A · P · U

ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

Machine Learning Models for Heart Disease Prediction: A Comparative Analysis

Module Name	**:**	Applied Machine Learning
Intake Code	**:**	APDMF2310AI
Lecturer Name	**:**	Prof. Dr. Mandava Rajeswari
Hand in Date	**:**	1st of March, 2024
Student Name	**:**	Nurlan Amanov
Student Number	**:**	TP077526

Table of Contents

ABSTRACT

This study explores the performance of three machine learning algorithms – Logistic Regression, Decision Tree, Random Forest – in predicting heart disease based on dataset that was taken from Kaggle.com with 918 observation and 12 attributes. To conduct a better analysis study provides information about related works in the same field with 5 studies about usage of machine learning algorithms in heart disease prediction. Models use 5 metrics for performance evaluation: accuracy, precision, recall, F1 score and Area Under the Curve (AUC). This study aims to create the best model that predicts the most likely progression of heart disease. For model implementation R programming language has been used, that provides many various libraries that make the work accurate and precise.

1. INTRODUCTION

1.1. Overview

According to the World Health Organization (2021), cardiovascular disease (CVD) is the leading cause of death in the world. Statistics show that about 17.9 million people die each year from this disease, which is 31% of all deaths. So, what is it about CVD that claims so many lives? Cardiovascular Disease is a group of heart and blood vessel diseases that includes rheumatic heart disease, coronary heart disease, and other conditions. Taking a closer look, heart attacks and strokes account for four out of five cases, of which a third occur prematurely in people under the age of 70. The most important risk factors for developing the disease are unhealthy lifestyles: smoking, alcohol, low physical activity, and unhealthy diet. The effects of such activities manifest themselves in increased blood pressure, blood glucose levels, blood lipids and excess weight. Doing the opposite of what can cause disease reduces the risks of disease, i.e. quitting tobacco and alcohol, eating fruit and vegetables and regular physical activity reduces the risks. Identifying people who are most at risk and providing premature treatment can prevent fatalities, leading to the need for a machine that can identify the highest risk patients.

1.2. Context and Dataset

Machine learning is the best option when it comes to predicting something. The use of machine learning machines for pre-prediction is already practiced in modern medicine. Thus, for prediction using machine learning, this study will use the "Heart Failure Prediction Dataset" dataset taken from the website Kaggle.com. This dataset consists of 918 observations including 11 features.

1.3. Aim and Objectives

The main aim of the research is to create a machine learning model that will be able to predict the development of heart disease with the highest probability. The objectives of this research include:

1. Conducting Exploratory Data Analysis (EDA) to understand the characteristics of the dataset and identify the main features that are related to heart disease research.

2. Developing three machine learning models.
3. Analyzing the work carried out and comparing the results.

2. RELATED WORKS

With the development of Artificial Intelligence (AI), especially machine learning, the development in the field of medicine has taken a high speed. Thanks to such machines, the field of medical research has gained a huge amount of new work that uses different data sets to perform analyses, using different algorithms and processing the data to obtain the best models to determine the disease of interest.

2.1. Cardiovascular Health

As one of the most dangerous diseases, heart disease has become a problem for entire nations. Being a disease that is directly related to people's lifestyle American Heart Association (AHA) issued a report (Tsao et al., 2022) that aims to improve Cardiovascular Health (CVH) in the whole population by 20%. Along with this, the concept of CVH was introduced with 7 characteristics including: diet quality, Pulmonary Atresia (PA), smoking, blood cholesterol, Body Mass Index (BMI), Blood Pressure (BP), blood glucose. Each of the seven characteristics can be described by one of three words: poor, intermediate, and ideal. The importance of an ideal CVH has been proven many times by various studies and today many people are trying to instill healthy habits in people, so a meta-analysis of 12878 people showed that the presence of ideal components or the highest presence of ideal components is associated with a lower risk of mortality. And looking at studies that used the CVH scoring system, it showed that a one-point increase in CVH scores reduced mortality risks by 8 per cent. The same paper provided a table for Years of Life Lost (YLL). The data was provided from 1990 to 2019. The top 5 risk factors for YLL were: smoking, high SBP, high BMI, high FPG, and drug use. Moreover, drug use showed a significant increase, rising from 18th to 5th place. This only shows the impact of traditional risk factors, but also the emergence of new problems that threaten human health.

2.2. Machine Learning for Heart Disease Prediction

The trend of using datasets related to the use of medical data is one of the popular ones, and one can find a huge number of papers related to analyzing heart disease prediction.

The work of Ali et al (2021) aims at finding classifiers with highest accuracy for heart disease diagnosis. The dataset, which was taken from Kaggle.com, contained 1025 patient records with 14 attributes. Linear regression, k-nearest neighbor, decision tree and random forest were used, and the algorithms were compared for performance and accuracy. The results showed that k-nearest neighbor, decision tree and random forest performed 100%, achieving 100% accuracy, 100% sensitivity and 100% specificity. Linear regression was the worst method with an accuracy of about 89%, 82% sensitivity and 95% specificity.

Shah et al (2020) used the Cleveland database of UCI repository of heart disease patients with 303 instances and 76 attributes, but only 14 of them were used. The main aim was to look at the odds of developing heart disease. Four algorithms were used for analysis: naïve bayes, k-nearest neighbor, decision tree and random forest. At the end of the study, the best accuracy was achieved by k-nearest neighbor with 90.789%.

The work of Sharma et al (2020) used UCI Heart disease prediction dataset with 14 attributes. Support vector machine, naïve bayes, decision tree and random forest were used. The study aimed to find correlation between attributes and use them for better prediction. The results showed that random forest had the best accuracy, with less time used for prediction than the others.

Prediction of heart disease from the UCI Repository with 76 attributes was used by Katarya et al. (2020), but only 14 of the 76 attributes were used. In this paper logistic regression, naïve bayes, support vector machine, k-nearest neighbor, decision tree, random forest was used. At the end of the study, the accuracy for each of the modules was derived where logistic regression accuracy is 93.40$, naive bayes is 90.10%, support vector machine is 92.30%, k-nearest neighbor is 71.24%, decision tree is 81.31%, random forest 95.60%. As we can understand, random forest was the best and k-nearest neighbor was the worst.

The latest work that is related to heart disease prediction is Singh et al (2020) where UCI repository dataset was used. In this paper algorithms like support vector machine, decision tree, linear regression, k-nearest neighbor was used. The best model was k-nearest neighbor with 87% accuracy and the worst was linear regression with 78% accuracy.

2.3. Supervised Machine Learning Algorithms

Many different algorithms were used in the various studies reviewed, but mostly the same algorithms were used and are summarized below:

2.3.1. Support Vector Machine

According to Wikipedia, Support Vector Machine (SVM) is a machine learning classification technique that is used to analyze data and detect patterns in classification and regression analysis. SVM is usually used when data is characterized as a two-class problem and the higher the separation between the two classes, the better the model itself. The actual basis of SVM is the mathematical methods used to solve complex problems. SVM was used in Sharma et al. (2020), Katarya et al. (2020) and Singh et al. (2020) and showed good results, although it was not the best model but not the worst either.

2.3.2. K-Nearest Neighbor

K-Nearest Neighbor (K-NN) is one of the oldest algorithms based on statistical learning methods. Being one of the oldest, its essence lies in its simplicity. How? Objects are classified based on their proximity to other data in the set, so in K-Nearest, the letter "K" stands for the number of nearest neighbors. The number of nearest neighbors is either given or inferred. When a new sample comes in, it is compared with every existing sample, and the resulting predictions are plotted against the distance between neighbors. The advantages are its simplicity and workability. With these advantages, K-NN is well suited for disease prediction. It is worth noting that one of the features of K-NN is that it can fill in missing values. K-NN was used in four out of the five papers reviewed and performed well except Katarya et al. (2020) where the algorithm fared the worst.

2.3.3. Random Forest

Random Forests (RF) are a powerful method in the field of machine learning, the foundation of which is Decision Tree (DT). Its job is to create an ensemble or "forest". Each tree in the forest predicts class labels during the testing period, then a vote is taken to decide which class label is the most appropriate and thus this process is repeated for each data point in the collection. It is worth noting that the more trees there are, the more accurate the result is. However, with a large dataset and many trees, RF produces predictions slowly but is considered reliable. RF was used in four of the five papers reviewed and showed the best results in three of the studies.

2.3.4. Decision Tree

Decision Tree (DT) is one of the oldest algorithms. Decision Tree is popular for its simplicity and is widely used in medical diagnosis. The structure of the decision-making logic is like a tree structure, where the structure has several levels, and the whole structure is created using three key nodes:

- Root Node - determines the function of all other nodes.
- Interior Node - manage various attributes.
- Leaf Node - provides the test results.

The results that are obtained from DT are easy to interpret, which increases their accuracy. However, DT performed poorly in the reviewed papers - in three out of five papers it was the worst model. However, it turned out to be the best model on par with K-NN and RF.

2.3.5. Logistic Regression

Logistic Regression (LR) is a simple, interpretable model and a robust algorithm. LR gives probability a value between 0 and 1 and to facilitate the process of dividing them into classes it uses thresholds. So, when dividing into two classes, a threshold can be set to "greater than 0.5" for one class and values below that in the other. It is worth noting that LR uses the principles of general regression modelling, thus considering the probability of having or not having a particular instance. LR was used in three papers and the five papers reviewed where it performed the worst, in two of the three papers.

2.3.6. Naive Bayes

Naive Bayes is an algorithm known for its simplicity and efficiency. The algorithm works based on Bayes' theorem. Bayes theorem is a mathematical concept that facilitates the calculation of probability. If it is now clear where "Bayes" comes from in the name of the algorithm, the term "Naive" speaks of attribute independence, which means the predictors are not related to each other. The algorithm was used in two of their five studies we reviewed and showed excellent results. Thus, each unique characteristic can contribute to the result. The formula itself looks as follows:

$$\boxed{P(A|B) = \frac{P(B|A)P(A)}{P(B)}}$$

Figure 1. Bayes' Theorem

Table 1. Literature Matrix

Citation	Dataset	Pre-processing	Algorithms	Results
(Ali et al., 2021)	Heart Disease Dataset from Kaggle with 1025 patient records and 14 attributes	- Missing Values - Outlier Detection using Interquartile Range - Balancing the Imbalanced Dataset using SMOTE - Exploratory Data Analysis (EDA)	- Logistic Regressions (LR) - K-Nearest Neighbor (K-NN) - Decision Tree (DT) - Random Forest (RF)	Best Models: - K-NN, DT, RF: Sensitivity: 1.000 Specificity: 1.000 Accuracy: 100.000 Precision: 1.000 Recall: 1.000 F-Measure: 1.000 Worst Model: - LR: Sensitivity: 0.820 Specificity: 0.950 Accuracy: 89.627 Precision: 0.896 Recall: 0.896 F-Measure: 0.896
(Shah et al., 2020)	The Cleveland database of UCI repository of heart disease patients with 303 instances and 76 attributes	- Data Cleaning (noise and missing values) - Transformation (smoothing, normalization, aggregation) - Integration	- Naïve Bayes (NB) - K-Nearest Neighbor (K-NN) - Decision Tree (DT)	Accuracy: - NB: 88.157% - K-NN: 90.789% - DT: 80.263% - RF: 86.84%

	(only 14 considered for testing)	- Reduction (data is complex, need to be formatted) - Feature Selection	- Random Forest (RF)	
(Sharma et al., 2020)	The Cleveland Heart Disease Dataset with 14 attributes considered	- Replacing missing values with mean - Data Transformation (from numeric to nominal)	- Support Vector Machine (SVM) - Random Forest (RF) - Decision Tree (DT) - Naïve Bayes (NB)	Best Model: - RF: Accuracy: 99% Precision: 0.997 Recall: 0.997 ROC Area: 1.00 Worst Model: - DT: Accuracy: 85% Precision: 0.851 Recall: 0.848 ROC Area: 0.889
(Katarya et al., 2020)	Prediction of heart disease from the UCI Repository with 76 attributes (only 14 chosen)	- Feature Selection - Replacing missing values (using Python library) - Data Transformation	- Logistic Regression (LS) - Naïve Bayes (NB) - Support Vector Machine (SVM) - K-Nearest Neighbor (K-NN) - Decision Tree (DT) - Random Forest (RF)	Accuracy: - LR: 93.40% - NB: 90.10% - SVM: 92.30% - K-NN: 71.42% - DT: 81.31% - RF: 95.60%
(Singh et al., 2020)	UCI repository dataset which is	- Handling null values	- Support Vector Machine (SVM)	Accuracy: - SVM: 83%

	well verified by number of researchers and authority of the UCI		- Decision Tree (DT) - Linear Regression (LR) - K-Nearest Neighbor (K-NN)	- DT: 79% - LR: 78% - K-NN: 87%

After parsing each of the studies and gaining information about the algorithms, it can be concluded that each algorithm has its own strengths and weaknesses. Using the results and knowledge gained, this research will use 3 supervised machine learning algorithms: Linear Regression (LR), Decision Tree (DT) and Random Forest (RF). RF is considered a powerful method and has shown some of the best results out of all the models, it is also great for heart disease prediction. DT, although it did not perform the best, is considered easy to use and is widely used in the interest of this study. LR also did not perform well in other studies, however LR is a simple and interpretable model, which is suitable for binary classification tasks.

3. METHODS

3.1. Dataset Description

The analyses will be conducted in R programming language. The name of dataset, that is going to be used in this document, is “Heart Failure Prediction Dataset” from Kaggle.com. The dataset can be accessed via [Heart Failure Prediction Dataset (kaggle.com)]. Several datasets that were previously available separately but had never been combined were combined to create this new dataset. This dataset is the largest heart disease dataset currently available for research purposes because it combines five different heart datasets with eleven common features. The five datasets that were curated for it are:

- Cleveland: 303 observations
- Hungarian: 294 observations
- Switzerland: 123 observations
- Long Beach VA: 200 observations
- Stalog (Heart) Data Set: 270 observations

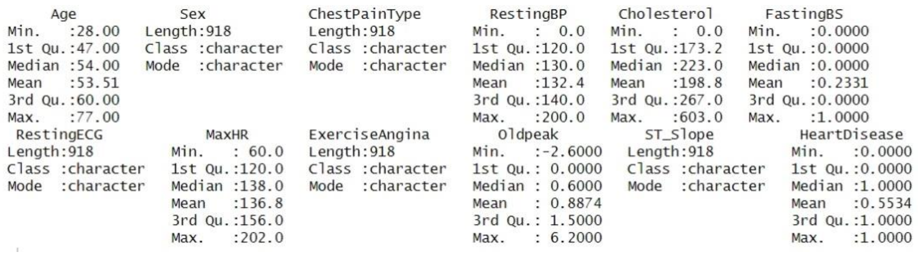
From the total value of 1190 observations, 272 observations got removed due to being duplicated. The final dataset consists of 918 observations with 12 attributes. Attribute information:

Table 2. Attribute Information

Attribute	Data Type	Description
Age	Numeric	Age of patient
Sex	Categorical	Male of female
ChestPainType	Categorical	TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic
RestingBP	Numeric	resting blood pressure [mm Hg]
Cholesterol	Numeric	serum cholesterol [mm/dl]

FastingBS	Numeric	1: if FastingBS > 120 mg/dl, 0: otherwise
RestingECG	Categorical	Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria
MaxHR	Numeric	Numeric value between 60 and 202
ExerciseAngina	Categorical	Y: Yes, N: No
Oldpeak	Numeric	Numeric value measured in depression
ST_Slope	Categorical	Up: upsloping, Flat: flat, Down: downsloping
HeartDisease	Categorical	1: heart disease, 0: Normal

Or the summary of the dataset given by R (Figure 2).



```
      Age             Sex            ChestPainType         RestingBP      Cholesterol       FastingBS     
 Min.   :28.00   Length:918         Length:918         Min.   :  0.0   Min.   :  0.0   Min.   :0.0000  
 1st Qu.:47.00   Class :character   Class :character   1st Qu.:120.0   1st Qu.:173.2   1st Qu.:0.0000  
 Median :54.00   Mode  :character   Mode  :character   Median :130.0   Median :223.0   Median :0.0000  
 Mean   :53.51                                         Mean   :132.4   Mean   :198.8   Mean   :0.2331  
 3rd Qu.:60.00                                         3rd Qu.:140.0   3rd Qu.:267.0   3rd Qu.:0.0000  
 Max.   :77.00                                         Max.   :200.0   Max.   :603.0   Max.   :1.0000  
  RestingECG            MaxHR       ExerciseAngina        Oldpeak          ST_Slope          HeartDisease   
 Length:918         Min.   : 60.0   Length:918         Min.   :-2.6000   Length:918         Min.   :0.0000  
 Class :character   1st Qu.:120.0   Class :character   1st Qu.: 0.0000   Class :character   1st Qu.:0.0000  
 Mode  :character   Median :138.0   Mode  :character   Median : 0.6000   Mode  :character   Median :1.0000  
                    Mean   :136.8                      Mean   : 0.8874                      Mean   :0.5534  
                    3rd Qu.:156.0                      3rd Qu.: 1.5000                      3rd Qu.:1.0000  
                    Max.   :202.0                      Max.   : 6.2000                      Max.   :1.0000  
```

Figure 2. Summary of the Dataset in R

3.2. Chosen Algorithms

Let's look at all three models that have been chosen in this paper and give reasons for their selection.

3.2.1. Logistic Regression

The main reason for choosing Logistic Regression (LR) is that it is well suited for our problem, i.e. binary classification problem. Given this fact and the fact that the target variable is binary value (with 1 and 0) LR is suitable for our work. LR is also easy to use and is suitable for heart disease prediction.

3.2.2. Random Forest

Random Forest (RF) in the studies that have been reviewed in this paper has shown excellent results and being a model that is a powerful method is excellent for this study.

3.2.3. Decision Tree

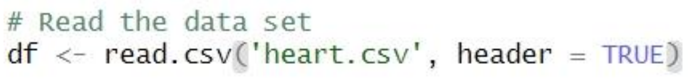
Decision Tree (DT) can easily analyze numerical and categorical data, making it an excellent choice for this work. DT is also used already in the field, which adds another reason for its use.

3.3. Metrics for Performance Evaluation

The analyses will be conducted in R programming language. There are 5 metrics that will be used in this paper:

1. Accuracy – overall correctness of model prediction.
2. Precision - proportion of true positive predictions among instances predicted as positive.
3. Recall - proportion of true positive predictions among actual positive instances.
4. F1 Score – harmonic mean of Precision and Recall
5. Area Under the ROC-Curve - measures the model's ability to distinguish between classes across various thresholds.

