Red Wine Capstone Project - A Classification Model

Harvard EdX Data Science with R Professional Certificate Course

Albertus Erwin Susanto 阎余文 (IDN)

2024-12-13

Contents

INTRODUCTION

Intro of the Data Set:

The data set is related to red variants of the Portuguese "Vinho Verde" wine. It is downloaded from Kaggle.com: https://www.kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009. For more details, consult the reference [P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009]. Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (there is no data about grape types, wine brand, wine selling price, etc.).

Intro to Our Model:

We are building a classification model using the data set. A classification model is a type of machine learning (ML) model that predicts discrete outcomes. It is used when the response variable (y) is categorical, meaning it consists of distinct classes or labels, such as "Yes" or "No," or multiple categories like "Low," "Medium," and "High."

This classification approach complements the prediction model developed in the previous capstone project of the Harvard EdX Data Science Professional Certificate Course — the Movielens Project (Movie Recommendation System). That project focused on a prediction model dealing with continuous variable data as the response (y), such as predicting movie ratings. In contrast, the current classification model targets categorical outcomes, enabling distinct decision-making applications.

The goodness or fit of a classification model is evaluated using various metrics, depending on the nature of the data set and the objectives of the analysis. Key performance metrics commonly used:

- **Confusion Matrix**: A tabular representation of actual versus predicted classifications. It includes the following components:
  - **True Positives (TP)**: Correctly predicted positive instances.
  - **True Negatives (TN)**: Correctly predicted negative instances.
  - **False Positives (FP)**: Negative instances incorrectly classified as positive.
  - **False Negatives (FN)**: Positive instances incorrectly classified as negative. The confusion matrix serves as the foundation for deriving other metrics.
- **Accuracy**: The proportion of correctly classified instances out of the total number of instances. While accuracy is straightforward, it may not reflect true model performance for imbalanced data sets.
- **Precision**: The proportion of true positive predictions among all predicted positives. Precision is crucial when false positives are costly.
- **Recall (Sensitivity)**: The proportion of true positives among all actual positives. This metric is critical in scenarios where false negatives have significant consequences.

- **F1 Score**: The harmonic mean of precision and recall, balancing these two metrics into a single value. It is particularly useful for imbalanced datasets.
- **ROC-AUC (Receiver Operating Characteristic - Area Under the Curve)**: AUC measures the model's ability to distinguish between classes across different thresholds. A higher AUC indicates better model performance and is particularly useful for evaluating classifiers on imbalanced data sets.
- **Logarithmic Loss (Log Loss)**: This metric captures the uncertainty of predictions by penalizing incorrect classifications with probabilities closer to 0 or 1. Lower log loss values indicate better performance.

In this project, we will use **AUC (Area Under the Curve)** as the primary metric to compare models. AUC provides a robust and intuitive measure of model performance by assessing the trade-off between true positive and false positive rates. It offers a comprehensive view of the model's ability to distinguish between classes, making it particularly effective for imbalanced data sets.

Intro to Our Model Building-Evaluation Process

For easier navigation throughout this report on the process of building our models and evaluating them, let me explain the flow of our work.

First, we begin by preparing the data and libraries essential for building a robust classification model. This foundational step ensures the data set is clean and ready for analysis. To do that we loaded the dataset and conducted a thorough pre-processing phase, including handling missing values, removing duplicates, and inspecting the data's structure, i.e. the variables or the features.

Following the initial preparation, we performed a detailed exploratory data analysis (EDA) to understand the underlying patterns and characteristics of the data. Visualizations and summary statistics were instrumental in identifying skewed distributions and potential outliers. Beside eliminating outliers caused most-likely by error in data inputting, we also find the skewness nature of the data. Recognizing this, we applied transformations, such as Box-Cox and Yeo-Johnson, to normalize the data and address skewness. These transformations tailored the data set for better compatibility with machine learning algorithms.

Next, in the Chapter 4, to create a binary classification task, we categorized wine quality into two distinct classes: high-quality and low-quality. Addressing class imbalance was critical at this stage, as the data set exhibited unequal class distributions of the wine quality. We implemented techniques like ROSE (Random Over-Sampling Examples) to balance the data and improve model performance. But note also that we will keep the imbalanced data for the next processes and compare the models built using the original imbalanced data and the balanced data, to see which one works better.

Still in the Chapter 4, for feature selection, we employed both statistical approach, such as the Chi-Square Test, and machine learning technique, Random Forest importance ranking, to identify the most relevant predictors for the model. We use both as a combination to decide which features are significant and have less correlation with other features (to prevent multicolinearity).

With the features and response variable finalized, we moved to model training. We experimented with various algorithms, including logistic regression, KNN, SVM, random forest, and gradient boosting, to

ensure a comprehensive evaluation of different approaches. Each model was trained and evaluated on both imbalanced and balanced data sets, allowing us to understand the impact of class distribution on their performance.

For evaluation, at Chapter 5, we focused on a range of metrics, including sensitivity, specificity, precision, recall, ROC, AUC, and F1 score, to provide a holistic view of model performance. These metrics not only measured the accuracy of the predictions but also highlighted the trade-offs between false positives and false negatives, which are critical in classification tasks. Among all the metrics, we will use AUC to compare all our models and see the best threshold of the models.

Lastly, we will discuss our findings in the conclusion part. I hope you will enjoy this report. I am still learning anyway so I assume I will still make mistakes. When you notice some significant misunderstanding, I would love and appreciate your inputs and feedback. You may email me at: rafaelerwin1412@gmail.com.

Disclaimer: I use GenerativeAI - ChatGPT to help me troubleshoot as well as correct my codes and spellings. However the whole process and logic is still an original work of mine. I would also thanks Prof. Rafael in the course, all the EdX team behind, (http://rafalab.dfci.harvard.edu/dsbook/), Kaggle community, and the fantastic explanation of Josh Starmer in StatQuest Youtube channel.

Let's dive in!

1. DATA AND LIBRARY PREPARATION

```r
# Suppress all warnings globally
options(warn = -1)
```

1.1. Libraries

If you have not installed the libraries below in your system, please have them install first before trying running my codes.

```r
# Load necessary libraries
suppressPackageStartupMessages(library(ggplot2))
suppressPackageStartupMessages(library(caret))
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(randomForest))
suppressPackageStartupMessages(library(gbm))
suppressPackageStartupMessages(library(class))
suppressPackageStartupMessages(library(e1071))
suppressPackageStartupMessages(library(pROC))
suppressPackageStartupMessages(library(gridExtra))
suppressPackageStartupMessages(library(patchwork))
suppressPackageStartupMessages(library(kableExtra))
suppressPackageStartupMessages(library(bestNormalize))
suppressPackageStartupMessages(library(ROSE))
suppressPackageStartupMessages(library(flextable))
suppressPackageStartupMessages(library(tibble))
suppressPackageStartupMessages(library(tidyr))
suppressPackageStartupMessages(library(stringr))
```

1.2. Data Set

Please download the data set from : "https://raw.githubusercontent.com/aerwins-yyw/harvarddswithr_capstone2/4d56be70677d40c5e63562e34e4382c1ab9db03f/winequality-red.csv" and then change the path to the file downloaded in your local system.

```r
path <- "C:/Users/HP/Downloads/Harvard DS/Red Wine/winequality-red.csv"

# Read the CSV file into a data frame
df <- read.csv(path, sep = ",")
```

```r
# Convert all columns to numeric
df_num <- data.frame(lapply(df, as.numeric))

# View the numeric data frame
# Format the table
df_num %>%
  head(n = 10) %>% # Take the first few rows
  kable(
    format = "latex", # Use "latex" for PDF output
    booktabs = TRUE,  # Use booktabs for better LaTeX formatting
    caption = "Data Set Variables/Features",
    align = "c" # Center align columns for better readability
  ) %>%
  kable_styling(
    latex_options = c("striped", "hold_position"), # Add striped rows and prevent table from floating
    font_size = 4
  ) %>%
  scroll_box(width = "100%", height = "300px") # Add scrolling box (if needed in HTML)
```

Table 1: Data Set Variables/Features

fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides	free.sulfur.dioxide	total.sulfur.dioxide	density	pH	sulphates	alcohol	quality
7.4	0.70	0.00	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5
7.8	0.88	0.00	2.6	0.098	25	67	0.9968	3.20	0.68	9.8	5
7.8	0.76	0.04	2.3	0.092	15	54	0.9970	3.26	0.65	9.8	5
11.2	0.28	0.56	1.9	0.075	17	60	0.9980	3.16	0.58	9.8	6
7.4	0.70	0.00	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5
7.4	0.66	0.00	1.8	0.075	13	40	0.9978	3.51	0.56	9.4	5
7.9	0.60	0.06	1.6	0.069	15	59	0.9964	3.30	0.46	9.4	5
7.3	0.65	0.00	1.2	0.065	15	21	0.9946	3.39	0.47	10.0	7
7.8	0.58	0.02	2.0	0.073	9	18	0.9968	3.36	0.57	9.5	7
7.5	0.50	0.36	6.1	0.071	17	102	0.9978	3.35	0.80	10.5	5

2. DATA EXPLORATION

We shall first check the data set, to get the sense of what it contains, i.e. the variables or the features.

```
# Checking the dimensions of the data Set
dim(df_num)
```

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

```
# Generate summary statistics
summary_stats <- as.data.frame(summary(df_num))

# Reformating the summary stats
summary_result <- summary_stats %>%
  select(-Var1) %>%                              # Remove Var1
  mutate(
    Statistic = str_extract(Freq, "^[^:]+"),  # Extract the statistic (e.g., Min, 1st Qu.)
    Freq = str_remove(Freq, "^[^:]+:\\s*")    # Remove text before the colon in Freq
  ) %>%
  pivot_wider(
    names_from = Var2,                           # Use Var2 as column names
    values_from = Freq                           # Use cleaned Freq as the cell values
  ) %>%
  select(Statistic, everything())              # Move Statistic as the first column

# Showing the summary stats
summary_result %>%
  kable(
    format = "latex",       # Use "latex" for PDF output
    booktabs = TRUE,        # Enhance formatting with booktabs
    caption = "Summary Statistics of Data Set", # Add table caption
    align = "c"             # Center-align columns
  ) %>%
  kable_styling(
    latex_options = c("striped", "hold_position", "H"), # Prevent floating and add styling
    font_size = 4                                       # Adjust font size for readability
  )
```

[!h]

Table 2: Summary Statistics of Data Set

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

We have 12 columns in the data, here are what we know about them:

(1) fixed.acidity: most acids involved with wine or fixed or nonvolatile (do not evaporate readily).

(2) volatile.acidity: the amount of acetic acid in wine, which at too high of levels can lead to an unpleasant, vinegar taste.

(3) citric.acid: found in small quantities, citric acid can add 'freshness' and flavor to wines.

(4) residual.sugar: the amount of sugar remaining after fermentation stops, it's rare to find wines with less than 1 gram/liter and wines with greater than 45 grams/liter are considered sweet.

