IST 707 Data Analytics HOMEWORK 4: Disease Prediction for Patients using LR, DT, ANN0, ANN1 and ANN2

Submitted by BHAVISH KUMAR on April 26th 2020

A. Executive Summary:

The purpose of this assignment is to use the Decision Tree, Logistic Regression classification models as well as the Artificial Neural Networks (Deep Learning) models that we have 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. 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 Machine Learning algorithms that were used to achieve this outcome were Decision Tree and Logistic Regression as well three Deep Learning algorithms with 0 hidden layers (ANN0), 1 hidden layer (ANN1) and 2 hidden layers (ANN2) were used. 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 Learning Algorithms have been built: i. Logistic Regression: This is a regression based algorithm that converts the regression output to a probability ranging between 0 & 1 by using the sigmoid activation function. The Elastic Net Regularization parameters such as alpha and lambda can be used to control for overfitting.
ii. Decision Tree: This uses a tree-like model of decisions and their possible consequences. The tree contains a root node, internal nodes and leaf nodes along with splitting attributes that form the branches of the tree. The hyperparameters such as Max Depth, Minbucket, Minsplit etc. can be used to prune the tree in order to reduce variance without compromising on bias.
iii. Artificial Neural Network 0 layers (ANN0): This is a deep learning neural network model consisting of 0 hidden layers. This network consists of directly the output layer followed by the activation function to transform the output. The neural network uses Gradient Descent to identify the weights for the variables which will minimize the loss function.
iv. Artificial Neural Network 1 layer (ANN1): This is a feed forward Neural network containing 1 hidden layer followed by a relu activation function after the hidden layer and a an output layer followed by sigmoid activation function. The output layer consists of only 1 node and the sigmoid function in this case since this is a binary classification problem.
v. Artificial Neural Network 2 layers (ANN2): This is a feed forward Neural network containing 2 hidden layers followed by a relu activation function after each of the hidden layer and a an output layer followed by sigmoid activation function. The output layer consists of only 1 node and the sigmoid function in this case since this is a binary classification problem. The number of nodes and the layer dropout rate are parameters that can be tuned in order to reduce the variance as well as the bias. The batch size and

number of epochs are also two additional parameters which can be tuned to produce best results.
The following techniques were used for model performance evaluation:
K fold Cross Validation was done while training Logistic Regression & Decision Tree 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, ROC curve and Area Under ROC curve metrics.

C. Body of the Report:

Reading the Training data csv and storing it into a dataframe

```
setwd("D:/SYR ADS/Sem 2/IST_707_Data_Analytics/HW4")
getwd
```

```
## function ()
## .Internal(getwd())
## <bytecode: 0x000002478470eac8>
## <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  12 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 ...
```

```
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
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.0
##  1st Qu.:0.00000   1st Qu.:1.0000   1st Qu.:0.0
##  Median :0.00000   Median :1.0000   Median :0.0
##  Mean   :0.05424   Mean   :0.8032   Mean   :0.5
##  3rd Qu.:0.00000   3rd Qu.:1.0000   3rd Qu.:1.0
##  Max.   :1.00000   Max.   :1.0000   Max.   :1.0
```

SECTION 1: DATA PREPARATION & Exploratory 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)
```

```
## Warning: package 'tidyverse' was built under R version 3.6.2
```

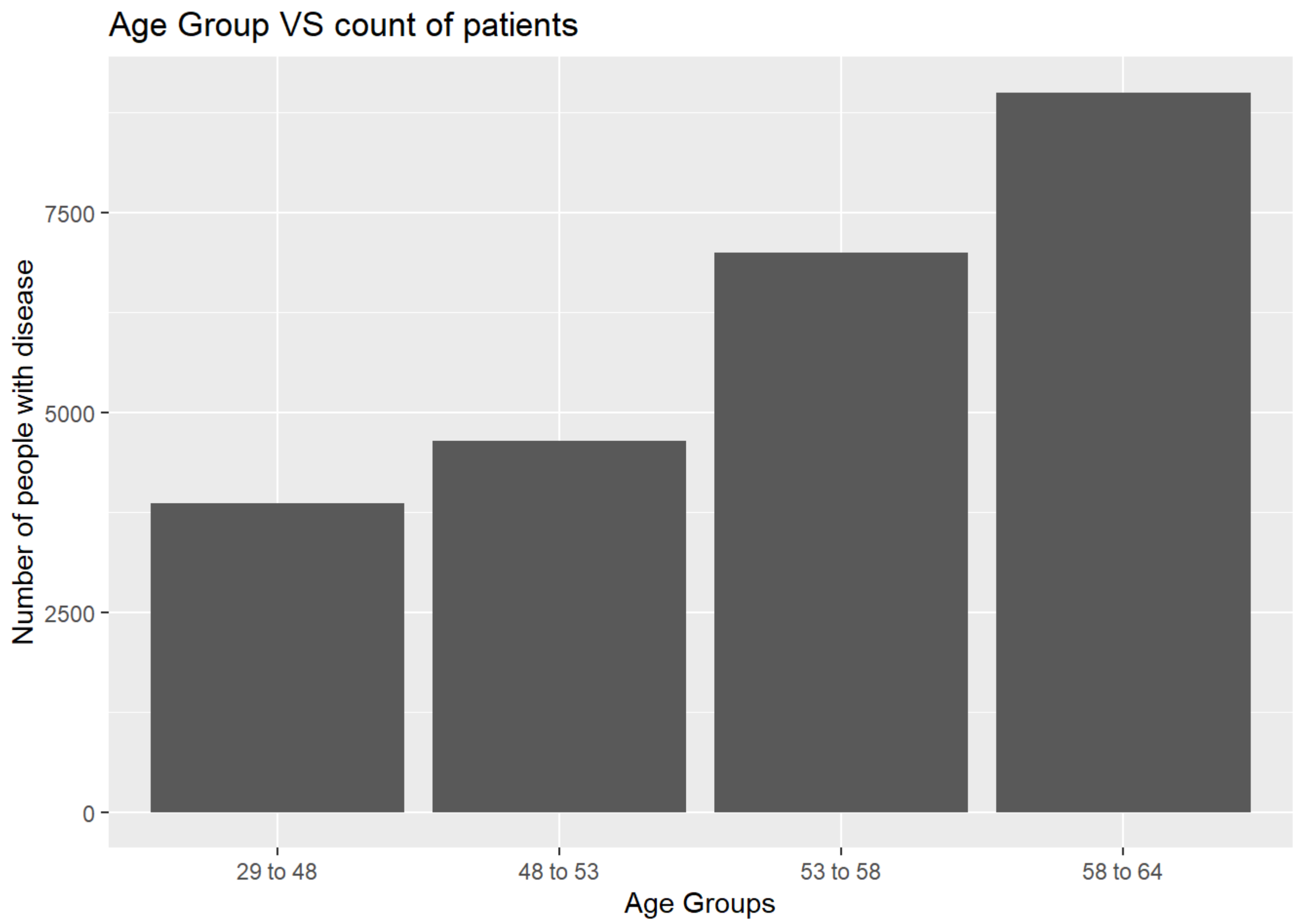
```
## -- Attaching packages ---------------------------------------- tidyverse 1.3.0 --
```

```
## <U+2713> ggplot2 3.2.1     <U+2713> purrr   0.3.3
## <U+2713> tibble  2.1.3     <U+2713> dplyr   0.8.4
## <U+2713> tidyr   1.0.0     <U+2713> forcats 0.4.0
## <U+2713> readr   1.3.1
```

```
## Warning: package 'ggplot2' was built under R version 3.6.2
```

```
## Warning: package 'dplyr' was built under R version 3.6.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
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
```



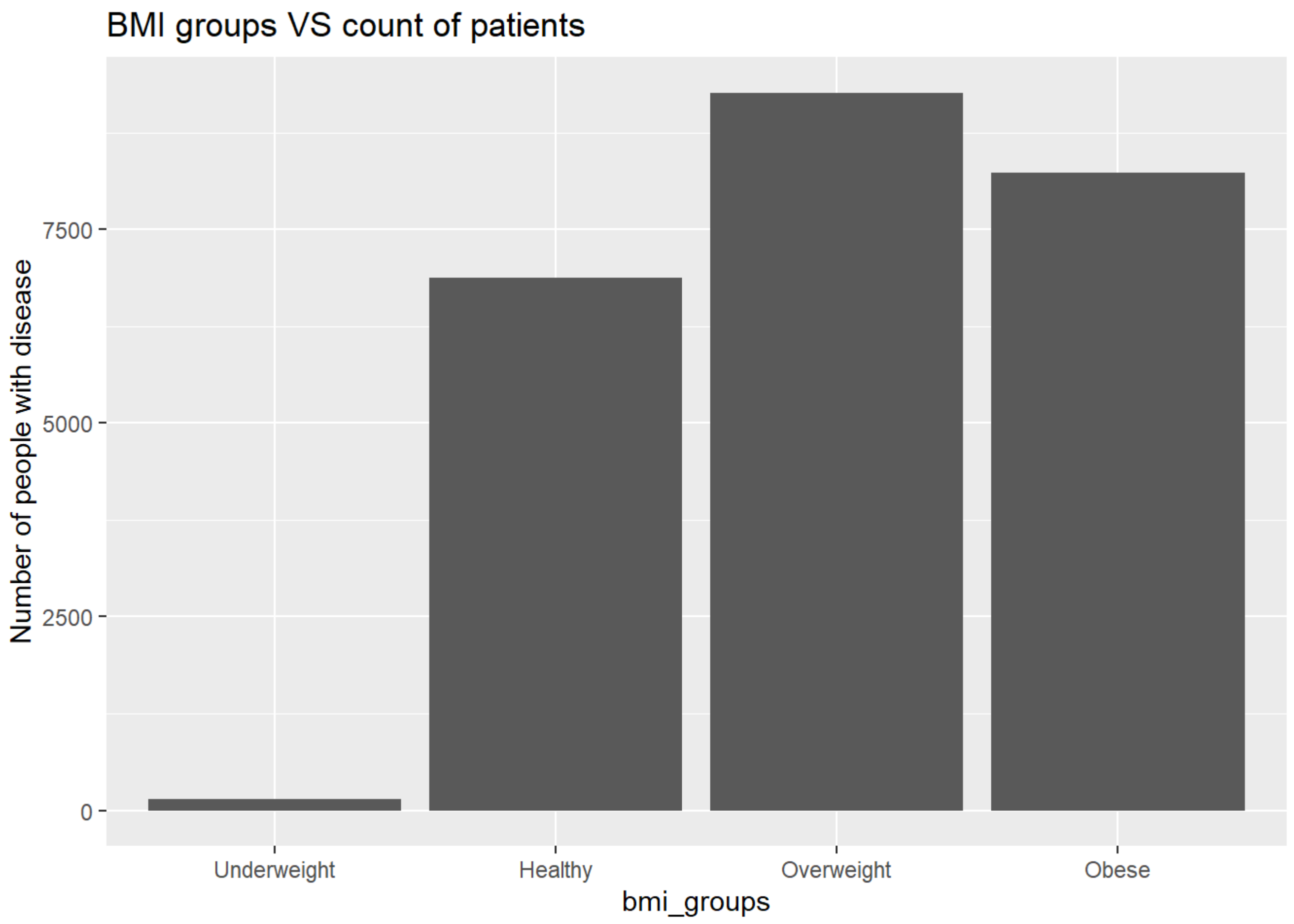
2.2. Bi Variate Analysis between Height+Weight (BMI) and Disease column

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

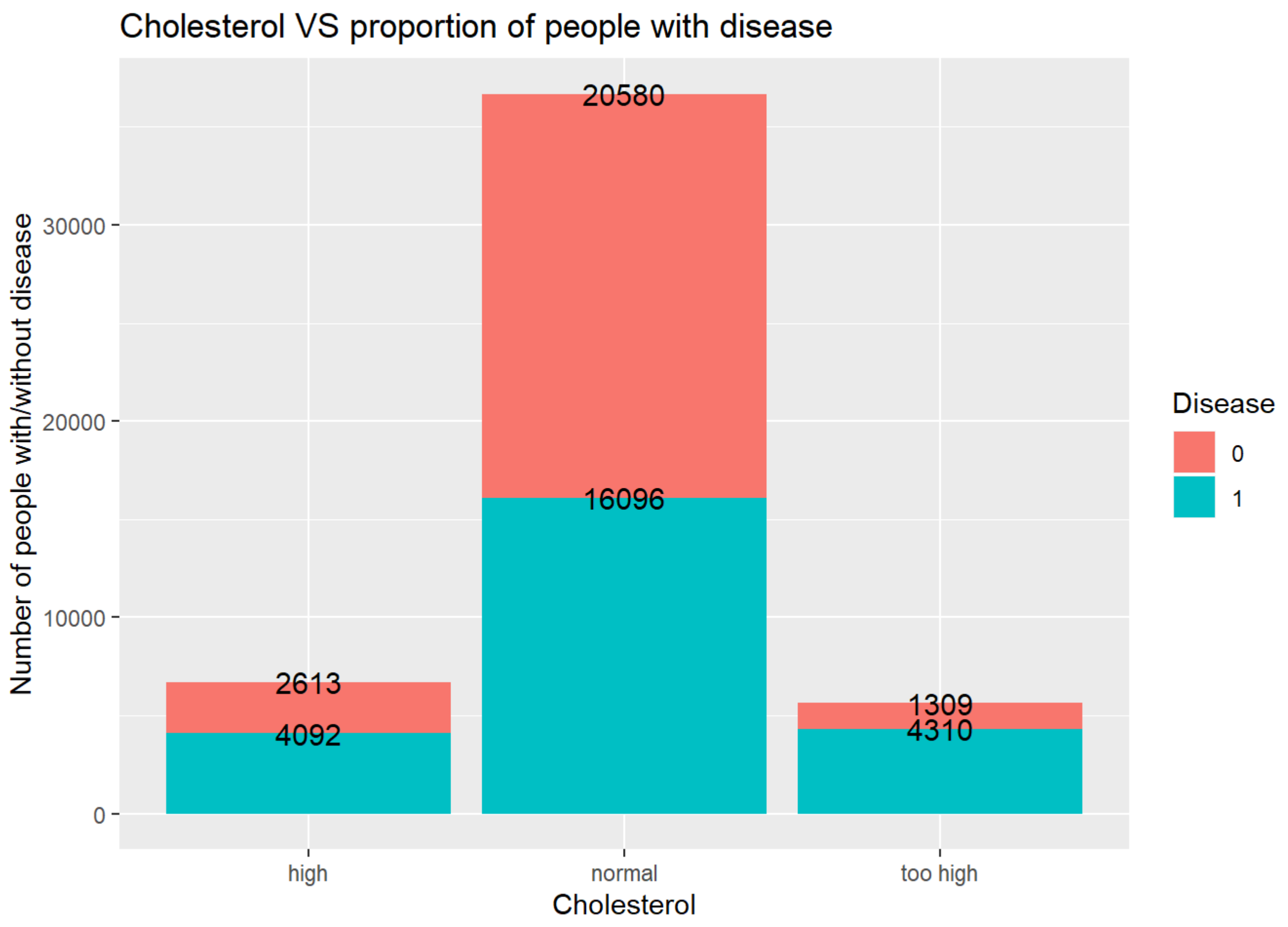
```
disease_count_by_bmi_groups <- disease_prediction_training %>%
      group_by(bmi_groups)%>%
      summarise(sum(Disease))
colnames(disease_count_by_bmi_groups) <- c('bmi_groups','NO_ppl_with_disease')

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

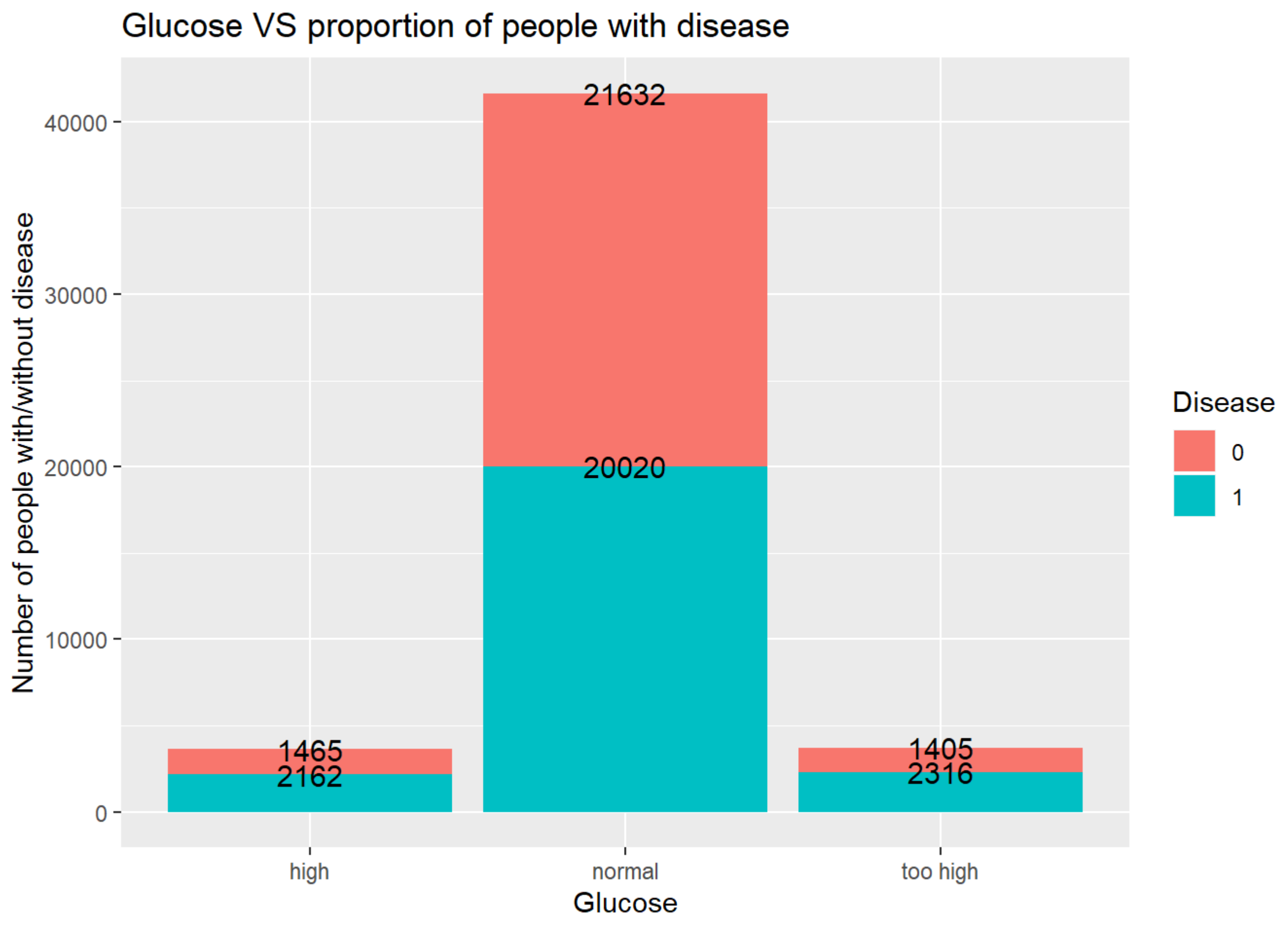
```
disease_count_by_cholestrol <- disease_prediction_training %>%
      group_by(Cholesterol,as.factor(Disease))%>%
      summarise(n())
colnames(disease_count_by_cholestrol) <- c('Cholesterol','Disease','NO_ppl')

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
```