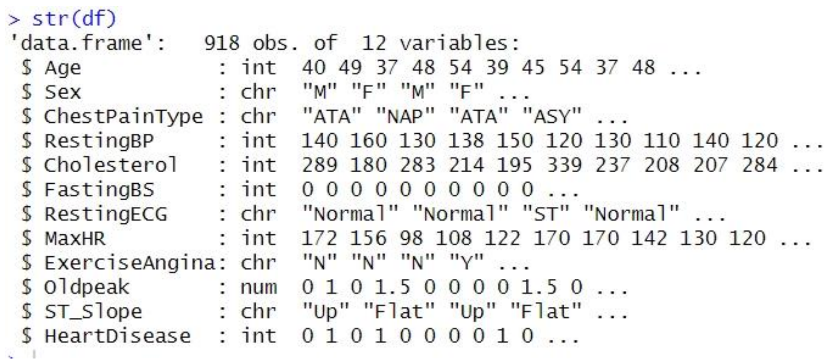
4. DATASET PREPARATION



```
# Read the data set
df <- read.csv('heart.csv', header = TRUE)
```

Figure 3. Read Dataset

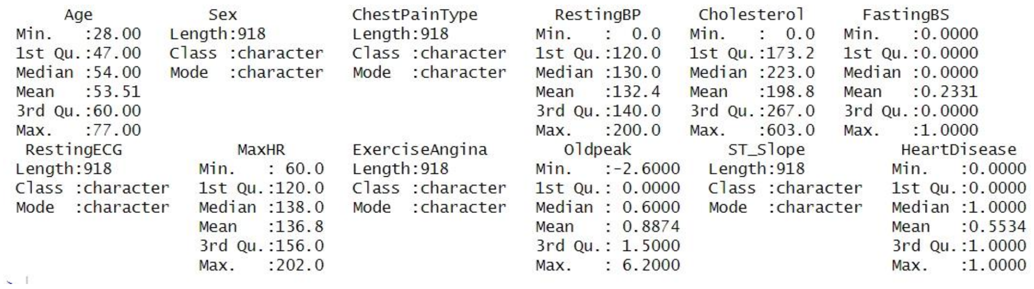
Figure 3 shows the code on how we read the dataset. With our dataset being loaded we can start working with it.



```
> str(df)
'data.frame':   918 obs. of  12 variables:
 $ Age           : int  40 49 37 48 54 39 45 54 37 48 ...
 $ Sex           : chr  "M" "F" "M" "F" ...
 $ ChestPainType : chr  "ATA" "NAP" "ATA" "ASY" ...
 $ RestingBP     : int  140 160 130 138 150 120 130 110 140 120 ...
 $ Cholesterol   : int  289 180 283 214 195 339 237 208 207 284 ...
 $ FastingBS     : int  0 0 0 0 0 0 0 0 0 0 ...
 $ RestingECG    : chr  "Normal" "Normal" "ST" "Normal" ...
 $ MaxHR         : int  172 156 98 108 122 170 170 142 130 120 ...
 $ ExerciseAngina: chr  "N" "N" "N" "Y" ...
 $ Oldpeak       : num  0 1 0 1.5 0 0 0 0 1.5 0 ...
 $ ST_Slope      : chr  "Up" "Flat" "Up" "Flat" ...
 $ HeartDisease  : int  0 1 0 1 0 0 0 0 1 0 ...
```

Figure 4. Structure of Dataset

With the code str(df) we can get the structure of our dataset (Figure 4). We can see that categorical valuables have chr datatype, that we need to change.



```
      Age            Sex            ChestPainType         RestingBP       Cholesterol       FastingBS     
 Min.   :28.00   Length:918         Length:918         Min.   :  0.0   Min.   :  0.0   Min.   :0.0000  
 1st Qu.:47.00   Class :character   Class :character   1st Qu.:120.0   1st Qu.:173.2   1st Qu.:0.0000  
 Median :54.00   Mode  :character   Mode  :character   Median :130.0   Median :223.0   Median :0.0000  
 Mean   :53.51                                         Mean   :132.4   Mean   :198.8   Mean   :0.2331  
 3rd Qu.:60.00                                         3rd Qu.:140.0   3rd Qu.:267.0   3rd Qu.:0.0000  
 Max.   :77.00                                         Max.   :200.0   Max.   :603.0   Max.   :1.0000  
  RestingECG            MaxHR       ExerciseAngina        Oldpeak          ST_Slope          HeartDisease   
 Length:918         Min.   : 60.0   Length:918         Min.   :-2.6000   Length:918         Min.   :0.0000  
 Class :character   1st Qu.:120.0   Class :character   1st Qu.: 0.0000   Class :character   1st Qu.:0.0000  
 Mode  :character   Median :138.0   Mode  :character   Median : 0.6000   Mode  :character   Median :1.0000  
                    Mean   :136.8                      Mean   : 0.8874                      Mean   :0.5534  
                    3rd Qu.:156.0                      3rd Qu.: 1.5000                      3rd Qu.:1.0000  
                    Max.   :202.0                      Max.   : 6.2000                      Max.   :1.0000  
```

Figure 5. Summary of Dataset

Figure 5 provides information into summary of dataset, where we can get information about minimum value, maximum value, mean value for integer values, and see those categorical values taken as character.

4.1. Exploratory Data Analysis

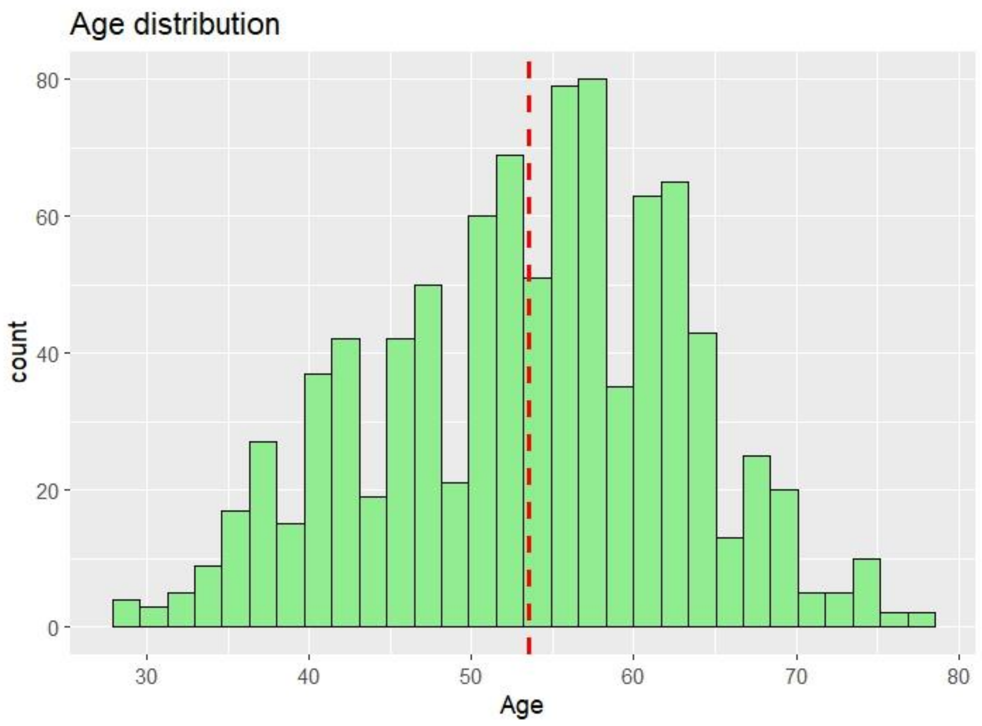


Figure 6. Age Distribution Plot

We can see that most people aged between 50 and 60 years old with mean value of age being 53.51 years old, according to Figure 5.

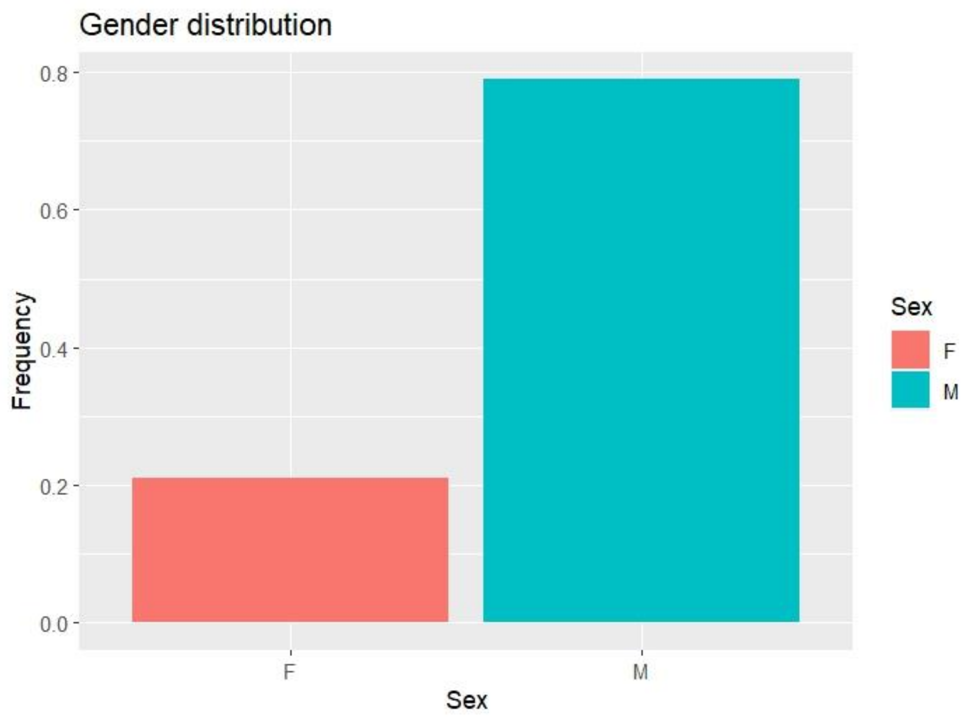


Figure 7. Gender Distribution

The graphic in Figure 7 provides information about gender distribution between patients. Most of the patient are male, which could give us outcome of that most of the people with heart disease are male, but due to low number of observations we cannot tell that surely.

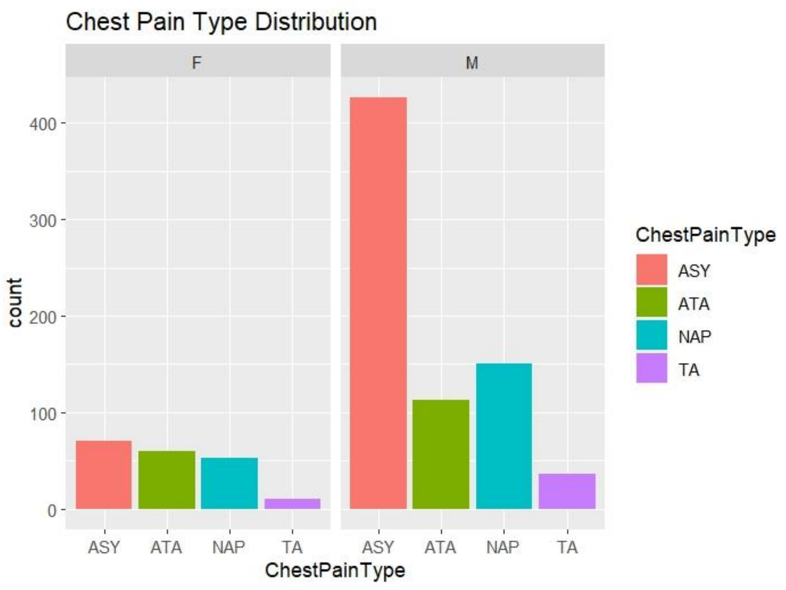


Figure 8. Chest Pain Type Distribution

If we summarize every asymptomatic chest pain type (ASY) we can see that there are all around 500 cases, that means half of the patients had no symptoms that led to heart disease. Men graph is dominated by asymptomatic chest pain type, while women's graph evenly distributed among all.

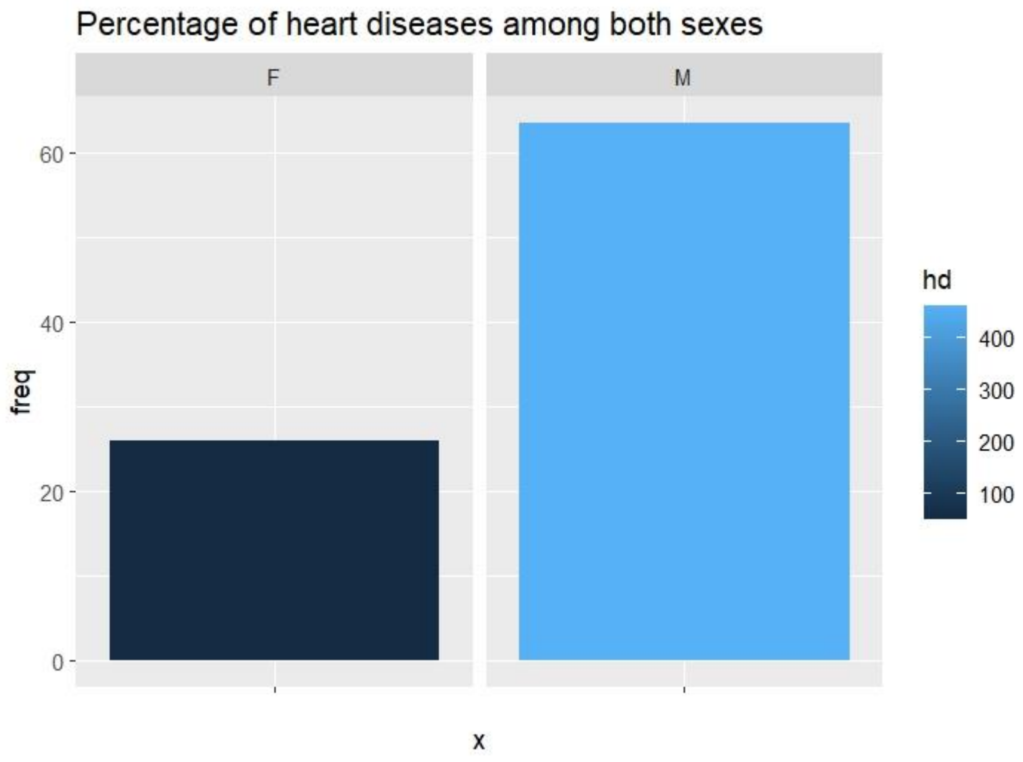


Figure 9. Percentage of Heart Diseases Among Both Sexes

Figure 9 confirms information that heart disease are more common among men with more than 60% of men having heart disease, while less 30% of woman have it. Huge difference between two genders.

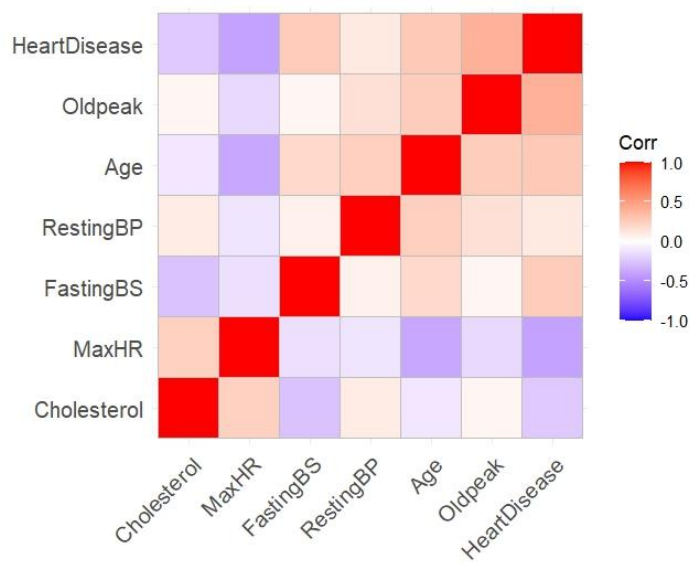


Figure 10. Correlation

There is high correlation between HeartDisease and FastinBS, Age, Oldpeak.

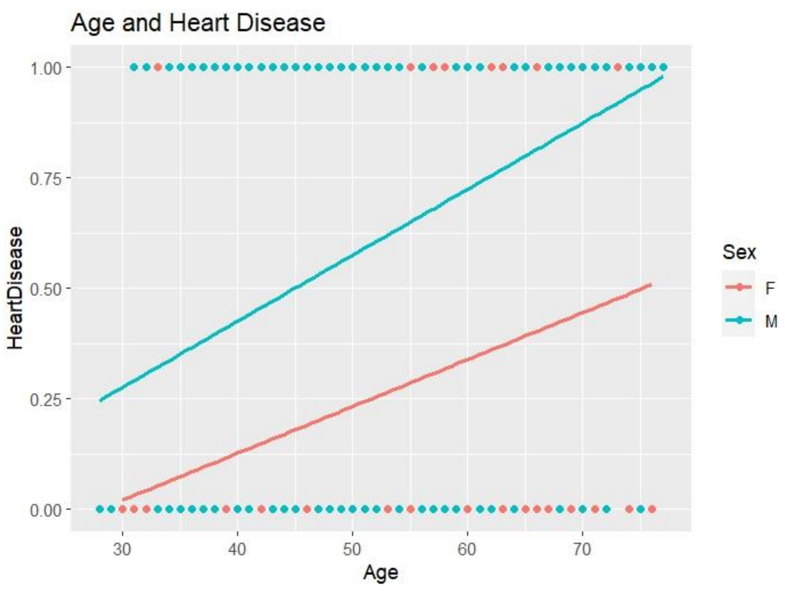


Figure 11. Age and Heart Disease Graphic

Figure 11 provides a graphic with relationship between age and heart disease, as we can that as older people get the chance of them getting a heart disease also increases. Old people tend to have more heart disease, especially men, but for women it is not that high. The number form is that high, that we can say 70+ years old men going to have at least one heart disease.

