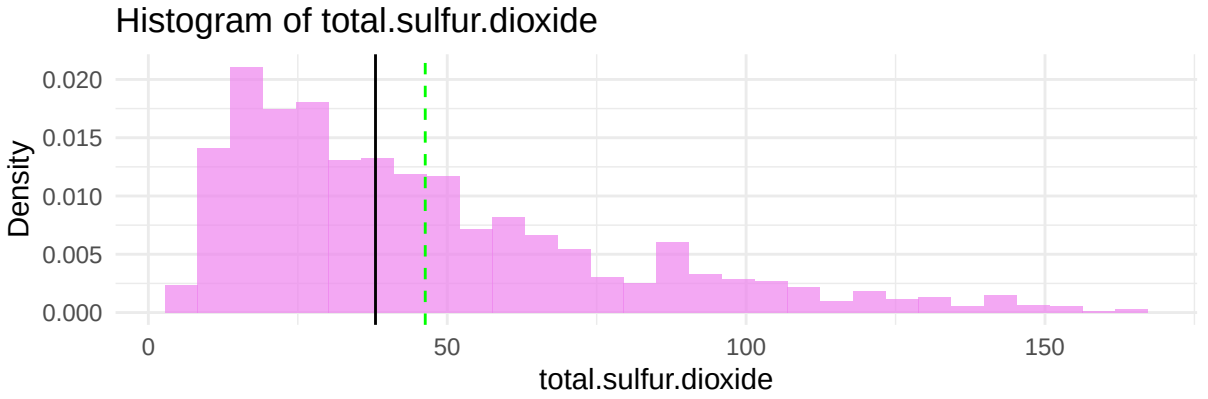

```
histogram_boxplot(data, 'total.sulfur.dioxide')
```



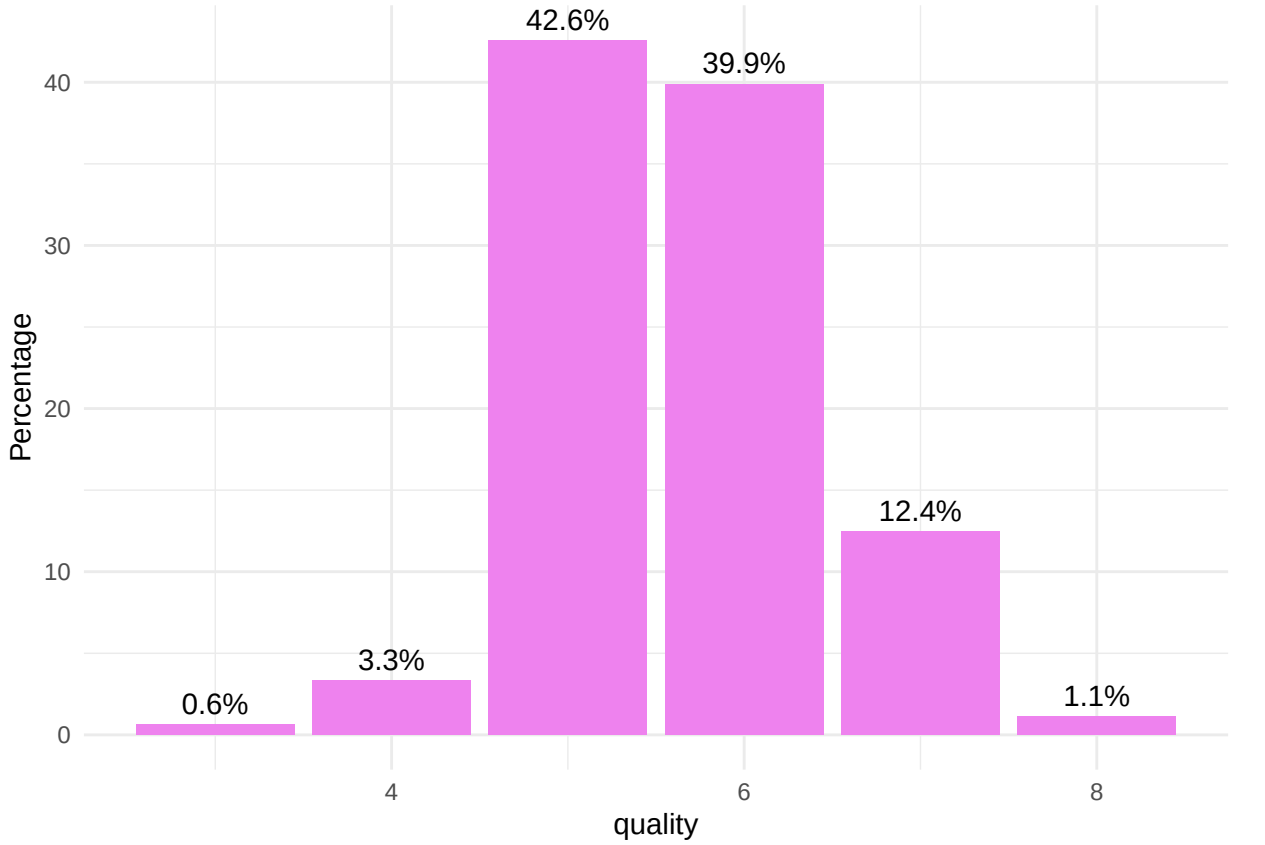



```
data$`total.sulfur.dioxide` <- pmin(data$`total.sulfur.dioxide`, 165)
histogram_boxplot(data, 'total.sulfur.dioxide')
```





```r
labeled_barplot <- function(data, feature, perc = FALSE, n = NULL) {
  if (perc) {
    data %>%
      count(!!sym(feature)) %>%
      mutate(percentage = n / sum(n) * 100) %>%
      ggplot(aes_string(x = feature, y = ifelse(perc, "percentage", "n"))) +
      geom_bar(stat = "identity", fill = "violet") +
      geom_text(aes(label = sprintf("%.1f%%", percentage)), vjust = -0.5) +
      theme_minimal() +
      labs(x = feature, y = ifelse(perc, "Percentage", "Count"))
  } else {
    data %>%
      count(!!sym(feature)) %>%
      ggplot(aes_string(x = feature, y = "n")) +
      geom_bar(stat = "identity", fill = "violet") +
      geom_text(aes(label = n), vjust = -0.5) +
      theme_minimal() +
      labs(x = feature, y = "Count")
  }
}

labeled_barplot(data, "quality", perc = TRUE)
```




```r
#this code creates a variable called quality_class which will help me
#in the machine learning models.

range(data$quality)
```

```
## [1] 3 8
```

```
bins <- c(min(data$quality), 6, max(data$quality) + 1)
labels <- c('non-premium', 'premium')


data$quality_class <- cut(data$quality, breaks = bins, labels = labels, right = FALSE)


sum(is.na(data$quality_class))
```

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

```
table(data$quality_class)
```

```
##
## non-premium     premium
##         744         855
```

```
install.packages(c("ggplot2", "dplyr", "corrplot", "GGally"))
```

```
## Installing packages into '/cloud/lib/x86_64-pc-linux-gnu-library/4.4'
## (as 'lib' is unspecified)
```

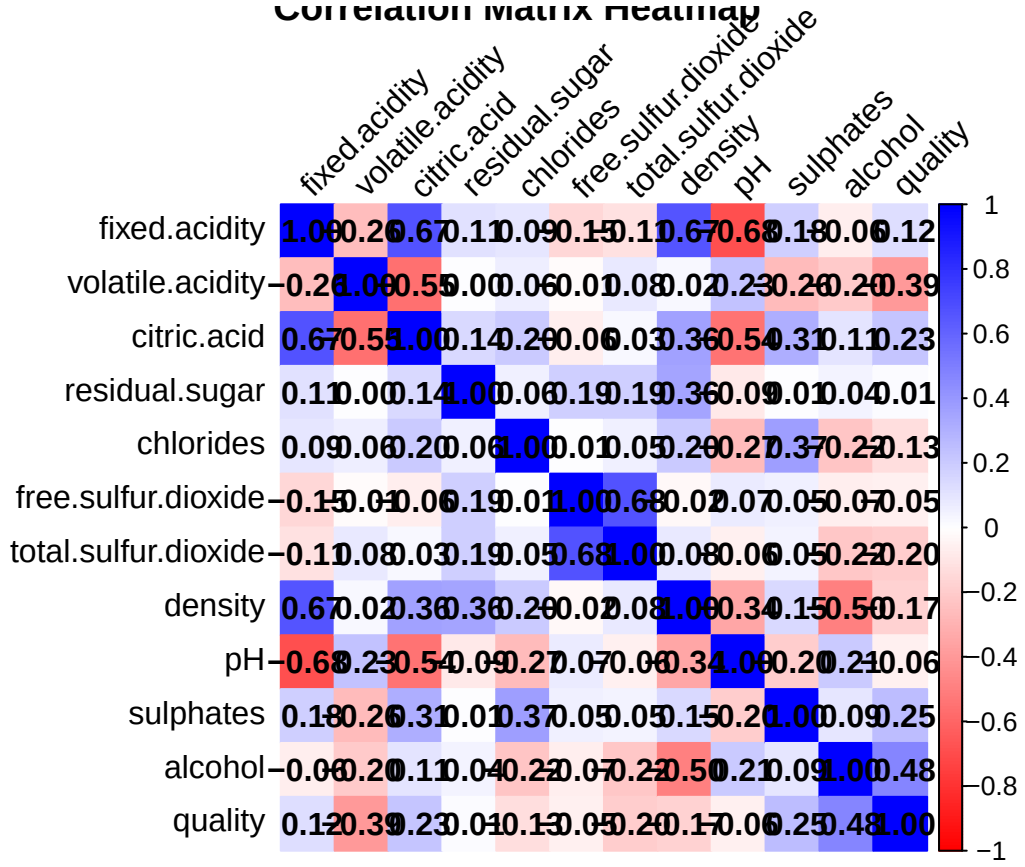
```
# Load libraries
library(ggplot2)
library(dplyr)
library(corrplot)
```

```
## corrplot 0.94 loaded
```