(5) chlorides: the amount of salt in the wine.

(6) free.sulfur.dioxide: the free form of SO2 exists in equilibrium between molecular SO2 (as a dissolved gas) and bisulfite ion; it prevents microbial growth and the oxidation of wine.

(7) total.sulfur.dioxide: amount of free and bound forms of S02; in low concentrations, SO2 is mostly undetectable in wine, but at free SO2 concentrations over 50 ppm, SO2 becomes evident in the nose and taste of wine.

(8) density: the density of water is close to that of water depending on the percent alcohol and sugar content.

(9) pH: describes how acidic or basic a wine is on a scale from 0 (very acidic) to 14 (very basic); most wines are between 3-4 on the pH scale.

(10) sulphates: a wine additive which can contribute to sulfur dioxide gas (S02) levels, which acts as an antimicrobial and antioxidant.

(11) alcohol: the percent alcohol content of the wine.

(12) quality (our y variable): output variable (based on sensory data, score between 0 and 10). This is done by wine connoisseurs.

```r
# Checking NAs in the data
na_data <- is.na(df_num)
na_indices <- which(na_data, arr.ind = TRUE)
df_num[na_data]
```

```
## numeric(0)
```

We find that there's no NA.

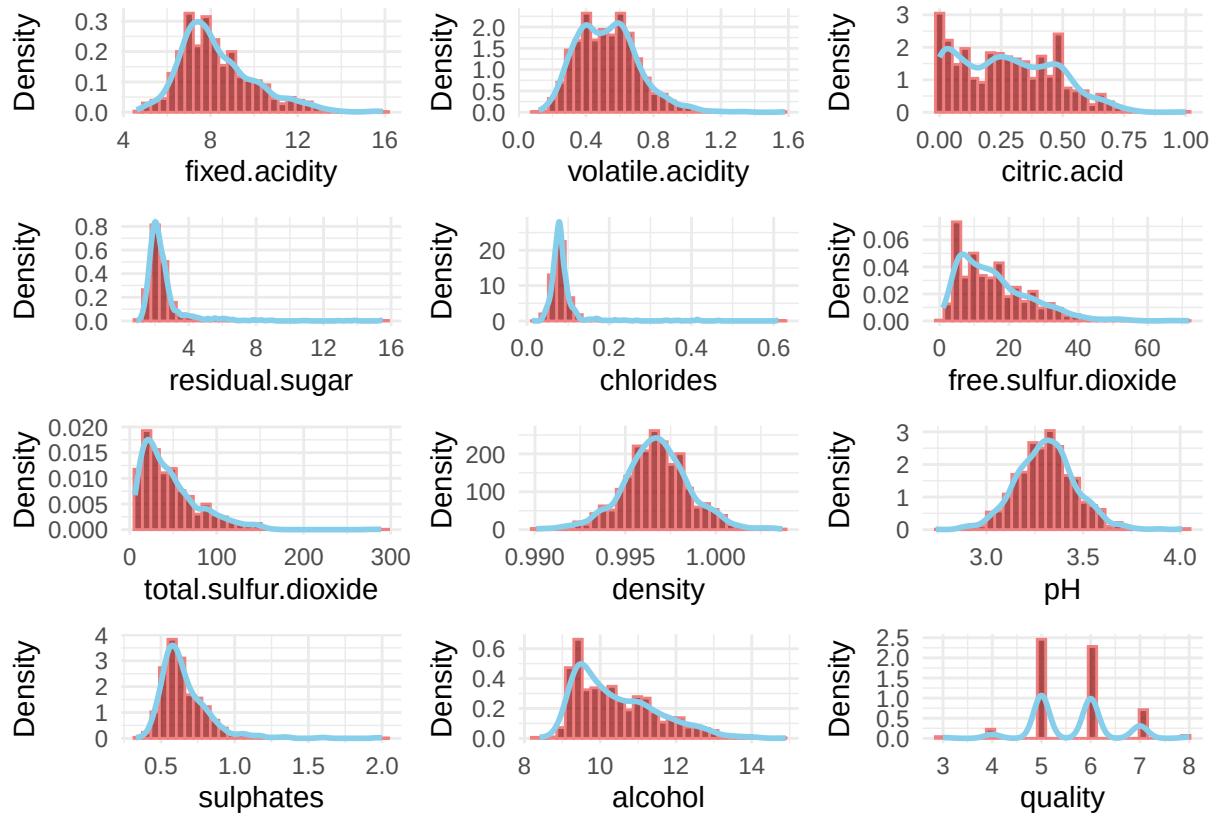
```
# Checking for duplicates
duplicates <- duplicated(df_num) # Logical vector indicating duplicate rows
sum(duplicates)                  # Counts the number of duplicate rows
```

```
## [1] 240
```

```
df_unique <- df_num[!duplicates, ] # Removes duplicate rows from the dataset
nrow(df_unique)                    # Counts the number of unique rows
```

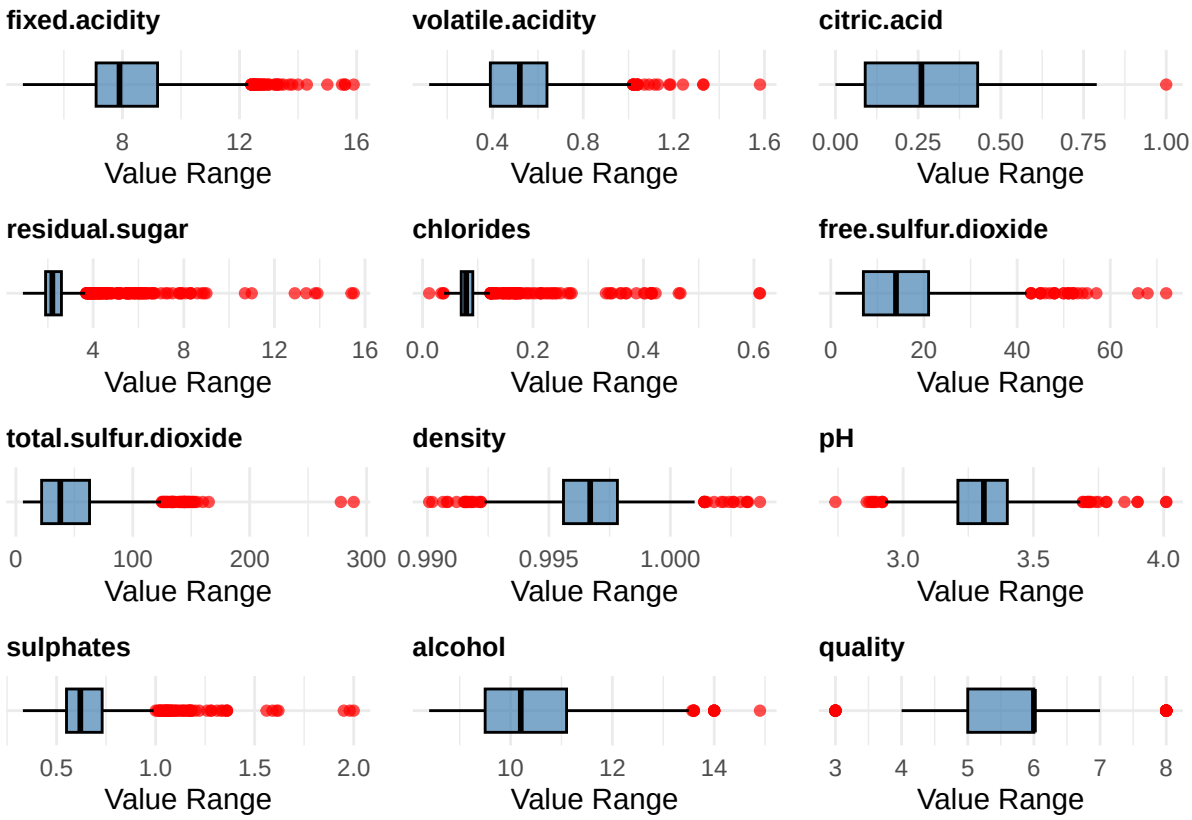
```
## [1] 1359
```

```
# Create individual plots for each column, to see their distribution
columns <- colnames(df_unique) # Get column names
plots <- lapply(columns, function(column) { # Iterate over column names
  ggplot(df_unique, aes_string(x = column)) +
    geom_histogram(color = "LightCoral", fill = "DarkRed",
                   alpha = 0.7, bins = 30, aes(y = ..density..)) +
    # Use density scaling
    geom_density(color = "SkyBlue", linewidth = 1) + # Add density line
    labs(x = column, y = "Density") +
    theme_minimal()
})

# Combine plots using patchwork
plot_grid <- wrap_plots(plots, ncol = 3)

# Adjust ncol to set the number of columns
plot_grid
```




```r
# Create individual box plots for each column
plots <- lapply(columns, function(column) {
  ggplot(df, aes(x = column), ax = axes[row, col]) +
    geom_boxplot(fill = "steelblue", color = "black", alpha = 0.7) +
    labs(title = paste("Box Plot of", column), y = "Value Range") +
    theme_minimal() +
    theme(plot.title = element_text(size = 10, face = "bold"))
})

# Create individual horizontal box plots with red outliers
plots <- lapply(columns, function(column) {
  ggplot(df_unique, aes(x = .data[[column]],
                        y = "")) + # Swap x and y for horizontal orientation
    geom_boxplot(fill = "steelblue", color = "black",
                 alpha = 0.7, outlier.color = "red") + # Customize outliers
    labs(title = column, x = "Value Range", y = NULL) +
    theme_minimal() +
    theme(plot.title = element_text(size = 10, face = "bold"))
})
```

```
# Combine plots into a grid
plot_grid <- wrap_plots(plots, ncol = 3) # Adjust ncol to set num. of columns
plot_grid
```



3. DATA WRANGGLING

3.1. Removing Outliers

Observations from the Box Plot:

- Almost all of the data technically have outliers.
- However, qualitative research reveals that most of these are legitimate data points, not true outliers.
- The "total sulfur dioxide" is flagged as an anomaly based on two considerations:

  (1) Red wines typically have a total sulfur dioxide level of 150 ppm (150 mg/litre) or less, so values above 150 may indicate input errors. See: https://grape-to-glass.com/index.php/sulphur-levels-wine/

  (2) If the value of total sulfur dioxide exceeds 150, it should also correspond to high levels of free sulfur dioxide, which will be checked below.