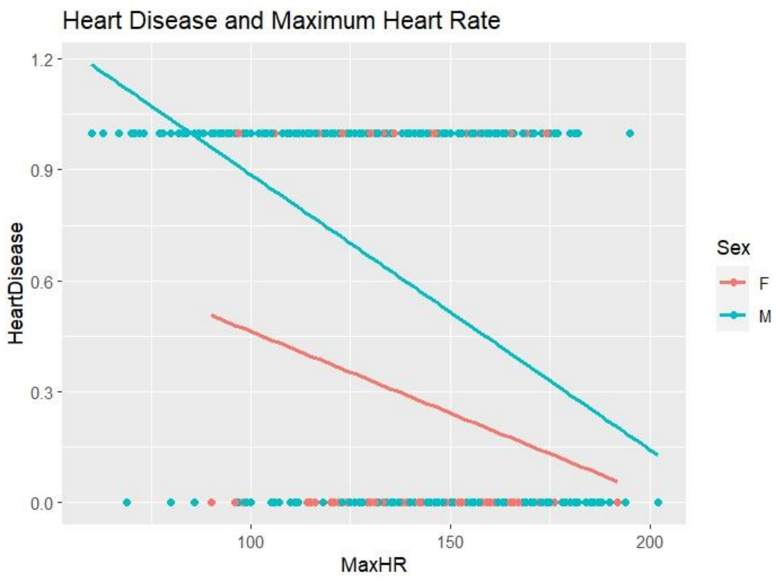


Figure 12. Heart Disease and Maximum Heart Rate

People with heart disease have less maximum heart rate recorded (Figure 12).

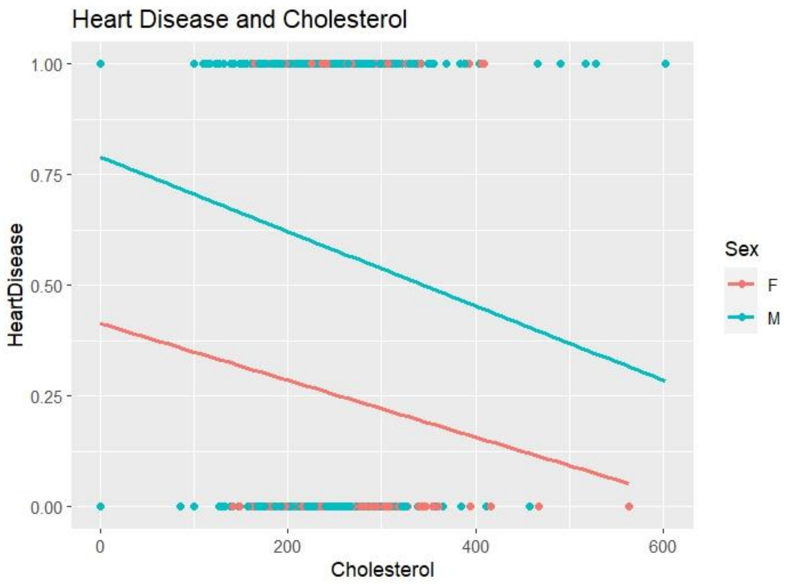


Figure 13. Heart Disease and Cholesterol

People with heart disease have low levels of cholesterol, but according to MayoClinic (2023) high levels of cholesterol can increase your risk of heart disease. Graphics do not go with norms.

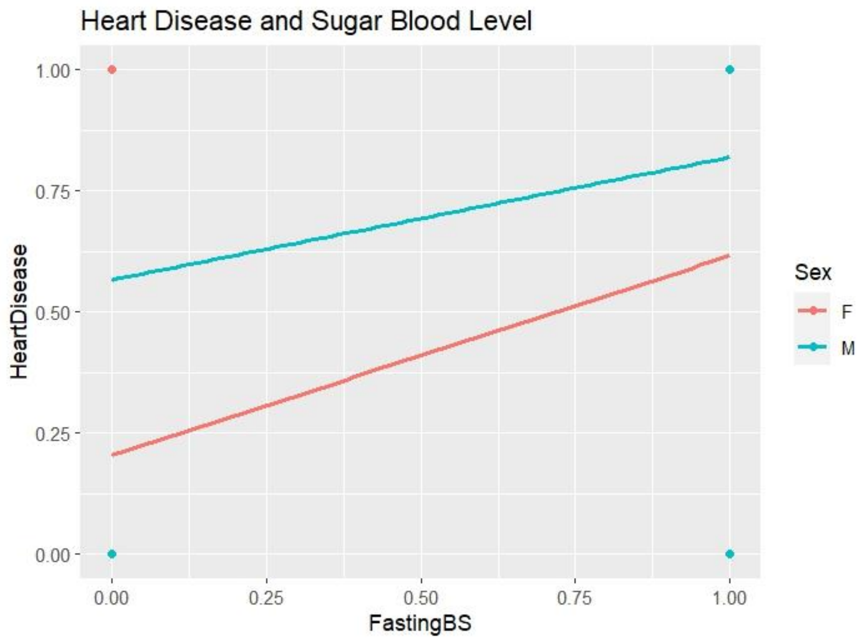


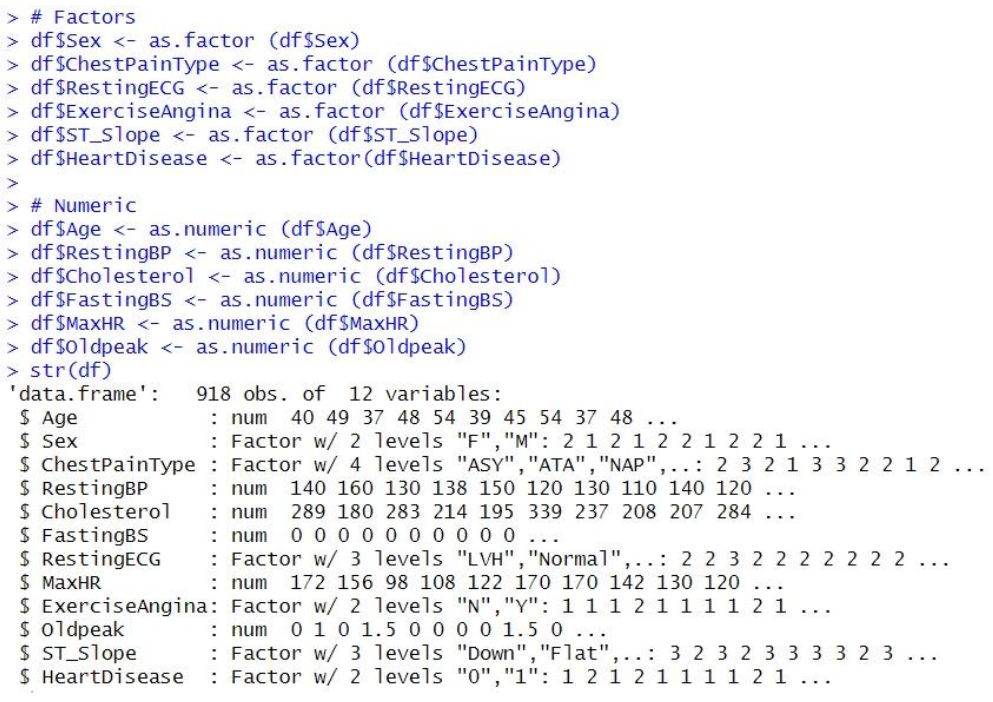
Figure 14. Heart Disease and Sugar Blood Level

People with heart disease have high sugar blood level, according to Figure 14.

The graph gave us information about heart diseases and how they connected to other variables. Most patients that have heart disease are men and older they get higher the probability of getting heart disease. Especially, graphs showed that after turning 70 it is very likely for men to have heart disease. Almost 500 patients did not have any symptoms when they had heart disease, which is 50% of patients. There is a high correlation between blood sugar level and heart disease, which relates to the fact of a person's lifestyle, and their eating habits.

4.2. Pre-processing

Looking at the structure of dataset (Figure 4), we can see that data types of values consist of integers and characters. We need to change them to factors and numerical.



```
> # Factors
> df$Sex <- as.factor (df$Sex)
> df$ChestPainType <- as.factor (df$ChestPainType)
> df$RestingECG <- as.factor (df$RestingECG)
> df$ExerciseAngina <- as.factor (df$ExerciseAngina)
> df$ST_Slope <- as.factor (df$ST_Slope)
> df$HeartDisease <- as.factor(df$HeartDisease)
>
> # Numeric
> df$Age <- as.numeric (df$Age)
> df$RestingBP <- as.numeric (df$RestingBP)
> df$Cholesterol <- as.numeric (df$Cholesterol)
> df$FastingBS <- as.numeric (df$FastingBS)
> df$MaxHR <- as.numeric (df$MaxHR)
> df$Oldpeak <- as.numeric (df$Oldpeak)
> str(df)
'data.frame':   918 obs. of  12 variables:
 $ Age           : num  40 49 37 48 54 39 45 54 37 48 ...
 $ Sex           : Factor w/ 2 levels "F","M": 2 1 2 1 2 2 1 2 2 1 ...
 $ ChestPainType : Factor w/ 4 levels "ASY","ATA","NAP",..: 2 3 2 1 3 3 2 2 1 2 ...
 $ RestingBP     : num  140 160 130 138 150 120 130 110 140 120 ...
 $ Cholesterol   : num  289 180 283 214 195 339 237 208 207 284 ...
 $ FastingBS     : num  0 0 0 0 0 0 0 0 0 0 ...
 $ RestingECG    : Factor w/ 3 levels "LVH","Normal",..: 2 2 3 2 2 2 2 2 2 2 ...
 $ MaxHR         : num  172 156 98 108 122 170 170 142 130 120 ...
 $ ExerciseAngina: Factor w/ 2 levels "N","Y": 1 1 1 2 1 1 1 1 2 1 ...
 $ Oldpeak       : num  0 1 0 1.5 0 0 0 0 1.5 0 ...
 $ ST_Slope      : Factor w/ 3 levels "Down","Flat",..: 3 2 3 2 3 3 3 3 2 3 ...
 $ HeartDisease  : Factor w/ 2 levels "0","1": 1 2 1 2 1 1 1 1 2 1 ...
```

Figure 15. Factor and Numeric Variables

As our dataset had int and chr type values by default, we do data type conversion to get factors and numerical variables (Figure 15).

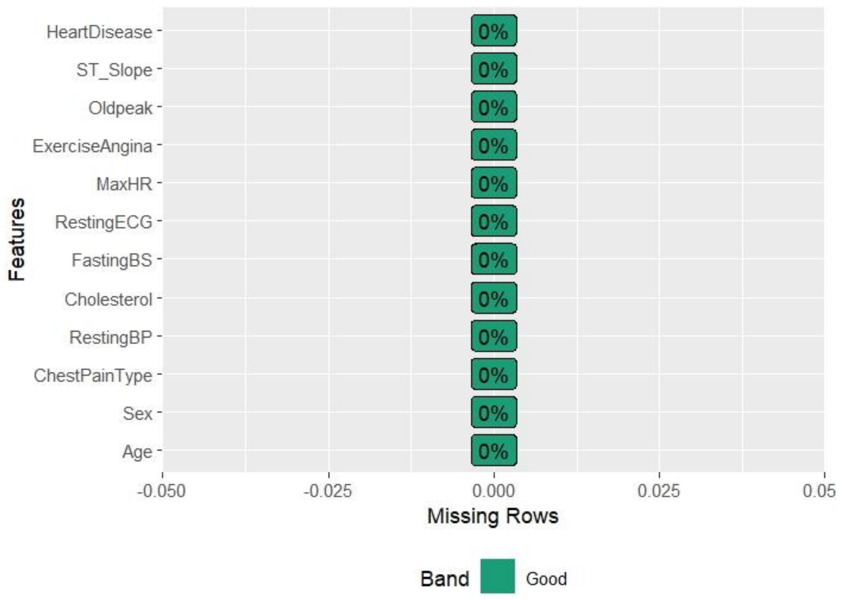


Figure 16. Missing Values

Through plot_missing(df) get information about any missing values in dataset. The graph shows no missing values in dataset (Figure 16).

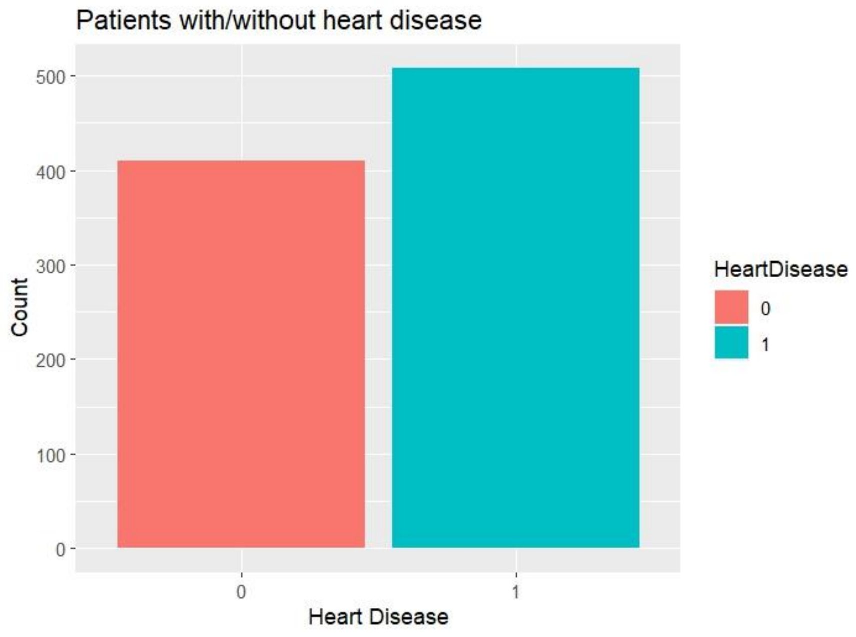
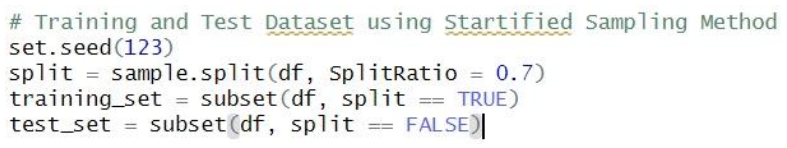


Figure 17. Patients With/Without Heart Disease

Figure 17 provides good graphs that showcase a good balance of data in HeartDisease.

The dataset that was taken from Kaggle was very good in use as it only needed data type conversion. The dataset had no missing values, target variable HeartDisease had good Data Balance.

5. MODEL IMPLEMENTATION



```
# Training and Test Dataset using Startified Sampling Method
set.seed(123)
split = sample.split(df, SplitRatio = 0.7)
training_set = subset(df, split == TRUE)
test_set = subset(df, split == FALSE)
```

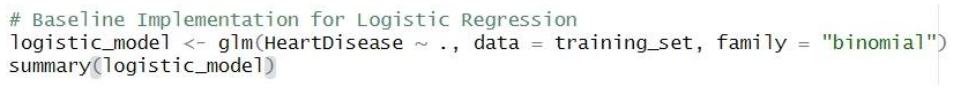
Figure 18. Training and Test Dataset

Before model implementation we need to create training and test sets that we are going to use. The random number value in the line set.seed(123) is set to 123. Replicating a specific series of "random" numbers is the primary goal of using the seed. and sed(n) uses a seed to replicate random number results. By diving 70% of the data as training set and other 30% to test set, the model can train on those 70% and then validate on unseen data.

5.1. Logistic Regression

First, we will go through our Logistic Regression (LR) algorithm.

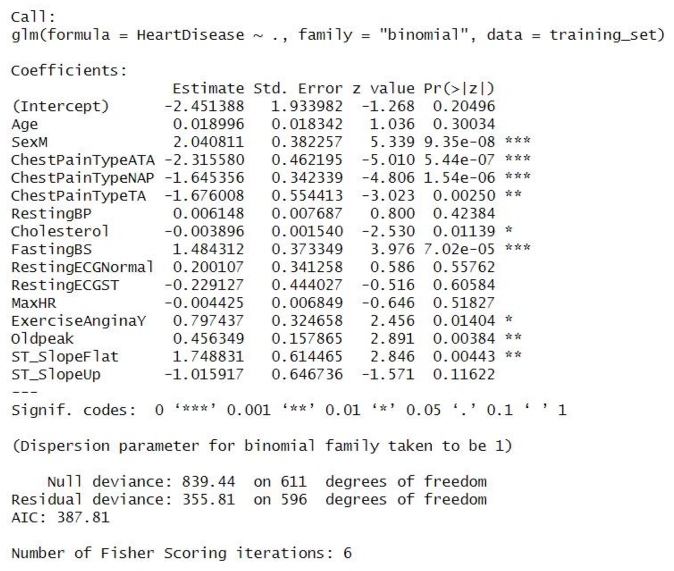
5.1.1. Baseline Implementation



```
# Baseline Implementation for Logistic Regression
logistic_model <- glm(HeartDisease ~ ., data = training_set, family = "binomial")
summary(logistic_model)
```

Figure 19. Baseline Implementation for LR

The code (Figure 19) uses glm function to create logistic regression model, where training set being used, while HeartDisease variable is the target for the class verification. The model summary shown below:

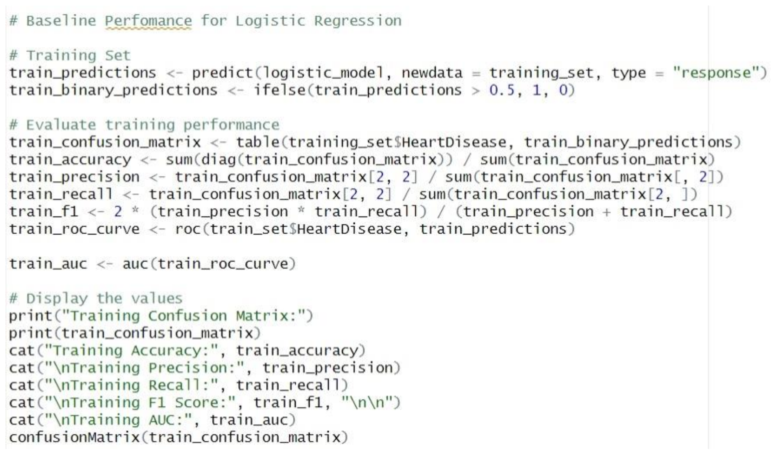


```
Call:
glm(formula = HeartDisease ~ ., family = "binomial", data = training_set)

Coefficients:
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)      -2.451388   1.933982  -1.268  0.20496
Age               0.018996   0.018342   1.036  0.30034
SexM              2.040811   0.382257   5.339 9.35e-08 ***
ChestPainTypeATA -2.315580   0.462195  -5.010 5.44e-07 ***
ChestPainTypeNAP -1.645356   0.342339  -4.806 1.54e-06 ***
ChestPainTypeTA  -1.676008   0.554413  -3.023  0.00250 **
RestingBP         0.006148   0.007687   0.800  0.42384
Cholesterol      -0.003896   0.001540  -2.530  0.01139 *
FastingBS         1.484312   0.373349   3.976 7.02e-05 ***
RestingECGNormal  0.200107   0.341258   0.586  0.55762
RestingECGST     -0.229127   0.444027  -0.516  0.60584
MaxHR            -0.004425   0.006849  -0.646  0.51827
ExerciseAnginaY   0.797437   0.324658   2.456  0.01404 *
Oldpeak           0.456349   0.157865   2.891  0.00384 **
ST_SlopeFlat      1.748831   0.614465   2.846  0.00443 **
ST_SlopeUp       -1.015917   0.646736  -1.571  0.11622
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 839.44  on 611  degrees of freedom
Residual deviance: 355.81  on 596  degrees of freedom
AIC: 387.81

Number of Fisher Scoring iterations: 6
```

Figure 20. Summary of LR's Baseline Implementation

From model summary we can say that variables like SexM, ChestPainTypeATA, ChestPainTypeNAP, FastingBS, ExerciseAnginaY, Oldpeak, and ST_SlopeFlat are significant predictors, because they have low p-values. The value AIC is low, so we suggest we have better-fitting model. The prediction model's (355.81) deviance from the null model (839.44) decreases, indicating that the predictors play a major role in explaining the response variable's variability.