2.11. Uni Variate Analysis - Histogram of Numeric Variables to view the distribution of the numeric variables

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

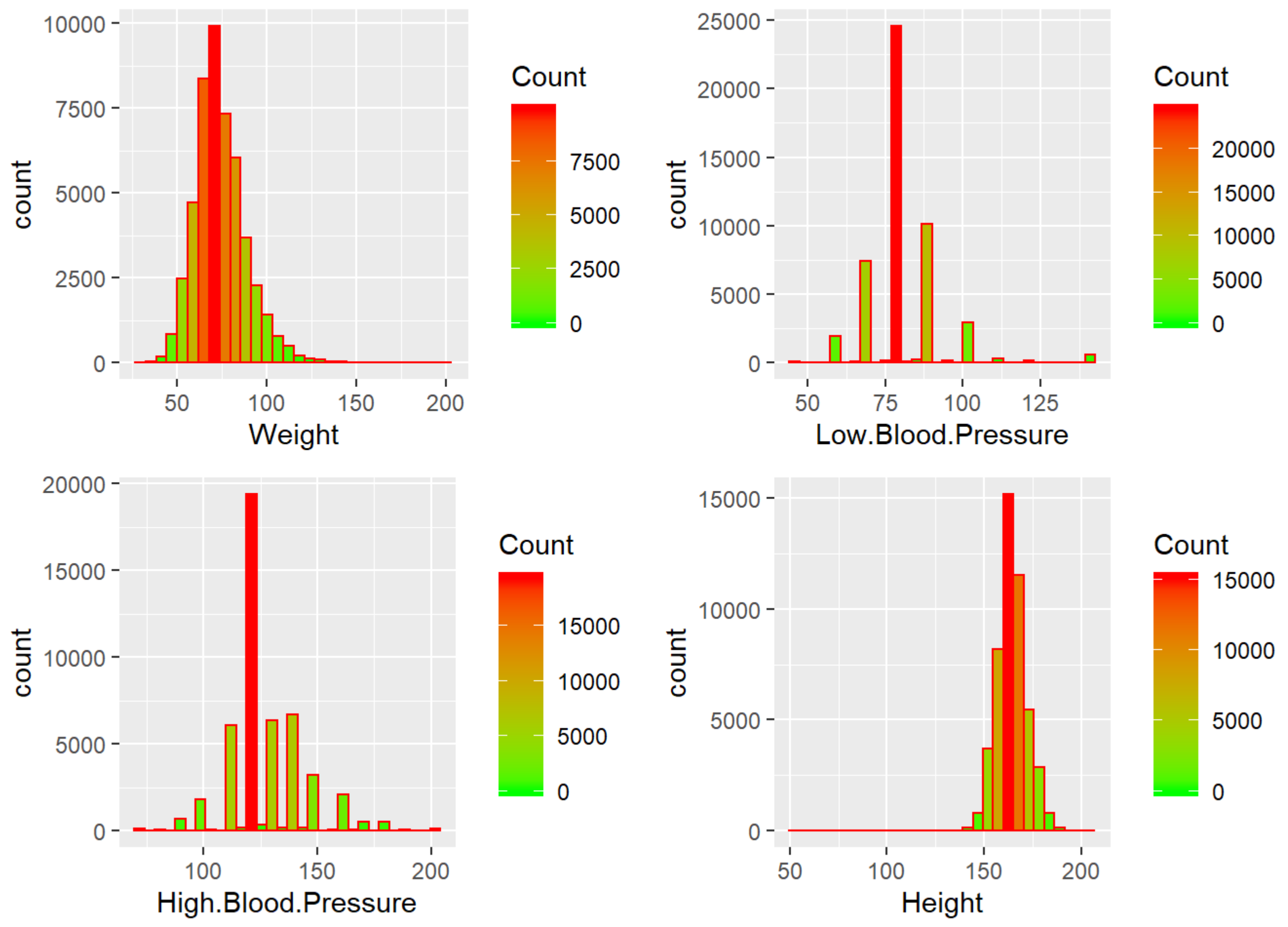
```
library(gridExtra)
```

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

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
hist_plot <- function(data_in, i)
  {
  data <- data.frame(x=data_in[[i]])
  p <- ggplot(data=data, aes(data[,1])) +
    geom_histogram(bins = 30, col="red", aes(fill=..count..)) +
    xlab(colnames(data_in)[i]) +
    scale_fill_gradient("Count", low="green", high="red")
  return (p)
}
mygrid2 <- list()
for (i in 1:length(disease_prediction_training[,c('Weight','Low.Blood.Pressure','High.Blood.Pres
sure','Height')])){
  myplot2 <- hist_plot(disease_prediction_training[,c('Weight','Low.Blood.Pressure','High.Blood.
Pressure','Height')], i)
  mygrid2 <- c(mygrid2, list(myplot2))
}
do.call("grid.arrange", c(mygrid2, ncol=2))
```



3. DATA PREPARATION for Logistic & ANN :

For Logistic and ANN 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 other numeric columns using Min Max scaler

```
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
library(fastDummies)
```

```
## Warning: package 'fastDummies' was built under R version 3.6.2
```

```
disease_prediction_training_df <- disease_prediction_training
disease_prediction_training_df<-fastDummies::dummy_cols(disease_prediction_training_df,select_co
lumns=c('Gender','Cholesterol','Glucose'))

disease_prediction_training_df$Age <- normalize(disease_prediction_training_df$Age)
disease_prediction_training_df$Height <- normalize(disease_prediction_training_df$Height)
disease_prediction_training_df$Weight <- normalize(disease_prediction_training_df$Weight)
disease_prediction_training_df$bmi <- normalize(disease_prediction_training_df$bmi)
disease_prediction_training_df$Low.Blood.Pressure <- normalize(disease_prediction_training_df$Lo
w.Blood.Pressure)
disease_prediction_training_df$High.Blood.Pressure <- normalize(disease_prediction_training_df$H
igh.Blood.Pressure)
```

Getting rid of non numeric columns

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

disease_prediction_training_df$bmi_groups <- NULL
disease_prediction_training_df$Gender_male <- NULL
disease_prediction_training_df$Disease <- as.factor(disease_prediction_training_df$Disease)

i <- sapply(disease_prediction_training_df, is.integer)
disease_prediction_training_df[,i] <- lapply(disease_prediction_training_df[i], as.factor)
```

SECTION 2: BUILD, TUNE & EVALUATE various ML ALGORITHMS

2.1 Logistic Regression

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)
```

```
## Warning: package 'caret' was built under R version 3.6.2
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
set.seed(73)
train_index <- createDataPartition(disease_prediction_training_df$Disease, p = 0.7, list = FALSE
)

disease_prediction_training_df_tr_split <- disease_prediction_training_df[train_index, ]


disease_prediction_training_df_test_split <- disease_prediction_training_df[-train_index, ]

disease_prediction_training_df_tr_split$Height <- NULL
disease_prediction_training_df_tr_split$Weight <- NULL
disease_prediction_training_df_test_split$Height <- NULL
disease_prediction_training_df_test_split$Weight <- NULL
```

BUILDING BASELINE MODEL for Logistic Regression

```
library(glmnet)
```

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

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
## Loaded glmnet 3.0-2
```

```
baseline_model_logistic <- train(Disease ~ ., data = disease_prediction_training_df_tr_split, me
thod = "glmnet", family = "binomial")
```

```
predict_disease_logistic <- predict(baseline_model_logistic, newdata = disease_prediction_traini
ng_df_test_split)
```

The baseline model produced the following accuracies on the training data for different values of elastic net regularization parameters

```
baseline_model_logistic
```

```
## glmnet
##
## 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:
##
##   alpha  lambda        Accuracy   Kappa
##   0.10   0.0004282798  0.7298731  0.4596417
##   0.10   0.0042827984  0.7299367  0.4597687
##   0.10   0.0428279836  0.7289992  0.4578877
##   0.55   0.0004282798  0.7300351  0.4599655
##   0.55   0.0042827984  0.7299529  0.4597998
##   0.55   0.0428279836  0.7281104  0.4560975
##   1.00   0.0004282798  0.7300287  0.4599525
##   1.00   0.0042827984  0.7297086  0.4593087
##   1.00   0.0428279836  0.7248386  0.4495295
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.55 and lambda = 0.0004282798.
```

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

```
confusionMatrix(predict_disease_logistic,disease_prediction_training_df_test_split$Disease, posi
tive = "1")
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    0    1
##          0 5797 2444
##          1 1553 4905
##                                           
##                Accuracy : 0.7281          
##                  95% CI : (0.7208, 0.7353)
##     No Information Rate : 0.5             
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.4561          
##                                           
##  Mcnemar's Test P-Value : < 2.2e-16       
##                                           
##             Sensitivity : 0.6674          
##             Specificity : 0.7887          
##          Pos Pred Value : 0.7595          
##          Neg Pred Value : 0.7034          
##              Prevalence : 0.5000          
##          Detection Rate : 0.3337          
##    Detection Prevalence : 0.4393          
##       Balanced Accuracy : 0.7281          
##                                           
##        'Positive' Class : 1               
## 
```

Fine tuning the hyperparameter ‘alpha’ & ‘lambda’ for producing the best possible accuracy on validation data.

The Hyperparameter ‘alpha’ in Logistic algorithm denotes the balance between L1 & L2 regularization, whereas lambda is shrinkage parameter which denotes the strength of regularization
alpha & lambda are the elastic net regularization parameters of a logistic regression model and are used to control for overfitting by reducing the coefficients of the features as these parameters are added to the Loss function in order to reduce the coefficients of the features which will in turn reduce overfitting
Fine tuning the model to produce maximum accuracy and using 5 fold Cross Validation to train the model by controlling for overfitting

```
tuned_model_logistic <- train(Disease ~ ., data = disease_prediction_training_df_tr_split, metho
d = "glmnet",family = "binomial",
                    tuneLength = 15,
                    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 elastic net regularization parameters alpha and lambda

```
print(tuned_model_logistic)
```

```
## glmnet
##
## 34301 samples
##    14 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 27440, 27440, 27442, 27441, 27441, 27440, ...
## Resampling results across tuning parameters:
##
##   alpha      lambda        Accuracy   Kappa
##   0.1000000  0.0007616012  0.7303385  0.4606717
##   0.1000000  0.0013543398  0.7303385  0.4606717
##   0.1000000  0.0024083945  0.7303385  0.4606717
##   0.1000000  0.0042827984  0.7303676  0.4607300
##   0.1000000  0.0076160121  0.7300761  0.4601469
##   0.1000000  0.0135433976  0.7300371  0.4600690
##   0.1000000  0.0240839450  0.7299108  0.4598163
##   0.1000000  0.0428279836  0.7299302  0.4598551
##   0.1000000  0.0761601214  0.7296484  0.4592913
##   0.1000000  0.1354339757  0.7279575  0.4559089
##   0.1000000  0.2408394504  0.7271607  0.4543148
##   0.1642857  0.0007616012  0.7302413  0.4604773
##   0.1642857  0.0013543398  0.7302413  0.4604773
##   0.1642857  0.0024083945  0.7302413  0.4604773
##   0.1642857  0.0042827984  0.7303579  0.4607105
##   0.1642857  0.0076160121  0.7302801  0.4605550
##   0.1642857  0.0135433976  0.7298622  0.4597192
##   0.1642857  0.0240839450  0.7298331  0.4596608
##   0.1642857  0.0428279836  0.7302607  0.4605159
##   0.1642857  0.0761601214  0.7292306  0.4584553
##   0.1642857  0.1354339757  0.7282880  0.4565696
##   0.1642857  0.2408394504  0.7276952  0.4553833
##   0.2285714  0.0007616012  0.7302413  0.4604773
##   0.2285714  0.0013543398  0.7302413  0.4604773
##   0.2285714  0.0024083945  0.7303384  0.4606717
##   0.2285714  0.0042827984  0.7302899  0.4605745
##   0.2285714  0.0076160121  0.7302024  0.4603996
##   0.2285714  0.0135433976  0.7300080  0.4600107
##   0.2285714  0.0240839450  0.7298720  0.4597385
##   0.2285714  0.0428279836  0.7299691  0.4599326
##   0.2285714  0.0761601214  0.7286086  0.4572111
##   0.2285714  0.1354339757  0.7282393  0.4564721
##   0.2285714  0.2408394504  0.7279867  0.4559658
##   0.2928571  0.0007616012  0.7302899  0.4605745
##   0.2928571  0.0013543398  0.7302899  0.4605745
##   0.2928571  0.0024083945  0.7302899  0.4605745
##   0.2928571  0.0042827984  0.7302704  0.4605356
##   0.2928571  0.0076160121  0.7303093  0.4606133
##   0.2928571  0.0135433976  0.7300858  0.4601662
##   0.2928571  0.0240839450  0.7299886  0.4599717
##   0.2928571  0.0428279836  0.7299011  0.4597964
##   0.2928571  0.0761601214  0.7284920  0.4569777
##   0.2928571  0.1354339757  0.7281422  0.4562773
##   0.2928571  0.2408394504  0.7256057  0.4512029
##   0.3571429  0.0007616012  0.7302024  0.4603995
##   0.3571429  0.0013543398  0.7302024  0.4603995
##   0.3571429  0.0024083945  0.7301052  0.4602052
##   0.3571429  0.0042827984  0.7302316  0.4604579
##   0.3571429  0.0076160121  0.7303482  0.4606911
##   0.3571429  0.0135433976  0.7305037  0.4610020
##   0.3571429  0.0240839450  0.7301052  0.4602047
##   0.3571429  0.0428279836  0.7295707  0.4591354
##   0.3571429  0.0761601214  0.7282491  0.4564916
##   0.3571429  0.1354339757  0.7276563  0.4553051
##   0.3571429  0.2408394504  0.7189296  0.4378496
##   0.4214286  0.0007616012  0.7301830  0.4603607
##   0.4214286  0.0013543398  0.7301830  0.4603607
##   0.4214286  0.0024083945  0.7301149  0.4602246
##   0.4214286  0.0042827984  0.7303676  0.4607300
##   0.4214286  0.0076160121  0.7303773  0.4607493
##   0.4214286  0.0135433976  0.7304259  0.4608464
##   0.4214286  0.0240839450  0.7301343  0.4602630
##   0.4214286  0.0428279836  0.7289779  0.4579497
##   0.4214286  0.0761601214  0.7278410  0.4556751
##   0.4214286  0.1354339757  0.7272384  0.4544690
##   0.4214286  0.2408394504  0.7179383  0.4358680
##   0.4857143  0.0007616012  0.7301927  0.4603801
##   0.4857143  0.0013543398  0.7301927  0.4603801
##   0.4857143  0.0024083945  0.7302510  0.4604967
##   0.4857143  0.0042827984  0.7303676  0.4607300
##   0.4857143  0.0076160121  0.7303773  0.4607493
##   0.4857143  0.0135433976  0.7305328  0.4610601
##   0.4857143  0.0240839450  0.7303481  0.4606906
##   0.4857143  0.0428279836  0.7284532  0.4569001
##   0.4857143  0.0761601214  0.7276077  0.4552084
##   0.4857143  0.1354339757  0.7240801  0.4481515
##   0.4857143  0.2408394504  0.7178023  0.4355961
##   0.5500000  0.0007616012  0.7301732  0.4603412
##   0.5500000  0.0013543398  0.7301732  0.4603412
##   0.5500000  0.0024083945  0.7302024  0.4603995
##   0.5500000  0.0042827984  0.7304065  0.4608077
##   0.5500000  0.0076160121  0.7303967  0.4607881
##   0.5500000  0.0135433976  0.7303287  0.4606519
##   0.5500000  0.0240839450  0.7301149  0.4602239
##   0.5500000  0.0428279836  0.7278313  0.4556560
##   0.5500000  0.0761601214  0.7275494  0.4550915
##   0.5500000  0.1354339757  0.7192503  0.4384912
##   0.5500000  0.2408394504  0.7177342  0.4354600
##   0.6142857  0.0007616012  0.7301732  0.4603412
##   0.6142857  0.0013543398  0.7301635  0.4603218
##   0.6142857  0.0024083945  0.7303093  0.4606134
##   0.6142857  0.0042827984  0.7304551  0.4609048
##   0.6142857  0.0076160121  0.7304648  0.4609241
##   0.6142857  0.0135433976  0.7301149  0.4602242
##   0.6142857  0.0240839450  0.7300080  0.4600101
##   0.6142857  0.0428279836  0.7273939  0.4547813
##   0.6142857  0.0761601214  0.7270830  0.4541584
##   0.6142857  0.1354339757  0.7189684  0.4379281
##   0.6142857  0.2408394504  0.7177537  0.4354989
##   0.6785714  0.0007616012  0.7301732  0.4603412
##   0.6785714  0.0013543398  0.7301927  0.4603801
##   0.6785714  0.0024083945  0.7303482  0.4606911
##   0.6785714  0.0042827984  0.7302704  0.4605356
##   0.6785714  0.0076160121  0.7304842  0.4609630
##   0.6785714  0.0135433976  0.7300177  0.4600298
##   0.6785714  0.0240839450  0.7290848  0.4581635
##   0.6785714  0.0428279836  0.7273356  0.4546644
##   0.6785714  0.0761601214  0.7260820  0.4521562
##   0.6785714  0.1354339757  0.7179480  0.4358876
##   0.6785714  0.2408394504  0.7177537  0.4354989
##   0.7428571  0.0007616012  0.7301441  0.4602829
##   0.7428571  0.0013543398  0.7301635  0.4603218
##   0.7428571  0.0024083945  0.7304162  0.4608272
##   0.7428571  0.0042827984  0.7301732  0.4603411
##   0.7428571  0.0076160121  0.7304162  0.4608270
##   0.7428571  0.0135433976  0.7300080  0.4600103
##   0.7428571  0.0240839450  0.7282103  0.4564143
##   0.7428571  0.0428279836  0.7273939  0.4547808
##   0.7428571  0.0761601214  0.7243133  0.4486183
##   0.7428571  0.1354339757  0.7177440  0.4354795
##   0.7428571  0.2408394504  0.7177537  0.4354989
##   0.8071429  0.0007616012  0.7301538  0.4603023
##   0.8071429  0.0013543398  0.7301247  0.4602440
##   0.8071429  0.0024083945  0.7303871  0.4607688
##   0.8071429  0.0042827984  0.7301733  0.4603412
##   0.8071429  0.0076160121  0.7302996  0.4605937
##   0.8071429  0.0135433976  0.7299691  0.4599325
##   0.8071429  0.0240839450  0.7278216  0.4556368
##   0.8071429  0.0428279836  0.7273065  0.4546057
##   0.8071429  0.0761601214  0.7210773  0.4421456
##   0.8071429  0.1354339757  0.7177342  0.4354600
##   0.8071429  0.2408394504  0.7178217  0.4356349
##   0.8714286  0.0007616012  0.7301538  0.4603024
##   0.8714286  0.0013543398  0.7302218  0.4604384
##   0.8714286  0.0024083945  0.7304648  0.4609243
##   0.8714286  0.0042827984  0.7301830  0.4603606
##   0.8714286  0.0076160121  0.7301441  0.4602826
##   0.8714286  0.0135433976  0.7301440  0.4602823
##   0.8714286  0.0240839450  0.7275397  0.4550730
##   0.8714286  0.0428279836  0.7267914  0.4535756
##   0.8714286  0.0761601214  0.7183854  0.4367614
##   0.8714286  0.1354339757  0.7177342  0.4354600
##   0.8714286  0.2408394504  0.7179092  0.4358098
##   0.9357143  0.0007616012  0.7301635  0.4603218
##   0.9357143  0.0013543398  0.7302607  0.4605162
##   0.9357143  0.0024083945  0.7303676  0.4607299
##   0.9357143  0.0042827984  0.7302315  0.4604577
##   0.9357143  0.0076160121  0.7301149  0.4602243
##   0.9357143  0.0135433976  0.7299885  0.4599712
##   0.9357143  0.0240839450  0.7271413  0.4542759
##   0.9357143  0.0428279836  0.7261694  0.4523314
##   0.9357143  0.0761601214  0.7185311  0.4370532
##   0.9357143  0.1354339757  0.7177342  0.4354600
##   0.9357143  0.2408394504  0.5000437  0.0000000
##   1.0000000  0.0007616012  0.7301635  0.4603218
##   1.0000000  0.0013543398  0.7302704  0.4605356
##   1.0000000  0.0024083945  0.7304259  0.4608465
##   1.0000000  0.0042827984  0.7301149  0.4602245
##   1.0000000  0.0076160121  0.7299691  0.4599327
##   1.0000000  0.0135433976  0.7294929  0.4589799
##   1.0000000  0.0240839450  0.7273550  0.4547034
##   1.0000000  0.0428279836  0.7255572  0.4511068
##   1.0000000  0.0761601214  0.7183368  0.4366649
##   1.0000000  0.1354339757  0.7177342  0.4354600
##   1.0000000  0.2408394504  0.5000437  0.0000000
## 
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.4857143 and lambda = 0.0135434.
```

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

