IST 707 Data Analytics HOMEWORK 3: Disease Prediction for Patients using NBC, KNN, SVM and Ensemble Learning methods

Submitted by BHAVISH KUMAR on April 5th 2020

A. Executive Summary:

The purpose of this assignment is to use the several classification techniques that we learned in the course to help solve the binary classification problem of predicting if a patient has the disease or not. The goal is to accurately predict if a given patient has the disease or not, thereby producing very good Accuracy, Recall and AUROC scores. Six different Supervised Machine Learning algorithms were used to build models on the training dataset (dataset which contains the Target variable 'Disease' with its 2 classes) and these models were later used to make predictions on the new test dataset. The six Machine Learning algorithms that were used to achieve this outcome were K Nearest Neighbours, Naive Bayes Classifier, Linear Support Vector Machines, Non Linear Support Vector Machines and the 2 Ensemble methods used were Gradient Boosting and Random Forest. The goal of each model is to produce unbiased and low variance predictions which was achieved by extensive hyperparameter tuning done through a grid search. Several model evaluation techniques, which we will observe in the upcoming sections of the report were used to produce the best model.

B. INTRODUCTION:

In the below Analysis the following Machine Learning Algorithms have been built: **i. K Nearest Neighbors: This is a distance based lazy evaluation algorithm that considers the class of the 'K' nearest training data neighbors of a test data point to classify every test datapoint. The number of neighbors to be considered 'K' is a hyperparameter that can be tuned. ii. Naive Bayes Classifier: This is a probability based classifier that uses Conditional Probabilites, Prior Probability and Evidence to calculate the probability of a test data point for each of the two classes. Bayes Theorem is used to calculate the probability of a test data belonging to each of the class. Laplace smoothing is used to avoid 0 probability issue and laplace is a hyperparameter that can be tuned iii. Random Forest: This is an ensemble method consisting of multiple decision trees which combines the predictions made by multiple decison trees and each tree is generated using a selected number of input features. The number of features required for each decision tree and the number of decision trees are the two hyperparameters that can be tuned. iv. Gradient Boosting: This is also an ensemble method consisting of multiple decision trees where each decision tree is built sequentially one after the other and the final ensemble model is produced by taking the weighted average of predictions made by each base classifier.The depth of each Tree, the number of trees the learning rate and the minimum number of data points needed in each node for splitting are the hyperparameters that can be tuned. v. Support Vector Machines: This is a distance based classification algorithm that uses lines or hyperplanes to classify the datapoints by producing a seperator to easily classify the data points. The cost function, gamma and kernel are the hyperparameters that can be tuned** *The following techniques were used for model performance evaluation:* **K fold Cross Validation was done while training each of the models on train data to control for overfitting, so that the model is built by ensuring that it performs well not just on train data but also on validation data which it has not seen before. Holdout method was also used where 70% of the data was used to train the model using K fold Cross Validation and the remaining 30%**

was used to measure the model performance on the data that it has not seen before. The model performance has been measured by using Accuracy, Precison, Recall, F1 score metrics, ROC curve and Area Under ROC curve

C. Body of the Report:

Reading the Training data csv and storing it into a dataframe

```
setwd("C:/Users/bhavi/OneDrive/Desktop/SYR ADS/Sem 2/IST_707_Data_Analytics/HW3")

getwd
```

```
## function ()
## .Internal(getwd())
## <bytecode: 0x0000019724700e38>
## <environment: namespace:base>
```

```
disease_prediction_training <- read.csv("Disease Prediction Training.csv")
```

VIEWING THE STRUCTURE and SUMMARY STATISTICS of the Data and checking for missing values

```
str(disease_prediction_training)
```

```
## 'data.frame':    49000 obs. of  13 variables:
##  $ Age                : int  59 64 41 50 39 54 48 51 42 41 ...
##  $ Gender             : Factor w/ 2 levels "female","male": 1 1 1 2 1 1 1 1 1 1 ...
##  $ Height             : int  167 150 166 172 162 163 159 171 161 159 ...
##  $ Weight             : num  88 71 83 110 61 61 89 71 72 43 ...
##  $ High.Blood.Pressure: int  130 140 100 130 110 120 150 110 150 90 ...
##  $ Low.Blood.Pressure : int  68 100 70 80 80 80 90 70 90 60 ...
##  $ Cholesterol        : Factor w/ 3 levels "high","normal",..: 2 2 2 2 1 2 1 2 1 2 ...
##  $ Glucose            : Factor w/ 3 levels "high","normal",..: 2 2 2 2 1 2 1 2 1 2 ...
##  $ Smoke              : int  0 0 0 1 0 0 0 0 0 0 ...
##  $ Alcohol            : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ Exercise           : int  1 0 1 1 1 1 1 1 1 1 ...
##  $ Disease            : int  0 1 0 0 0 0 1 0 1 0 ...
##  $ bmi                : num  31.6 31.6 30.1 37.2 23.2 ...
```

```
summary(disease_prediction_training)
```

```
##       Age           Gender          Height          Weight       High.Blood.Pressure Low.Bloo
d.Pressure   Cholesterol        Glucose          Smoke
##  Min.   :29.00   female:31863   Min.   : 55.0   Min.   : 10.00   Min.   : -150.0     Min.   :
0.00   high    : 6705   high    : 3627   Min.   :0.00000
##  1st Qu.:48.00   male  :17137   1st Qu.:159.0   1st Qu.: 65.00   1st Qu.:  120.0     1st Qu.:
80.00   normal  :36676   normal  :41652   1st Qu.:0.00000
##  Median :53.00                   Median :165.0   Median : 72.00   Median :  120.0     Median :
80.00   too high: 5619   too high: 3721   Median :0.00000
##  Mean   :52.85                   Mean   :164.4   Mean   : 74.19   Mean   :  128.7     Mean   :
96.92                                    Mean   :0.08827
##  3rd Qu.:58.00                   3rd Qu.:170.0   3rd Qu.: 82.00   3rd Qu.:  140.0     3rd Qu.:
90.00                                    3rd Qu.:0.00000
##  Max.   :64.00                   Max.   :207.0   Max.   :200.00   Max.   :14020.0     Max.   :
11000.00                                    Max.   :1.00000
##     Alcohol           Exercise         Disease         bmi
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.0   Min.   :  3.472
##  1st Qu.:0.00000   1st Qu.:1.0000   1st Qu.:0.0   1st Qu.: 23.875
##  Median :0.00000   Median :1.0000   Median :0.0   Median : 26.398
##  Mean   :0.05424   Mean   :0.8032   Mean   :0.5   Mean   : 27.550
##  3rd Qu.:0.00000   3rd Qu.:1.0000   3rd Qu.:1.0   3rd Qu.: 30.164
##  Max.   :1.00000   Max.   :1.0000   Max.   :1.0   Max.   :298.667
```

SECTION 1: DATA PREPARATION & Exploratoy Data Analysis

1. Identifying the Data Quality Issues:

As we can see from the above summary that the data has no missing values and hence NA imputation is not required However, we can observe issues with 2 columns Low Blood Pressure and High Blood Pressure. From the structure and summary of the data we can observe that the Min and Max values of the columns Low Blood Pressure and High Blood Pressure are not practically possible values and hence they are noise/outliers which need to be treated. Hence these columns need to be winsorized. Winsorization is a data treatment process where the extreme outlier values are replaced with less extreme values which are practically possible

```
quantile(disease_prediction_training$Low.Blood.Pressure,c(0.001))
```

```
## 0.1%
##   45
```

```
quantile(disease_prediction_training$Low.Blood.Pressure,c(0.986))
```

```
## 98.6%
##   140
```

From the above 0.1 & 98.6 percentile values of low BP column we can observe that the possible values for the min & max of low BP (diastolic BP) fall in the range of 45 to 140 and hence any value that is less than 45 is replaced with 45 and any value greater than 140 is replaced with 140

```
disease_prediction_training$Low.Blood.Pressure[disease_prediction_training$Low.Blood.Pressure<qu
antile(disease_prediction_training$Low.Blood.Pressure,c(0.001))] <- quantile(disease_prediction_
training$Low.Blood.Pressure,c(0.001))

disease_prediction_training$Low.Blood.Pressure[disease_prediction_training$Low.Blood.Pressure>qu
antile(disease_prediction_training$Low.Blood.Pressure,c(0.986))] <- quantile(disease_prediction_
training$Low.Blood.Pressure,c(0.986))
```

Verifying that the Min & Max values of Low Blood Pressure (Diastolic BP) are in the correct practically permissible range and the outliers have been eliminated

```
summary(disease_prediction_training$Low.Blood.Pressure)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   45.00   80.00   80.00   82.15   90.00  140.00
```

The high Blood Pressure column also needs to winsorized to ensure that the values fall in the practically permissible range and outliers are eliminated

```
quantile(disease_prediction_training$High.Blood.Pressure,c(0.003))
```

```
## 0.3%
##   70
```

```
quantile(disease_prediction_training$High.Blood.Pressure,c(0.998))
```

```
## 99.8%
##   200
```

From the above 0.3 & 99.8 percentile values of High BP column we can observe that the possible values for the min & max of High BP (Systolic BP) fall in the range of 70 to 200 and hence any value that is less than 70 is replaced with 70 and any value greater than 200 is replaced with 200

```
disease_prediction_training$High.Blood.Pressure[disease_prediction_training$High.Blood.Pressure<
quantile(disease_prediction_training$High.Blood.Pressure,c(0.003))] <- quantile(disease_predicti
on_training$High.Blood.Pressure,c(0.003))

disease_prediction_training$High.Blood.Pressure[disease_prediction_training$High.Blood.Pressure>
quantile(disease_prediction_training$High.Blood.Pressure,c(0.998))] <- quantile(disease_predicti
on_training$High.Blood.Pressure,c(0.998))
```

Verifying that the Min & Max values of High Blood Pressure (Systolic BP) are in the correct practically permissible range and the outliers have been eliminated

```
summary(disease_prediction_training$High.Blood.Pressure)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    70.0   120.0   120.0   126.9   140.0   200.0
```

There are 245 instances where Low BP is > high BP even after winsorizing which needs to treated by swapping the values

```
length(disease_prediction_training[disease_prediction_training$Low.Blood.Pressure>disease_predic
tion_training$High.Blood.Pressure,1])
```

```
## [1] 245
```

Swapping the values wherever Low BP > High BP which is not permissible

```
low_bp_values <- disease_prediction_training$Low.Blood.Pressure[disease_prediction_training$Low.
Blood.Pressure>disease_prediction_training$High.Blood.Pressure]

high_bp_values <- disease_prediction_training$High.Blood.Pressure[disease_prediction_training$Lo
w.Blood.Pressure>disease_prediction_training$High.Blood.Pressure]

disease_prediction_training$Low.Blood.Pressure[disease_prediction_training$Low.Blood.Pressure>di
sease_prediction_training$High.Blood.Pressure] <- high_bp_values

disease_prediction_training$High.Blood.Pressure[disease_prediction_training$Low.Blood.Pressure>d
isease_prediction_training$High.Blood.Pressure] <- low_bp_values
```

Verifying that there are no instances with low bp values > high bp values

```
length(disease_prediction_training[disease_prediction_training$Low.Blood.Pressure>disease_predic
tion_training$High.Blood.Pressure,1])
```

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

The weight column has very low values, two of which are as low as 10Kg and 11Kg, which are practically very unlikely and hence they need to be winsorized

```
quantile(disease_prediction_training$Weight,c(0.0001))
```

```
##   0.01%
## 28.8999
```

Any value that is less than 0.01 percentile value, are replaced with the 0.01 percentile value = 28.9

```
disease_prediction_training$Weight[disease_prediction_training$Weight<quantile(disease_predictio
n_training$Weight,c(0.0001))] <- quantile(disease_prediction_training$Weight,c(0.0001))
```

Verifying that the Min & Max values of Weight are in the correct practically permissible range and the outliers have been eliminated

```
summary(disease_prediction_training$Weight)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   28.90   65.00   72.00   74.19   82.00  200.00
```

2. EXPLORATORY DATA ANALYSIS:

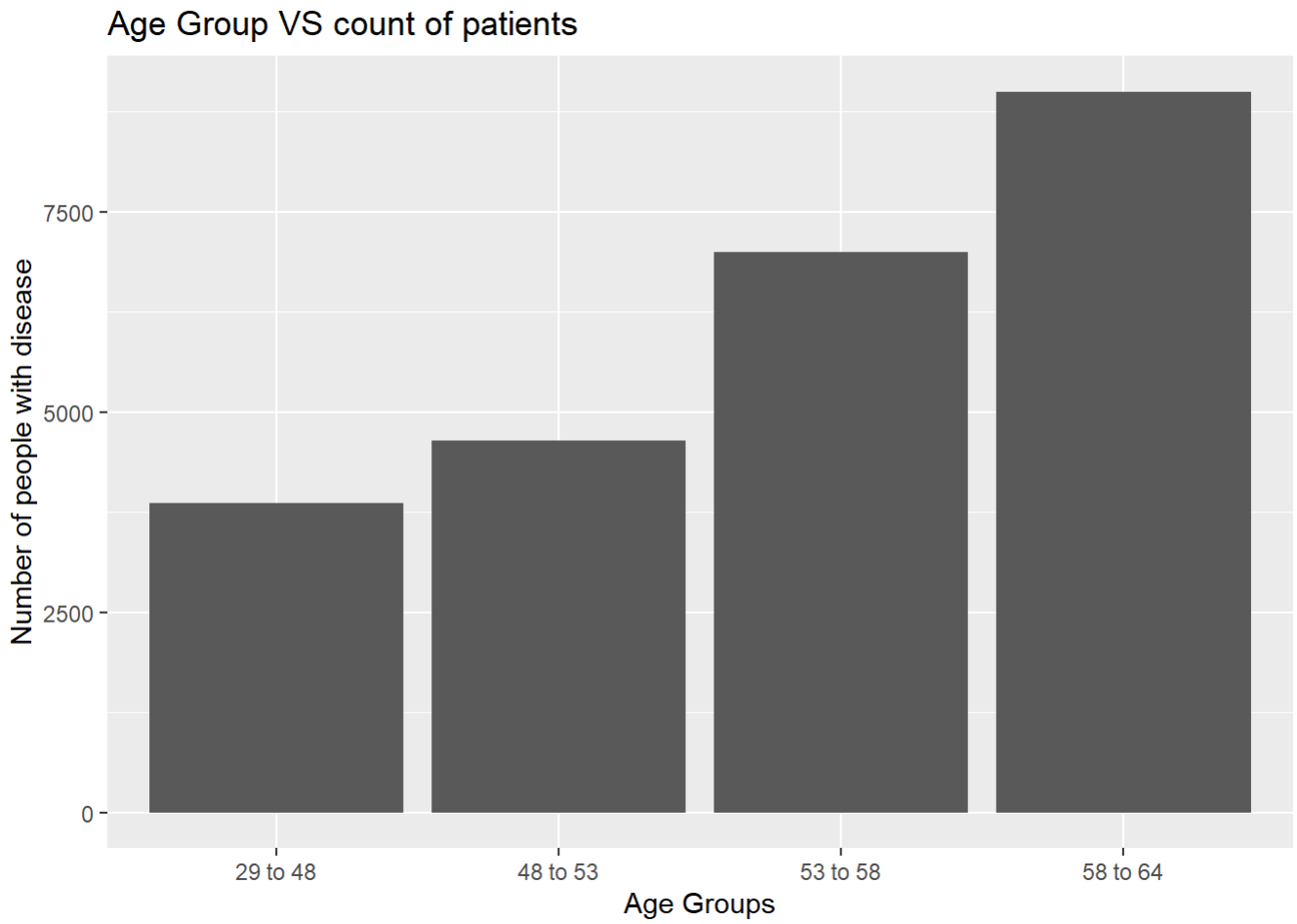
2.1. Bi Variate Analysis between Age and Disease column

Creating Age Groups column based on quartiles for EDA purpose **We assume that the number of people with the disease is higher for higher age groups, which we can verify by producing a bar graph**

```
library(stringr)
disease_prediction_training$age_groups <-cut(disease_prediction_training$Age, breaks = c(quantil
e(disease_prediction_training$Age, probs = c(0,0.25,0.5,0.75,1))),
     labels = c(str_c(quantile(disease_prediction_training$Age,probs = 0),quantile(disease_predi
ction_training$Age,probs = 0.25),sep = " to "),str_c(quantile(disease_prediction_training$Age,pr
obs = 0.25),quantile(disease_prediction_training$Age,probs = 0.5),sep = " to "),str_c(quantile(d
isease_prediction_training$Age,probs = 0.5),quantile(disease_prediction_training$Age,probs = 0.7
5),sep = " to "),str_c(quantile(disease_prediction_training$Age,probs = 0.75),quantile(disease_p
rediction_training$Age,probs = 1),sep = " to ")), right = FALSE, include.lowest=TRUE)
disease_prediction_training$age_groups<-as.factor(disease_prediction_training$age_groups)
#unique(disease_prediction_training$age_groups)
```

From the below bar group our assumption has been verified, as we can observe that the number of people with the disease increases as we go up the age groups and the older age groups have the highest number of patients with the disease

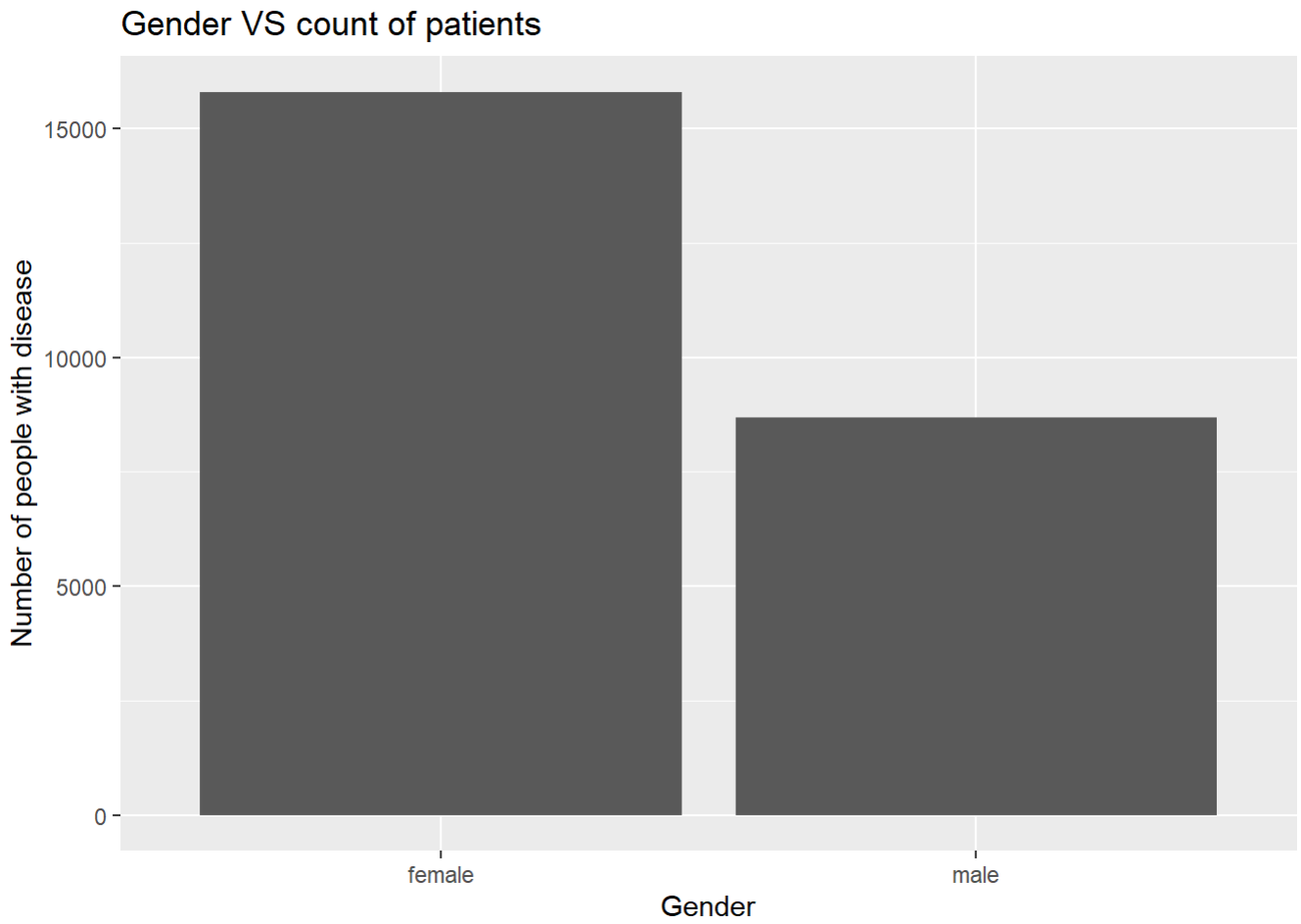
```
library(tidyverse)
library(ggplot2)
disease_count_by_ageGroups <- disease_prediction_training %>%
      group_by(age_groups)%>%
      summarise(sum(Disease))
colnames(disease_count_by_ageGroups) <- c('age_groups','NO_ppl_with_disease')
age_group_disease_plot <- ggplot(disease_count_by_ageGroups,aes(age_groups,NO_ppl_with_disease))
+geom_bar(stat = "identity")+
             xlab("Age Groups")+ ylab("Number of people with disease")+ ggtitle("Age Group VS co
unt of patients")
age_group_disease_plot
```