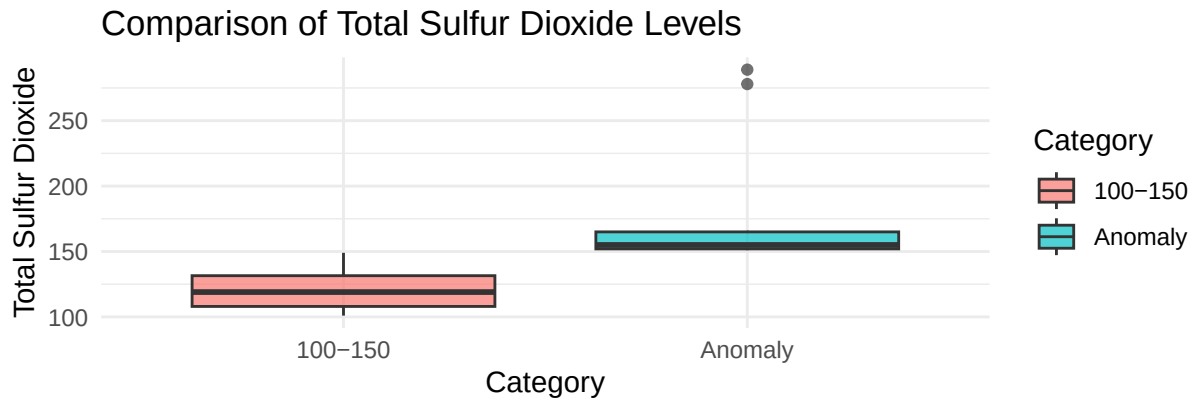
```
# Identifying anomalies in "total sulfur dioxide"
anomaly <- df_unique %>%
  # Sorts the dataset by total sulfur dioxide in descending order
  arrange(desc(total.sulfur.dioxide)) %>%
  filter(total.sulfur.dioxide > 150)

anomaly[, c("total.sulfur.dioxide", "free.sulfur.dioxide")]
```

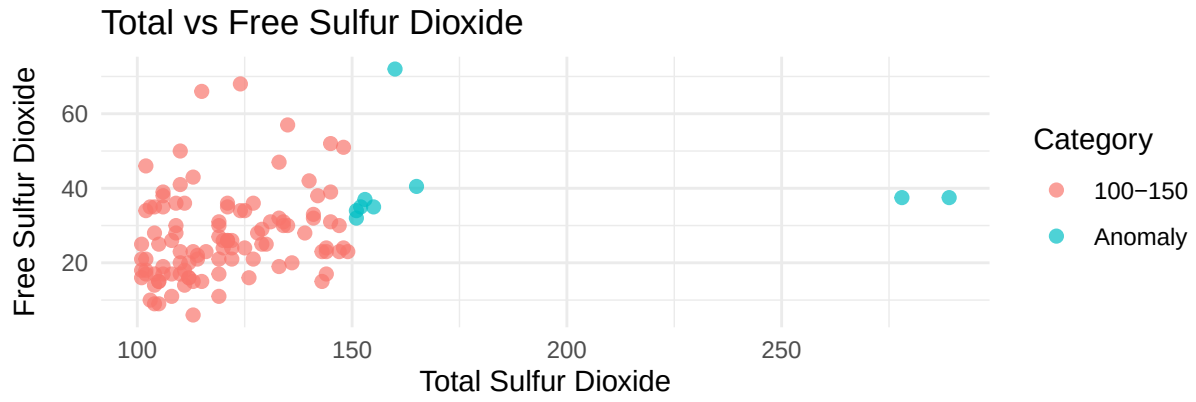
```
##   total.sulfur.dioxide free.sulfur.dioxide
## 1                  289                37.5
## 2                  278                37.5
## 3                  165                40.5
## 4                  160                72.0
## 5                  155                35.0
## 6                  153                37.0
## 7                  152                35.0
## 8                  151                34.0
## 9                  151                32.0
```

```
around100_150 <- df_unique %>%
  arrange(desc(total.sulfur.dioxide)) %>%
  filter(total.sulfur.dioxide > 100 & total.sulfur.dioxide < 150)

head(around100_150[, c("total.sulfur.dioxide", "free.sulfur.dioxide")], n=10)
```

```
##    total.sulfur.dioxide free.sulfur.dioxide
## 1                   149                  23
## 2                   148                  51
## 3                   148                  24
## 4                   147                  30
## 5                   147                  23
## 6                   145                  52
## 7                   145                  39
## 8                   145                  31
## 9                   144                  17
## 10                  144                  24
```

```r
# Combine the datasets with a category column
combined <- bind_rows(
  anomaly %>% mutate(Category = "Anomaly"),
  around100_150 %>% mutate(Category = "100-150")
)

# Create the box plot
box_plot <- ggplot(combined, aes(x = Category,
                                 y = total.sulfur.dioxide,
                                 fill = Category)) +
  geom_boxplot(alpha = 0.7) +
  labs(title = "Comparison of Total Sulfur Dioxide Levels",
       x = "Category",
       y = "Total Sulfur Dioxide") +
  theme_minimal()

# Create the scatter plot
scatter_plot <- ggplot(combined, aes(x = total.sulfur.dioxide,
                                     y = free.sulfur.dioxide,
                                     color = Category)) +
  geom_point(size = 2, alpha = 0.7) +
  labs(title = "Total vs Free Sulfur Dioxide",
       x = "Total Sulfur Dioxide",
       y = "Free Sulfur Dioxide") +
  theme_minimal()

# Combine the plots in a grid
combined_plot <- box_plot / scatter_plot

# Stacks the box plot and scatter plot vertically
combined_plot
```





The box plot reveals a clear difference between the anomaly group (total sulfur dioxide > 150) and the 100-150 group. However, the scatter plot shows that, despite the high levels of total sulfur dioxide, the anomaly group does not exhibit correspondingly high levels of free sulfur dioxide. This observation supports the suspicion that these data points may indeed be outliers caused by input errors.

Next, we aim to determine the number of these anomalies and assess whether their removal is justified, particularly if their proportion is insignificant.

```
# Count the number of anomalies
num_anomalies <- nrow(anomaly)
num_anomalies
```

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

```
# Calculate the percentage of anomalies
percentage_of_anomaly <- (num_anomalies / nrow(df_unique)) * 100
percentage_of_anomaly
```

```
## [1] 0.6622517
```

Since we find that there are only 9 anomalies, representing 0.66% of the total, we will simply drop them.

```r
# Dropping the anomalies
df_real <- df_unique[!rownames(df_unique) %in% rownames(anomaly), ]
```

3.2. Normalization

3.2.1. QQ Plot - Checking for Skewness

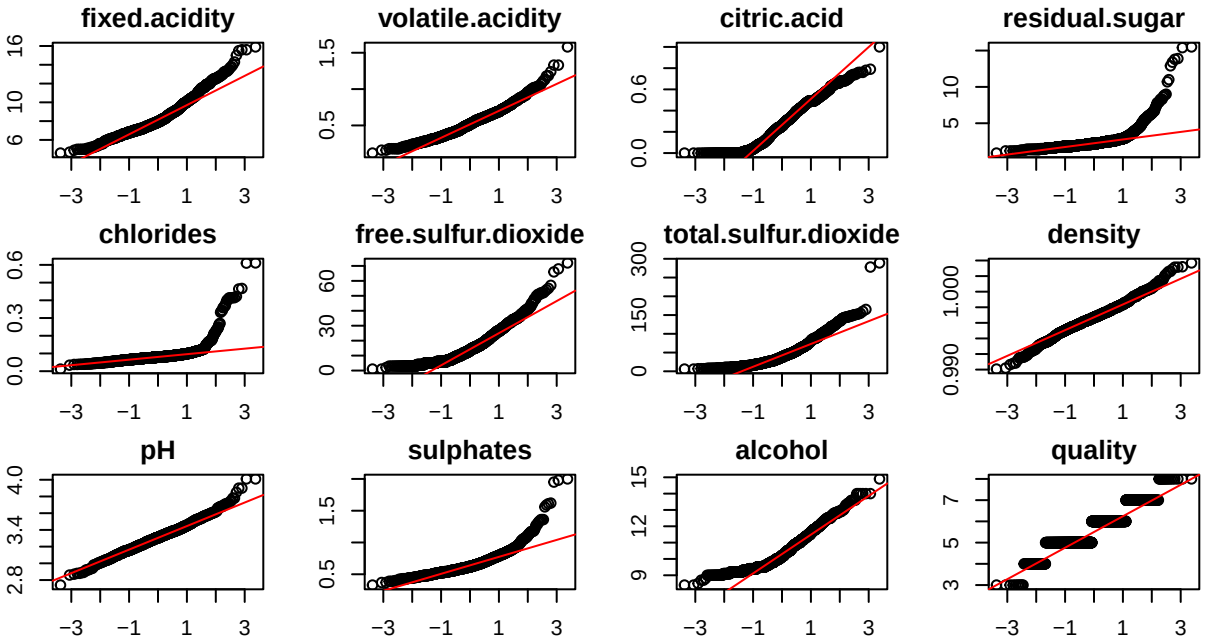
Before we normalize our data for better prediction, we need to check more in detail about the skewness of our data, all the features of wine (which are the columns) Above we have checked roughly using the density plot, we can do better however using the QQ plot.

As a note, skewness is a measure of the asymmetry of a probability distribution of a data set around its mean. It tells us whether the data is symmetrically distributed or if it is "skewed" (i.e., lopsided) in one direction.

In a perfectly symmetrical distribution, such as the normal distribution, skewness is 0. A skewed distribution can either have a positive or negative skew, depending on the direction of the tail.

Let's do that:

```r
# Calculate grid dimensions for all columns
num_cols <- ncol(df_real)
grid_dim <- ceiling(sqrt(num_cols))

# Set up the plotting grid
par(mfrow = c(grid_dim, grid_dim), mar = c(2, 2, 2, 2))  # Adjust margins

# Generate Q-Q plots for all columns
for (col in names(df_real)) {
  qqnorm(df_real[[col]], main = col)
  qqline(df_real[[col]], col = "red")
}

# Reset plotting parameters
par(mfrow = c(1, 1))
```



Conclusion from qq plots:

We learn that all our data has some degree of skewness, especially on their tails. Except of the y, which is the quality of wine that shows the step-like pattern. The step-like pattern shows that wine quality is an ordinal or discrete variable. This is not skewness but reflects the discrete nature of the data.

3.2.2. Doing the Normalization

First, what is normalization? We need first talk about "scaling and shifting" of a feature or a distribution of a data. Scaling means multiply or divide every data point in a distribution with a a constant. Shifting means add or subtract every data point in a distribution with a constant.

Normalization is a data pre-processing technique used to scale the values of numerical data into a specific range, typically 0 to 1 or -1 to 1 (if using min-max normalization). It is commonly used to ensure that all features contribute equally to the analysis or modeling process, especially when the scales of features vary widely. There are various ways to do this, one of it is mix-max normalization.

Note also about standardization. Standardization is a data pre-processing technique that transforms the values of a data set so that they have a mean of 0 and a standard deviation of 1. It adjusts the data to a standard normal distribution (also known as z-scores), making it easier to compare features with different units or scales. Standardization can be considered a type of normalization because both processes involve adjusting the data to improve comparability and suitability for analysis.

However, in data pre-processing, normalization and standardization are often treated as distinct techniques due to their specific purposes and methods.