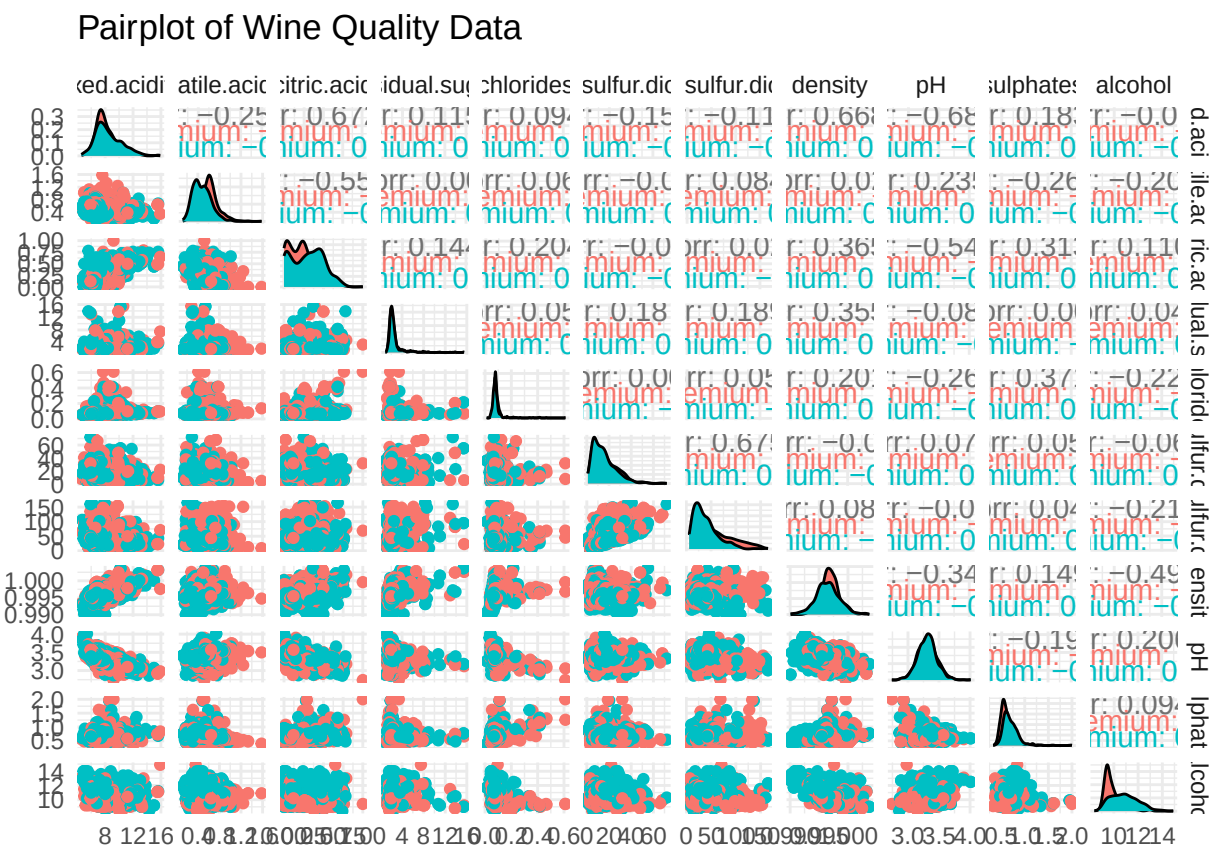
```
library(GGally)
```

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

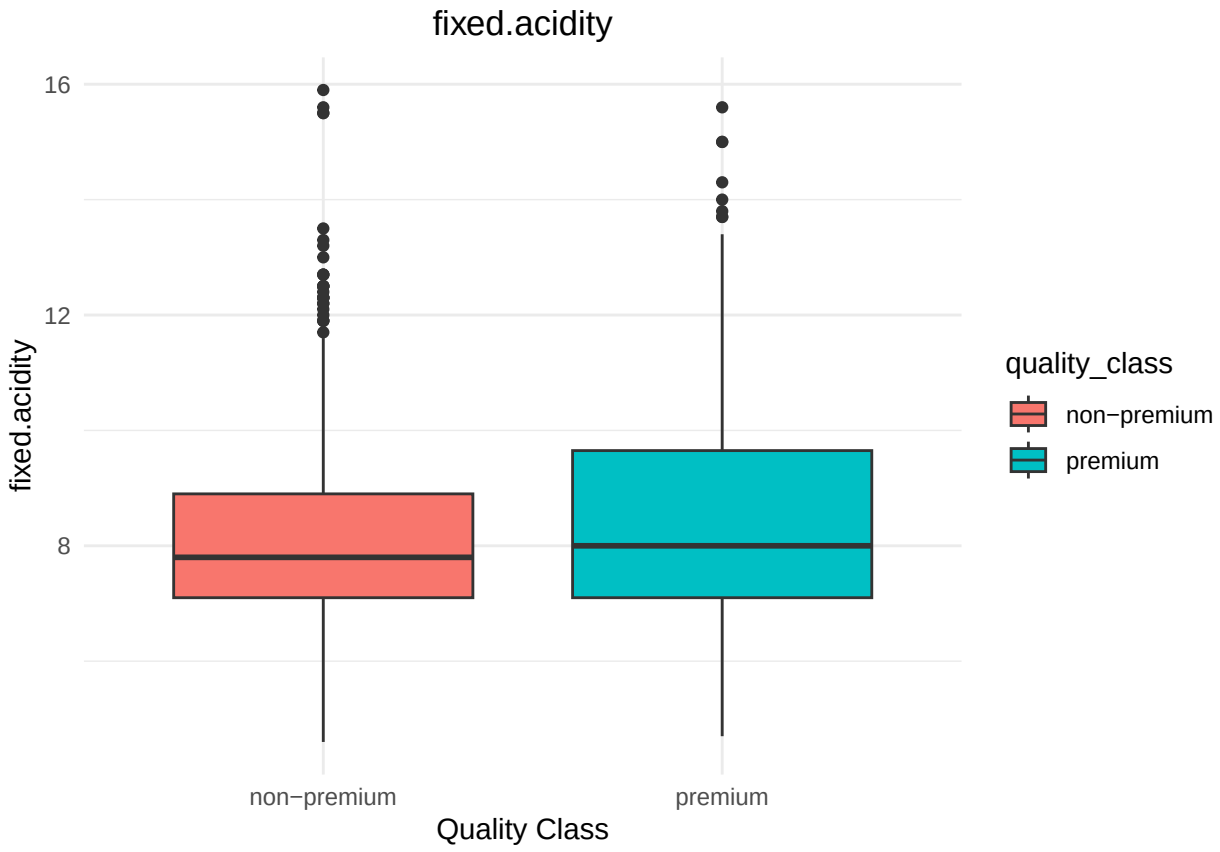
```
#this code allows me to develop correlation heat maps and pairs plots
numeric_data <- data %>%
  select_if(is.numeric)


cor_matrix <- cor(numeric_data, use = "complete.obs")


corrplot(cor_matrix, method = "color", col = colorRampPalette(c("red", "white", "blue"))(200),
         addCoef.col = "black", tl.col = "black", tl.srt = 45,
         title = "Correlation Matrix Heatmap")
```

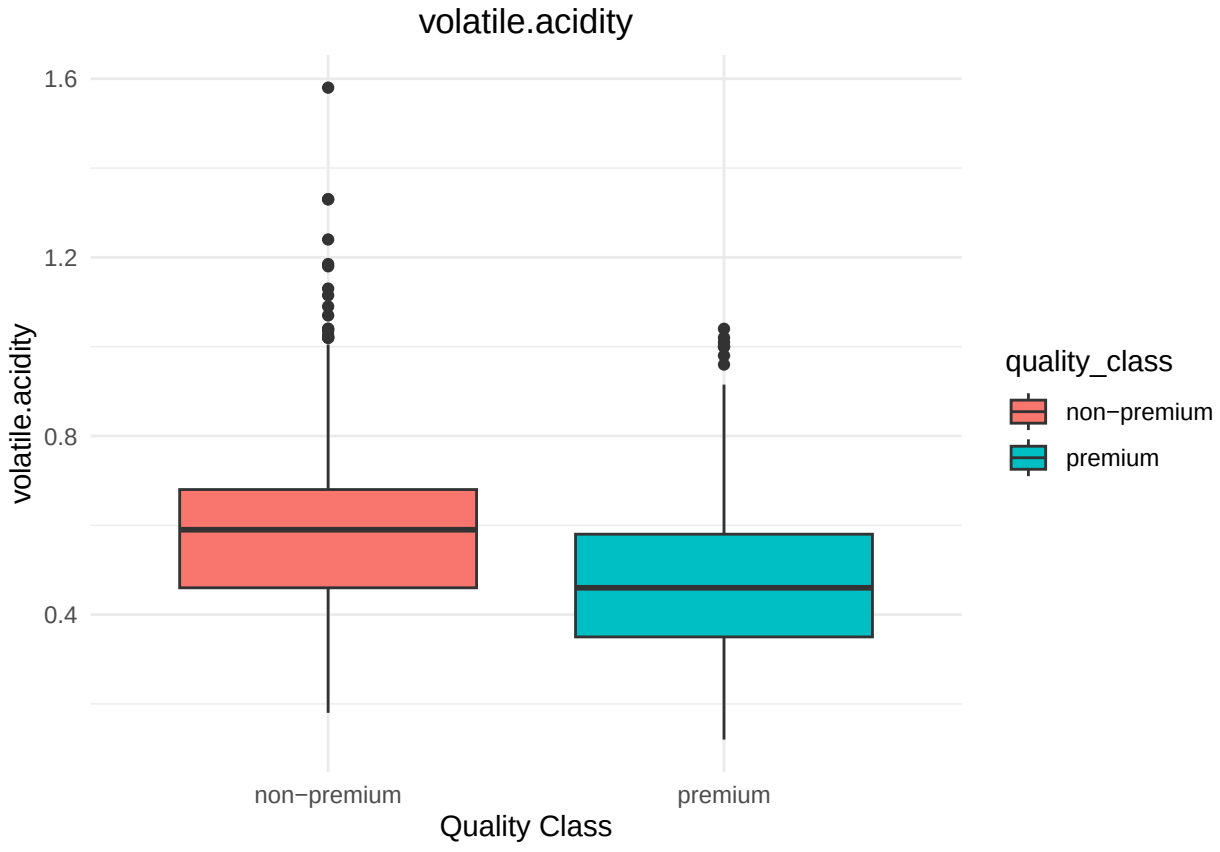


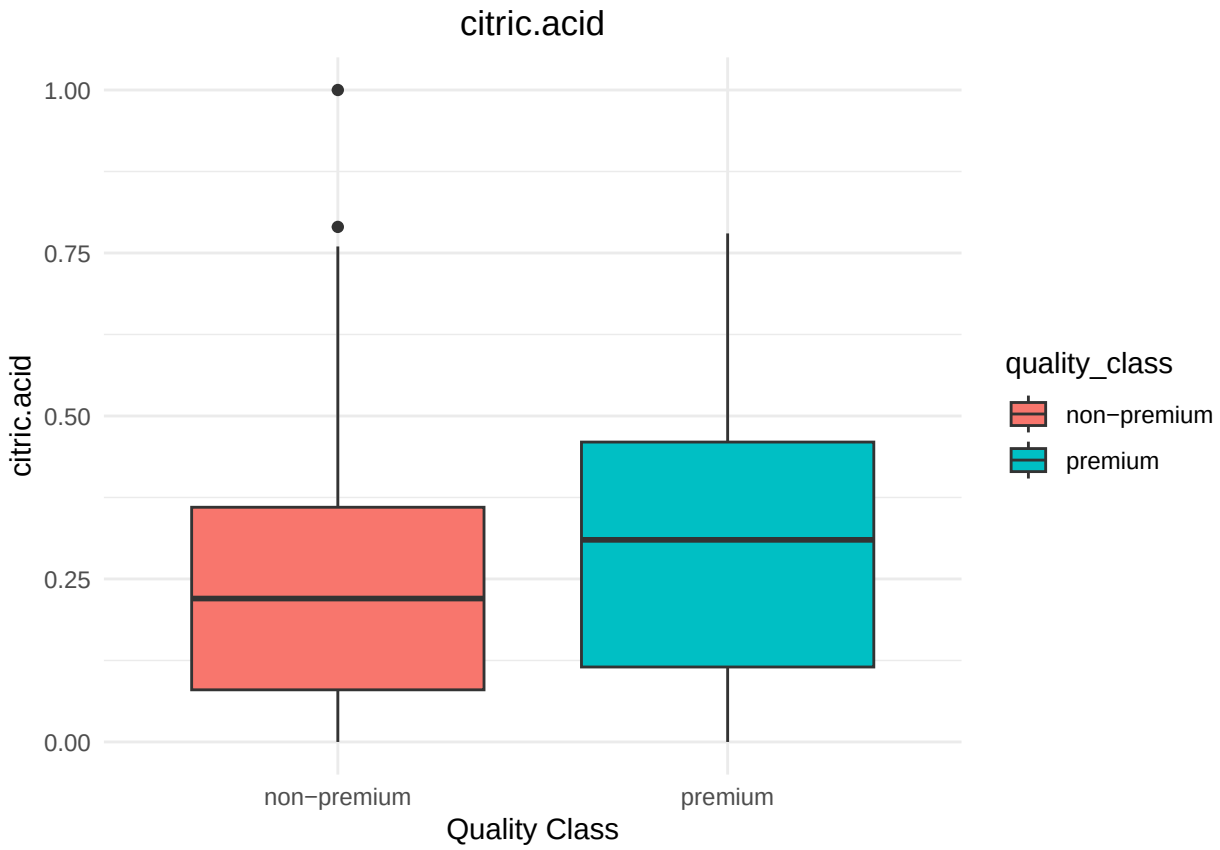

```
ggpairs(data, columns = 1:11, aes(color = quality_class)) +
  theme_minimal() +
  labs(title = "Pairplot of Wine Quality Data")
```



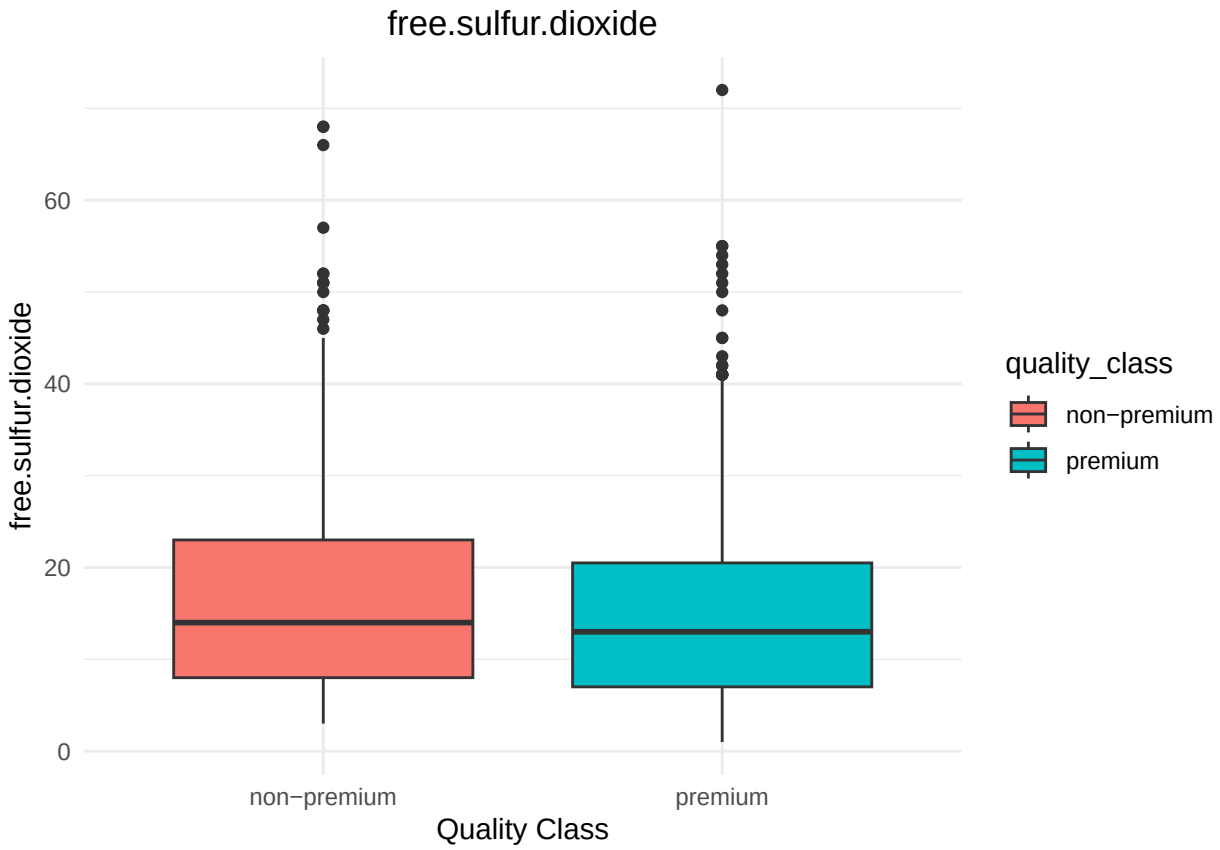

```
#now i am going to run some boxplots to visually analyze the relation between
#quality class and the other variables.

cols <- c('fixed.acidity', 'volatile.acidity', 'citric.acid')
par(mfrow = c(1, 3))
for (variable in cols) {
  print(ggplot(data, aes_string(x = 'quality_class', y = variable)) +
          geom_boxplot(aes(fill = quality_class), palette = "PuBu") +
          labs(title = variable, x = "Quality Class", y = variable) +
          theme_minimal() +
          theme(plot.title = element_text(hjust = 0.5)))
}
```