2.2. Bi Variate Analysis between Gender and Disease column

We can observe that there are more number of female patients with the disease than males

```
disease_count_by_gender <- disease_prediction_training %>%
      group_by(Gender)%>%
      summarise(sum(Disease))
colnames(disease_count_by_gender) <- c('Gender','NO_ppl_with_disease')

gender_disease_plot <- ggplot(disease_count_by_gender,aes(Gender,NO_ppl_with_disease))+geom_bar
(stat = "identity")+
            xlab("Gender")+ ylab("Number of people with disease")+ ggtitle("Gender VS count of
 patients")
gender_disease_plot
```



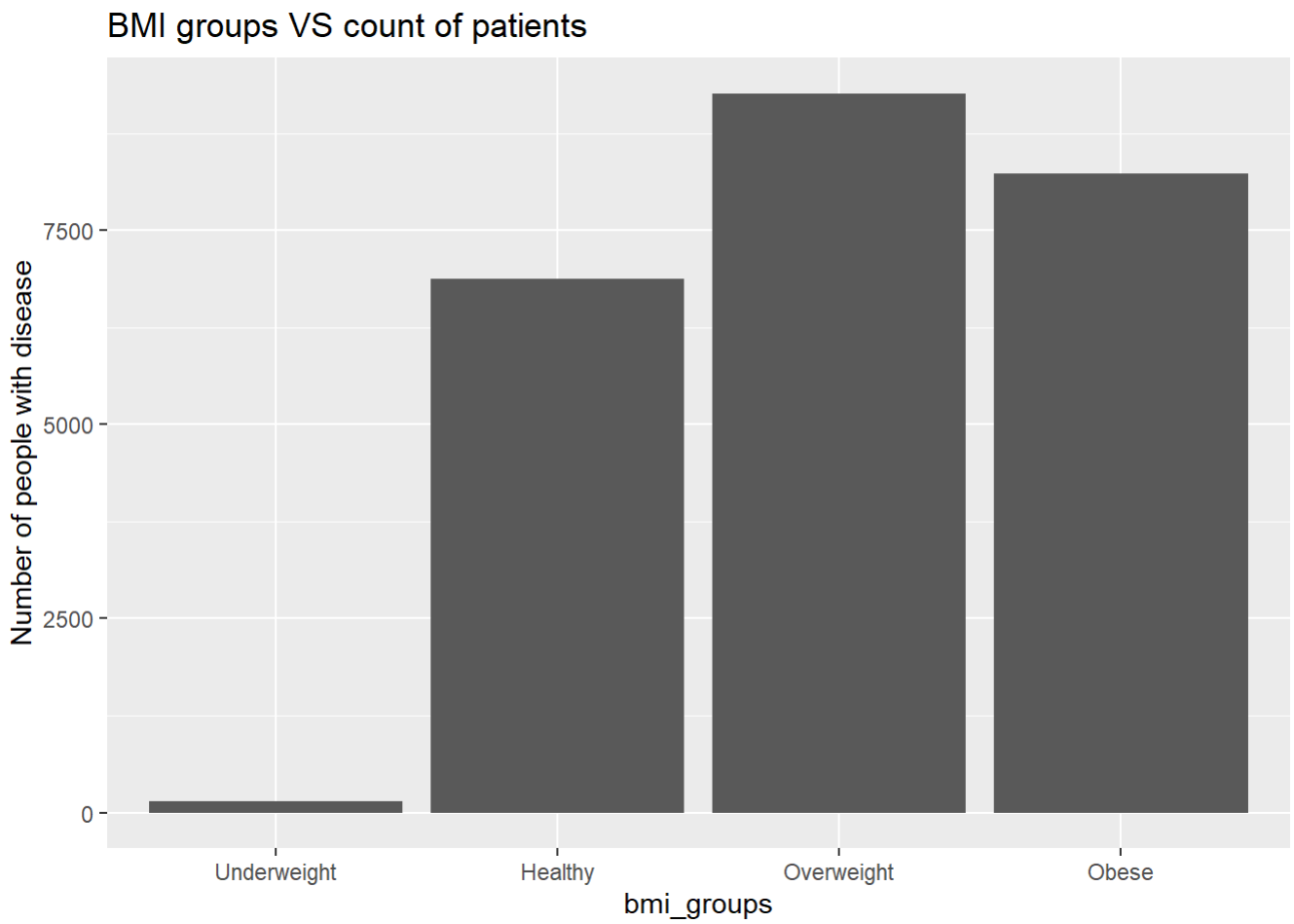
2.3. Bi Variate Analysis between Height+Weight (BMI) and Disease column

Creating a new Body Mass Index (BMI) column by combining Height and Weight column, where BMI = (Weight in Kg)/(Height in meteres)^2 The BMI column can be used to classify the patients as Underweight, Healthy, Overweight and Obese Underwight if BMI < 18.5; Healthy if BMI between 18.5 and 24.9; Overweight if BMI between 25 and 29.9; Obese if BMI greater than 30

```
disease_prediction_training$bmi <- disease_prediction_training$Weight/((disease_prediction_train
ing$Height/100)*(disease_prediction_training$Height/100))
disease_prediction_training$bmi_groups <-cut(disease_prediction_training$bmi, breaks = c(0,18.5,
24.9,29.9,Inf), labels = c('Underweight','Healthy','Overweight','Obese'))
```

We can observe that number of people with the disease is more for Overweight and Obese BMI groups in comparison to Healthy and Underweight BMI groups

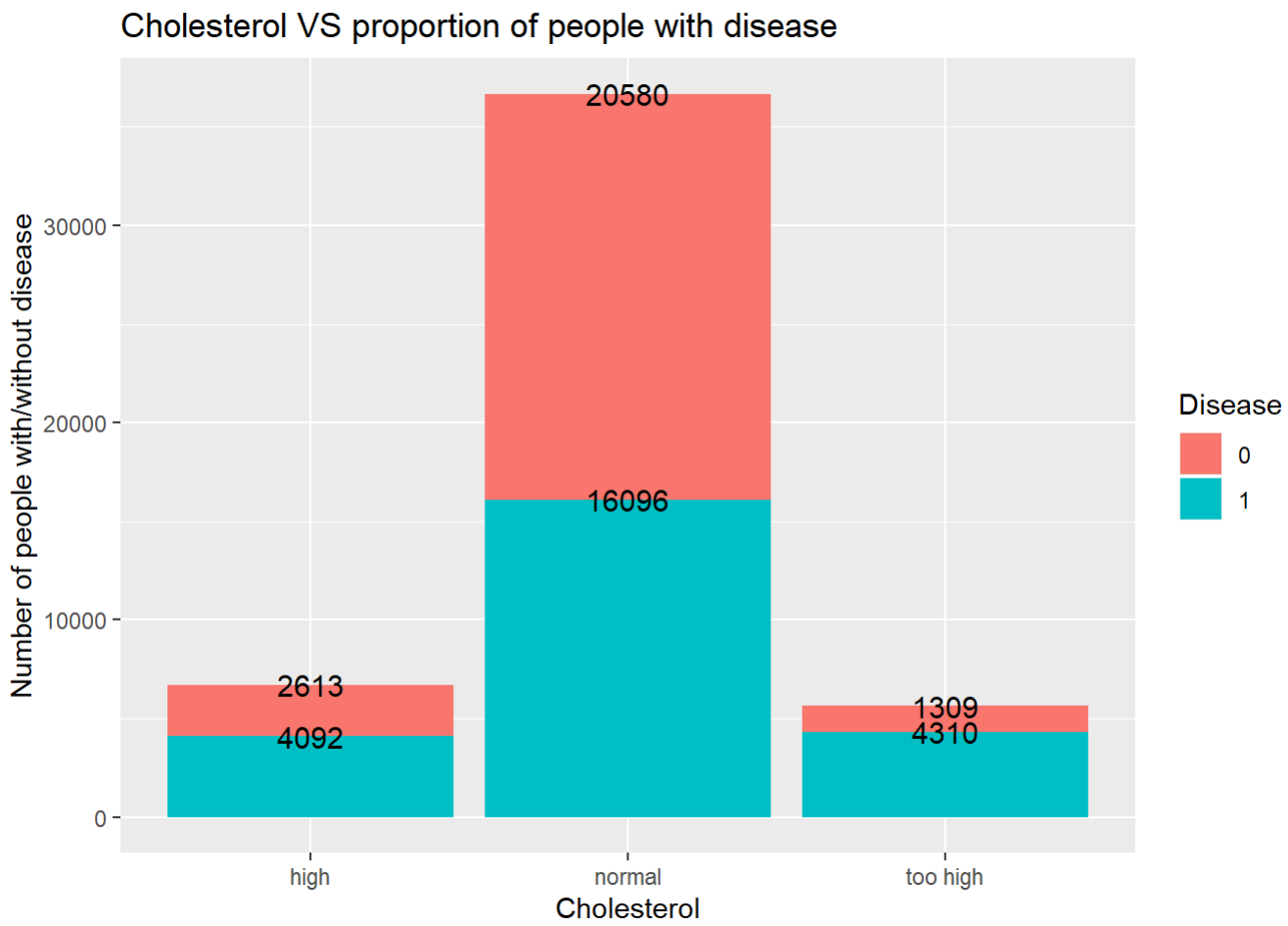
```
disease_count_by_bmi_groups <- disease_prediction_training %>%
      group_by(bmi_groups)%>%
      summarise(sum(Disease))
colnames(disease_count_by_bmi_groups) <- c('bmi_groups','NO_ppl_with_disease')

bmi_disease_plot <- ggplot(disease_count_by_bmi_groups,aes(bmi_groups,NO_ppl_with_disease))+geom
_bar(stat = "identity")+
             xlab("bmi_groups")+ ylab("Number of people with disease")+ ggtitle("BMI groups VS c
ount of patients")
bmi_disease_plot
```



2.4. Bi Variate Analysis between Cholestrol and Disease column

We can observe that the percentage of people with the disease is much higher amongst 'high' and 'too high' Cholesterol groups of people

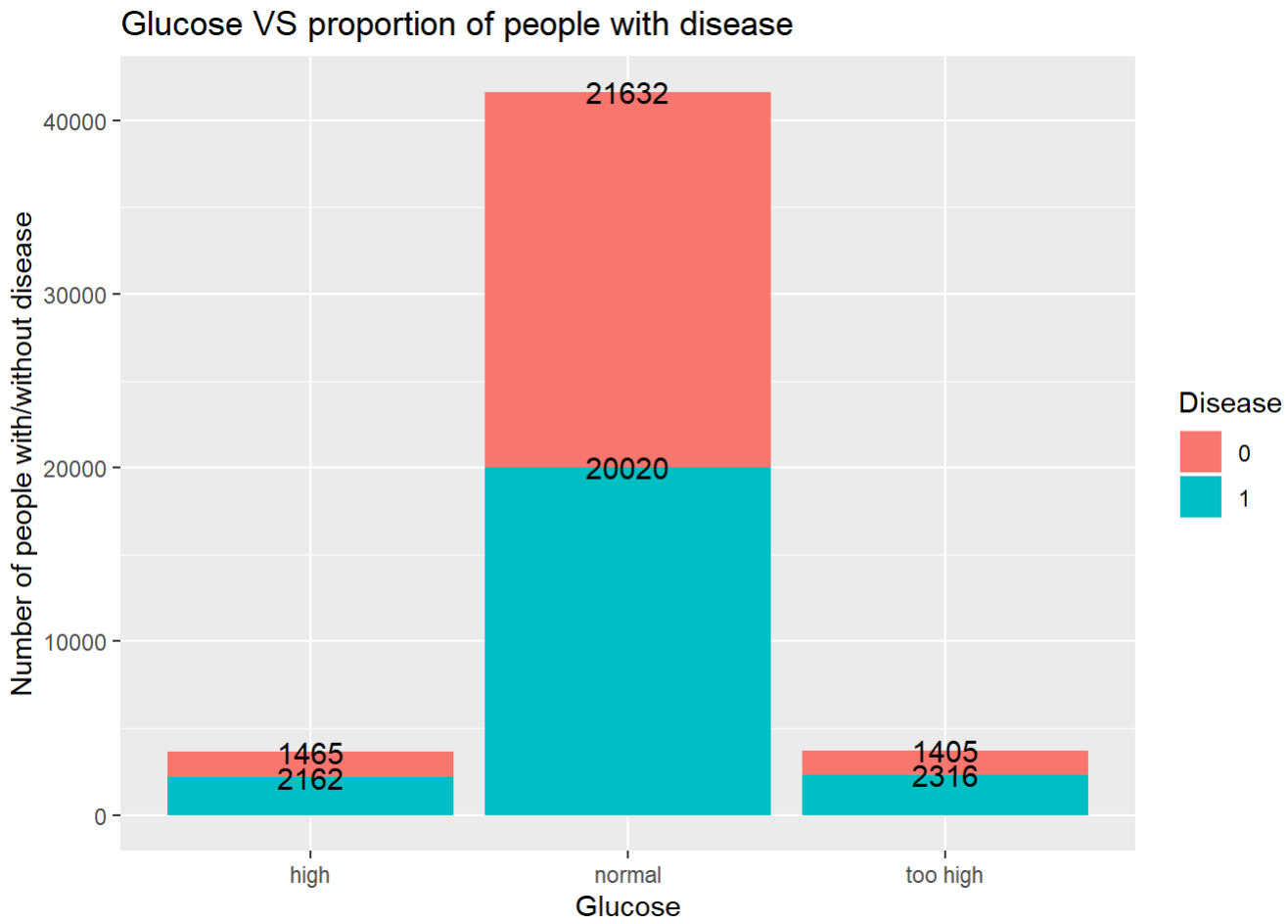
```
disease_count_by_cholestrol <- disease_prediction_training %>%
      group_by(Cholesterol,as.factor(Disease))%>%
      summarise(n())
colnames(disease_count_by_cholestrol) <- c('Cholesterol','Disease','NO_ppl')

Cholesterol_disease_plot <- ggplot(disease_count_by_cholestrol,aes(fill = Disease,x=Cholesterol,
y=NO_ppl))+geom_bar(position="stack",stat = "identity")+
             xlab("Cholesterol")+ ylab("Number of people with/without disease")+ ggtitle("Choles
terol VS proportion of people with disease")+ geom_text(aes(label = NO_ppl),position="stack",siz
e = 4)
Cholesterol_disease_plot
```



2.5. Bi Variate Analysis between Glucose and Disease column

We can observe that the percentage of people with the disease is higher amongst 'high' and 'too high' Glucose groups of people

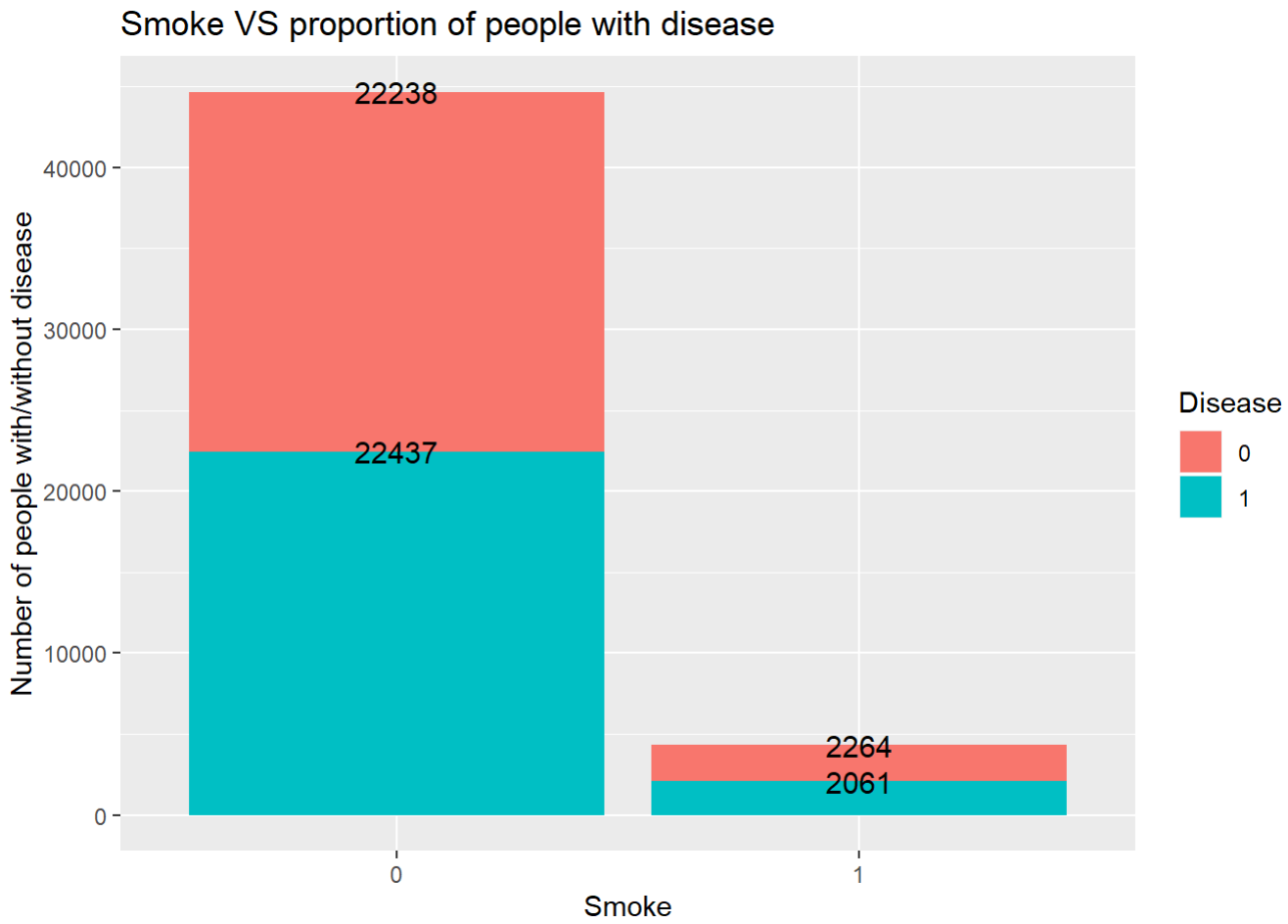
```
disease_count_by_Glucose <- disease_prediction_training %>%
      group_by(Glucose,as.factor(Disease))%>%
      summarise(n())
colnames(disease_count_by_Glucose) <- c('Glucose','Disease','NO_ppl')

Glucose_disease_plot <- ggplot(disease_count_by_Glucose,aes(fill = Disease,x=Glucose,y=NO_ppl))+
geom_bar(position="stack",stat = "identity")+
            xlab("Glucose")+ ylab("Number of people with/without disease")+ ggtitle("Glucose VS
proportion of people with disease")+ geom_text(aes(label = NO_ppl),position="stack",size = 4)
Glucose_disease_plot
```



