UNIVERSITÀ DEGLI STUDI DELL'INSUBRIA

DEPARTMENT OF THEORETICAL AND APPLIED SCIENCES

MASTER'S DEGREE IN
COMPUTER SCIENCE

Intelligent Systems Assignment

Sassi Gabriele

Contents

List of Figures

List of Tables

Chapter 1

Classification task

1.1 Introduction

The aim of this task is to conduct a binary classification problem, with R language, on a diabetes dataset, employing advanced machine learning models to predict the presence or absence of diabetes based on various health-related features. This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Prima Indian Heritage. The current information set, [1]consisting of 768 data points, includes:

Input Variables (Features):

- **Pregnancies**: to express the number of pregnancies;
- **Glucose**: to express the glucose level in blood;
- **Blood pressure**: to express the blood pressure measurement;
- **Skin thickness**: to express the thickness of the skin;
- **Insulin**: to express the insulin level in blood;
- **BMI**: to express the Body Mass Index;
- **Diabetes pedigree function**: to express the diabetes percentage;
- **Age**: to express the age;

Output Variable (Target):

- **Outcome**: Target dependent variable; to express the final result:
  - **1** = yes;
  - **0** = no.

[1]https://www.kaggle.com/datasets/mathchi/diabetes-data-set

1.1.1 Objective

The primary goal is to build a robust predictive models capable of classifying samples into distinct categories; those with and without diabetes. This classification task is crucial for early detection and effective management of diabetes, a prevalent health condition with significant public health implications.

1.1.2 Models to be employed

Two distinct machine learning models will be employed for this binary classification task:

1. **Random Forest Classifier**: is a bagging ensemble learning techniques that utilizes the power of decision trees. By aggregating the predictions of individual trees, Random Forest enhances the model's accuracy, generalization, and robustness to overfitting. This model excels in handling different and complex datasets, making it a well-suitable choice for out diabetes classification task.
2. **Support Vector Machine (SVM)**: is a powerful algorithm for both classification and regression tasks. SVM works by finding the optimal hyperplane that best separate the classes in the feature space. This model is particularly effective in scenarios with high-dimensional data and non-linear decision boundary. By exploiting the strengths of SVM, we aim to achieve a robust and accurate diabetes classification model.

1.1.3 Significance of the project

The successful implementation of these machine learning models holds great promise in contributing to the field of healthcare analytics. Accurate predictions can facilitate early intervention, personalized treatment plans, and improve healthcare outcomes for individuals at risk of diabetes. This project is not just an assessment of knowledge about "Intelligent Systems", but it could evolve into a practical application with the potential to positively impact public health strategies. The subsequent sections will be delve into data exploration, preprocessing, model development, and evaluation, providing a comprehensive walkthrough of the steps involved in achieving classification task objectives.

1.2 Data Exploration

1.2.1 Upload dataset and descriptive information

Before delving into the detailed statistical analysis, let's upload the dataset and explore the statistical summary for each variable. The following information provides an overview of the central tendencies, variability, and distribution shapes, helping us comprehend the underlying patterns and trends within the data:

```
library(psych)
library(corrplot)
library(caret)
library(randomForest)
library(e1071)
library(pROC)
library(ggplot2)
library(car)
library(gridExtra)

diabetes_dataset <- read.csv(file="/Users/lele1312/
IntelligentSystems/Assignment/Data/diabetes.csv",head=TRUE,sep=",")
diabetes_dataset
describe(diabetes_dataset)
```

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se
Pregnancies	1	768	3.85	3.37	3.00	3.46	2.97	0.00	17.00	17.00	0.90	0.14	0.12
Glucose	2	768	120.89	31.97	117.00	119.38	29.65	0.00	199.00	199.00	0.17	0.62	1.15
BloodPressure	3	768	69.11	19.36	72.00	71.36	11.86	0.00	122.00	122.00	-1.84	5.12	0.70
SkinThickness	4	768	20.54	15.95	23.00	19.94	17.79	0.00	99.00	99.00	0.11	-0.53	0.58
Insulin	5	768	79.80	115.24	30.50	56.75	45.22	0.00	846.00	846.00	2.26	7.13	4.16
BMI	6	768	31.99	7.88	32.00	31.96	6.82	0.00	67.10	67.10	-0.43	3.24	0.28
DiabetesPedigreeFunction	7	768	0.47	0.33	0.37	0.42	0.25	0.08	2.42	2.34	1.91	5.53	0.01
Age	8	768	33.24	11.76	29.00	31.54	10.38	21.00	81.00	60.00	1.13	0.62	0.42
Outcome	9	768	0.35	0.48	0.00	0.31	0.00	0.00	1.00	1.00	0.63	-1.60	0.02

Figure 1.1: Diabetes descriptive statistics.

The statistical summary in Fig. 1.1, shows several key characteristics of each variable. The dataset exhibits variations in mean and median values, a range of variability and dispersion, and several instances of skewness and kurtosis. It's clear that many of these variables are not normally distributed, with skewness in both directions and a range of kurtosis values.

The results of the Shapiro-Wilk test in Tab. 1.1 indicate that the variables in the dataset do not follow a normal distribution. This is deduced from the very low *p-values* for each variable. In statistics, a very low p-value (typically less that 0.05) rejects the null hypothesis, which in this case would be that the data follow a normal distribution.

Variable	p-value
Pregnancies	< 2.2e-16
Glucose	1.986e-11
BloodPressure	< 2.2e-16
SkinThickness	< 2.2e-16
Insulin	< 2.2e-16
BMI	1.842e-15
DiabetesPedigreeFunction	< 2.2e-16
Age	< 2.2e-16

Table 1.1: Diabetes Shapiro-Wilk results

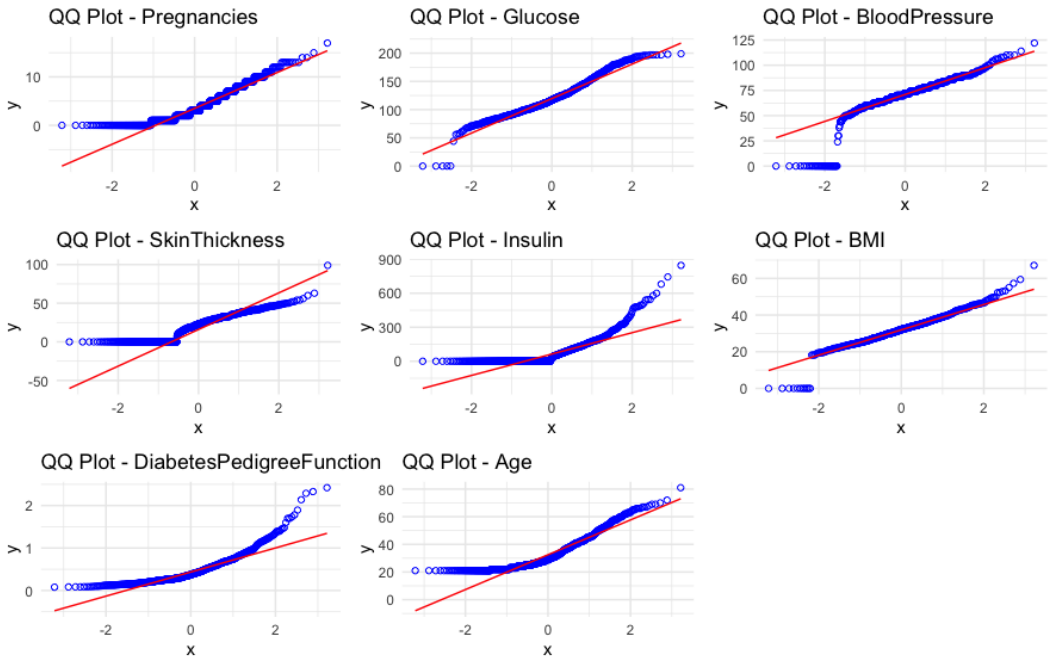

Figure 1.2: Diabetes QQ-Plots.

The graphical representations shown in Fig.1.2 further explain what the Shapiro-Wilk test p-value suggest. In QQ plots, the red line represents the theoretical normal distribution. If the points significantly deviate from this line, it indicates that the data do not fit well to a normal distribution.

1.2.2 Check balance of the data

In machine learning, the balance of data is a fundamental consideration that significantly influences the effectiveness of models, particularly in binary classification scenarios where *"outcome"* are categorized as 0 or 1. Several methods can be used to address imbalance, the choice depends on nature of the dataset and algorithms being used.

Outcome	Count
0	500
1	268

Table 1.2: Distribution of Outcome

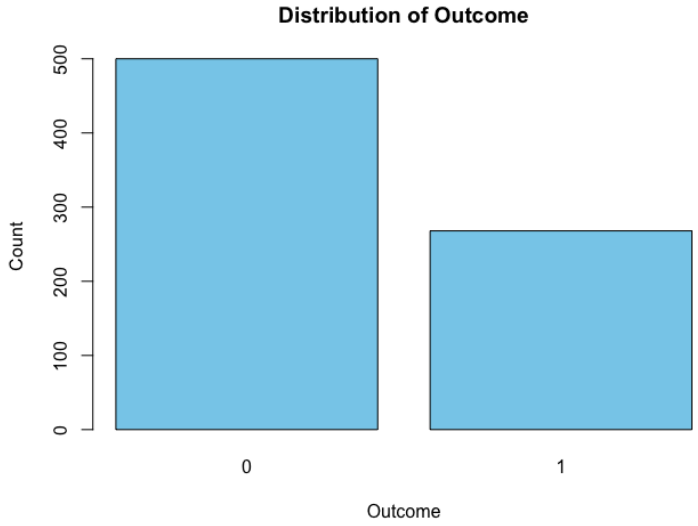

Figure 1.3: Distribution of outcome.

Through the visual representation provided by bar plot (In Fig. 1.3), it becomes evident that our dataset exhibits a significant imbalance between the two outcomes classes. The bar plot reveals a disparity, indicating that instances labeled as **"0"** are more predominant than those labeled as **"1"**. The visual insight further underscores the importance of addressing the imbalance to ensure a more equitable and robust performance of machine learning models, avoiding biases toward the majority class and fostering accurate predictions for both outcomes. In summary, we must taking into account this issue and try to find a valid trade-off before and during the training phase.