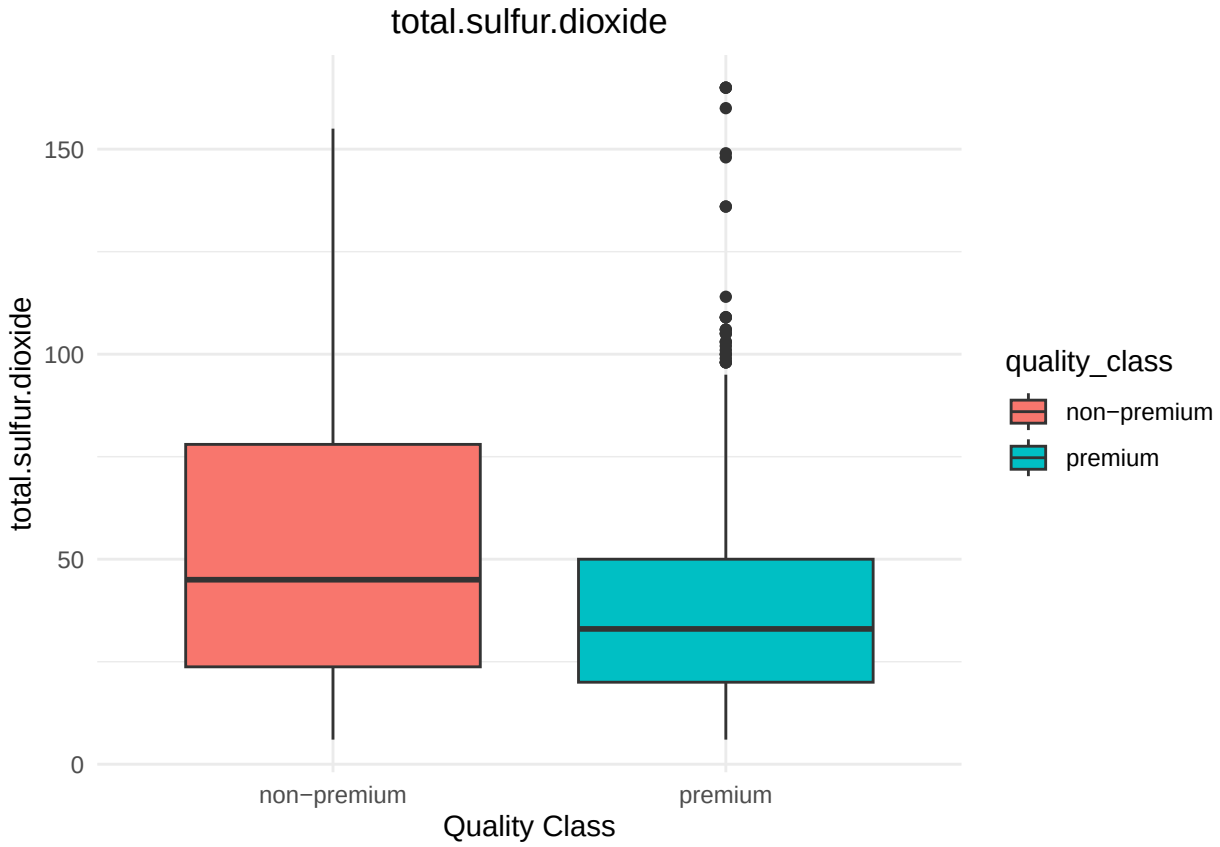

fixed.acidity
16
12
8
fixed.acidity
non−premium
premium
Quality Class
quality_class
non−premium
premium

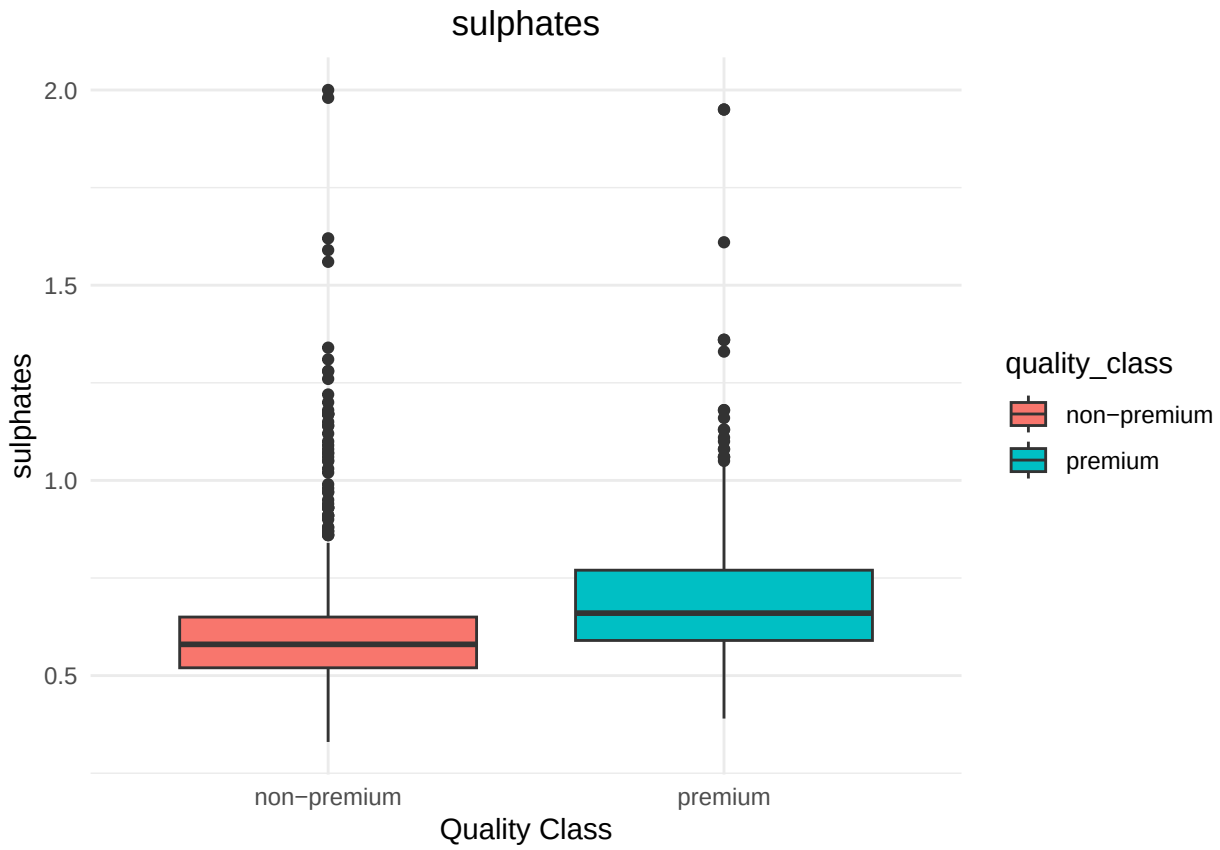

volatile.acidity
1.6
1.2
0.8
0.4
volatile.acidity
non−premium
premium
Quality Class
quality_class
non−premium
premium



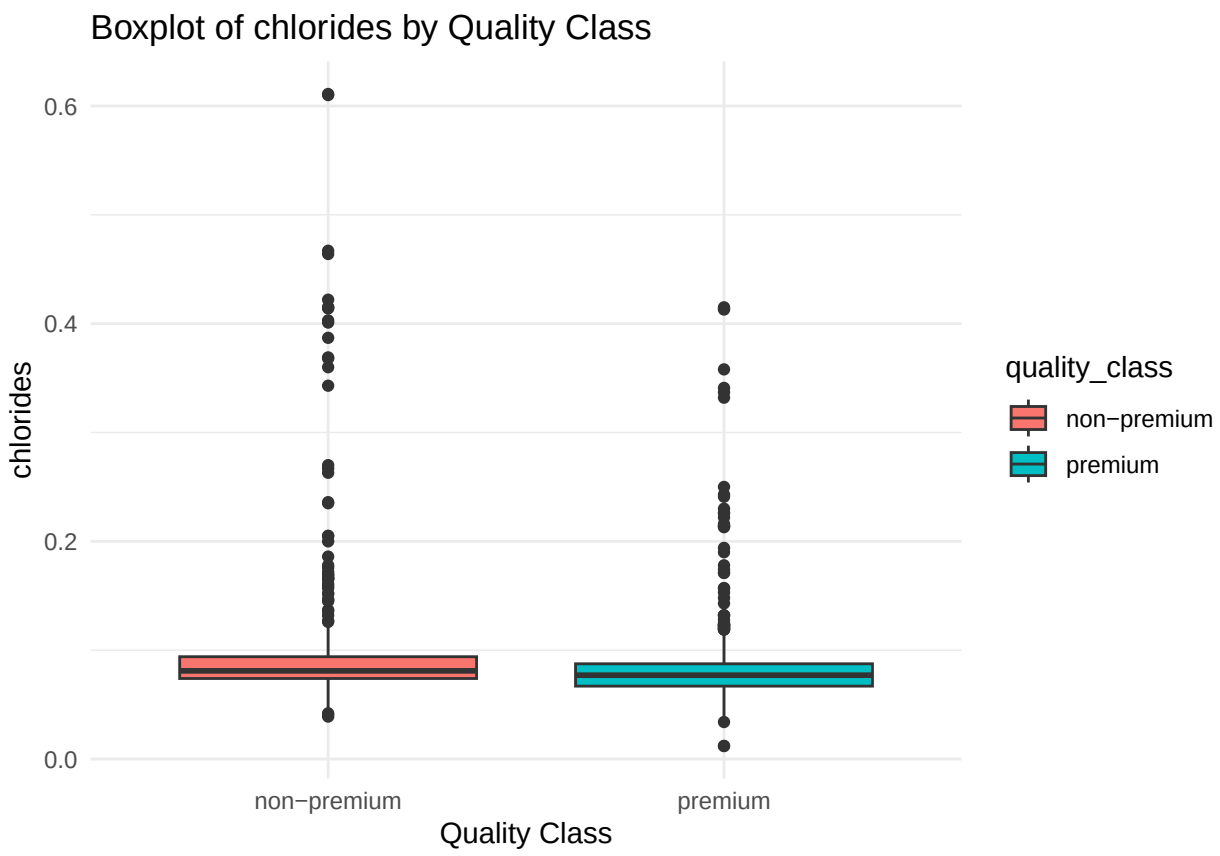

```
cols2 <- c('free.sulfur.dioxide', 'total.sulfur.dioxide', 'sulphates')
par(mfrow = c(1, 3))
for (variable in cols2) {
  print(ggplot(data, aes_string(x = 'quality_class', y = variable)) +
          geom_boxplot(aes(fill = quality_class), palette = "PuBu") +
          labs(title = variable, x = "Quality Class", y = variable) +
          theme_minimal() +
          theme(plot.title = element_text(hjust = 0.5)))
}
```


free.sulfur.dioxide
free.sulfur.dioxide
60
40
20
0
non−premium
premium
Quality Class
quality_class
non−premium
premium