2.6. Bi Variate Analysis between Smoke and Disease column

We can't observe a significant difference in the percentage of people with the disease amongst 'smoker' and 'non smoker' groups of people

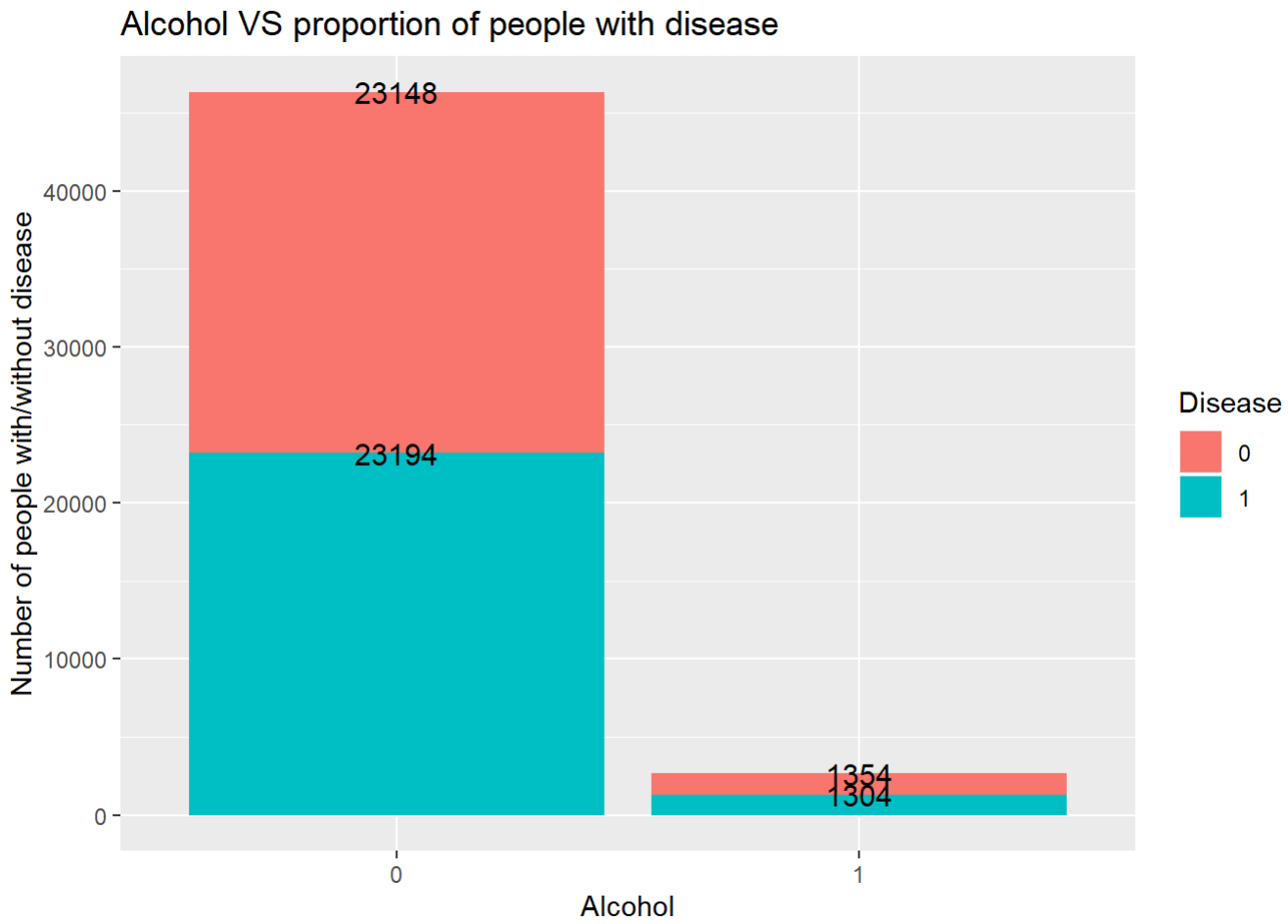
```
disease_count_by_Smoke <- disease_prediction_training %>%
      group_by(as.factor(Smoke),as.factor(Disease))%>%
      summarise(n())
colnames(disease_count_by_Smoke) <- c('Smoke','Disease','NO_ppl')

Smoke_disease_plot <- ggplot(disease_count_by_Smoke,aes(fill = Disease,x=Smoke,y=NO_ppl))+geom_b
ar(position="stack",stat = "identity")+
            xlab("Smoke")+ ylab("Number of people with/without disease")+ ggtitle("Smoke VS pro
portion of people with disease")+ geom_text(aes(label = NO_ppl),position="stack",size = 4)
Smoke_disease_plot
```



2.7. Bi Variate Analysis between Alcohol and Disease column

We can't observe a significant difference in the percentage of people with the disease amongst 'alcohol consumer' and 'non alcohol consumer' groups of people

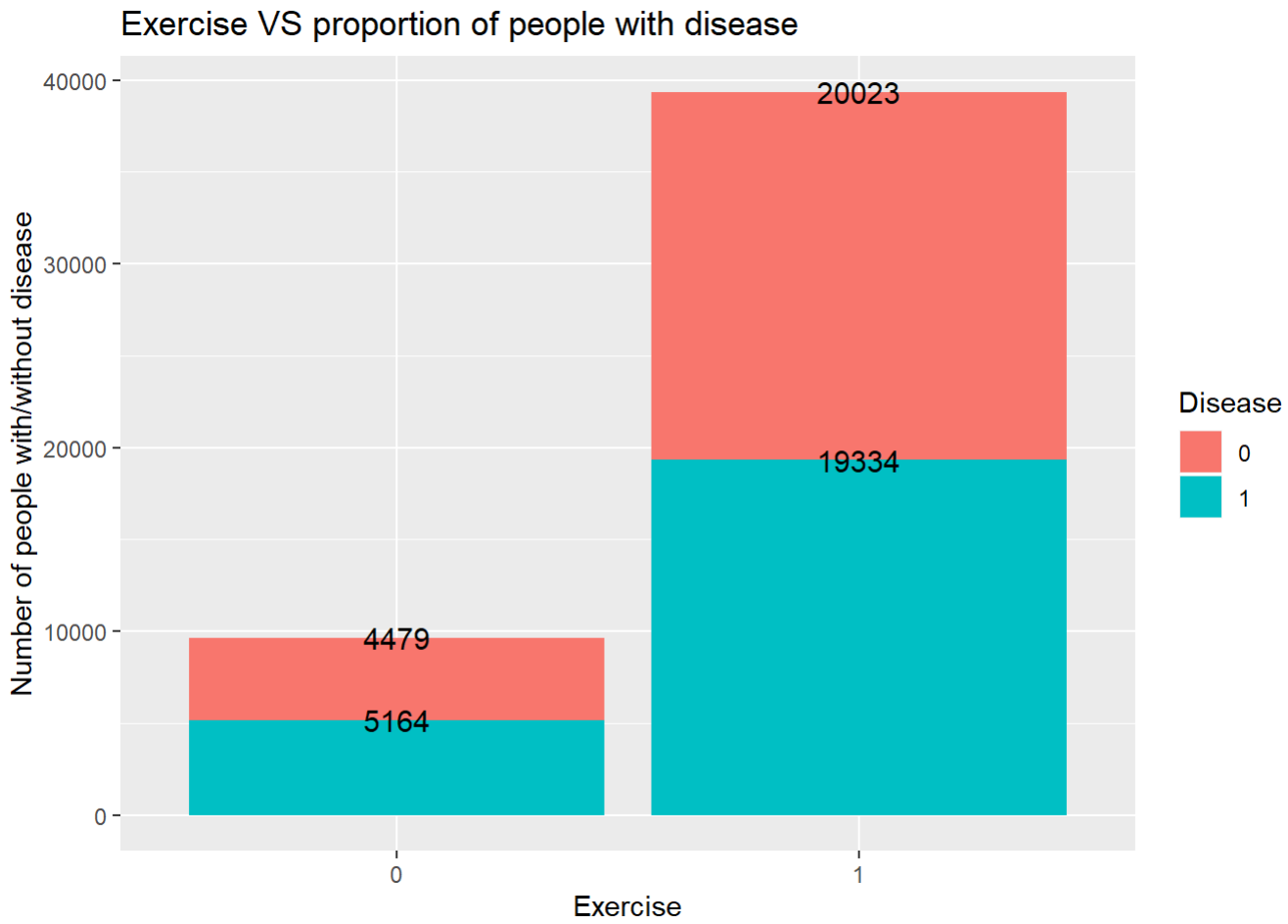
```
disease_count_by_Alcohol <- disease_prediction_training %>%
      group_by(as.factor(Alcohol),as.factor(Disease))%>%
      summarise(n())
colnames(disease_count_by_Alcohol) <- c('Alcohol','Disease','NO_ppl')

Alcohol_disease_plot <- ggplot(disease_count_by_Alcohol,aes(fill = Disease,x=Alcohol,y=NO_ppl))+
geom_bar(position="stack",stat = "identity")+
            xlab("Alcohol")+ ylab("Number of people with/without disease")+ ggtitle("Alcohol VS
proportion of people with disease")+ geom_text(aes(label = NO_ppl),position="stack",size = 4)
Alcohol_disease_plot
```



2.8. Bi Variate Analysis between Exercise and Disease column

We can't observe a significant difference in the percentage of people with the disease amongst 'Exercise' and 'non Exercise' groups of people

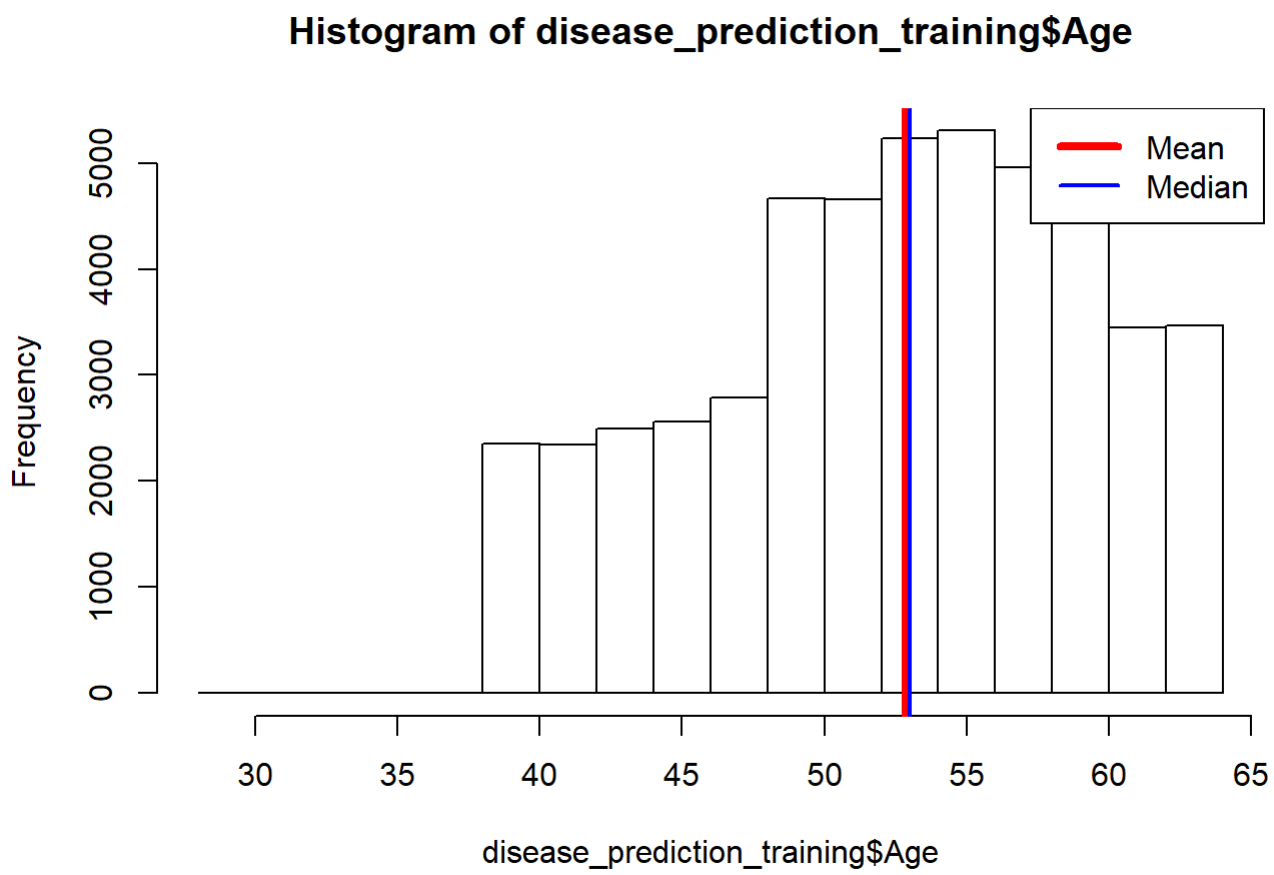
```
disease_count_by_Exercise <- disease_prediction_training %>%
      group_by(as.factor(Exercise),as.factor(Disease))%>%
      summarise(n())
colnames(disease_count_by_Exercise) <- c('Exercise','Disease','NO_ppl')

Exercise_disease_plot <- ggplot(disease_count_by_Exercise,aes(fill = Disease,x=Exercise,y=NO_pp
l))+geom_bar(position="stack",stat = "identity")+
             xlab("Exercise")+ ylab("Number of people with/without disease")+ ggtitle("Exercise
 VS proportion of people with disease")+ geom_text(aes(label = NO_ppl),position="stack",size = 4
)
Exercise_disease_plot
```



2.9. Uni Variate Analysis - Histogram of Age column to view the distribution

The Age column distribution is slightly left skewed

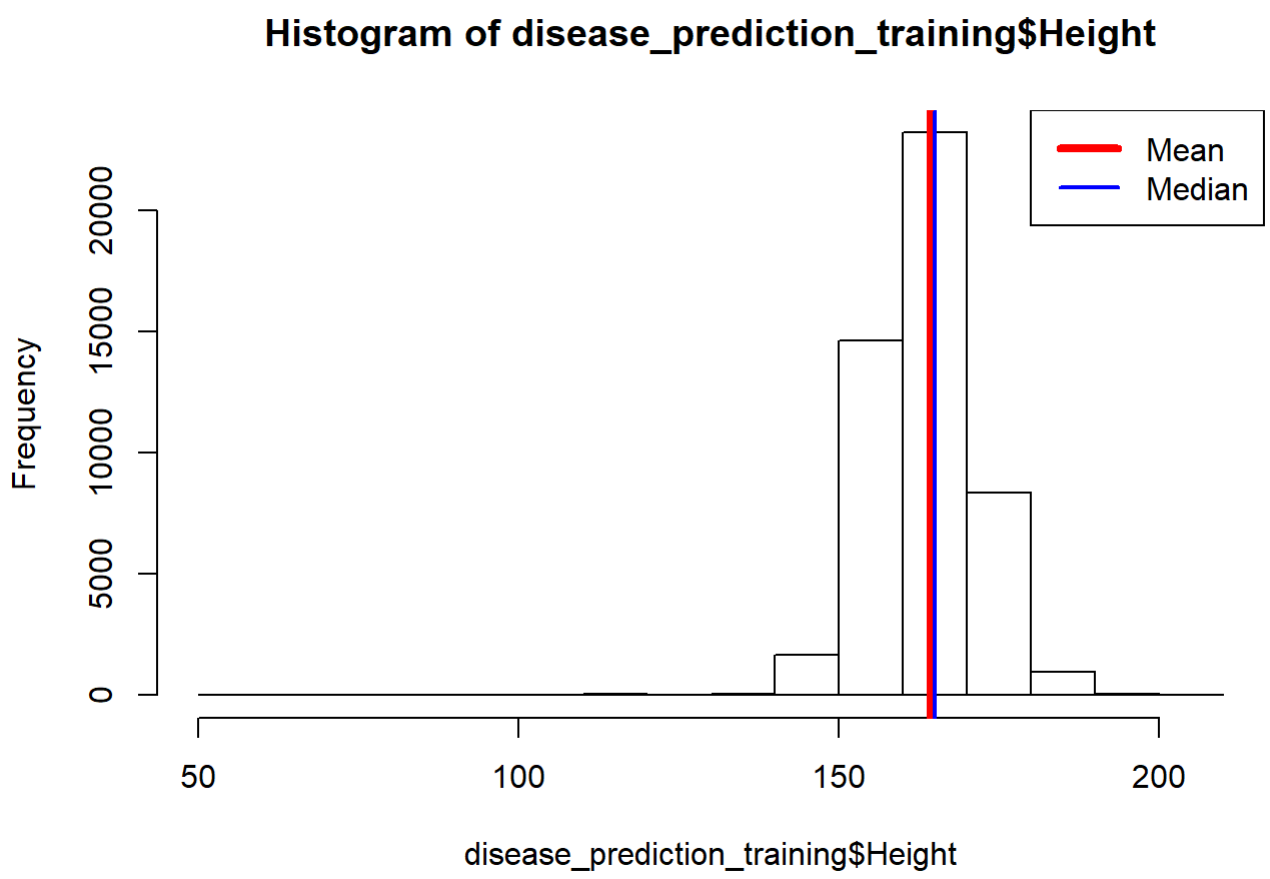
```
hist(disease_prediction_training$Age)
abline(v=mean(disease_prediction_training$Age),col = "red",lwd = 4)
abline(v=median(disease_prediction_training$Age),col = "blue", lwd = 2)
legend(x = "topright",
       c("Mean", "Median"),
       col = c("red", "blue"),lwd = c(4,2))
```



2.10. Uni Variate Analysis - Histogram of Height column to view the distribution

The Height column distribution is mostly normally distributed

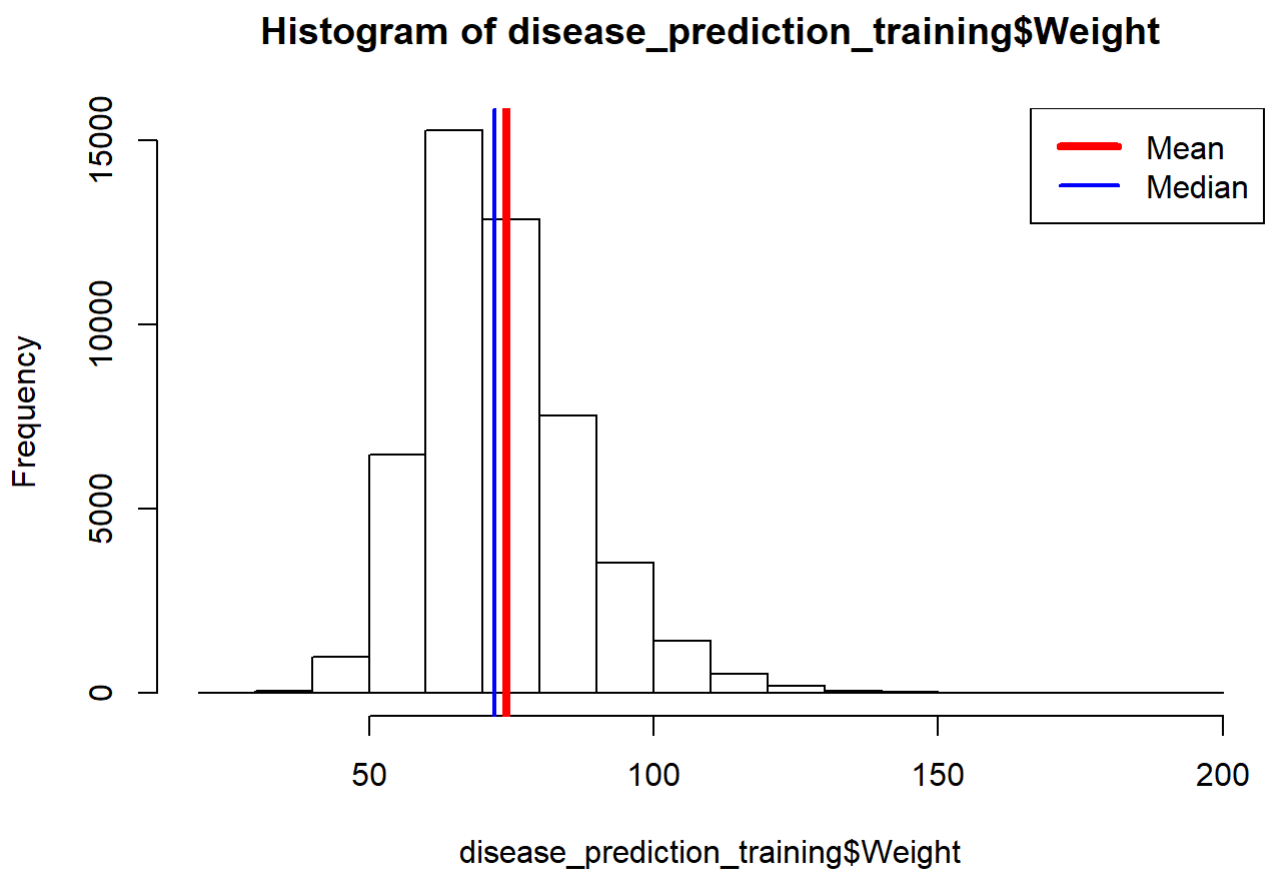
```
hist(disease_prediction_training$Height)
abline(v=mean(disease_prediction_training$Height),col = "red",lwd = 4)
abline(v=median(disease_prediction_training$Height),col = "blue", lwd = 2)
legend(x = "topright",
       c("Mean", "Median"),
       col = c("red", "blue"),lwd = c(4,2))
```