1.3 Preprocessing

1.3.1 Noise detection and treatment

The focus is on identify and handling null values within the dataset. Nullify values can significantly impact the accuracy of analyses, particularly when dealing with variables that cannot logically take a zero value. The first step involves a thorough examination of the dataset to identify any instances of missing and null values:

```
# Count missing values for each column
missing_values <- colSums(is.na(diabetes_dataset))
missing_values

# Count nullify values for each column
columns_to_exclude <- c("Pregnancies" , "Outcome")
filtered_data <- diabetes_dataset[, !colnames(diabets_dataset)
%in% columns_to_exclude]
nullify_values <- colSums(filtered_data== 0, na.rm = TRUE)
nullify_values
```

Following the identification process, a visual representation is created using a bar plot to showcase the distribution of nullify values across different observation:

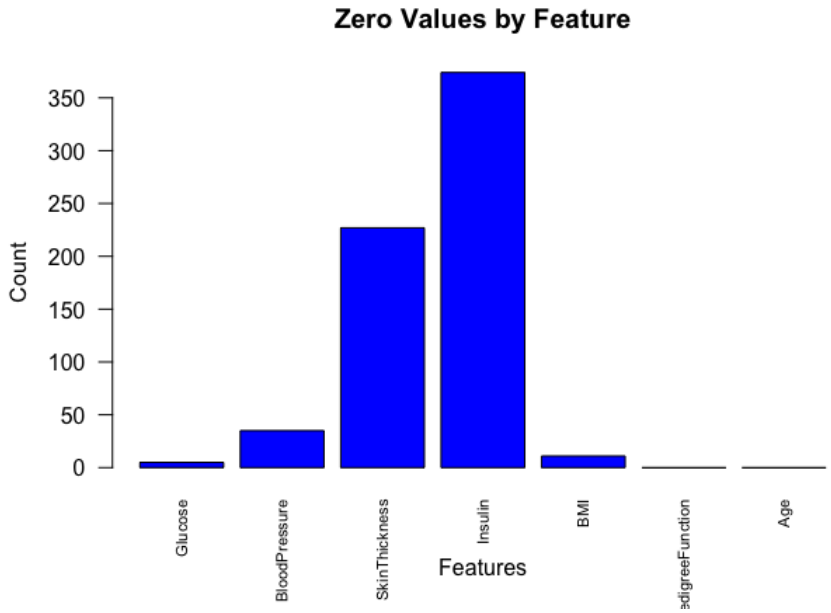

Figure 1.4: Bar plot of nullify values.

It's important to note that there are no missing values; however, nullify values, represented as zeros, are present in columns: *'Glucose', 'BloodPressure', 'SkinThickness' , 'Insulin', 'BMI'*. To rectify this issue, a crucial data cleaning step is applied. Specifically, the nullify values are systematically replaced with the standard missing value indicator, **"NA" (Not Available)**. This transformation ensures that subsequent analyses appropriately recognize and handle these values as missing data. Treating nullify values as missing data is essential for accurate statistical analyses, as these values can significantly impact calculations and interpretations:

```
       Glucose       BloodPressure       SkinThickness
             5                  35                 227
       Insulin                 BMI DiabetesPedigreeFunction
           374                  11                   0
           Age
             0
```

Figure 1.5: Count of nullify values.

```
cols <- c('Glucose', 'BloodPressure', 'SkinThickness',
'Insulin', 'BMI')
# Replace zero values with NA
diabetes_dataset[, cols][diabetes_dataset[, cols] == 0] <- NA
# Print the structure of the data frame
str(diabetes_dataset)
```

```
$ Pregnancies             : int  6 1 8 1 0 5 3 10 2 8 ...
$ Glucose                 : int  148 85 183 89 137 116 78 115 197 125 ...
$ BloodPressure           : int  72 66 64 66 40 74 50 NA 70 96 ...
$ SkinThickness           : int  35 29 NA 23 35 NA 32 NA 45 NA ...
$ Insulin                 : int  NA NA NA 94 168 NA 88 NA 543 NA ...
$ BMI                     : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 NA ...
$ DiabetesPedigreeFunction: num  0.627 0.351 0.672 0.167 2.288 ...
$ Age                     : int  50 31 32 21 33 30 26 29 53 54 ...
$ Outcome                 : int  1 0 1 0 1 0 1 0 1 1 ...
```

Figure 1.6: Nullify values replaced.

1.3.2 Outliers detection and treatment

Outliers are data points that significantly differ from the majority of the observations in a dataset. They can be values that are unusually high or low compared to the rest of the data. Identifying and addressing outliers is a critical step in the data analysis process, as they can have substantial effects on statistical analyses and machine learning models: they can distort and impact on measures of central tendency and dispersion, leading to an inaccurate representation of the spread of the data. In addition, and most importantly, outliers can disproportionately influence model training, resulting in models that do not generalize well to new data. One effective way to visually identify outliers is through box plots (Fig 1.7). In a box plot, extreme values are typically represented as points beyond the "whiskers" of the plot.

The box represents the IQR and the median, while the whiskers extend to the minimum and maximum values within a certain range:

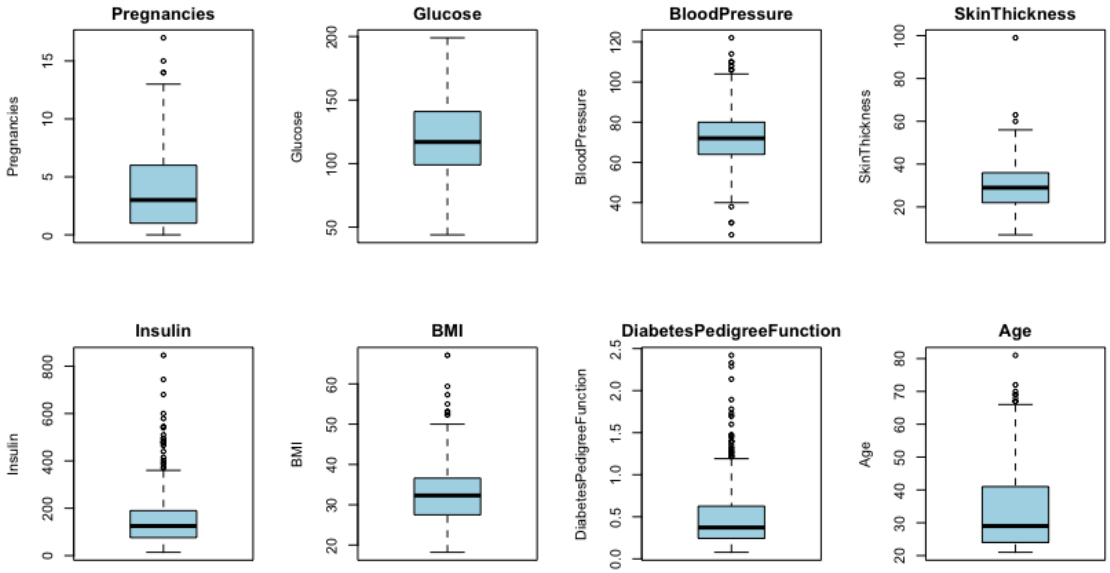

Figure 1.7: Box plot of outliers.

Detecting outliers using IQR method

Detecting outliers is crucial for ensuring the quality and reliability of the dataset, contributing to the development of more robust and accurate statistical models. The following code include functions designed to identify outliers in a dataset, employing a statistical technique known as **Interquantile Range (IQR)** method.
The first function systematically identifies outliers within a dataset column. Missing values are first removed to ensure a clean and accurate analysis; the remaining data is sorted and quartiles (Q1 and Q3) are calculated. Then, IQR, representing the spread of the middle 50% of the data, is determined. Lower and upper bounds are established based on the IQR, any data points falling outside these bounds are considered outliers, which are returned by the function as a list:

```r
    #IQR function to detect outliers
    detect_outliers_iqr <- function(data) {
    bool_series <- !is.na(data)
    data <- data[bool_series]
    data <- sort(data)

    q1 <- quantile(data, 0.25)
    q3 <- quantile(data, 0.75)

    IQR <- q3 - q1
    lwr_bound <- q1 - 1.5 * IQR
    upr_bound <- q3 + 1.5 * IQR

    outliers_down <- data[data < lwr_bound]
    outliers_up <- data[data > upr_bound]

    return(list(outliers_up = outliers_up, outliers_down = outliers_down))
}
```

The following functions serve as an interface to the *detect_ outliers_iqr* function, providing a clear and detailed presentation of the outlier information. For a specific column in the dataset, it prints values about the upper and lower detected outliers.

```r
    #Function to print outliers
    print_outliers_iqr <- function(data, column_name) {
    outliers <- detect_outliers_iqr(data)

    cat("IQR outliers:", column_name, "\n")
    cat("Outliers (Upper):", outliers$outliers_up, "\n")
    cat("Outliers (Lower):", outliers$outliers_down, "\n")

    }

    #Columns to check for outliers
    col_names <- c('Pregnancies', 'Glucose', 'BloodPressure',
    'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age')

    # Loop through columns and print outliers
    for (col in col_names) {
    x <- print_outliers_iqr(diabetes_dataset[[col]], col)
    }
```

Here's a summary result:

```
IQR outliers: Pregnancies
Outliers (Upper): 14 14 15 17
Outliers (Lower):
IQR outliers: Glucose
Outliers (Upper):
Outliers (Lower):
IQR outliers: BloodPressure
Outliers (Upper): 106 106 106 108 108 110 110 110 114 122
Outliers (Lower): 24 30 30 38
IQR outliers: SkinThickness
Outliers (Upper): 60 63 99
Outliers (Lower):
IQR outliers: Insulin
Outliers (Upper): 370 375 387 392 402 415 440 465 474 478 480 480 485 495 495 510 540 543 545 579 600 680 744 846
Outliers (Lower):
IQR outliers: BMI
Outliers (Upper): 52.3 52.3 52.9 53.2 55 57.3 59.4 67.1
Outliers (Lower):
IQR outliers: DiabetesPedigreeFunction
Outliers (Upper): 1.213 1.222 1.224 1.224 1.251 1.258 1.268 1.282 1.292 1.318 1.321 1.353 1.39 1.391 1.394 1.4 1.441 1.461
1.476 1.6 1.698 1.699 1.731 1.781 1.893 2.137 2.288 2.329 2.42
Outliers (Lower):
IQR outliers: Age
Outliers (Upper): 67 67 67 68 69 69 70 72 81
Outliers (Lower):
```

Figure 1.8: Outliers value.