To learn more, see this : https://www.youtube.com/watch?v=sxEqtjLC0aM. That guy explains it the best! He deserves an Oscar!

3.2.2.1. Box Cox Normalization

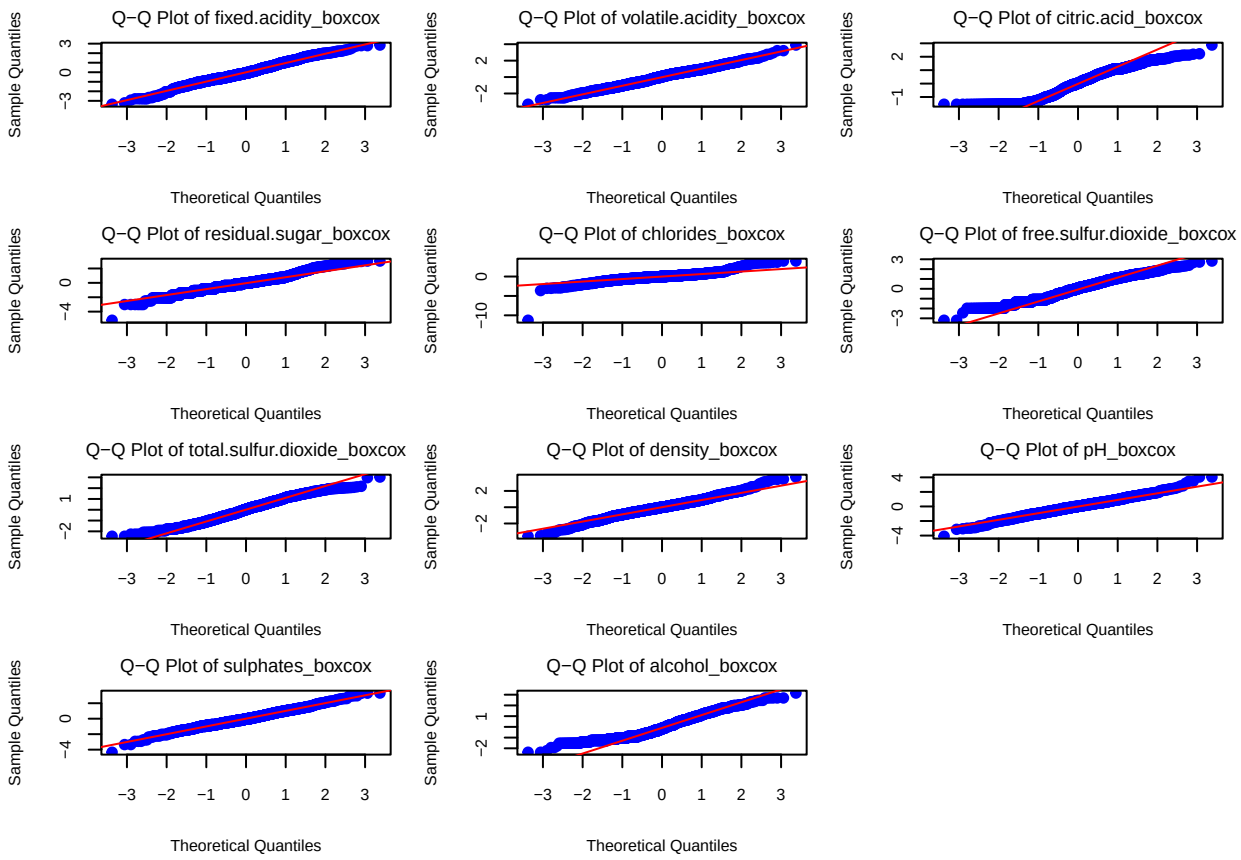
Now, for our project, we will first try using Box-Cox to normalize the data. This provides a more tailored transformation to suit the data's skewness. Log transformations, in comparison, are limited in their ability to address skewness.

```
# Define the columns to transform
skewed_columns <- columns[-12]  # Exclude the 'quality' column

# Apply Box-Cox transformation
transformed_columns <- list()

for (column in skewed_columns) {
  column_data <- df_real[[column]]

  # Check for negative or zero values and shift the data
  if (any(column_data <= 0, na.rm = TRUE)) {
    shift_value <- abs(min(column_data, na.rm = TRUE)) + 1

    # Shift to make all values positive
    column_data <- column_data + shift_value
    cat(sprintf("Column '%s' was shifted by %f to make values positive.\n",
                column, shift_value))
  }

  # Apply Box-Cox transformation
  boxcox_result <- boxcox(column_data)  # Automatically optimizes lambda
  transformed_columns[[paste0(column, "_boxcox")]] <- boxcox_result$x.t

  # Transformed data
  cat(sprintf("Optimal lambda for %s: %f\n", column, boxcox_result$lambda))
}
```

```
## Optimal lambda for fixed.acidity: -0.581839
## Optimal lambda for volatile.acidity: 0.314284
## Column 'citric.acid' was shifted by 1.000000 to make values positive.
## Optimal lambda for citric.acid: -0.281912
```

```
## Optimal lambda for residual.sugar: -0.999959
## Optimal lambda for chlorides: -0.570707
## Optimal lambda for free.sulfur.dioxide: 0.138681
## Optimal lambda for total.sulfur.dioxide: 0.048860
## Optimal lambda for density: -0.999958
## Optimal lambda for pH: -0.228259
## Optimal lambda for sulphates: -0.999958
## Optimal lambda for alcohol: -0.999958
```

```r
# Combine transformed columns into a new data frame
transformed_df <- as.data.frame(transformed_columns)

# Creating QQ Plot:
# Set up the plot layout for a 4x3 grid
grid_rows <- 4
grid_cols <- 3

# Ensure the layout fits within the knitted PDF
# Define the 4x3 grid layout
par(mfrow = c(grid_rows, grid_cols))
# Adjust margins and label sizes
par(mar = c(4, 4, 2, 1), cex.lab = 0.8, cex.axis = 0.8)

# Generate Q-Q plots for each transformed column
for (column in names(transformed_df)) {
  qqnorm(transformed_df[[column]],
         main = paste("Q-Q Plot of", column),
         col = "blue", pch = 19,
         cex.main = 0.9, font.main = 1,  # Smaller, non-bold title
         cex.lab = 0.8, cex.axis = 0.8)  # Smaller and uniform axis labels
  qqline(transformed_df[[column]], col = "red")  # Add a reference line
}

# Fill any empty slots in the grid if total columns < 12
total_plots <- length(names(transformed_df))
remaining_slots <- (grid_rows * grid_cols) - total_plots
if (remaining_slots > 0) {
  for (i in seq_len(remaining_slots)) {
    plot.new()  # Add blank plots to fill the grid
  }
}
```



3.2.2.2. Yeo-Johnson

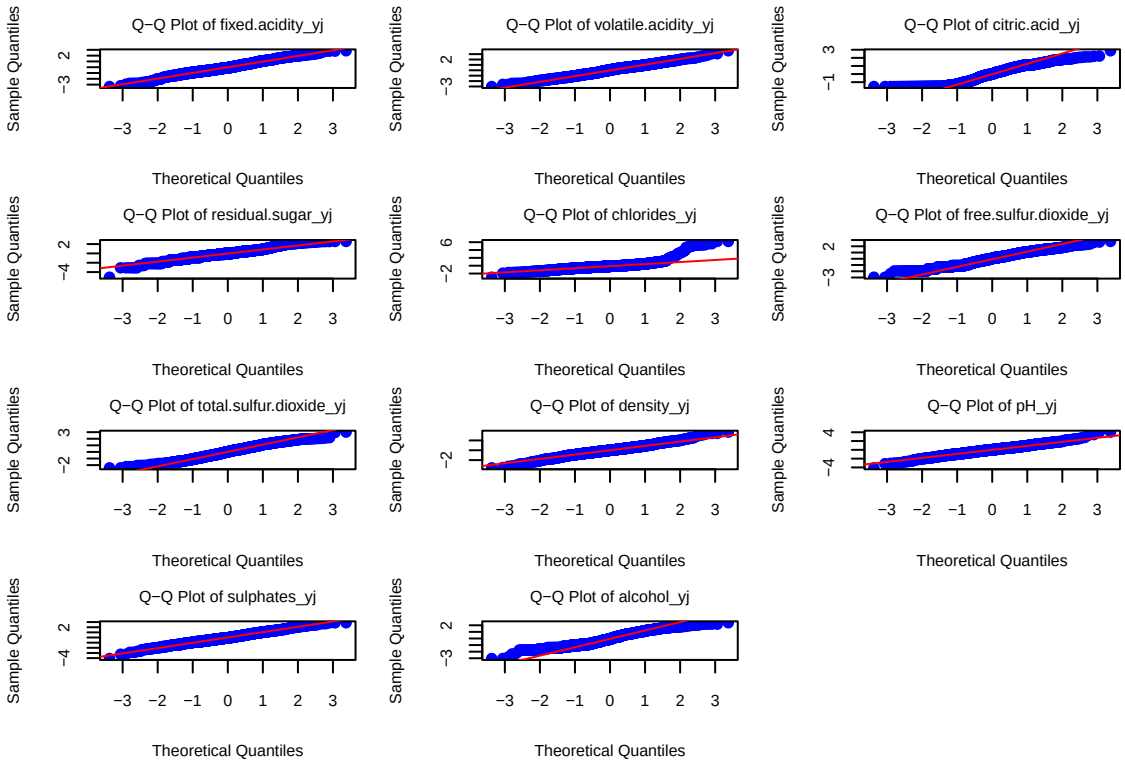
We would like to try another way of normalization: yeo-johnson.

```
# Apply Yeo-Johnson transformation
transformed_columns <- list()

for (column in skewed_columns) {
  yj_result <- yeojohnson(df_real[[column]])

  # Yeo-Johnson transformation
  transformed_columns[[paste0(column, "_yj")]] <- yj_result$x.t

  # Transformed data
  cat(sprintf("Optimal lambda for %s: %f\n", column, yj_result$lambda))
}
```