2.11. Uni Variate Analysis - Histogram of Weight column to view the distribution

The Weight column is mostly normally distributed with a slight right skew resulting in mean>median

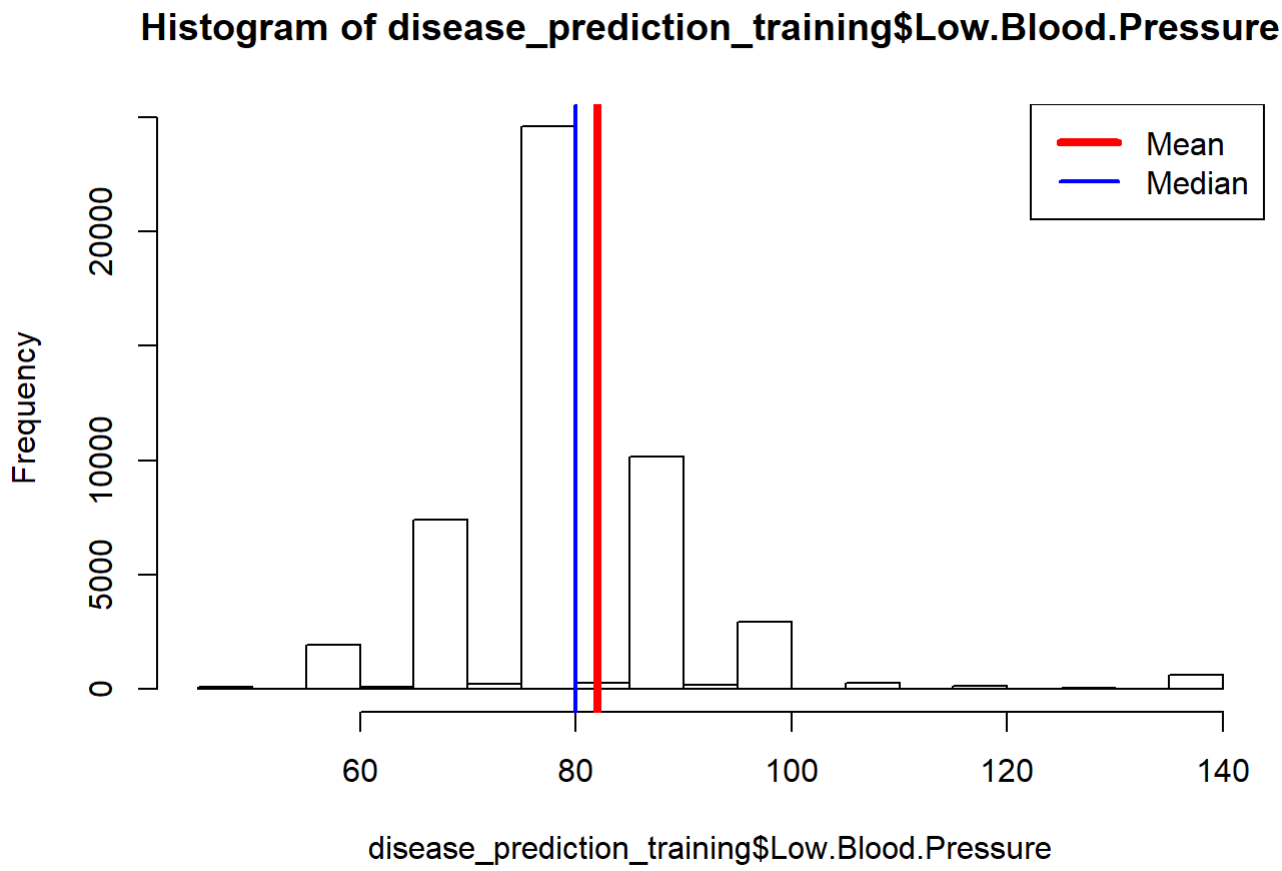
```
hist(disease_prediction_training$Weight)
abline(v=mean(disease_prediction_training$Weight),col = "red",lwd = 4)
abline(v=median(disease_prediction_training$Weight),col = "blue", lwd = 2)
legend(x = "topright",
       c("Mean", "Median"),
       col = c("red", "blue"),lwd = c(4,2))
```



2.12. Uni Variate Analysis - Histogram of Low.Blood.Pressure column to view the distribution

The Low.Blood.Pressure column is mostly normally distributed with a slight right skew resulting in mean>median

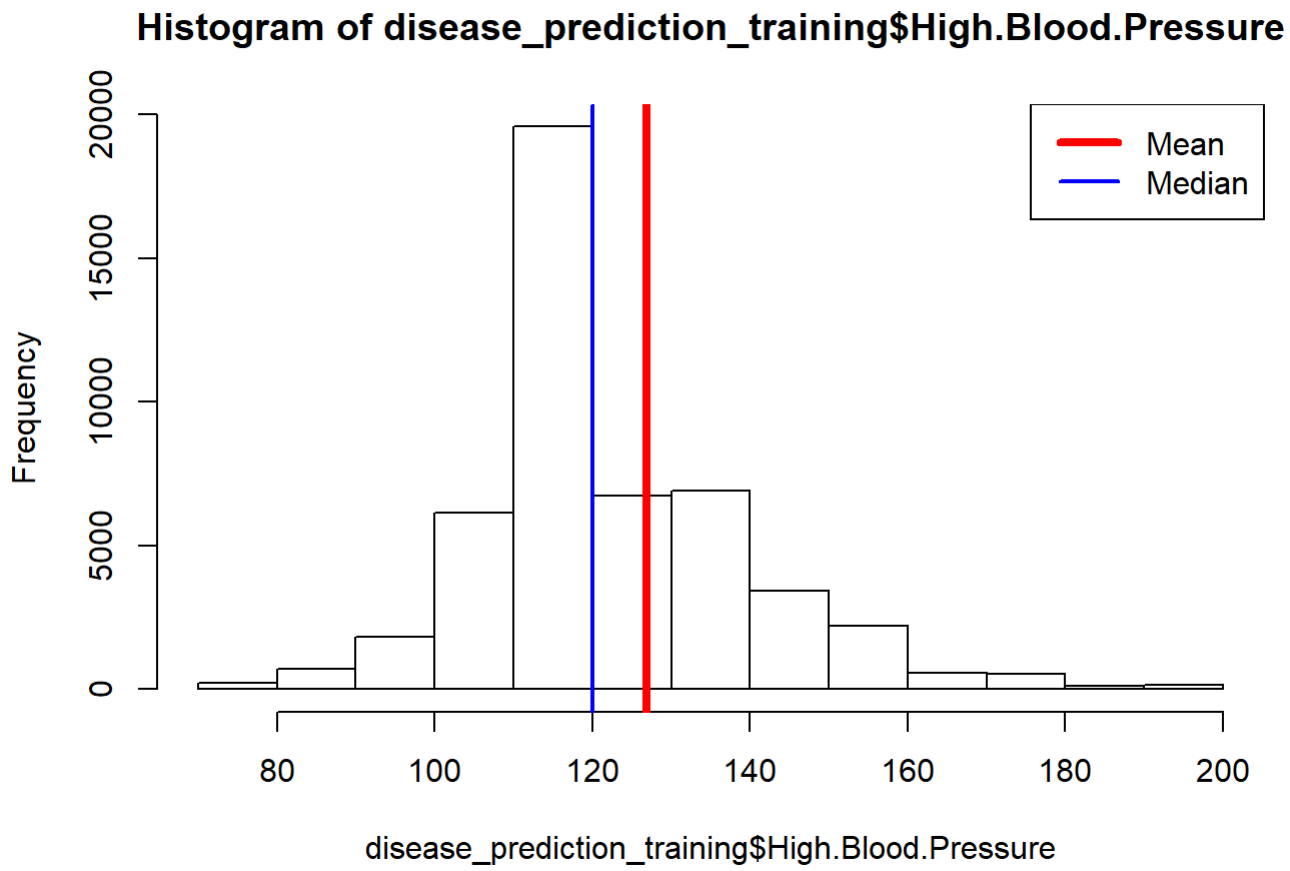
```
hist(disease_prediction_training$Low.Blood.Pressure)
abline(v=mean(disease_prediction_training$Low.Blood.Pressure),col = "red",lwd = 4)
abline(v=median(disease_prediction_training$Low.Blood.Pressure),col = "blue", lwd = 2)
legend(x = "topright",
       c("Mean", "Median"),
       col = c("red", "blue"),lwd = c(4,2))
```



2.13. Uni Variate Analysis - Histogram of High.Blood.Pressure column to view the distribution

The High.Blood.Pressure column is mostly normally distributed with a right skew resulting in mean>median

```
hist(disease_prediction_training$High.Blood.Pressure)
abline(v=mean(disease_prediction_training$High.Blood.Pressure),col = "red",lwd = 4)
abline(v=median(disease_prediction_training$High.Blood.Pressure),col = "blue", lwd = 2)
legend(x = "topright",
       c("Mean", "Median"),
       col = c("red", "blue"),lwd = c(4,2))
```



3. DATA PREPARATION for KNN & SVM :

For KNN and SVM, which are distance based algorithms, we require all the categorical variables to be converted to Numeric by performing one hot encoding and also all the numeric variables have to be normalized so that all the columns have values in the range of 0 to 1.

Creating Dummy Variables out of all categorical variables by performing one hot encoding and normalizing all numeric columns using Min Max scaler

```
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
library(fastDummies)
disease_prediction_training_knn_svm <- disease_prediction_training
disease_prediction_training_knn_svm<-fastDummies::dummy_cols(disease_prediction_training_knn_sv
m,select_columns=c('Gender','Cholesterol','Glucose'))

disease_prediction_training_knn_svm$Age <- normalize(disease_prediction_training_knn_svm$Age)
disease_prediction_training_knn_svm$Height <- normalize(disease_prediction_training_knn_svm$Heig
ht)
disease_prediction_training_knn_svm$Weight <- normalize(disease_prediction_training_knn_svm$Weig
ht)
disease_prediction_training_knn_svm$bmi <- normalize(disease_prediction_training_knn_svm$bmi)
disease_prediction_training_knn_svm$Low.Blood.Pressure <- normalize(disease_prediction_training_
knn_svm$Low.Blood.Pressure)
disease_prediction_training_knn_svm$High.Blood.Pressure <- normalize(disease_prediction_training
_knn_svm$High.Blood.Pressure)
```

Getting rid of non numeric columns

```
disease_prediction_training_knn_svm$Gender <- NULL
disease_prediction_training_knn_svm$Cholesterol <- NULL
disease_prediction_training_knn_svm$Glucose <- NULL
disease_prediction_training_knn_svm$age_groups <- NULL

disease_prediction_training_knn_svm$bmi_groups <- NULL
disease_prediction_training_knn_svm$Gender_male <- NULL
disease_prediction_training_knn_svm$Disease <- as.factor(disease_prediction_training_knn_svm$Dis
ease)
```

SECTION 2: BUILD, TUNE & EVALUATE various ML ALGORITHMS

2.1 K Nearest Neighbors (KNN)

Generating Training and Validation datasets for KNN algorithm with 70% & 30% splits respectively. (Hold one Out method to ensure that the model built doesn't overfit on train data by comparing the accuracies obtained on training data and validation data)

```
library(caret)
set.seed(188)
train_index <- createDataPartition(disease_prediction_training_knn_svm$Disease, p = 0.7, list =
FALSE)

disease_prediction_knn_train <- disease_prediction_training_knn_svm[train_index, ]
disease_prediction_knn_train$Disease <- as.factor(disease_prediction_knn_train$Disease)
disease_prediction_knn_test <- disease_prediction_training_knn_svm[-train_index, ]
disease_prediction_knn_test$Disease <- as.factor(disease_prediction_knn_test$Disease)
disease_prediction_knn_train$Height <- NULL
disease_prediction_knn_train$Weight <- NULL
disease_prediction_knn_test$Height <- NULL
disease_prediction_knn_test$Weight <- NULL
```

BUILDING BASELINE MODEL for KNN by maximizing Accuracy

```
#baseline_model_knn <- train(Disease ~ ., data = disease_prediction_knn_train, method = "knn")
predict_disease_knn <- predict(baseline_model_knn, newdata = disease_prediction_knn_test)
```

The baseline model produced the following accuracies on the training data for different values of K

```
print(baseline_model_knn)
```

```
## k-Nearest Neighbors
##
## 34301 samples
##    14 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 34301, 34301, 34301, 34301, 34301, 34301, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.6696718  0.3393303
##   7  0.6811563  0.3623155
##   9  0.6892389  0.3784812
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

The baseline model produced the below shown accuracy and sensitivity on validation data which it has not seen before.

```
confusionMatrix(predict_disease_knn,disease_prediction_knn_test$Disease, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 5364 2239
##          1 1986 5110
##
##                Accuracy : 0.7126
##                  95% CI : (0.7052, 0.7199)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4251
##
##  Mcnemar's Test P-Value : 0.0001058
##
##             Sensitivity : 0.6953
##             Specificity : 0.7298
##          Pos Pred Value : 0.7201
##          Neg Pred Value : 0.7055
##              Prevalence : 0.5000
##          Detection Rate : 0.3476
##    Detection Prevalence : 0.4828
##       Balanced Accuracy : 0.7126
##
##        'Positive' Class : 1
##
```

Fine tuning the hyperparameter ‘K’ for producing the best possible accuracy.

The Hyperparameter ‘K’ in KNN algorithm denotes the number of nearest neighbor rows in the training data that are used to classify each validation data row. The data point is classified based on the majority class of its nearest neighbors & in case of a tie, one class is randomly chosen. **If K is too small it will result in high variance and if K is too large it will result in high bias. Hence an optimum value of K must be chosen to produce unbiased and low variance estimates** *Fine tuning the model to produce maximum accuracy and using 5 fold Cross Validation to train the model by controlling for overfitting*

```
#tuned_model_knn <- train(Disease ~ ., data = disease_prediction_knn_train, method = "knn",
#                    tuneGrid = data.frame(k = seq(20, 40)),
#                    trControl = trainControl(method = "repeatedcv",
#                                             number = 5, repeats = 3))
```

From the below table summarizing the best performing models and their corresponding accuracies we can see the accuracies for different values of K

```
print(tuned_model_knn)
```

```
## k-Nearest Neighbors
##
## 34301 samples
##    14 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 27442, 27441, 27440, 27440, 27441, 27441, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##   20  0.7193569  0.4387109
##   21  0.7197845  0.4395658
##   22  0.7195900  0.4391768
##   23  0.7202217  0.4404401
##   24  0.7196484  0.4392936
##   25  0.7205035  0.4410038
##   26  0.7203772  0.4407512
##   27  0.7212518  0.4425002
##   28  0.7211255  0.4422475
##   29  0.7217377  0.4434718
##   30  0.7222139  0.4444242
##   31  0.7225444  0.4450851
##   32  0.7222139  0.4444242
##   33  0.7222916  0.4445797
##   34  0.7223208  0.4446379
##   35  0.7233023  0.4466009
##   36  0.7233217  0.4466398
##   37  0.7228941  0.4457844
##   38  0.7223596  0.4447156
##   39  0.7227386  0.4454736
##   40  0.7225054  0.4450071
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 36.
```

Obtaining Predicted values for disease on validation data using the fine tuned KNN model

```
predict_disease_tuned_knn <- predict(tuned_model_knn, newdata = disease_prediction_knn_test)
```

Evaluating Model Performance by calculating Accuracy Precision and Recall on the classification done on the Validation Data. (Hold one out method to control for overfitting) **After fine tuning the hyperparameters, the accuracy of KNN model has increased and the risk of overfitting also has been avoided. Since the accuracy of the validation data is inline with the accuracy of the train data, the model is not overfitting on the train data**

```
disease_prediction_knn_test$predicted_disease <- predict_disease_tuned_knn
conf_matrix <- data.frame(table(disease_prediction_knn_test$Disease,disease_prediction_knn_test
$predicted_disease))
colnames(conf_matrix)<- c('Actual class','Predicted Class','Count')

Accuracy_DT <- sum(conf_matrix$Count[conf_matrix$`Actual class`==conf_matrix$`Predicted Class
`])/sum(conf_matrix$Count)
Precision_DT <- conf_matrix$Count[conf_matrix$`Actual class`== 1 & conf_matrix$`Predicted Class`
== 1]/sum(conf_matrix$Count[conf_matrix$`Predicted Class`==1])
Recall_DT <- conf_matrix$Count[conf_matrix$`Actual class`==1 & conf_matrix$`Predicted Class`==1
]/sum(conf_matrix$Count[conf_matrix$`Actual class`==1])
F1_score_DT <- (2*Precision_DT*Recall_DT)/(Precision_DT+Recall_DT)

paste("Accuracy of best KNN Algorithm in classifying Disease is :",Accuracy_DT)
```

```
## [1] "Accuracy of best KNN Algorithm in classifying Disease is : 0.728416899108783"
```

```
paste("Precision of KNN Algorithm in classifying Disease is :",Precision_DT)
```

```
## [1] "Precision of KNN Algorithm in classifying Disease is : 0.753358490566038"
```

```
paste("Recall of KNN Algorithm in classifying Disease is :",Recall_DT)
```

```
## [1] "Recall of KNN Algorithm in classifying Disease is : 0.679140019050211"
```

```
paste("F1 Score of KNN Algorithm in classifying Disease is :",F1_score_DT)
```

```
## [1] "F1 Score of KNN Algorithm in classifying Disease is : 0.714326606555031"
```

Fine tuning the model to produce maximum Recall by tuning it on Sensitivity(Recall) metric **Recall is also an important metric to optimize in this case because, by maximizing Recall, we will be reducing False Negatives. False Negatives are cases where a patient actually has the disease but our model classifies the patient as not having the disease. False Negatives can be very dangerous because patients with the disease might go without receiving any treatment which is a highly undesirable situation. Hence we should focus on maximizing Recall to minimize False Negatives**

```
#tuned_model_knn_recall <- train(Disease ~ ., data = disease_prediction_knn_train, method = "kn
n",
#                    tuneGrid = data.frame(k = seq(20, 40)),metric="Sens",
#                    trControl = trainControl(method = "repeatedcv",
#                                             number = 5, repeats = 3,
#                                             summaryFunction = twoClassSummary))
```

From the below table summarizing the best performing models and their corresponding Sensitivities, we can see the Sensitivities for different values of K