Treating outliers using IQR method

Once outliers have been identified in the dataset, it becomes essential to address them appropriately to ensure the integrity and reliability of the data. The function responsible for treating outliers is designed to replace identified outlier values, within each specified column, with the corresponding lower or upper bounds. The bounds are established based on the calculated IQR: any data point falling below **Q1-1.5IQR** or above **Q3+1.5IQR** is considered an outlier.

```r
#Columns to analyze
col_names <- c('Pregnancies', 'Glucose', 'BloodPressure',
'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age')

#Loop through each specified column
for (col in col_names) {
#Detect outliers using the IQR method
outliers_info <- detect_outliers_iqr(diabetes_dataset[[col]])

#Calculate quartiles and IQR
q1 <- quantile(diabetes_dataset[[col]], 0.25, na.rm = TRUE)
q3 <- quantile(diabetes_dataset[[col]], 0.75, na.rm = TRUE)
IQR <- q3 - q1
```

```
#Define lower and upper bounds
lwr_bound <- q1 - 1.5 * IQR
upr_bound <- q3 + 1.5 * IQR

#Replace outliers with lower and upper bounds
diabetes_dataset[[col]][diabetes_dataset[[col]] %in%
outliers_info$outliers_down] <- lwr_bound
diabetes_dataset[[col]][diabetes_dataset[[col]] %in%
outliers_info$outliers_up] <- upr_bound

#Clear the lists of outliers
outliers_up <- c()
outliers_down <- c()
}
```

Now, using the previous box plot again, we can see in Fig. 1.9 that there are no longer values above the whiskers:

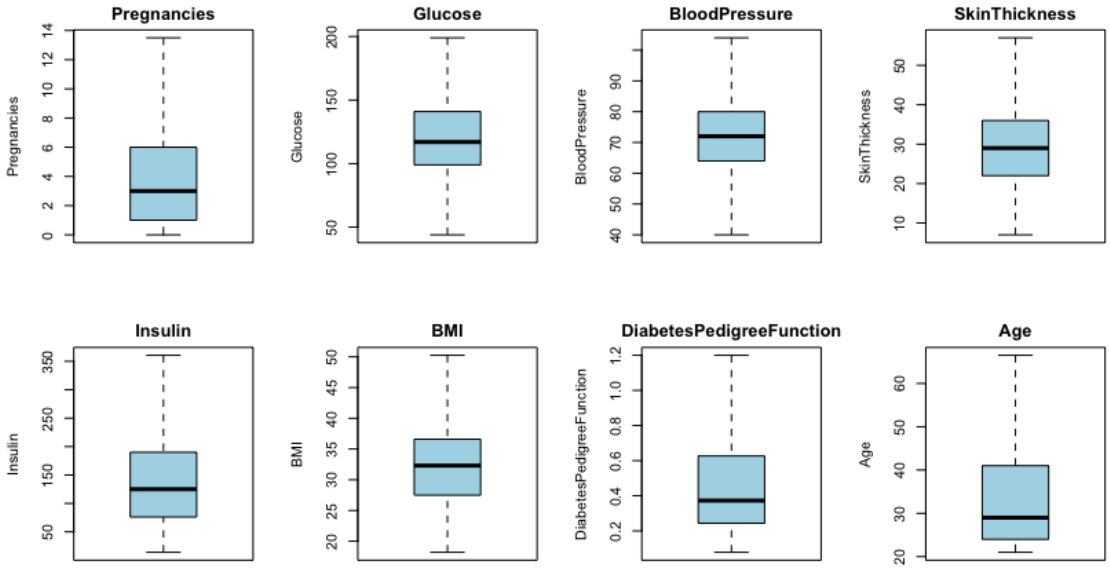

Figure 1.9: No outliers visualization.

The decision to use IQR method for outliers treatment is grounded in its robustness and ability to handle skewed distributions and detecting value that deviate significantly from the central tendency of the data. By focusing on the middle range of the data, the IQR method is less sensitive to extreme values than other methods.

1.3.3 Impute missing values

Impute missing values refers to the process of filling in or replacing missing data in our observations with estimated or substituted values. There are several methods. A common approach is to apply the "median imputation" method to features with e limited number of missing value such as *"Glucose", "BloodPressure", "BMI"*: missing values for each column are replaced with the median of the observed values. This method is straightforward and computationally less intensive compared to some other imputation methods. Its simplicity makes it an attractive choice, especially when dealing with a limited number of missing values.

```
    #Fill 3 features having few NAs using median method.
    features_to_fill <- c('Glucose', 'BloodPressure', 'BMI',
    'SkinThickness', 'Insulin')

    for (feature in features_to_fill) {
    #Replace missing values with the median of the observed
    values in that feature
    diabetes_dataset[[feature]][is.na(diabetes_dataset[[feature]])] <-
    median(diabetes_dataset[[feature]], na.rm = TRUE)
}
```

Now, turn on attention to features that exhibit a substantial number of missing values: *"SkinThickness", "Insulin"*. For these features, a fundamental strategy involves leveraging the correlation between the feature with missing values and other variables in the dataset for imputation. By measuring the correlation, the existing relationships are used to estimate the missing value accurately, in a manner that considers the broader data structure. It provides a more sophisticated approach compared to generic methods like median imputation. This technique aligns with a data-driven philosophy, where imputation is guided by observed pattern and dependecies within the dataset.

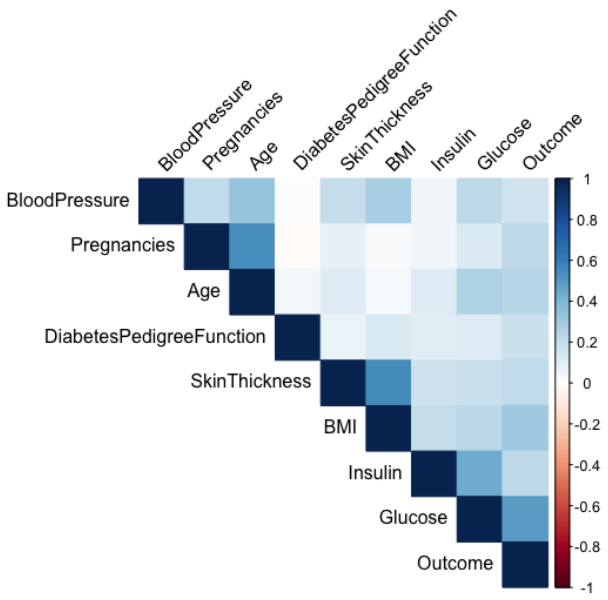

Figure 1.10: Diabetes Correlation Matrix.

The correlation matrix in Fig. 1.10 illustrates a strong positive correlation between:

- **SkinThickness-BMI**: Individuals with higher BMI values may tend to have thicker skin fold.
- **Insuline-Glucose**: Indicates a connection between the insulin levels and glucose concentrations in the dataset. Insulin production is intricately linked to blood glucose levels.

Given these observed correlations, we can leverage this information to impute missing values in the dataset by using "forward filling" approach. It replaces missing values with the most recent non-missing value observed before them in the sorted dataset. For example, if there is a missing value in the *"Insulin"* column at a certain row, it will be replaced with the last observed non-missing value in the sorted sequence based on *"Glucose"* column. The assumption here, previously checked, is that, in the dataset related to glucose and insulin levels, there might be a natural order where observations with similar or close glucose levels are expected to have similar insulin levels.

```
#Treat missing values in Insulin highly correlated with Glucose
#Sort the dataframe by Glucose
diabetes_dataset <- diabetes_dataset[order(diabetes_dataset$Glucose),]
```

```r
#Forward fill missing values in Insulin
for (i in 2:nrow(diabetes_dataset)) {
if (is.na(diabetes_dataset$Insulin[i])) {
    diabetes_dataset$Insulin[i] <- diabetes_dataset$Insulin[i - 1]
    }
}

#Treat missing values in SkinThickness highly correlated with BMI
#Sort the dataframe by BMI
diabetes_dataset <- diabetes_dataset[order(diabetes_dataset$BMI),]

#Forward fill missing values in Insulin
for (i in 2:nrow(diabetes_dataset)) {
if (is.na(diabetes_dataset$SkinThickness[i])) {
    diabetes_dataset$SkinThickness[i] <- diabetes_dataset$SkinThickness
    [i - 1]
    }
}
```

The previous steps are crucial in data analysis because many statistical algorithms and models cannot handle missing data. Imputation helps maintain the integrity of the dataset, ensuring that the model training can proceed with as much information as possible.

With the preprocessing step completed, including crucial activities such as data preparation, outliers detection, and handling missing values,the dataset has been carefully prepared. Now we transition into the modeling phase. It involves the application of machine learning algorithms to the selected dataset. As previously mentioned in the documentation, two distinct algorithms will be employed to uncover pattern and make predictions. The modelling phase encompasses essential steps, including data splitting, data scaling and training/predictions processes with the chosen machine learning models. Through these efforts, our goal is not just to see how well the models perform on the training data but also to check how good they are at making predictions on new, unseen data.

1.3.4 Data splitting

The process involves preparing the data to for training and testing the classifiers. The predictor variables ($\mathbf{X}$) are isolated, excluding the *"Outcome"* column, which is the response variable ($\mathbf{y}$) we aim to predict. For reproducibility, a seed value of 42 is established, ensuring consistent and repeatable results whenever the code is executed. A deliberate decision is made to allocate 25% of the dataset for testing purposes. To achieve this, random indices are generated, corresponding to a quarter of the dataset.

The dataset is then effectively partitioned, based on these randomly selected indices, into:

- **X_train - y_train**: the training sets, ensuring a robust training process.
- **X_test - y_test**: the test sets, consist of instances specifically designated for testing the model's generalization ability.

```
#Extract predictor variables (X) excluding 'Outcome'
X <- diabetes_dataset[, !names(diabetes_dataset) %in% c('Outcome')]
#Extract the response variable (y)
y <- diabetes_dataset$Outcome
#Set the seed for reproducibility
set.seed(42)
#Specify the proportion for the test set
test_proportion <- 0.25
#Generate random indices for the test set
test_indices <- sample(1:length(y), size = test_proportion * length(y))
#Split the data into training and test sets
X_train <- X[-test_indices, ]
X_test <- X[test_indices, ]
y_train <- y[-test_indices]
y_test <- y[test_indices]
```

This process is fundamental to machine learning workflow, allowing us to assess the model's performance on unseen data. By introducing randomness in the selection of test indices, we unsure a different representation of instances, enhancing the model's capacity to handle various scenarios.