```
predict_disease_tuned_logistic <- predict(tuned_model_logistic, newdata = disease_prediction_tra
ining_df_test_split)
```

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 alpha and lambda elastic net regularization hyperparameters, the risk of overfitting can be avoided
Since the training and validation metric scores are in line, we can be assured that the model is producing low bias and low variance estimate

```
disease_prediction_training_df_test_split$predicted_disease_logistic <- predict_disease_tuned_lo
gistic
conf_matrix <- data.frame(table(disease_prediction_training_df_test_split$Disease,disease_predic
tion_training_df_test_split$predicted_disease_logistic))
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 Logistic Algorithm in classifying Disease is :",Accuracy_DT)
```

```
## [1] "Accuracy of best Logistic Algorithm in classifying Disease is : 0.727396421525274"
```

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

```
## [1] "Precision of Logistic Algorithm in classifying Disease is : 0.76036148332814"
```

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

```
## [1] "Recall of Logistic Algorithm in classifying Disease is : 0.664035923254865"
```

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

```
## [1] "F1 Score of Logistic Algorithm in classifying Disease is : 0.708941672114477"
```

ROC & AUROC for the best Logistic Regression Model

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

```
## Warning: package 'pROC' was built under R version 3.6.2
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

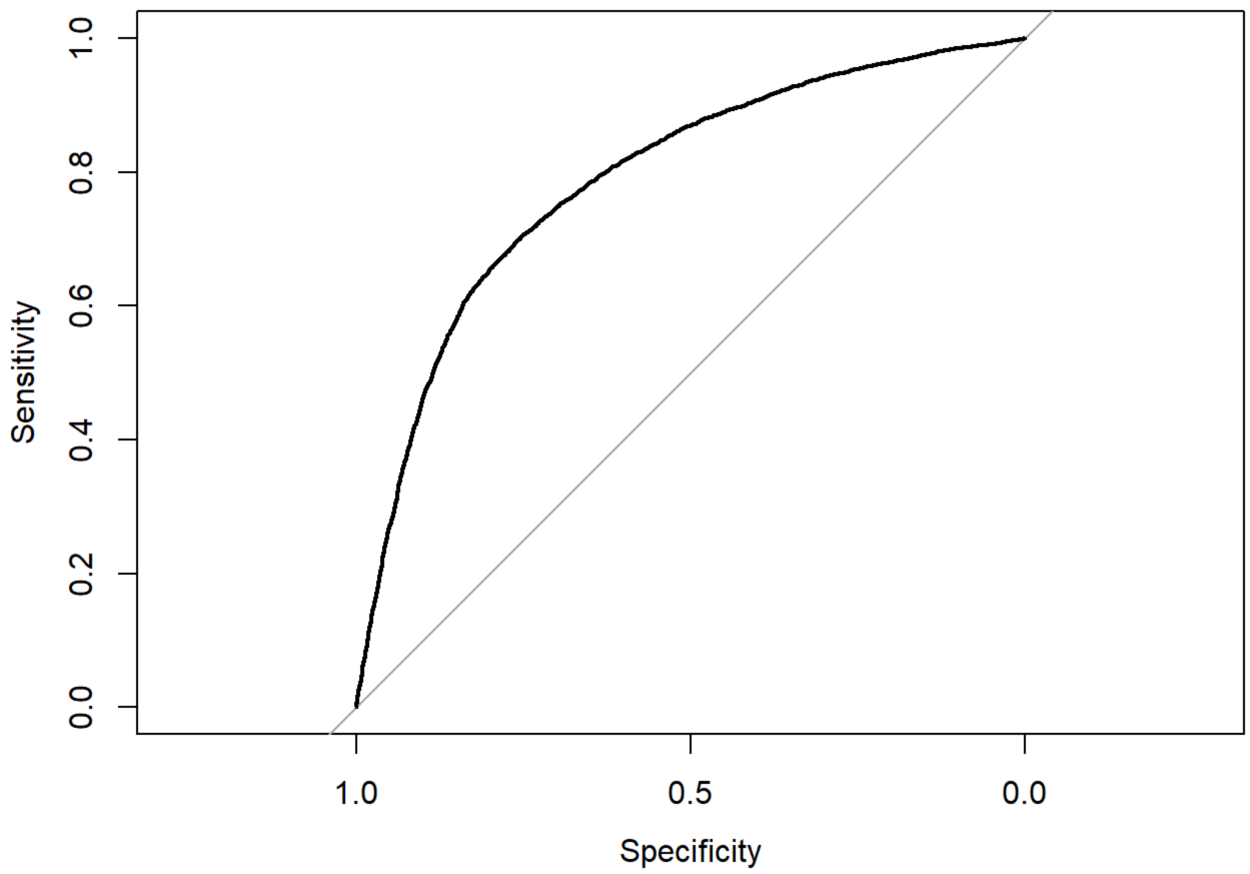
Generated ROC curve and calculated Area Under Curve metric for the identified best performing Logistic Regression model

```
logistic_pred_disease <- predict(tuned_model_logistic, newdata = disease_prediction_training_df_
test_split, na.action = na.omit, type = "prob")
roc_curve <- roc(disease_prediction_training_df_test_split$Disease,logistic_pred_disease$`1`)
```

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

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

```
plot(roc_curve)
```



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

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

```
## [1] "Area Under the ROC curve is : 0.789499122005585"
```

END OF BUILDING LOGISTIC Model (ML algorithm number 1)

2.2 Artificial Neural Network - 0 layers (ANN0)

Data Splitting

```
library(rsample)
```

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

```
train_test_split <- initial_split(disease_prediction_training_df, prop = 0.8)

disease_prediction_train_df_spit <- training(train_test_split)
disease_prediction_test_df_spit <- testing(train_test_split)

i <- sapply(disease_prediction_train_df_spit, is.factor)
disease_prediction_train_df_spit[,i] <- lapply(disease_prediction_train_df_spit[i], as.characte
r)

j <- sapply(disease_prediction_train_df_spit, is.character)
disease_prediction_train_df_spit[,j] <- lapply(disease_prediction_train_df_spit[j], as.integer)

i <- sapply(disease_prediction_test_df_spit, is.factor)
disease_prediction_test_df_spit[,i] <- lapply(disease_prediction_test_df_spit[i], as.character)

j <- sapply(disease_prediction_test_df_spit, is.character)
disease_prediction_test_df_spit[,j] <- lapply(disease_prediction_test_df_spit[j], as.integer)
```

Data PreProcessing, creating rec_obj

step_center() to mean-center the data & step_scale() to scale the data
The last step is to prepare the recipe with the prep() function

```
#library(recipes)
disease_rec_obj <- recipes::recipe(Disease~.,data = disease_prediction_train_df_spit)%>%

  recipes::step_center(Age,Height,Weight,High.Blood.Pressure,Low.Blood.Pressure,bmi)%>%
  recipes::step_scale(Age,Height,Weight,High.Blood.Pressure,Low.Blood.Pressure,bmi)%>%
  recipes::prep(data = disease_prediction_train_df_spit)
```

Data PreProcessing: Creating Train and Test datasets and Train Test target variable Vectors

Applying the “recipe” to train and test data sets with the bake() function which processes the data following our recipe steps

```
x_train_disease_pred_tbl <- recipes::bake(disease_rec_obj, new_data = disease_prediction_train_d
f_spit)%>% select(-Disease)
x_test_disease_pred_tbl <- recipes::bake(disease_rec_obj, new_data = disease_prediction_test_df_
spit)%>%select(-Disease)
y_train_vec <- ifelse(pull(disease_prediction_train_df_spit, Disease) == "1",1,0)
y_test_vec <- ifelse(pull(disease_prediction_test_df_spit, Disease) == "1",1,0)
str(x_train_disease_pred_tbl)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    39201 obs. of  16 variables:
##  $ Age                 : num  0.909 1.649 -1.752 -0.422 -2.048 ...
##  $ Height              : num  0.324 -1.742 0.203 0.932 -0.283 ...
##  $ Weight              : num  0.967 -0.223 0.617 2.505 -0.922 ...
##  $ High.Blood.Pressure : num  0.179 0.757 -1.553 0.179 -0.976 ...
##  $ Low.Blood.Pressure  : num  -1.217 1.558 -1.043 -0.176 -0.176 ...
##  $ 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 ...
##  $ bmi                 : num  0.653 0.653 0.418 1.574 -0.707 ...
##  $ Gender_female       : int  1 1 1 0 1 1 1 1 1 0 ...
##  $ Cholesterol_high    : int  0 0 0 0 1 0 1 1 0 0 ...
##  $ Cholesterol_normal  : int  1 1 1 1 0 1 0 0 1 1 ...
##  $ Cholesterol_too high: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Glucose_high        : int  0 0 0 0 1 0 1 1 0 0 ...
##  $ Glucose_normal      : int  1 1 1 1 0 1 0 0 1 1 ...
##  $ Glucose_too high    : int  0 0 0 0 0 0 0 0 0 0 ...
```

```
#use_condaenv("r-tensorflow")

#install_tensorflow(package_url = "https://pypi.python.org/packages/b8/d6/af3d52dd52150ec4a6ceb7
788bfeb2f62ecb6aa2d1172211c4db39b3#49a2/tensorflow-1.3.0rc0-cp27-cp27mu-manylinux1_x86_64.whl#md
5=1cf77a2360ae2e38dd3578618eacc03b")
```

Building Deep Learning Models

Building ANN with 0 hidden layers(ANN0)

The ANN0 has no hidden layers and only has one output layer.
The output layer has one node as we are solving a binary classification problem and uses the sigmoid function to produce the output between 0 and 1
Using the “binary_crossentropy” loss function since this is a binary classification problem, and the goal of the 0 hidden layer ANN0 model is to minimize the binary_crossentropy loss through Gradient Descent
Also writing a function below, which takes in the ‘optimizer’ as the input parameter and returns the 0 layer ANN architecture as output. We will comoare the performance with both ADAM & SGD optimizers

```
library(keras)
```

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

```
##
## Attaching package: 'keras'
```

```
## The following object is masked _by_ '.GlobalEnv':
##
##     normalize
```

```
library(tensorflow)
```

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

```
##
## Attaching package: 'tensorflow'
```

```
## The following object is masked from 'package:caret':
##
##     train
```

```
model_keras_ann0 <- function(opt) {

  model_keras_ann <- keras_model_sequential()
model_keras_ann %>%
  layer_dense(units = 1,
    kernel_initializer = "uniform",
    activation = "sigmoid") %>%
    compile(optimizer = opt,
    loss = "binary_crossentropy",
    metrics = c("accuracy")
  )
  return (model_keras_ann)
}
```

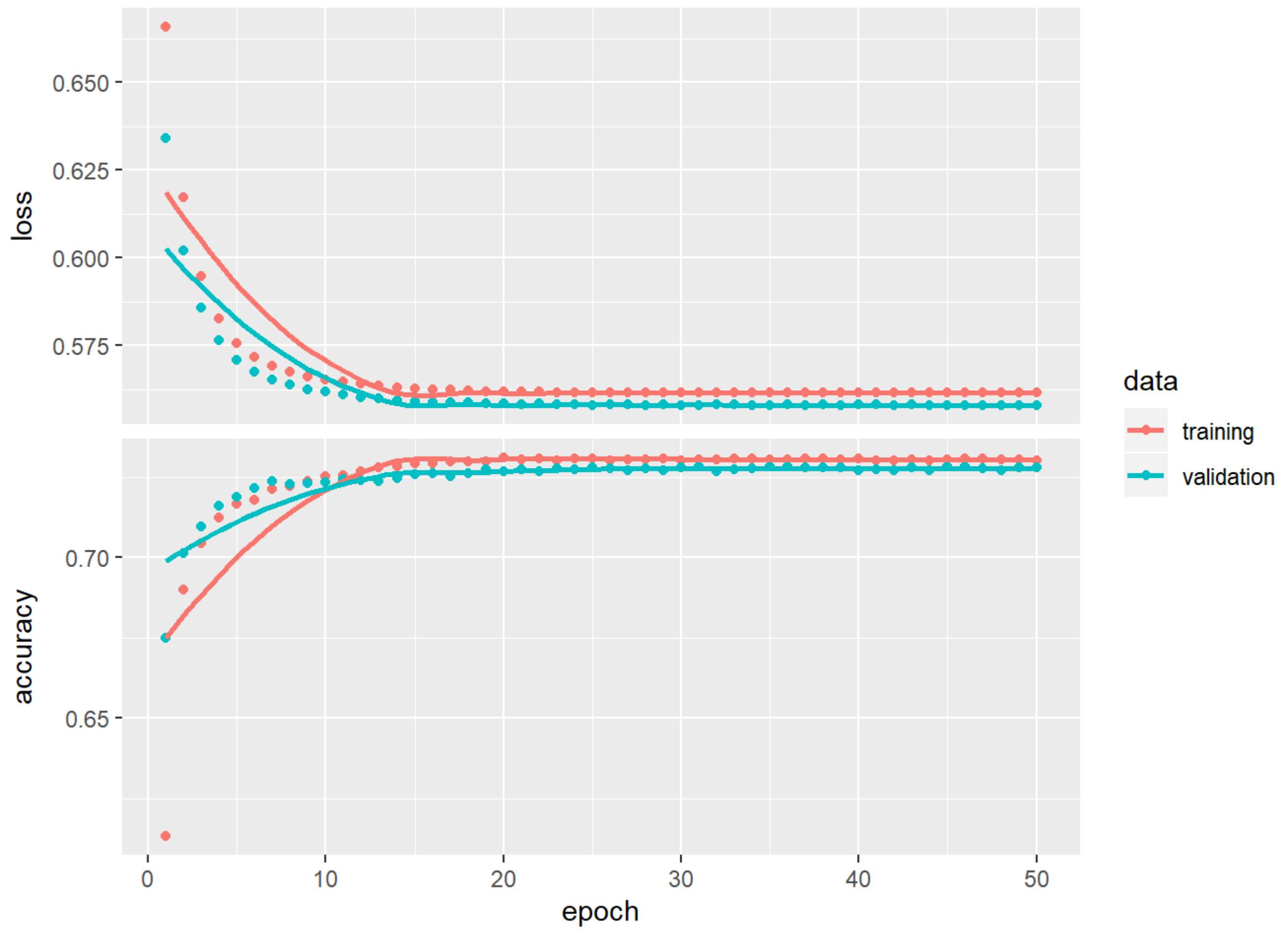
Fitting the Neural Network with 0 layers onto the train data using “ADAM” optimizer

```
model_ann0_fit <- fit(verbose=0,
  object = model_keras_ann0("adam"),
  x = as.matrix(x_train_disease_pred_tbl),
  y = y_train_vec,
  batch_size = 200,
  epochs = 50,
  validation_split = 0.30
)
```

From the below plot we can observe how training & validation accuracies as well as losses vary with every epoch.