```
print(tuned_model_knn_recall)
```

```
## k-Nearest Neighbors
##
## 34301 samples
##    14 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 27442, 27441, 27441, 27440, 27440, 27441, ...
## Resampling results across tuning parameters:
##
##   k   ROC  Sens       Spec
##   20  NaN  0.7494171  0.6874647
##   21  NaN  0.7509327  0.6879311
##   22  NaN  0.7516519  0.6865705
##   23  NaN  0.7538871  0.6869398
##   24  NaN  0.7541785  0.6859096
##   25  NaN  0.7554998  0.6856568
##   26  NaN  0.7549363  0.6846655
##   27  NaN  0.7572296  0.6834217
##   28  NaN  0.7570159  0.6836744
##   29  NaN  0.7578904  0.6830913
##   30  NaN  0.7584347  0.6834801
##   31  NaN  0.7594646  0.6836549
##   32  NaN  0.7598531  0.6839076
##   33  NaN  0.7607472  0.6835383
##   34  NaN  0.7611165  0.6839076
##   35  NaN  0.7616413  0.6836549
##   36  NaN  0.7612719  0.6837522
##   37  NaN  0.7624769  0.6840437
##   38  NaN  0.7626906  0.6838882
##   39  NaN  0.7626712  0.6829552
##   40  NaN  0.7630016  0.6821193
##
## Sens was used to select the optimal model using the largest value.
## The final value used for the model was k = 40.
```

Obtaining Predicted values for disease on validation data using the best fine tuned KNN model

```
predict_disease_tuned_knn_recall <- predict(tuned_model_knn_recall, newdata = disease_prediction
_knn_test)
```

Evaluating Model Performance by calculating Accuracy Precision and Recall on the classification done on the Validation Data **After fine tuning the hyperparameters and optimizing for Recall metric, the accuracy, precision, recall and F1 scores on the validation data did not change much in comparison to the values obtained by maximizing the Accuracy metric** *Hence we can proceed with the model that was built by maximing the accuracy metric itself as our best model*

```
disease_prediction_knn_test$predicted_disease_recallTuned <- predict_disease_tuned_knn_recall
conf_matrix <- data.frame(table(disease_prediction_knn_test$Disease,disease_prediction_knn_test
$predicted_disease_recallTuned))
colnames(conf_matrix)<- c('Actual class','Predicted Class','Count')

Accuracy_DT <- sum(conf_matrix$Count[conf_matrix$`Actual class`==conf_matrix$`Predicted Class
`])/sum(conf_matrix$Count)
Precision_DT <- conf_matrix$Count[conf_matrix$`Actual class`== 1 & conf_matrix$`Predicted Class`
== 1]/sum(conf_matrix$Count[conf_matrix$`Predicted Class`==1])
Recall_DT <- conf_matrix$Count[conf_matrix$`Actual class`==1 & conf_matrix$`Predicted Class`==1
]/sum(conf_matrix$Count[conf_matrix$`Actual class`==1])
F1_score_DT <- (2*Precision_DT*Recall_DT)/(Precision_DT+Recall_DT)

paste("Accuracy of best KNN Algorithm in classifying Disease is :",Accuracy_DT)
```

```
## [1] "Accuracy of best KNN Algorithm in classifying Disease is : 0.726307912102864"
```

```
paste("Precision of KNN Algorithm in classifying Disease is :",Precision_DT)
```

```
## [1] "Precision of KNN Algorithm in classifying Disease is : 0.745860437610881"
```

```
paste("Recall of KNN Algorithm in classifying Disease is :",Recall_DT)
```

```
## [1] "Recall of KNN Algorithm in classifying Disease is : 0.686487957545244"
```

```
paste("F1 Score of KNN Algorithm in classifying Disease is :",F1_score_DT)
```

```
## [1] "F1 Score of KNN Algorithm in classifying Disease is : 0.714943668957699"
```

ROC & AUROC for the best KNN Model

```
#install.packages("pROC")
library(pROC)
```

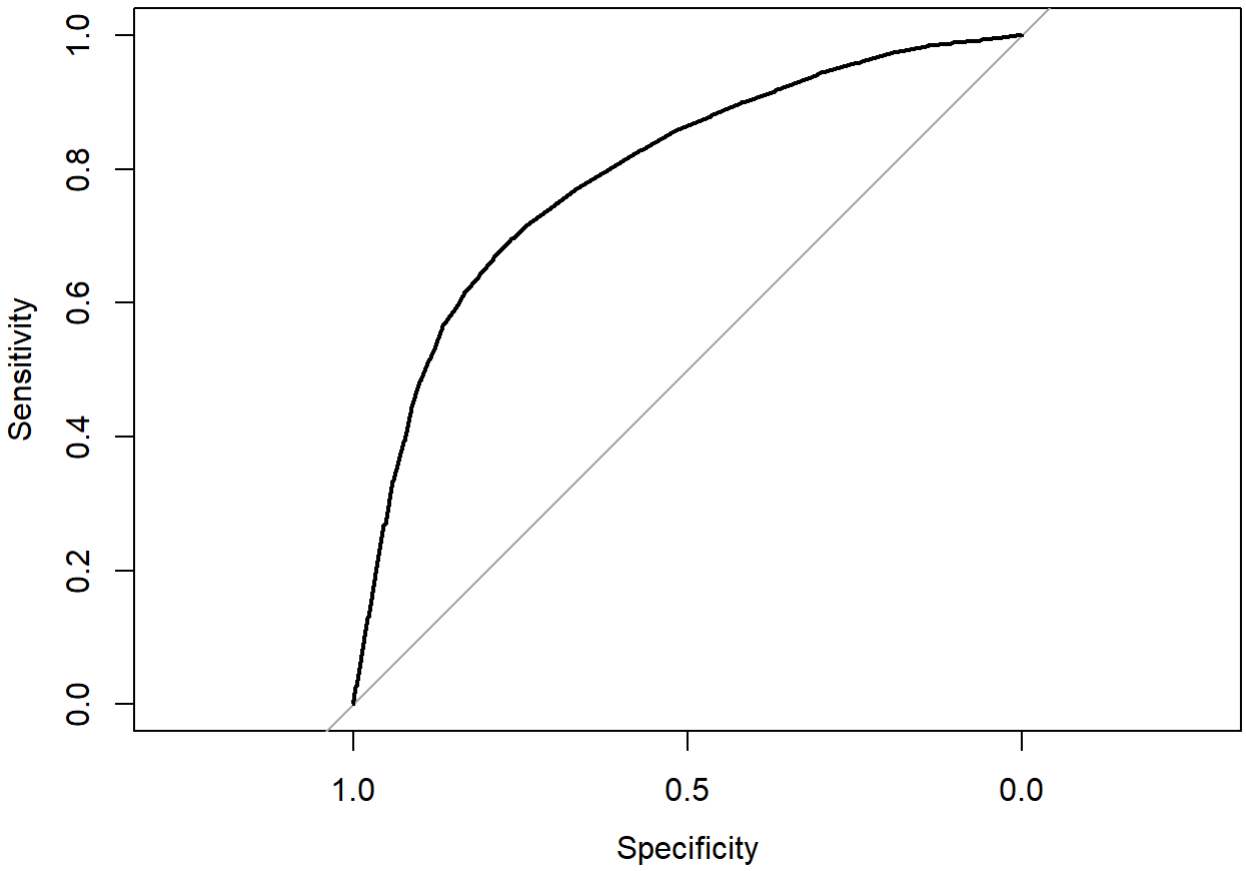
Generated ROC curve and calculated Area Under Curve metric for the identified best performing KNN model

```
knn_pred_disease <- predict(tuned_model_knn, newdata = disease_prediction_knn_test, na.action =
na.omit, type = "prob")
roc_curve <- roc(disease_prediction_knn_test$Disease,knn_pred_disease$`1`)
```

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

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

```
plot(roc_curve)
```



Obtained the area under the ROC curve for the best KNN model

```
paste("Area Under the ROC curve is :",auc(roc_curve))
```

```
## [1] "Area Under the ROC curve is : 0.791844565830142"
```

END OF BUILDING KNN Model (ML algorithm number 1)

2.2 Naive Bayes Classifier Model (NBC)

Creating Training and Validation data splits (Hold One Out method)

```
library(caret)
set.seed(188)
train_index <- createDataPartition(disease_prediction_training$Disease, p = 0.7, list = FALSE)

disease_prediction_nb_train <- disease_prediction_training[train_index, ]
disease_prediction_nb_train$Disease <- as.factor(disease_prediction_nb_train$Disease)
disease_prediction_nb_train$age_groups <- NULL
disease_prediction_nb_train$Height <- NULL
disease_prediction_nb_train$Weight <- NULL
disease_prediction_nb_train$Smoke <- as.factor(disease_prediction_nb_train$Smoke)
disease_prediction_nb_train$Alcohol <- as.factor(disease_prediction_nb_train$Alcohol)
disease_prediction_nb_train$Exercise <- as.factor(disease_prediction_nb_train$Exercise)

disease_prediction_nb_test <- disease_prediction_training[-train_index, ]
disease_prediction_nb_test$Disease <- as.factor(disease_prediction_nb_test$Disease)
disease_prediction_nb_test$age_groups <- NULL
disease_prediction_nb_test$Height <- NULL
disease_prediction_nb_test$Weight <- NULL
disease_prediction_nb_test$Smoke <- as.factor(disease_prediction_nb_test$Smoke)
disease_prediction_nb_test$Alcohol <- as.factor(disease_prediction_nb_test$Alcohol)
disease_prediction_nb_test$Exercise <- as.factor(disease_prediction_nb_test$Exercise)
```

```
#install.packages("e1071", dependencies = TRUE)
library(e1071)
```

Naive Bayes baseline model

```
#install.packages("klaR")
library(klaR)
nb_baseline_model <- naiveBayes(Disease~ ., data = disease_prediction_nb_train)
```

```
predict_disease_nb <- predict(nb_baseline_model, newdata = disease_prediction_nb_test,  type = c
("class", "raw"))
```

Measuring the accuracy, precision, recall and F1 Score of the Naive Bayes base model by making predictions on the validation dataset and comparing it with actual values. These metrics are used to evaluate the Naive Bayes model performance.

```
disease_prediction_nb_test$predicted_disease_nb <- predict_disease_nb
conf_matrix <- data.frame(table(disease_prediction_nb_test$Disease,disease_prediction_nb_test$pr
edicted_disease_nb))
colnames(conf_matrix)<- c('Actual class','Predicted Class','Count')

Accuracy_DT <- sum(conf_matrix$Count[conf_matrix$`Actual class`==conf_matrix$`Predicted Class
`])/sum(conf_matrix$Count)
Precision_DT <- conf_matrix$Count[conf_matrix$`Actual class`== 1 & conf_matrix$`Predicted Class`
== 1]/sum(conf_matrix$Count[conf_matrix$`Predicted Class`==1])
Recall_DT <- conf_matrix$Count[conf_matrix$`Actual class`==1 & conf_matrix$`Predicted Class`==1
]/sum(conf_matrix$Count[conf_matrix$`Actual class`==1])
F1_score_DT <- (2*Precision_DT*Recall_DT)/(Precision_DT+Recall_DT)

paste("Accuracy of NB Algorithm in classifying Disease is :",Accuracy_DT)
```

```
## [1] "Accuracy of NB Algorithm in classifying Disease is : 0.713537414965986"
```

```
paste("Precision of NB in classifying Disease is :",Precision_DT)
```

```
## [1] "Precision of NB in classifying Disease is : 0.771515871668647"
```

```
paste("Recall of NB in classifying Disease is :",Recall_DT)
```

```
## [1] "Recall of NB in classifying Disease is : 0.61336032388664"
```

```
paste("F1 Score of NB in classifying Disease is :",F1_score_DT)
```

```
## [1] "F1 Score of NB in classifying Disease is : 0.683407262611834"
```

Tuning the Laplace Hyperparameter for Naive Bayes Classifier

Performing Grid Search with different values of laplace and measuring the accuracy for each one

```
laplace_values <- c(1,2,3,4,5)
accuracy_values <- c()
for (i in laplace_values) {

  nb_tuned_model <- naiveBayes(Disease~ ., data = disease_prediction_nb_train, laplace = i)
  predict_disease_nb <- predict(nb_tuned_model, newdata = disease_prediction_nb_test,  type = c(
"class", "raw"))

disease_prediction_nb_test$predicted_disease_nb <- predict_disease_nb
conf_matrix <- data.frame(table(disease_prediction_nb_test$Disease,disease_prediction_nb_test$pr
edicted_disease_nb))
colnames(conf_matrix)<- c('Actual class','Predicted Class','Count')

Accuracy_DT <- sum(conf_matrix$Count[conf_matrix$`Actual class`==conf_matrix$`Predicted Class
`])/sum(conf_matrix$Count)
  accuracy_values<- c(accuracy_values,Accuracy_DT)
}
```

From the below table we can observe that there is no significant difference in accuracy for different values of laplace. Hence we can proceed with the baseline model and produce the ROc curve and AUROC

```
data.frame(accuracy_values,laplace_values)
```

```
##   accuracy_values laplace_values
## 1       0.7135374              1
## 2       0.7135374              2
## 3       0.7135374              3
## 4       0.7135374              4
## 5       0.7135374              5
```

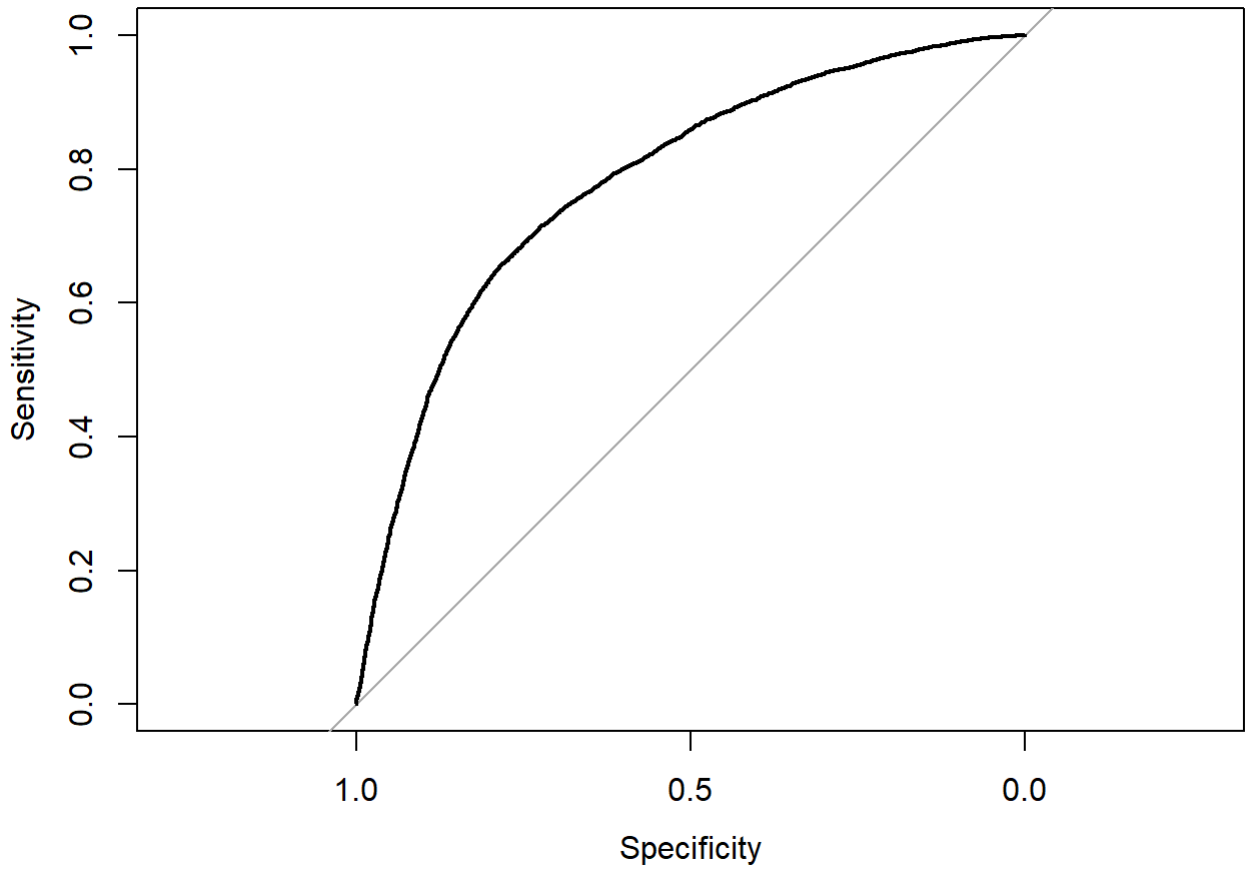
Generated ROC curve and calculated Area Under Curve metric for the identified best performing Naive Bayes Classifier model

```
nb_pred_disease <- predict(nb_baseline_model, newdata = disease_prediction_nb_test, na.action =
na.omit, type = "raw")
roc_curve <- roc(disease_prediction_nb_test$Disease,nb_pred_disease[,2])
```

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

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

```
plot(roc_curve)
```



Obtained the area under the ROC curve for the best NBC model

```
paste("Area Under the ROC curve is :",auc(roc_curve))
```

```
## [1] "Area Under the ROC curve is : 0.78131616897049"
```

END OF BUILDING Naive Bayes Classifier (ML algorithm number 2)

2.3 Random Forest Model (RF)

Creating Training and Validation data splits (Hold One Out method)

```
library(caret)
set.seed(188)
train_index <- createDataPartition(disease_prediction_training$Disease, p = 0.7, list = FALSE)

disease_prediction_rf_train <- disease_prediction_training[train_index, ]
disease_prediction_rf_train$Disease <- as.factor(disease_prediction_rf_train$Disease)


disease_prediction_rf_test <- disease_prediction_training[-train_index, ]
disease_prediction_rf_test$Disease <- as.factor(disease_prediction_rf_test$Disease)
```

Running baseline model for Random Forest

```
#baselinemodel_rf <- train(Disease ~ ., data = disease_prediction_rf_train, method = "rf")
```

The baseline model accuracies on training dataset for different values of mtry where mtry is the number of features selected for each tree

```
baselinemodel_rf
```

```
## Random Forest
##
## 34300 samples
##    14 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 34300, 34300, 34300, 34300, 34300, 34300, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.7348514  0.4694990
##   11    0.7062992  0.4125676
##   20    0.7007327  0.4014495
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
predict_disease_rf_base <- predict(baselinemodel_rf, newdata = disease_prediction_rf_test)
```

Evaluating the Random Forest Baseline Model Performance by calculating Accuracy Precision and Recall on the classification done on the Validation Data.Verified that the accuracy of the baseline model on training data is in line with the accuracy of the model on validation data.