1.3.5 Data scaling

In the data scaling phase, we aim to ensure that all features are on a similar scale for optimal model performance. "Scale" refers to the process of adjusting the numerical values of features in a dataset to a standard distribution. This is typically done to ensure that all features contribute equally to the modelling process, preventing one feature from dominating due to significantly values difference. Training and test sets are combined and "Min-Max scaling" is applied. This approach involves transforms the values of each feature to a specific range. Finally we print the dimensions (shapes) of the scaled datasets in order to helps us verify the successful scaling of data.

```
#DATA SCALING
#Combine X_train and X_test for scaling
combined_data <- rbind(X_train, X_test)
#Specify the method for scaling
scaling_method <- c("range")
```

```r
#Use preProcess to scale the data
scaling_model <- preProcess(combined_data, method = scaling_method)
#Transform X_train and X_test using the scaling model
scaled_X_train <- predict(scaling_model, newdata = X_train)
scaled_X_test <- predict(scaling_model, newdata = X_test)
#Print the shapes
cat("Scaled X_train shape:", dim(scaled_X_train), "\n")
cat("Scaled X_test shape:", dim(scaled_X_test), "\n")
```

1.4 Random Forest classifier

1.4.1 Model's implementation

The provided code implements a Random Forest classifier using the *'randomForest'* function:

```r
#RANDOM FOREST CLASSIFIER
#Set the seed for reproducibility
set.seed(42)

#Create the random forest model
model <- randomForest(
x = scaled_X_train,
y = as.factor(y_train),
classwt = c(1, 2), #Class weights {0: 1, 1: 2}
ntree = 25,
mtry = sqrt(ncol(scaled_X_train)),
minsplit = 5,
maxdepth = 15,
nodesize = 15
)

#Make predictions on the training and test sets
train_predictions <- predict(model, newdata = scaled_X_train)
test_predictions <- predict(model, newdata = scaled_X_test)
```

Inside function there are parameters used to create the model, such as: the number of trees, the number of features considered for each split and tree depth, in order to avoid overfitting. In addition class weight are assigned to handle imbalanced classes. In this case, class 0 is assigned a weight of 1, and class 1 is assigned a weight of 2. At the end, the trained model is used to make predictions on both the training and test sets.

It's crucial to be aware that the Random Forest algorithm does not support datasets with missing values. Therefore, prior to applying the Random Forest model, ensure that all missing values have been appropriately handled. If you previously imputed missing values, it is recommended to revisit this step. Verify that no missing values remain in the dataset, especially in features used as predictors, by conducting through checks to confirm that your dataset is devoid of missing values before proceeding with the the Random Forest modeling phase. The figure 1.11 suggests a potential error when attempting to create the model with missing values in the dataset:

```
Error in randomForest.default(x = scaled_X_train, y = as.factor(y_train),  :
  NA not permitted in predictors
```

Figure 1.11: Implementation with NA values.

1.4.2 Model's performance evaluation

In evaluating the performance of the model, we focus on key metrics derived form the confusion matrix. These metrics provide insights into how well the model has learned patterns from training and test data and its ability to make accurate predictions.

Classification Report - Training Set

	Reference	
Prediction	**0**	**1**
0	300	5
1	80	191

Table 1.3: Random Forest confusion matrix training set

The confusion matrix shows the counts of true positive(TP), true negative(TN), false positive(FP) and false negative(FN) predictions. The table. 1.3 provides a snapshot of the model's performance on the training set by summarizing the predictions made by the model compared to the actual outcomes:

- **300** instances are correctly predicted as class 0(**TN**).
- **191** instances are correctly predicted as class 1(**TP**).
- **80** instances are incorrectly predicted as class 0 when they are actually class 1(**FN**).
- **5** instances are incorrectly predicted as class 1 when they are actually class 0(**FP**).

Metric	Value
Accuracy	85.24%
Sensitivity (Recall)	78.95%
Specificity	97.45%
Precision	98.36%
F1 score	87.59%

Table 1.4: Random Forest training set evaluation metrics

Analyzing the results shown in Table 1.4, the model exhibits a high **accuracy** of **85.24%**, indicating a substantial proportion of correct predictions. Notably, it demonstrates a commendable balance between sensitivity and specificity, achieving around **78.95% recall** for positive instances and an impressive **97.45% specificity** for negative instances. The high **precision** of **98.36%** underscores the model's ability to accurately identify positive cases among its predictions. The **F1 score**, which combines precision and recall, further solidifies the model's performance with a score of **87.59%**. These metrics collectively suggest that the model has learned well from the training data. However, it's crucial to exercise caution and consider the model's generalization performance on an independent test set. Additionally, the imbalance nature of the dataset, as observed during the initial data exploration, may influence the model's behavior. Further assessment on unseen data will provide a more comprehensive evaluation of its reliability in real-word scenarios.

Classification Report - Test Set

	Reference	
Prediction	**0**	**1**
0	81	10
1	39	62

Table 1.5: Random Forest confusion matrix test set

In Tab 1.5 the test set confusion matrix shows that:

- **81** instances are correctly predicted as class 0(**TN**).
- **39** instances are correctly predicted as class 1(**TP**).
- **10** instances are incorrectly predicted as class 0 when they are actually class 1(**FN**).
- **62** instances are incorrectly predicted as class 1 when they are actually class 0(**FP**).

Metric	Value
Accuracy	74.48%
Sensitivity (Recall)	67.50%
Specificity	86.11%
Precision	98.36%
F1 score	87.59%

Table 1.6: Random Forest test set evaluation metrics

Summarizing the test set evaluation metrics presented in Table 1.6, the models performs well on new, unseen data. The **accuracy** of **74-48%** indicates a substantial proportion of correct predictions, demonstrating the model's ability to generalize effectively. Notably, the **sensitivity** of **67.50%** showcases the model's capability to identify a considerable percentage of actual positive cases, while the **specificity** of **86.11%** highlights it accuracy in correctly identifying negative cases. This balanced performance is crucial because is essential minimizing both false positives and false negatives. The **precision** of **98.36%** underscores the model's accuracy in predicting positives cases, showcasing a low rate of false positives. The **F1 score**, considering both precision and recall, achieves a harmonious balance at **87.59%**, further supporting the model's robustness.

Final evaluation

The Random Forest model demonstrates promising performance, even considering the differences observed between the training and test sets. Although there are slight variations in metrics like accuracy and recall, these difference are not substantial. The model's ability to maintain a consistent level of accuracy and balance between positive and negative predictions across both subsets suggests its overall reliability. In machine learning, it's common for models to exhibit minor variations in performance between training and test sets. The crucial aspect is to ensure that these differences remain within an acceptable range. The observed small disparities in metrics indicate that the models has learned generalizable patterns from the training data and can effectively apply them to new, unseen instances. In conclusion, the ROC curve provided in Fig. 1.12 indicates that the Random Forest model is proficient at distinguishing between positive and negative classes. Complementing this visual indicator, an Area Under the Curve (AUC) of **0.8465** signifies that the model possesses a substantial discriminatory ability and appears to be reliable and valid for binary classification.

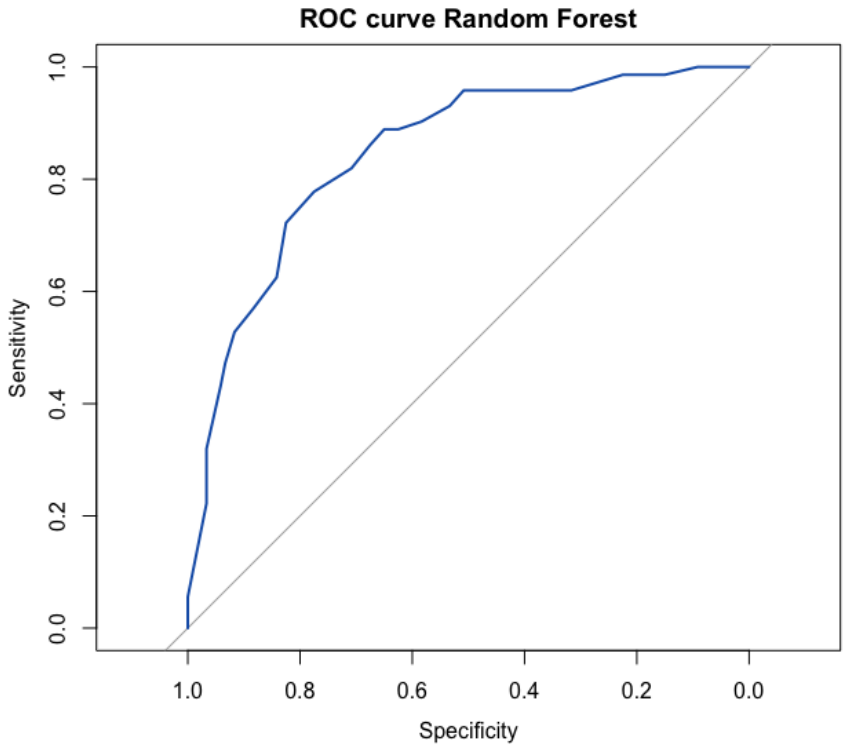

Figure 1.12: ROC curve of Random Forest model.

1.5 Support Vector Machine (SVM)

1.5.1 Model's implementation

The provided code implements a Support Vector Machine using the *'svm'* function:

```
#SUPPORT VECTOR MACHINE
model2 <- svm(as.factor(y_train) ~ ., data = scaled_X_train,
kernel = "radial", cost = 1, class.weights = c("0" = 1, "1" = 2))
#Print the model details
print(model2)

#Make predictions on the training and test sets
train_predictionsSVM <- predict(model2, newdata = scaled_X_train)
test_predictionsSVM <- predict(model2, newdata = scaled_X_test)
```

Inside the function there are parameters used to create the model, including "kernel" and "cost". The *'radial kernel'* transforms input data into a higher-dimensional space where non-linear relationships can be captured more effectively. The **'cost'** parameter controls the trade-off between a smooth decision boundary and accurate classification. Additionally, in the same way of Random Forest classifier, this SVM model is configured to handle imbalanced classes by assigning different weights to each class.

1.5.2 Model's performance evaluation

In evaluating the performance of the SVM model, the process is the same of the Random Forest Classifier, both training and test sets are assessed.

Classification Report - Training Set

	Reference	
Prediction	**0**	**1**
0	299	27
1	81	169

Table 1.7: SVM confusion matrix training set

The table. 1.7 provides a snapshot of the model's performance on the training set by summarizing the predictions made by the SVM model compared to the actual outcomes:

- **299** instances are correctly predicted as class 0(**TN**).
- **169** instances are correctly predicted as class 1(**TP**).

- **27** instances are incorrectly predicted as class 0 when they are actually class 1(**FN**).
- **81** instances are incorrectly predicted as class 1 when they are actually class 0(**FP**).