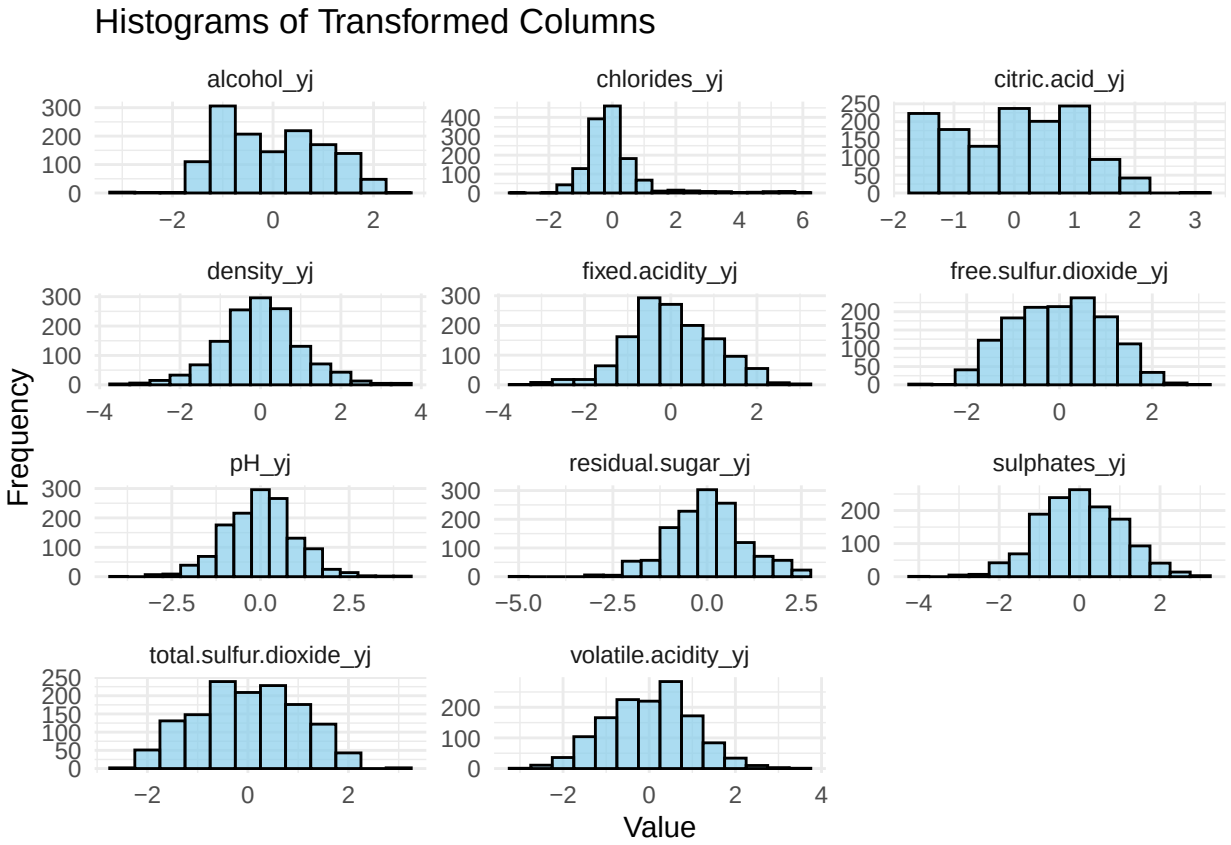
```
## Optimal lambda for fixed.acidity: -0.777568
## Optimal lambda for volatile.acidity: -0.995767
## Optimal lambda for citric.acid: -0.281918
```

```
## Optimal lambda for residual.sugar: -2.046246
## Optimal lambda for chlorides: -4.999940
## Optimal lambda for free.sulfur.dioxide: 0.065622
## Optimal lambda for total.sulfur.dioxide: 0.022355
## Optimal lambda for density: -4.999940
## Optimal lambda for pH: -0.616654
## Optimal lambda for sulphates: -4.168497
## Optimal lambda for alcohol: -3.677622
```

```r
# Combine transformed columns into a new data frame
transformed_df <- as.data.frame(transformed_columns)

# Set up the plot layout for a 4x3 grid
grid_rows <- 4
grid_cols <- 3

# Adjust the layout to fit in the knitted PDF
par(mfrow = c(grid_rows, grid_cols), oma = c(2, 2, 2, 2))  # Define the 4x3 grid layout and outer margi
par(mar = c(4, 4, 2, 1), cex.lab = 0.7, cex.axis = 0.7)  # Adjust inner margins and label sizes

# Generate Q-Q plots for each transformed column
for (column in names(transformed_df)) {
  qqnorm(transformed_df[[column]],
         main = paste("Q-Q Plot of", column),
         col = "blue", pch = 19,
         cex.main = 0.8, font.main = 1,  # Adjusted title size
         cex.lab = 0.7, cex.axis = 0.7)  # Adjusted axis labels
  qqline(transformed_df[[column]], col = "red")  # Add a reference line
}

# Fill any empty slots in the grid if total columns < 12
total_plots <- length(names(transformed_df))
remaining_slots <- (grid_rows * grid_cols) - total_plots
if (remaining_slots > 0) {
  for (i in seq_len(remaining_slots)) {
    plot.new()  # Add blank plots to fill the grid
  }
}
```




```r
# Creating the Histogram Plots
# Reshape the data for `facet_wrap()`
library(tidyr)
long_df <- pivot_longer(transformed_df, cols = everything(),
                        names_to = "Variable", values_to = "Value")


# Plot histograms using `facet_wrap()`
ggplot(long_df, aes(x = Value)) +
  geom_histogram(binwidth = 0.5, fill = "skyblue", color = "black", alpha = 0.7) +
  facet_wrap(~ Variable, scales = "free", ncol = 3) +  # 3 columns in the grid
  labs(title = "Histograms of Transformed Columns",
       x = "Value",
       y = "Frequency") +
  theme_minimal()
```



It seems that Yeo-Johnson method works better with our data set. We then use the result of this method. See that now the data are more normally distributed.

4. MODEL BUILDING

4.1. Preparing the Y (Independent Variable)

First, we change the quality into two categorical groups of 'good' and 'bad'.

```
# Re-code quality into binary classes: good (>=7) and bad (<7)
df_real$quality_binary <- ifelse(df_real$quality >= 7, "good", "bad")

# Convert to factor with correct levels ("bad" first, "good" second)
df_real$quality_binary <- factor(df_real$quality_binary,
                                 levels = c("bad", "good"))

# Add quality_binary to transformed_df
transformed_df$quality_binary <- df_real$quality_binary

# Calculate the proportion of each class in quality_binary
quality_proportions <- prop.table(table(transformed_df$quality_binary)) * 100

# Display the proportions
quality_proportions <- as.data.frame(quality_proportions)
colnames(quality_proportions) <- c("quality", "proportion")
print(quality_proportions)
```

```
##   quality proportion
## 1     bad    86.5285
## 2    good    13.4715
```

Note that:

Our transformation of the quality variable into binary variable resulted in a highly imbalanced distribution, with approximately 86.5% of the samples labeled as "bad" and only 13.5% as "good." This pronounced imbalance poses a significant challenge, as it risks biasing the model towards the majority class and underestimating the performance for the minority "good" class. Consequently, addressing this imbalance became a critical aspect of the analysis, influencing both feature selection and the evaluation process. Hence we need to address this class imbalance issue.

4.2. Balancing the Data of Good and Bad Wines

We will try using Random Over Sampling Examples (ROSE).

```r
# Apply ROSE to balance the data set
set.seed(42)
balanced_data <- ROSE(quality_binary ~ ., data = transformed_df, seed = 42)$data

# Check the class distribution
table(balanced_data$quality_binary)
```

```
##
##  bad good
##  696  655
```

Next on, we will try our models on these two data sets, to see the difference.

The first one is with the original imbalanced proportion:

```r
# Format and display the first few rows of transformed_df
transformed_df %>%
  head(n=10) %>%
  kable(
    format = "latex",                          # Use "latex" format for PDF output
    booktabs = TRUE,                           # Enhance table appearance with booktabs
    caption = "Transformed Original/Imbalanced Data", # Table caption
    align = "c"                                # Center align columns
  ) %>%
  kable_styling(
    latex_options = c("striped", "hold_position"), # Add striped rows and fix position
    font_size = 3                                  # Adjust font size
  ) %>%
  scroll_box(width = "100%", height = "300px")     # Add scrolling box for HTML output
```

Table 3: Transformed Original/Imbalanced Data

fixed.acidity_yj	volatile.acidity_yj	citric.acid_yj	residual.sugar_yj	chlorides_yj	free.sulfur.dioxide_yj	total.sulfur.dioxide_yj	density_yj	pH_yj	sulphates_yj	alcohol_yj	quality_binary
-0.3677730	-0.0472762	0.5240148	2.1308579	-0.4391737	0.3842475	1.4537010	0.5851174	0.2877887	1.0738546	0.2794524	bad
-0.9854344	0.3956238	-1.0056451	-0.8505578	0.4211497	0.1931028	0.7970529	-0.4291128	-0.1633600	-0.7903238	-1.3933704	bad
-2.0486177	0.5756231	-1.5388645	-1.4242086	0.1694766	0.2913032	0.6572827	-1.2877058	1.6910559	-1.0086812	-0.3776503	bad
-0.1620416	0.5503870	0.1821039	-1.4242086	0.9208429	-0.5551032	-0.3502931	0.3720761	-0.2944573	2.6400187	-1.5664025	bad
0.4965702	0.6007037	-0.4062537	1.4900081	2.4040969	2.2074211	1.9730112	1.0104330	-0.9651767	1.4112095	-1.3933704	bad
0.4965702	0.6007037	-0.3499106	1.5380466	2.2805557	2.1741309	2.0034122	1.0104330	-0.8969360	1.5871426	-1.3933704	bad
0.2727806	-1.5500322	1.3883384	-0.8505578	0.2651519	1.5395534	1.4680285	0.1054143	-0.0332494	0.8203048	0.2794524	good
0.0315887	0.2891485	0.1313188	-1.1212106	4.9341842	0.2913032	0.5821854	0.0520339	-1.3103994	2.3491104	-1.2280946	bad
-0.4392463	0.4478591	-1.0056451	1.7390336	0.0722067	-1.1076961	-0.3502931	0.3720761	0.4775631	-1.2424065	-1.7476372	bad
-0.0962096	-1.2396268	1.1861351	-0.8505578	4.6995322	0.3842475	0.5821854	0.1054143	-1.8052809	1.9949552	-1.3933704	bad

The second one is the balanced one:

```r
balanced_data %>%
  head(n=10) %>%
  kable(
    format = "latex",                          # Use "latex" format for PDF output
```

```
    booktabs = TRUE,                                  # Enhance table appearance with booktabs
    caption = "Tranformed Balanced Data",             # Table caption
    align = "c"                                       # Center align columns
  ) %>%
  kable_styling(
    latex_options = c("striped", "hold_position"), # Add striped rows and fix position
    font_size = 3                                     # Adjust font size
  ) %>%
  scroll_box(width = "100%", height = "300px")     # Add scrolling box for HTML output
```