```
disease_prediction_rf_test$predicted_disease_rf_baseline <- predict_disease_rf_base
conf_matrix <- data.frame(table(disease_prediction_rf_test$Disease,disease_prediction_rf_test$pr
edicted_disease_rf_baseline))
colnames(conf_matrix)<- c('Actual class','Predicted Class','Count')

Accuracy_DT <- sum(conf_matrix$Count[conf_matrix$`Actual class`==conf_matrix$`Predicted Class
`])/sum(conf_matrix$Count)
Precision_DT <- conf_matrix$Count[conf_matrix$`Actual class`== 1 & conf_matrix$`Predicted Class`
== 1]/sum(conf_matrix$Count[conf_matrix$`Predicted Class`==1])
Recall_DT <- conf_matrix$Count[conf_matrix$`Actual class`==1 & conf_matrix$`Predicted Class`==1
]/sum(conf_matrix$Count[conf_matrix$`Actual class`==1])
F1_score_DT <- (2*Precision_DT*Recall_DT)/(Precision_DT+Recall_DT)

paste("Accuracy of baseline RF Algorithm in classifying Disease is :",Accuracy_DT)
```

```
## [1] "Accuracy of baseline RF Algorithm in classifying Disease is : 0.735170068027211"
```

```
paste("Precision of baseline RF Algorithm in classifying Disease is :",Precision_DT)
```

```
## [1] "Precision of baseline RF Algorithm in classifying Disease is : 0.763051608077786"
```

```
paste("Recall of baseline RF Algorithm in classifying Disease is :",Recall_DT)
```

```
## [1] "Recall of baseline RF Algorithm in classifying Disease is : 0.688394062078273"
```

```
paste("F1 Score of baseline RF Algorithm in classifying Disease is :",F1_score_DT)
```

```
## [1] "F1 Score of baseline RF Algorithm in classifying Disease is : 0.723802766938631"
```

Checking the importance of variables to exclude the least important variables before running the grid search **From the below table we can conclude that age_groups and Alcohol are less important features**

```
varImp(baselinemodel_rf)
```

```
## rf variable importance
##
##                       Overall
## High.Blood.Pressure  100.0000
## Low.Blood.Pressure    48.3152
## Age                   26.5902
## bmi                   23.0726
## Weight                18.4724
## Height                14.0101
## Cholesterolnormal     11.5563
## Cholesteroltoo high   11.1156
## age_groups58 to 64     9.6938
## bmi_groupsHealthy      2.4783
## bmi_groupsObese        2.3947
## Glucosenormal          1.7013
## Exercise               1.5920
## age_groups53 to 58     1.4094
## Gendermale             1.0737
## Glucosetoo high        0.5127
## Smoke                  0.5014
## age_groups48 to 53     0.2473
## Alcohol                0.1490
## bmi_groupsOverweight   0.0000
```

Performing Grid Search by tuning hyperparameters to maximize Accuracy

mtry and ntree are the two tuning hyperparameters that are used to tune the Random Forest Model. mtry denotes the number of features that are selected for each decision tree and ntree is the number of decision trees that are going to get created. For Producing unbiased estimates we should let each of our decision tree grow to its full size

without pruning and in order to reduce variance we need to optimize the number of features selected for each Tree (mtry) and try to keep it at a smaller number. We can also control the number of trees (ntree) to reduce the variance 3 Fold Cross Validation is also done on the training data to control for Overfitting

```
#library(caret)
#control <- trainControl(method="repeatedcv", number=3, repeats=3, search="grid")
#tunegrid <- expand.grid(.mtry=c(2,3,4))
#modellist <- list()
#for (ntree in c(400,600,800,1000)) {
#   set.seed(100)
#  tuned_model_rf <- train(Disease~., data=disease_prediction_rf_train, method="rf", metric="Acc
uracy", tuneGrid=tunegrid, trControl=control, ntree=ntree)
#    key <- toString(ntree)
#    modellist[[key]] <- tuned_model_rf
#}
```

From the below grid search results table we can observe the accuracies of the models based on number of trees as shown in the table below

```
accuracies <- c(max(modellist$`400`$results$Accuracy),max(modellist$`600`$results$Accuracy),max
(modellist$`800`$results$Accuracy),max(modellist$`1000`$results$Accuracy))
number_of_tress <- c(400,600,800,1000)
data.frame(number_of_tress,accuracies)
```

```
##   number_of_tress accuracies
## 1             400  0.7374344
## 2             600  0.7371040
## 3             800  0.7375802
## 4            1000  0.7374927
```

Proceeding with the model with 800 trees as the best model. Also mtry = 2 was chosen as the best model which produced maximum accuracy

```
tuned_model_rf <- modellist$`800`
tuned_model_rf
```

```
## Random Forest
##
## 34300 samples
##    13 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold, repeated 3 times)
## Summary of sample sizes: 22867, 22867, 22866, 22867, 22866, 22867, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.7375802  0.4749965
##   3     0.7349271  0.4696973
##   4     0.7305540  0.4609681
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
predict_disease_rf_tuned <- predict(tuned_model_rf, newdata = disease_prediction_rf_test)
```

Evaluating the Random Forest tuned Model Performance by calculating Accuracy Precision and Recall on the classification done on the Validation Data (Hold One Out Method) **We don't see a sigificant improvement in accuracy, recall and F1 score between the baseline model and fine tuned model. Also since the validation data is producing an accuracy inline with the training data accuracy we can be assured that the model is not overfitting on the training data**

```
disease_prediction_rf_test$predicted_disease_rf_tuned <- predict_disease_rf_tuned
conf_matrix <- data.frame(table(disease_prediction_rf_test$Disease,disease_prediction_rf_test$pr
edicted_disease_rf_tuned))
colnames(conf_matrix)<- c('Actual class','Predicted Class','Count')

Accuracy_DT <- sum(conf_matrix$Count[conf_matrix$`Actual class`==conf_matrix$`Predicted Class
`])/sum(conf_matrix$Count)
Precision_DT <- conf_matrix$Count[conf_matrix$`Actual class`== 1 & conf_matrix$`Predicted Class`
== 1]/sum(conf_matrix$Count[conf_matrix$`Predicted Class`==1])
Recall_DT <- conf_matrix$Count[conf_matrix$`Actual class`==1 & conf_matrix$`Predicted Class`==1
]/sum(conf_matrix$Count[conf_matrix$`Actual class`==1])
F1_score_DT <- (2*Precision_DT*Recall_DT)/(Precision_DT+Recall_DT)

paste("Accuracy of Tuned RF Algorithm in classifying Disease is :",Accuracy_DT)
```

```
## [1] "Accuracy of Tuned RF Algorithm in classifying Disease is : 0.734149659863946"
```

```
paste("Precision of Tuned RF Algorithm in classifying Disease is :",Precision_DT)
```

```
## [1] "Precision of Tuned RF Algorithm in classifying Disease is : 0.762047291230171"
```

```
paste("Recall of Tuned RF Algorithm in classifying Disease is :",Recall_DT)
```

```
## [1] "Recall of Tuned RF Algorithm in classifying Disease is : 0.687179487179487"
```

```
paste("F1 Score of Tuned RF Algorithm in classifying Disease is :",F1_score_DT)
```

```
## [1] "F1 Score of Tuned RF Algorithm in classifying Disease is : 0.722679534487653"
```

Performing Grid Search by tuning the same hyperparameters to maximize Recall

```
control <- trainControl(method="repeatedcv", number=3, repeats=3, search="grid",summaryFunction
 = twoClassSummary)
tunegrid <- expand.grid(.mtry=c(2,3,4))
modellist <- list()
for (ntree in c(400,600,800,1000)) {
    set.seed(100)
    tuned_model_rf_recall <- train(Disease~., data=disease_prediction_rf_train, method="rf", met
ric="Sens", tuneGrid=tunegrid, trControl=control, ntree=ntree)
    key <- toString(ntree)
    modellist[[key]] <- tuned_model_rf_recall
}
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, : There were m
issing values in resampled performance measures.

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, : There were m
issing values in resampled performance measures.

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, : There were m
issing values in resampled performance measures.

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, : There were m
issing values in resampled performance measures.
```

```
predict_disease_rf_tuned_recall <- predict(tuned_model_rf_recall, newdata = disease_prediction_r
f_test)
```

Evaluating the Random Forest tuned Model Performance by calculating Accuracy Precision and Recall on the classification done on the Validation Data Since the Recall is not improving between the 2 models, we will go ahead with the first model (the model that maximizes accuracy) that was built by maximizing Accuracy

```
disease_prediction_rf_test$predicted_disease_rf_tuned_recall <- predict_disease_rf_tuned_recall
conf_matrix <- data.frame(table(disease_prediction_rf_test$Disease,disease_prediction_rf_test$pr
edicted_disease_rf_tuned_recall))
colnames(conf_matrix)<- c('Actual class','Predicted Class','Count')

Accuracy_DT <- sum(conf_matrix$Count[conf_matrix$`Actual class`==conf_matrix$`Predicted Class
`])/sum(conf_matrix$Count)
Precision_DT <- conf_matrix$Count[conf_matrix$`Actual class`== 1 & conf_matrix$`Predicted Class`
== 1]/sum(conf_matrix$Count[conf_matrix$`Predicted Class`==1])
Recall_DT <- conf_matrix$Count[conf_matrix$`Actual class`==1 & conf_matrix$`Predicted Class`==1
]/sum(conf_matrix$Count[conf_matrix$`Actual class`==1])
F1_score_DT <- (2*Precision_DT*Recall_DT)/(Precision_DT+Recall_DT)

paste("Accuracy of Tuned RF Algorithm in classifying Disease is :",Accuracy_DT)
```

```
## [1] "Accuracy of Tuned RF Algorithm in classifying Disease is : 0.735510204081633"
```

```
paste("Precision of Tuned RF Algorithm in classifying Disease is :",Precision_DT)
```

```
## [1] "Precision of Tuned RF Algorithm in classifying Disease is : 0.762835820895522"
```

```
paste("Recall of Tuned RF Algorithm in classifying Disease is :",Recall_DT)
```

```
## [1] "Recall of Tuned RF Algorithm in classifying Disease is : 0.68974358974359"
```

```
paste("F1 Score of Tuned RF Algorithm in classifying Disease is :",F1_score_DT)
```

```
## [1] "F1 Score of Tuned RF Algorithm in classifying Disease is : 0.724450744153083"
```

ROC & AUROC for the best Random Forest Model

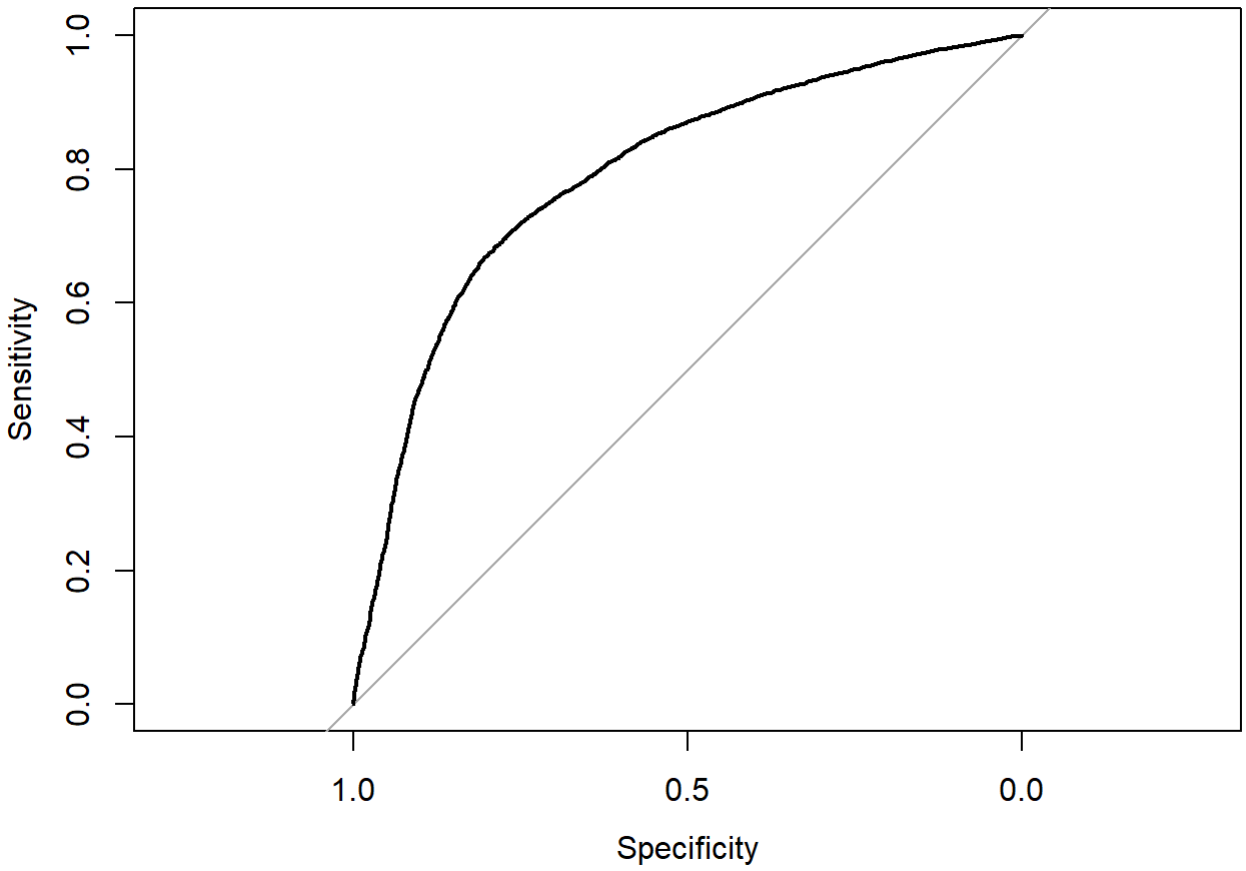
Generated ROC curve and calculated Area Under Curve metric for the identified best performing RF model. The ROC curve is plotted by obtaining probabilities of the target variable classes.

```
rf_pred_disease <- predict(tuned_model_rf, newdata = disease_prediction_rf_test, na.action = na.
omit, type = "prob")
roc_curve <- roc(disease_prediction_rf_test$Disease,rf_pred_disease$`1`)
```

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

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

```
plot(roc_curve)
```



Obtained the area under the ROC curve for the best RF model

```
paste("Area Under the ROC curve is :",auc(roc_curve))
```

```
## [1] "Area Under the ROC curve is : 0.791343742653034"
```

END OF BUILDING RF Model (ML algorithm number 3)

2.4 Gradient Boosting Algorithm

Creating Training and Validation data splits (Hold one out method)

```
library(caret)
set.seed(188)
train_index <- createDataPartition(disease_prediction_training$Disease, p = 0.7, list = FALSE)

disease_prediction_gb_train <- disease_prediction_training[train_index, ]
disease_prediction_gb_train$Disease <- as.factor(disease_prediction_gb_train$Disease)


disease_prediction_gb_test <- disease_prediction_training[-train_index, ]
disease_prediction_gb_test$Disease <- as.factor(disease_prediction_gb_test$Disease)
```

```
#install.packages("gbm")
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 3.6.3
```

```
## Loaded gbm 2.1.5
```

Running baseline model for Gradient Boosting Algorithm

```
#baselinemodel_gb <- train(Disease ~ ., data = disease_prediction_gb_train, method = "gbm")
```

From the below results we can observe the accuracies of the baseline model for each value of n.trees and interaction.depth. The combinations of depth and number of trees that produced the best accuracy was chosen as the final model

```
baselinemodel_gb
```

```
## Stochastic Gradient Boosting
##
## 34300 samples
##    12 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 34300, 34300, 34300, 34300, 34300, 34300, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7258158  0.4514465
##   1                  100      0.7296882  0.4592068
##   1                  150      0.7320394  0.4639253
##   2                   50      0.7326965  0.4652638
##   2                  100      0.7356241  0.4711474
##   2                  150      0.7361597  0.4722255
##   3                   50      0.7352901  0.4705025
##   3                  100      0.7366092  0.4731513
##   3                  150      0.7365389  0.4730157
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 100, interaction.depth = 3, shrinkage = 0.
1 and n.minobsinnode = 10.
```

```
predict_disease_gb_base <- predict(baselinemodel_gb, newdata = disease_prediction_gb_test)
```

Evaluating the Gradient Boosting Baseline Model Performance by calculating Accuracy Precision and Recall on the classification done on the Validation Data (Hold One Out method) ***By verifying that the accuracy of the model on training data and validation data are in line, we can ensure that the model is not overfitting on the training data.***

```
disease_prediction_gb_test$predicted_disease_gb_baseline <- predict_disease_gb_base
conf_matrix <- data.frame(table(disease_prediction_gb_test$Disease,disease_prediction_gb_test$pr
edicted_disease_gb_baseline))
colnames(conf_matrix)<- c('Actual class','Predicted Class','Count')

Accuracy_DT <- sum(conf_matrix$Count[conf_matrix$`Actual class`==conf_matrix$`Predicted Class
`])/sum(conf_matrix$Count)
Precision_DT <- conf_matrix$Count[conf_matrix$`Actual class`== 1 & conf_matrix$`Predicted Class`
== 1]/sum(conf_matrix$Count[conf_matrix$`Predicted Class`==1])
Recall_DT <- conf_matrix$Count[conf_matrix$`Actual class`==1 & conf_matrix$`Predicted Class`==1
]/sum(conf_matrix$Count[conf_matrix$`Actual class`==1])
F1_score_DT <- (2*Precision_DT*Recall_DT)/(Precision_DT+Recall_DT)

paste("Accuracy of baseline GB Algorithm in classifying Disease is :",Accuracy_DT)
```

```
## [1] "Accuracy of baseline GB Algorithm in classifying Disease is : 0.73312925170068"
```

```
paste("Precision of baseline GB Algorithm in classifying Disease is :",Precision_DT)
```

```
## [1] "Precision of baseline GB Algorithm in classifying Disease is : 0.758181548941211"
```

```
paste("Recall of baseline GB Algorithm in classifying Disease is :",Recall_DT)
```

```
## [1] "Recall of baseline GB Algorithm in classifying Disease is : 0.690958164642375"
```

```
paste("F1 Score of baseline GB Algorithm in classifying Disease is :",F1_score_DT)
```

```
## [1] "F1 Score of baseline GB Algorithm in classifying Disease is : 0.723010661582998"
```

Performing Grid Search by tuning hyperparameters to maximize Accuracy

The hyperparameters are interaction.depth (tree depth) which controls the complexity of the trees, n.trees to limit the number of iterations(trees), shrinkage which measures the learning rate and n.minobsinnode which denotes the number of rows in a node to begin splitting. For controlling overfitting we should limit the depth of the tree and increase the number of trees and also increase the minimum number of rows required in a node Increasing the tree depth may reduce the bias but can increase variance, whereas increasing the number of trees can reduce variance and increase bias. 3 Fold Cross Validation technique is also used to train the model on the training data

```
#set.seed(188)
#gbm_tuned <- train(Disease ~ ., data = disease_prediction_gb_train,
#                       tuneGrid = expand.grid(interaction.depth=c(1, 3, 5), n.trees = c(50,100,
150,200),
#                       shrinkage=c(0.01, 0.05,0.1),n.minobsinnode=c(5,10)), method = "gbm",
#                       trControl = trainControl(method="cv", number=3, repeats = 3),
#                       metric="Accuracy")
```

From the below table we can see the training data Accuracies for different values of shrinkage, interaction depth, minobsinnode, number of trees The model with the best training data accuracy is chosen as the final model

```
gbm_tuned
```

```
## Stochastic Gradient Boosting
##
## 34300 samples
##    12 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 22866, 22867, 22867
## Resampling results across tuning parameters:
##
##   shrinkage  interaction.depth  n.minobsinnode  n.trees  Accuracy   Kappa
##   0.01       1                   5               50      0.7185715  0.4367866
##   0.01       1                   5              100      0.7186006  0.4368448
##   0.01       1                   5              150      0.7186298  0.4369031
##   0.01       1                   5              200      0.7186006  0.4368448
##   0.01       1                  10               50      0.7186298  0.4369031
##   0.01       1                  10              100      0.7186298  0.4369031
##   0.01       1                  10              150      0.7186298  0.4369031
##   0.01       1                  10              200      0.7186298  0.4369031
##   0.01       3                   5               50      0.7265599  0.4528477
##   0.01       3                   5              100      0.7276677  0.4550891
##   0.01       3                   5              150      0.7281050  0.4559760
##   0.01       3                   5              200      0.7289213  0.4576257
##   0.01       3                  10               50      0.7246066  0.4489227
##   0.01       3                  10              100      0.7276386  0.4550272
##   0.01       3                  10              150      0.7279593  0.4556858
##   0.01       3                  10              200      0.7289797  0.4577394
##   0.01       5                   5               50      0.7279301  0.4556291
##   0.01       5                   5              100      0.7279301  0.4556262
##   0.01       5                   5              150      0.7289214  0.4576160
##   0.01       5                   5              200      0.7300584  0.4599184
##   0.01       5                  10               50      0.7277260  0.4552130
##   0.01       5                  10              100      0.7278426  0.4554482
##   0.01       5                  10              150      0.7287756  0.4573243
##   0.01       5                  10              200      0.7300875  0.4599785
##   0.05       1                   5               50      0.7186006  0.4368448
##   0.05       1                   5              100      0.7259768  0.4516332
##   0.05       1                   5              150      0.7290380  0.4577686
##   0.05       1                   5              200      0.7292712  0.4582450
##   0.05       1                  10               50      0.7186298  0.4369031
##   0.05       1                  10              100      0.7256560  0.4509922
##   0.05       1                  10              150      0.7283383  0.4563705
##   0.05       1                  10              200      0.7290963  0.4578928
##   0.05       3                   5               50      0.7308455  0.4614737
##   0.05       3                   5              100      0.7355977  0.4710255
##   0.05       3                   5              150      0.7367931  0.4734361
##   0.05       3                   5              200      0.7376677  0.4751927
##   0.05       3                  10               50      0.7305831  0.4609519
##   0.05       3                  10              100      0.7354228  0.4706717
##   0.05       3                  10              150      0.7375219  0.4748849
##   0.05       3                  10              200      0.7370554  0.4739578
##   0.05       5                   5               50      0.7328864  0.4656184
```