Metric	Value
Accuracy	81.25%
Sensitivity (Recall)	78.68%
Specificity	86.22%
Precision	91.71%
F1 score	84.70%

Table 1.8: SVM training set evaluation metrics

Considering the results described above (Tab. 1.8), the SVM model exhibits a solid performance on the training set, showcasing an **accuracy** of **81.25%**. This indicates that more than 80% of the predictions made by the model align with the actual outcomes in the training data. In terms of **sensitivity**, the model demonstrates a commendable **78.68%**, signifying its ability to effectively capture the majority of positive instances within the dataset. On the **specificity** front, with a score of **86.22%**, the model displays efficiency in correctly identifying negative instances. When the model predicts a positive outcome, it is accurate more than 91% of the time, due to **91.71%** of **precision** metric. The **F1 score** is observed to be **84.70%**. Overall, the SVM model's evaluation metrics on the training set indicate a robust and balanced performance, capturing its ability to generalize well to the training data.

Classification Report - Test Set

	Reference	
Prediction	**0**	**1**
0	91	9
1	29	63

Table 1.9: SVM confusion matrix test set

In Table 1.9 the test set confusion matrix shows that:

- **91** instances are correctly predicted as class 0(**TN**).
- **63** instances are correctly predicted as class 1(**TP**).
- **9** instances are incorrectly predicted as class 0 when they are actually class 1(**FN**).
- **29** instances are incorrectly predicted as class 1 when they are actually class 0(**FP**).

Metric	Value
Accuracy	80.21%
Sensitivity (Recall)	75.83%
Specificity	87.50%
Precision	91.00%
F1 score	82.72%

Table 1.10: SVM test set evaluation metrics

Examining the Table 1.10 the SVM model displays commendable performance on test set. The **accuracy**, standing at **80.21%**, indicates the proportion of corectly predicted instances. The model showcases a balanced performance between **sensitivity** (**75.83%**) and **specificity** (**87.50%**), affirming its capability in correctly identifying both positive and negative cases. Moreover, with a **precision** of **91.00%**, the model exhibits accuracy in predicting positive cases within its outputs. The **F1 score**, which considers the trade-off between precision and recall, is at a noteworthy **82.72%**, underscoring the model's overall effectiveness. These metrics collectively suggest that the SVM model has translated its learning from the training set to male accurate predictions on unseen data, as demonstrated by its robust performance on the test set.

Final evaluation

In summary, the SVM model exhibits consistent and reliable performance on both the training and test sets, suggesting its potential for real-word applications. The minimal disparity between the training and test metrics signifies the model's ability to generalize effectively without overfitting to the training data. The ROC curve (Fig. 1.13) and the associated AUC of **0.8755**, both point towards a strong predictive performance. Well above the diagonal line representing a random-chance classifier, Support Vector Machine curve is positioned significantly higher, indicating that the model's predictions are far from random and have statistical validity.

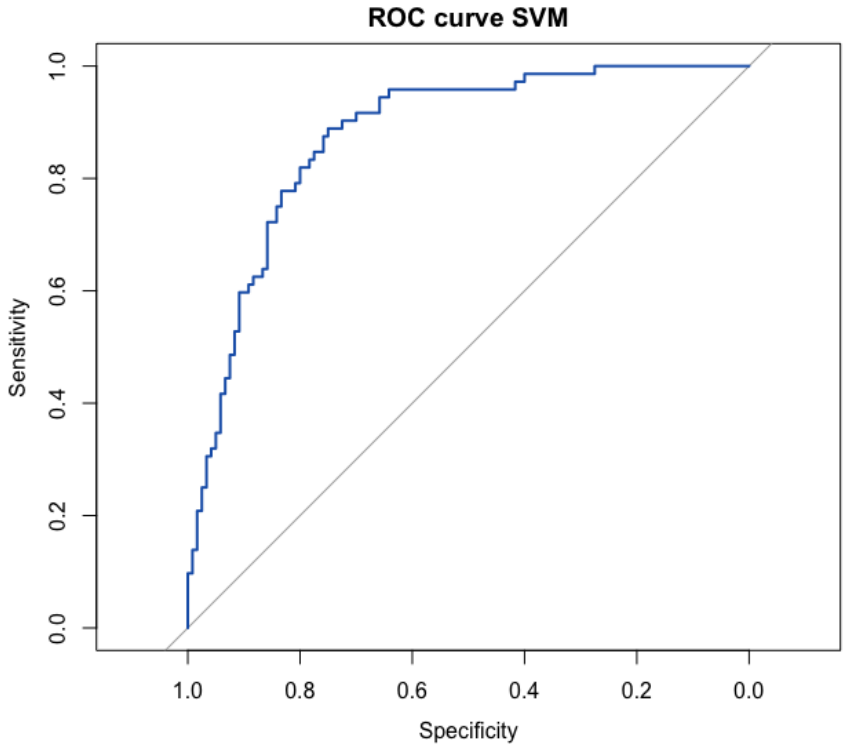

Figure 1.13: ROC curve of Support Vector Machine model.

1.6 Conclusions

In the realm of diabetes classification, the pursuit for a robust and reliable model has led to employ two powerful algorithms: the Random Forest and the Support Vector Machine. Each trained model can leverage its own strengths to create a genuine predictions with the available data.

In terms of overall comparison, the Random Forest model has a higher accuracy and specificity than the Support Vector Machine model, but it also has lower sensitivity. The Support Vector Machine has a more consistent performance across training and test sets, but it has lower specificity and precision.

Based on the performance metrics, the Random Forest models appears to be the stronger model overall. It has higher accuracy and specificity, and it has excellent precision. However, the Support Vector Machine model may be a more suitable choice for applications where sensitivity is more important than specificity. In addition, the performance of both models may be improved by using a larger training dataset.

In conclusion, the two models presented in this analysis have both demonstrated promising performance in diabetes classification. The random forest model is generally a more robust and versatile choice for diabetes prediction, offering a balance between accuracy, specificity and sensitivity. On the other hand, The Support Vector Machine model can be a valuable alternative when sensitivity predominant or when consistent performance across data sets is critical.

Chapter 2

Regression task

2.1 Introduction

The focus of this task in to conduct the regression analysis of a dataset associated with the red variants of Portuguese "Vinho Verde" wine. This dataset serves as a comprehensive resource for investigating the relationships between physicochemical attributes and sensory characteristics of the wine. To provide context, the dataset is deliberately limited to physicochemical variables as inputs and sensory variable as the output. Notably, information as grape types, wine brand and selling price has been omitted due to privacy and logistical considerations. The current information set, available from the UCI machine learning repository[1], consists of 1599 data points and includes:

Input Variables (Features):

- **Fixed acidity**: The amount of non-volatile acids in the wine, do not evaporate readily.
- **Volatile acidity**: The amount of volatile acids in the wine, which can contribute to unpleasant aromas.
- **Citric acid**: they can add freshness and flavor to the wine.
- **Residual sugar**: The amount of sugar remaining after fermentation, influencing sweetness.
- **Chlorides**: The presence of salt in the wine, which can affect taste.
- **Free sulfur dioxide**: The presence of sulfur dioxide, which acts as a preservative and antioxidant.
- **Total sulfur dioxide**: The total amount of sulfur dioxide, including both free and bound forms.
- **Density**: The density of the wine, which can be related to sweetness.
- **pH**: describes how acidic or basic a wine is on a scale from 0 (very acidic) to 14 (very basic).

[1] https://archive.ics.uci.edu/dataset/186/wine+quality

- **Sulphates**: The amount of sulfates, which can contribute to wine stability and aroma.

Output Variable (Target):

- **Quality**: output variable, it reflects the overall perception of the wine's taste and aroma, combining various sensory attributes. It' a score between 0 and 10.

2.1.1 Objective

The dataset becomes an invaluable tool for unraveling the intricate interplay between these physicochemical features and the sensory aspects of "Vinho Verde" wine. The primary goal is to employ regression techniques, to analyze the dataset and derive models that can predict the quality of wine based on its physicochemical properties. Be delving into these relationships, the regression task aim on understanding the influence of various input features on the sensory output, and ultimately, finding optimal coefficients that result in models capable of accurately predicting the final evaluation of the dependent variable.

2.1.2 Models to be employed

Two distinct machine learning models will be employed for this regression task:

1. **K-Nearest Neighbors Regression (KNN)**: is a algorithm that predicts the output of a data point by considering the average or weighted average of the output of its k-nearest neighbors. In other words, it makes predictions based on the similarity of neighboring data points. KNN can be valuable for predicting the quality of wine based on physicochemical features. By identifying wines with similar chemical profiles, KNN allows to infer the likely sensory attributes of a particular wine, contributing to a better understanding of the relationship between input variables and output variable.

2. **Ridge Regression**: is a linear regression technique that introduces a regularization term to traditional linear regression model. This regularization term (L2 regularization) helps prevent overfitting by penalizing large coefficients. In the context of wine quality regression, it is beneficial for handling multicollinearity among physicochemical features. It provides a robust approach to estimating the coefficients of the regression model, enhancing the model's generalization performance and ensuring that each input variable contributes meaningfully to the prediction of sensory output.

2.1.3 Significance of the project

This exploration is particularly significant for wine enthusiasts and industry professional seeking a deeper understanding of the factors influencing wine quality. In essence, this regression analysis is not just an academic exercise but a valuable tool with practical applications is the wine industry. It facilitates the production of wines that align with customer expectations and preferences, contributing to both the quality of the product and the success of winemaking business.

2.2 Data exploration

2.2.1 Upload dataset and data exploration

This section prepares the project by reading in and loading the data *.csv* file into a dataframe.

```
library(psych)
library(corrplot)
library(caret)
library(tidyverse)
library(ggplot2)
library(randomForest)
library(kknn)
library(dplyr)
library(reshape2)

wine_dataset <- read.csv(file="/Users/lele1312/IntelligentSystems
/Assignment/Data/winequality-red.csv",head=TRUE,sep=",")
wine_dataset
```

Then it provides some brief insight into the structure and summary statistics of the dataframe, available in Fig. 2.1, in order to understand the dataset's central tendencies and variations. The dataset under analysis presents a variety of distributions across its variables. Specifically, the variables *"Residual Sugar"*, *"Chlorides"*, *"Sulphates"* and *"Total Sulfur Dioxide"*, demonstrate significantly non-normal distributions as evidenced by their high skewness and kurtosis values. These measures suggest pronounced asymmetry in their distributions, with *"Residual Sugar"* and *"Chlorides"* showing especially heavy tails. Such characteristics indicate a concentration of data points far from the mean, which are typical markers of non-normality.