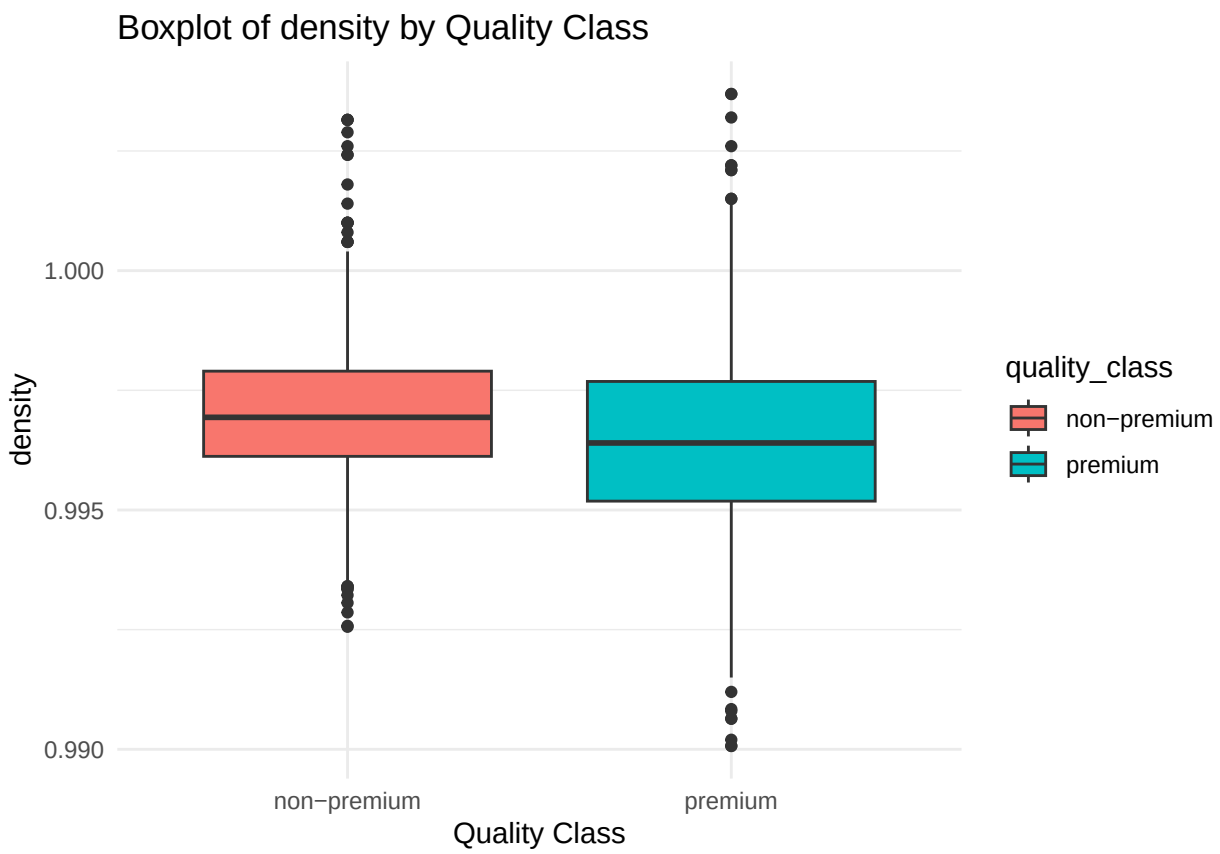

total.sulfur.dioxide
total.sulfur.dioxide
150
100
50
0
non−premium
premium
Quality Class
quality_class
non−premium
premium



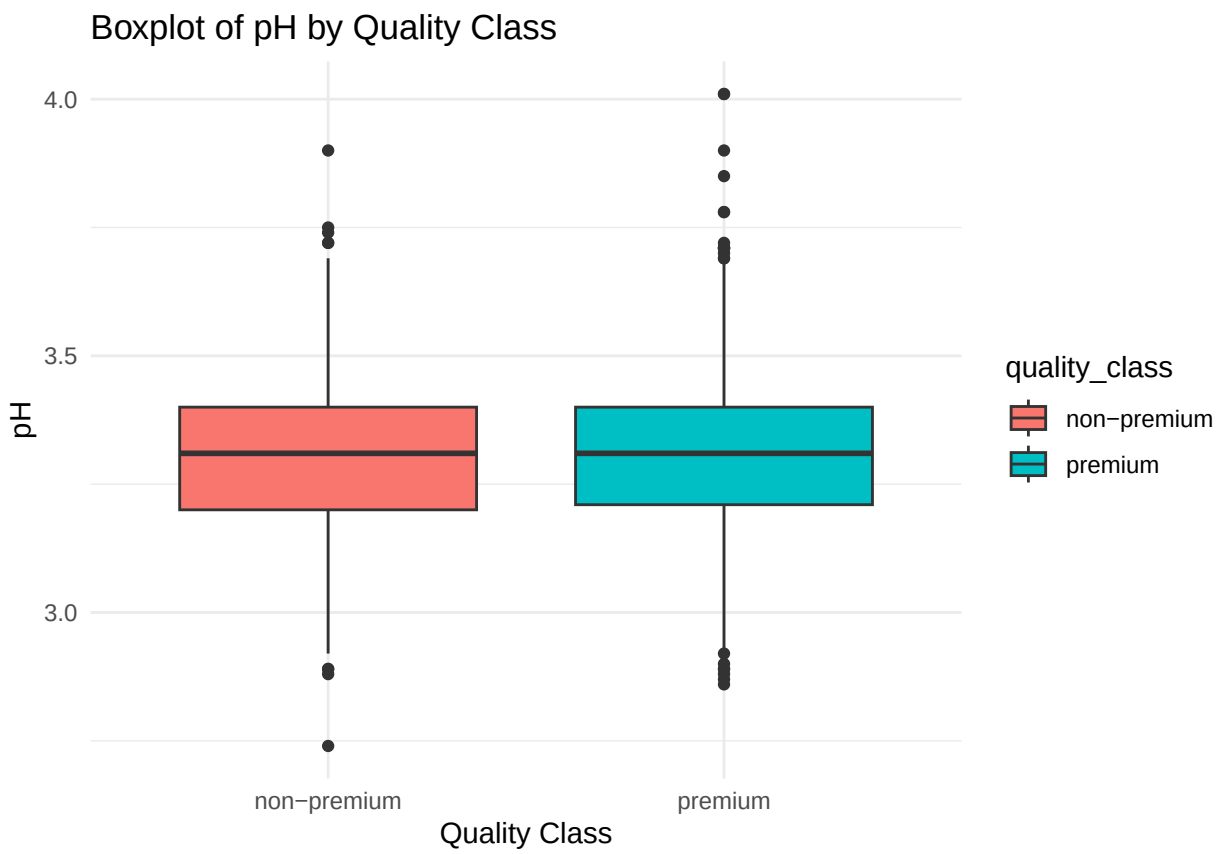

```
boxplot_quality <- function(variable) {
  print(ggplot(data, aes_string(x = 'quality_class', y = variable)) +
          geom_boxplot(aes(fill = quality_class), palette = "PuBu") +
          labs(title = paste("Boxplot of", variable, "by Quality Class"), x = "Quality Class", y = varia
          theme_minimal())
}


boxplot_quality('chlorides')
```



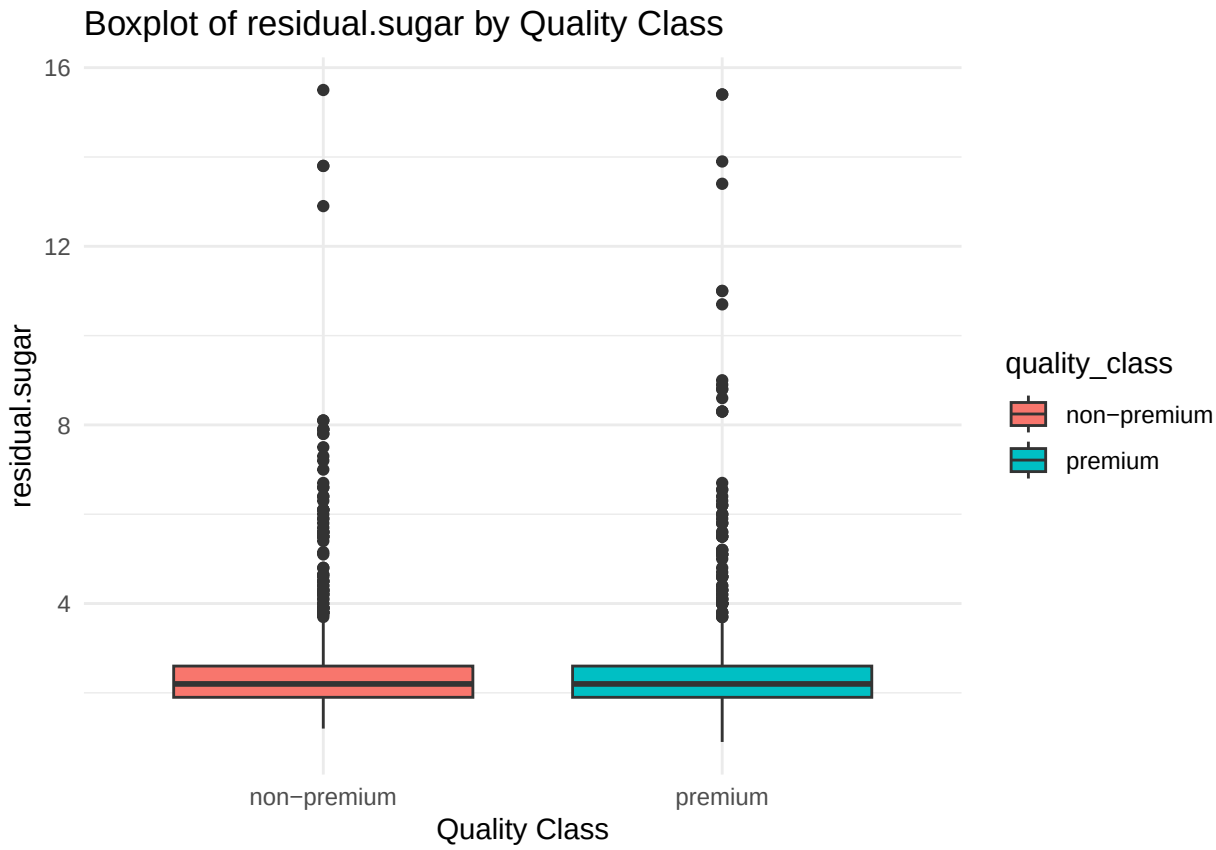

```
boxplot_quality('density')
```



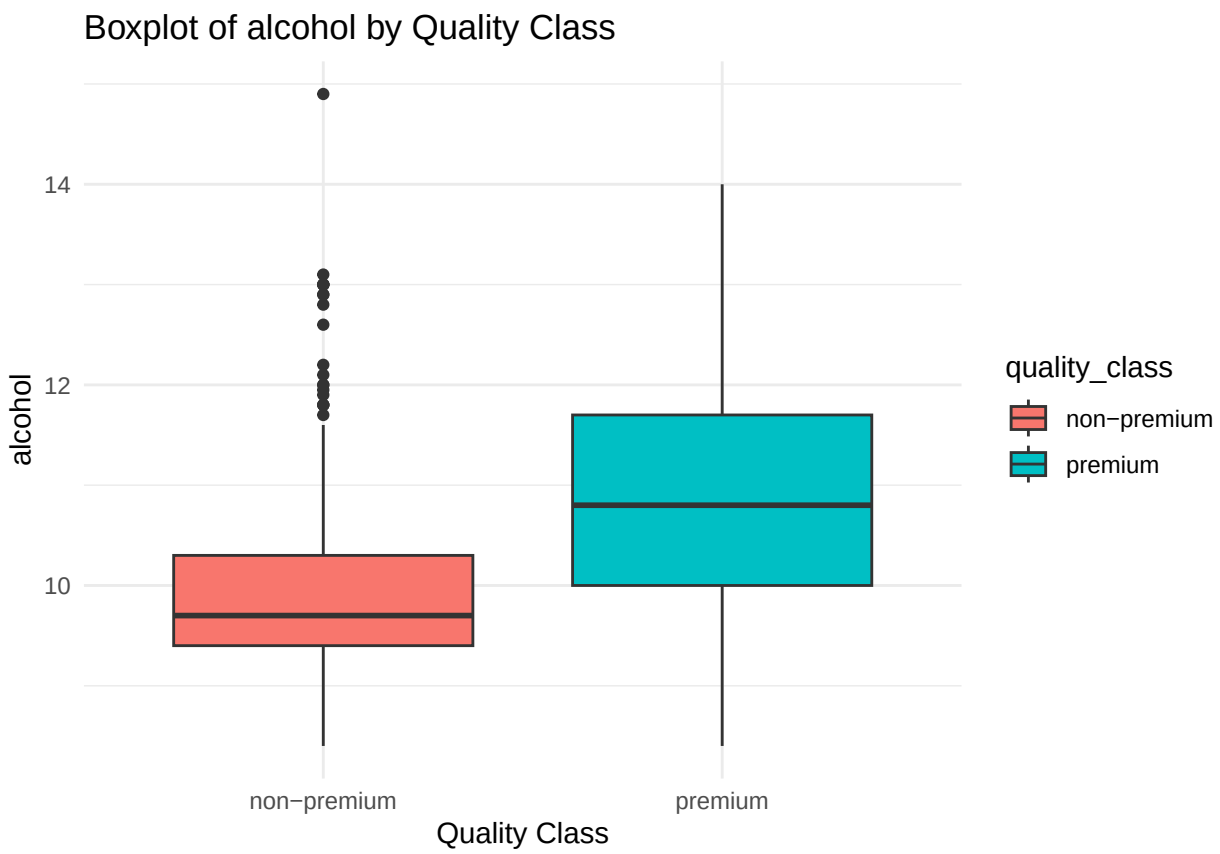

```
boxplot_quality('pH')
```




```
boxplot_quality('residual.sugar')
```




```
boxplot_quality('alcohol')
```




```
#The following code develops a decision tree model to predict wine quality
#using the independent variables.

if (!require(rpart)) install.packages("rpart", dependencies = TRUE)
```