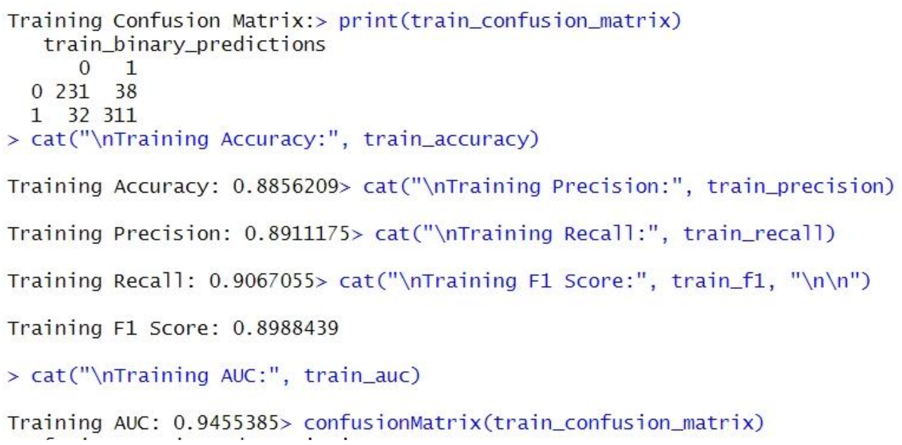
5.1.2. Baseline Performance



```
# Baseline Perfomance for Logistic Regression

# Training Set
train_predictions <- predict(logistic_model, newdata = training_set, type = "response")
train_binary_predictions <- ifelse(train_predictions > 0.5, 1, 0)

# Evaluate training performance
train_confusion_matrix <- table(training_set$HeartDisease, train_binary_predictions)
train_accuracy <- sum(diag(train_confusion_matrix)) / sum(train_confusion_matrix)
train_precision <- train_confusion_matrix[2, 2] / sum(train_confusion_matrix[, 2])
train_recall <- train_confusion_matrix[2, 2] / sum(train_confusion_matrix[2, ])
train_f1 <- 2 * (train_precision * train_recall) / (train_precision + train_recall)
train_roc_curve <- roc(train_set$HeartDisease, train_predictions)

train_auc <- auc(train_roc_curve)

# Display the values
print("Training Confusion Matrix:")
print(train_confusion_matrix)
cat("Training Accuracy:", train_accuracy)
cat("\nTraining Precision:", train_precision)
cat("\nTraining Recall:", train_recall)
cat("\nTraining F1 Score:", train_f1, "\n\n")
cat("\nTraining AUC:", train_auc)
confusionMatrix(train_confusion_matrix)
```

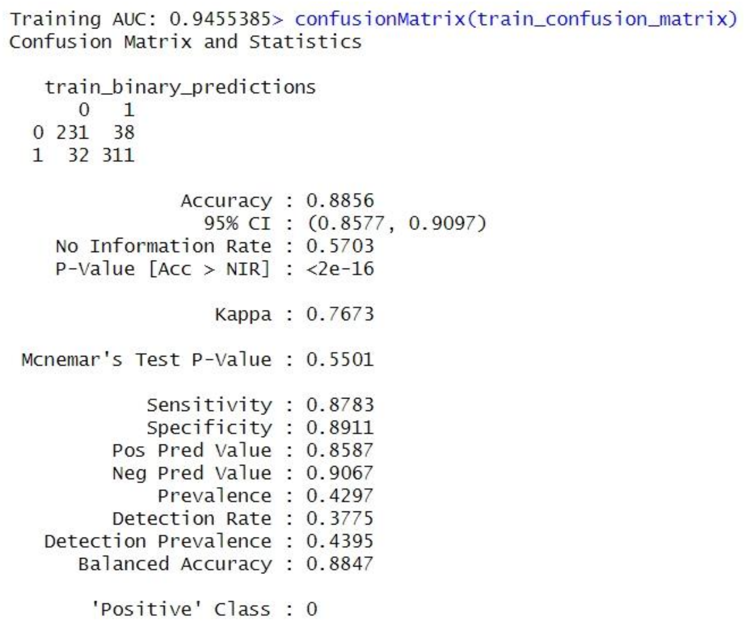
Figure 21. Code for LR's Baseline Performance on Training Set

Figure 21 provides us the information with the code of baseline performance on training set. The images below provide the results of that code:



```
Training Confusion Matrix:> print(train_confusion_matrix)
   train_binary_predictions
      0   1
  0 231  38
  1  32 311
> cat("\nTraining Accuracy:", train_accuracy)

Training Accuracy: 0.8856209> cat("\nTraining Precision:", train_precision)

Training Precision: 0.8911175> cat("\nTraining Recall:", train_recall)

Training Recall: 0.9067055> cat("\nTraining F1 Score:", train_f1, "\n\n")

Training F1 Score: 0.8988439

> cat("\nTraining AUC:", train_auc)

Training AUC: 0.9455385> confusionMatrix(train_confusion_matrix)
```

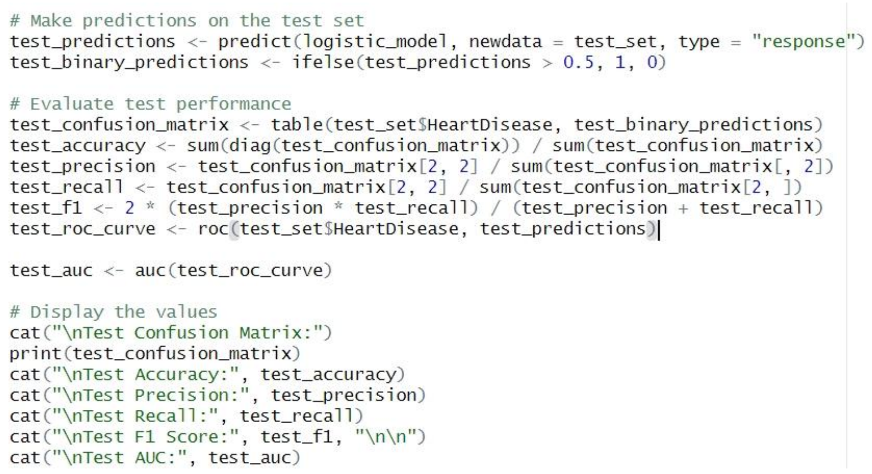
Figure 22. Output of Training Set for LR's Baseline Performance



```
Training AUC: 0.9455385> confusionMatrix(train_confusion_matrix)
Confusion Matrix and Statistics

   train_binary_predictions
      0   1
  0 231  38
  1  32 311

               Accuracy : 0.8856
                 95% CI : (0.8577, 0.9097)
    No Information Rate : 0.5703
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.7673

 Mcnemar's Test P-Value : 0.5501

            Sensitivity : 0.8783
            Specificity : 0.8911
         Pos Pred Value : 0.8587
         Neg Pred Value : 0.9067
             Prevalence : 0.4297
         Detection Rate : 0.3775
   Detection Prevalence : 0.4395
      Balanced Accuracy : 0.8847

       'Positive' Class : 0
```

Figure 23. Confusion Matrix and Statistics for LR's Baseline Performance on Training Set

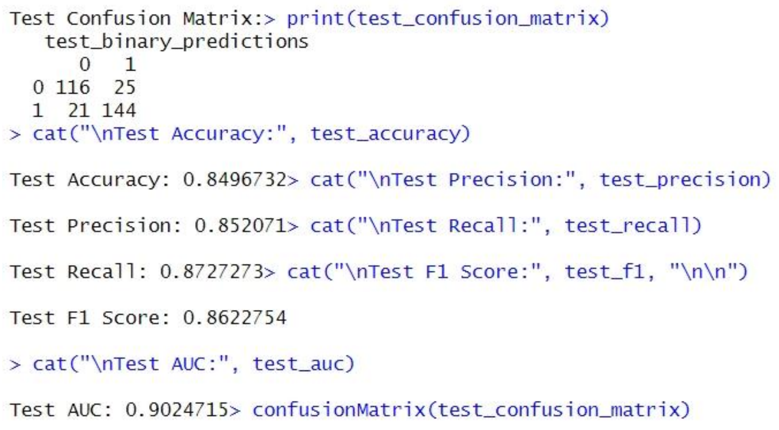
Logistic Regression (LR) provided good results with the accuracy of 0.8856. High accuracy, precision, recall, and AUC, along with a balanced F1 Score, indicate a well-performing model.



```
# Make predictions on the test set
test_predictions <- predict(logistic_model, newdata = test_set, type = "response")
test_binary_predictions <- ifelse(test_predictions > 0.5, 1, 0)

# Evaluate test performance
test_confusion_matrix <- table(test_set$HeartDisease, test_binary_predictions)
test_accuracy <- sum(diag(test_confusion_matrix)) / sum(test_confusion_matrix)
test_precision <- test_confusion_matrix[2, 2] / sum(test_confusion_matrix[, 2])
test_recall <- test_confusion_matrix[2, 2] / sum(test_confusion_matrix[2, ])
test_f1 <- 2 * (test_precision * test_recall) / (test_precision + test_recall)
test_roc_curve <- roc(test_set$HeartDisease, test_predictions)

test_auc <- auc(test_roc_curve)

# Display the values
cat("\nTest Confusion Matrix:")
print(test_confusion_matrix)
cat("\nTest Accuracy:", test_accuracy)
cat("\nTest Precision:", test_precision)
cat("\nTest Recall:", test_recall)
cat("\nTest F1 Score:", test_f1, "\n\n")
cat("\nTest AUC:", test_auc)
```

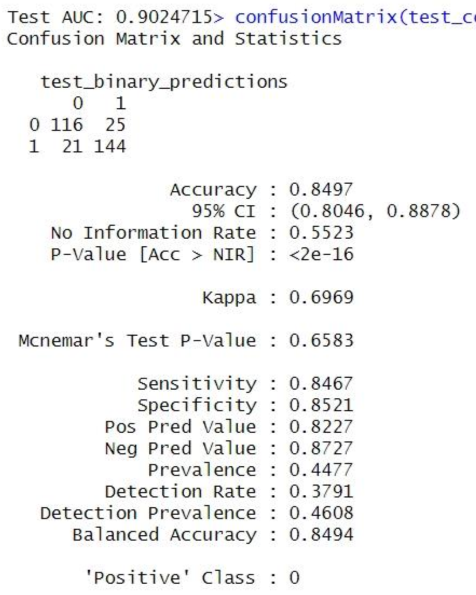
Figure 24. Code for LR’s Baseline Performance on Test Set

Pretty much the same code, but now we will use the test set.



```
Test Confusion Matrix:> print(test_confusion_matrix)
   test_binary_predictions
      0   1
  0 116  25
  1  21 144
> cat("\nTest Accuracy:", test_accuracy)

Test Accuracy: 0.8496732> cat("\nTest Precision:", test_precision)

Test Precision: 0.852071> cat("\nTest Recall:", test_recall)

Test Recall: 0.8727273> cat("\nTest F1 Score:", test_f1, "\n\n")

Test F1 Score: 0.8622754

> cat("\nTest AUC:", test_auc)

Test AUC: 0.9024715> confusionMatrix(test_confusion_matrix)
```

Figure 25. Output of Test Set for LR’s Baseline Performance on Test Set

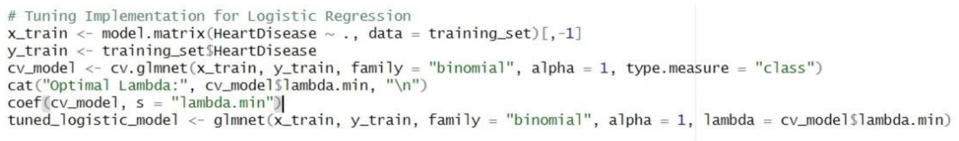


```
Test AUC: 0.9024715> confusionMatrix(test_confusion_matrix)
Confusion Matrix and Statistics

   test_binary_predictions
      0   1
  0 116  25
  1  21 144

               Accuracy : 0.8497
                 95% CI : (0.8046, 0.8878)
    No Information Rate : 0.5523
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.6969

 Mcnemar's Test P-Value : 0.6583

            Sensitivity : 0.8467
            Specificity : 0.8521
         Pos Pred Value : 0.8227
         Neg Pred Value : 0.8727
             Prevalence : 0.4477
         Detection Rate : 0.3791
   Detection Prevalence : 0.4608
      Balanced Accuracy : 0.8494

       'Positive' Class : 0
```

Figure 26. Confusion Matrix and Statistics for LR's Baseline Performance on Test Set

Compared to the training set, values from the test set decreased, as accuracy went down by 4%. Both sensitivity and specificity values indicate their balanced performance. Overall, the model seems to perform good, with high accuracy, precision, recall, F1 score, and AUC.

5.1.3. Tuning Implementation

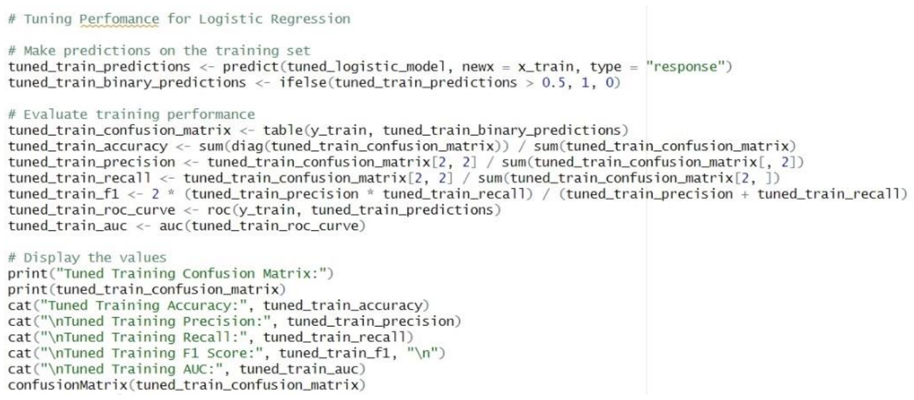


```
# Tuning Implementation for Logistic Regression
x_train <- model.matrix(HeartDisease ~ ., data = training_set)[,-1]
y_train <- training_set$HeartDisease
cv_model <- cv.glmnet(x_train, y_train, family = "binomial", alpha = 1, type.measure = "class")
cat("Optimal Lambda:", cv_model$lambda.min, "\n")
coef(cv_model, s = "lambda.min")
tuned_logistic_model <- glmnet(x_train, y_train, family = "binomial", alpha = 1, lambda = cv_model$lambda.min)
```

Figure 27. Tuning Implementation for LR

With the use of “glmnet” library, tune our model. Create matrix of predictors and set up cross validation with glmnet, then display optimal lambda value. The optimal lambda value is 0.0182.