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se
fixed.acidity	1	1599	8.32	1.74	7.90	8.15	1.48	4.60	15.90	11.30	0.98	1.12	0.04
volatile.acidity	2	1599	0.53	0.18	0.52	0.52	0.18	0.12	1.58	1.46	0.67	1.21	0.00
citric.acid	3	1599	0.27	0.19	0.26	0.26	0.25	0.00	1.00	1.00	0.32	-0.79	0.00
residual.sugar	4	1599	2.54	1.41	2.20	2.26	0.44	0.90	15.50	14.60	4.53	28.49	0.04
chlorides	5	1599	0.09	0.05	0.08	0.08	0.01	0.01	0.61	0.60	5.67	41.53	0.00
free.sulfur.dioxide	6	1599	15.87	10.46	14.00	14.58	10.38	1.00	72.00	71.00	1.25	2.01	0.26
total.sulfur.dioxide	7	1599	46.47	32.90	38.00	41.84	26.69	6.00	289.00	283.00	1.51	3.79	0.82
density	8	1599	1.00	0.00	1.00	1.00	0.00	0.99	1.00	0.01	0.07	0.92	0.00
pH	9	1599	3.31	0.15	3.31	3.31	0.15	2.74	4.01	1.27	0.19	0.80	0.00
sulphates	10	1599	0.66	0.17	0.62	0.64	0.12	0.33	2.00	1.67	2.42	11.66	0.00
alcohol	11	1599	10.42	1.07	10.20	10.31	1.04	8.40	14.90	6.50	0.86	0.19	0.03
quality	12	1599	5.64	0.81	6.00	5.59	1.48	3.00	8.00	5.00	0.22	0.29	0.02

Figure 2.1: Wine descriptive statistics.

The extremely lowest *p-values* from the Shapiro-Wilk test (Tab. 2.1) and the visual evidence from the QQ plots in Fig. 2.2, suggest that the data do not follow a normal distribution, which implies that the variables exhibit significant levels of skewness and/or kurtosis.

Variable	**p-value**
Fixed acidity	$<$ 2.2e-16
Volatile acidity	2.693e-16
Citric acid	$<$ 2.2e-16
Residual sugar	$<$ 2.2e-16
Chlorides	$<$ 2.2e-16
Free sulfur dioxide	$<$ 2.2e-16
Total sulfur dioxide	$<$ 2.2e-16
pH	1.712e-06
Sulphates	$<$ 2.2e-16
Alcohol	$<$ 2.2e-16

Table 2.1: Wine Shapiro-Wilk results

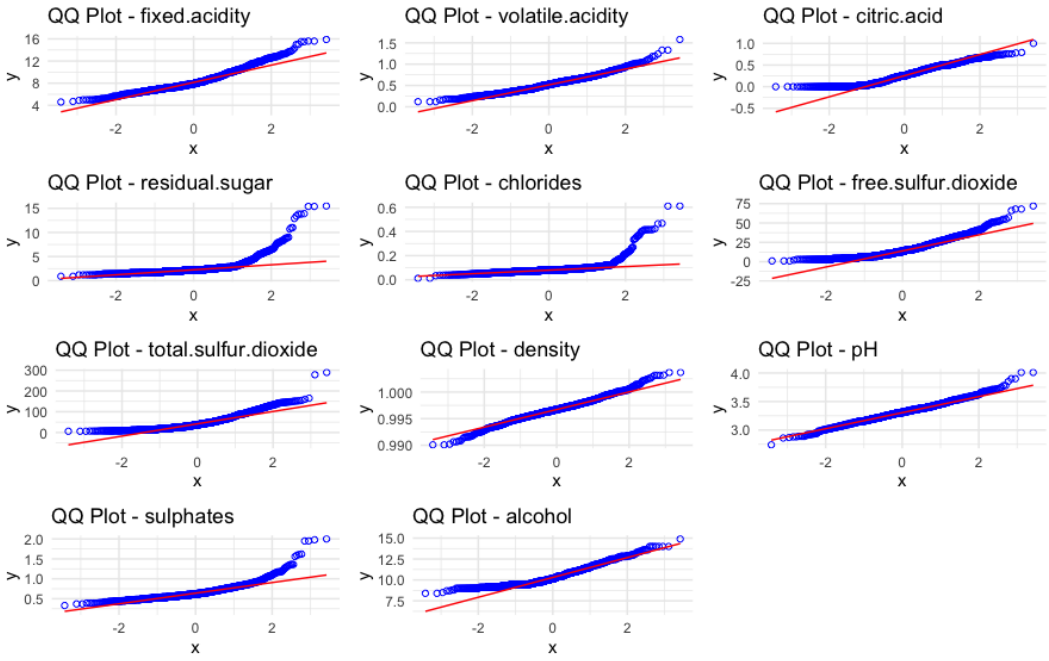

Figure 2.2: Wine QQ-plots.

In addition, this section also includes different types of plots to visualize the data. Beginning with a detailed histogram (Fig. 2.3) showcasing the distribution of quality levels, we proceed to reveal the counts associated with each rating, showing in Tab. 2.2.

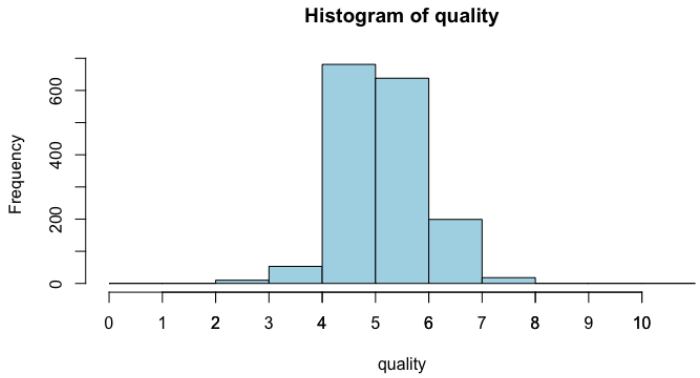

Figure 2.3: Histogram of quality.

Value	Count
3	10
4	53
5	681
6	638
7	199
8	18

Table 2.2: Distribution of wine quality

The distribution of wine quality in the dataset suggests a potential issue with class imbalance, where lower and higher quality ratings are under represented. This imbalance may negatively impact the performance and reliability of predictive models. Additionally, the absence of certain quality levels (1,2,10) necessitates the evaluation of techniques for dealing with missing or unobserved data. Addressing these issues in crucial for guaranteeing the robustness and generalizability of subsequent analyses and predictive modeling in wine quality assessment.

2.2.2 Features correlation

Generating plots of each features against the target variables allow us to visually assess the strength and direction of the linear relationship between each feature and the wine quality.

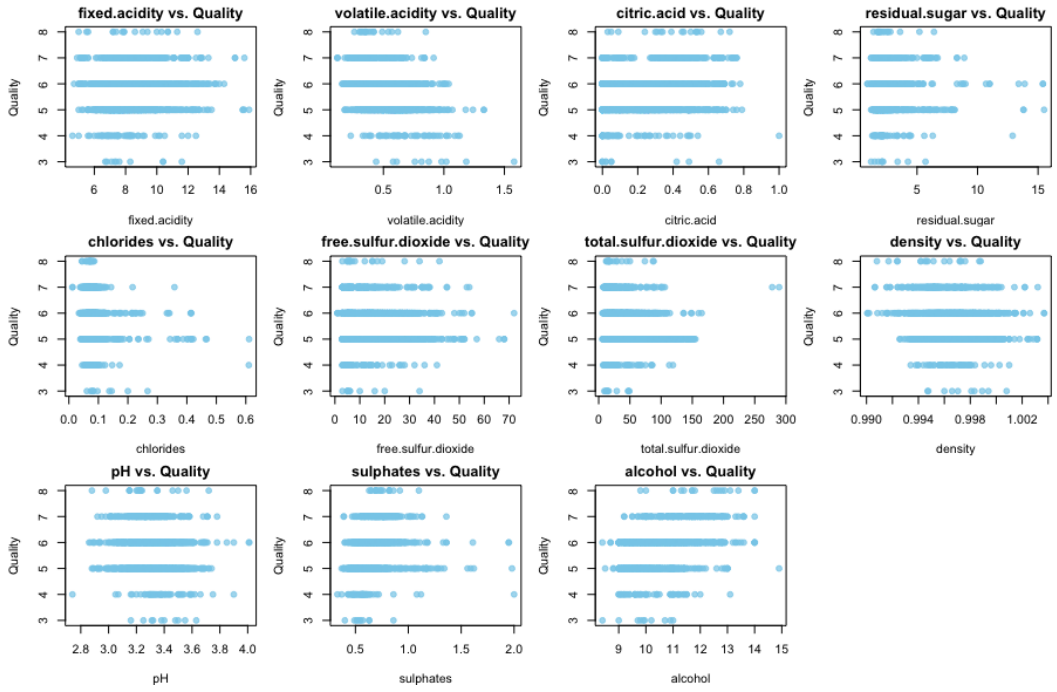

Figure 2.4: Correlation features vs Quality.

By considering the first plot in Fig 2.4, while no explicit linear dependencies are readily noticeable, there seems to be a nuanced positive linear correlation between *"quality"* and *"alcohol"*. Additionally a slight negative linear association exists between *"quality"* and *"volatile acidity"*.

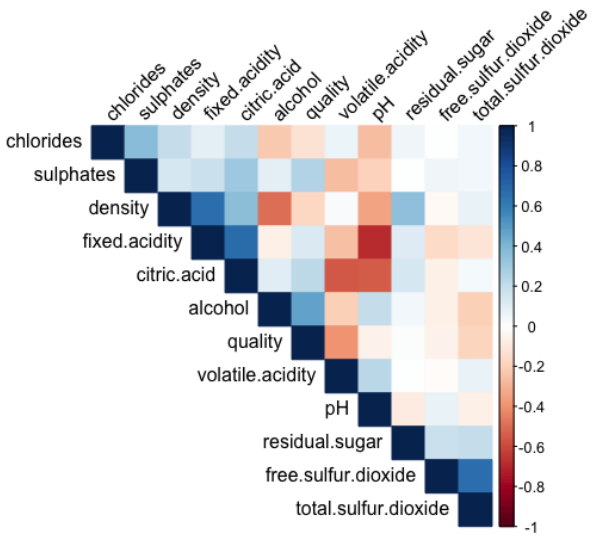

Figure 2.5: Wine correlation matrix.