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

```
if (!require(caret)) install.packages("caret", dependencies = TRUE)

library(rpart)
library(caret)

X <- data %>% select(-quality_class, -quality)  # Feature matrix
y <- as.factor(data$quality_class)  # Target variable

set.seed(1)
trainIndex <- createDataPartition(y, p = 0.7, list = FALSE, times = 1)
X_train <- X[trainIndex, ]
X_test <- X[-trainIndex, ]
y_train <- y[trainIndex]
y_test <- y[-trainIndex]


d_tree <- rpart(as.factor(y_train) ~ ., data = X_train, method = "class")


print(d_tree)
```

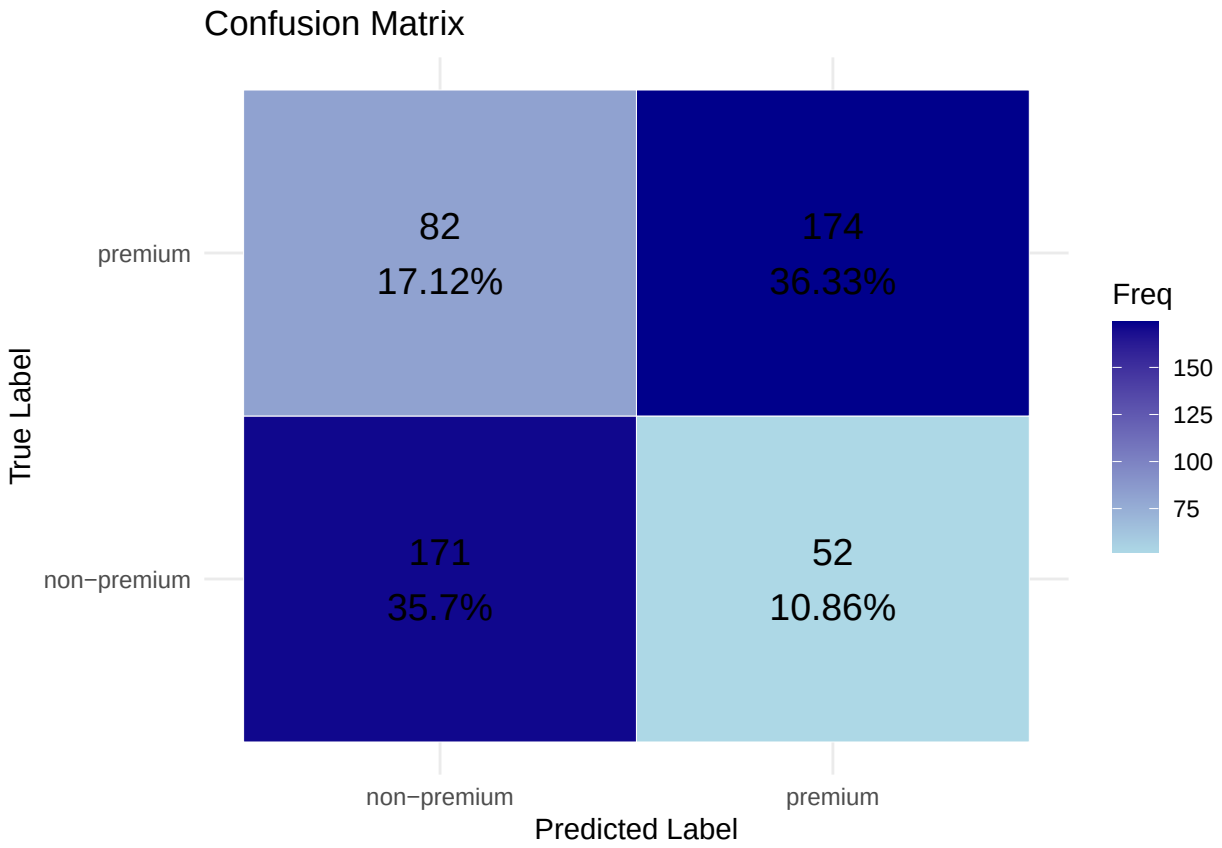
```
## n= 1120
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 1120 521 premium (0.4651786 0.5348214)
##     2) alcohol< 9.975 487 141 non-premium (0.7104723 0.2895277)
##       4) sulphates< 0.575 226  36 non-premium (0.8407080 0.1592920) *
##       5) sulphates>=0.575 261 105 non-premium (0.5977011 0.4022989)
##        10) fixed.acidity< 10.65 241  87 non-premium (0.6390041 0.3609959)
##          20) volatile.acidity>=0.275 234  80 non-premium (0.6581197 0.3418803) *
##          21) volatile.acidity< 0.275 7   0 premium (0.0000000 1.0000000) *
##        11) fixed.acidity>=10.65 20   2 premium (0.1000000 0.9000000) *
##     3) alcohol>=9.975 633 175 premium (0.2764613 0.7235387)
##       6) alcohol< 11.25 383 144 premium (0.3759791 0.6240209)
##        12) sulphates< 0.545 61  20 non-premium (0.6721311 0.3278689) *
##        13) sulphates>=0.545 322 103 premium (0.3198758 0.6801242)
##          26) total.sulfur.dioxide>=83.5 23   4 non-premium (0.8260870 0.1739130) *
##          27) total.sulfur.dioxide< 83.5 299  84 premium (0.2809365 0.7190635)
##            54) sulphates< 0.745 211  74 premium (0.3507109 0.6492891)
##             108) pH>=3.425 41  15 non-premium (0.6341463 0.3658537) *
##             109) pH< 3.425 170  48 premium (0.2823529 0.7176471) *
##            55) sulphates>=0.745 88  10 premium (0.1136364 0.8863636) *
##       7) alcohol>=11.25 250  31 premium (0.1240000 0.8760000) *
```

```
predictions <- predict(d_tree, X_test, type = "class")


conf_matrix <- confusionMatrix(predictions, factor(y_test))


print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    non-premium premium
##   non-premium         171      82
##   premium              52     174
##
##                Accuracy : 0.7203
##                  95% CI : (0.6777, 0.76)
##     No Information Rate : 0.5344
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.4427
##
##  Mcnemar's Test P-Value : 0.01224
##
##             Sensitivity : 0.7668
##             Specificity : 0.6797
##          Pos Pred Value : 0.6759
##          Neg Pred Value : 0.7699
##              Prevalence : 0.4656
##          Detection Rate : 0.3570
```

```
##    Detection Prevalence : 0.5282
##       Balanced Accuracy : 0.7233
##
##        'Positive' Class : non-premium
##
```

```
plot_confusion_matrix <- function(predictions, actual) {
  cm <- table(Prediction = predictions, Actual = actual)


  cm_perc <- prop.table(cm) * 100


  labels <- paste0(cm, "\n", round(cm_perc, 2), "%")


  cm_df <- as.data.frame(cm)
  cm_df$Perc <- as.vector(cm_perc)
  cm_df$Label <- as.vector(labels)


  ggplot(data = cm_df, aes(x = Prediction, y = Actual, fill = Freq)) +
    geom_tile(color = "white") +
    geom_text(aes(label = Label), size = 5) +
    scale_fill_gradient(low = "lightblue", high = "darkblue") +
    labs(title = "Confusion Matrix", x = "Predicted Label", y = "True Label") +
    theme_minimal()
}


plot_confusion_matrix(predictions, factor(y_test))
```




```
#This code uses hyperparameters tuning to better tune the decision tree model

str(X_train)
```

```
## 'data.frame':    1120 obs. of  11 variables:
##  $ fixed.acidity       : num  7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.8 7.5 6.7 ...
##  $ volatile.acidity    : num  0.7 0.88 0.76 0.28 0.7 0.66 0.6 0.58 0.5 0.58 ...
##  $ citric.acid         : num  0 0 0.04 0.56 0 0 0.06 0.02 0.36 0.08 ...
##  $ residual.sugar      : num  1.9 2.6 2.3 1.9 1.9 1.8 1.6 2 6.1 1.8 ...
##  $ chlorides           : num  0.076 0.098 0.092 0.075 0.076 0.075 0.069 0.073 0.071 0.097 ...
##  $ free.sulfur.dioxide : num  11 25 15 17 11 13 15 9 17 15 ...
##  $ total.sulfur.dioxide: num  34 67 54 60 34 40 59 18 102 65 ...
##  $ 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 3.36 3.35 3.28 ...
##  $ sulphates           : num  0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.57 0.8 0.54 ...
##  $ alcohol             : num  9.4 9.8 9.8 9.8 9.4 9.4 9.4 9.5 10.5 9.2 ...
```

```
str(y_train)
```

```
##  Factor w/ 2 levels "non-premium",..: 1 1 1 2 1 1 1 2 1 1 ...
```

```
set.seed(1)
trainIndex <- createDataPartition(y, p = 0.7, list = FALSE, times = 1)
X_train <- X[trainIndex, ]
X_test <- X[-trainIndex, ]
y_train <- y[trainIndex]
y_test <- y[-trainIndex]
```

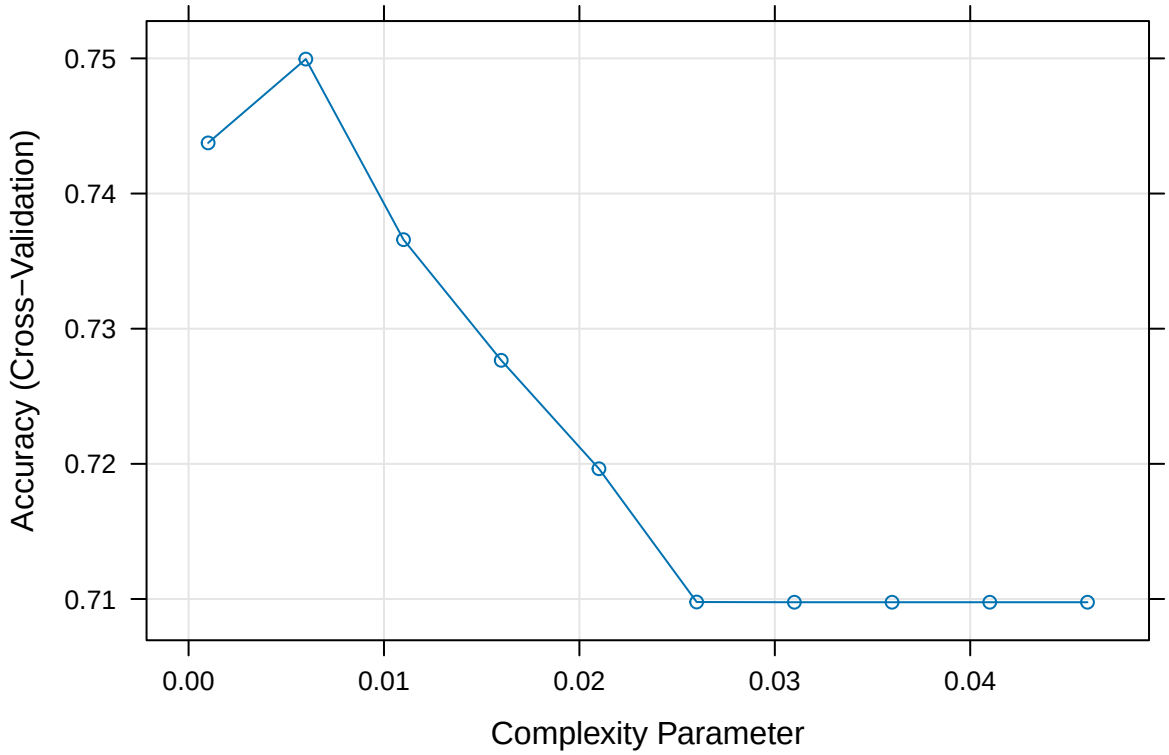
```r
train_data <- data.frame(X_train, y_train = as.factor(y_train))


train_control <- trainControl(method = "cv",
                              number = 10,
                              verboseIter = TRUE)


tune_grid <- expand.grid(
  cp = seq(0.001, 0.05, by = 0.005)
)


set.seed(1)
tuned_model <- train(
  y_train ~ .,
  data = train_data,
  method = "rpart",
  trControl = train_control,
  tuneGrid = tune_grid,
  control = rpart.control(maxdepth = 20, minsplit = 20)
)
```

```
## + Fold01: cp=0.001
## - Fold01: cp=0.001
## + Fold02: cp=0.001
## - Fold02: cp=0.001
## + Fold03: cp=0.001
## - Fold03: cp=0.001
## + Fold04: cp=0.001
## - Fold04: cp=0.001
## + Fold05: cp=0.001
## - Fold05: cp=0.001
## + Fold06: cp=0.001
## - Fold06: cp=0.001
## + Fold07: cp=0.001
## - Fold07: cp=0.001
## + Fold08: cp=0.001
## - Fold08: cp=0.001
## + Fold09: cp=0.001
## - Fold09: cp=0.001
## + Fold10: cp=0.001
## - Fold10: cp=0.001
## Aggregating results
## Selecting tuning parameters
## Fitting cp = 0.006 on full training set
```

```r
print(tuned_model$bestTune)
```

```
##      cp
## 2 0.006
```

```r
plot(tuned_model)
```