```
##   0.05       5                   5              100      0.7370846  0.4740498
##   0.05       5                   5              150      0.7371429  0.4741743
##   0.05       5                   5              200      0.7374345  0.4747567
##   0.05       5                  10               50      0.7332071  0.4662377
##   0.05       5                  10              100      0.7369680  0.4737921
##   0.05       5                  10              150      0.7366765  0.4732300
##   0.05       5                  10              200      0.7372595  0.4744003
##   0.10       1                   5               50      0.7262391  0.4521554
##   0.10       1                   5              100      0.7296211  0.4589457
##   0.10       1                   5              150      0.7323324  0.4643920
##   0.10       1                   5              200      0.7336735  0.4670782
##   0.10       1                  10               50      0.7252770  0.4502294
##   0.10       1                  10              100      0.7287756  0.4572467
##   0.10       1                  10              150      0.7319534  0.4636264
##   0.10       1                  10              200      0.7332654  0.4662626
##   0.10       3                   5               50      0.7354228  0.4706752
##   0.10       3                   5              100      0.7375219  0.4748887
##   0.10       3                   5              150      0.7366181  0.4730991
##   0.10       3                   5              200      0.7372595  0.4743867
##   0.10       3                  10               50      0.7358601  0.4715477
##   0.10       3                  10              100      0.7369388  0.4737349
##   0.10       3                  10              150      0.7372304  0.4743284
##   0.10       3                  10              200      0.7366764  0.4732207
##   0.10       5                   5               50      0.7363558  0.4725791
##   0.10       5                   5              100      0.7377843  0.4754543
##   0.10       5                   5              150      0.7360059  0.4718974
##   0.10       5                   5              200      0.7363848  0.4726534
##   0.10       5                  10               50      0.7370846  0.4740428
##   0.10       5                  10              100      0.7377843  0.4754537
##   0.10       5                  10              150      0.7362391  0.4723620
##   0.10       5                  10              200      0.7358018  0.4714775
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 100, interaction.depth = 5, shrinkage = 0.
1 and n.minobsinnode = 10.
```

```
predict_disease_gb_tuned <- predict(gbm_tuned, newdata = disease_prediction_gb_test)
```

Evaluating the Grdient Boosting tuned Model Performance by calculating Accuracy Precision and Recall on the classification done on the Validation Data **We can see an improvement in recall and F1 score between the baseline model and fine tuned model. Since the accuracies of the training data is comparable to the validation data accuracies, we can be sure that the model is not overfitting on the training data**

```
disease_prediction_gb_test$predicted_disease_gb_tuned <- predict_disease_gb_tuned
conf_matrix <- data.frame(table(disease_prediction_gb_test$Disease,disease_prediction_gb_test$pr
edicted_disease_gb_tuned))
colnames(conf_matrix)<- c('Actual class','Predicted Class','Count')

Accuracy_DT <- sum(conf_matrix$Count[conf_matrix$`Actual class`==conf_matrix$`Predicted Class
`])/sum(conf_matrix$Count)
Precision_DT <- conf_matrix$Count[conf_matrix$`Actual class`== 1 & conf_matrix$`Predicted Class`
== 1]/sum(conf_matrix$Count[conf_matrix$`Predicted Class`==1])
Recall_DT <- conf_matrix$Count[conf_matrix$`Actual class`==1 & conf_matrix$`Predicted Class`==1
]/sum(conf_matrix$Count[conf_matrix$`Actual class`==1])
F1_score_DT <- (2*Precision_DT*Recall_DT)/(Precision_DT+Recall_DT)

paste("Accuracy of tuned GB Algorithm in classifying Disease is :",Accuracy_DT)
```

```
## [1] "Accuracy of tuned GB Algorithm in classifying Disease is : 0.732789115646258"
```

```
paste("Precision of tuned GB Algorithm in classifying Disease is :",Precision_DT)
```

```
## [1] "Precision of tuned GB Algorithm in classifying Disease is : 0.752538439222512"
```

```
paste("Recall of tuned GB Algorithm in classifying Disease is :",Recall_DT)
```

```
## [1] "Recall of tuned GB Algorithm in classifying Disease is : 0.700134952766532"
```

```
paste("F1 Score of tuned GB Algorithm in classifying Disease is :",F1_score_DT)
```

```
## [1] "F1 Score of tuned GB Algorithm in classifying Disease is : 0.725391498881432"
```

ROC & AUROC for the best Gradient Boost Model

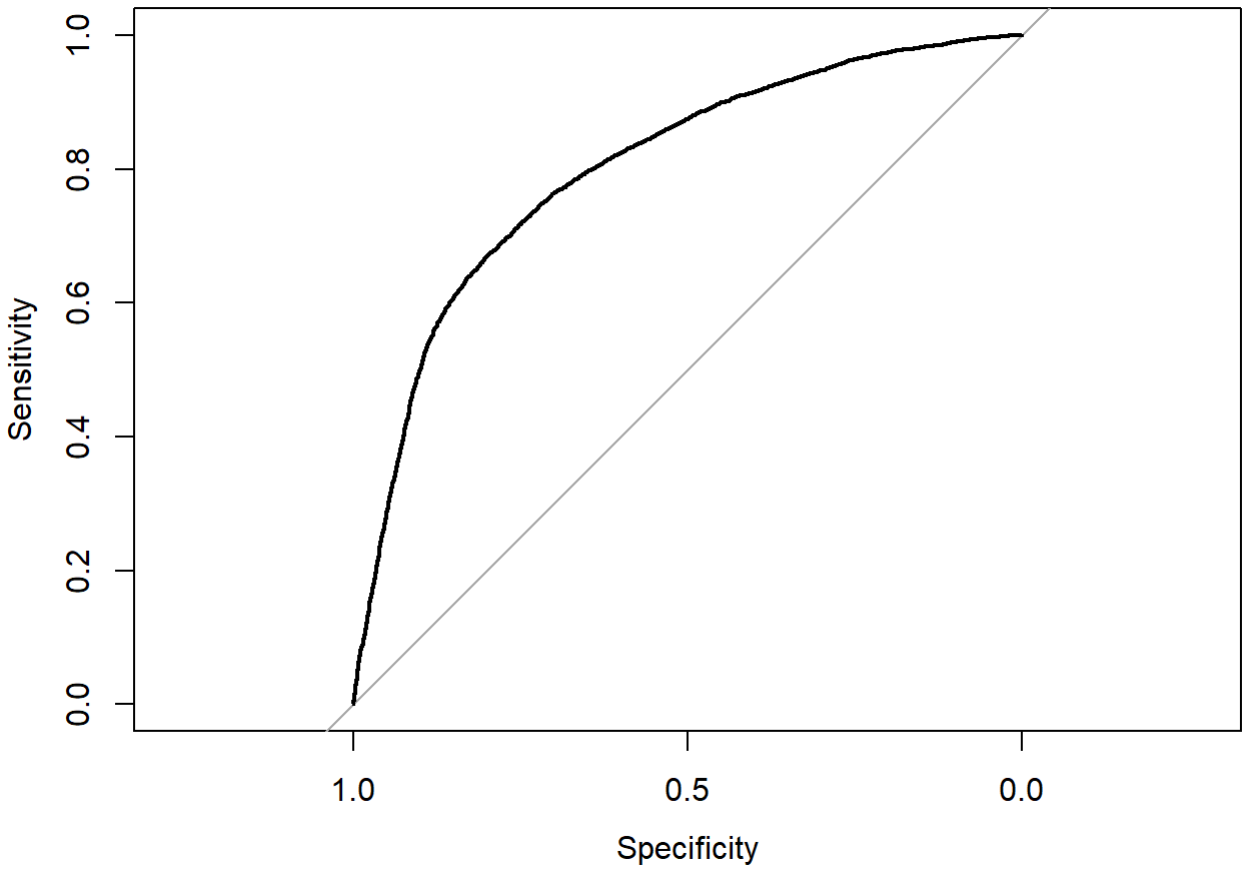
Generated ROC curve and calculated Area Under Curve metric for the identified best performing Gradient Boosting Model

```
library(pROC)
gb_pred_disease <- predict(gbm_tuned, newdata = disease_prediction_gb_test, na.action = na.omit,
type = "prob")
roc_curve <- roc(disease_prediction_gb_test$Disease,gb_pred_disease$`1`)
```

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

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

```
plot(roc_curve)
```



Obtained the area under the ROC curve for the best GB model

```
paste("Area Under the ROC curve is :",auc(roc_curve))
```

```
## [1] "Area Under the ROC curve is : 0.800688203573194"
```

END OF BUILDING GB Model (ML algorithm number 4)

2.5 LINEAR SUPPORT VECTOR MACHINE

Creating Training and Validation data splits

```
library(caret)
set.seed(188)
train_index <- createDataPartition(disease_prediction_training_knn_svm$Disease, p = 0.7, list =
FALSE)

disease_prediction_svml_train <- disease_prediction_training_knn_svm[train_index, ]
disease_prediction_svml_train$Disease <- as.factor(disease_prediction_svml_train$Disease)
disease_prediction_svml_test <- disease_prediction_training_knn_svm[-train_index, ]
disease_prediction_svml_test$Disease <- as.factor(disease_prediction_knn_test$Disease)
disease_prediction_svml_train$Height <- NULL
disease_prediction_svml_train$Weight <- NULL
disease_prediction_svml_test$Height <- NULL
disease_prediction_svml_test$Weight <- NULL
```

```
#install.packages("mlbench")
library(mlbench)
```

```
## Warning: package 'mlbench' was built under R version 3.6.3
```

Running baseline model for Linear SVM Algorithm

```
#baselinemodel_svml <- train(Disease ~ ., data = disease_prediction_svml_train, method = "svmLin
ear")
```

The below result shows the accuracy of the baseline model

```
baselinemodel_svml
```

```
## Support Vector Machines with Linear Kernel
##
## 34301 samples
##    14 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 34301, 34301, 34301, 34301, 34301, 34301, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.7264849  0.4530504
##
## Tuning parameter 'C' was held constant at a value of 1
```

```
predict_disease_svml_base <- predict(baselinemodel_svml, newdata = disease_prediction_svml_test)
```

Evaluating the Linear SVM Baseline Model Performance by calculating Accuracy Precision and Recall on the classification done on the Validation Data (Hold One out method to ensure that the model is performing well on the data that it was trained on and also on a new data that it has not seen before) **The model produced an accuracy on validation data is inline with the accuracy obtained on training data, hence the model is not overfitting on train data.**

```
disease_prediction_svml_test$predicted_disease_svml_baseline <- predict_disease_svml_base
conf_matrix <- data.frame(table(disease_prediction_svml_test$Disease,disease_prediction_svml_tes
t$predicted_disease_svml_baseline))
colnames(conf_matrix)<- c('Actual class','Predicted Class','Count')

Accuracy_DT <- sum(conf_matrix$Count[conf_matrix$`Actual class`==conf_matrix$`Predicted Class
`])/sum(conf_matrix$Count)
Precision_DT <- conf_matrix$Count[conf_matrix$`Actual class`== 1 & conf_matrix$`Predicted Class`
== 1]/sum(conf_matrix$Count[conf_matrix$`Predicted Class`==1])
Recall_DT <- conf_matrix$Count[conf_matrix$`Actual class`==1 & conf_matrix$`Predicted Class`==1
]/sum(conf_matrix$Count[conf_matrix$`Actual class`==1])
F1_score_DT <- (2*Precision_DT*Recall_DT)/(Precision_DT+Recall_DT)

paste("Accuracy of baseline SVM Linear Algorithm in classifying Disease is :",Accuracy_DT)
```

```
## [1] "Accuracy of baseline SVM Linear Algorithm in classifying Disease is : 0.728961153819988"
```

```
paste("Precision of baseline SVM Linear Algorithm in classifying Disease is :",Precision_DT)
```

```
## [1] "Precision of baseline SVM Linear Algorithm in classifying Disease is : 0.79900479829394"
```

```
paste("Recall of baseline SVM Linear Algorithm in classifying Disease is :",Recall_DT)
```

```
## [1] "Recall of baseline SVM Linear Algorithm in classifying Disease is : 0.611783916179072"
```

```
paste("F1 Score of baseline SVM Linear Algorithm in classifying Disease is :",F1_score_DT)
```

```
## [1] "F1 Score of baseline SVM Linear Algorithm in classifying Disease is : 0.692971639950678"
```

Performing Grid Search by tuning hyperparameters to maximize Accuracy

The hyperparameters Cost Function (C) is being tuned to produce the best model with low bias and variance. By increasing C, the bias is reduced and variance is increased thereby increasing the risk of overfitting, whereas by decreasing C the variance will decrease and reduce the risk of overfitting, but the bias increases which in turn increases the risk of underfitting. Hence an optimum value of C has to be chosen which produces the right balance between Bias and Variance

```
#set.seed(188)
#svml_tuned <- train(Disease ~ ., data = disease_prediction_svml_train,
#                          method = "svmLinear",
#                          trControl = trainControl(method = "cv", number = 3,repeats = 3),
#                          tuneGrid = expand.grid(C = seq(0, 1, 0.05)),
#                          metric="Accuracy")
```

From the below table we can see the Accuracies for different values of Cost Function for Linear SVM models The model which prodcued the best accuracy was chosen as the final model

```
svml_tuned
```

```
## Support Vector Machines with Linear Kernel
##
## 34301 samples
##    14 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 22867, 22868, 22867
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.00        NaN        NaN
##   0.05  0.7279963  0.4559839
##   0.10  0.7281129  0.4562171
##   0.15  0.7281129  0.4562171
##   0.20  0.7281712  0.4563337
##   0.25  0.7281712  0.4563337
##   0.30  0.7281129  0.4562171
##   0.35  0.7281712  0.4563337
##   0.40  0.7282003  0.4563920
##   0.45  0.7282295  0.4564503
##   0.50  0.7282003  0.4563920
##   0.55  0.7281420  0.4562754
##   0.60  0.7282295  0.4564503
##   0.65  0.7281420  0.4562754
##   0.70  0.7281712  0.4563337
##   0.75  0.7282003  0.4563920
##   0.80  0.7281129  0.4562171
##   0.85  0.7282003  0.4563920
##   0.90  0.7282586  0.4565086
##   0.95  0.7282295  0.4564503
##   1.00  0.7281712  0.4563337
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.9.
```

```
predict_disease_svml_tuned <- predict(svml_tuned, newdata = disease_prediction_svml_test)
```

Evaluating the Linear SVM tuned model Performance by calculating Accuracy Precision and Recall on the classification done on the Validation Data (Hold One out method to ensure that the model is performing well on the data that it was trained on and also on a new data that it has not seen before) **The Accuracy score on validation data is in line with the accuracy produced on training data and hence the model is not overfitting. We can see not much significant difference in the metrics scores between the baseline model and fine tuned model.**

```
disease_prediction_svml_test$predict_disease_svml_tuned <- predict_disease_svml_tuned
conf_matrix <- data.frame(table(disease_prediction_svml_test$Disease,disease_prediction_svml_tes
t$predict_disease_svml_tuned))
colnames(conf_matrix)<- c('Actual class','Predicted Class','Count')

Accuracy_DT <- sum(conf_matrix$Count[conf_matrix$`Actual class`==conf_matrix$`Predicted Class
`])/sum(conf_matrix$Count)
Precision_DT <- conf_matrix$Count[conf_matrix$`Actual class`== 1 & conf_matrix$`Predicted Class`
== 1]/sum(conf_matrix$Count[conf_matrix$`Predicted Class`==1])
Recall_DT <- conf_matrix$Count[conf_matrix$`Actual class`==1 & conf_matrix$`Predicted Class`==1
]/sum(conf_matrix$Count[conf_matrix$`Actual class`==1])
F1_score_DT <- (2*Precision_DT*Recall_DT)/(Precision_DT+Recall_DT)

paste("Accuracy of tuned Linear SVM Algorithm in classifying Disease is :",Accuracy_DT)
```

```
## [1] "Accuracy of tuned Linear SVM Algorithm in classifying Disease is : 0.728893121981087"
```

```
paste("Precision of tuned Linear SVM Algorithm in classifying Disease is :",Precision_DT)
```

```
## [1] "Precision of tuned Linear SVM Algorithm in classifying Disease is : 0.798862828713575"
```

```
paste("Recall of tuned Linear SVM Algorithm in classifying Disease is :",Recall_DT)
```

```
## [1] "Recall of tuned Linear SVM Algorithm in classifying Disease is : 0.611783916179072"
```

```
paste("F1 Score of tuned Linear SVM Algorithm in classifying Disease is :",F1_score_DT)
```

```
## [1] "F1 Score of tuned Linear SVM Algorithm in classifying Disease is : 0.692918239963011"
```

END OF BUILDING Linear SVM Model (ML algorithm number 5)

2.6 NON LINEAR SUPPORT VECTOR MACHINE

Creating Training and Validation data splits

```
library(caret)
set.seed(188)
train_index <- createDataPartition(disease_prediction_training_knn_svm$Disease, p = 0.7, list =
FALSE)

disease_prediction_svmnl_train <- disease_prediction_training_knn_svm[train_index, ]
disease_prediction_svmnl_train$Disease <- as.factor(disease_prediction_svmnl_train$Disease)
disease_prediction_svmnl_test <- disease_prediction_training_knn_svm[-train_index, ]
disease_prediction_svmnl_test$Disease <- as.factor(disease_prediction_svmnl_test$Disease)
disease_prediction_svmnl_train$Height <- NULL
disease_prediction_svmnl_train$Weight <- NULL
disease_prediction_svmnl_test$Height <- NULL
disease_prediction_svmnl_test$Weight <- NULL
```

```
#install.packages("mlbench")
library(mlbench)
```

Running baseline model for Non Linear SVM using RBF kernel

```
#baselinemodel_svmnl <- train(Disease ~ ., data = disease_prediction_svmnl_train, method = "svmR
adial")
```

The result shows the accuracy of the baseline model on training dataset for different values of C

```
baselinemodel_svmnl
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 34301 samples
##    14 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 34301, 34301, 34301, 34301, 34301, 34301, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.7305060  0.4610574
##   0.50  0.7298698  0.4597893
##   1.00  0.7291822  0.4584165
##
## Tuning parameter 'sigma' was held constant at a value of 0.09847744
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.09847744 and C = 0.25.
```

```
predict_disease_svmnl_base <- predict(baselinemodel_svmnl, newdata = disease_prediction_svmnl_te
st)
```

Evaluating the Linear SVM Baseline Model Performance by calculating Accuracy Precision and Recall on the classification done on the Validation Data (Hold One out method to ensure that the model is performing well on the data that it was trained on and also on a new data that it has not seen before) **The model accuracy on validation data is inline with the accuracy obtained on training data, hence the model is not ovverfitting on train data.**

```
disease_prediction_svmnl_test$predicted_disease_svmnl_baseline <- predict_disease_svmnl_base
conf_matrix <- data.frame(table(disease_prediction_svmnl_test$Disease,disease_prediction_svmnl_t
est$predicted_disease_svmnl_baseline))
colnames(conf_matrix)<- c('Actual class','Predicted Class','Count')