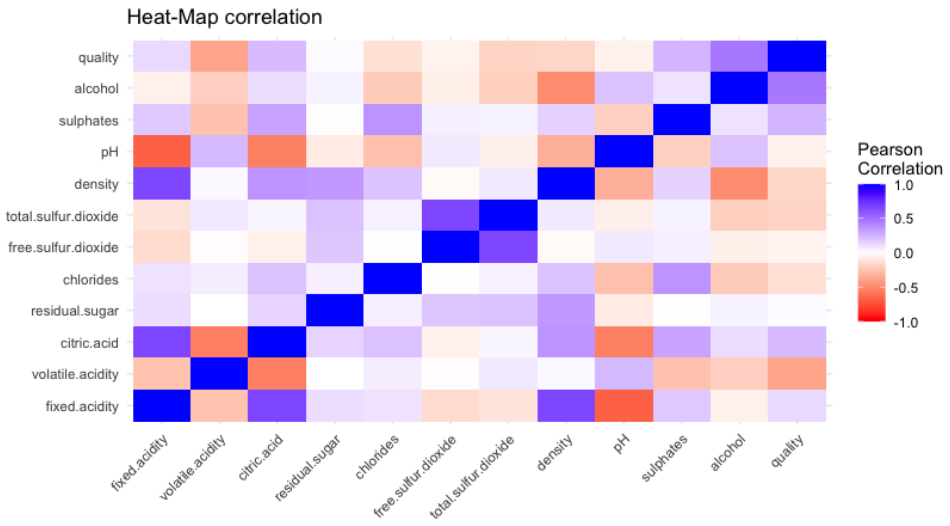

Figure 2.6: Wine heat-map correlation.

In order to be sure of our assumption, above in both Fig. 2.5 and Fig. 2.6 we can see the correlation results of the dataframe. *"Alcohol"* has the highest correlation to *"quality"* (**0.48**). Based on this information, we can say that wines with higher levels of alcohol may also have higher quality scores. In opposite, the strongest inverse correlation to *"quality"* is *"volatile.acidity"* (**-0.39**).Based on the knowledge, this relationship should make sense, as a higher level of volatile acetic acid can create a wine with an unpleasant vinegar taste. The other feature displaying a slightly high correlation with *"quality"* is *"sulphates"* (**0.25**).

It should also be noted that there are relatively strong correlations between several other features, such as:

- *Citric Acid - Fixed Acidity*: **0.67**;
- *Density - Fixed Acidity*: **0.66**;
- *Total Sulfur Dioxide - Free Sulfur Dioxide*: **0.66**.

But, given that our goal is to predict the quality features, we must consider only the correlations with *"quality"*.

In conclusion, considering the insights from these graphs, engaging in feature selection could prove valuable if our model does not achieve optimal performance. This is particularly relevant because, as indicated in the earlier plots, there is no prominent linear relationship between individual features and quality. Therefore, identifying and retaining only the most influential features could improve our model's predictive capability.

2.3 Preprocessing

2.3.1 Outliers detection and treatment

In the data cleaning and preparation process, special attention has been paid to the detection and handling of outliers. Outliers are values that deviate significantly from the rest of the data and can adversely affect the analysis, leading to misleading results.

Initially, Boxplots (Fig. 2.7) are employed to graphically display the data distribution,this provides an immediate visual representation of data points that might be considered anomalous.

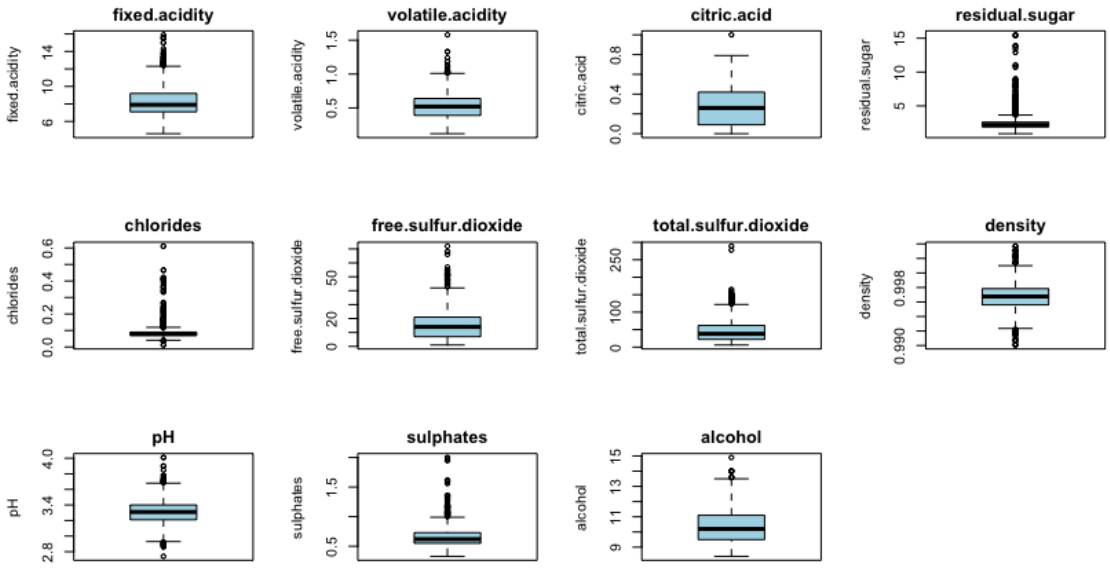

Figure 2.7: Wine outliers boxplots.

Following the initial visualization, two statistical methods are applied to more precisely identify outliers:

- IQR Method;
- Z-score Method.

IQR method

The IQR is calculated for each variable and defined outliers as those values that lay below the first quartile minus 1.5 times the IQR or above the third quartile plus 1.5 times the IQR.

Description	Count	%
Total Rows with Outliers	405	25.33
Total Rows in Dataset	1599	100

Table 2.3: Outlier Detection Results Using IQR Method

The findings, shown in Tab. 2.3, suggest that a significant portion of the dataset contains values that are considered outliers based on the IQR criterion. The fact that over a quarter of the rows contain at least one outlier suggests that the dataset may have a wide variance or a number of exceptional values that deviate from the typical range of the data. The presence of such a high percentage of outliers could have implications for any subsequent data analysis.

Z-score method

Z-Score is computed for each observation, which measures how many standard deviations a data point is from the mean. Outliers were identified as those data points whose absolute Z-Score exceeded a threshold, typically set at 3 or -3, indicating that the data point significantly deviates from the mean of the population.

Description	Count	%
Total Rows with Outliers	141	8.81
Total Rows in Dataset	1599	100

Table 2.4: Outlier Detection Results Using Z-score Method

From the Table 2.4, one can immediately notice the significant difference in the number of outliers identified with Z-score.

Methods comparison

As observed from the descriptive statistics of the dataset, it presents a variety of distribution across its variables. When applying the Z-score method for outlier detection, the number of outliers identified was relatively low. This can be attributed to the assumption inherent in the Z-score methodology that the data follows a normal distribution. The Z-score calculates the number of standard deviations a data point is from the mean; however, for distributions that are not normal, particularly those with heavy tails or skewness, the mean and standard deviation are not adequate measures of central tendency and dispersion. Consequently, the Z-score method may not capture the true extent of the deviation of these points from the core data set.

In contrast, the IQR method identified a larger number of outliers. This discrepancy arises because the IQR method is based on quartiles, which are less affected by non-normal distributions and extreme values. The IQR method considers data points as outliers if they lie beyond 1.5 times the IQR below the first quartile or above the third quartile, making it more suitable for skewed data.

2.3.2 Feature Selection

Feature selection is a critical process in the field of machine learning, essential for enhancing the effectiveness and efficiency of predictive models. This procedure involves identifying the most influential features within a dataset to reduce the risk of overfitting, improve model performance, and simplify the interpretation of results. Feature selection becomes particularly important in contexts with a high number of variables, as is often the case in complex datasets.

In this regression task, feature selection was applied using a Random Forest model (provided by package *'randomForest'*) to determine the importance of each feature in the dataset. The target variable, defined as *"quality"*, was placed at the center of the model's predictions. Features were selected by excluding the target variable from the dataset. Subsequently, a Random Forest model was trained with the *"importance = TRUE"* option to calculate the importance of each feature, using 500 decision trees.

```
#preparing variables
target_variable <- 'quality'
features <- setdiff(names(wine_dataset_clean), target_variable)

#train random forest model
rf_model <- randomForest(x = wine_dataset_clean[features],
                         y = wine_dataset_clean[[target_variable]],
                         importance = TRUE,
                         ntree = 500)
```

After training, the importance of the features was assessed based on the average square error increase (**%IncMSE**) indicated by the model. This method allowed for the quantification of the impact of each feature on the model's performance.

```
#Feature importance >MSE >importance
feature_importance <- rf_model$importance[, '%IncMSE']

#Data fram for feature importance
importance_df <- data.frame(Feature = names(feature_importance),
Importance = feature_importance)
```

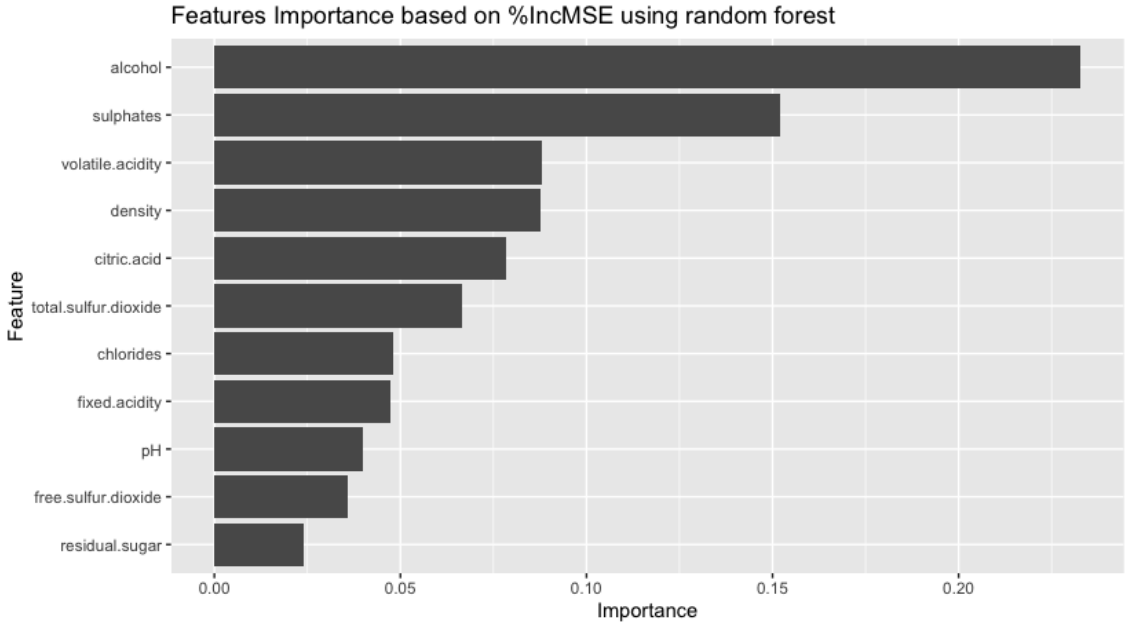

Figure 2.8: Wine features importance.