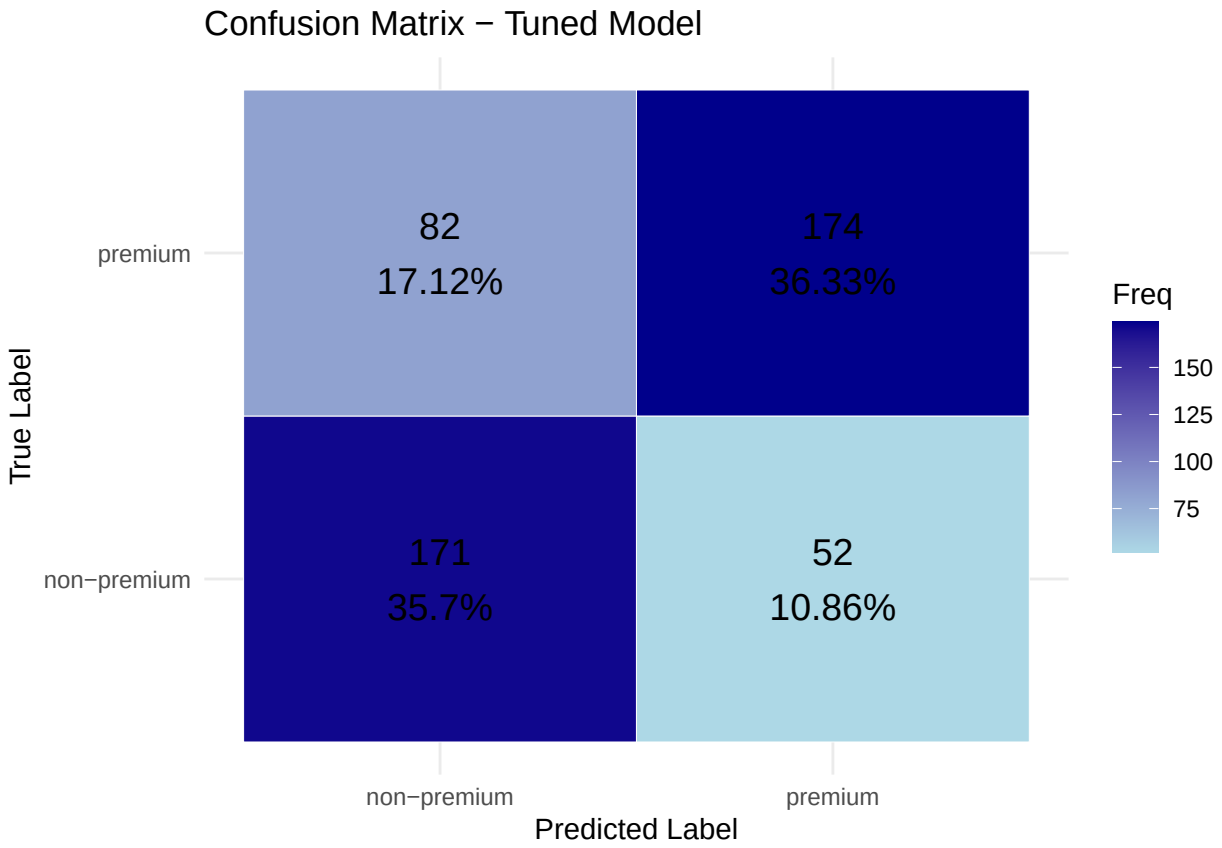



```r
best_predictions <- predict(tuned_model, newdata = X_test)


conf_matrix_tuned <- confusionMatrix(best_predictions, factor(y_test))


print(conf_matrix_tuned)
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    non-premium premium
##   non-premium         171      82
##   premium              52     174
##
##                Accuracy : 0.7203
##                  95% CI : (0.6777, 0.76)
##     No Information Rate : 0.5344
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.4427
##
##  Mcnemar's Test P-Value : 0.01224
##
##             Sensitivity : 0.7668
##             Specificity : 0.6797
##          Pos Pred Value : 0.6759
##          Neg Pred Value : 0.7699
```

```
##              Prevalence : 0.4656
##          Detection Rate : 0.3570
##    Detection Prevalence : 0.5282
##       Balanced Accuracy : 0.7233
##
##        'Positive' Class : non-premium
##
```

```
plot_confusion_matrix <- function(predictions, actual) {
  cm <- table(Prediction = predictions, Actual = actual)


  cm_perc <- prop.table(cm) * 100


  labels <- paste0(cm, "\n", round(cm_perc, 2), "%")


  cm_df <- as.data.frame(cm)
  cm_df$Perc <- as.vector(cm_perc)
  cm_df$Label <- as.vector(labels)


  ggplot(data = cm_df, aes(x = Prediction, y = Actual, fill = Freq)) +
    geom_tile(color = "white") +
    geom_text(aes(label = Label), size = 5) +
    scale_fill_gradient(low = "lightblue", high = "darkblue") +
    labs(title = "Confusion Matrix - Tuned Model", x = "Predicted Label", y = "True Label") +
    theme_minimal()
}


plot_confusion_matrix(best_predictions, factor(y_test))
```




```
#This code develops a random forest model to predict wine quality based on the
#independent variables.


if (!require(randomForest)) install.packages("randomForest", dependencies = TRUE)
```

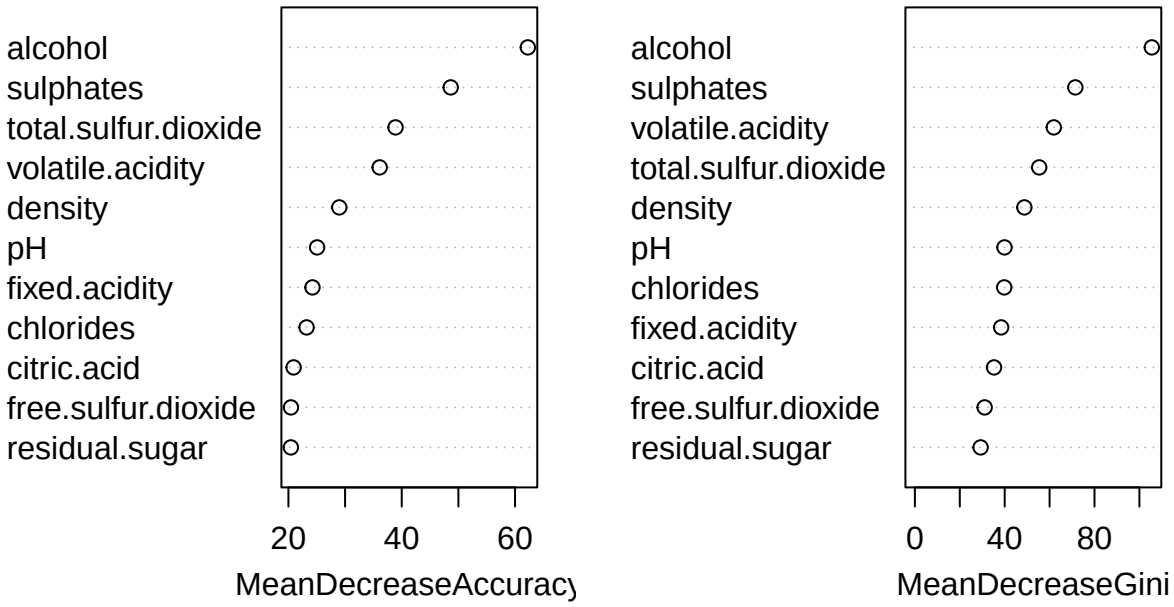
```
## Loading required package: randomForest
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
## The following object is masked from 'package:ggplot2':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

```
library(randomForest)


X <- data %>% select(-quality_class, -quality)
y <- as.factor(data$quality_class)


set.seed(1)
```

```r
trainIndex <- createDataPartition(y, p = 0.7, list = FALSE, times = 1)
X_train <- X[trainIndex, ]
X_test <- X[-trainIndex, ]
y_train <- y[trainIndex]
y_test <- y[-trainIndex]


set.seed(1)
rf_model <- randomForest(X_train, y_train,
                         ntree = 500,
                         mtry = sqrt(ncol(X_train)),
                         importance = TRUE)


print(rf_model)
```

```
##
## Call:
##  randomForest(x = X_train, y = y_train, ntree = 500, mtry = sqrt(ncol(X_train)),      importance = TR
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 20.27%
## Confusion matrix:
##             non-premium premium class.error
## non-premium         403     118   0.2264875
## premium             109     490   0.1819699
```

```r
varImpPlot(rf_model, main = "Variable Importance in Random Forest")
```

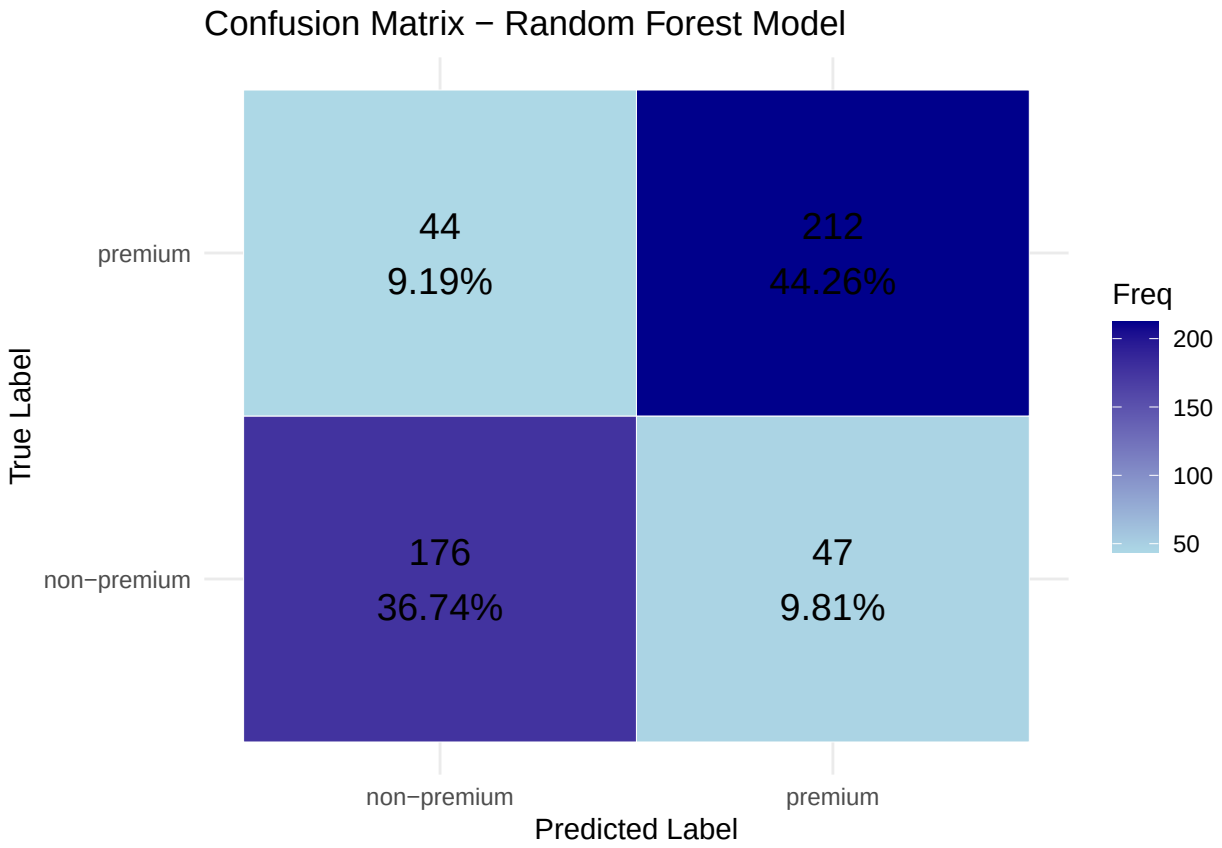