Accuracy_DT <- sum(conf_matrix$Count[conf_matrix$`Actual class`==conf_matrix$`Predicted Class
`])/sum(conf_matrix$Count)
Precision_DT <- conf_matrix$Count[conf_matrix$`Actual class`== 1 & conf_matrix$`Predicted Class`
== 1]/sum(conf_matrix$Count[conf_matrix$`Predicted Class`==1])
Recall_DT <- conf_matrix$Count[conf_matrix$`Actual class`==1 & conf_matrix$`Predicted Class`==1
]/sum(conf_matrix$Count[conf_matrix$`Actual class`==1])
F1_score_DT <- (2*Precision_DT*Recall_DT)/(Precision_DT+Recall_DT)

paste("Accuracy of baseline SVM Non Linear Algorithm in classifying Disease is :",Accuracy_DT)
```

```
## [1] "Accuracy of baseline SVM Non Linear Algorithm in classifying Disease is : 0.738553643104
973"
```

```
paste("Precision of baseline SVM Non Linear Algorithm in classifying Disease is :",Precision_DT)
```

```
## [1] "Precision of baseline SVM Non Linear Algorithm in classifying Disease is : 0.78210492436
4339"
```

```
paste("Recall of baseline SVM Non Linear Algorithm in classifying Disease is :",Recall_DT)
```

```
## [1] "Recall of baseline SVM Non Linear Algorithm in classifying Disease is : 0.661314464553"
```

```
paste("F1 Score of baseline SVM Non Linear Algorithm in classifying Disease is :",F1_score_DT)
```

```
## [1] "F1 Score of baseline SVM Non Linear Algorithm in classifying Disease is : 0.716655607166
556"
```

Performing Grid Search by tuning hyperparameters to maximize Accuracy

The hyperparameters Cost Function (C) is being tuned to produce the best model with low bias and variance. By increasing C, the bias is reduced and variance is increased thereby increasing the risk of overfitting, whereas by decreasing C the variance will decrease and reduce the risk of overfitting, but the bias increases which in turn increases the risk of underfitting. Hence an optimum value of C has to be chosen which produces the right balance between Bias and Variance Similarly, higher sigma (gamma) increases variance, thereby increasing the risk of overfitting and reduces bias. Lower sigma increases bias and reduces variance thereby increasing the risk of underfitting.Hence both C and gamma move in the same direction when it comes to impacting the model performance. 3 Fold Cross Validation is also done to control for overfitting

```
#set.seed(188)
#svmnl_tuned <- train(Disease ~ ., data = disease_prediction_svmnl_train,
#                          method = "svmRadial",
#                          trControl = trainControl(method = "cv", number = 3,repeats = 3),
#                          tuneGrid = expand.grid(sigma = c(0.1,0.5,1),
#                                                 C = c(0.1,0.5,1
#                                                       )),
#                          metric="Accuracy")
```

From the below table we can see the Accuracies for different values of Cost Function and sigma for Non Linear SVM models The model which produced the best accuracy was chosen as the final model

```
svmnl_tuned
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 34301 samples
##    14 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 22867, 22868, 22867
## Resampling results across tuning parameters:
##
##   sigma  C    Accuracy   Kappa
##   0.1    0.1  0.7286377  0.4572710
##   0.1    0.5  0.7307659  0.4615267
##   0.1    1.0  0.7302411  0.4604769
##   0.5    0.1  0.7184047  0.4368106
##   0.5    0.5  0.7271800  0.4543574
##   0.5    1.0  0.7266844  0.4533651
##   1.0    0.1  0.7039444  0.4078946
##   1.0    0.5  0.7223696  0.4447386
##   1.0    1.0  0.7234775  0.4469521
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.1 and C = 0.5.
```

```
predict_disease_svmnl_tuned <- predict(svmnl_tuned, newdata = disease_prediction_svmnl_test)
```

Evaluating the Non Linear SVM tuned model Performance by calculating Accuracy Precision and Recall on the classification done on the Validation Data (Hold One out method to ensure that the model is performing well on the data that it was trained on and also on a new data that it has not seen before) **The Accuracy score on validation data is in line with the accuracy produced on training data and hence the model is not overfitting. We can see not much significant difference in the metrics scores between the baseline model and fine tuned model.**

```
disease_prediction_svmnl_test$predict_disease_svmnl_tuned <- predict_disease_svmnl_tuned
conf_matrix <- data.frame(table(disease_prediction_svmnl_test$Disease,disease_prediction_svmnl_t
est$predict_disease_svmnl_tuned))
colnames(conf_matrix)<- c('Actual class','Predicted Class','Count')

Accuracy_DT <- sum(conf_matrix$Count[conf_matrix$`Actual class`==conf_matrix$`Predicted Class
`])/sum(conf_matrix$Count)
Precision_DT <- conf_matrix$Count[conf_matrix$`Actual class`== 1 & conf_matrix$`Predicted Class`
== 1]/sum(conf_matrix$Count[conf_matrix$`Predicted Class`==1])
Recall_DT <- conf_matrix$Count[conf_matrix$`Actual class`==1 & conf_matrix$`Predicted Class`==1
]/sum(conf_matrix$Count[conf_matrix$`Actual class`==1])
F1_score_DT <- (2*Precision_DT*Recall_DT)/(Precision_DT+Recall_DT)

paste("Accuracy of tuned RBF SVM Algorithm in classifying Disease is :",Accuracy_DT)
```

```
## [1] "Accuracy of tuned RBF SVM Algorithm in classifying Disease is : 0.738417579427172"
```

```
paste("Precision of tuned RBF SVM Algorithm in classifying Disease is :",Precision_DT)
```

```
## [1] "Precision of tuned RBF SVM Algorithm in classifying Disease is : 0.784600389863548"
```

```
paste("Recall of tuned RBF SVM Algorithm in classifying Disease is :",Recall_DT)
```

```
## [1] "Recall of tuned RBF SVM Algorithm in classifying Disease is : 0.657232276500204"
```

```
paste("F1 Score of tuned RBF SVM Algorithm in classifying Disease is :",F1_score_DT)
```

```
## [1] "F1 Score of tuned RBF SVM Algorithm in classifying Disease is : 0.715290633098852"
```

END OF BUILDING Non Linear SVM Model (ML algorithm number 6)

Comparing Gradient Boosting Machine, Linear and Non Linear SVM models

We can observe that Gradient Boosting Machine has the best Accuracy and Kappa scores

```
model_comparison <- resamples(list(Gradient_Boosting_Machine = gbm_tuned,
                                   SVM_Linear = svml_tuned, SVM_RBF = svmnl_tuned))
summary(model_comparison)
```

```
##
## Call:
## summary.resamples(object = model_comparison)
##
## Models: Gradient_Boosting_Machine, SVM_Linear, SVM_RBF
## Number of resamples: 3
##
## Accuracy
##                               Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## Gradient_Boosting_Machine 0.7353507 0.7366511 0.7379515 0.7377843 0.7390011 0.7400507    0
## SVM_Linear                0.7273045 0.7275550 0.7278055 0.7282586 0.7287357 0.7296659    0
## SVM_RBF                   0.7260801 0.7290857 0.7320913 0.7307659 0.7331088 0.7341263    0
##
## Kappa
##                               Min.   1st Qu.    Median      Mean   3rd Qu.     Max. NA's
## Gradient_Boosting_Machine 0.4706024 0.4732066 0.4758108 0.4754537 0.4778794 0.479948    0
## SVM_Linear                0.4546091 0.4551060 0.4556029 0.4565086 0.4574584 0.459314    0
## SVM_RBF                   0.4521602 0.4581691 0.4641780 0.4615267 0.4662100 0.468242    0
```

END of Model Building, Tuning and Evaluation

SECTION 3: PREDICTION and INTERPRETATION

Importing the TEST Dataset to apply all the built models on the test dataset to predict if each person in the testing dataset has the disease

```
setwd("C:/Users/bhavi/OneDrive/Desktop/SYR ADS/Sem 2/IST_707_Data_Analytics/HW3")

getwd
```

```
## function ()
## .Internal(getwd())
## <bytecode: 0x0000019724700e38>
## <environment: namespace:base>
```

```
disease_prediction_testing <- read.csv("Disease Prediction Testing.csv")
```

VIEWING THE STRUCTURE and SUMMARY STATISTICS of the Data and checking for missing values

```
str(disease_prediction_testing)
```

```
## 'data.frame':    21000 obs. of  12 variables:
##  $ ID                 : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ Age                : int  44 41 63 55 55 58 45 52 58 52 ...
##  $ Gender             : Factor w/ 2 levels "female","male": 1 1 2 1 1 1 1 1 2 1 ...
##  $ Height             : int  160 169 168 158 167 162 161 149 168 165 ...
##  $ Weight             : num  59 74 84 108 67 95 68 85 64 92 ...
##  $ High.Blood.Pressure: int  100 120 120 160 120 130 120 160 140 150 ...
##  $ Low.Blood.Pressure : int  80 70 80 100 80 70 70 90 90 100 ...
##  $ Cholesterol        : Factor w/ 3 levels "high","normal",..: 1 2 2 2 2 2 2 2 2 2 ...
##  $ Glucose            : Factor w/ 3 levels "high","normal",..: 2 2 1 2 2 2 2 2 2 2 ...
##  $ Smoke              : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Alcohol            : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Exercise           : int  1 1 1 0 1 1 1 1 1 1 ...
```

```
summary(disease_prediction_testing)
```

```
##        ID             Age           Gender          Height          Weight       High.Blood.P
ressure Low.Blood.Pressure   Cholesterol        Glucose
##  Min.   :    0   Min.   :29.00   female:13667   Min.   : 64.0   Min.   : 21.00   Min.   :   1
0.0     Min.   : -70.00    high    : 2844   high    : 1563
##  1st Qu.: 5250   1st Qu.:48.00   male  : 7333   1st Qu.:159.0   1st Qu.: 65.00   1st Qu.:  12
0.0     1st Qu.:  80.00    normal  :15709   normal  :17827
##  Median :10500   Median :53.00                  Median :165.0   Median : 72.00   Median :  12
0.0     Median :  80.00    too high: 2447   too high: 1610
##  Mean   :10500   Mean   :52.81                  Mean   :164.3   Mean   : 74.24   Mean   :  12
9.1     Mean   :  95.96
##  3rd Qu.:15749   3rd Qu.:58.00                  3rd Qu.:170.0   3rd Qu.: 82.00   3rd Qu.:  14
0.0     3rd Qu.:  90.00
##  Max.   :20999   Max.   :64.00                  Max.   :250.0   Max.   :183.00   Max.   :1602
0.0     Max.   :8500.00
##      Smoke            Alcohol           Exercise
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.000
##  1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:1.000
##  Median :0.00000   Median :0.00000   Median :1.000
##  Mean   :0.08781   Mean   :0.05267   Mean   :0.805
##  3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:1.000
##  Max.   :1.00000   Max.   :1.00000   Max.   :1.000
```

DATA PREPARATION of Testing Data

From the structure and summary of the data we can observe that the Min and Max values of the columns Low Blood Pressure and High Blood Pressure are not practically possible values and hence they are noise/outliers which need to be treated. Hence these columns need to be winsorized. Winsorization is a data treatment process where the extreme outlier values are replaced with less extreme values which are practically possible **The min & max of low BP (diastolic BP) fall in the range of 45 to 140 and hence any value that is less than 45 is replaced with 45 and any value greater than 140 is replaced with 140**

```
disease_prediction_testing$Low.Blood.Pressure[disease_prediction_testing$Low.Blood.Pressure<45]
 <- 45

disease_prediction_testing$Low.Blood.Pressure[disease_prediction_testing$Low.Blood.Pressure>140]
<- 140
```

Verifying that the Min & Max values of Low Blood Pressure (Diastolic BP) are in the correct practically permissible range and the outliers have been eliminated

```
summary(disease_prediction_testing$Low.Blood.Pressure)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   45.00   80.00   80.00   82.13   90.00  140.00
```

The possible values for the min & max of High BP (Systolic BP) fall in the range of 70 to 200 and hence any value that is less than 70 is replaced with 70 and any value greater than 200 is replaced with 200

```
disease_prediction_testing$High.Blood.Pressure[disease_prediction_testing$High.Blood.Pressure<70
] <- 70

disease_prediction_testing$High.Blood.Pressure[disease_prediction_testing$High.Blood.Pressure>20
0] <- 200
```

Verifying that the Min & Max values of High Blood Pressure (Systolic BP) are in the correct practically permissible range and the outliers have been eliminated

```
summary(disease_prediction_testing$High.Blood.Pressure)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    70.0   120.0   120.0   126.8   140.0   200.0
```

There are 84 instances where Low BP is > high BP even after winsorizing which needs to treated by swapping the values

```
length(disease_prediction_testing[disease_prediction_testing$Low.Blood.Pressure>disease_predicti
on_testing$High.Blood.Pressure,1])
```

```
## [1] 84
```

Swapping the values wherever Low BP > High BP which is not permissible

```
low_bp_values <- disease_prediction_testing$Low.Blood.Pressure[disease_prediction_testing$Low.Bl
ood.Pressure>disease_prediction_testing$High.Blood.Pressure]

high_bp_values <- disease_prediction_testing$High.Blood.Pressure[disease_prediction_testing$Low.
Blood.Pressure>disease_prediction_testing$High.Blood.Pressure]

disease_prediction_testing$Low.Blood.Pressure[disease_prediction_testing$Low.Blood.Pressure>dise
ase_prediction_testing$High.Blood.Pressure] <- high_bp_values

disease_prediction_testing$High.Blood.Pressure[disease_prediction_testing$Low.Blood.Pressure>dis
ease_prediction_testing$High.Blood.Pressure] <- low_bp_values
```

Verifying that there are no instances with low bp values > high bp values

```
length(disease_prediction_testing[disease_prediction_testing$Low.Blood.Pressure>disease_predicti
on_testing$High.Blood.Pressure,1])
```

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

Any value that is less than 28.9 are replaced with 28.9

```
disease_prediction_testing$Weight[disease_prediction_testing$Weight<28.9] <- 28.9
```

Verifying that the Min & Max values of Weight are in the correct practically permissible range and the outliers have been eliminated

```
summary(disease_prediction_testing$Weight)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   28.90   65.00   72.00   74.24   82.00  183.00
```

DATA PREPARATION for KNN & SVM :

For KNN and SVM, which are distance based algorithms, we require all the categorical variables to be converted to Numeric by performing one hot encoding and also all the numeric variables have to be normalized so that all the columns have values in the range of 0 to 1.

Creating Dummy Variables out of all categorical variables by performing one hot encoding and normalizing all numeric columns using Min Max scaler

```
disease_prediction_testing$bmi <- disease_prediction_testing$Weight/((disease_prediction_testing
$Height/100)*(disease_prediction_testing$Height/100))
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
library(fastDummies)
disease_prediction_testing_knn_svm <- disease_prediction_testing
disease_prediction_testing_knn_svm<-fastDummies::dummy_cols(disease_prediction_testing_knn_svm,s
elect_columns=c('Gender','Cholesterol','Glucose'))

disease_prediction_testing_knn_svm$Age <- normalize(disease_prediction_testing_knn_svm$Age)
disease_prediction_testing_knn_svm$Height <- normalize(disease_prediction_testing_knn_svm$Heigh
t)
disease_prediction_testing_knn_svm$Weight <- normalize(disease_prediction_testing_knn_svm$Weigh
t)
disease_prediction_testing_knn_svm$bmi <- normalize(disease_prediction_testing_knn_svm$bmi)
disease_prediction_testing_knn_svm$Low.Blood.Pressure <- normalize(disease_prediction_testing_kn
n_svm$Low.Blood.Pressure)
disease_prediction_testing_knn_svm$High.Blood.Pressure <- normalize(disease_prediction_testing_k
nn_svm$High.Blood.Pressure)
```

Getting rid of non numeric columns

```
disease_prediction_testing_knn_svm$Gender <- NULL
disease_prediction_testing_knn_svm$Cholesterol <- NULL
disease_prediction_testing_knn_svm$Glucose <- NULL
disease_prediction_testing_knn_svm$age_groups <- NULL

disease_prediction_testing_knn_svm$bmi_groups <- NULL
disease_prediction_testing_knn_svm$Gender_male <- NULL
disease_prediction_testing_knn_svm$ID <- NULL
```

1. Obtaining Predicted values for disease on TEST data using the final KNN model**

```
KNN <- predict(tuned_model_knn, newdata = disease_prediction_testing_knn_svm)
disease_prediction_testing$KNN <- KNN
```

2. Naive Bayes Classifier Model (NBC)

```
library(caret)
disease_prediction_nb_testing <- disease_prediction_testing
disease_prediction_nb_testing$age_groups <- NULL
disease_prediction_nb_testing$Height <- NULL
disease_prediction_nb_testing$Weight <- NULL
disease_prediction_nb_testing$Smoke <- as.factor(disease_prediction_nb_testing$Smoke)
disease_prediction_nb_testing$Alcohol <- as.factor(disease_prediction_nb_testing$Alcohol)
disease_prediction_nb_testing$Exercise <- as.factor(disease_prediction_nb_testing$Exercise)
```

Obtaining Predicted values for disease on TEST data using the final NBC model

```
NBC <- predict(nb_baseline_model, newdata = disease_prediction_nb_testing,  type = c("class", "r
aw"))
disease_prediction_testing$NBC <- NBC
```

3. Random Forest Model (RF)

```
disease_prediction_testing$age_groups <-cut(disease_prediction_testing$Age, breaks = c(quantile
(disease_prediction_testing$Age, probs = c(0,0.25,0.5,0.75,1))),labels = c(str_c(quantile(diseas
e_prediction_testing$Age,probs = 0),quantile(disease_prediction_testing$Age,probs = 0.25),sep =
" to "),str_c(quantile(disease_prediction_testing$Age,probs = 0.25),quantile(disease_prediction_
testing$Age,probs = 0.5),sep = " to "),str_c(quantile(disease_prediction_testing$Age,probs = 0.5
),quantile(disease_prediction_testing$Age,probs = 0.75),sep = " to "),str_c(quantile(disease_pre
diction_testing$Age,probs = 0.75),quantile(disease_prediction_testing$Age,probs = 1),sep = " to
 ")), right = FALSE, include.lowest=TRUE)
disease_prediction_testing$age_groups<-as.factor(disease_prediction_testing$age_groups)
```

Obtaining Predicted values for disease on TEST data using the Random Forest model

```
RF <- predict(tuned_model_rf, newdata = disease_prediction_testing)
disease_prediction_testing$RF <- RF
```

4. Gradient Boosting Algorithm

Obtaining Predicted values for disease on TEST data using the Gradient Boosting model

```
GBM <- predict(gbm_tuned, newdata = disease_prediction_testing)
disease_prediction_testing$GBM <- GBM
```

5. LINEAR SUPPORT VECTOR MACHINE

Obtaining Predicted values for disease on TEST data using the Linear SVM model

```
SVM_Linear <- predict(svml_tuned, newdata = disease_prediction_testing_knn_svm)
disease_prediction_testing$SVM_Linear <- SVM_Linear
```

6. RBF SUPPORT VECTOR MACHINE

Obtaining Predicted values for disease on TEST data using the RBF SVM model

```
SVM_RBF <- predict(svmnl_tuned, newdata = disease_prediction_testing_knn_svm)
disease_prediction_testing$SVM_RBF <- SVM_RBF
```

Writing the final predictions to a CSV

```
final_predictions <- disease_prediction_testing[,c('ID','NBC','KNN','SVM_Linear','SVM_RBF','RF',
'GBM')]
write.csv(final_predictions,"HW_03_Kumar_Bhavish_Predictions.csv")
```

D. CONCLUSION:

From the above model building, tuning and evaluation we can understand the importance of tuning hyperparameters through a grid search to improve the model performance. We also the learned the importance of K fold cross validation and hold one out techniques to ensure that the model performs well not only on the training data but also on the validation data which it has not seen before. From the above 6 algorithms we can observe that the ensemble methods like Gradient Boosting and Random Forest are one of the best performing models