Table 4: Tranformed Balanced Data

fixed.acidity_yj	volatile.acidity_yj	citric.acid_yj	residual.sugar_yj	chlorides_yj	free.sulfur.dioxide_yj	total.sulfur.dioxide_yj	density_yj	pH_yj	sulphates_yj	alcohol_yj	quality_binary
-1.2914998	-0.0138541	-0.5949956	-1.1778292	-1.5686402	0.1330573	-1.1651961	-1.1458955	0.7605485	0.0101251	0.0921346	bad
0.5442775	1.9005236	-0.0898651	0.6827814	0.6054621	0.5694271	1.3177890	1.6974542	0.7800295	-0.5444569	1.0514919	bad
-0.7347886	-2.4821757	0.7892823	-1.2271035	-0.2485892	0.8412901	0.5849248	-0.3788933	1.4675057	0.3481685	1.5925420	bad
-0.0629156	0.1031810	-0.4110307	0.4438937	-0.7280469	-0.3285817	-0.3057181	-1.7747978	0.0543080	-0.5540444	0.5935172	bad
-1.0828964	0.3065800	-1.5437643	-0.9717575	0.5734577	-0.9102884	-0.0937004	-1.8802702	0.6852853	-1.1632285	1.4263704	bad
1.6847892	-1.0030511	1.9962229	-0.2696023	0.5576228	-1.9616231	-1.1795009	-0.4022619	-0.4922148	-0.4461527	1.6165148	bad
-0.8054125	0.1085607	0.5932076	-1.1452394	0.8447619	0.6376474	-0.0381757	-0.5023505	-1.3888944	-0.2727279	-0.9380455	bad
-2.3363289	0.5184614	-1.2294894	-0.6886639	-1.2217182	1.6989294	-1.0927081	-1.1523790	1.2798660	0.9622193	0.9222291	bad
-0.1320161	0.9119884	-0.1805870	-0.2414855	-0.5882404	-0.3365783	-0.6275038	1.4247037	0.5053647	-0.8869176	-1.5627598	bad
0.1788954	-1.3751549	0.1891719	-1.1579973	0.3927802	-1.6117865	-2.0004033	0.2407293	-0.8050948	0.3464939	-1.2003746	bad

The methods build using the original proportion data set will be marked “A”, while the balanced one will be marked “B” (at the numbering).

4.3. Choosing Features

We will be using Chi-Square Test and Random Forest to choose the features

4.3.1. Chi-Square Test

We will do the chi-square test on both the imbalanced and balanced data set.

4.3.1.A. Chi-Square Test on Imbalanced Data Set

```
# Applying Chi-Square Test on Imbalanced Data Set
chi_square_results_A <- sapply(
  transformed_df[, -which(names(transformed_df) == "quality_binary")],
  function(feature) {chisq.test(table(cut(feature, breaks = 5),
                                      transformed_df$quality_binary),
                                simulate.p.value = TRUE)$p.value
  })

# Filter significant features
```

```r
significant_features_A <- names(chi_square_results_A[chi_square_results_A < 0.05])

# Print significant features line-by-line using sep
cat("Significant features (Imbalanced Dataset):\n",
    paste(significant_features_A, collapse = "\n"), "\n")
```

```
## Significant features (Imbalanced Dataset):
##  fixed.acidity_yj
## volatile.acidity_yj
## citric.acid_yj
## chlorides_yj
## free.sulfur.dioxide_yj
## total.sulfur.dioxide_yj
## density_yj
## sulphates_yj
## alcohol_yj
```

4.3.1.B. Chi-Square Test on Balanced Data Set

```r
# Applying Chi-Square Test on Balanced Data Set
chi_square_results_B <- sapply(
  balanced_data[, -which(names(balanced_data) == "quality_binary")],
  function(feature){chisq.test(table(cut(feature, breaks = 5),
                                     balanced_data$quality_binary),
                               simulate.p.value = TRUE)$p.value
  })

# Filter significant features
significant_features_B <- names(chi_square_results_B[chi_square_results_B < 0.05])

# Print significant features line-by-line using sep
cat("Significant features (Balanced Dataset):\n",
    paste(significant_features_B, collapse = "\n"), "\n")
```

```
## Significant features (Balanced Dataset):
##  fixed.acidity_yj
## volatile.acidity_yj
## citric.acid_yj
## residual.sugar_yj
## chlorides_yj
## total.sulfur.dioxide_yj
```

```
## density_yj
## sulphates_yj
## alcohol_yj
```

4.3.2. Random Forest

4.3.2.A. Random Forest on Imbalanced Data Set

```
set.seed(42)
rf_model_A <- randomForest(
  quality_binary ~ .,
  data = transformed_df,
  importance = TRUE,
  ntree = 200
)

# Get feature importance
feature_importance_A <- importance(rf_model_A, type = 1)

# Ensure feature_importance_A is a valid data frame
feature_importance_A <- data.frame(
  Feature = rownames(feature_importance_A),
  Importance = feature_importance_A[, 1],
  row.names = NULL                          # Remove row names for clarity
)

# Sort features by importance
sorted_features_A <- feature_importance_A %>%
  arrange(desc(Importance)) %>%  # Sort by descending importance
  head(10)                       # Select the top 10 features

# Ensure that the sorted_features_A table is displayed properly
sorted_features_A %>%
  kable(
    format = "latex",       # Use "latex" for PDF output
    booktabs = TRUE,        # Enhance appearance with booktabs
    caption = "Feature Importance from Random Forest on Imbalanced Data", # Table caption
    align = "c"             # Center align columns
  ) %>%
  kable_styling(
    latex_options = c("striped", "hold_position", "H"), # Fix table placement, add styling
```

```
    font_size = 7                                          # Adjust font size
  )
```

[!h]

Table 5: Feature Importance from Random Forest on Imbalanced Data

Feature	Importance
alcohol_yj	20.391727
sulphates_yj	16.880670
density_yj	10.067875
total.sulfur.dioxide_yj	9.071247
citric.acid_yj	8.981692
volatile.acidity_yj	8.531458
residual.sugar_yj	8.528662
chlorides_yj	8.025589
free.sulfur.dioxide_yj	7.927937
fixed.acidity_yj	5.686880

4.3.2.B. Random Forest on Balanced Data Set

```
set.seed(42)
rf_model_B <- randomForest(
  quality_binary ~ .,
  data = balanced_data,
  importance = TRUE,
  ntree = 200
)

# Get feature importance for Balanced Dataset
feature_importance_B <- importance(rf_model_B, type = 1)

# Feature Importance B
feature_importance_B <- data.frame(
  Feature = rownames(feature_importance_B),
  Importance = feature_importance_B[, 1],
  row.names = NULL                          # Remove row names for clarity
)

# Sort features by importance
sorted_features_B <- feature_importance_B %>%
  arrange(desc(Importance)) %>%  # Sort by descending importance
  head(10)                       # Select the top 10 features
```

```r
# Display the sorted feature importance table
sorted_features_B %>%
  kable(
    format = "latex",       # Use "latex" for PDF output
    booktabs = TRUE,        # Enhance appearance with booktabs
    caption = "Feature Importance from Random Forest on Balanced Data", # Table caption
    align = "c"             # Center align columns
  ) %>%
  kable_styling(
    latex_options = c("striped", "hold_position", "H"), # Fix table placement, add styling
    font_size = 7                                       # Adjust font size
  )
```

[!h]

Table 6: Feature Importance from Random Forest on Balanced Data

Feature	Importance
alcohol_yj	42.316418
sulphates_yj	36.376967
volatile.acidity_yj	23.445601
total.sulfur.dioxide_yj	14.031595
density_yj	13.464022
citric.acid_yj	12.685624
fixed.acidity_yj	12.483227
residual.sugar_yj	9.381066
pH_yj	7.697137
chlorides_yj	7.625345

4.3.3. Choosing the Final Feature Set

```r
# Function to finalize the feature set
finalize_features <- function(data,
                              significant_features,
                              important_features,
                              target_var = "quality_binary") {

  # Combine features from Chi-Square and Random Forest
  combined_features <- union(significant_features, important_features)

  # Ensure combined_features is not empty
  if (length(combined_features) == 0) {
    cat("No features selected.\n")
    return(data.frame())  # Return empty dataframe if no features
```

```
  }

  # Print combined features line-by-line using sep
  cat("Combined Features:\n", paste(combined_features,
                                    collapse = "\n"), "\n")

  # Subset the dataset with combined features
  combined_data <- data[, c(combined_features, target_var), drop = FALSE]

  # Perform correlation analysis only if there are multiple features
  if (ncol(combined_data) > 2) {  # More than just the target variable

    # Calculate correlation matrix
    cor_matrix <- cor(combined_data[, -which(names(combined_data) == target_var)],
                      use = "complete.obs")

    # Identify highly correlated features
    high_corr <- findCorrelation(cor_matrix, cutoff = 0.6, names = TRUE) # threshold C.A.: 0.6
    if (length(high_corr) > 0) {
      cat("Features removed due to high correlation:\n",
          high_corr, "\n")

      # Remove highly correlated features
      combined_features <- setdiff(combined_features, high_corr)

      # Subset the dataset again
      combined_data <- data[, c(combined_features, target_var), drop = FALSE]
    }
  }

  # Return the final dataset
  return(combined_data)
}
```

4.3.3.A. Final Feature Set for Imbalanced Data Set

```
# Extract the feature names from the sorted top 10 features
important_features_A <- sorted_features_A$Feature

# Finalize the feature set
final_data_A <- finalize_features(
```

```
  data = transformed_df,
  significant_features = significant_features_A,
  important_features = important_features_A
)
```

```
## Combined Features:
##  fixed.acidity_yj
## volatile.acidity_yj
## citric.acid_yj
## chlorides_yj
## free.sulfur.dioxide_yj
## total.sulfur.dioxide_yj
## density_yj
## sulphates_yj
## alcohol_yj
## residual.sugar_yj
## Features removed due to high correlation:
##  density_yj fixed.acidity_yj total.sulfur.dioxide_yj
```