The point where the validation accuracy & loss curve begins to flatten will be the point where we can stop training the neural network

```
plot(model_ann0_fit)
```



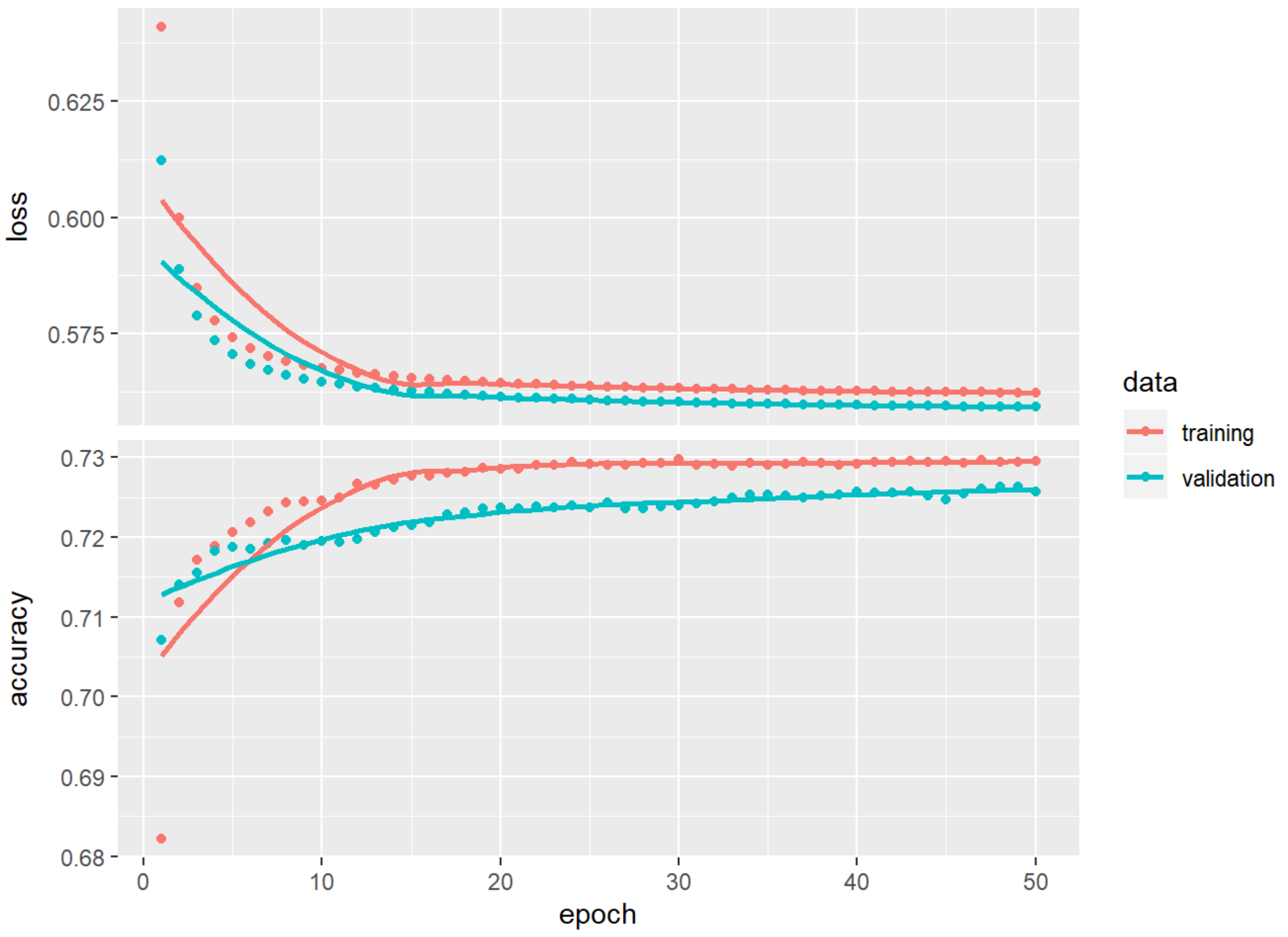
Assesing the model performance using Stochastic Gradient Descent optimization algorithm

Fitting the Neural Network with 0 layers onto the train data using SGD optimizer

```
model_ann0_sgdfit <- fit(verbose=0,
  object = model_keras_ann0("SGD"),
  x = as.matrix(x_train_disease_pred_tbl),
  y = y_train_vec,
  batch_size = 200,
  epochs = 50,
  validation_split = 0.30
)
```

From the below plot we can observe how the training and validation accuracies & losses vary after every epoch

```
plot(model_ann0_sgdfit)
```



We can observe that there is not much significant difference in performance between SGD & ADAM optimizers (The training and validation accuracies are in line for both ADAM & SGD optimizers), hence going ahead with ADAM, as it is marginally better than SGD

```
paste("MODEL PERFORMANCE with SGD optimizer",model_ann0_sgdfit)
```

```
## [1] "MODEL PERFORMANCE with SGD optimizer list(batch_size = 200, epochs = 50, steps = 138, sa
mples = 27440, verbose = 0, do_validation = TRUE, metrics = c(\"loss\", \"accuracy\", \"val_loss
\", \"val_accuracy\"))"
## [2] "MODEL PERFORMANCE with SGD optimizer list(loss = c(0.640947024172319, 0.599813293872005,
0.584758857075049, 0.577840238809586, 0.574053966529863, 0.571720123986113, 0.570171061392776,
0.569065503928126, 0.568234582814461, 0.56756776862346, 0.567015108665989, 0.566544360565375, 0.
56614710729428, 0.565805925746006, 0.565489257737429, 0.565212579101932, 0.564960707684995, 0.56
4740976508783, 0.564531860693898, 0.564346161597672, 0.564172875038389, 0.564027532804812, 0.563
87802689138, 0.563746517062535, 0.563627377586879, 0.563521958605193, \n0.563422465550309, 0.563
330646115559, 0.5632355204911, 0.563163639418238, 0.563078774828953, 0.563013483023505, 0.562948
705242952, 0.562881647063414, 0.562832106958325, 0.562773106092962, 0.562723933434, 0.5626677458
47049, 0.562639228282795, 0.56258883186054, 0.562546493013121, 0.562505133471753, 0.562465838588
362, 0.562431134392152, 0.562403600261093, 0.562366940624165, 0.562336855614151, 0.5623044187838
41, 0.562279450215682, 0.562251160817313), accuracy = c(0.682142853736877, 0.711880445480347, \n
0.717201173305511, 0.718913972377777, 0.720590353012085, 0.721829473972321, 0.723287165164948,
0.72434401512146, 0.724453330039978, 0.724562704563141, 0.724963545799255, 0.726749241352081, 0.
726567029953003, 0.727150142192841, 0.727733254432678, 0.727733254432678, 0.728097677230835, 0.7
28243410587311, 0.728680729866028, 0.728607892990112, 0.728534996509552, 0.729118049144745, 0.72
9008734226227, 0.729409635066986, 0.729190945625305, 0.729008734226227, 0.729118049144745, 0.729
336738586426, 0.729336738586426, \n0.729774057865143, 0.729081630706787, 0.729190945625305, 0.72
8972315788269, 0.729263842105865, 0.729081630706787, 0.729227423667908, 0.729482531547546, 0.729
336738586426, 0.729045212268829, 0.729154527187347, 0.729482531547546, 0.729446053504944, 0.7295
91846466064, 0.729409635066986, 0.729555368423462, 0.729336738586426, 0.729701161384583, 0.72940
9635066986, 0.729446053504944, 0.729518949985504), val_loss = c(0.612229292504835, 0.58885497099
5661, 0.578784378630482, 0.573572448114688, 0.570435131698956, \n0.568446560626066, 0.5670163551
97311, 0.565976066883704, 0.565215583196678, 0.564549375959891, 0.564047064077549, 0.56357455010
3375, 0.563246395558324, 0.562881302416968, 0.562587346342204, 0.562312871711857, 0.562011445504
185, 0.561788109484887, 0.561617523870159, 0.561437856055332, 0.561239840906095, 0.5610902323405
38, 0.560927261235651, 0.560827251458166, 0.560702667340931, 0.560568217502019, 0.56046747121364
1, 0.560338970856869, 0.560245324687359, 0.560191500669393, 0.56010092972655, 0.560016281106482,
\n0.559918271001226, 0.559872094622559, 0.559790839559822, 0.55975428391107, 0.559707875757961,
0.559715771328459, 0.559623364404436, 0.559570173297485, 0.559484750131251, 0.559441265393131,
0.559375805870407, 0.559373188820555, 0.559350304985298, 0.559279032436097, 0.559225316756579,
0.559185315922869, 0.559161047593498, 0.559160545360636), val_accuracy = c(0.707167744636536, 0.
714054942131042, 0.715500354766846, 0.718221247196198, 0.71881639957428, 0.718476295471191, 0.71
9241559505463, 0.719581663608551, \n0.718986451625824, 0.719496667385101, 0.719326615333557, 0.7
19751715660095, 0.720601975917816, 0.721282184123993, 0.721452236175537, 0.72187739610672, 0.722
812712192535, 0.723067760467529, 0.723577916622162, 0.723747968673706, 0.723577916622162, 0.7238
33024501801, 0.723747968673706, 0.72391802072525, 0.723747968673706, 0.724343180656433, 0.723662
972450256, 0.723577916622162, 0.723833024501801, 0.72391802072525, 0.724173128604889, 0.72442817
6879883, 0.72502338886261, 0.725363492965698, 0.725363492965698, \n0.725278437137604, 0.72493833
3034515, 0.725193440914154, 0.725363492965698, 0.725703597068787, 0.725618541240692, 0.725533545
017242, 0.725703597068787, 0.725193440914154, 0.724768280982971, 0.725448489189148, 0.7261287569
99969, 0.726383805274963, 0.726383805274963, 0.725703597068787))"
```

```
paste("MODEL PERFORMANCE with ADAM optimizer",model_ann0_fit)
```

```
## [1] "MODEL PERFORMANCE with ADAM optimizer list(batch_size = 200, epochs = 50, steps = 138, s
amples = 27440, verbose = 0, do_validation = TRUE, metrics = c(\"loss\", \"accuracy\", \"val_los
s\", \"val_accuracy\"))"
## [2] "MODEL PERFORMANCE with ADAM optimizer list(loss = c(0.665677072789857, 0.61728368095684
3, 0.594813277961214, 0.582656037181877, 0.575694208459673, 0.571612087005081, 0.56912829285981
7, 0.567451958232301, 0.566260115528593, 0.565310904739897, 0.564610833198962, 0.56406251534428
6, 0.563480261333135, 0.563111475535801, 0.56277735974455, 0.562557864206525, 0.562339211214041,
0.562141366802569, 0.562010313392381, 0.561904796054342, 0.561856921968585, 0.561794181259311,
0.561744006095405, 0.561653072476039, 0.561666187452853, 0.561695466409967, \n0.561630451210039,
0.561633256060389, 0.56165993183243, 0.561653129951947, 0.561603711793096, 0.561631871208158, 0.
561612466551125, 0.561607389642963, 0.561605033174201, 0.56162929665243, 0.56161719834318, 0.561
662558868049, 0.561614842569515, 0.561574467286772, 0.561626436411466, 0.561588743554955, 0.5615
99448888017, 0.561605889838917, 0.561593451062027, 0.561571806278243, 0.561620656107675, 0.56162
6491411087, 0.56161304096265, 0.561575083882051), accuracy = c(0.613265335559845, 0.690051019191
742, \n0.704409599304199, 0.712463557720184, 0.716618061065674, 0.717857122421265, 0.72113704681
3965, 0.722266793251038, 0.723615169525146, 0.725218653678894, 0.725437343120575, 0.726895034313
202, 0.727878987789154, 0.728389203548431, 0.729154527187347, 0.729227423667908, 0.7297740578651
43, 0.729883372783661, 0.729883372783661, 0.730976700782776, 0.730575799942017, 0.73061221837997
4, 0.7301384806633, 0.730903804302216, 0.730721592903137, 0.730065584182739, 0.730502903461456,
0.730393588542938, 0.730721592903137, \n0.730247795581818, 0.73035717010498, 0.730430006980896,
0.730758011341095, 0.730648696422577, 0.730502903461456, 0.730466485023499, 0.730612218379974,
0.730830907821655, 0.730320692062378, 0.730758011341095, 0.730247795581818, 0.73028427362442, 0.
73035717010498, 0.730174899101257, 0.730539381504059, 0.730685114860535, 0.730721592903137, 0.73
0502903461456, 0.730393588542938, 0.730102062225342), val_loss = c(0.634121375311578, 0.60200721
7157273, 0.585738700705821, 0.57654950973706, 0.570973160350802, \n0.567562223393012, 0.56518364
9571126, 0.563812055693994, 0.56250044486827, 0.561842776993973, 0.561002788880419, 0.5603147988
59874, 0.559844208350545, 0.559505692946141, 0.559145234117864, 0.558944688936184, 0.55890088946
5673, 0.558794134080374, 0.558460820056723, 0.558613301109025, 0.55836958699603, 0.5584989070497
03, 0.558266419561002, 0.558392389327853, 0.558095124077811, 0.55819083919899, 0.55829259072849
4, 0.558108621025337, 0.558129865329957, 0.558086636261859, 0.558028559575893, 0.55834593470633
2, \n0.558134651647296, 0.558107615961589, 0.558085675041303, 0.558132083153869, 0.5579814826151
86, 0.558145254067383, 0.558079716908102, 0.558394433831532, 0.558133697273597, 0.55810536210426
1, 0.558144487944218, 0.55803299702792, 0.55810127877812, 0.557918511420405, 0.557974537647801,
0.558001543898615, 0.558023436424653, 0.557969523509053), val_accuracy = c(0.674857556819916, 0.
701470971107483, 0.709633529186249, 0.716180622577667, 0.71881639957428, 0.721452236175537, 0.72
3662972450256, 0.722727656364441, \n0.72298276424408, 0.723492920398712, 0.724173128604889, 0.72
4088072776794, 0.723833024501801, 0.724513232707977, 0.725703597068787, 0.726213753223419, 0.725
10838508606, 0.726298809051514, 0.727319121360779, 0.726638913154602, 0.727234065532684, 0.72672
3909378052, 0.727744221687317, 0.727234065532684, 0.727999329566956, 0.727744221687317, 0.727064
01348114, 0.727744221687317, 0.727149069309235, 0.727914273738861, 0.728339433670044, 0.72680896
5206146, 0.727489173412323, 0.727659225463867, 0.728339433670044, \n0.728254377841949, 0.7280843
25790405, 0.727914273738861, 0.728084325790405, 0.72697901725769, 0.727489173412323, 0.726979017
25769, 0.727999329566956, 0.72706401348114, 0.728339433670044, 0.728254377841949, 0.727829277515
411, 0.72697901725769, 0.728084325790405, 0.727914273738861))"
```

Tuning the number of epochs and batch size parameters by calling the model_keras_ann0 function written above which returns a 0 layer ANN0 network in every iteration of the loop. Seeing which batch size & epochs combination produces the best validation accuracy using ADAM optimizer network

The number of epochs and batch size should be chosen in a way that both training and validation accuracies are high thus ensuring that the model doesn’t overfit by producing a high training accuracy and low validation accuracy

```
batch_size = c(50,100,150,200)
epochs = c(10,15,20,40,60,100)
model_ann0_tuned_fit <- c(0)
for (i in batch_size) {
  for (j in epochs) {

    model_fit <- fit(verbose=0,
  object = model_keras_ann0('adam'),
  x = as.matrix(x_train_disease_pred_tbl),
  y = y_train_vec,
  batch_size = i,
  epochs = j,
  validation_split = 0.30
)
  model_ann0_tuned_fit <- c(model_ann0_tuned_fit,model_fit)
  }
}
```

Storing the training and validation accuracies for different batch size and epoch values to compare the accuracies for each combination

```
batch_sizes <- c()
epochs_run <- c()
validation_accuracies <- c()
training_accuracies <- c()
for (i in seq(2,48,2))
{
  batch_sizes <- c(batch_sizes,model_ann0_tuned_fit[i]$params$batch_size)
  epochs_run <- c(epochs_run,model_ann0_tuned_fit[i]$params$epochs)
}

for (i in seq(3,49,2)) {
  validation_accuracies <- c(validation_accuracies,tail(model_ann0_tuned_fit[i]$metrics$val_accu
racy,n=1))
  training_accuracies <- c(training_accuracies, tail(model_ann0_tuned_fit[i]$metrics$accuracy,n=
1))
}
```

Table summarizing the model performances and their corresponding batch size and epoch hyperparameters

The batch_size values, epochs values and their corresponding training accuracy as well as validation accuracy can be identified from the below table

```
performance_summary_df <- data.frame(batch_sizes,epochs_run,training_accuracies,validation_accur
acies)
performance_summary_df
```

```
##    batch_sizes epochs_run training_accuracies validation_accuracies
## 1           50         10           0.7307216             0.7274041
## 2           50         15           0.7315598             0.7277442
## 3           50         20           0.7305394             0.7279993
## 4           50         40           0.7302114             0.7264689
## 5           50         60           0.7305029             0.7273191
## 6           50        100           0.7304665             0.7271491
## 7          100         10           0.7301749             0.7265539
## 8          100         15           0.7310131             0.7266389
## 9          100         20           0.7302843             0.7278293
## 10         100         40           0.7316326             0.7266389
## 11         100         60           0.7315962             0.7281694
## 12         100        100           0.7303572             0.7275742
## 13         150         10           0.7277697             0.7241731
## 14         150         15           0.7305029             0.7280843
## 15         150         20           0.7309402             0.7283394
## 16         150         40           0.7308673             0.7275742
## 17         150         60           0.7305758             0.7268090
## 18         150        100           0.7302843             0.7269790
## 19         200         10           0.7247449             0.7240881
## 20         200         15           0.7293732             0.7267239
## 21         200         20           0.7312317             0.7275742
## 22         200         40           0.7303936             0.7269790
## 23         200         60           0.7304665             0.7266389
## 24         200        100           0.7311224             0.7271491
```

Identifying the number of Epochs and Batch Size which produced the maximum validation accuracy.

```
performance_summary_df$epochs_run[which.max(performance_summary_df$validation_accuracies)]
```

```
## [1] 20
```

```
performance_summary_df$batch_sizes[which.max(performance_summary_df$validation_accuracies)]
```

```
## [1] 150
```