```r
rf_predictions <- predict(rf_model, newdata = X_test)


rf_conf_matrix <- confusionMatrix(rf_predictions, factor(y_test))


print(rf_conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    non-premium premium
##   non-premium         176      44
##   premium              47     212
##
##                Accuracy : 0.81
##                  95% CI : (0.772, 0.8442)
##     No Information Rate : 0.5344
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6179
##
##  Mcnemar's Test P-Value : 0.8339
##
##             Sensitivity : 0.7892
##             Specificity : 0.8281
##          Pos Pred Value : 0.8000
##          Neg Pred Value : 0.8185
##              Prevalence : 0.4656
```

```
##          Detection Rate : 0.3674
##    Detection Prevalence : 0.4593
##       Balanced Accuracy : 0.8087
##
##        'Positive' Class : non-premium
##
```

```r
plot_confusion_matrix <- function(predictions, actual) {
  cm <- table(Prediction = predictions, Actual = actual)


  cm_perc <- prop.table(cm) * 100


  labels <- paste0(cm, "\n", round(cm_perc, 2), "%")


  cm_df <- as.data.frame(cm)
  cm_df$Perc <- as.vector(cm_perc)
  cm_df$Label <- as.vector(labels)


  ggplot(data = cm_df, aes(x = Prediction, y = Actual, fill = Freq)) +
    geom_tile(color = "white") +
    geom_text(aes(label = Label), size = 5) +
    scale_fill_gradient(low = "lightblue", high = "darkblue") +
    labs(title = "Confusion Matrix - Random Forest Model", x = "Predicted Label", y = "True Label") +
    theme_minimal()
}


plot_confusion_matrix(rf_predictions, factor(y_test))
```




```r
#this code uses hypertuning parameters to better tune the previous random forest model


set.seed(1)

X <- data %>% select(-quality_class, -quality)
y <- as.factor(data$quality_class)


trainIndex <- createDataPartition(y, p = 0.7, list = FALSE, times = 1)
X_train <- X[trainIndex, ]
X_test <- X[-trainIndex, ]
y_train <- y[trainIndex]
y_test <- y[-trainIndex]


train_data <- data.frame(X_train, y_train = y_train)


train_control <- trainControl(
  method = "cv",  # Cross-validation
  number = 5,     # Number of folds
  search = "grid",  # Grid search
  verboseIter = TRUE
)
```

```r
tune_grid <- expand.grid(
  mtry = c(2, 4, 6, 8),
  splitrule = "gini",
  min.node.size = c(1, 5, 10)
)


set.seed(1)
tuned_rf_model <- train(
  y_train ~ .,
  data = train_data,
  method = "ranger",
  trControl = train_control,
  tuneGrid = tune_grid,
  importance = "impurity"
)
```

```
## + Fold1: mtry=2, splitrule=gini, min.node.size= 1
## - Fold1: mtry=2, splitrule=gini, min.node.size= 1
## + Fold1: mtry=4, splitrule=gini, min.node.size= 1
## - Fold1: mtry=4, splitrule=gini, min.node.size= 1
## + Fold1: mtry=6, splitrule=gini, min.node.size= 1
## - Fold1: mtry=6, splitrule=gini, min.node.size= 1
## + Fold1: mtry=8, splitrule=gini, min.node.size= 1
## - Fold1: mtry=8, splitrule=gini, min.node.size= 1
## + Fold1: mtry=2, splitrule=gini, min.node.size= 5
## - Fold1: mtry=2, splitrule=gini, min.node.size= 5
## + Fold1: mtry=4, splitrule=gini, min.node.size= 5
## - Fold1: mtry=4, splitrule=gini, min.node.size= 5
## + Fold1: mtry=6, splitrule=gini, min.node.size= 5
## - Fold1: mtry=6, splitrule=gini, min.node.size= 5
## + Fold1: mtry=8, splitrule=gini, min.node.size= 5
## - Fold1: mtry=8, splitrule=gini, min.node.size= 5
## + Fold1: mtry=2, splitrule=gini, min.node.size=10
## - Fold1: mtry=2, splitrule=gini, min.node.size=10
## + Fold1: mtry=4, splitrule=gini, min.node.size=10
## - Fold1: mtry=4, splitrule=gini, min.node.size=10
## + Fold1: mtry=6, splitrule=gini, min.node.size=10
## - Fold1: mtry=6, splitrule=gini, min.node.size=10
## + Fold1: mtry=8, splitrule=gini, min.node.size=10
## - Fold1: mtry=8, splitrule=gini, min.node.size=10
## + Fold2: mtry=2, splitrule=gini, min.node.size= 1
## - Fold2: mtry=2, splitrule=gini, min.node.size= 1
## + Fold2: mtry=4, splitrule=gini, min.node.size= 1
## - Fold2: mtry=4, splitrule=gini, min.node.size= 1
## + Fold2: mtry=6, splitrule=gini, min.node.size= 1
## - Fold2: mtry=6, splitrule=gini, min.node.size= 1
## + Fold2: mtry=8, splitrule=gini, min.node.size= 1
## - Fold2: mtry=8, splitrule=gini, min.node.size= 1
## + Fold2: mtry=2, splitrule=gini, min.node.size= 5
## - Fold2: mtry=2, splitrule=gini, min.node.size= 5
## + Fold2: mtry=4, splitrule=gini, min.node.size= 5
## - Fold2: mtry=4, splitrule=gini, min.node.size= 5
## + Fold2: mtry=6, splitrule=gini, min.node.size= 5
```

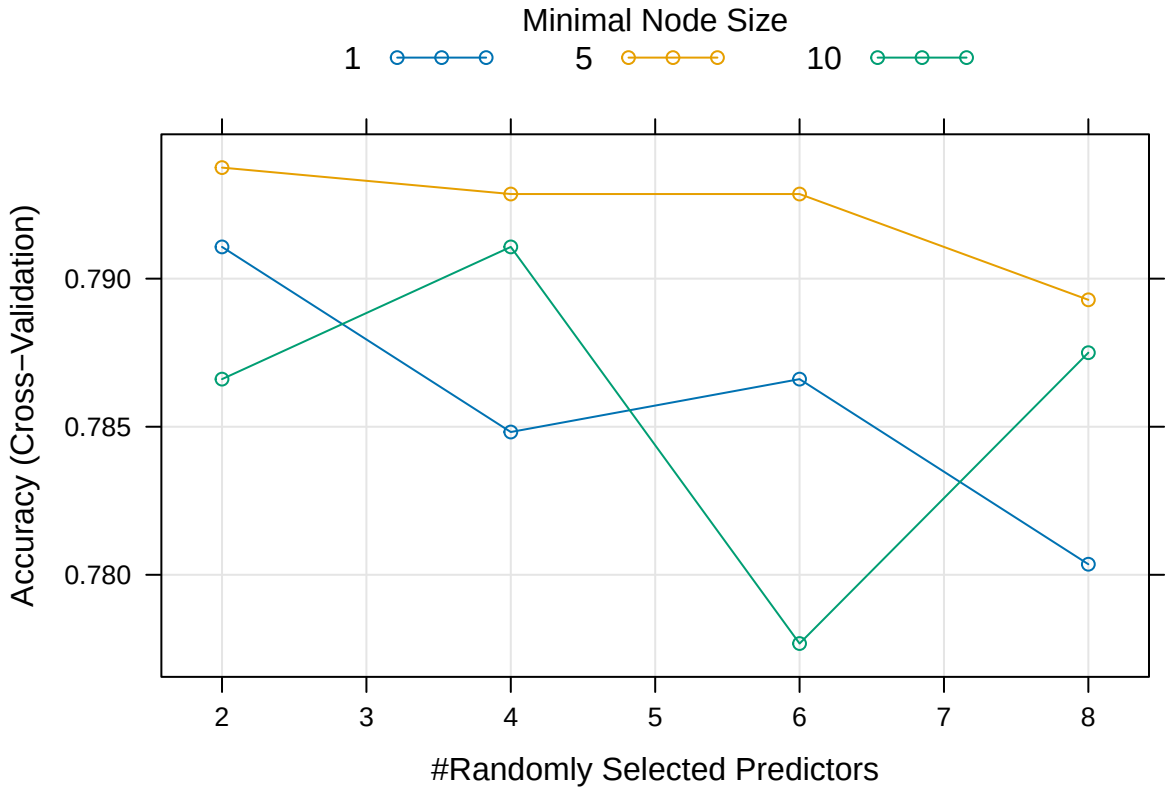
```
## - Fold2: mtry=6, splitrule=gini, min.node.size= 5
## + Fold2: mtry=8, splitrule=gini, min.node.size= 5
## - Fold2: mtry=8, splitrule=gini, min.node.size= 5
## + Fold2: mtry=2, splitrule=gini, min.node.size=10
## - Fold2: mtry=2, splitrule=gini, min.node.size=10
## + Fold2: mtry=4, splitrule=gini, min.node.size=10
## - Fold2: mtry=4, splitrule=gini, min.node.size=10
## + Fold2: mtry=6, splitrule=gini, min.node.size=10
## - Fold2: mtry=6, splitrule=gini, min.node.size=10
## + Fold2: mtry=8, splitrule=gini, min.node.size=10
## - Fold2: mtry=8, splitrule=gini, min.node.size=10
## + Fold3: mtry=2, splitrule=gini, min.node.size= 1
## - Fold3: mtry=2, splitrule=gini, min.node.size= 1
## + Fold3: mtry=4, splitrule=gini, min.node.size= 1
## - Fold3: mtry=4, splitrule=gini, min.node.size= 1
## + Fold3: mtry=6, splitrule=gini, min.node.size= 1
## - Fold3: mtry=6, splitrule=gini, min.node.size= 1
## + Fold3: mtry=8, splitrule=gini, min.node.size= 1
## - Fold3: mtry=8, splitrule=gini, min.node.size= 1
## + Fold3: mtry=2, splitrule=gini, min.node.size= 5
## - Fold3: mtry=2, splitrule=gini, min.node.size= 5
## + Fold3: mtry=4, splitrule=gini, min.node.size= 5
## - Fold3: mtry=4, splitrule=gini, min.node.size= 5
## + Fold3: mtry=6, splitrule=gini, min.node.size= 5
## - Fold3: mtry=6, splitrule=gini, min.node.size= 5
## + Fold3: mtry=8, splitrule=gini, min.node.size= 5
## - Fold3: mtry=8, splitrule=gini, min.node.size= 5
## + Fold3: mtry=2, splitrule=gini, min.node.size=10
## - Fold3: mtry=2, splitrule=gini, min.node.size=10
## + Fold3: mtry=4, splitrule=gini, min.node.size=10
## - Fold3: mtry=4, splitrule=gini, min.node.size=10
## + Fold3: mtry=6, splitrule=gini, min.node.size=10
## - Fold3: mtry=6, splitrule=gini, min.node.size=10
## + Fold3: mtry=8, splitrule=gini, min.node.size=10
## - Fold3: mtry=8, splitrule=gini, min.node.size=10
## + Fold4: mtry=2, splitrule=gini, min.node.size= 1
## - Fold4: mtry=2, splitrule=gini, min.node.size= 1
## + Fold4: mtry=4, splitrule=gini, min.node.size= 1
## - Fold4: mtry=4, splitrule=gini, min.node.size= 1
## + Fold4: mtry=6, splitrule=gini, min.node.size= 1
## - Fold4: mtry=6, splitrule=gini, min.node.size= 1
## + Fold4: mtry=8, splitrule=gini, min.node.size= 1
## - Fold4: mtry=8, splitrule=gini, min.node.size= 1
## + Fold4: mtry=2, splitrule=gini, min.node.size= 5
## - Fold4: mtry=2, splitrule=gini, min.node.size= 5
## + Fold4: mtry=4, splitrule=gini, min.node.size= 5
## - Fold4: mtry=4, splitrule=gini, min.node.size= 5
## + Fold4: mtry=6, splitrule=gini, min.node.size= 5
## - Fold4: mtry=6, splitrule=gini, min.node.size= 5
## + Fold4: mtry=8, splitrule=gini, min.node.size= 5
## - Fold4: mtry=8, splitrule=gini, min.node.size= 5
## + Fold4: mtry=2, splitrule=gini, min.node.size=10
## - Fold4: mtry=2, splitrule=gini, min.node.size=10
## + Fold4: mtry=4, splitrule=gini, min.node.size=10
```