4.3.3.B. Final Feature Set for Balanced Data Set

```
# Extract the top 10 feature names from sorted_features_B
important_features_B <- sorted_features_B$Feature


# Finalize the feature set
final_data_B <- finalize_features(
  data = balanced_data,
  significant_features = significant_features_B,
  important_features = important_features_B
)
```

```
## Combined Features:
##  fixed.acidity_yj
## volatile.acidity_yj
## citric.acid_yj
## residual.sugar_yj
## chlorides_yj
## total.sulfur.dioxide_yj
## density_yj
## sulphates_yj
```

```
## alcohol_yj
## pH_yj
```

```
# Inspect final datasets
cat("Final Dataset (Imbalanced):\n")
```

```
## Final Dataset (Imbalanced):
```

```
print(dim(final_data_A))
```

```
## [1] 1351    8
```

```
str(final_data_A)
```

```
## 'data.frame':    1351 obs. of  8 variables:
##  $ volatile.acidity_yj   : num  -0.0473 0.3956 0.5756 0.5504 0.6007 ...
##  $ citric.acid_yj        : num  0.524 -1.006 -1.539 0.182 -0.406 ...
##  $ chlorides_yj          : num  -0.439 0.421 0.169 0.921 2.404 ...
##  $ free.sulfur.dioxide_yj: num  0.384 0.193 0.291 -0.555 2.207 ...
##  $ sulphates_yj          : num  1.07 -0.79 -1.01 2.64 1.41 ...
##  $ alcohol_yj            : num  0.279 -1.393 -0.378 -1.566 -1.393 ...
##  $ residual.sugar_yj     : num  2.131 -0.851 -1.424 -1.424 1.49 ...
##  $ quality_binary        : Factor w/ 2 levels "bad","good": 1 1 1 1 1 1 2 1 1 1 ...
```

```
cat("Final Dataset (Balanced):\n")
```

```
## Final Dataset (Balanced):
```

```
print(dim(final_data_B))
```

```
## [1] 1351   11
```

```
str(final_data_B)
```

```
## 'data.frame':    1351 obs. of  11 variables:
##  $ fixed.acidity_yj       : num  -1.2915 0.5443 -0.7348 -0.0629 -1.0829 ...
##  $ volatile.acidity_yj    : num  -0.0139 1.9005 -2.4822 0.1032 0.3066 ...
##  $ citric.acid_yj         : num  -0.595 -0.0899 0.7893 -0.411 -1.5438 ...
##  $ residual.sugar_yj      : num  -1.178 0.683 -1.227 0.444 -0.972 ...
##  $ chlorides_yj           : num  -1.569 0.605 -0.249 -0.728 0.573 ...
##  $ total.sulfur.dioxide_yj: num  -1.1652 1.3178 0.5849 -0.3057 -0.0937 ...
##  $ density_yj             : num  -1.146 1.697 -0.379 -1.775 -1.88 ...
```

```
##  $ sulphates_yj           : num  0.0101 -0.5445 0.3482 -0.554 -1.1632 ...
##  $ alcohol_yj             : num  0.0921 1.0515 1.5925 0.5935 1.4264 ...
##  $ pH_yj                  : num  0.7605 0.78 1.4675 0.0543 0.6853 ...
##  $ quality_binary         : Factor w/ 2 levels "bad","good": 1 1 1 1 1 1 1 1 1 1 ...
```

The result shows that there are 8 features/variables retained in the imbalanced data set, while there are 11 features/variables retained in the balanced data set (all retained). The threshold of our correlation analysis is 0.6 (both positive or negative).

4.4. Model Training

4.4.1. Models: Logistic Reg., KNN, SVM, SGD, Random Forest, Grad. Boost.

```r
# Function to Train Models
train_models <- function(train_data) {

  # Define control for cross-validation with ROC
  train_control <- trainControl(
    method = "cv",                      # Cross-validation
    number = 5,                         # 5-fold cross-validation
    classProbs = TRUE,                  # Compute class probabilities
    savePredictions = "final",          # Save predictions for debugging
    summaryFunction = twoClassSummary   # Use ROC as the primary metric
  )

  # Train Models
  set.seed(42)
  models <- list(
    logistic_regression = train(
      quality_binary ~ .,
      data = train_data,
      method = "glm",
      family = "binomial",
      trControl = train_control,
      metric = "ROC"  # Optimize for ROC
    ),

    knn = train(
      quality_binary ~ .,
      data = train_data,
      method = "knn",
```

```
    tuneGrid = expand.grid(k = c(3, 5, 7, 9)),
    trControl = train_control,
    metric = "ROC"
  ),

  svm = train(
    quality_binary ~ .,
    data = train_data,
    method = "svmRadial",
    tuneGrid = expand.grid(C = c(0.1, 1, 10), sigma = c(0.01, 0.1)),
    trControl = train_control,
    metric = "ROC"
  ),

  sgd = train(
    quality_binary ~ .,
    data = train_data,
    method = "glmnet",
    trControl = train_control,
    metric = "ROC"
  ),

  random_forest = train(
    quality_binary ~ .,
    data = train_data,
    method = "rf",
    tuneGrid = expand.grid(mtry = c(2, 4, 6)),
    trControl = train_control,
    metric = "ROC"
  ),

  gradient_boosting = train(
    quality_binary ~ .,
    data = train_data,
    method = "gbm",
    tuneGrid = expand.grid(
      n.trees = c(50, 100, 200),
      interaction.depth = c(1, 3, 5),
      shrinkage = c(0.05, 0.1, 0.2),
      n.minobsinnode = c(10, 20)
    ),

    trControl = train_control,
```

```
      metric = "ROC",
      verbose = FALSE
    )
  )

  return(models)
}
```

4.4.2. Training Models From Data Set

4.4.2.A. Train Models from Imbalanced Data

```
set.seed(42)
train_index_imbalanced <- createDataPartition(final_data_A$quality_binary,
                                              p = 0.7, list = FALSE)
train_data_imbalanced <- final_data_A[train_index_imbalanced, ]
test_data_imbalanced <- final_data_A[-train_index_imbalanced, ]

models_imbalanced <- train_models(train_data_imbalanced)
```

4.4.2.B. Train Models from Balanced Data

```
set.seed(42)
train_index_balanced <- createDataPartition(final_data_B$quality_binary,
                                            p = 0.7, list = FALSE)
train_data_balanced <- final_data_B[train_index_balanced, ]
test_data_balanced <- final_data_B[-train_index_balanced, ]

models_balanced <- train_models(train_data_balanced)
```

5. MODEL EVALUATION

5.1. ROC, AUC, Sensitivity, Specificity, Precision, Accuracy

```r
# Function to Evaluate Models
evaluate_models <- function(models, test_data, dataset_name) {
  x_test <- test_data[, -which(names(test_data) == "quality_binary")]
  y_test <- factor(test_data$quality_binary, levels = c("bad", "good"))

  results <- data.frame()  # Initialize an empty data frame

  for (model_name in names(models)) {
    model <- models[[model_name]]

    # Predict probabilities for "good"
    probs <- predict(model, newdata = x_test, type = "prob")

    # Calculate ROC and AUC
    roc_curve <- roc(
      response = y_test,
      predictor = probs[, "good"],
      levels = c("bad", "good"),
      ci = TRUE  # Request confidence intervals
    )
    auc_score <- auc(roc_curve)

    # Find the Optimal Threshold
    optimal_index <- which.max(roc_curve$sensitivities +
                               roc_curve$specificities - 1)
    optimal_threshold <- roc_curve$thresholds[optimal_index]
    optimal_tpr <- roc_curve$sensitivities[optimal_index]
    # TPR at optimal threshold
    optimal_fpr <- 1 - roc_curve$specificities[optimal_index]
    # FPR at optimal threshold

    # Calculate Confusion Matrix at the Optimal Threshold
    predictions <- ifelse(probs[, "good"] > optimal_threshold, "good", "bad")
    conf_matrix <- confusionMatrix(
      factor(predictions, levels = c("good", "bad")),
      factor(y_test, levels = c("good", "bad")),
      positive = "good"
```

```
    )

    sensitivity <- conf_matrix$byClass["Sensitivity"]  # Sensitivity at optimal threshold
    specificity <- conf_matrix$byClass["Specificity"]  # Specificity at optimal threshold
    precision <- conf_matrix$byClass["Pos Pred Value"]  # Precision at optimal threshold
    accuracy <- conf_matrix$overall["Accuracy"]  # Accuracy at optimal threshold

    # Handle Confidence Intervals for AUC
    if (!is.null(roc_curve$ci)) {
      ci <- paste0(
        round(roc_curve$ci[1], 3), " - ", round(roc_curve$ci[3], 3)
      )
    } else {
      ci <- "N/A"  # Fallback if confidence intervals are not computed
    }

    # Add results to the table
    results <- rbind(
      results,
      data.frame(
        Dataset = dataset_name,
        Model = model_name,
        AUC = round(auc_score, 3),
        ROC = paste0("TPR=",
                     round(optimal_tpr, 3),
                     " | FPR=", round(optimal_fpr, 3)),
        Optimal_Threshold = round(optimal_threshold, 3),
        Sensitivity = round(sensitivity, 3),  # Adjusted to optimal threshold
        Specificity = round(specificity, 3),  # Adjusted to optimal threshold
        Precision = round(precision, 3),
        Accuracy = round(accuracy, 3),
        CI = ci,
        row.names = NULL  # Reset row names to avoid issues
      )
    )
  }

  # Sort results: Dataset first, then AUC descending
  results <- results[order(results$Dataset, -results$AUC), ]

  rownames(results) <- NULL  # Explicitly reset row names
  return(results)  # Return the sorted results table
}
```

5.2. Evaluating Our Models

```r
# Evaluate models for imbalanced dataset
evaluation_imbalanced <- suppressMessages(
  evaluate_models(models_imbalanced,
                  test_data_imbalanced,
                  "Imbalanced Dataset")
)

# Evaluate models for balanced dataset
evaluation_balanced <- suppressMessages(
  evaluate_models(models_balanced,
                  test_data_balanced,
                  "Balanced Dataset")
)

# Combine Results into a Single Table
comparison_table <- rbind(evaluation_imbalanced, evaluation_balanced)

# Display the Table
comparison_table %>%
  kable(
    format = "latex", # Use "latex" format for PDF
    booktabs = TRUE,  # Use booktabs style for LaTeX
    caption = "Comparison of Model Performance on Imbalanced and Balanced Datasets",
    col.names = c(
      "Dataset", "Model", "AUC", "ROC", "Optimal Threshold",
      "Sensitivity", "Specificity", "Precision", "Accuracy", "Confidence Interval"
    ),
    align = c("l", "l", "c", "c", "c", "c", "c", "c", "c", "c")
  ) %>%
  kable_styling(
    latex_options = c("striped", "hold_position", "H"), # Prevent floating with "H"
    font_size = 4
  ) %>%
  add_header_above(c(" " = 2, "Performance Metrics" = 8)) # Add header for grouped columns
```