Hence going ahead with the batch size and epochs that produced the maximum validation accuracy to fit the final ANN0 deep learning model. When the validation accuracy is also as high as training accuracy, we can be assured that the model will produce low bias and low variance estimates.

```
dl_model_keras_ann0 <- model_keras_ann0('adam')
model_ann0_fit <- fit(verbose=0,
  object = dl_model_keras_ann0,
  x = as.matrix(x_train_disease_pred_tbl),
  y = y_train_vec,
  batch_size = performance_summary_df$batch_sizes[which.max(performance_summary_df$validation_ac
curacies)],
  epochs = performance_summary_df$epochs_run[which.max(performance_summary_df$validation_accurac
ies)],
  validation_split = 0.30
)
```

Making Predictions on the split Test Data using the final ANN0 deep learning model built above using the tibble function and yardstick library
Accuracy and Recall are the 2 most important metrics because Recall helps in minimizing the False Negatives which are highly undesirable when doing disease prediction

```
yhat_keras_class_vec <- predict_classes(object = dl_model_keras_ann0, x = as.matrix(x_test_disea
se_pred_tbl)) %>% as.vector()
yhat_keras_prob_vec <- predict_proba(object = dl_model_keras_ann0, x = as.matrix(x_test_disease_
pred_tbl)) %>% as.vector()
estimates_keras_tbl <- tibble::tibble(
  truth = as.factor(y_test_vec),
  estimate = as.factor(yhat_keras_class_vec),
  class_prob = yhat_keras_prob_vec
)
#estimates_keras_tbl
```

Evaluating the ANN0 model performance on the test dataset (Hold One out method), using the same set of metrics: Accuracy, Precision, Recall, F1 score and AUC score to evaluate the performance of the ANN0 Deep Learning model

Since the accuracy on the testing data is completely inline with the training accuracy we can be assured that the model is producing low bias and low variance estimate

```
library(yardstick)
```

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

```
## For binary classification, the first factor level is assumed to be the event.
## Set the global option `yardstick.event_first` to `FALSE` to change this.
```

```
##
## Attaching package: 'yardstick'
```

```
## The following object is masked from 'package:keras':
##
##     get_weights
## The following objects are masked from 'package:caret':
##
##     precision, recall, sensitivity, specificity
```

```
## The following object is masked from 'package:readr':
##
##     spec
```

```
options(yardstick.event_first = F)
ann0_accuracy <- estimates_keras_tbl %>% metrics(truth, estimate)
ACCURACY<- ann0_accuracy$.estimate[1]

ann0_auc <-estimates_keras_tbl %>% roc_auc(truth, class_prob)
AUC<-ann0_auc$.estimate[1]

ann0_precision <-estimates_keras_tbl %>% precision(truth, estimate)
PRECISION <- ann0_precision$.estimate[1]

ann0_recall <-estimates_keras_tbl %>% recall(truth, estimate)
RECALL<- ann0_recall$.estimate[1]

ann0_f1 <-estimates_keras_tbl %>% f_meas(truth, estimate)
F1 <- ann0_f1$.estimate[1]

data.frame(ACCURACY,AUC,PRECISION,RECALL,F1)
```

```
##    ACCURACY       AUC PRECISION    RECALL        F1
## 1 0.7272171 0.7933942 0.7522584 0.6788262 0.7136583
```

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

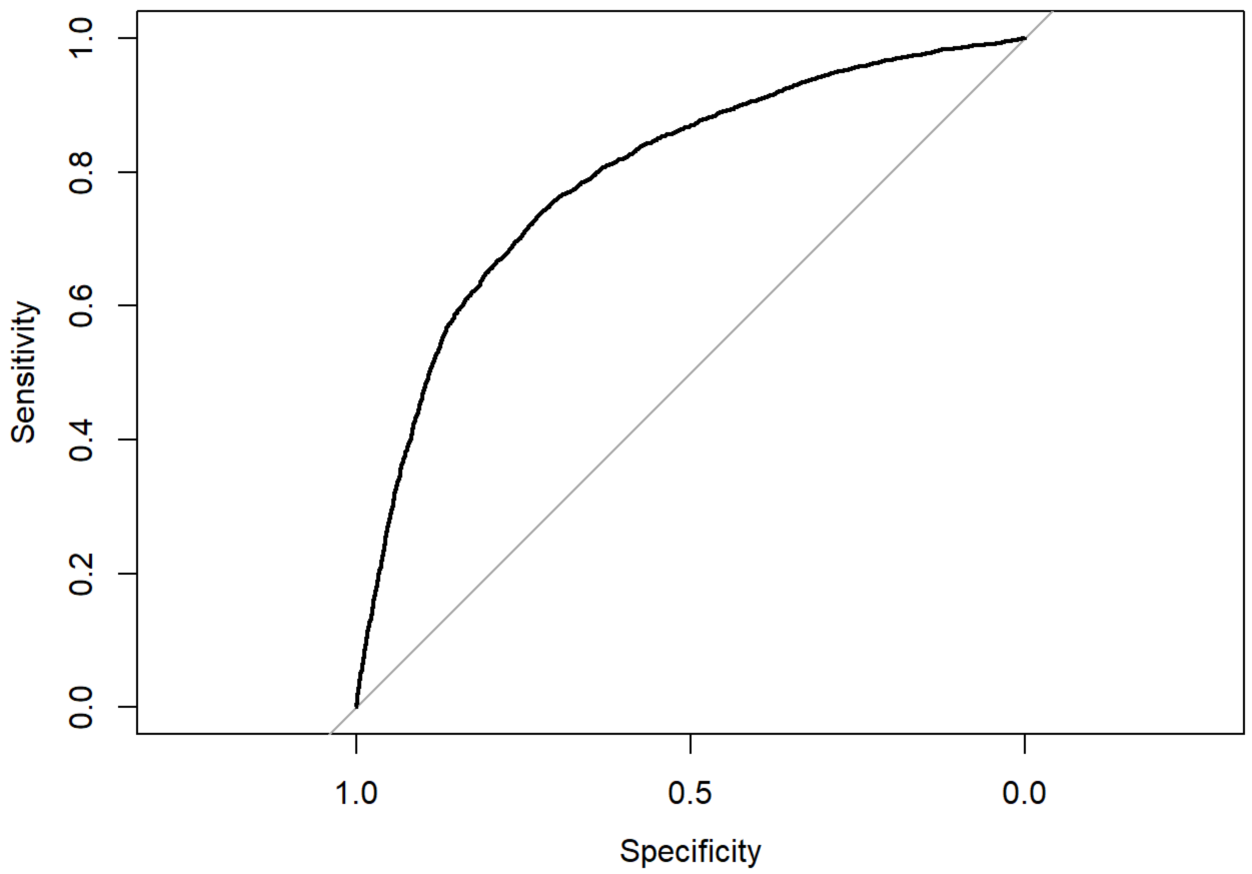
Generated ROC curve and calculated Area Under Curve metric for the identified best performing ANN0 DL model

```
roc_curve <- roc(y_test_vec,yhat_keras_prob_vec)
```

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

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

```
plot(roc_curve)
```



Obtained the area under the ROC curve for the best ANN0 model

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

```
## [1] "Area Under the ROC curve is : 0.793394150829301"
```

END OF BUILDING ANN0 Deep Learning Model (algorithm number 2)

Comparing preformances of Linear SVM, Logistic Regression and ANN0 (Perceptron with 0 hidden layers)

```
## LOGISTIC REGRESSION
disease_prediction_training_df_test_split$predicted_disease_logistic <- predict_disease_tuned_lo
gistic
conf_matrix <- data.frame(table(disease_prediction_training_df_test_split$Disease,disease_predic
tion_training_df_test_split$predicted_disease_logistic))
colnames(conf_matrix)<- c('Actual class','Predicted Class','Count')
Accuracy_Logistic <- sum(conf_matrix$Count[conf_matrix$`Actual class`==conf_matrix$`Predicted Cl
ass`])/sum(conf_matrix$Count)
Precision_Logistic <- conf_matrix$Count[conf_matrix$`Actual class`== 1 & conf_matrix$`Predicted
 Class`== 1]/sum(conf_matrix$Count[conf_matrix$`Predicted Class`==1])
Recall_Logistic <- conf_matrix$Count[conf_matrix$`Actual class`==1 & conf_matrix$`Predicted Clas
s`==1]/sum(conf_matrix$Count[conf_matrix$`Actual class`==1])
F1_score_Logistic <- (2*Precision_DT*Recall_DT)/(Precision_DT+Recall_DT)
logistic_pred_disease <- predict(tuned_model_logistic, newdata = disease_prediction_training_df_
test_split, na.action = na.omit, type = "prob")
roc_curve_logistic <- roc(disease_prediction_training_df_test_split$Disease,logistic_pred_diseas
e$`1`)
```

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

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

```
auc_logistic <- auc(roc_curve_logistic)

## PERFORMANCE COMPARISON
ALGORITHMs <- c('LOGISTIC REGRESSION','ANN0','LINEAR SVM')
ACCURACIES <- c(Accuracy_Logistic,ACCURACY,0.7288)
PRECISIONs <- c(Precision_Logistic,PRECISION,0.7988)
RECALLs <- c(Recall_Logistic,RECALL,0.61178)
F1_scores <- c(F1_score_Logistic,F1,0.72558)

data.frame(ALGORITHMs,ACCURACIES,PRECISIONs,RECALLs,F1_scores)
```

```
##            ALGORITHMs ACCURACIES PRECISIONs   RECALLs F1_scores
## 1 LOGISTIC REGRESSION  0.7273964  0.7603615 0.6640359 0.7089417
## 2                ANN0  0.7272171  0.7522584 0.6788262 0.7136583
## 3          LINEAR SVM  0.7288000  0.7988000 0.6117800 0.7255800
```

Discussion on comparison of Linear SVM, Logistic Regression and ANN0 (Perceptron with 0 hidden layers)

The performances of Logistic Regression, Linear SVM and single layer peceptron (ANN0) are very much comparable because all the algorithms work on the concept of dividing the data points by constructing a linear hyperplane that separates the data into 2 classes. The single layer perceptron is also a linear combination of predictor variables (weighted sum of predictors) just like logistic regression, because of which both these algorithms construct a linear hyperplane seperating the data points. Also linear SVM works on the exact same principle of constructing a linear hyperplane seperating the data points, similar to logistic regression and ANN0. Moreover, the method of obtaining target variable probabilities is same in

Logistic Regression and ANN0, both these methods start of by taking a weighted sum (linear combination) of predictor variables and then apply the sigmoid function to supress the output between 0 and 1. Hence all the 3 algorithms have similar performance.

2.3 Artificial Neural Network - 1 hidden layer (ANN1)

Using the same X train, X test dataset tables and Y train, Y test vectors

Building neural network architecture with 1 hidden layer (ANN1)

The number of nodes in the first hidden layer should be equal to the number of input attributes in the datatset
The Dropout layers are used to control overfitting by eliminating weights below a cutoff threshold in order to prevent low weights from overfitting the layers. Here we are removing weights below 10%.
Using the relu activation function after the hidden layer as relu produces 0 as output for all negative inputs and the input value itself is produced as output if the input is > 0. Hence, the relu activation function is more suitable for taking partial derivatives. The Sigmoid function is being used after the output layer to produce output between 0 and 1.
Using the binary_crossentropy loss function since it is a binary classification problem
Wriiten a function below, which returns a Neural Network with one hidden layer by taking in the optimizer as input parameter

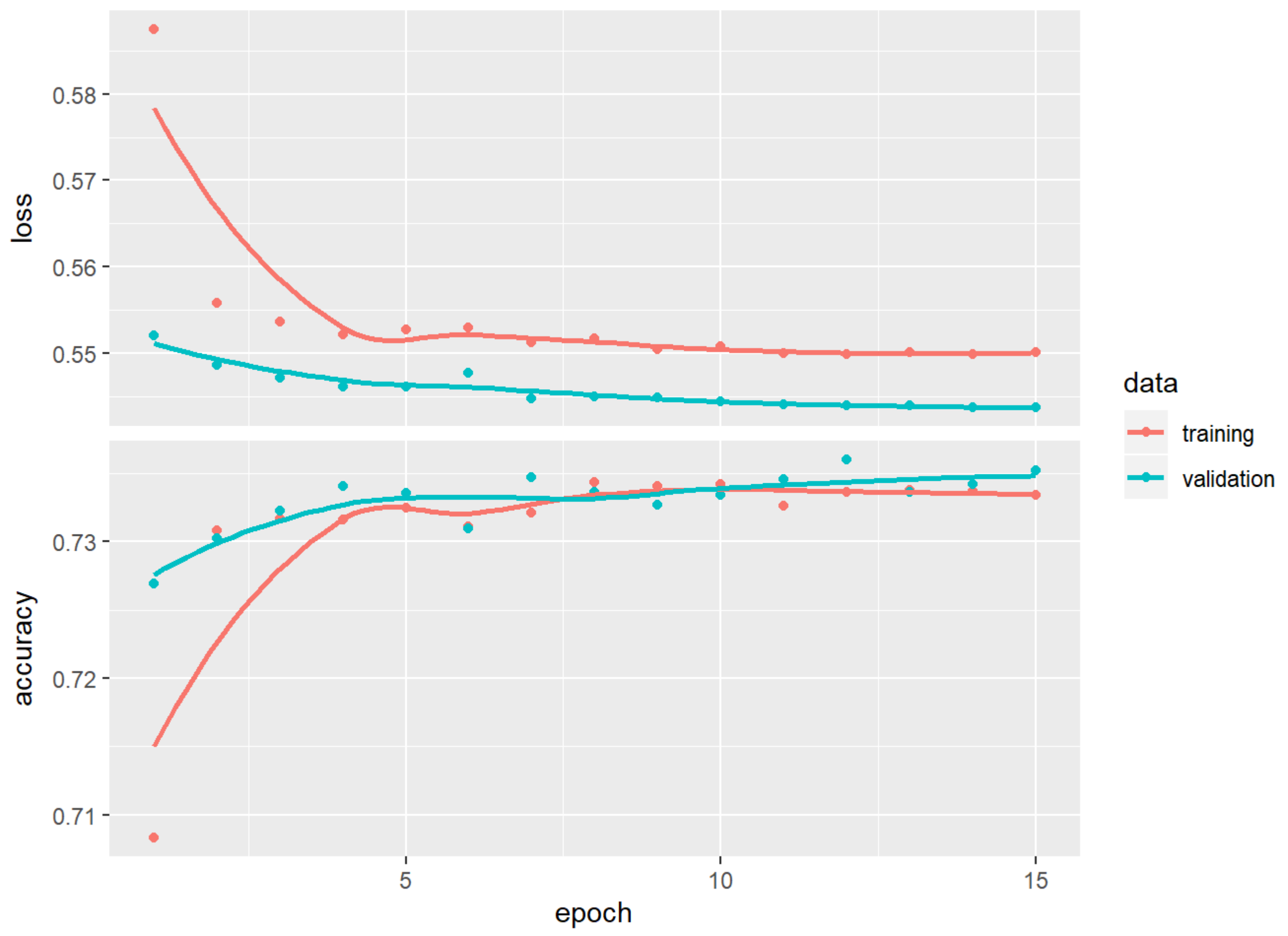
```
library(keras)
library(tensorflow)

model_keras_ann1 <- function(opt) {

model_keras_ann <- keras_model_sequential()
model_keras_ann %>%
  ## hidden Layer 1
  layer_dense(units = 16,
    kernel_initializer = "uniform",
    activation = "relu",
    input_shape = ncol(x_train_disease_pred_tbl)) %>%
  layer_dropout(rate = 0.1)%>%
  ## Output Layer
  layer_dense(units = 1,
    kernel_initializer = "uniform",
    activation = "sigmoid") %>%
    compile(optimizer = "ADAM",
    loss = "binary_crossentropy",
    metrics = c("accuracy")
  )

return(model_keras_ann)

}
```

Fitting the Neural Network with 1 hidden layer onto the train data

```
model_ann1_fit <- fit(verbose=0,
  object = model_keras_ann1('adam'),
  x = as.matrix(x_train_disease_pred_tbl),
  y = y_train_vec,
  batch_size = 50,
  epochs = 15,
  validation_split = 0.30
)
```

From the below plot also we can identify the suitable number of epochs based on the point where the curve begins to flatten

```
plot(model_ann1_fit)
```



Tuning the number of epochs and batch size parameters to see which combination produces the best validation accuracy using ADAM optimizer network by using the ANN1 function written above which return a DL neural network with 1 hidden layer in every iteration of the loop, by taking the optimizer as the function input parameter.

The number of epochs & batch size should be chosen in a way that both training and validation accuracies are high thus ensuring that the model doesn't overfit by producing a high training accuracy and low validation accuracy

```
batch_size = c(50,100,150,200)
epochs = c(10,15,20,40,60,100)
model_ann1_tuned_fit <- c(0)
for (i in batch_size) {
  for (j in epochs) {

    model_fit <- fit(verbose=0,
  object = model_keras_ann1('adam'),
  x = as.matrix(x_train_disease_pred_tbl),
  y = y_train_vec,
  batch_size = i,
  epochs = j,
  validation_split = 0.30
)
  model_ann1_tuned_fit <- c(model_ann1_tuned_fit,model_fit)
  }

 }
```

Storing the training and validation accuracies for different batch size and epoch values to compare the accuracies for each combination

```
batch_sizes <- c()
epochs_run <- c()
validation_accuracies <- c()
training_accuracies <- c()
for (i in seq(2,48,2))
{
  batch_sizes <- c(batch_sizes,model_ann1_tuned_fit[i]$params$batch_size)
  epochs_run <- c(epochs_run,model_ann1_tuned_fit[i]$params$epochs)
}

for (i in seq(3,49,2)) {
  validation_accuracies <- c(validation_accuracies,tail(model_ann1_tuned_fit[i]$metrics$val_accu
racy,n=1))
  training_accuracies <- c(training_accuracies, tail(model_ann1_tuned_fit[i]$metrics$accuracy,n=
1))
}
```

Table summarizing the model performances and their corresponding batch size and epoch hyperparameters

The batch_size value & epochs values with their correpsonding training accuracy and validation accuracy can be seen in the below table

```
performance_summary_df <- data.frame(batch_sizes,epochs_run,training_accuracies,validation_accur
acies)
performance_summary_df
```

```
##    batch_sizes epochs_run training_accuracies validation_accuracies
## 1           50         10           0.7326531             0.7323357
## 2           50         15           0.7348032             0.7347164
## 3           50         20           0.7335641             0.7324207
## 4           50         40           0.7334912             0.7304651
## 5           50         60           0.7353134             0.7342913
## 6           50        100           0.7344024             0.7322506
## 7          100         10           0.7331268             0.7303801
## 8          100         15           0.7340379             0.7322506
## 9          100         20           0.7336735             0.7330159
## 10         100         40           0.7340744             0.7357367
## 11         100         60           0.7348032             0.7339512
## 12         100        100           0.7342566             0.7332710
## 13         150         10           0.7331268             0.7309753
## 14         150         15           0.7343295             0.7329309
## 15         150         20           0.7325802             0.7321656
## 16         150         40           0.7355685             0.7342913
## 17         150         60           0.7339286             0.7331010
## 18         150        100           0.7352770             0.7354817
## 19         200         10           0.7336006             0.7343763
## 20         200         15           0.7325437             0.7315704
## 21         200         20           0.7331997             0.7310603
## 22         200         40           0.7330904             0.7325908
## 23         200         60           0.7361881             0.7348865
## 24         200        100           0.7347303             0.7352266
```

Identified the number of Epochs and Batch Size which produced the highest validation accuracy. High validation accuracy is important to ensure that the model produces low variance estimate.

```
performance_summary_df$epochs_run[which.max(performance_summary_df$validation_accuracies)]
```

```
## [1] 40
```

```
performance_summary_df$batch_sizes[which.max(performance_summary_df$validation_accuracies)]
```

```
## [1] 100
```

Hence going ahead with the batch size and epochs that produce the maximum validation accuracy to fit the final ANN1 deep learning model.