The bar chart in Fig. 2.8 visualizes the result of the analysis conducted. The features are ranked in descending order of importance. These findings suggest that *"alcohol"*, *"sulphates"*, and *"volatile acidity"* are the most critical factors in predicting the quality of wine within this dataset, and therefore might be the primary focus for further model refinement or domain analysis. The less important features could potentially be omitted from the model to reduce complexity without significantly affecting model performance.

In conclusion, a subset of the most 8 significant features was chosen based on the importance rankings obtained from the wrapping technique. Through this process, it was possible to identify and focus on the most significant features, optimizing the Random Forest model for the project and ensuring more accurate and reliable results.

```
#Selecting features
features_number <- 8
selected_features <- names(sort(feature_importance,
decreasing = TRUE)[1:features_number])

#Dataset with no outliers and selected features
wine_dataset_noOutliers_selected <- wine_dataset_clean
[c(selected_features, target_variable)]
wine_dataset_noOutliers_selected
```

2.3.3 Data splitting and normalization

The data is divided into training and test sets to evaluate the models performance. The package *'caret'* provides a function that helps in creating balanced splits of the data based on the *"quality"* variable:

- **train_data**: for training the models; represents the 80% of the data.
- **test_data**: for testing the models; represents the remaining 20% of the data.

Normalization is implemented using the *'preProcess'* function from the *'caret'* packages, with the methods "center" (to subtract the mean) and "scale" (to divide by the standard deviation). This is important because features in your dataset might be measured in different scales, and not normalizing them can lead a model to improperly weigh the features. Normalization ensures each feature contributes equally to the model, improving the training process and the performance of the model.

```
#Data normalization
preproc <- preProcess(train_data[, -ncol(train_data)],
method = c("center", "scale"))
train_norm <- predict(preproc, train_data[, -ncol(train_data)])
test_norm <- predict(preproc, test_data[, -ncol(test_data)])
```

2.4 K-Nearest Neighbors Regression

2.4.1 Model's implementation

After completing the data preparation and feature selection, we proceed with the actual training of the K-Nearest Neighbors (KNN) regression model. The KNN model is a supervised learning algorithm that finds a group of samples (the nearest neighbors) in the training set and makes predictions based on their mean or median. In the implementation presented, the *'caret'* package in R is used, which provides extended functions for training and validating machine learning models.

```
#KNN Regression with Hyperparameter Tuning (k)
train_control <- trainControl(
    method = "repeatedcv",
    number = 10,
    repeats = 10)
set.seed(123)
knn_fit <- train(
    quality ~ .,
    data = train_norm,
    method = 'knn',
    tuneLength = 10,
    trControl = train_control
)
```

The training of the KNN model begins with defining a training control, where the method of repeated cross-validation is specified. In this case, a 10-fold cross-validation is performed 10 times. This used to guide the cross-validation and parameter tuning process capabilities. The training of the KNN model follows, indicating the intention to predict the *"quality"* variable, based on different values of k.

2.4.2 Model's performance evaluation

The tuning of the **k** value in KNN regression is a crucial step that directly impacts the model's predictive accuracy. The graph provided in Fig. 2.9 illustrates the relationship between different different k values and the model's performance as measured by the Root Mean Square Error (RMSE). A lower RMSE indicates a better fit to the data, as it means the model's predictions are closer to the actual values.

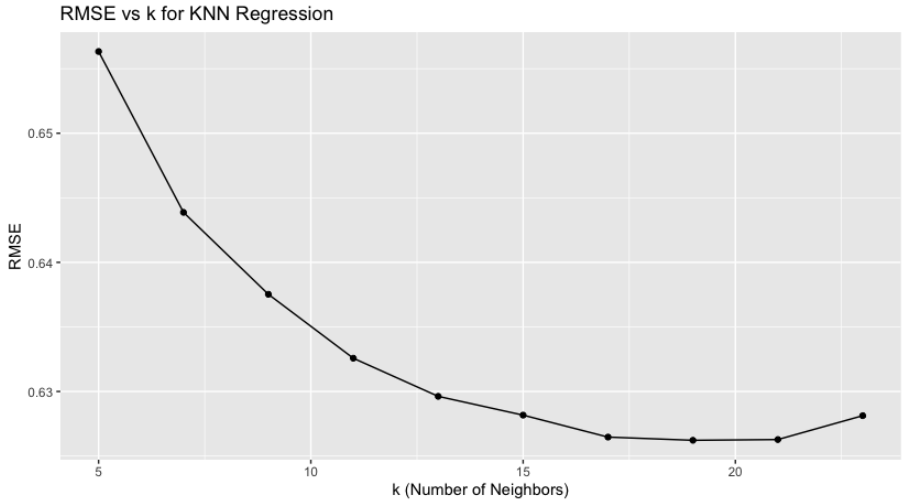

Figure 2.9: RMSE vs k in tuning process.

The process of tuning involves running the KNN algorithm multiple times, each time with a different number of neighbors (k). This is done within a cross-validation framework to ensure that the model's performance is robust and not just tailored to a specific subset of the data. To select the optimal k, we look for the point where increasing k no longer results in substantial improvements in RMSE. At $\mathbf{k = 19}$, as shown in Tab 2.9, we have the lowest RMSE before it slightly increases or remains constant as k grows further, this is the optimal value.

k	**RMSE**
19	0.63

Table 2.5: Best KNN parameter

2.5 Ridge Regression

2.5.1 Model's implementation

The same procedure, with the identical cross-validation approach, is applied for another model: the Ridge Regression. A regularization parameter lambda (λ), also known as the ridge penalty, is introduced to prevent overfitting, enhance the model's generalization capabilities, and handle multicollinearity. In the provided code snippet, the function from the *'caret'* package in R is employed to train a Ridge Regression model.

```
#RIDGE Regression with Hyperparameter Tuning (lambda)
ridge_fit <- train(
    quality ~ .,
    data = train_norm,
    method = 'ridge',
    tuneLength = 10,
    trControl = train_control
)
```

This process helps in selecting the lambda that leads to the best model performance, balancing the trade-off between bias and variance.

2.5.2 Model's performance evaluation

The process of tuning involves running the Ridge Regression algorithm multiple times, each time with a different lambda value (λ). The graph in Fig. 2.10 provided illustrates how the choice of λ influences the model's performance in terms of the Root Mean Square Error (RMSE). The aim is to find a value that minimizes RMSE, indicating a model that is well-fitted to the data but also generalizes well to unseen data.

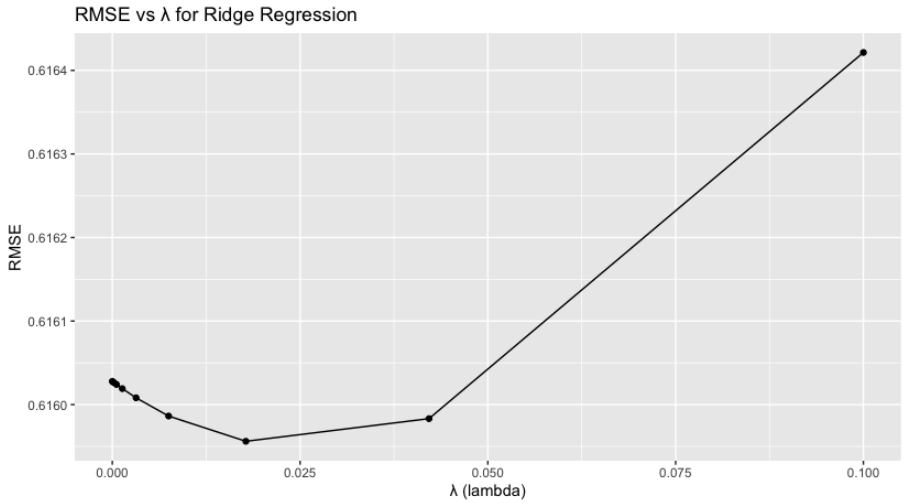

Figure 2.10: RMSE vs λ in tuning process.

From the graph and cross-validation process, we can see that RMSE reaches its minimum around $\lambda = \mathbf{0.02}$. Beyond this value, the RMSE starts to increase, which suggests that the penalty is too strong and the model is starting to underfit the data. This value is chosen as the optimal model because it has the lowest RMSE (Tab. 2.6), striking a balance between bias (underfitting) and variance (overfitting).

λ	**RMSE**
0.02	0.62

Table 2.6: Best Ridge Regression parameter

2.6 Conclusions

In this research, the aim was to predict wine quality, expressed as a discrete variable with scores ranging from 1 to 10. The performance was evaluated using three key metrics:

- **Mean Absolute Error(MAE)**: it gives an idea of how wrong the predictions were; the lower the MAE, the better model's performance.
- **Root Mean Error(RMSE)**: it provides a measure of how well the model's predictions are, but gives a relatively high weight to large errors.
- **R-squared(R^2)**: is a statistical measure of how close the data are to the fitted regression line. It's interpreted as the proportion of the response variable's variation that is explained by a linear model; for a perfect model, it would be 1.

To assess the effectiveness of the employed machine learning models, we used a baseline model as a point of reference. This simple model, which makes predictions based on the average values, serves as a basic comparator. The obtained values are shown in Tab. 2.7, indicating limited predictive capacity.

Metric	**Value**
MAE	0.6283
RMSE	0.7165
R^2	-0.00053

Table 2.7: Baseline Model Performance Metrics

In comparison, the K-Nearest Neighbors (KNN) model (Tab. 2.8)and the Ridge Regression model (Tab. 2.9)showed significant improvements. These results indicate a higher accuracy in predictions compared to the baseline model, with lower average and quadratic errors, and a higher proportion of variance explained. The Ridge Regression model was even more performant: it not only further reduces prediction errors but also explains a larger percentage of the variance in the dataset compared to the other models.

Metric	**KNN Model**
MAE	0.4730
RMSE	0.5685
R^2	0.3729

Table 2.8: KNN Model Performance Metrics

Metric	**Ridge Regression**
MAE	0.4370
RMSE	0.5499
R^2	0.4109

Table 2.9: Ridge Regression Model Performance Metrics

In summary, the comparison of the models shows that both KNN and Ridge Regression significantly outperform the baseline model in predicting wine quality. Among these, Ridge Regression stands out as the most effective model, offering the most accurate and reliable predictions.