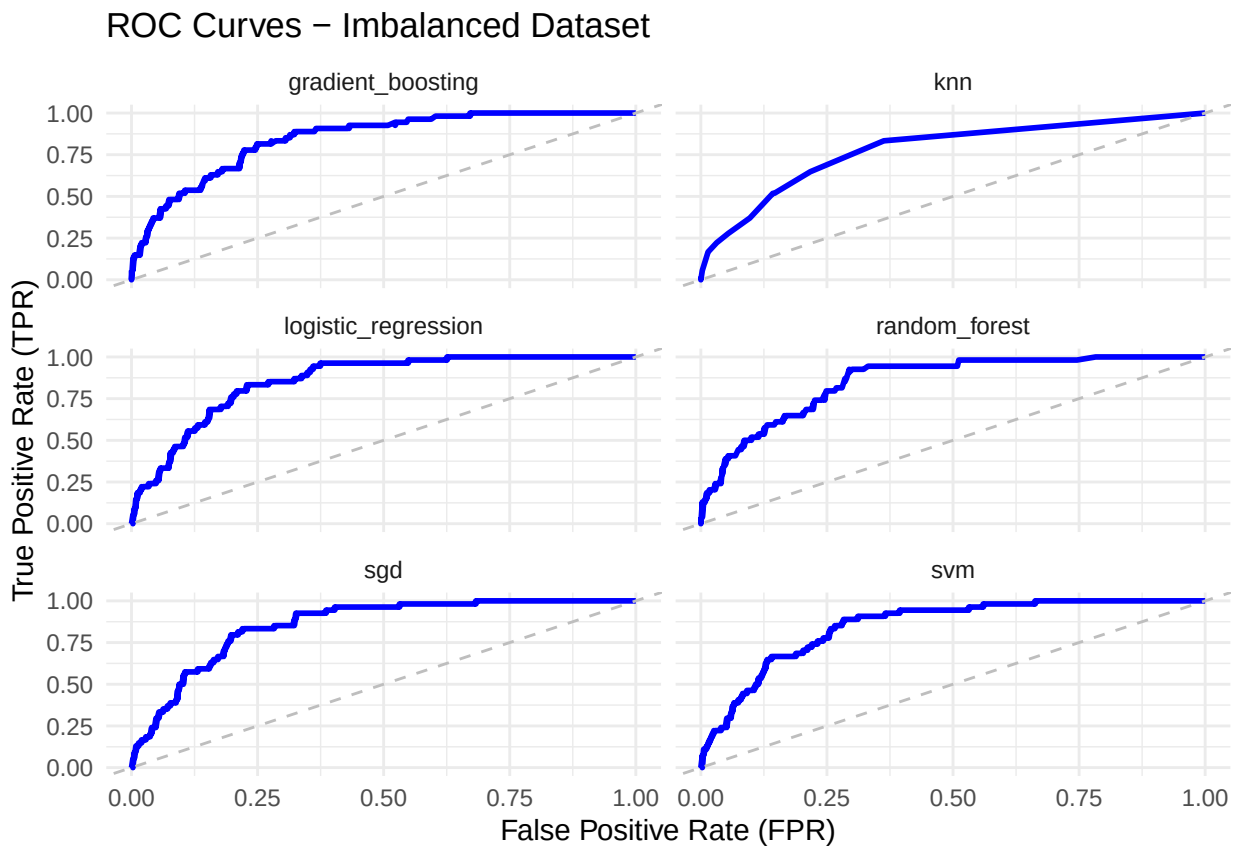
[!h]

Table 7: Comparison of Model Performance on Imbalanced and Balanced Datasets

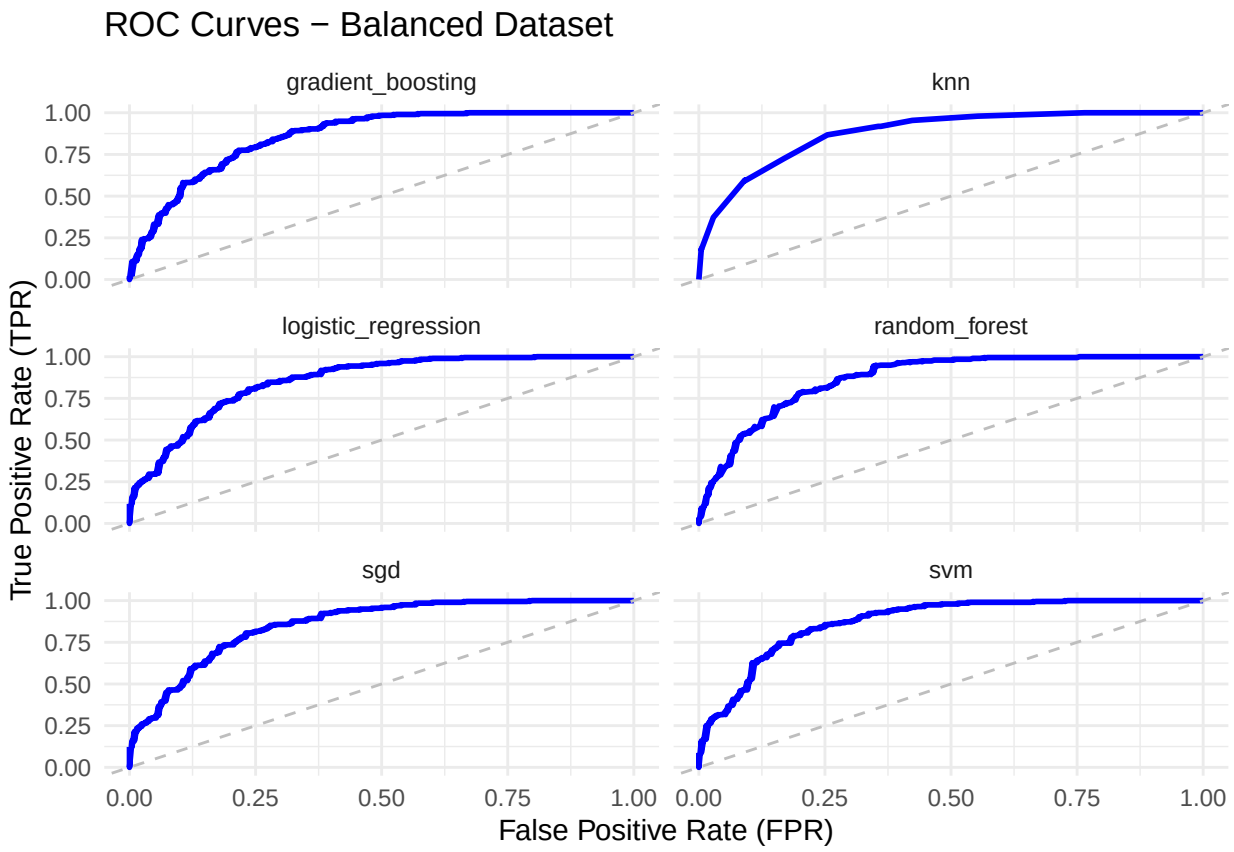
		Performance Metrics							
Dataset	Model	AUC	ROC	Optimal Threshold	Sensitivity	Specificity	Precision	Accuracy	Confidence Interval
Imbalanced Dataset	logistic_regression	0.858	TPR=0.833 | FPR=0.229	0.130	0.833	0.771	0.360	0.780	0.814 - 0.901
Imbalanced Dataset	sgd	0.856	TPR=0.833 | FPR=0.22	0.161	0.833	0.780	0.369	0.787	0.812 - 0.901
Imbalanced Dataset	random_forest	0.853	TPR=0.926 | FPR=0.294	0.099	0.926	0.706	0.327	0.735	0.807 - 0.9
Imbalanced Dataset	svm	0.851	TPR=0.889 | FPR=0.283	0.103	0.889	0.717	0.327	0.740	0.805 - 0.897
Imbalanced Dataset	gradient_boosting	0.847	TPR=0.815 | FPR=0.249	0.100	0.815	0.751	0.336	0.760	0.797 - 0.896
Imbalanced Dataset	knn	0.780	TPR=0.833 | FPR=0.363	0.056	0.833	0.637	0.262	0.663	0.714 - 0.845
Balanced Dataset	knn	0.880	TPR=0.867 | FPR=0.255	0.500	0.867	0.745	0.762	0.804	0.849 - 0.912
Balanced Dataset	svm	0.874	TPR=0.832 | FPR=0.221	0.522	0.832	0.779	0.780	0.804	0.84 - 0.908
Balanced Dataset	random_forest	0.870	TPR=0.949 | FPR=0.351	0.337	0.949	0.649	0.718	0.795	0.836 - 0.904
Balanced Dataset	gradient_boosting	0.858	TPR=0.893 | FPR=0.322	0.300	0.893	0.678	0.723	0.782	0.822 - 0.894
Balanced Dataset	sgd	0.856	TPR=0.806 | FPR=0.231	0.517	0.806	0.769	0.767	0.787	0.82 - 0.892
Balanced Dataset	logistic_regression	0.855	TPR=0.847 | FPR=0.274	0.431	0.847	0.726	0.744	0.785	0.819 - 0.891

```r
# Function to Plot ROC Curves for Models in a Grid
plot_roc_curves_grid <- function(models, test_data, dataset_name) {
  x_test <- test_data[, -which(names(test_data) == "quality_binary")]
  y_test <- factor(test_data$quality_binary, levels = c("bad", "good"))

  roc_data <- data.frame()  # Initialize an empty data frame for ROC data

  for (model_name in names(models)) {
    model <- models[[model_name]]

    # Predict probabilities for "good"
    probs <- predict(model, newdata = x_test, type = "prob")
    roc_curve <- roc(response = y_test, predictor = probs[, "good"],
                     levels = c("bad", "good"))

    # Extract TPR and FPR for the ROC curve
    roc_df <- data.frame(
      FPR = 1 - roc_curve$specificities,  # False Positive Rate
      TPR = roc_curve$sensitivities,      # True Positive Rate
      Model = model_name                  # Model name for grouping
    )

    roc_data <- rbind(roc_data, roc_df)  # Append data for this model
  }

  # Plot the ROC curves in a grid
  ggplot(roc_data, aes(x = FPR, y = TPR)) +
    # Line for each ROC curve
    geom_line(size = 1, color = "blue") +
    # Random guess line
```

```
    geom_abline(slope = 1, intercept = 0, linetype = "dashed", color = "gray") +
    labs(
      title = paste("ROC Curves -", dataset_name),
      x = "False Positive Rate (FPR)",
      y = "True Positive Rate (TPR)"
    ) +
    theme_minimal() +
    facet_wrap(~ Model, ncol = 2)  # Create a grid of plots with 2 columns
}


# Plot ROC Curves for Imbalanced Data Set
roc_plot_imbalanced <- suppressMessages(suppressWarnings(
  plot_roc_curves_grid(models_imbalanced,
                       test_data_imbalanced,
                       "Imbalanced Dataset")
))

print(roc_plot_imbalanced)
```



```r
# Plot ROC Curves for Balanced Dataset
roc_plot_balanced <- suppressMessages(suppressWarnings(
  plot_roc_curves_grid(models_balanced,
                       test_data_balanced,
                       "Balanced Dataset")
))

print(roc_plot_balanced)
```



CONCLUSION

Using AUC as our evaluation standard, for the imbalanced data set, logistic regression with a threshold of 0.130 is the best-performing model, achieving an AUC of 0.858, a sensitivity of 0.833, and a specificity of 0.771. For the balanced data set, KNN with a threshold of 0.5 demonstrates superior performance, achieving an AUC of 0.880, a sensitivity of 0.867, and a specificity of 0.745.

If the balanced data set approach is adopted for future predictions, the data must first be balanced using the ROSE package to ensure consistency in the pre-processing pipeline. This step is critical for replicating the conditions under which the KNN model was trained and evaluated.

I believe that this model is yet the best one. Further research on better model would be valuable. As a comparison, a guy in Kaggle community is able to build a Random Forest model (with feature engineering) with accuracy up to 92% (https://www.kaggle.com/code/karishmathakrar/92-accuracy-wine-insights-uncorked). Our KNN model on balanced data set only hits 80% of accuracy. Yes, need to learn more! Apart from that, better/more significant features in the data set would be helpful in real life to produce better classification model (remember that some features of the original data set are not open to public so we are not able to utilize them, which might be even more valuable).