When the validation accuracy is also as high as training accuracy, we can be assured that the model will produce low bias and low variance estimates.

```
dl_model_keras_ann1 <- model_keras_ann1('adam')

model_ann1_fit <- fit(verbose=0,
  object = dl_model_keras_ann1,
  x = as.matrix(x_train_disease_pred_tbl),
  y = y_train_vec,
  batch_size = performance_summary_df$batch_sizes[which.max(performance_summary_df$validation_ac
curacies)],
  epochs = performance_summary_df$epochs_run[which.max(performance_summary_df$validation_accurac
ies)],
  validation_split = 0.30
)
```

Making Predictions on the split Test Data using the final ANN1 DL model built using the tibble function and yardstick library
Accuracy and Recall are the 2 most important metrics because Recall helps in minimizing the False Negatives which are highly undesirable when doing disease prediction

```
yhat_keras_class_vec <- predict_classes(object = dl_model_keras_ann1, x = as.matrix(x_test_disea
se_pred_tbl)) %>% as.vector()
yhat_keras_prob_vec <- predict_proba(object = dl_model_keras_ann1, x = as.matrix(x_test_disease_
pred_tbl)) %>% as.vector()
estimates_keras_tbl <- tibble::tibble(
  truth = as.factor(y_test_vec),
  estimate = as.factor(yhat_keras_class_vec),
  class_prob = yhat_keras_prob_vec
)
#estimates_keras_tbl
```

Evaluating the ANN1 model performance on the test dataset (Hold One out method), using the same set of metrics: Accuracy, Precision, Recall, F1 score and AUC score to evaluate the performance of the ANN1 Deep Learning model

Since the accuracy on the testing data is completely inline with the training accuracy we can be assured that the model is producing low bias and low variance estimate

```
library(yardstick)
options(yardstick.event_first = F)
ann1_accuracy <- estimates_keras_tbl %>% metrics(truth, estimate)
ACCURACY<- ann1_accuracy$.estimate[1]

ann1_auc <-estimates_keras_tbl %>% roc_auc(truth, class_prob)
AUC<-ann1_auc$.estimate[1]

ann1_precision <-estimates_keras_tbl %>% precision(truth, estimate)
PRECISION <- ann1_precision$.estimate[1]

ann1_recall <-estimates_keras_tbl %>% recall(truth, estimate)
RECALL<- ann1_recall$.estimate[1]

ann1_f1 <-estimates_keras_tbl %>% f_meas(truth, estimate)
F1 <- ann1_f1$.estimate[1]

data.frame(ACCURACY,AUC,PRECISION,RECALL,F1)
```

```
##    ACCURACY      AUC PRECISION    RECALL        F1
## 1 0.7333401 0.801648 0.7475183 0.7059303 0.7261293
```

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

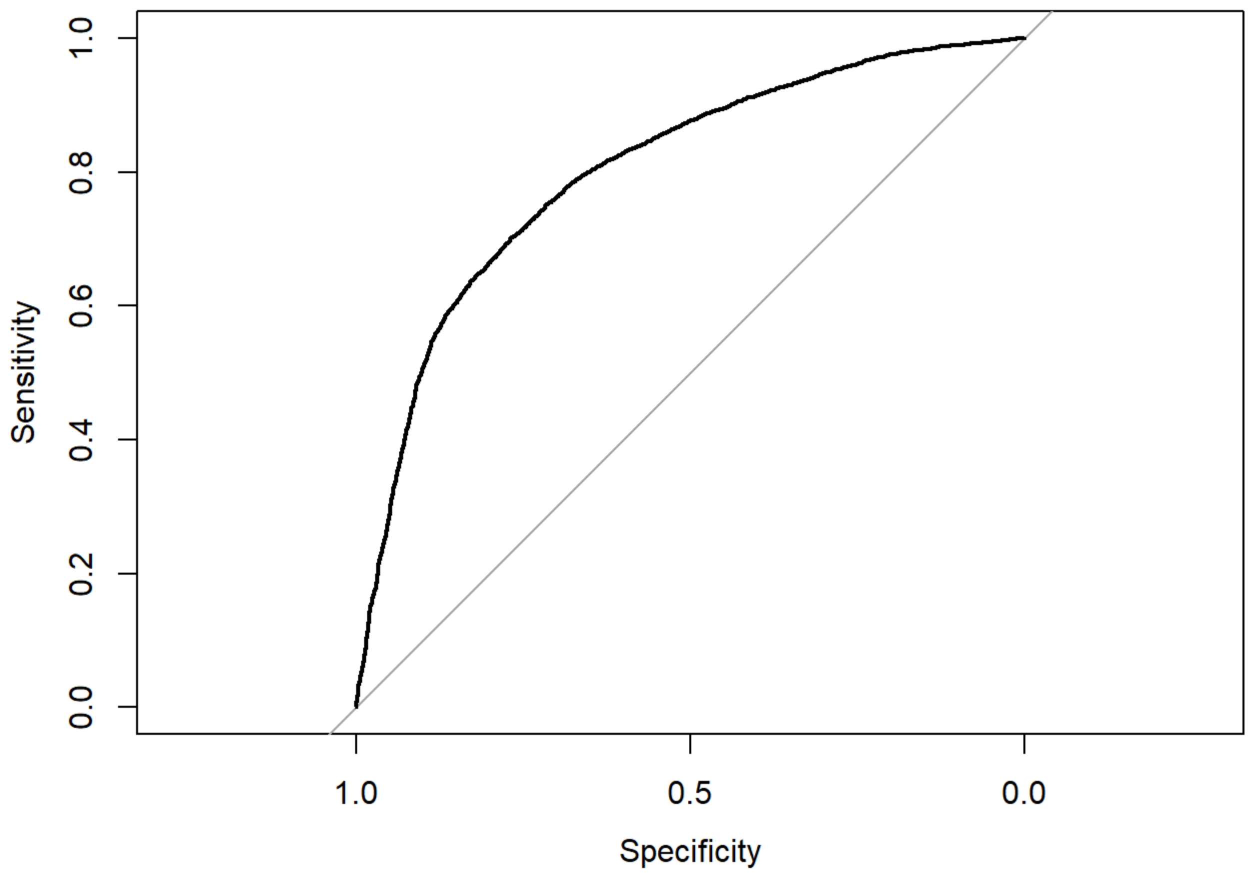
Generated ROC curve and calculated Area Under Curve metric for the identified best performing ANN1 model

```
roc_curve <- roc(y_test_vec,yhat_keras_prob_vec)
```

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

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

```
plot(roc_curve)
```



Obtained the area under the ROC curve for the best ANN1 model

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

```
## [1] "Area Under the ROC curve is : 0.801648020307732"
```

We can observe the improvement in model performance between ANN0 and ANN1

END OF BUILDING ANN1 Deep Learning Model (algorithm number 3)

2.4 Artificial Neural Network - 2 hidden layers (ANN2)

Using the same X train, X test dataset tables and Y train, Y test vectors

Building neural network architecture with 2 hidden layers (ANN2)

The number of nodes in the first hidden layer should be equal to the number of input attributes in the datatset, whereas the number of nodes in the second hidden layer can vary anywhere between 1 and two times of number of input attributes.
Using the relu activation function after the hidden layers as relu produces 0 as output for all negative inputs and the input value itself is produced as output if the input is > 0. Hence, the relu activation function is more suitable for taking partial derivatives. The Sigmoid function itself is being used after the output layer to produce output between 0 and 1.
The Dropout layers are used to control overfitting by eliminating weights below a cutoff threshold in order to prevent low weights from overfitting the layers. Here we are removing weights below 10%.

Using the binary_crossentropy loss function since it is a binary classification problem
Written a function below, which returns a Neural Network with two hidden layers by taking in the optimizer as input parameter

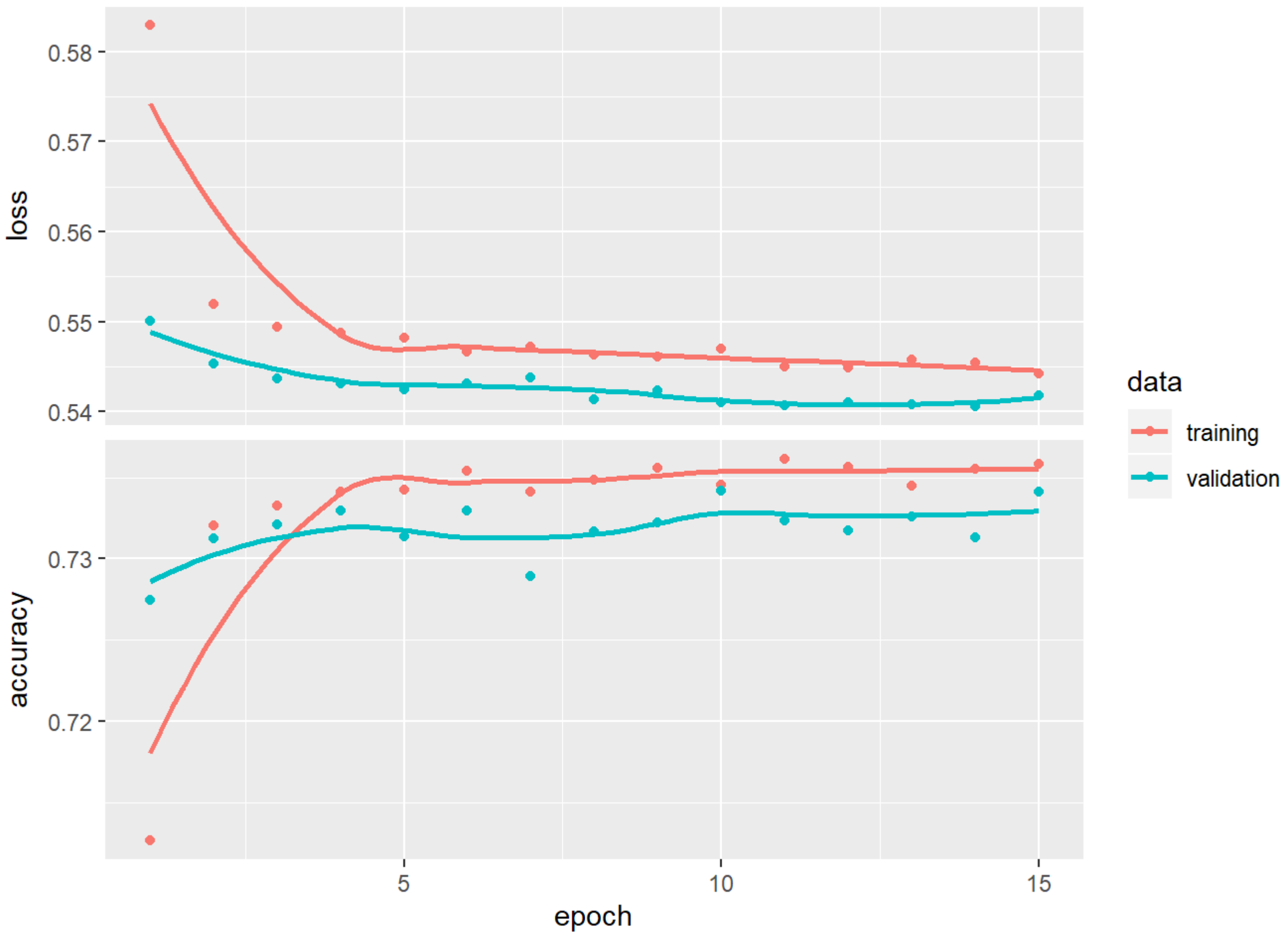
```
library(keras)
library(tensorflow)

model_keras_ann2 <- function(opt) {

model_keras_ann <- keras_model_sequential()
model_keras_ann %>%
  ## Hidden Layer 1
  layer_dense(units = 16,
    kernel_initializer = "uniform",
    activation = "relu",
    input_shape = ncol(x_train_disease_pred_tbl)) %>%

  ## Hidden Layer 2
  layer_dense(units = 12,
    kernel_initializer = "uniform",
    activation = "relu") %>%
  layer_dropout(rate = 0.1)%>%

  ## Output Layer
  layer_dense(units = 1,
    kernel_initializer = "uniform",
    activation = "sigmoid") %>%
    compile(optimizer = opt,
    loss = "binary_crossentropy",
    metrics = c("accuracy")
  )

return(model_keras_ann)

}
```

Fitting the Neural Network with 2 hidden layers onto the train data

```
model_ann2_fit <- fit(verbose=0,
  object = model_keras_ann2('adam'),
  x = as.matrix(x_train_disease_pred_tbl),
  y = y_train_vec,
  batch_size = 50,
  epochs = 15,
  validation_split = 0.30
)
```

From the below plot also we can identify the suitable number of epochs based on the point where the curve begins to flatten

```
plot(model_ann2_fit)
```



Tuning the number of epochs and batch size parameters to see which combination produces the best validation accuracy using ADAM optimizer network by using the ANN2 function written above which returns a DL neural network with 2 hidden layers in every iteration of the loop, by taking the optimizer as the function input parameter.

The number of epochs & batch size should be chosen in a way that both training and validation accuracies are high thus ensuring that the model doesn’t overfit by producing a high training accuracy and low validation accuracy

```
batch_size = c(50,100,150,200)
epochs = c(10,15,20,40,60,100)
model_ann2_tuned_fit <- c(0)
for (i in batch_size) {
  for (j in epochs) {

    model_fit <- fit(verbose=0,
  object = model_keras_ann2('adam'),
  x = as.matrix(x_train_disease_pred_tbl),
  y = y_train_vec,
  batch_size = i,
  epochs = j,
  validation_split = 0.30
)
  model_ann2_tuned_fit <- c(model_ann2_tuned_fit,model_fit)
  }

 }
```

Storing the training and validation accuracies for different batch size and epoch values to compare the accuracies for each combination

```
batch_sizes <- c()
epochs_run <- c()
validation_accuracies <- c()
training_accuracies <- c()
for (i in seq(2,48,2))
{
  batch_sizes <- c(batch_sizes,model_ann2_tuned_fit[i]$params$batch_size)
  epochs_run <- c(epochs_run,model_ann2_tuned_fit[i]$params$epochs)
}

for (i in seq(3,49,2)) {
  validation_accuracies <- c(validation_accuracies,tail(model_ann2_tuned_fit[i]$metrics$val_accu
racy,n=1))
  training_accuracies <- c(training_accuracies, tail(model_ann2_tuned_fit[i]$metrics$accuracy,n=
1))
}
```

Table summarizing the model performances and their corresponding batch size and epoch hyperparameters

The batch_size value & epochs values with their correpsonding training accuracy and validation accuracy can be seen in the below table

```
performance_summary_df <- data.frame(batch_sizes,epochs_run,training_accuracies,validation_accur
acies)
performance_summary_df
```

```
##    batch_sizes epochs_run training_accuracies validation_accuracies
## 1           50         10           0.7354956             0.7321656
## 2           50         15           0.7343295             0.7319105
## 3           50         20           0.7361152             0.7360769
## 4           50         40           0.7400146             0.7344614
## 5           50         60           0.7385204             0.7358218
## 6           50        100           0.7385204             0.7333560
## 7          100         10           0.7352405             0.7324207
## 8          100         15           0.7345481             0.7334411
## 9          100         20           0.7353498             0.7328458
## 10         100         40           0.7362610             0.7338662
## 11         100         60           0.7390306             0.7348015
## 12         100        100           0.7379009             0.7343763
## 13         150         10           0.7350219             0.7341213
## 14         150         15           0.7337463             0.7324207
## 15         150         20           0.7357872             0.7327608
## 16         150         40           0.7363338             0.7347164
## 17         150         60           0.7383747             0.7351416
## 18         150        100           0.7381560             0.7368421
## 19         200         10           0.7344388             0.7325057
## 20         200         15           0.7349125             0.7336111
## 21         200         20           0.7346939             0.7339512
## 22         200         40           0.7373178             0.7324207
## 23         200         60           0.7377186             0.7342913
## 24         200        100           0.7387755             0.7358218
```

Identified the number of Epochs and Batch Size which produced the highest validation accuracy

```
performance_summary_df$epochs_run[which.max(performance_summary_df$validation_accuracies)]
```

```
## [1] 100
```

```
performance_summary_df$batch_sizes[which.max(performance_summary_df$validation_accuracies)]
```

```
## [1] 150
```

Hence going ahead with the batch size and epochs that produce the maximum validation accuracy to fit the final ANN2 deep learning model

When the validation accuracy is also as high as training accuracy, we can be assured that the model will produce low bias and low variance estimates

```
dl_model_keras_ann2 <- model_keras_ann2('adam')

model_ann2_fit <- fit(verbose=0,
  object = dl_model_keras_ann2,
  x = as.matrix(x_train_disease_pred_tbl),
  y = y_train_vec,
  batch_size = performance_summary_df$batch_sizes[which.max(performance_summary_df$validation_ac
curacies)],
  epochs = performance_summary_df$epochs_run[which.max(performance_summary_df$validation_accurac
ies)],
  validation_split = 0.30
)
```