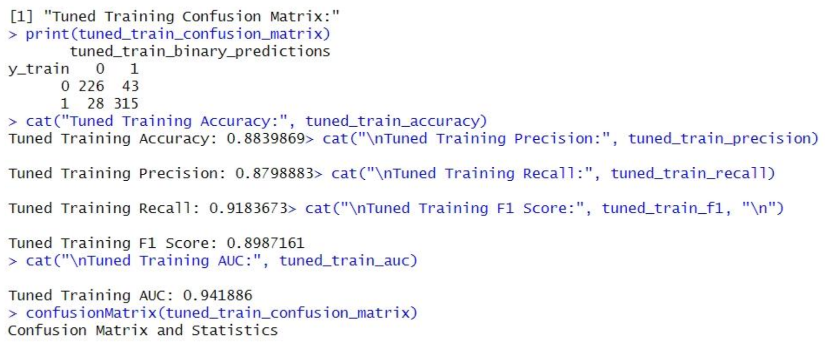
5.1.4. Tuning Performance



```
# Tuning Perfomance for Logistic Regression

# Make predictions on the training set
tuned_train_predictions <- predict(tuned_logistic_model, newx = x_train, type = "response")
tuned_train_binary_predictions <- ifelse(tuned_train_predictions > 0.5, 1, 0)

# Evaluate training performance
tuned_train_confusion_matrix <- table(y_train, tuned_train_binary_predictions)
tuned_train_accuracy <- sum(diag(tuned_train_confusion_matrix)) / sum(tuned_train_confusion_matrix)
tuned_train_precision <- tuned_train_confusion_matrix[2, 2] / sum(tuned_train_confusion_matrix[, 2])
tuned_train_recall <- tuned_train_confusion_matrix[2, 2] / sum(tuned_train_confusion_matrix[2, ])
tuned_train_f1 <- 2 * (tuned_train_precision * tuned_train_recall) / (tuned_train_precision + tuned_train_recall)
tuned_train_roc_curve <- roc(y_train, tuned_train_predictions)
tuned_train_auc <- auc(tuned_train_roc_curve)

# Display the values
print("Tuned Training Confusion Matrix:")
print(tuned_train_confusion_matrix)
cat("Tuned Training Accuracy:", tuned_train_accuracy)
cat("\nTuned Training Precision:", tuned_train_precision)
cat("\nTuned Training Recall:", tuned_train_recall)
cat("\nTuned Training F1 Score:", tuned_train_f1, "\n")
cat("\nTuned Training AUC:", tuned_train_auc)
confusionMatrix(tuned_train_confusion_matrix)
```

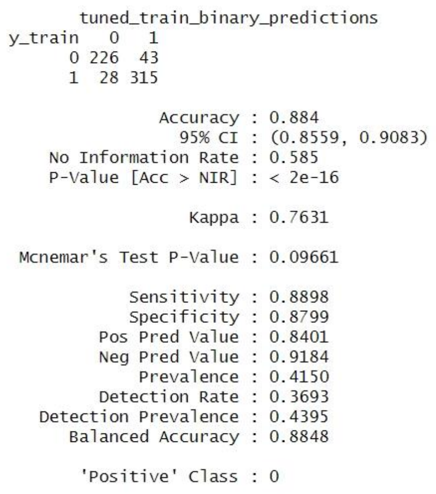
Figure 28. Code for LR’s Tuning Performance on Training Set

Figure 28 provides code for LR’s Tuning Performance on Training Set; the results show in images below:



```
[1] "Tuned Training Confusion Matrix:"
> print(tuned_train_confusion_matrix)
       tuned_train_binary_predictions
y_train   0   1
      0 226  43
      1  28 315
> cat("Tuned Training Accuracy:", tuned_train_accuracy)
Tuned Training Accuracy: 0.8839869> cat("\nTuned Training Precision:", tuned_train_precision)

Tuned Training Precision: 0.8798883> cat("\nTuned Training Recall:", tuned_train_recall)

Tuned Training Recall: 0.9183673> cat("\nTuned Training F1 Score:", tuned_train_f1, "\n")

Tuned Training F1 Score: 0.8987161
> cat("\nTuned Training AUC:", tuned_train_auc)

Tuned Training AUC: 0.941886
> confusionMatrix(tuned_train_confusion_matrix)
Confusion Matrix and Statistics
```

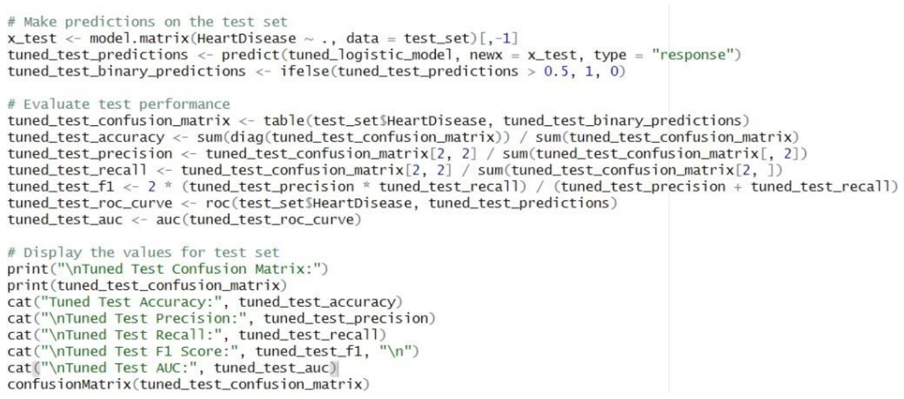
Figure 29. Output of Training Set for LR’s Tuning Performance



```
       tuned_train_binary_predictions
y_train   0   1
      0 226  43
      1  28 315

               Accuracy : 0.884
                 95% CI : (0.8559, 0.9083)
    No Information Rate : 0.585
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.7631

 Mcnemar's Test P-Value : 0.09661

            Sensitivity : 0.8898
            Specificity : 0.8799
         Pos Pred Value : 0.8401
         Neg Pred Value : 0.9184
             Prevalence : 0.4150
         Detection Rate : 0.3693
   Detection Prevalence : 0.4395
      Balanced Accuracy : 0.8848

       'Positive' Class : 0
```

Figure 30. Confusion Matrix and Statistics for LR’s Tuning Performance on Training Set

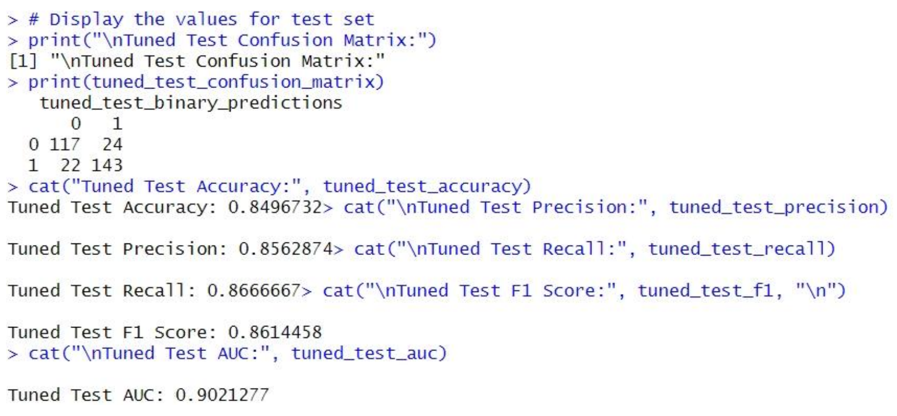
From the confusion matrix: 226 true negative, 43 false positive, 28 false negatives, 315 true negatives with the model’s 88.40% accuracy, which is the same results as baseline performance. Comparing to baseline performance on training set, we can say that precision dropped by 2%, recall raised by 1%, F1 score and AUC stayed the same.



```
# Make predictions on the test set
x_test <- model.matrix(HeartDisease ~ ., data = test_set)[,-1]
tuned_test_predictions <- predict(tuned_logistic_model, newx = x_test, type = "response")
tuned_test_binary_predictions <- ifelse(tuned_test_predictions > 0.5, 1, 0)

# Evaluate test performance
tuned_test_confusion_matrix <- table(test_set$HeartDisease, tuned_test_binary_predictions)
tuned_test_accuracy <- sum(diag(tuned_test_confusion_matrix)) / sum(tuned_test_confusion_matrix)
tuned_test_precision <- tuned_test_confusion_matrix[2, 2] / sum(tuned_test_confusion_matrix[, 2])
tuned_test_recall <- tuned_test_confusion_matrix[2, 2] / sum(tuned_test_confusion_matrix[2, ])
tuned_test_f1 <- 2 * (tuned_test_precision * tuned_test_recall) / (tuned_test_precision + tuned_test_recall)
tuned_test_roc_curve <- roc(test_set$HeartDisease, tuned_test_predictions)
tuned_test_auc <- auc(tuned_test_roc_curve)

# Display the values for test set
print("\nTuned Test Confusion Matrix:")
print(tuned_test_confusion_matrix)
cat("Tuned Test Accuracy:", tuned_test_accuracy)
cat("\nTuned Test Precision:", tuned_test_precision)
cat("\nTuned Test Recall:", tuned_test_recall)
cat("\nTuned Test F1 Score:", tuned_test_f1, "\n")
cat("\nTuned Test AUC:", tuned_test_auc)
confusionMatrix(tuned_test_confusion_matrix)
```

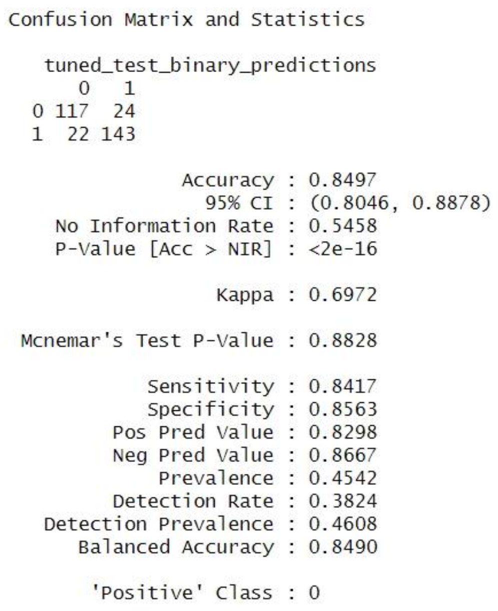
Figure 31. Code for LR’s Tuning Performance on Training Set

Figure 31 provides code for LR’s Tuning Performance on Training Set, performance on images shown below:



```
> # Display the values for test set
> print("\nTuned Test Confusion Matrix:")
[1] "\nTuned Test Confusion Matrix:"
> print(tuned_test_confusion_matrix)
   tuned_test_binary_predictions
      0   1
  0 117  24
  1  22 143
> cat("Tuned Test Accuracy:", tuned_test_accuracy)
Tuned Test Accuracy: 0.8496732> cat("\nTuned Test Precision:", tuned_test_precision)

Tuned Test Precision: 0.8562874> cat("\nTuned Test Recall:", tuned_test_recall)

Tuned Test Recall: 0.8666667> cat("\nTuned Test F1 Score:", tuned_test_f1, "\n")

Tuned Test F1 Score: 0.8614458
> cat("\nTuned Test AUC:", tuned_test_auc)

Tuned Test AUC: 0.9021277
```

Figure 32. Output of Test Set for LR’s Tuning Performance



```
Confusion Matrix and Statistics

   tuned_test_binary_predictions
      0   1
  0 117  24
  1  22 143

               Accuracy : 0.8497
                 95% CI : (0.8046, 0.8878)
    No Information Rate : 0.5458
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.6972

 Mcnemar's Test P-Value : 0.8828

            Sensitivity : 0.8417
            Specificity : 0.8563
         Pos Pred Value : 0.8298
         Neg Pred Value : 0.8667
             Prevalence : 0.4542
         Detection Rate : 0.3824
   Detection Prevalence : 0.4608
      Balanced Accuracy : 0.8490

       'Positive' Class : 0
```

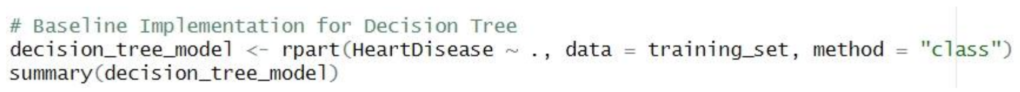
Figure 33. Confusion Matrix and Statistics for LR's Tuning Performance on Test Set

The same as baseline performance, accuracy dropped comparing training set and test set, but if we compare tuning performance with baseline performance, then values stayed the same, except for recall, where tuning performance's recall raised by 1%.

5.2. Decision Tree

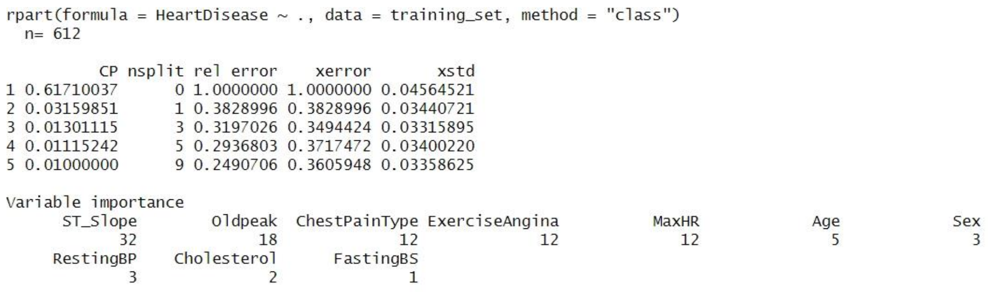
After the Logistic Regression, we are going to Decision Tree (DT) – to one of the oldest algorithms. Keep doing models in R programming language.

5.2.1. Baseline Implementation



```
# Baseline Implementation for Decision Tree
decision_tree_model <- rpart(HeartDisease ~ ., data = training_set, method = "class")
summary(decision_tree_model)
```

Figure 34. Code for DT's Baseline Implementation

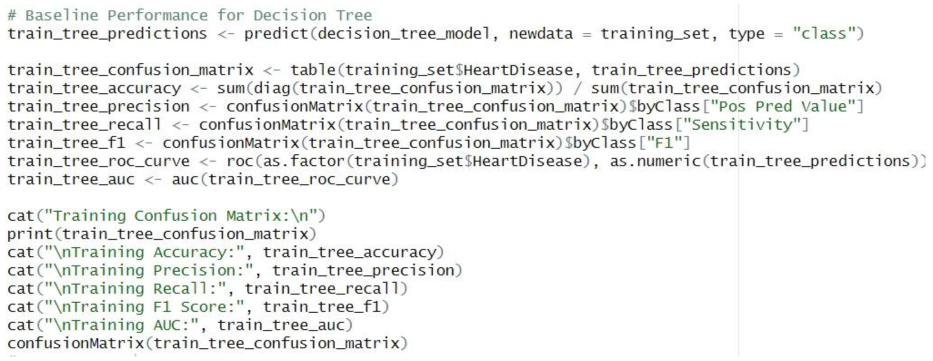


```
rpart(formula = HeartDisease ~ ., data = training_set, method = "class")
  n= 612

          CP nsplit rel error    xerror       xstd
1 0.61710037      0 1.0000000 1.0000000 0.04564521
2 0.03159851      1 0.3828996 0.3828996 0.03440721
3 0.01301115      3 0.3197026 0.3494424 0.03315895
4 0.01115242      5 0.2936803 0.3717472 0.03400220
5 0.01000000      9 0.2490706 0.3605948 0.03358625

Variable importance
      ST_Slope        Oldpeak  ChestPainType ExerciseAngina          MaxHR            Age            Sex
            32             18             12             12             12              5              3
     RestingBP    Cholesterol      FastingBS
             3              2              1
```

Figure 35. Summary for DT’s Baseline Implementation

Using library rpart we can create decision tree’s model, where HeartDisease is target variable with method “class”. From the summary, we can say that due to CP = 0.617 our DT is complex with most important predictors being ST_Slope, Oldpeak, ChestPainType. CP value decreased, so we can say that the tree undergone pruning to reduce overfitting.

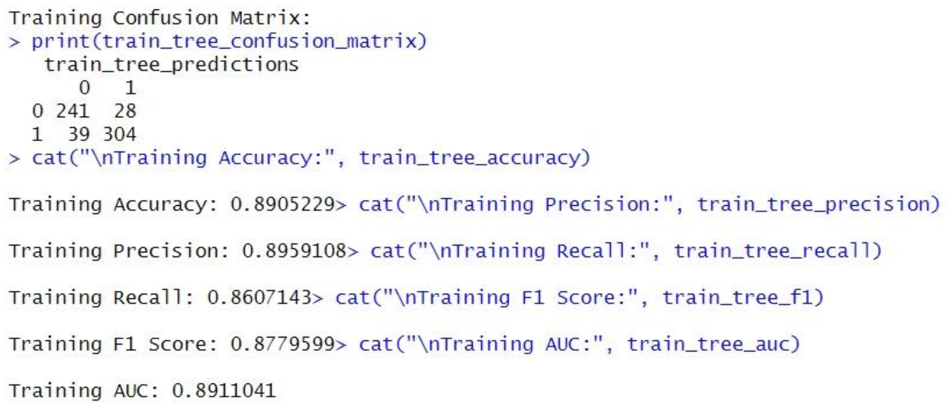
5.2.1. Baseline Performance



```
# Baseline Performance for Decision Tree
train_tree_predictions <- predict(decision_tree_model, newdata = training_set, type = "class")

train_tree_confusion_matrix <- table(training_set$HeartDisease, train_tree_predictions)
train_tree_accuracy <- sum(diag(train_tree_confusion_matrix)) / sum(train_tree_confusion_matrix)
train_tree_precision <- confusionMatrix(train_tree_confusion_matrix)$byClass["Pos Pred Value"]
train_tree_recall <- confusionMatrix(train_tree_confusion_matrix)$byClass["Sensitivity"]
train_tree_f1 <- confusionMatrix(train_tree_confusion_matrix)$byClass["F1"]
train_tree_roc_curve <- roc(as.factor(training_set$HeartDisease), as.numeric(train_tree_predictions))
train_tree_auc <- auc(train_tree_roc_curve)

cat("Training Confusion Matrix:\n")
print(train_tree_confusion_matrix)
cat("\nTraining Accuracy:", train_tree_accuracy)
cat("\nTraining Precision:", train_tree_precision)
cat("\nTraining Recall:", train_tree_recall)
cat("\nTraining F1 Score:", train_tree_f1)
cat("\nTraining AUC:", train_tree_auc)
confusionMatrix(train_tree_confusion_matrix)
```

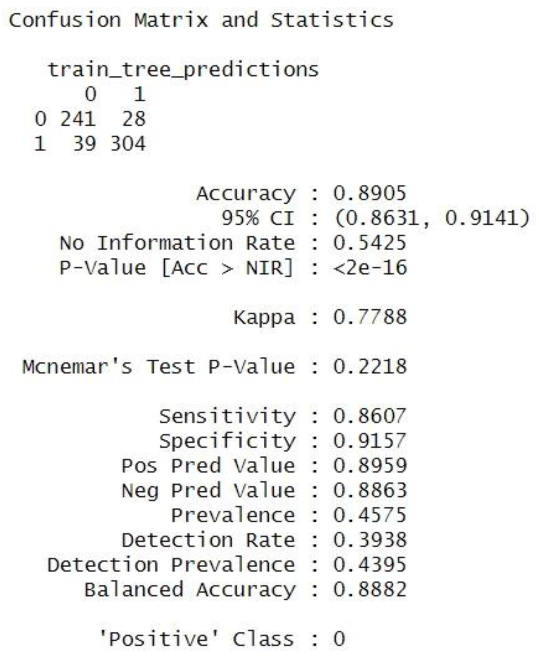
Figure 36. Code for DT’s Baseline Performance on Training Set

Code, on Figure 36, is the code for DT’s Baseline Performance on Training Set. Results of the code shows in images below:



```
Training Confusion Matrix:
> print(train_tree_confusion_matrix)
   train_tree_predictions
      0   1
  0 241  28
  1  39 304
> cat("\nTraining Accuracy:", train_tree_accuracy)

Training Accuracy: 0.8905229> cat("\nTraining Precision:", train_tree_precision)

Training Precision: 0.8959108> cat("\nTraining Recall:", train_tree_recall)

Training Recall: 0.8607143> cat("\nTraining F1 Score:", train_tree_f1)

Training F1 Score: 0.8779599> cat("\nTraining AUC:", train_tree_auc)

Training AUC: 0.8911041
```