```
## - Fold4: mtry=4, splitrule=gini, min.node.size=10
## + Fold4: mtry=6, splitrule=gini, min.node.size=10
## - Fold4: mtry=6, splitrule=gini, min.node.size=10
## + Fold4: mtry=8, splitrule=gini, min.node.size=10
## - Fold4: mtry=8, splitrule=gini, min.node.size=10
## + Fold5: mtry=2, splitrule=gini, min.node.size= 1
## - Fold5: mtry=2, splitrule=gini, min.node.size= 1
## + Fold5: mtry=4, splitrule=gini, min.node.size= 1
## - Fold5: mtry=4, splitrule=gini, min.node.size= 1
## + Fold5: mtry=6, splitrule=gini, min.node.size= 1
## - Fold5: mtry=6, splitrule=gini, min.node.size= 1
## + Fold5: mtry=8, splitrule=gini, min.node.size= 1
## - Fold5: mtry=8, splitrule=gini, min.node.size= 1
## + Fold5: mtry=2, splitrule=gini, min.node.size= 5
## - Fold5: mtry=2, splitrule=gini, min.node.size= 5
## + Fold5: mtry=4, splitrule=gini, min.node.size= 5
## - Fold5: mtry=4, splitrule=gini, min.node.size= 5
## + Fold5: mtry=6, splitrule=gini, min.node.size= 5
## - Fold5: mtry=6, splitrule=gini, min.node.size= 5
## + Fold5: mtry=8, splitrule=gini, min.node.size= 5
## - Fold5: mtry=8, splitrule=gini, min.node.size= 5
## + Fold5: mtry=2, splitrule=gini, min.node.size=10
## - Fold5: mtry=2, splitrule=gini, min.node.size=10
## + Fold5: mtry=4, splitrule=gini, min.node.size=10
## - Fold5: mtry=4, splitrule=gini, min.node.size=10
## + Fold5: mtry=6, splitrule=gini, min.node.size=10
## - Fold5: mtry=6, splitrule=gini, min.node.size=10
## + Fold5: mtry=8, splitrule=gini, min.node.size=10
## - Fold5: mtry=8, splitrule=gini, min.node.size=10
## Aggregating results
## Selecting tuning parameters
## Fitting mtry = 2, splitrule = gini, min.node.size = 5 on full training set
```

```
print(tuned_rf_model$bestTune)
```

```
##   mtry splitrule min.node.size
## 2    2      gini             5
```

```
plot(tuned_rf_model)
```

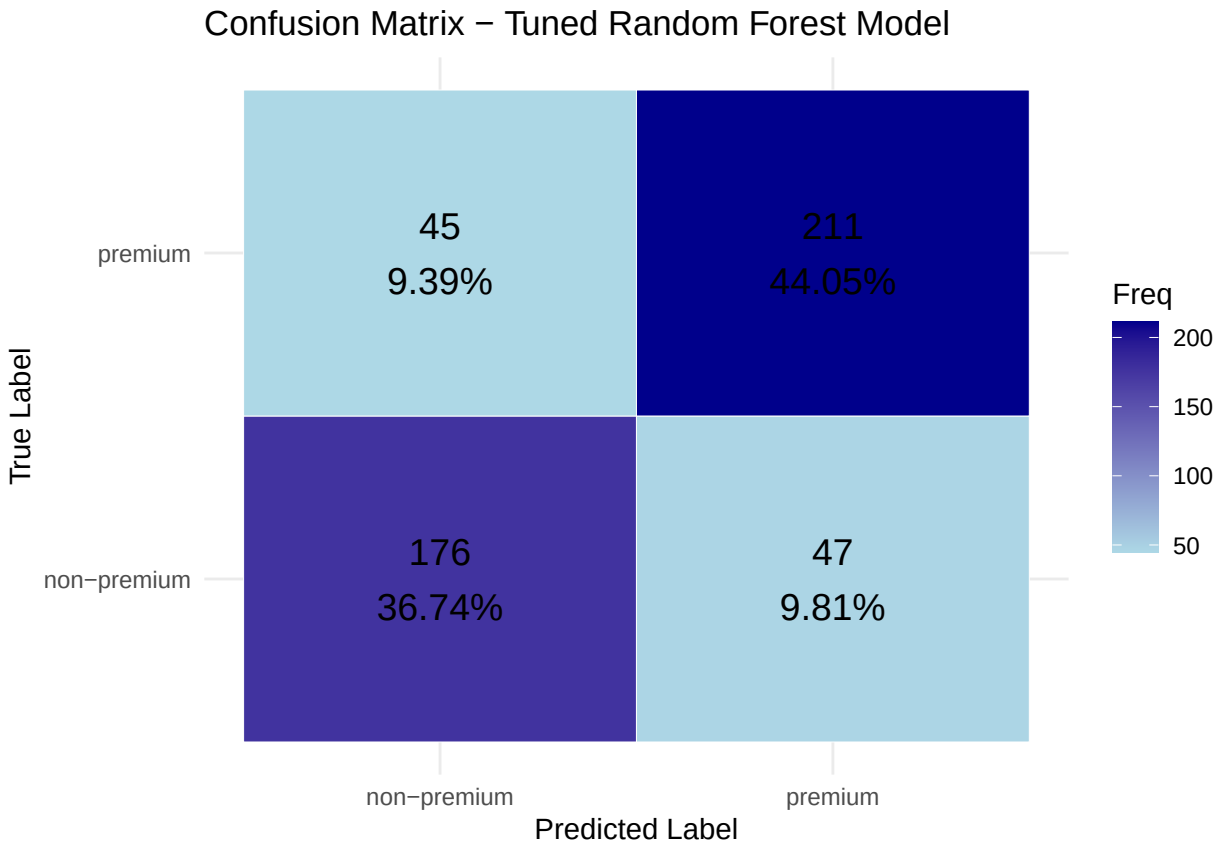



```
best_rf_predictions <- predict(tuned_rf_model, newdata = X_test)


rf_conf_matrix_tuned <- confusionMatrix(best_rf_predictions, factor(y_test))


print(rf_conf_matrix_tuned)
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    non-premium premium
##   non-premium         176      45
##   premium              47     211
##
##                Accuracy : 0.8079
##                  95% CI : (0.7698, 0.8423)
##     No Information Rate : 0.5344
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6138
##
##  Mcnemar's Test P-Value : 0.917
##
##             Sensitivity : 0.7892
##             Specificity : 0.8242
##          Pos Pred Value : 0.7964
##          Neg Pred Value : 0.8178
##              Prevalence : 0.4656
```

```
##          Detection Rate : 0.3674
##    Detection Prevalence : 0.4614
##       Balanced Accuracy : 0.8067
##
##        'Positive' Class : non-premium
##
```

```r
plot_confusion_matrix <- function(predictions, actual) {
  cm <- table(Prediction = predictions, Actual = actual)


  cm_perc <- prop.table(cm) * 100


  labels <- paste0(cm, "\n", round(cm_perc, 2), "%")


  cm_df <- as.data.frame(cm)
  cm_df$Perc <- as.vector(cm_perc)
  cm_df$Label <- as.vector(labels)


  ggplot(data = cm_df, aes(x = Prediction, y = Actual, fill = Freq)) +
    geom_tile(color = "white") +
    geom_text(aes(label = Label), size = 5) +
    scale_fill_gradient(low = "lightblue", high = "darkblue") +
    labs(title = "Confusion Matrix - Tuned Random Forest Model", x = "Predicted Label", y = "True Label
    theme_minimal()
}


plot_confusion_matrix(best_rf_predictions, factor(y_test))
```




```r
#This code compares the performance metrics of the 4 models developed:
#decision tree, tuned decision tree, random forest, and tuned random forest.


get_metrics <- function(conf_matrix) {

  accuracy <- conf_matrix$overall["Accuracy"]


  precision <- tryCatch(conf_matrix$byClass["Pos Pred Value"], error = function(e) NA)
  recall <- tryCatch(conf_matrix$byClass["Sensitivity"], error = function(e) NA)
  f1 <- tryCatch(conf_matrix$byClass["F1"], error = function(e) NA)


  return(c(Accuracy = as.numeric(accuracy),
           Precision = as.numeric(precision),
           Recall = as.numeric(recall),
           F1_Score = as.numeric(f1)))
}


dt_metrics <- get_metrics(conf_matrix)


tuned_dt_metrics <- get_metrics(conf_matrix_tuned)
```

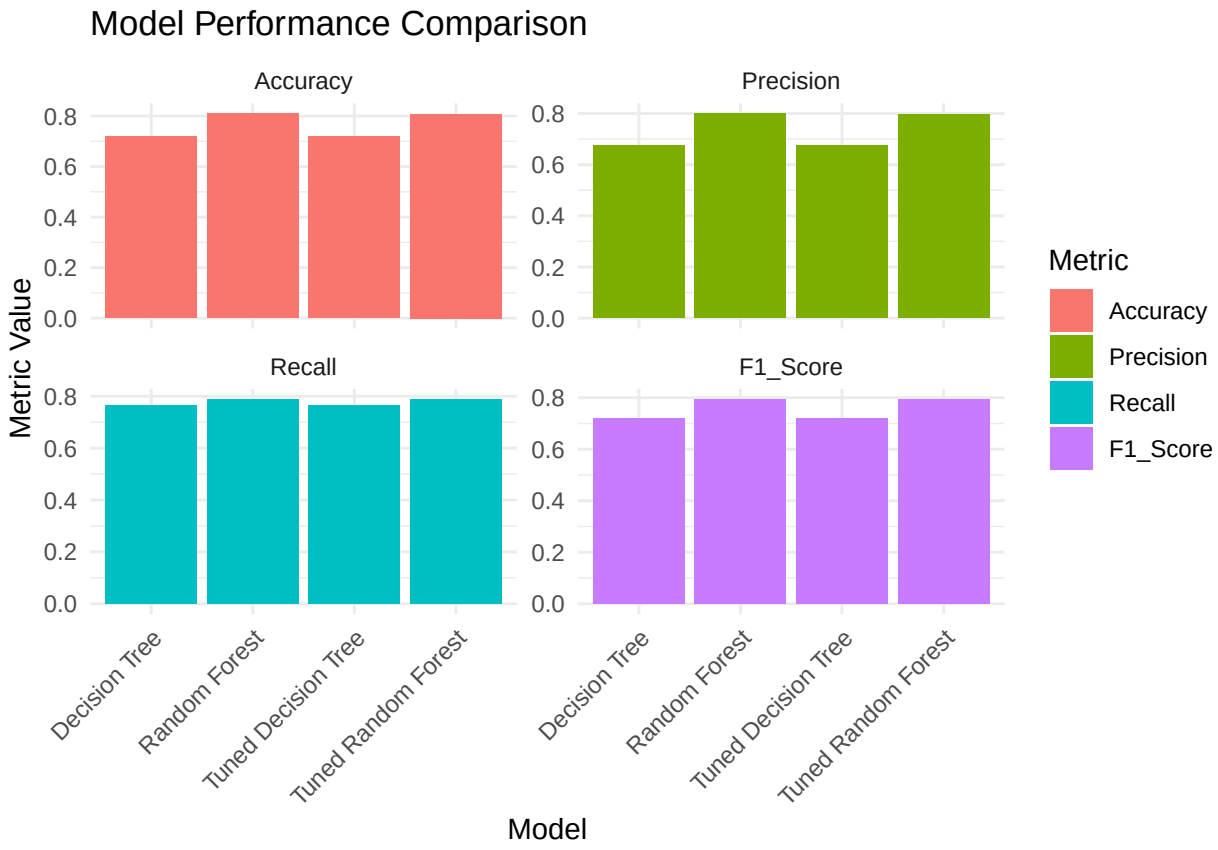
```r
rf_metrics <- get_metrics(rf_conf_matrix)


tuned_rf_metrics <- get_metrics(rf_conf_matrix_tuned)


comparison_df <- data.frame(
  Model = c("Decision Tree", "Tuned Decision Tree", "Random Forest", "Tuned Random Forest"),
  Accuracy = c(dt_metrics["Accuracy"], tuned_dt_metrics["Accuracy"], rf_metrics["Accuracy"], tuned_rf_m
  Precision = c(dt_metrics["Precision"], tuned_dt_metrics["Precision"], rf_metrics["Precision"], tuned_
  Recall = c(dt_metrics["Recall"], tuned_dt_metrics["Recall"], rf_metrics["Recall"], tuned_rf_metrics["
  F1_Score = c(dt_metrics["F1_Score"], tuned_dt_metrics["F1_Score"], rf_metrics["F1_Score"], tuned_rf_m
)


print(comparison_df)
```

```
##                 Model  Accuracy Precision    Recall  F1_Score
## 1       Decision Tree 0.7202505 0.6758893 0.7668161 0.7184874
## 2 Tuned Decision Tree 0.7202505 0.6758893 0.7668161 0.7184874
## 3       Random Forest 0.8100209 0.8000000 0.7892377 0.7945824
## 4 Tuned Random Forest 0.8079332 0.7963801 0.7892377 0.7927928
```

```r
library(ggplot2)
library(reshape2)
```

```
##
## Attaching package: 'reshape2'
##
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```r
comparison_melted <- melt(comparison_df, id.vars = "Model")


ggplot(comparison_melted, aes(x = Model, y = value, fill = variable)) +
  geom_bar(stat = "identity", position = position_dodge()) +
  facet_wrap(~ variable, scales = "free_y") +
  labs(title = "Model Performance Comparison",
       x = "Model",
       y = "Metric Value",
       fill = "Metric") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```


Model Performance Comparison
Accuracy
Precision
Recall
F1_Score
Metric Value
0.0
0.2
0.4
0.6
0.8
Decision Tree
Random Forest
Tuned Decision Tree
Tuned Random Forest
Model
Metric
Accuracy
Precision
Recall
F1_Score