Making Predictions on the split Test Data using the final ANN2 DL model built using the tibble function and yardstick library
Accuracy and Recall are the 2 most important metrics because Recall helps in minimizing the False Negatives which are highly undesirable when doing disease prediction

```
yhat_keras_class_vec <- predict_classes(object = dl_model_keras_ann2, x = as.matrix(x_test_disea
se_pred_tbl)) %>% as.vector()
yhat_keras_prob_vec <- predict_proba(object = dl_model_keras_ann2, x = as.matrix(x_test_disease_
pred_tbl)) %>% as.vector()
estimates_keras_tbl <- tibble::tibble(
  truth = as.factor(y_test_vec),
  estimate = as.factor(yhat_keras_class_vec),
  class_prob = yhat_keras_prob_vec
)
#estimates_keras_tbl
```

Evaluating the ANN2 model performance on the test dataset (Hold One out method), using the same set of metrics: Accuracy, Precision, Recall, F1 score and AUC score to evaluate the performance of the ANN2 Deep Learning model

Since the accuracy on the testing data is completely inline with the training accuracy we can be assured that the model is producing low bias and low variance estimate

```
library(yardstick)
options(yardstick.event_first = F)
ann2_accuracy <- estimates_keras_tbl %>% metrics(truth, estimate)
ACCURACY<- ann2_accuracy$.estimate[1]

ann2_auc <-estimates_keras_tbl %>% roc_auc(truth, class_prob)
AUC<-ann2_auc$.estimate[1]

ann2_precision <-estimates_keras_tbl %>% precision(truth, estimate)
PRECISION <- ann2_precision$.estimate[1]

ann2_recall <-estimates_keras_tbl %>% recall(truth, estimate)
RECALL<- ann2_recall$.estimate[1]

ann2_f1 <-estimates_keras_tbl %>% f_meas(truth, estimate)
F1 <- ann2_f1$.estimate[1]

data.frame(ACCURACY,AUC,PRECISION,RECALL,F1)
```

```
##    ACCURACY       AUC PRECISION    RECALL        F1
## 1 0.7398714 0.8055349 0.7531129 0.7148971 0.7335076
```

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

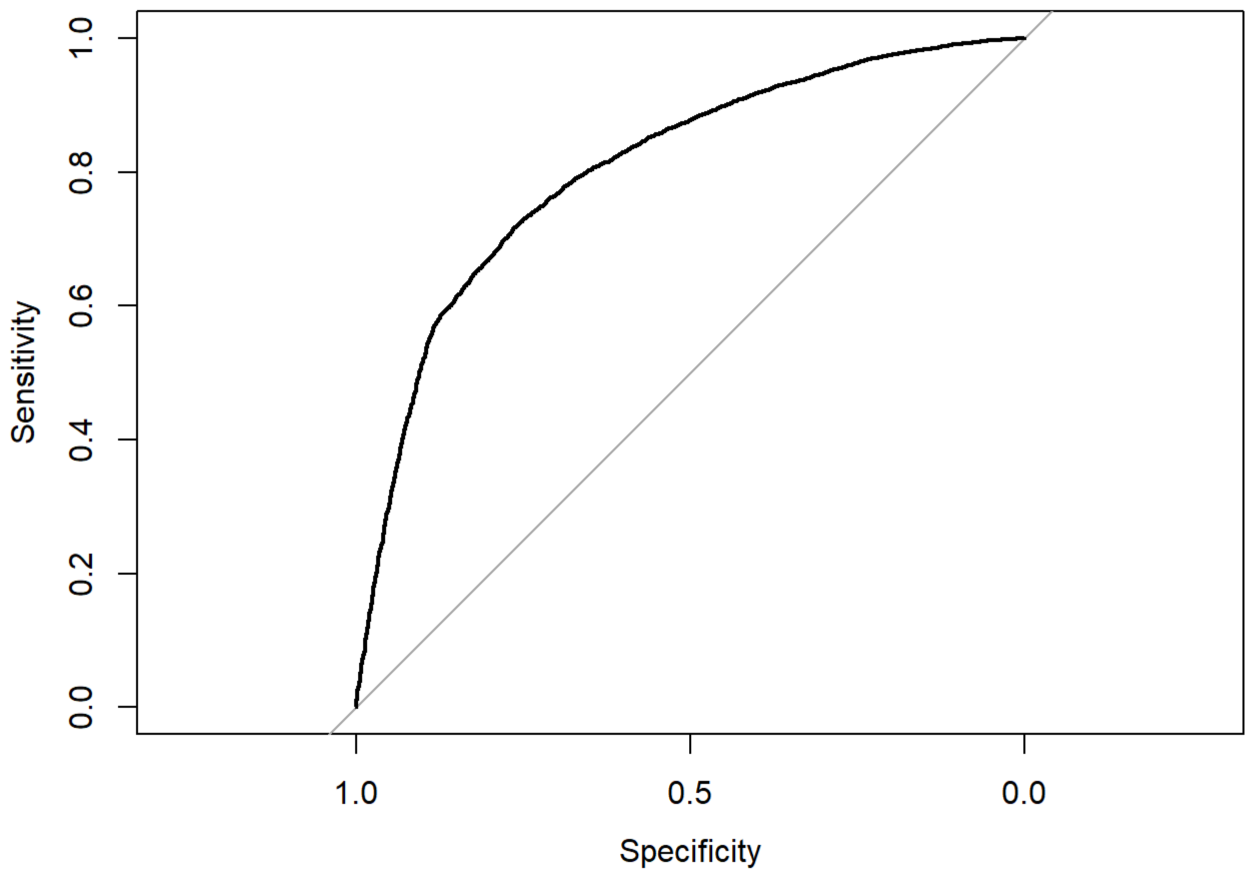
Generated ROC curve and calculated Area Under Curve metric for the identified best performing ANN2 model

```
roc_curve <- roc(y_test_vec,yhat_keras_prob_vec)
```

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

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

```
plot(roc_curve)
```



Obtained the area under the ROC curve for the best ANN2 model

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

```
## [1] "Area Under the ROC curve is : 0.805534911746048"
```

END OF BUILDING ANN2 Deep Learning Model (algorithm number 4)

SECTION 3: Combination & Comparison of multiple Machine Learning Algorithms

3.1 Applying DECISION TREE INDUCTION

Generating Training and Validation datasets for decision tree with 70% & 30% splits respectively

```
library(caret)
library(rpart)
```

```
## Warning: package 'rpart' was built under R version 3.6.2
```

```
set.seed(73)

disease_prediction_training$Disease <- as.factor(disease_prediction_training$Disease)
disease_prediction_training$Smoke <- as.factor(disease_prediction_training$Smoke)
disease_prediction_training$Alcohol <- as.factor(disease_prediction_training$Alcohol)
disease_prediction_training$Exercise <- as.factor(disease_prediction_training$Exercise)
disease_prediction_training$Height <- as.numeric(disease_prediction_training$Height)
disease_prediction_training$age_groups <- NULL
disease_prediction_training$bmi_groups <- NULL

train_index <- createDataPartition(disease_prediction_training$Disease, p = 0.7, list = FALSE)

disease_prediction_dt_tr_df <- disease_prediction_training[train_index, ]

disease_prediction_dt_test_df <- disease_prediction_training[-train_index, ]
```

Building a Decision Tree baseline model only by tuning the Complexity Parameter (CP) to produce best accuracy

```
#dt_model_disease <- train(Disease~., data = disease_prediction_dt_tr_df, method = "rpart",
                     #  metric = "Accuracy",
                      # tuneLength = 8)
```

8 models were tuned and the model with lowest cp produced the highest accuracy, which was used as the final resulting model

```
print(dt_model_disease)
```

```
## CART
##
## 34301 samples
##    12 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:
##
##   cp           Accuracy   Kappa
##   0.000971874  0.7330178  0.46615745
##   0.001166249  0.7327223  0.46554948
##   0.002157560  0.7312533  0.46256670
##   0.002449122  0.7307790  0.46163769
##   0.003032247  0.7294912  0.45908802
##   0.003323809  0.7293075  0.45871772
##   0.009709021  0.7234220  0.44699905
##   0.436235349  0.5414907  0.08659074
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.000971874.
```

Making Classifications on Validation Data using the baseline Decision Tree Model built above using the Training data

```
disease_dt_predict <- predict(dt_model_disease, newdata = disease_prediction_dt_test_df, na.acti
on = na.omit, type = "raw")
```

Evaluating baseline decision tree Model Performance by calculating Accuracy Precision Recall & F1 score on the classification done on the Validation Data

```
disease_prediction_dt_test_df$predicted_disease_baseline <- disease_dt_predict
conf_matrix <- data.frame(table(disease_prediction_dt_test_df$Disease,disease_prediction_dt_test
_df$predicted_disease_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 DT Algorithm in classifying Disease is :",Accuracy_DT)
```

```
## [1] "Accuracy of DT Algorithm in classifying Disease is : 0.72583168923056"
```

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

```
## [1] "Precision of DT Algorithm in classifying Disease is : 0.726182363363773"
```

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

```
## [1] "Recall of DT Algorithm in classifying Disease is : 0.724996598176623"
```

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

```
## [1] "F1 Score of DT Algorithm in classifying Disease is : 0.725588996323029"
```

Introducing the Hyperparameters minbucket, minsplit and maxdepth alongwith CP for tuning the model

Minbucket is the minimum number of observations in any leaf node, Minsplit is the minimum number of observations that must exist in a node in order for a split to be attempted and MaxDepth is the maximum depth of any node of the final tree

Our aim is to tune these parameters to produce maximum accuracy, i.e. a model with minimum bias and at the same time the variance should not be too high by producing a good trade off between Bias and

Variance to avoid underfitting and overfitting.

Hence minsplit, maxdepth and minbucket should be adjusted so as to pre prune the model to produce a least complex model(to reduce overfitting and variance) with maximum accuracy (to minimize bias)

Creating a Grid with all combinations of 4 different values of minsplit, maxdepth and minbucket

```
library(tidyverse)
gs <- list(minsplit = c(20,15,10,5),
           maxdepth = c(30,25,20,15),
           minbucket = c(6,5,4,3)) %>%
  cross_d() # Convert to data frame grid
```

```
## Warning: `cross_d()` is deprecated; please use `cross_df()` instead.
```

Performing Grid Search with all combinations of different values of minsplit, maxdepth and minbucket created above and measuring the accuracy for each combination

```
#accuracy_values <- c()
#for (i in seq(1,nrow(gs))) {

#  dt_model_disease_tuned <- train(Disease ~ ., data = disease_prediction_dt_tr_df,method = "rpa
rt",metric = "Accuracy",
#                         tuneLength = 8,control = rpart.control(minsplit =as.numeric(gs[i,"minspl
it"]), minbucket = as.numeric(gs[i,"minbucket"]),       #maxdepth = as.numeric(gs[i,"maxdept
h"])))
#  accuracy_values<- c(accuracy_values,max(dt_model_disease_tuned$results$Accuracy))
#}
```

Displaying the Accuracy of 64 Decision Tree Models for different combinations of the hyperparameters minsplit, maxdepth and minbucket ordered in descending order by accuracy.

We can see that the first row in the below table with the below shown minsplit, maxdepth and minbucket values has the best accuracy and hence can be considered the best model.

But since the accuracy values are approximately equal, we will use a simpler model with lower maxdepth to avoid overfitting and produce low variance estimates. Based on Occam's Razor principle, we need to chose a simpler model, amongst the models that produce the same accuracy

```
gs$accuracy <- accuracy_values
gs[order(-gs$accuracy),]
```

```
## # A tibble: 64 x 4
##    minsplit maxdepth minbucket accuracy
##       <dbl>    <dbl>     <dbl>    <dbl>
##  1        5       20         6    0.734
##  2       20       15         3    0.733
##  3       15       20         3    0.733
##  4       10       20         3    0.733
##  5       20       30         3    0.733
##  6       20       15         4    0.733
##  7       20       25         4    0.733
##  8       20       20         3    0.733
##  9       15       15         6    0.733
## 10        5       25         4    0.733
## # … with 54 more rows
```

Creating a decision tree model with maxdepth 15 and the above obtained minsplit & minbucket so as avoid overfitting and reduce estimate variance

```
# dt_model_disease_final_tuned <- train(Disease ~ ., data = disease_prediction_dt_tr_df, method
 = "rpart",metric = "Accuracy",
#                        tuneLength = 8,control = rpart.control(minsplit = gs$minsplit[which.max
(gs$accuracy)], minbucket = #gs$minbucket[which.max(gs$accuracy)], maxdepth = 15))
# print(dt_model_disease_final_tuned)
```

Using the final best performing model that produced unbiased and low variance estimates to predict the classes (classify) on the validation dataset

```
disease_dt_prediction <- predict(dt_model_disease_final_tuned, newdata = disease_prediction_dt_t
est_df, na.action = na.omit, type = "raw")
```

Evaluating Model Performance by calculating Accuracy, Precision, Recall & F1 score on the classification done on the Test Data
The training accuracy and validation accuracy are in line, hence the model variance is low

```
disease_prediction_dt_test_df$predicted_disease_tuned <- disease_dt_prediction
conf_matrix <- data.frame(table(disease_prediction_dt_test_df$Disease,disease_prediction_dt_test
_df$predicted_disease_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 DT Algorithm in classifying Disease :",Accuracy_DT)
```

```
## [1] "Accuracy of tuned DT Algorithm in classifying Disease : 0.72583168923056"
```

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

```
## [1] "Precision of tuned DT Algorithm in classifying Disease is : 0.726182363363773"
```

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

```
## [1] "Recall of tuned DT Algorithm in classifying Disease is : 0.724996598176623"
```

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

```
## [1] "F1 Score of tuned DT Algorithm in classifying Disease is : 0.725588996323029"
```

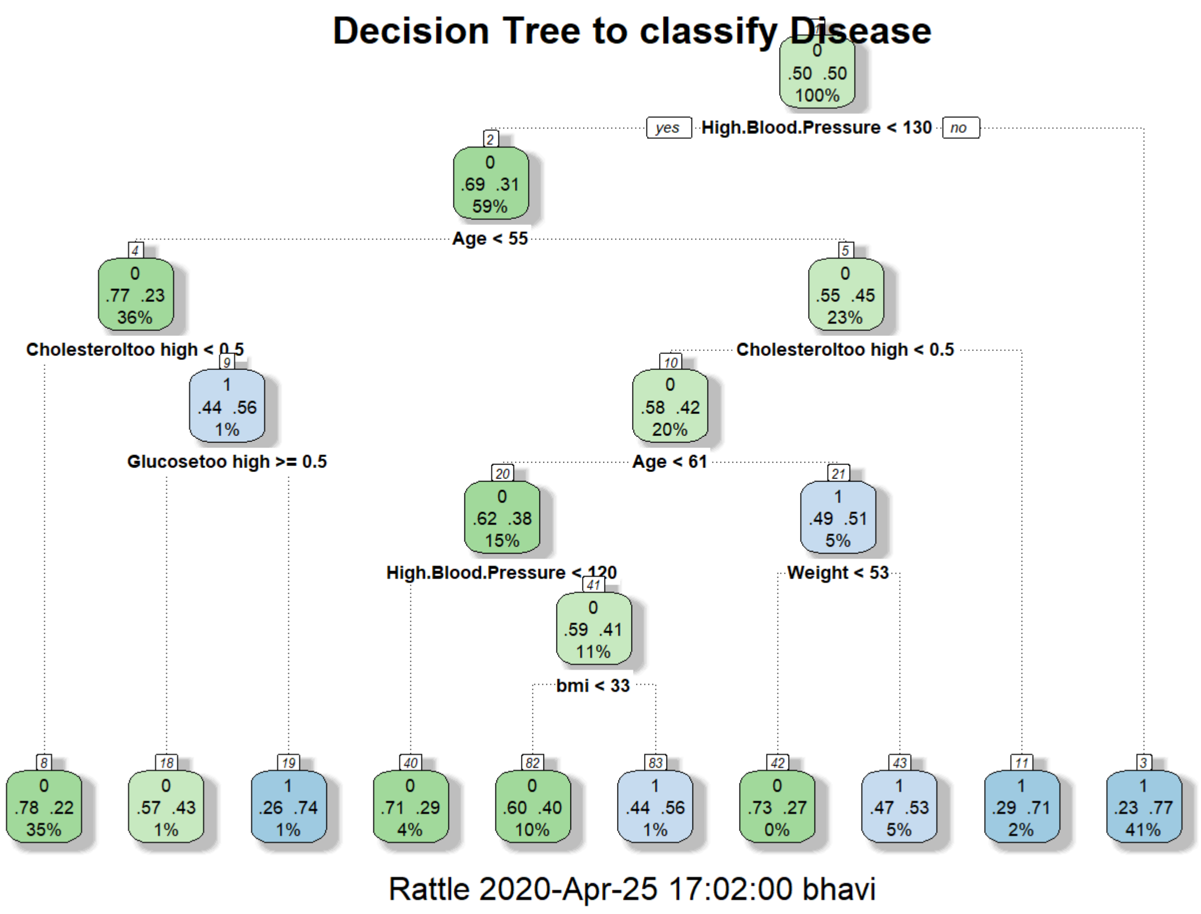
```
#install.packages("rattle")
library(rattle)
```

```
## Warning: package 'rattle' was built under R version 3.6.2
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.3.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

displaying the best performing Decision Tree model