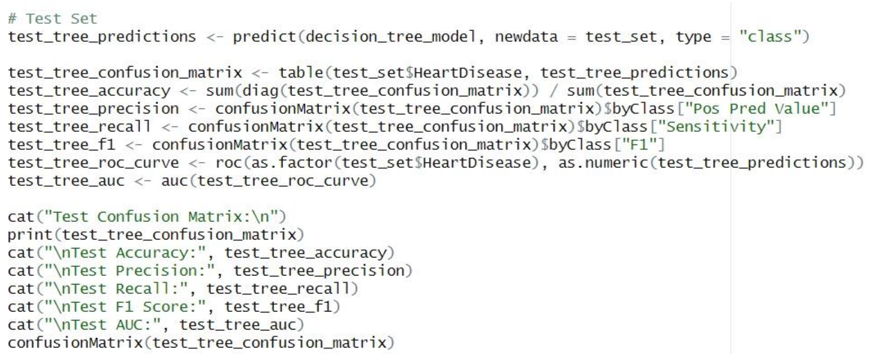
Figure 37. Output of Training Set for DT's Baseline Performance



```
Confusion Matrix and Statistics

   train_tree_predictions
      0   1
  0 241  28
  1  39 304

               Accuracy : 0.8905
                 95% CI : (0.8631, 0.9141)
    No Information Rate : 0.5425
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.7788

 Mcnemar's Test P-Value : 0.2218

            Sensitivity : 0.8607
            Specificity : 0.9157
         Pos Pred Value : 0.8959
         Neg Pred Value : 0.8863
             Prevalence : 0.4575
         Detection Rate : 0.3938
   Detection Prevalence : 0.4395
      Balanced Accuracy : 0.8882

       'Positive' Class : 0
```

Figure 37. Confusion Matrix and Statistics for DT's Baseline Performance on Training Set

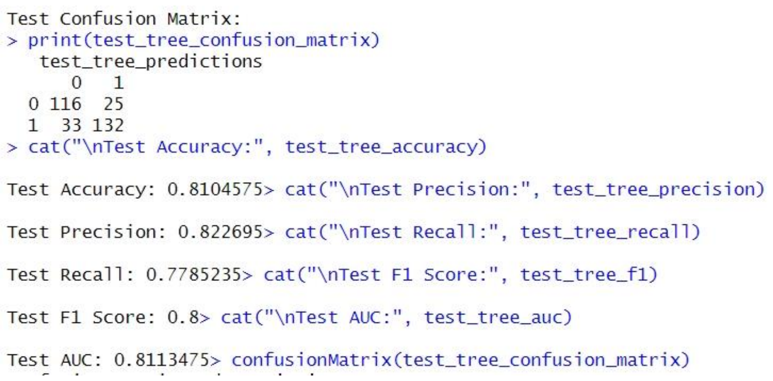
The training accuracy of 89.05% is the highest accuracy that has been recorded for now. Other metrics are also around 86-89%, so we can say that DT model provides good performance on training set.



```
# Test Set
test_tree_predictions <- predict(decision_tree_model, newdata = test_set, type = "class")

test_tree_confusion_matrix <- table(test_set$HeartDisease, test_tree_predictions)
test_tree_accuracy <- sum(diag(test_tree_confusion_matrix)) / sum(test_tree_confusion_matrix)
test_tree_precision <- confusionMatrix(test_tree_confusion_matrix)$byClass["Pos Pred Value"]
test_tree_recall <- confusionMatrix(test_tree_confusion_matrix)$byClass["Sensitivity"]
test_tree_f1 <- confusionMatrix(test_tree_confusion_matrix)$byClass["F1"]
test_tree_roc_curve <- roc(as.factor(test_set$HeartDisease), as.numeric(test_tree_predictions))
test_tree_auc <- auc(test_tree_roc_curve)

cat("Test Confusion Matrix:\n")
print(test_tree_confusion_matrix)
cat("\nTest Accuracy:", test_tree_accuracy)
cat("\nTest Precision:", test_tree_precision)
cat("\nTest Recall:", test_tree_recall)
cat("\nTest F1 Score:", test_tree_f1)
cat("\nTest AUC:", test_tree_auc)
confusionMatrix(test_tree_confusion_matrix)
```

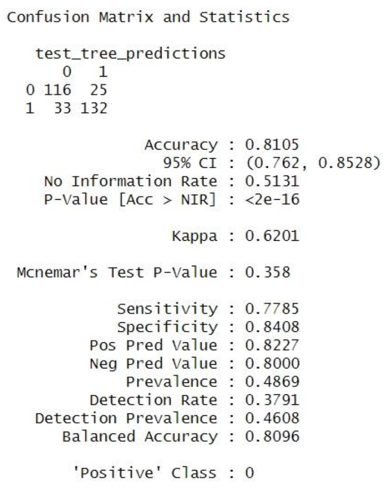
Figure 38. Code for DT's Baseline Performance on Test Set

There's code of DT's baseline performance on test set (Figure 38). The results of code shown below:



```
Test Confusion Matrix:
> print(test_tree_confusion_matrix)
   test_tree_predictions
      0   1
  0 116  25
  1  33 132
> cat("\nTest Accuracy:", test_tree_accuracy)

Test Accuracy: 0.8104575> cat("\nTest Precision:", test_tree_precision)

Test Precision: 0.822695> cat("\nTest Recall:", test_tree_recall)

Test Recall: 0.7785235> cat("\nTest F1 Score:", test_tree_f1)

Test F1 Score: 0.8> cat("\nTest AUC:", test_tree_auc)

Test AUC: 0.8113475> confusionMatrix(test_tree_confusion_matrix)
```

Figure 39. Output for DT's Baseline Performance on Test Set

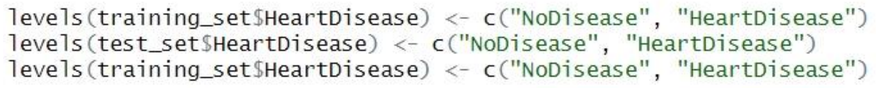


```
Confusion Matrix and Statistics

   test_tree_predictions
      0   1
  0 116  25
  1  33 132

               Accuracy : 0.8105
                 95% CI : (0.762, 0.8528)
    No Information Rate : 0.5131
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.6201

 Mcnemar's Test P-Value : 0.358

            Sensitivity : 0.7785
            Specificity : 0.8408
         Pos Pred Value : 0.8227
         Neg Pred Value : 0.8000
             Prevalence : 0.4869
         Detection Rate : 0.3791
   Detection Prevalence : 0.4608
      Balanced Accuracy : 0.8096

       'Positive' Class : 0
```

Figure 40. Confusion Matrix and Statistics for DT's Baseline Performance on Test Set

Comparing results of the training set and test set, we can see a dropout in numbers. Every metric dropped by 7-8% with accuracy being 81%. The model is marginally more adept at identifying true negatives than true positives, as evidenced by the fact that sensitivity is somewhat lower than specificity. For now, this is the worst performance from model. Overall, it is good performance for models, but for specific goals model needs tuning.

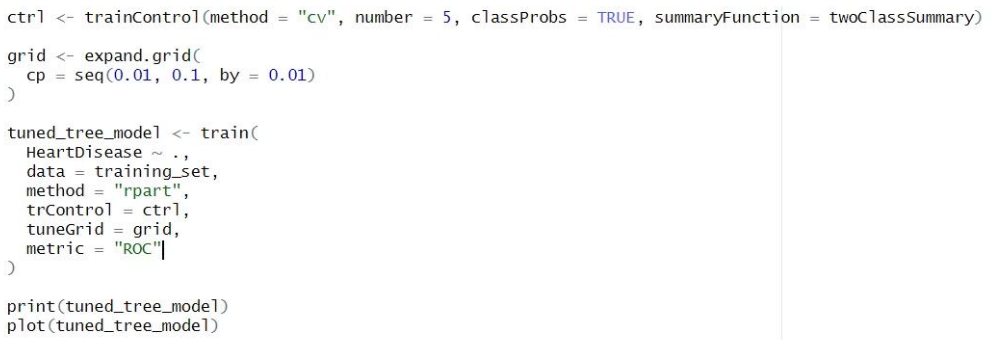
5.2.3. Tuning Implementation



```
levels(training_set$HeartDisease) <- c("NoDisease", "HeartDisease")
levels(test_set$HeartDisease) <- c("NoDisease", "HeartDisease")
levels(training_set$HeartDisease) <- c("NoDisease", "HeartDisease")
```

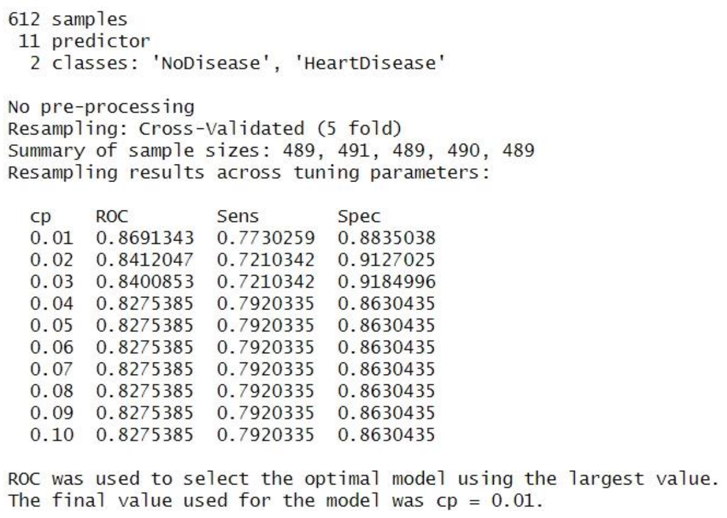
Figure 41. Changing Levels for HeartDisease

To continue with tuning implementation, we need to change levels of HeartDisease from 0 and 1 to NoDisease and HeartDisease.



```
ctrl <- trainControl(method = "cv", number = 5, classProbs = TRUE, summaryFunction = twoClassSummary)

grid <- expand.grid(
  cp = seq(0.01, 0.1, by = 0.01)
)

tuned_tree_model <- train(
  HeartDisease ~ .,
  data = training_set,
  method = "rpart",
  trControl = ctrl,
  tuneGrid = grid,
  metric = "ROC"
)

print(tuned_tree_model)
plot(tuned_tree_model)
```

Figure 42. Code for DT’s Tuning Implementation

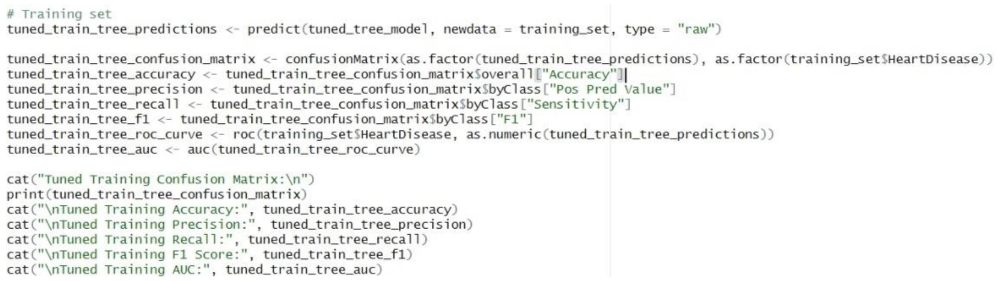


```
612 samples
 11 predictor
  2 classes: 'NoDisease', 'HeartDisease'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 489, 491, 489, 490, 489
Resampling results across tuning parameters:

  cp    ROC        Sens       Spec
  0.01  0.8691343  0.7730259  0.8835038
  0.02  0.8412047  0.7210342  0.9127025
  0.03  0.8400853  0.7210342  0.9184996
  0.04  0.8275385  0.7920335  0.8630435
  0.05  0.8275385  0.7920335  0.8630435
  0.06  0.8275385  0.7920335  0.8630435
  0.07  0.8275385  0.7920335  0.8630435
  0.08  0.8275385  0.7920335  0.8630435
  0.09  0.8275385  0.7920335  0.8630435
  0.10  0.8275385  0.7920335  0.8630435

ROC was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.01.
```

Figure 43. Summary of DT's Code for Tuning Implementation

For the model tuning we will use cp value, which helps with overfitting in DT, because before our analysis showed problems in that area. Ten values ranging from 0.01 to 0.1 were taken, the final value used for the model was 0.01. In the method = cv, cv stands for cross-validation and will help prevent overfitting on multiple subsests. classProbs = TRUE helps to keep probabilities estimated.

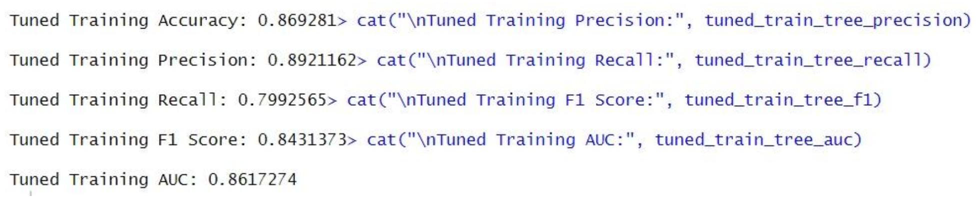
5.2.4. Tuning Performance



```
# Training set
tuned_train_tree_predictions <- predict(tuned_tree_model, newdata = training_set, type = "raw")

tuned_train_tree_confusion_matrix <- confusionMatrix(as.factor(tuned_train_tree_predictions), as.factor(training_set$HeartDisease))
tuned_train_tree_accuracy <- tuned_train_tree_confusion_matrix$overall["Accuracy"]
tuned_train_tree_precision <- tuned_train_tree_confusion_matrix$byClass["Pos Pred Value"]
tuned_train_tree_recall <- tuned_train_tree_confusion_matrix$byClass["Sensitivity"]
tuned_train_tree_f1 <- tuned_train_tree_confusion_matrix$byClass["F1"]
tuned_train_tree_roc_curve <- roc(training_set$HeartDisease, as.numeric(tuned_train_tree_predictions))
tuned_train_tree_auc <- auc(tuned_train_tree_roc_curve)

cat("Tuned Training Confusion Matrix:\n")
print(tuned_train_tree_confusion_matrix)
cat("\nTuned Training Accuracy:", tuned_train_tree_accuracy)
cat("\nTuned Training Precision:", tuned_train_tree_precision)
cat("\nTuned Training Recall:", tuned_train_tree_recall)
cat("\nTuned Training F1 Score:", tuned_train_tree_f1)
cat("\nTuned Training AUC:", tuned_train_tree_auc)
```

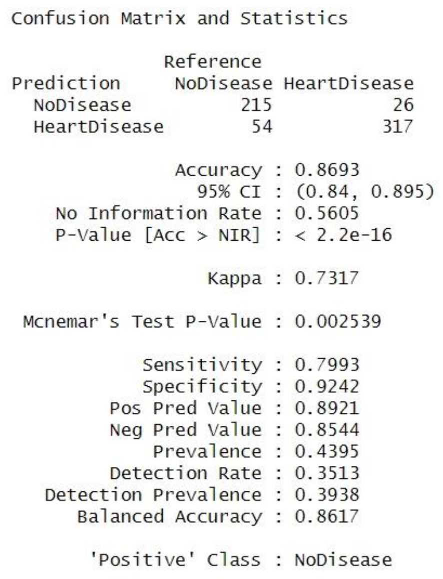
Figure 44. Code for DT's Tuning Performance on Training Set

Figure 44 shows the code for DT's tuning performance on training set. The results are show below:



```
Tuned Training Accuracy: 0.869281> cat("\nTuned Training Precision:", tuned_train_tree_precision)

Tuned Training Precision: 0.8921162> cat("\nTuned Training Recall:", tuned_train_tree_recall)

Tuned Training Recall: 0.7992565> cat("\nTuned Training F1 Score:", tuned_train_tree_f1)

Tuned Training F1 Score: 0.8431373> cat("\nTuned Training AUC:", tuned_train_tree_auc)

Tuned Training AUC: 0.8617274
```

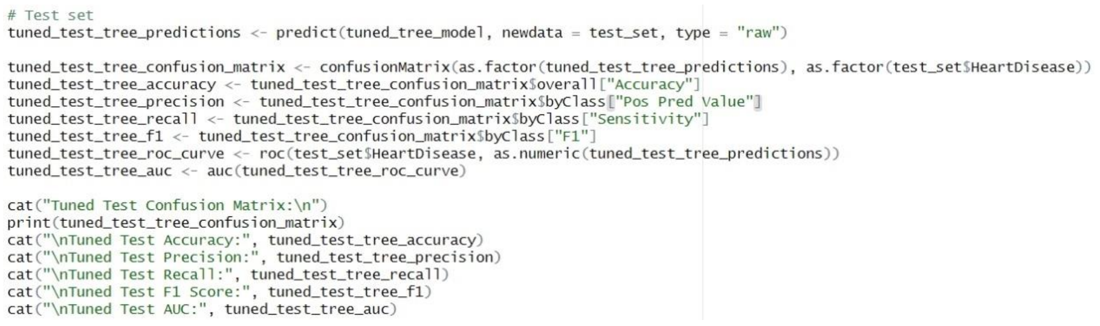
Figure 45. Output for DT’s Tuning Performance on Training Set



```
Confusion Matrix and Statistics

              Reference
Prediction     NoDisease HeartDisease
  NoDisease          215           26
  HeartDisease        54          317

               Accuracy : 0.8693
                 95% CI : (0.84, 0.895)
    No Information Rate : 0.5605
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7317

 Mcnemar's Test P-Value : 0.002539

            Sensitivity : 0.7993
            Specificity : 0.9242
         Pos Pred Value : 0.8921
         Neg Pred Value : 0.8544
             Prevalence : 0.4395
         Detection Rate : 0.3513
   Detection Prevalence : 0.3938
      Balanced Accuracy : 0.8617

       'Positive' Class : NoDisease
```

Figure 46. Confusion Matrix and Statistics for DT’s Tuning Performance on Training Set