```
fancyRpartPlot(dt_model_disease_final_tuned$finalModel, main = "Decision Tree to classify Diseas
e")
```



ROC & AUROC for Decision Tree Induction

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

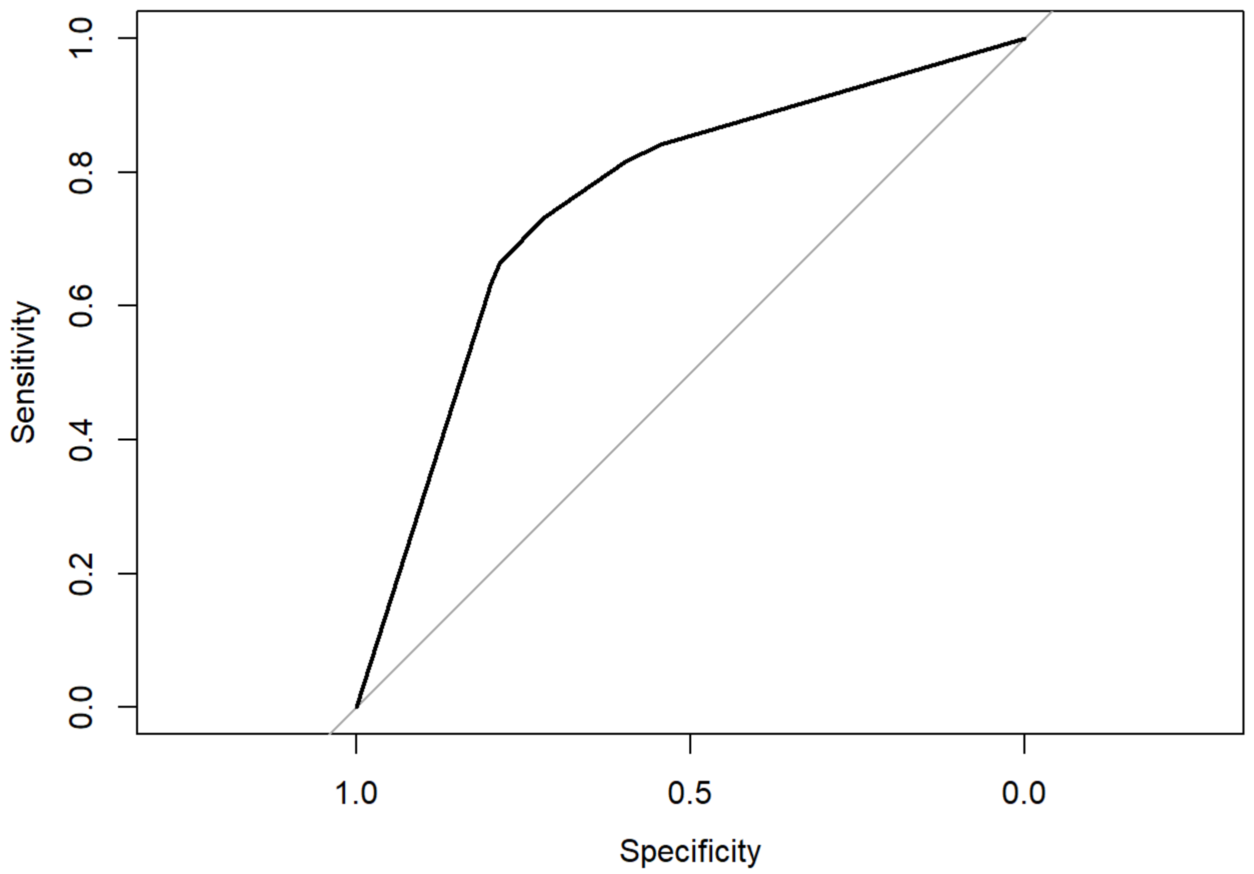
Generated ROC curve and calculated Area Under Curve metric for the identified best performing decision tree model

```
dt_pred_disease_prob <- predict(dt_model_disease_final_tuned, newdata = disease_prediction_dt_te
st_df, na.action = na.omit, type = "prob")
roc_curve <- roc(disease_prediction_dt_test_df$Disease,dt_pred_disease_prob$`1`)
```

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

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

```
plot(roc_curve)
```



Obtained the area under the ROC curve

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

```
## [1] "Area Under the ROC curve is : 0.757847298396839"
```

Analyzing the importance of all the variables in the best performing Decision Tree model & comparing with the importance of variables as per other learning algorithms

1. The top 5 most important features as per Decision Tree are “High Blood Pressure”, “Low Blood Pressure”, “Age”, “Cholestrol high” & “cholestrol normal”

```
varImp(dt_model_disease_final_tuned)
```

```
## rpart variable importance
##
##                         Overall
## High.Blood.Pressure   100.00000
## Low.Blood.Pressure     64.38748
## Age                    35.68229
## Cholesteroltoo high    32.23398
## Cholesterolnormal      31.92579
## bmi                     4.22820
## Exercise1               1.69968
## Weight                  0.96418
## Glucosetoo high         0.64575
## Glucosenormal           0.52399
## Gendermale              0.09931
## Height                  0.08340
## Alcohol1                0.00000
## Smoke1                  0.00000
## `Cholesteroltoo high`   0.00000
## `Glucosetoo high`       0.00000
```

2. The top 5 most important variables as per Logistic Regression which have the lowest P-values and the highest absolute coefficients are, “High Blood Pressure”, “cholestrol_normal”, “Age”, “Cholestrol high” & “bmi”, which is mostly consistent with the top 5 most important variables as per Decision Tree. The only difference is that “bmi” is slightly more important than “low blood pressure” as per logistic regression, due to larger coefficient whereas the vice versa is true for Decision Tree.

```
#library(caret)
#logistic_summary_disease <- train(Disease ~ ., data = disease_prediction_training_df_tr_split,
 method = "glm", family = "binomial")
summary(logistic_summary_disease)
```

```
## 
## Call:
## NULL
## 
## Deviance Residuals: 
##     Min       1Q   Median       3Q      Max  
## -3.6029  -0.9217  -0.1818   0.9268   2.7994  
## 
## Coefficients: (2 not defined because of singularities)
##                              Estimate Std. Error z value Pr(>|z|)    
## (Intercept)                  -4.28167    0.10239 -41.819  < 2e-16 ***
## Age                           1.79840    0.06704  26.824  < 2e-16 ***
## High.Blood.Pressure           6.90232    0.15944  43.290  < 2e-16 ***
## Low.Blood.Pressure            1.30998    0.16306   8.034 9.45e-16 ***
## Smoke1                       -0.14621    0.04897  -2.986  0.00283 ** 
## Alcohol1                     -0.09309    0.05960  -1.562  0.11832    
## Exercise1                    -0.22280    0.03099  -7.188 6.55e-13 ***
## bmi                           6.35296    0.69319   9.165  < 2e-16 ***
## Gender_female1               -0.01696    0.02781  -0.610  0.54184    
## Cholesterol_high1            -0.66489    0.05958 -11.159  < 2e-16 ***
## Cholesterol_normal1          -1.08463    0.05061 -21.432  < 2e-16 ***
## `\\`Cholesterol_too high\\`1`       NA         NA      NA       NA    
## Glucose_high1                 0.31133    0.07311   4.258 2.06e-05 ***
## Glucose_normal1               0.30442    0.05605   5.431 5.60e-08 ***
## `\\`Glucose_too high\\`1`           NA         NA      NA       NA    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## (Dispersion parameter for binomial family taken to be 1)
## 
##     Null deviance: 47551  on 34300  degrees of freedom
## Residual deviance: 38386  on 34288  degrees of freedom
## AIC: 38412
## 
## Number of Fisher Scoring iterations: 4
```

3. The top 5 most important variables as per Random Forest are “High Blood Pressure”, “Low Blood Pressure”, “Age”, “bmi” and “weight” as shown below. The top 3 are same as that of Decision Tree and only the 4th and 5th most important variables are different between Decision Tree and Random Forest.

```
varImp(tuned_model_rf)
```

```
## rf variable importance
## 
##                       Overall
## High.Blood.Pressure 100.00000
## Low.Blood.Pressure   54.69398
## Age                  29.27259
## bmi                  28.70573
## Weight               21.94487
## Height               16.66411
## Cholesterolnormal    12.30199
## Cholesteroltoo high  11.23895
## age_groups58 to 64    8.52774
## Glucosenormal         1.57926
## Exercise              1.54634
## age_groups53 to 58    1.17262
## Gendermale            0.99495
## Glucosetoo high       0.34094
## Smoke                 0.33531
## age_groups48 to 53    0.02419
## Alcohol               0.00000
```

4. The top 5 most important variables as per Gradient Boosting Machine are “High Blood Pressure”, “Age”, “Cholestrol high”, “bmi” and “Cholestrol normal” as shown below. This is mostly consistent with the top 5 most important variables as per Decision Tree, with the only difference being that “Low Blood Pressure” is in the top 5 instead of “bmi” in case of Decision Tree, whereas the vice versa is true for Gradient Boosting Machine.

```r
library(gbm)
```

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

```
## Loaded gbm 2.1.5
```

```r
varImp(gbm_tuned)
```

```
## gbm variable importance
##
##                       Overall
## High.Blood.Pressure 100.0000
## Age                  17.2772
## Cholesteroltoo high   6.4313
## bmi                   3.6047
## Cholesterolnormal     3.2442
## Weight                2.3744
## Low.Blood.Pressure    2.2933
## Height                1.1410
## Exercise              0.9054
## Smoke                 0.4171
## Glucosetoo high       0.3683
## Alcohol               0.1544
## Glucosenormal         0.1303
## Gendermale            0.0000
```

Creating Model Performance Master Table

The 2 most important performance evaluation metrics for this dataset are “ACCURACY” & “RECALL”. Since the dataset is balanced, with the number of people with disease and without disease being approximately equal, we can certainly use ‘accuracy’ to measure the model performance.

The next most important metric is “RECALL”. “Recall” is also extremely important when doing disease prediction, because we want to minimize false negatives, but without compromising much on ACCURACY. False Negatives are cases when the patient actually has the disease but our ML model falsely classifies the patient as not having the disease, which is highly undesireable and such cases have to be minimized by increasing our RECALL.

If we simply reduce the probability threshold to a very low value and classify everything as Positive, our False Negatives will be 0 and our RECALL may reach 100%, but then by doing so our ACCURACY will be very low, which will make it a bad model. Hence Recall has to be maximized without compromising on ACCURACY.

In our below master table, we can see that Accuracies of most of the models are almost inline with only a minor difference. Hence we will sort the table in decreasing order of their “RECALL” scores.

We can see the models with the highest RECALL score and also high AccURACY scores. We can observe that the Deep Learning Neural Network models (ANN0, ANN1, ANN2) and the Ensemble Models (GBM, RF) are one of the top performing models.

```
Algorithm <- c('KNN','Naive Bayes Classifier','Random Forest','Gradient Boosting Machine','Linea
r SVM','Non Linear SVM','Logistic Regression','Decision Tree','ANN0','ANN1','ANN2')
Hyperparameters <- c('K','Laplace','ntree,.mtry','depth,trees,shrinkage,minobs','C','sigma,C','a
lpha,lambda','minsplit,maxdepth,minbucket','batchsize,epochs','batchsize,epochs,layerdropout,uni
ts','batchsize,epochs,layerdropout,units')
Accuracy <- c(0.7284,0.7135,0.7341,0.7327,0.7288,0.7384,Accuracy_Logistic,Accuracy_DT,ann0_accur
acy$.estimate[1],ann1_accuracy$.estimate[1],ann2_accuracy$.estimate[1])
Recall <- c(0.67914,0.6133,0.6871,0.7001,0.6117,0.6572,Recall_Logistic,Recall_DT,ann0_recall$.es
timate[1],ann1_recall$.estimate[1],ann2_recall$.estimate[1])
F1_score <- c(0.7143,0.6834,0.7226,0.7253,0.6929,0.7152,F1_score_Logistic,F1_score_DT,ann0_f1$.e
stimate[1],ann1_f1$.estimate[1],ann2_f1$.estimate[1])
AUCs <- c(0.7918,0.7813,0.7913,0.8006,NA,NA,auc_logistic,auc(roc_curve),ann0_auc$.estimate[1],an
n1_auc$.estimate[1],ann2_auc$.estimate[1])
ApproxRun_Time <- c('30 minutes','5 minutes','2 hours','1 hour','2 hours','2 hours','10 minutes'
,'30 minutes','10 minutes','10 minutes','10 minutes')

master_table_df <- data.frame(Algorithm,Hyperparameters,Accuracy,Recall,ApproxRun_Time)

master_table_df <- master_table_df[order(-Recalls,-Accuracies),]



master_table_df
```

```
##                    Algorithm                    Hyperparameters  Accuracy    Recall ApproxRu
n_Time
## 8              Decision Tree         minsplit,maxdepth,minbucket 0.7258317 0.7249966     30 m
inutes
## 11                      ANN2 batchsize,epochs,layerdropout,units 0.7398714 0.7148971     10 m
inutes
## 10                      ANN1 batchsize,epochs,layerdropout,units 0.7333401 0.7059303     10 m
inutes
## 4  Gradient Boosting Machine       depth,trees,shrinkage,minobs 0.7327000 0.7001000         
1 hour
## 3              Random Forest                         ntree,.mtry 0.7341000 0.6871000        2
hours
## 1                        KNN                                   K 0.7284000 0.6791400     30 m
inutes
## 9                       ANN0                    batchsize,epochs 0.7272171 0.6788262     10 m
inutes
## 7        Logistic Regression                        alpha,lambda 0.7273964 0.6640359     10 m
inutes
## 6             Non Linear SVM                             sigma,C 0.7384000 0.6572000        2
hours
## 2     Naive Bayes Classifier                             Laplace 0.7135000 0.6133000      5 m
inutes
## 5                 Linear SVM                                   C 0.7288000 0.6117000        2
hours
```

END of Combination & Comparison of multiple algorithms

SECTION 4: 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("D:/SYR ADS/Sem 2/IST_707_Data_Analytics/HW4")

getwd
```

```
## function ()
## .Internal(getwd())
## <bytecode: 0x000002478470eac8>
## <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 LOGISTIC REGRESSION:

For LOGISTIC REGRESSION, we require all the categorical variables to be converted to dummies 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_testing_df <- disease_prediction_testing
disease_prediction_testing_df<-fastDummies::dummy_cols(disease_prediction_testing_df,select_colu
mns=c('Gender','Cholesterol','Glucose'))

disease_prediction_testing_df$bmi <- disease_prediction_testing_df$Weight/((disease_prediction_t
esting_df$Height/100)*(disease_prediction_testing_df$Height/100))
disease_prediction_testing_df$Age <- normalize(disease_prediction_testing_df$Age)
disease_prediction_testing_df$Height <- normalize(disease_prediction_testing_df$Height)
disease_prediction_testing_df$Weight <- normalize(disease_prediction_testing_df$Weight)
disease_prediction_testing_df$bmi <- normalize(disease_prediction_testing_df$bmi)
disease_prediction_testing_df$Low.Blood.Pressure <- normalize(disease_prediction_testing_df$Low.
Blood.Pressure)
disease_prediction_testing_df$High.Blood.Pressure <- normalize(disease_prediction_testing_df$Hig
h.Blood.Pressure)

i <- sapply(disease_prediction_testing_df, is.integer)
disease_prediction_testing_df[,i] <- lapply(disease_prediction_testing_df[i], as.factor)

disease_prediction_testing_df$Gender <- NULL
disease_prediction_testing_df$Cholesterol <- NULL
disease_prediction_testing_df$Glucose <- NULL
disease_prediction_testing_df$age_groups <- NULL
disease_prediction_testing_df$ID <- NULL
disease_prediction_testing_df$bmi_groups <- NULL
disease_prediction_testing_df$Gender_male <- NULL
disease_prediction_testing_df$ID <- NULL
```

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

```
LR <- predict(tuned_model_logistic, newdata = disease_prediction_testing_df)
```

PREPARING DATA for ANN

step_center() to mean-center the data & step_scale() to scale the data
The last step is to prepare the recipe with the prep() function

```
library(recipes)
```

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

```
##
## Attaching package: 'recipes'
```

```
## The following object is masked from 'package:stringr':
##
##     fixed
```

```
## The following object is masked from 'package:stats':
##
##     step
```

```
disease_rec_obj_test <- recipes::recipe(x = disease_prediction_testing_df)%>%
  recipes::step_center(Age,Height,Weight,High.Blood.Pressure,Low.Blood.Pressure,bmi)%>%
  recipes::step_scale(Age,Height,Weight,High.Blood.Pressure,Low.Blood.Pressure,bmi)%>%
  recipes::prep(data = disease_prediction_testing_df)
```

Applying the "recipe" to data set with the bake() function which processes the data following our recipe steps

```
disease_prediction_testing_baked_df <- recipes::bake(disease_rec_obj_test, new_data = disease_pr
ediction_testing_df)

i <- sapply(disease_prediction_testing_baked_df, is.factor)
disease_prediction_testing_baked_df[,i] <- lapply(disease_prediction_testing_baked_df[i], as.cha
racter)

j <- sapply(disease_prediction_testing_baked_df, is.character)
disease_prediction_testing_baked_df[,j] <- lapply(disease_prediction_testing_baked_df[j], as.int
eger)
```

2. Obtaining Predicted values for disease on TEST data using the final ANN0, ANN1 & ANN2 models

```
library(yardstick)
library(keras)
library(tensorflow)
ANN2 <- predict_classes(object = dl_model_keras_ann2, x = as.matrix(disease_prediction_testing_b
aked_df)) %>% as.vector()

ANN1 <- predict_classes(object = dl_model_keras_ann1, x = as.matrix(disease_prediction_testing_b
aked_df)) %>% as.vector()

ANN0 <- predict_classes(object = dl_model_keras_ann0, x = as.matrix(disease_prediction_testing_b
aked_df)) %>% as.vector()
```

Preparing Data for Decision Tree

```
disease_prediction_testing_dt <- disease_prediction_testing
disease_prediction_testing_dt$bmi <- disease_prediction_testing_dt$Weight/((disease_prediction_t
esting_dt$Height/100)*(disease_prediction_testing_dt$Height/100))
disease_prediction_testing_dt$ID <- NULL
disease_prediction_testing_dt$Smoke <- as.factor(disease_prediction_testing_dt$Smoke)
disease_prediction_testing_dt$Alcohol <- as.factor(disease_prediction_testing_dt$Alcohol)
disease_prediction_testing_dt$Exercise <- as.factor(disease_prediction_testing_dt$Exercise)
disease_prediction_testing_dt$Height <- as.numeric(disease_prediction_testing_dt$Height)
```

3. Obtaining Predicted values for disease on TEST data using the DECISION TREE model

```
DT <- predict(dt_model_disease_final_tuned, newdata = disease_prediction_testing_dt, na.action =
na.omit, type = "raw")
```

Writing the final predictions to a CSV

```
disease_prediction_testing$DT <- DT
disease_prediction_testing$LR <- LR
disease_prediction_testing$ANN0 <- ANN0
disease_prediction_testing$ANN1 <- ANN1
disease_prediction_testing$ANN2 <- ANN2

final_predictions <- disease_prediction_testing[,c('ID','DT','LR','ANN0','ANN1','ANN2')]
write.csv(final_predictions,"HW_04_Kumar_Bhavish_Predictions.csv", row.names = FALSE)
```

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 5 algorithms we can observe that the Deep Learning Models like Artificial Neural Networks with 0 or more hidden layers are one of the best performing models. For ANN, we can tune the batch size and epochs along with number of nodes and dropout rate in order to produce low bias and low variance estimates such that the model performs well not only on training data, but also on validation data.