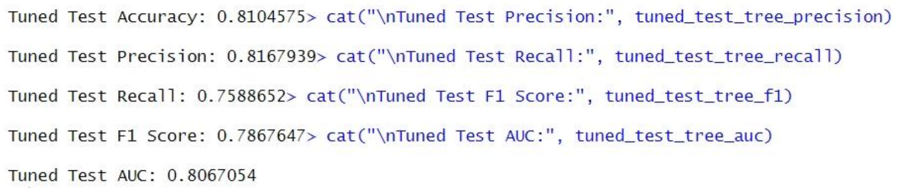
Comparing tuned accuracy on training set with the baseline accuracy on training set, we see that baseline’s accuracy are higher by 2%. Precision stayed pretty much the same, while there's gap between recall with baseline recall being 86% and tuned recall being at 79%. Baseline F1 score higher by 2%, baseline AUC higher by 3%. The huge difference between specificity 92.42% and sensitivity 79.93% could show the imbalanced nature of dataset. It is not good, and for medical observations it is better to see high sensitivity, as we want to be more focused on people with heart diseases.



```
# Test set
tuned_test_tree_predictions <- predict(tuned_tree_model, newdata = test_set, type = "raw")

tuned_test_tree_confusion_matrix <- confusionMatrix(as.factor(tuned_test_tree_predictions), as.factor(test_set$HeartDisease))
tuned_test_tree_accuracy <- tuned_test_tree_confusion_matrix$overall["Accuracy"]
tuned_test_tree_precision <- tuned_test_tree_confusion_matrix$byClass["Pos Pred Value"]
tuned_test_tree_recall <- tuned_test_tree_confusion_matrix$byClass["Sensitivity"]
tuned_test_tree_f1 <- tuned_test_tree_confusion_matrix$byClass["F1"]
tuned_test_tree_roc_curve <- roc(test_set$HeartDisease, as.numeric(tuned_test_tree_predictions))
tuned_test_tree_auc <- auc(tuned_test_tree_roc_curve)

cat("Tuned Test Confusion Matrix:\n")
print(tuned_test_tree_confusion_matrix)
cat("\nTuned Test Accuracy:", tuned_test_tree_accuracy)
cat("\nTuned Test Precision:", tuned_test_tree_precision)
cat("\nTuned Test Recall:", tuned_test_tree_recall)
cat("\nTuned Test F1 Score:", tuned_test_tree_f1)
cat("\nTuned Test AUC:", tuned_test_tree_auc)
```

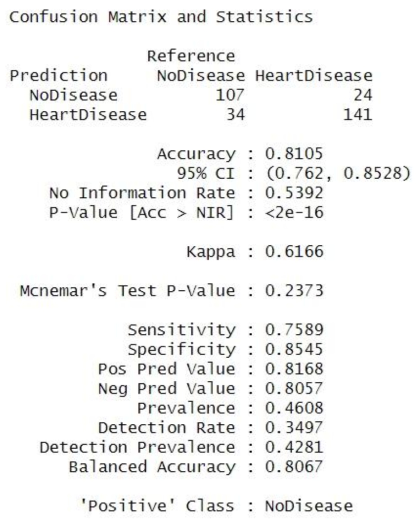
Figure 47. Code for DT’s Tuning Performance on Test Set

Code, in Figure 47, is for DT’s tuning performance on test set. Results are shown below:



```
Tuned Test Accuracy: 0.8104575> cat("\nTuned Test Precision:", tuned_test_tree_precision)

Tuned Test Precision: 0.8167939> cat("\nTuned Test Recall:", tuned_test_tree_recall)

Tuned Test Recall: 0.7588652> cat("\nTuned Test F1 Score:", tuned_test_tree_f1)

Tuned Test F1 Score: 0.7867647> cat("\nTuned Test AUC:", tuned_test_tree_auc)

Tuned Test AUC: 0.8067054
```

Figure 48. Output for DT’s Tuning Performance on Test Set



```
Confusion Matrix and Statistics

              Reference
Prediction     NoDisease HeartDisease
  NoDisease          107           24
  HeartDisease        34          141

               Accuracy : 0.8105
                 95% CI : (0.762, 0.8528)
    No Information Rate : 0.5392
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.6166

 Mcnemar's Test P-Value : 0.2373

            Sensitivity : 0.7589
            Specificity : 0.8545
         Pos Pred Value : 0.8168
         Neg Pred Value : 0.8057
             Prevalence : 0.4608
         Detection Rate : 0.3497
   Detection Prevalence : 0.4281
      Balanced Accuracy : 0.8067

       'Positive' Class : NoDisease
```

Figure 49. Confusion Matrix and Statistics for DT’s Tuning Performance on Test Set

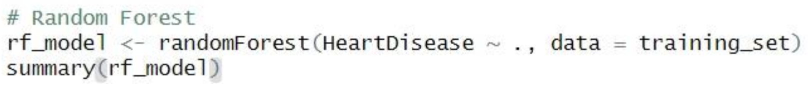
Looking at the results we can see a dropout in accuracy comparing training set and test set: 86.93% against 81.05%. Eventually, comparing baseline accuracy and tuning accuracy both on test set we see that they are identical with both being 81.05%. Other metrics, if we are comparing baseline and tuning performances on test set: precision 82.27% against 81.68%, recall 82.27%

against 75.88%, F1 score 80% against 78.68%, AUC 81.13% against 80.67%. One more noticeable change is the dropout of F1 score compared to tuning performance on training set, with F1 score on training set being 84.31%, while F1 score on test set 78.68%.

5.3. Random Forest

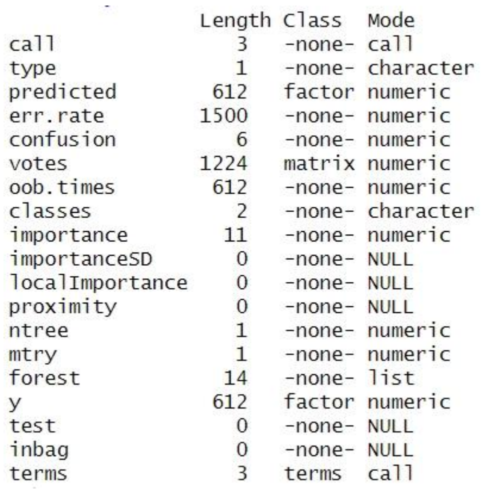
Let's finish our model implementation with our third model, which is Random Forest (RF).

5.3.1. Baseline Implementation



```
# Random Forest
rf_model <- randomForest(HeartDisease ~ ., data = training_set)
summary(rf_model)
```

Figure 50. Code for RF's Baseline Implementation

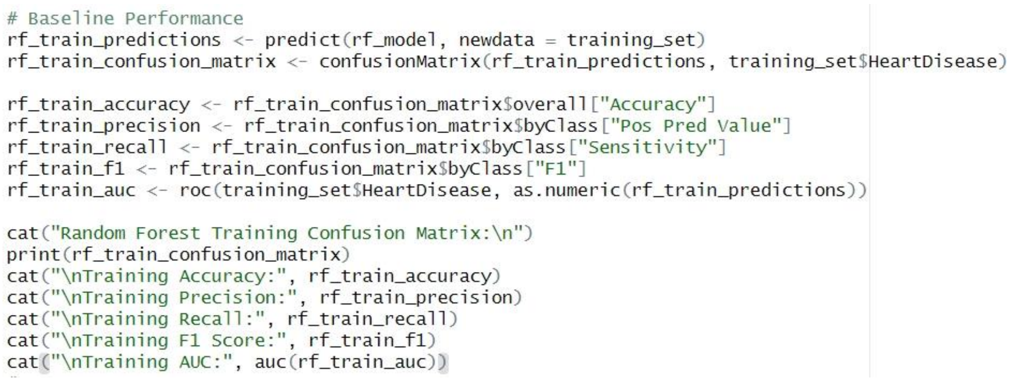


```
                Length Class  Mode
call               3   -none- call
type               1   -none- character
predicted        612   factor numeric
err.rate        1500   -none- numeric
confusion          6   -none- numeric
votes           1224   matrix numeric
oob.times        612   -none- numeric
classes            2   -none- character
importance        11   -none- numeric
importanceSD       0   -none- NULL
localImportance    0   -none- NULL
proximity          0   -none- NULL
ntree              1   -none- numeric
mtry               1   -none- numeric
forest            14   -none- list
y                612   factor numeric
test               0   -none- NULL
inbag              0   -none- NULL
terms              3   terms  call
```

Figure 51. Summary of RF's Baseline Implementation

Code, in Figure 50, creates an RF model that we can work with. Looking at 'ntree' we can say that model was built using one tree. 'classes' identifies that there are two classes in classification problem.

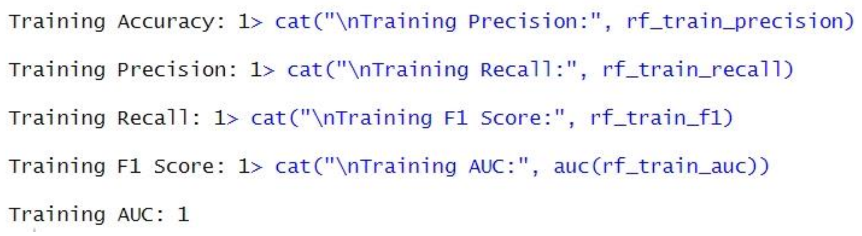
5.3.2. Baseline Performance



```
# Baseline Performance
rf_train_predictions <- predict(rf_model, newdata = training_set)
rf_train_confusion_matrix <- confusionMatrix(rf_train_predictions, training_set$HeartDisease)

rf_train_accuracy <- rf_train_confusion_matrix$overall["Accuracy"]
rf_train_precision <- rf_train_confusion_matrix$byClass["Pos Pred Value"]
rf_train_recall <- rf_train_confusion_matrix$byClass["Sensitivity"]
rf_train_f1 <- rf_train_confusion_matrix$byClass["F1"]
rf_train_auc <- roc(training_set$HeartDisease, as.numeric(rf_train_predictions))

cat("Random Forest Training Confusion Matrix:\n")
print(rf_train_confusion_matrix)
cat("\nTraining Accuracy:", rf_train_accuracy)
cat("\nTraining Precision:", rf_train_precision)
cat("\nTraining Recall:", rf_train_recall)
cat("\nTraining F1 Score:", rf_train_f1)
cat("\nTraining AUC:", auc(rf_train_auc))
```

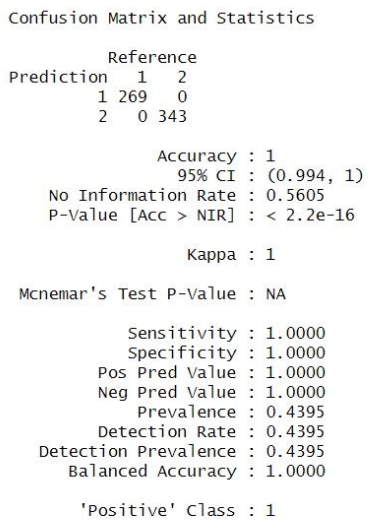
Figure 52. Code of RF’s Baseline Performance for Training Set

Code provided for RF’s baseline performance for training set. Results shown below:



```
Training Accuracy: 1> cat("\nTraining Precision:", rf_train_precision)

Training Precision: 1> cat("\nTraining Recall:", rf_train_recall)

Training Recall: 1> cat("\nTraining F1 Score:", rf_train_f1)

Training F1 Score: 1> cat("\nTraining AUC:", auc(rf_train_auc))

Training AUC: 1
```

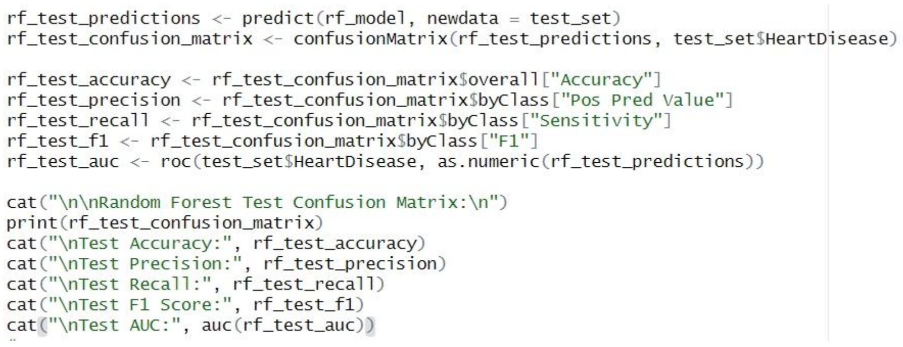
Figure 53. Output of RF’s Baseline Performance for Training Set



```
Confusion Matrix and Statistics

          Reference
Prediction   1   2
         1 269   0
         2   0 343

               Accuracy : 1
                 95% CI : (0.994, 1)
    No Information Rate : 0.5605
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.4395
         Detection Rate : 0.4395
   Detection Prevalence : 0.4395
      Balanced Accuracy : 1.0000

       'Positive' Class : 1
```

Figure 54. Confusion Matrix and Statistics of RF’s Baseline Performance for Training Set

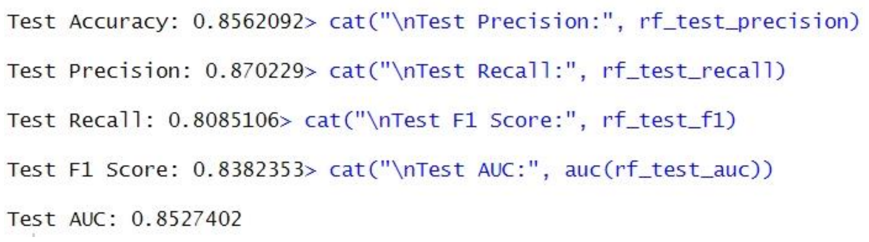
The model 100% results in every metric, predicting every case right, but sometimes perfect performances could be sign of overfitting.



```
rf_test_predictions <- predict(rf_model, newdata = test_set)
rf_test_confusion_matrix <- confusionMatrix(rf_test_predictions, test_set$HeartDisease)

rf_test_accuracy <- rf_test_confusion_matrix$overall["Accuracy"]
rf_test_precision <- rf_test_confusion_matrix$byClass["Pos Pred Value"]
rf_test_recall <- rf_test_confusion_matrix$byClass["Sensitivity"]
rf_test_f1 <- rf_test_confusion_matrix$byClass["F1"]
rf_test_auc <- roc(test_set$HeartDisease, as.numeric(rf_test_predictions))

cat("\n\nRandom Forest Test Confusion Matrix:\n")
print(rf_test_confusion_matrix)
cat("\nTest Accuracy:", rf_test_accuracy)
cat("\nTest Precision:", rf_test_precision)
cat("\nTest Recall:", rf_test_recall)
cat("\nTest F1 Score:", rf_test_f1)
cat("\nTest AUC:", auc(rf_test_auc))
```

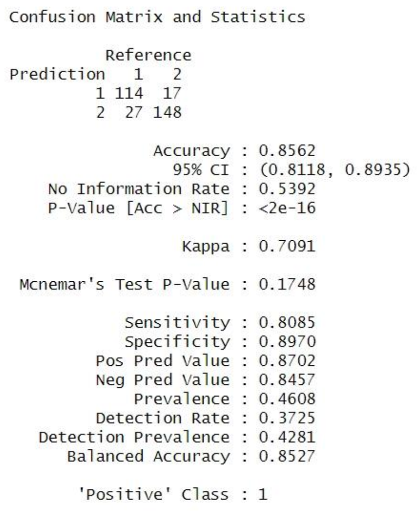
Figure 55. Code of RF’s Baseline Performance on Test Set

After applying this code, let’s look at the results on images below:



```
Test Accuracy: 0.8562092> cat("\nTest Precision:", rf_test_precision)

Test Precision: 0.870229> cat("\nTest Recall:", rf_test_recall)

Test Recall: 0.8085106> cat("\nTest F1 Score:", rf_test_f1)

Test F1 Score: 0.8382353> cat("\nTest AUC:", auc(rf_test_auc))

Test AUC: 0.8527402
```

Figure 56. Output of RF’s Baseline Performance on Test Set

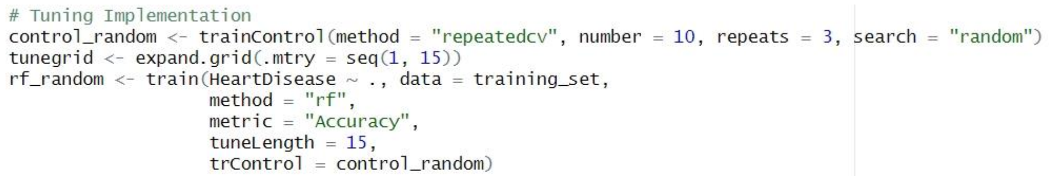


```
Confusion Matrix and Statistics

          Reference
Prediction   1   2
         1 114  17
         2  27 148

               Accuracy : 0.8562
                 95% CI : (0.8118, 0.8935)
    No Information Rate : 0.5392
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.7091

 Mcnemar's Test P-Value : 0.1748

            Sensitivity : 0.8085
            Specificity : 0.8970
         Pos Pred Value : 0.8702
         Neg Pred Value : 0.8457
             Prevalence : 0.4608
         Detection Rate : 0.3725
   Detection Prevalence : 0.4281
      Balanced Accuracy : 0.8527

       'Positive' Class : 1
```

Figure 57. Confusion Matrix and Statistics of RF’s Baseline Performance on Test Set

With the test set we can see more realistic performance, without perfect one. The accuracy of the model is 85.62%.

5.3.3. Tuning Implementation

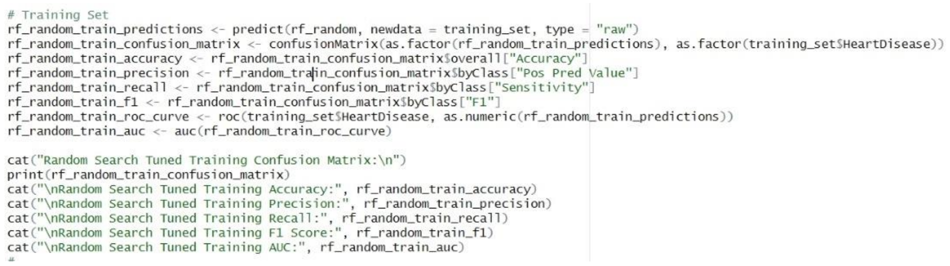


```
# Tuning Implementation
control_random <- trainControl(method = "repeatedcv", number = 10, repeats = 3, search = "random")
tunegrid <- expand.grid(.mtry = seq(1, 15))
rf_random <- train(HeartDisease ~ ., data = training_set,
                   method = "rf",
                   metric = "Accuracy",
                   tuneLength = 15,
                   trControl = control_random)
```

Figure 58. Code for RF's Tuning Implementation

Looking at 'mtry', explore values from 2 to the number of features minus 1, so to check performances with different numbers. 'nodesize = seq(1, 10, by = 1) explore values from 1 to 10, to control minimum number of observations in terminal nodes. 'ntree' from 100 to 500, to explore number of trees. So going to the building of tunned RF model – 'ntreeTry' is the number of trees to grow, 'stepFactor' controls how many mtry values to try, 'improve' improves classification error.

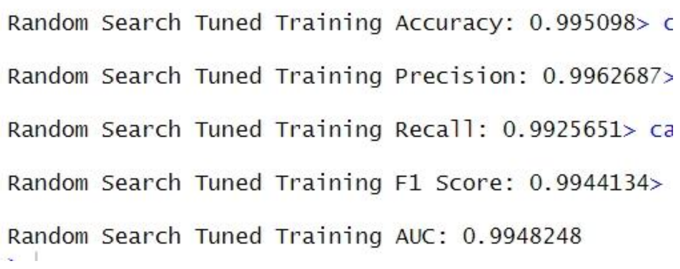
5.3.4. Tuning Performance



```
# Training Set
rf_random_train_predictions <- predict(rf_random, newdata = training_set, type = "raw")
rf_random_train_confusion_matrix <- confusionMatrix(as.factor(rf_random_train_predictions), as.factor(training_set$HeartDisease))
rf_random_train_accuracy <- rf_random_train_confusion_matrix$overall["Accuracy"]
rf_random_train_precision <- rf_random_train_confusion_matrix$byClass["Pos Pred Value"]
rf_random_train_recall <- rf_random_train_confusion_matrix$byClass["Sensitivity"]
rf_random_train_f1 <- rf_random_train_confusion_matrix$byClass["F1"]
rf_random_train_roc_curve <- roc(training_set$HeartDisease, as.numeric(rf_random_train_predictions))
rf_random_train_auc <- auc(rf_random_train_roc_curve)

cat("Random Search Tuned Training Confusion Matrix:\n")
print(rf_random_train_confusion_matrix)
cat("\nRandom Search Tuned Training Accuracy:", rf_random_train_accuracy)
cat("\nRandom Search Tuned Training Precision:", rf_random_train_precision)
cat("\nRandom Search Tuned Training Recall:", rf_random_train_recall)
cat("\nRandom Search Tuned Training F1 Score:", rf_random_train_f1)
cat("\nRandom Search Tuned Training AUC:", rf_random_train_auc)
#
```

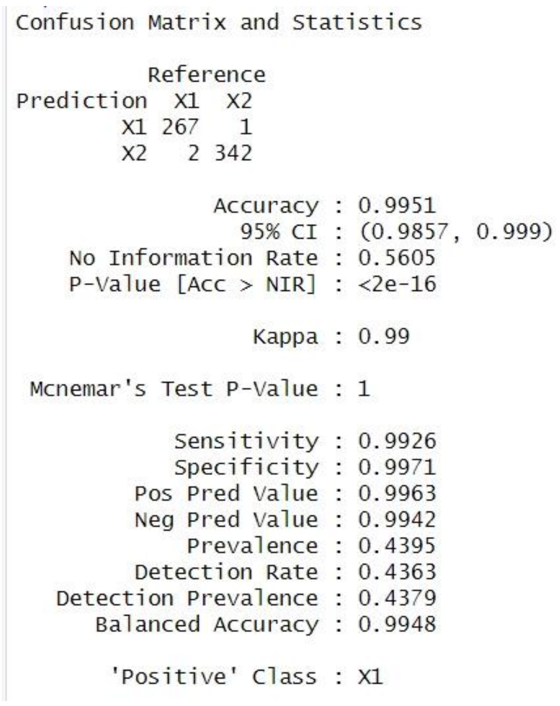
Figure 59. Code for RF's Tuning Performance on Training Set

The code created for RF's tuning performance on training set. The results shown below:



```
Random Search Tuned Training Accuracy: 0.995098> c

Random Search Tuned Training Precision: 0.9962687>

Random Search Tuned Training Recall: 0.9925651> ca

Random Search Tuned Training F1 Score: 0.9944134>

Random Search Tuned Training AUC: 0.9948248
```

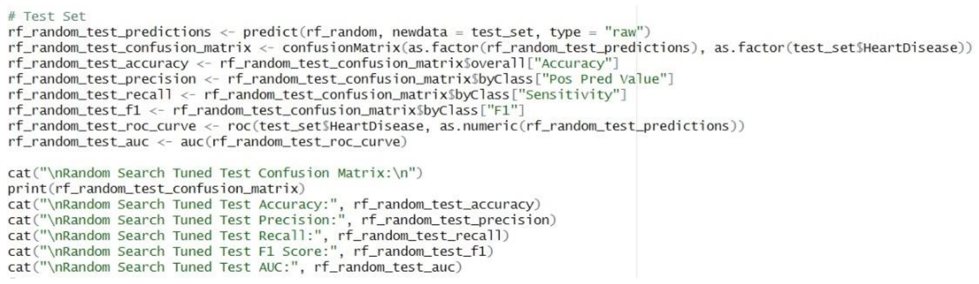
Figure 60. Output for RF's Tuning Performance on Training Set



```
Confusion Matrix and Statistics

          Reference
Prediction  X1  X2
        X1 267   1
        X2   2 342

               Accuracy : 0.9951
                 95% CI : (0.9857, 0.999)
    No Information Rate : 0.5605
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.99

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9926
            Specificity : 0.9971
         Pos Pred Value : 0.9963
         Neg Pred Value : 0.9942
             Prevalence : 0.4395
         Detection Rate : 0.4363
   Detection Prevalence : 0.4379
      Balanced Accuracy : 0.9948

       'Positive' Class : X1
```

Figure 61. Confusion Matrix and Statistics for RF's Tuning Performance on Training Set

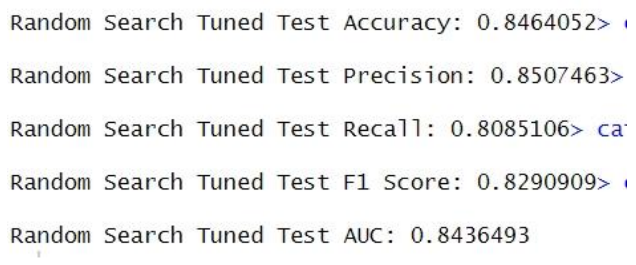
The accuracy of tuning model is 99.51% which is one of the highest accuracies noted in this document, but still less than baseline performance accuracy on training set with them being perfect 100%. The other metrics also around 99% which shows how good the model is, being almost perfect.



```
# Test Set
rf_random_test_predictions <- predict(rf_random, newdata = test_set, type = "raw")
rf_random_test_confusion_matrix <- confusionMatrix(as.factor(rf_random_test_predictions), as.factor(test_set$HeartDisease))
rf_random_test_accuracy <- rf_random_test_confusion_matrix$overall["Accuracy"]
rf_random_test_precision <- rf_random_test_confusion_matrix$byClass["Pos Pred Value"]
rf_random_test_recall <- rf_random_test_confusion_matrix$byClass["Sensitivity"]
rf_random_test_f1 <- rf_random_test_confusion_matrix$byClass["F1"]
rf_random_test_roc_curve <- roc(test_set$HeartDisease, as.numeric(rf_random_test_predictions))
rf_random_test_auc <- auc(rf_random_test_roc_curve)

cat("\nRandom Search Tuned Test Confusion Matrix:\n")
print(rf_random_test_confusion_matrix)
cat("\nRandom Search Tuned Test Accuracy:", rf_random_test_accuracy)
cat("\nRandom Search Tuned Test Precision:", rf_random_test_precision)
cat("\nRandom Search Tuned Test Recall:", rf_random_test_recall)
cat("\nRandom Search Tuned Test F1 Score:", rf_random_test_f1)
cat("\nRandom Search Tuned Test AUC:", rf_random_test_auc)
```

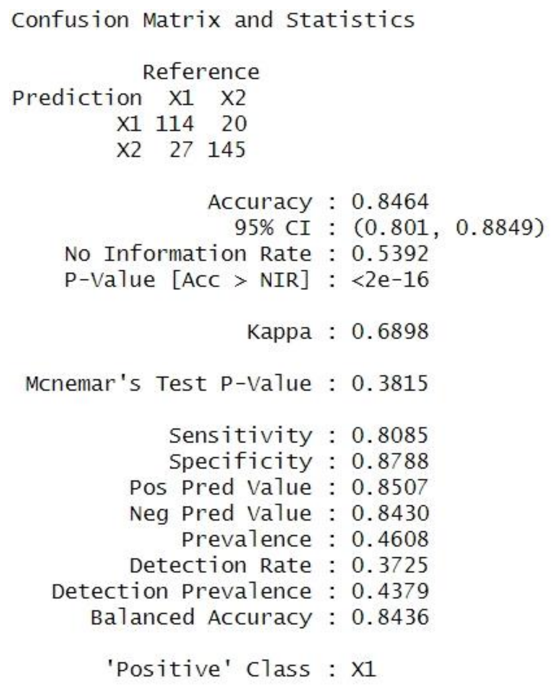
Figure 62. Code for RF's Tuning Performance on Test Set

The code for RF's tuning performance on test set, the results shown below:



```
Random Search Tuned Test Accuracy: 0.8464052>

Random Search Tuned Test Precision: 0.8507463>

Random Search Tuned Test Recall: 0.8085106> ca

Random Search Tuned Test F1 Score: 0.8290909>

Random Search Tuned Test AUC: 0.8436493
```

Figure 63. Output for RF’s Tuning Performance on Test Set



```
Confusion Matrix and Statistics

          Reference
Prediction  X1  X2
        X1 114  20
        X2  27 145

               Accuracy : 0.8464
                 95% CI : (0.801, 0.8849)
    No Information Rate : 0.5392
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.6898

 Mcnemar's Test P-Value : 0.3815

            Sensitivity : 0.8085
            Specificity : 0.8788
         Pos Pred Value : 0.8507
         Neg Pred Value : 0.8430
             Prevalence : 0.4608
         Detection Rate : 0.3725
   Detection Prevalence : 0.4379
      Balanced Accuracy : 0.8436

       'Positive' Class : X1
```

Figure 64. Confusion Matrix and Statistics for RF’s Tuning Performance on Test Set

The accuracy on test set is 84.64%, which is lower than accuracy on training set by 15%. We can see the same pattern of huge dropout in accuracy after switching from training set to test one, as baseline performance recorded the same dropout from training set to test set. The other metrics also dropped with recall getting the highest dropout in percentages out them all with almost 20%.

6. MODEL VALIDATION

Table 3. Results from All Models on Training Set

Name	Accuracy	Precision	Recall	F1 Score	AUC
Baseline LG	0.8856	0.8911	0.9067	0.8988	0.9455
Tuned LG	0.8840	0.8799	0.9184	0.8987	0.9419
Baseline DT	0.8905	0.8959	0.8607	0.8780	0.8911
Tuned DT	0.8692	0.8921	0.7992	0.8431	0.8617
Baseline RF	1.0000	1.0000	1.0000	1.0000	1.0000
Tuned RF	0.9951	0.9963	0.9926	0.9944	0.9948

Firstly, let's analyze results of all models based on the training set (Table 3). Out of 6 outputs, the best result was provided by RF’s model with baseline accuracy of 1.000 and tuned accuracy pf 0.9951. Alongside 1.0000 accuracy other metrics reached the same number, which defines baseline RF’s model as perfect one. Next to the perfect performance is tuned RF’s model with almost perfect performance, providing 99% close to 100% values from all metrics. The worst model is tuned DT’s model with accuracy 0.8692 and other metrics being the worst in their categories. With the best metrics reaching 1.0000 from baseline RF, let’s look at the worst metrics in the list below:

1. Worst Accuracy – 0.8692 (Tuned DT)
2. Worst Precision – 0.8799 (Tuned LG)
3. Worst Recall – 0.7992 (Tuned DT)
4. Worst F1 Score – 0.8431 (Tuned DT)
5. Worst AUC – 0.8617 (Tuned DT)

Table 4. Results from All Models on Test Set

Name	Accuracy	Precision	Recall	F1 Score	AUC
Baseline LG	0.8497	0.8521	0.8727	0.8623	0.9025
Tuned LG	0.8497	0.8563	0.8667	0.8614	0.9021
Baseline DT	0.8105	0.8227	0.7785	0.8000	0.8113

Tuned DT	0.8105	0.8168	0.7589	0.7867	0.8067
Baseline RF	0.8562	0.8702	0.8085	0.8382	0.8527
Tuned RF	0.8464	0.8507	0.8085	0.8291	0.8436

Let's analyze the results on the test set with Table 4. The noticeable thing is the overall drop in every metric for every model, but the distribution between them is less than for training sets. Distribution in accuracy between models on training set from 86% to 100%, while for test set is from 81% to 85%. The best model is baseline RF's model, once again with 0.8562 in accuracy, but after that comes baseline and tuned LG's models with 0.8497 accuracy. DT's models do not record any change in accuracy, but changes in other metrics with them still being the worst models. List for best metrics below:

1. Best Accuracy – 0.8562 (Baseline RF)
2. Best Precision – 0.8702 (Baseline RF)
3. Best Recall – 0.8727 (Baseline LG)
4. Best F1 Score – 0.8623 (Baseline LG)
5. Best AUC – 0.9025 (Baseline LG)

List for worst metrics below:

1. Worst Accuracy – 0.8105 (Baseline and Tuned DT)
2. Worst Precision – 0.8168 (Tuned DT)
3. Worst Recall – 0.7589 (Tuned DT)
4. Worst F1 Score – 0.7867 (Baseline DT)
5. Worst AUC – 0.8067 (Tuned DT)

Models perform better on training set and show dropout when it comes to test set. Let's look at the dropout comparing their accuracy:

1. Baseline LG: 88.56% to 84.97% (-3.59%)
2. Tuned LG: 88.40% to 84.97% (-3.43%)
3. Baseline DT: 89.05% to 81.05% (-8%)
4. Tuned DT: 86.92% to 81.05% (-5.87%)
5. Baseline RF: 100% to 85.62% (-14.38%)
6. Tuned RF: 99.51% to 84.64% (-14.87%)

The most negative change comes to RF's model, but still models are the best model for both training and test sets.

The next outcome is the dropout in values from baseline to tuned. It is pattern for every model, but LG’s models and DT’s models on test with them staying the same, basically no changes.

7. ANALYSIS AND RECOMMENDATIONS

Table 5. Comparing Logistic Regression Results of Related Works

Work	Name	Accuracy	Precision	Recall	F1 Score	AUC
(Ali et al., 2021)	Logistic Regression	0.8963	0.8960	0.8960	-	-
This study	Baseline LG	0.8497	0.8521	0.8727	0.8623	0.9025
	Tuned LG	0.8497	0.8563	0.8667	0.8614	0.9021

Table 6. Comparing Accuracies of This Study and Related Works

Study	Logistic Regression	Decision Tree	Random Forest
(Ali et at., 2021)	0.8963	1.0000	1.0000
(Shah et at., 2020)	-	0.8026	-
(Sharma et al., 2020)	0.9900	0.8500	-
(Katarya et al., 2020)	0.9340	0.8131	0.9560
(Singh et al., 2020)	0.7800	0.7900	-
This study (baseline)	0.8497	0.8105	0.8562
This study (tuned)	0.8497	0.8105	0.8464

Overall, the best model on the training set is RF's baseline model with perfect performance of 100%. The next close results are tuned RF with accuracy of 99.51%. The worst model, based on results, is tuned DT showing the lowest scores in accuracy, recall, F1 score and AUC. When it comes to train set, the best model remains the same with baseline RF's accuracy of 85.62%. The next close is baseline LG providing best scores in 3 metrics out of 5 with the best scores in recall, F1 score and AUC. Worst is tuned DT with the lowest scores in every metric.

All models experience drops in performance transitioning from training set to test set, which means that models overfit training data. Logistic Regression shows robust performance and the most stability.

Analyzing related works, DT or LG is expected to be the worst model, with DT being a little better, while RF is expected to become the best. As our results showed the expectation has been met. However, taking the test as the result and comparing this study to other related works that was mentioned (Table 6):

1. Logistic Regression models, from this study, did not provide worst accuracy but was the worst second and third.

2. Decision Tree models, from this study, did not provide worst accuracy and were in the middle with both baseline and tuned.

3. Random Forest models, from this study, had the worst accuracy but was compared to two studies that provided accuracy of 95.60% and 100%.

For recommendations, there is should be better regularization for every model to reduce overfitting, as they all experience a dropout.

8. CONCLUSION

At the end of the study, one of the main problems was overfitting, where each model performed worse when comparing training set and test set. Next, tuning problems, where tuning models performed worse than baseline. However, the models showed good results where Random Forest was the best algorithm and even showed a perfect result.

By conducting such a study, the main plus point was to gain new knowledge in using different algorithms and by conducting a review on an existing study that used other algorithms as well.

For future work, it is worth investigating other additional characteristics that may affect performance, which will affect ignoring. Also, it is worth paying attention to tuning - it is much better to perform tuning to improve results rather than to degrade them. Learn more techniques to improve the reliability of the study.

REFERENCES

Ali, M. M., Paul, B. K., Ahmed, K., Bui, F. M., Quinn, J. M., & Moni, M. A. (2021). Heart disease prediction using supervised machine learning algorithms: Performance analysis and comparison. *Computers in Biology and Medicine*, *136*, 104672.

fedesoriano. (September 2021). Heart Failure Prediction Dataset. Retrieved [Date Retrieved] from https://www.kaggle.com/fedesoriano/heart-failure-prediction

Katarya, R., & Meena, S. K. (2021). Machine learning techniques for heart disease prediction: a comparative study and analysis. *Health and Technology*, *11*, 87-97.

Singh, A., & Kumar, R. (2020, February). Heart disease prediction using machine learning algorithms. In *2020 international conference on electrical and electronics engineering (ICE3)* (pp. 452-457). IEEE.

Sharma, V., Yadav, S., & Gupta, M. (2020, December). Heart disease prediction using machine learning techniques. In *2020 2nd international conference on advances in computing, communication control and networking (ICACCCN)* (pp. 177-181). IEEE.

Shah, D., Patel, S., & Bharti, S. K. (2020). Heart disease prediction using machine learning techniques. *SN Computer Science*, *1*, 1-6.

Tsao, C. W., Aday, A. W., Almarzooq, Z. I., Alonso, A., Beaton, A. Z., Bittencourt, M. S., ... & American Heart Association Council on Epidemiology and Prevention Statistics, Committee and Stroke Statistics Subcommittee. (2022). Heart disease and stroke statistics—2022 update: a report from the American Heart Association. *Circulation*, *145*(8), e153-e639.

Wikipedia contributors (2022). Support Vector Machine. Wikipedia. https://en.wikipedia.org/wiki/Support_vector_machine

World Health Organization (WHO). (2021). Cardiovascular Diseases (CVDs). https://www.who.int/en/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds)