Disease Prediction using Machine Learning

Juilee

4/25/2020

Introduction: The objective of the assignment is to predict whether or not a patient has a certain unspecified disease.It is a binary classification problem.Multiple machine learning algorithms,including logistic regression, multiple artificial neural networks,SVM Linear, Random Forest and Gradient Boosting Classifier to build a disease diagnosis model will be used.

Loading packages

```
library(tidyverse)
library(ggplot2)
library(caret)
library(caretEnsemble)
library(rpart)
library(randomForest)
library(e1071)
library(klaR)
library(naivebayes)
library(doParallel)
library(Amelia)
library(pROC)
library(gridExtra)
library(grid)
library(resample)
library(ggpubr)
library(raster)
library(forcats)
library(yardstick)
library(recipes)
```

Loading training dataset.

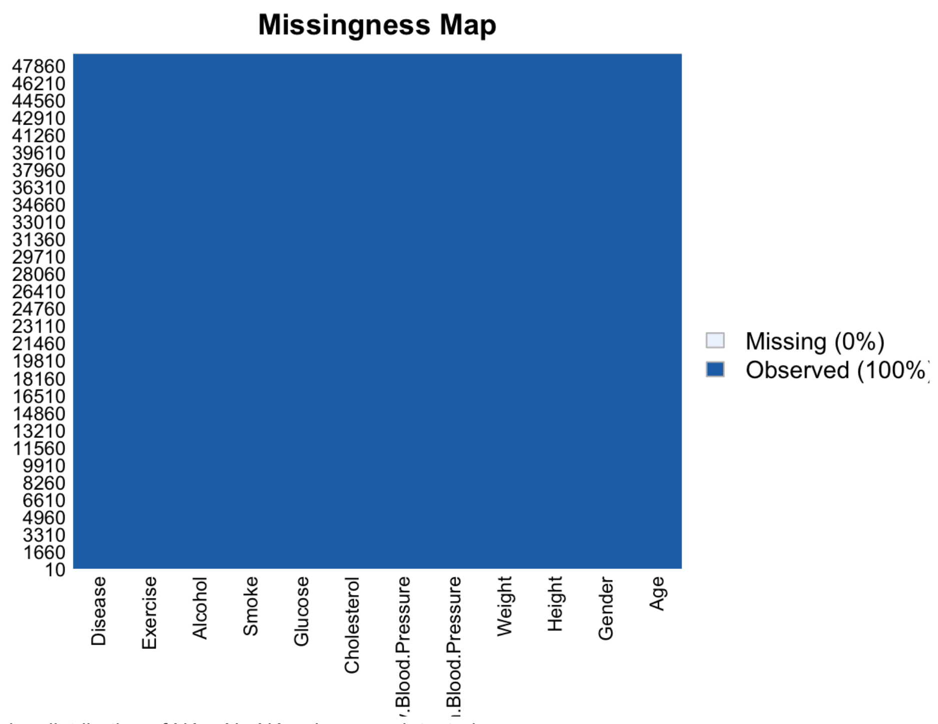
```
d_train<- read.csv("/Users/juilee81/Desktop/DA/DA_HW_03/Disease\ Prediction\ Training.csv",header=TRUE)
#View(d_train)
```

Checking for missing values column-wise and visualize the data:

```
colSums(is.na(d_train))
```

```
##                 Age              Gender              Height 
##                   0                   0                   0 
##              Weight High.Blood.Pressure  Low.Blood.Pressure 
##                   0                   0                   0 
##         Cholesterol             Glucose               Smoke 
##                   0                   0                   0 
##             Alcohol            Exercise             Disease 
##                   0                   0                   0
```

```
#visualize the missing data
missmap(d_train)
```



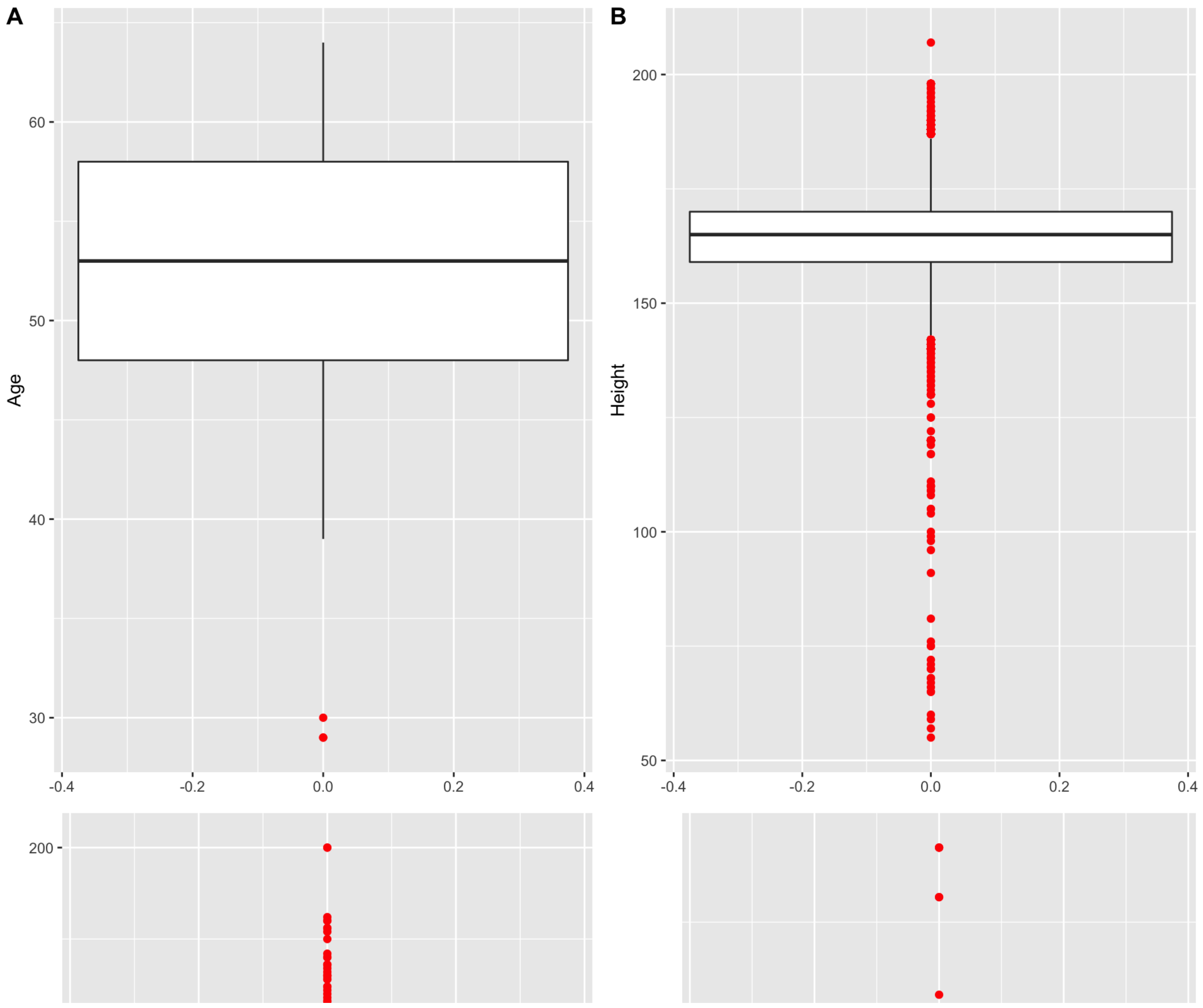
The plot and output shows columnwise distribution of NAs. No NA values are detected.

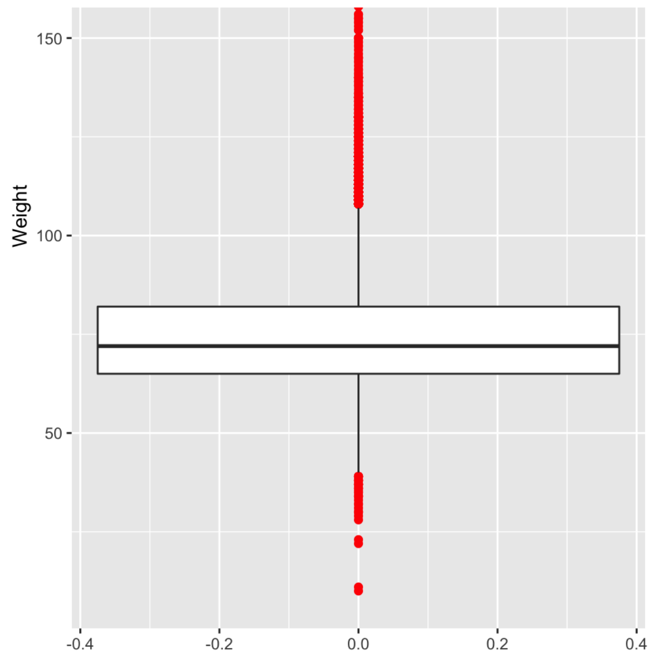
Outlier detection

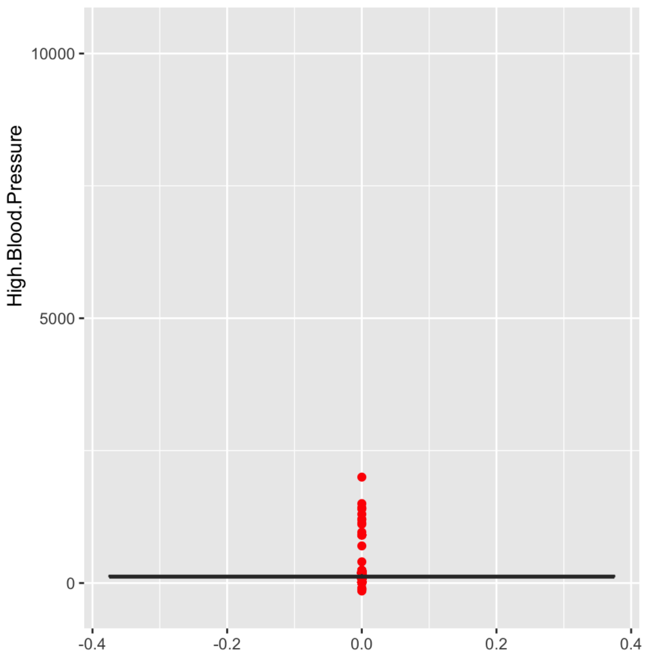
```
summary(d_train)
```

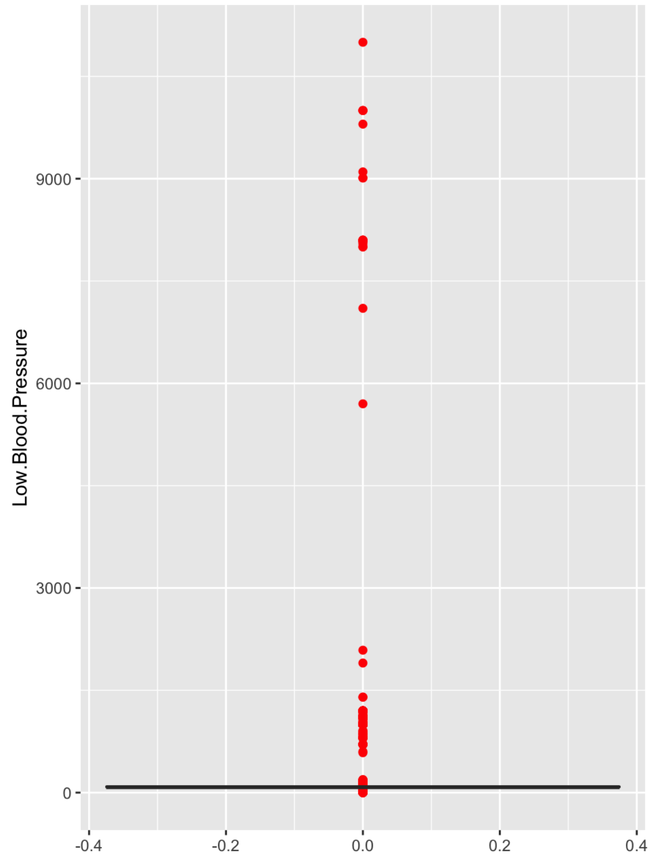
```
##       Age            Gender           Height          Weight      
##  Min.   :29.00   female:31863   Min.   : 55.0   Min.   : 10.00  
##  1st Qu.:48.00   male  :17137   1st Qu.:159.0   1st Qu.: 65.00  
##  Median :53.00                  Median :165.0   Median : 72.00  
##  Mean   :52.85     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

```
bxp_Age<- ggplot(d_train, aes(y=Age))+
              geom_boxplot(outlier.colour="red",
              outlier.shape=16,
              outlier.size=2, notch=FALSE)

bxp_Height<- ggplot(d_train, aes(y=Height))+
              geom_boxplot(outlier.colour="red",
              outlier.shape=16,
              outlier.size=2, notch=FALSE)

bxp_Weight<- ggplot(d_train, aes(y=Weight))+
              geom_boxplot(outlier.colour="red",
              outlier.shape=16,
              outlier.size=2, notch=FALSE)

bxp_High.Blood.Pressure<- ggplot(d_train, aes(y=High.Blood.Pressure))+
              geom_boxplot(outlier.colour="red",
              outlier.shape=16,
              outlier.size=2, notch=FALSE)

bxp_Low.Blood.Pressure<- ggplot(d_train, aes(y=Low.Blood.Pressure))+
              geom_boxplot(outlier.colour="red",
              outlier.shape=16,
              outlier.size=2, notch=FALSE)



figure <- ggarrange(bxp_Age,bxp_Height,bxp_Weight,bxp_High.Blood.Pressure,bxp_Low.Blood.Pressure,
                    labels = c("A", "B"),
                    ncol = 2, nrow = 3)
figure
```









Outliers are detected in the variables of High and low blood Pressure, Weight and height.This can be seen from the summary and boxplots.

Treating Outliers:

```
#install.packages("raster")
out<- function(q1,q3) {
  l<- q1-1.5*(q3-q1)
  u<- q3+1.5*(q3-q1)
  print(paste(l,u))
}
#Clamping using inter-quartile range
out(159,170)
```

```
## [1] "142.5 186.5"
```

```
d_train$Height<-clamp(d_train$Height, lower=142.5, upper=186.5)
out(65,82)
```

```
## [1] "39.5 107.5"
```

```
d_train$Weight<-clamp(d_train$Weight, lower=39.5, upper=107.5)
out(80,90)#low blood pressure
```

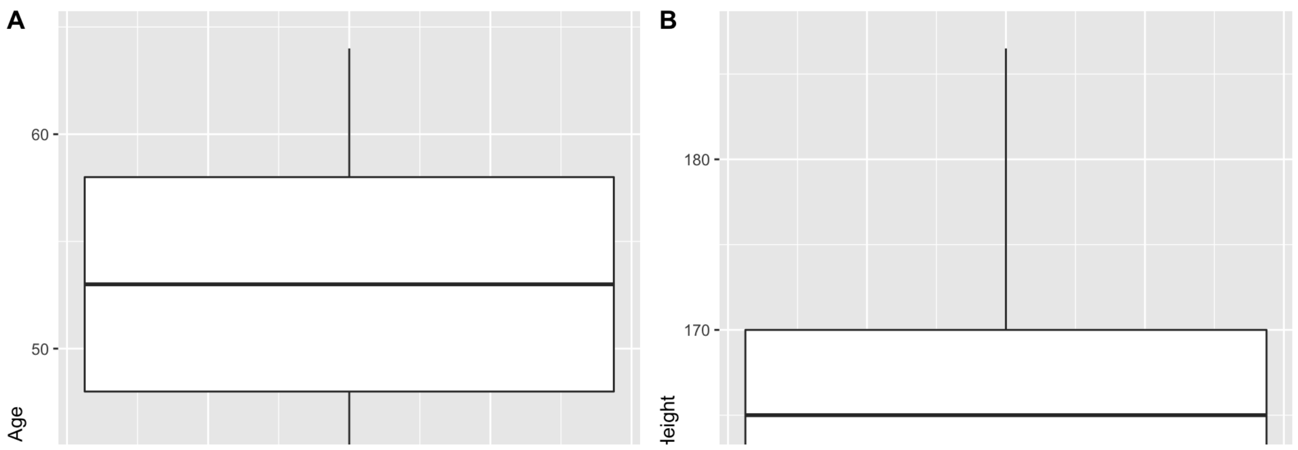
```
## [1] "65 105"
```

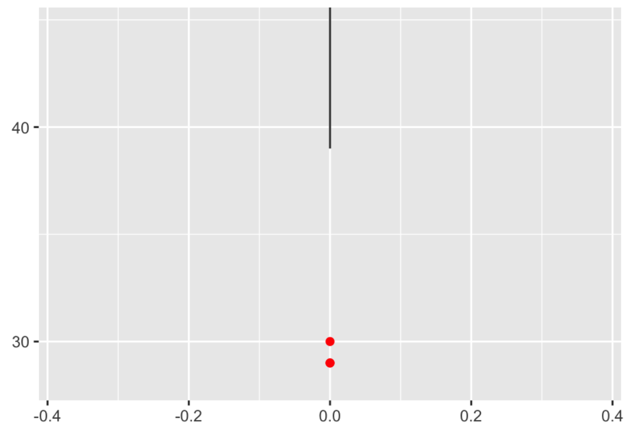
```
out(120,140)#High blood pressure
```

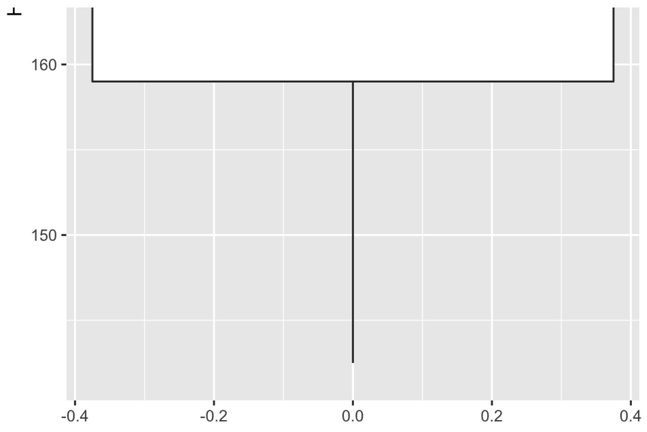
```
## [1] "90 170"
```

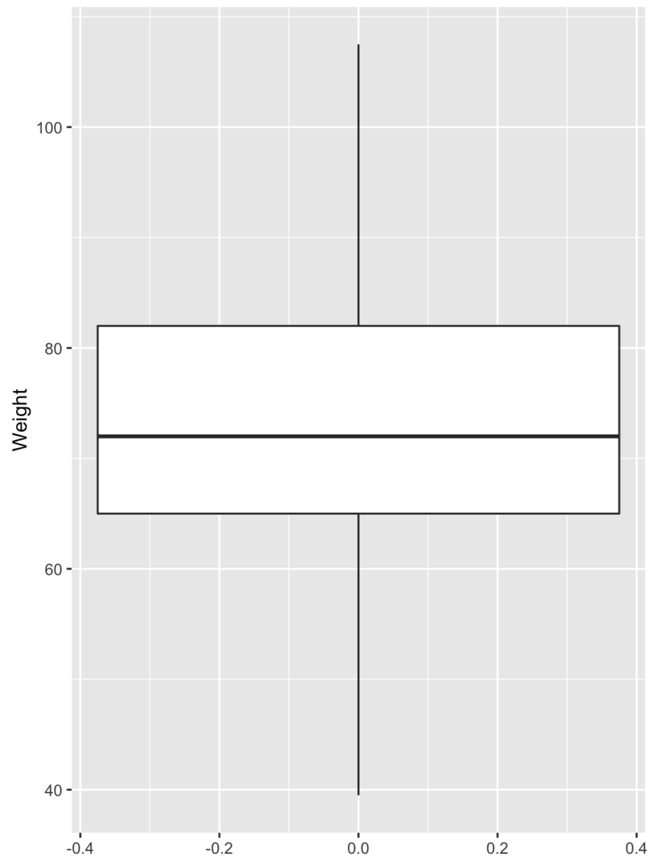
```
#Since low blood pressure cannot be greater than high blood pressure, therefore swapping lower value of High
blood pressure and upper value of low blood pressure.
d_train$High.Blood.Pressure<-clamp(d_train$High.Blood.Pressure, lower=105, upper=170)
d_train$Low.Blood.Pressure<-clamp(d_train$Low.Blood.Pressure, lower=65, upper=90)
```

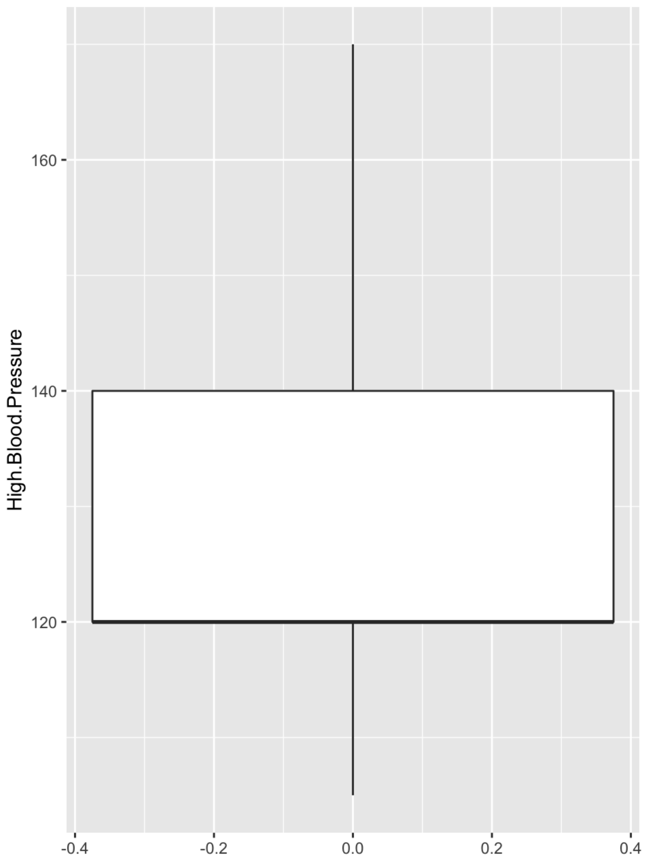
```
bxp_Age<- ggplot(d_train, aes(y=Age))+
              geom_boxplot(outlier.colour="red",
              outlier.shape=16,
              outlier.size=2, notch=FALSE)

bxp_Height<- ggplot(d_train, aes(y=Height))+
              geom_boxplot(outlier.colour="red",
              outlier.shape=16,
              outlier.size=2, notch=FALSE)

bxp_Weight<- ggplot(d_train, aes(y=Weight))+
              geom_boxplot(outlier.colour="red",
              outlier.shape=16,
              outlier.size=2, notch=FALSE)

bxp_High.Blood.Pressure<- ggplot(d_train, aes(y=High.Blood.Pressure))+
              geom_boxplot(outlier.colour="red",
              outlier.shape=16,
              outlier.size=2, notch=FALSE)

bxp_Low.Blood.Pressure<- ggplot(d_train, aes(y=Low.Blood.Pressure))+
              geom_boxplot(outlier.colour="red",
              outlier.shape=16,
              outlier.size=2, notch=FALSE)



figure <- ggarrange(bxp_Age,bxp_Height,bxp_Weight,bxp_High.Blood.Pressure,bxp_Low.Blood.Pressure,
                    labels = c("A", "B"),
                    ncol = 2, nrow = 3)
figure
```

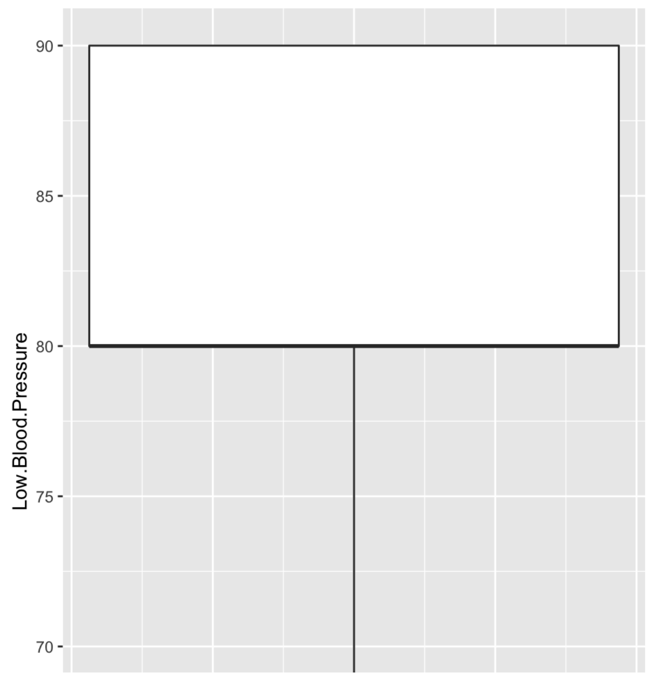


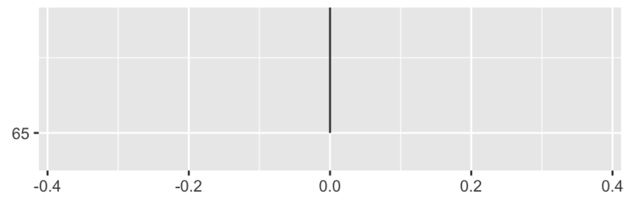

40
30
-0.4
-0.2
0.0
0.2
0.4


160
150
-0.4
-0.2
0.0
0.2
0.4


Weight
100
80
60
40
-0.4
-0.2
0.0
0.2
0.4


High.Blood.Pressure
160
140
120
-0.4
-0.2
0.0
0.2
0.4


Low.Blood.Pressure
90
85
80
75
70



All outliers are treated.

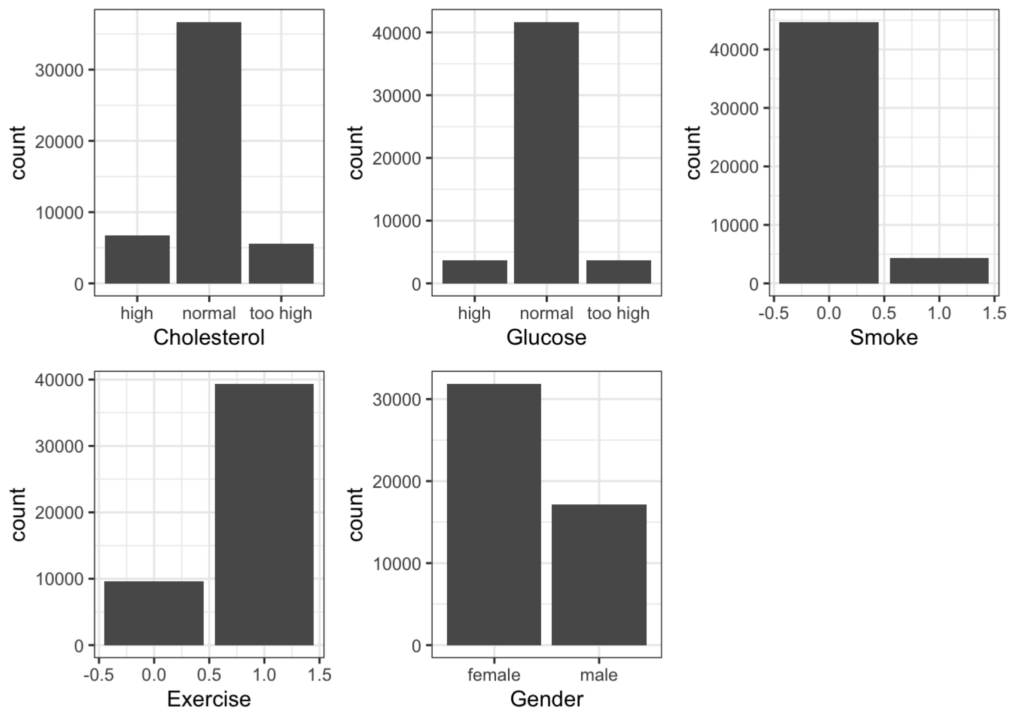
Normalising the numeric data

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

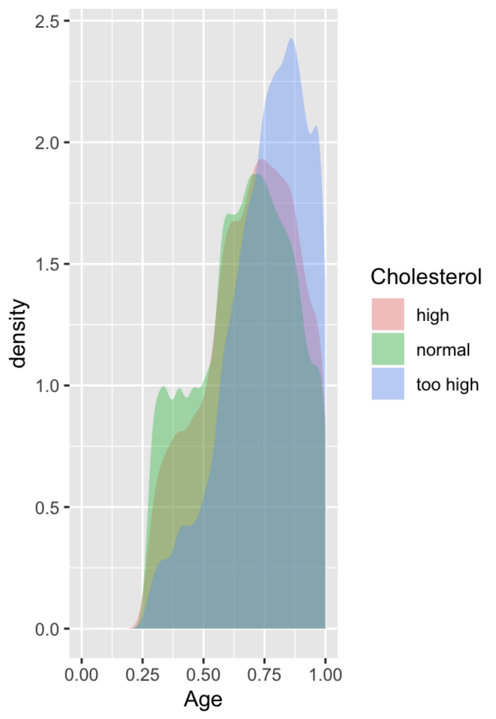
```
d_train$Age <- normalize(d_train$Age)
d_train$Height <- normalize(d_train$Height)
d_train$Weight <- normalize(d_train$Weight)
d_train$Low.Blood.Pressure <- normalize(d_train$Low.Blood.Pressure)
d_train$High.Blood.Pressure  <- normalize(d_train$High.Blood.Pressure)
```

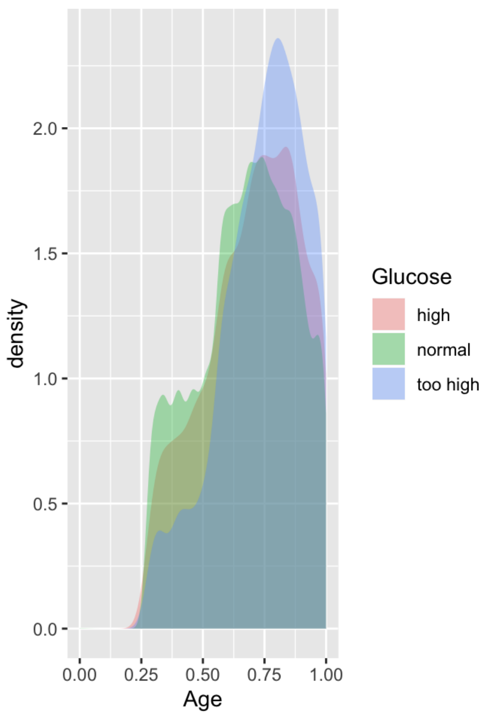
Exploratory Data Analysis:

```
#Univariate Analysis of variables
p1<-ggplot(d_train, aes(x=Cholesterol,  fill=Disease)) +
  geom_bar(stat="count",position="dodge")+theme_bw()
p2<-ggplot(d_train, aes(x=Glucose,  fill=Disease)) +
  geom_bar(stat="count",position="dodge")+theme_bw()
p3<-ggplot(d_train, aes(x=Smoke,  fill=Disease)) +
  geom_bar(stat="count",position="dodge")+theme_bw()
p4<-ggplot(d_train, aes(x=Exercise,  fill=Disease)) +
  geom_bar(stat="count",position="dodge")+theme_bw()
p5<-ggplot(d_train, aes(x=Gender,  fill=Disease)) +
  geom_bar(stat="count",position="dodge")+theme_bw()
grid.arrange(p1,p2,p3,p4,p5,ncol=3, nrow= 2)
```



```
#Bi-variate Analysis
t1<-ggplot(d_train,aes(x=Age,fill=Cholesterol))+geom_density(col=NA,alpha=0.35)
t2<-ggplot(d_train,aes(x=Age,fill=Glucose))+geom_density(col=NA,alpha=0.35)

grid.arrange(t1,t2,ncol=2, nrow= 1)
```





The Age group that is between 50-60 years old have both “too high” Glucose and Cholesterol.

Creating dummy variables of all the categorical variables

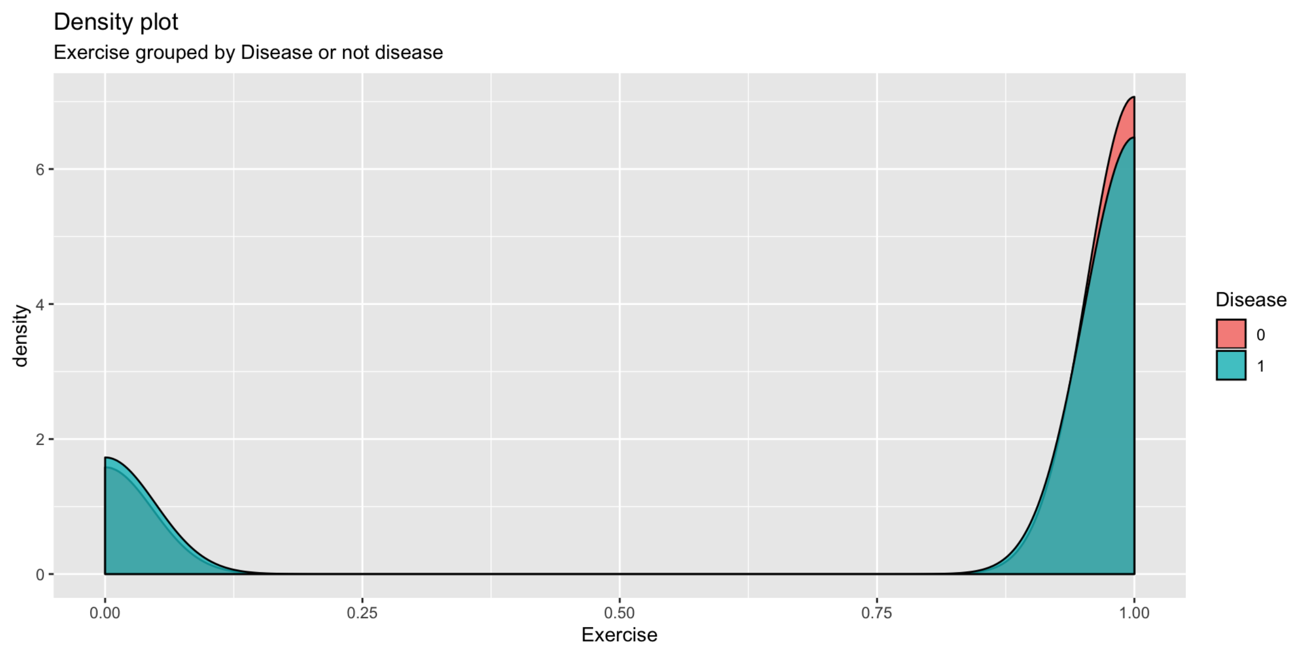
```
library(fastDummies)
d_dummies <- fastDummies::dummy_cols(d_train,select_columns=c('Gender','Cholesterol','Glucose'))
d_dummies <- d_dummies[,c(-2,-7,-8)]
```

```
d_dummies$Disease <-factor(d_dummies$Disease,labels = c("False", "True"))
```

This is necessary because our output will be in the form of 2 classes, True or False. Where true will denote that a patient has a disease and false denotes that a person is disease free.

```
#Renaming the columns
colnames(d_dummies)[14] <- "Cholesterol_too_high"
colnames(d_dummies)[17] <- "Glucose_too_high"
```

```
g <- ggplot(d_train, aes(Exercise))
g + geom_density(aes(fill=factor(Disease)), alpha=0.8) +
    labs(title="Density plot",
         subtitle="Exercise grouped by Disease or not disease",
         x="Exercise",
         fill="Disease")
```



```
#pairs(Disease~., data=d_dummies[,1:9], col=d_dummies$Disease)
```

IF exercised,the number of people not getting disease is higher than people getting disease(orange part at the top).

Split data into training and test data sets

```
#Building a model
indxTrain <- createDataPartition(y = d_dummies$Disease,p = 0.75,list = FALSE)
training <- d_dummies[indxTrain,]
testing <- d_dummies[-indxTrain,]
prop.table(table(d_dummies$Disease)) * 100
```

```
##
##    False     True
## 50.00408 49.99592
```

```
testing$Disease <- factor(testing$Disease,labels = c("False", "True"))
```

BUILDING MODELS Logistic Regression

```
set.seed(123)
model_lr <- train(Disease ~ ., data = training,
                   method = "glm", family = "binomial")

#print(model_lr)
```

```
predict_lr <- predict(model_lr, newdata = testing)
confusionMatrix(predict_lr, testing$Disease)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  4779 1984
##      True   1346 4140
##
##                Accuracy : 0.7281
##                  95% CI : (0.7202, 0.736)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4563
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7802
##             Specificity : 0.6760
##          Pos Pred Value : 0.7066
##          Neg Pred Value : 0.7546
##              Prevalence : 0.5000
##          Detection Rate : 0.3902
##    Detection Prevalence : 0.5521
##       Balanced Accuracy : 0.7281
##
##        'Positive' Class : False
##
```

Regularization **Elastic Net Regression** Elastic net regression combines the properties of ridge and lasso regression. It works by penalizing the model using both the 1l2-norm1 and the 1l1-norm1. The first line of code creates the training control object train_cont which specifies how the repeated cross validation will take place. The second line builds the elastic regression model in which a range of possible alpha and lambda values are tested and their optimum value is selected. The argument tuneLength specifies that 10 different combinations of values for alpha and lambda that are to be tested. alpha and lambda values:Lambda is a term that controls the learning rate. In other words, how much change do you want the model to make during each iteration of learning. If lambda value is too high, model will be simple, but we run the risk of underfitting our data. Model won't learn enough about the training data to make useful predictions. If lambda value is too low, model will be more complex, and we run the risk of overfitting our data. Model will learn too much about the particularities of the training data, and won't be able to generalize to new data

```r
# Set training control
train_cont <- trainControl(method = "repeatedcv",
                              number = 10,
                              repeats = 5,
                              search = "random",
                              verboseIter = FALSE)
set.seed(123)
# Train the model
elastic_reg <- train(Disease ~ .,
                           data = training,
                           method = "glmnet",
                           preProcess = c("center", "scale"),
                           tuneLength = 10,
                           trControl = train_cont)


# Best tuning parameter
elastic_reg$bestTune
```

```
##       alpha      lambda
## 9 0.9404673 0.002468792
```

```r
predict_lr_e <- predict(elastic_reg, newdata = testing)
confusionMatrix(predict_lr_e, testing$Disease)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  4795 1995
##      True   1330 4129
##
##                Accuracy : 0.7285
##                  95% CI : (0.7206, 0.7364)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4571
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7829
##             Specificity : 0.6742
##          Pos Pred Value : 0.7062
##          Neg Pred Value : 0.7564
##              Prevalence : 0.5000
##          Detection Rate : 0.3915
##    Detection Prevalence : 0.5543
##       Balanced Accuracy : 0.7285
##
##        'Positive' Class : False
##
```

After performing elastic net regression, the accuracy has slightly increased.

```
base_tune_lr<- c("Base","Tuned")
Accuracy_of_lr<- c(73.48,73.48)
a_lr_models<-data.frame(base_tune_lr,Accuracy_of_lr)
a_lr_models
```

```
##   base_tune_lr Accuracy_of_lr
## 1         Base          73.48
## 2        Tuned          73.48
```

ANN0,1,2

```
rec_obj <- recipe(Disease ~ ., data = training)
```

```
set.seed(123)
x_train_tbl <- bake(rec_obj, new_data = training)%>%dplyr::select(-Disease)
x_test_tbl <- bake(rec_obj, new_data = testing)%>% dplyr::select(-Disease)
y_train_vec <- ifelse(pull(training, Disease) == "True", 1, 0)
y_test_vec <-  ifelse(pull(testing, Disease) == "True", 1, 0)
#str(x_train_tbl)
```

0 hidden layer

```
library(keras)
set.seed(123)
model_keras0 <- keras_model_sequential()
model_keras0 %>%
layer_dense(units=1, kernel_initializer="uniform",activation = "sigmoid") %>%
compile(optimizer = "adam",loss = "binary_crossentropy",metrics = c("accuracy"))

history0 <- model_keras0 %>% fit(
  as.matrix(x_train_tbl),
  y_train_vec,
  batch_size = 32,
  epochs = 30,
  validation_split = 0.3)
#plot(history0)
```

```
yhat<- function(obj){
set.seed(123)
library(forcats)
yhat_keras_class_vec <- predict_classes(object = obj, x = as.matrix(x_test_tbl)) %>%as.vector()
yhat_keras_prob_vec <- predict_proba(object = obj, x = as.matrix(x_test_tbl)) %>%as.vector()
estimates_keras_tbl <- tibble(truth = as.factor(y_test_vec)%>% fct_recode(true="1",false="0"),
                              estimate = as.factor(yhat_keras_class_vec)%>% fct_recode(true="1",false="0"),
                              class_prob = yhat_keras_prob_vec)

library(yardstick)
options(yardstick.event_first = F)
estimates_keras_tbl %>% conf_mat(truth, estimate)
estimates_keras_tbl %>% metrics(truth, estimate)
estimates_keras_tbl %>% roc_auc(truth, class_prob)
tab<-data.frame(estimates_keras_tbl %>% metrics(truth, estimate),estimates_keras_tbl %>% roc_auc(truth, clas
s_prob))
tab
}
yhat(model_keras0)
```

```
##    .metric .estimator .estimate .metric.1 .estimator.1 .estimate.1
## 1 accuracy     binary 0.7268348   roc_auc       binary   0.7896461
## 2      kap     binary 0.4536647   roc_auc       binary   0.7896461
```

Hyperparameter tuning:

1. batch_size: Minibatch Gradient Descent:Batch size is set to more than one and less than the total number of examples in the training dataset.Large values give a learning process that converges slowly with accurate estimates of the error gradient.The problem with large batch sizes are “sharp” local minima which leads to overfitting.
2. Epoch:One Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE.As the number of epochs increases, more number of times the weight are changed in the neural network and the curve goes from underfitting to optimal to overfitting curve.

3.A common way to mitigate overfitting is to put constraints on the complexity of a network by forcing its weights to only take on small values, which makes the distribution of weight values more “regular”. This is called “weight regularization”, and it is done by adding to the loss function of the network a cost associated with having large weights. This cost comes in two flavors: L1 regularization, where the cost added is proportional to the absolute value of the weights coefficients (i.e. to what is called the “L1 norm” of the weights). L2 regularization, where the cost added is proportional to the square of the value of the weights coefficients (i.e. to what is called the “L2 norm” of the weights). L2 regularization is also called weight decay in the context of neural networks. Don’t let the different name confuse you: weight decay is mathematically the exact same as L2 regularization.

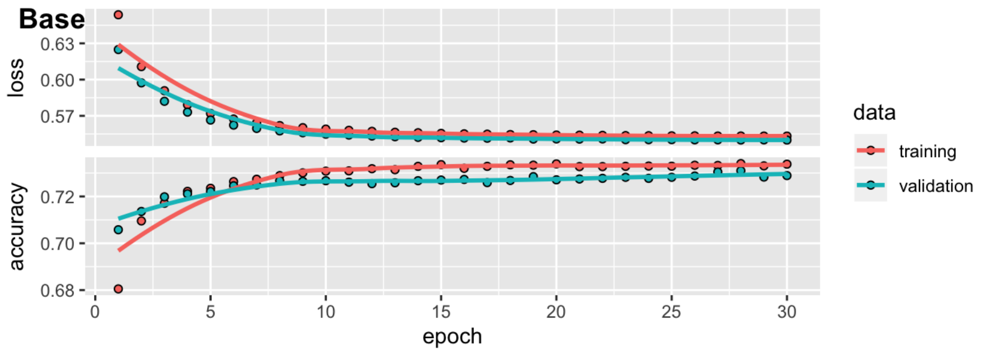
4. validation_split:Float between 0 and 1. The model sets apart the last fraction of the x and y data provided and use it as a validation set.
5. No of hidden nodes

```
#hyperparameter tuning 1
set.seed(123)
model_keras0_tune <- keras_model_sequential()
model_keras0_tune %>%
layer_dense(units=1, kernel_initializer="uniform",activation = "sigmoid",kernel_regularizer = regularizer_l2
(l = 0.001)) %>%
compile(optimizer = "adam",loss = "binary_crossentropy",metrics = c("accuracy"))
```

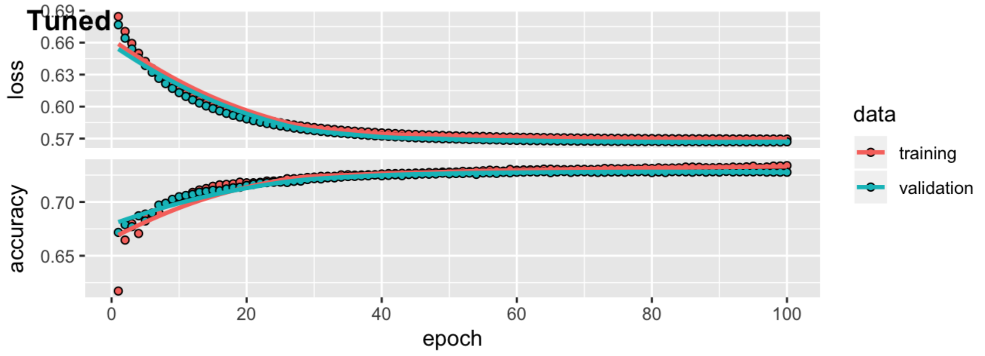
```
history0_tune <- model_keras0_tune %>% fit(
  as.matrix(x_train_tbl),
  y_train_vec,
  batch_size = 32,
  epochs = 10,
  validation_split = 0.3,
  verbose=0)


#hyperparameter tuning 2
set.seed(123)
model_keras0_tune <- keras_model_sequential()
model_keras0_tune %>%
layer_dense(units=1, kernel_initializer="uniform",activation = "sigmoid",kernel_regularizer = regularizer_l2
(l = 0.001)) %>%
compile(optimizer = "adam",loss = "binary_crossentropy",metrics = c("accuracy"))
history0_tune  <- model_keras0_tune %>% fit(
  as.matrix(x_train_tbl),
  y_train_vec,
  batch_size = 64,
  epochs = 50,
  validation_split = 0.25,
  verbose=0)


#hyperparameter tuning 3
set.seed(123)
model_keras0_tune <- keras_model_sequential()
model_keras0_tune %>%
layer_dense(units=1, kernel_initializer="uniform",activation = "sigmoid",kernel_regularizer = regularizer_l2
(l = 0.001)) %>%
compile(optimizer = "adam",loss = "binary_crossentropy",metrics = c("accuracy"))
history0_tune  <- model_keras0_tune %>% fit(
  as.matrix(x_train_tbl),
  y_train_vec,
  batch_size = 256,
  epochs = 100,
  validation_split = 0.5,
  verbose=0)
yhat(model_keras0_tune)
```

```
##    .metric .estimator .estimate .metric.1 .estimator.1 .estimate.1
## 1 accuracy     binary 0.7283044   roc_auc       binary   0.7882267
## 2      kap     binary 0.4566031   roc_auc       binary   0.7882267
```

```
tv_plot0 <- ggarrange(plot(history0),plot(history0_tune),
                    labels = c("Base", "Tuned"),
                    ncol = 1, nrow = 2)
tv_plot0
```



Training loss and validation



loss both are reducing, it proves that our data is not overfitting. Since training and validation accuracy is almost equal this model is good.

```
base_tune0<- c("Base","Tuned")
Accuracy_of_ANN0<- c(73.37,73.22)
a0_models<-data.frame(base_tune0,Accuracy_of_ANN0)
a0_models
```

```
##   base_tune0 Accuracy_of_ANN0
## 1       Base            73.37
## 2      Tuned            73.22
```

Since slight decrease in accuracy of the validation data is normal. **1 hidden layer Base**

```
library(keras)
set.seed(123)
model_keras1 <- keras_model_sequential()
model_keras1 %>%
  layer_dense(units = 16,
              kernel_initializer = "uniform",
              activation = "relu",
              input_shape = ncol(x_train_tbl)) %>%
  layer_dense(units = 1,
              kernel_initializer = "uniform",
              activation = "sigmoid") %>%
  compile(optimizer = "adam",
          loss = "binary_crossentropy",
          metrics = c("accuracy"))

history1 <- model_keras1 %>% fit(
  as.matrix(x_train_tbl),
  y_train_vec,
  batch_size = 32,
  epochs = 27,
  validation_split = 0.3
)
yhat(model_keras1)
```

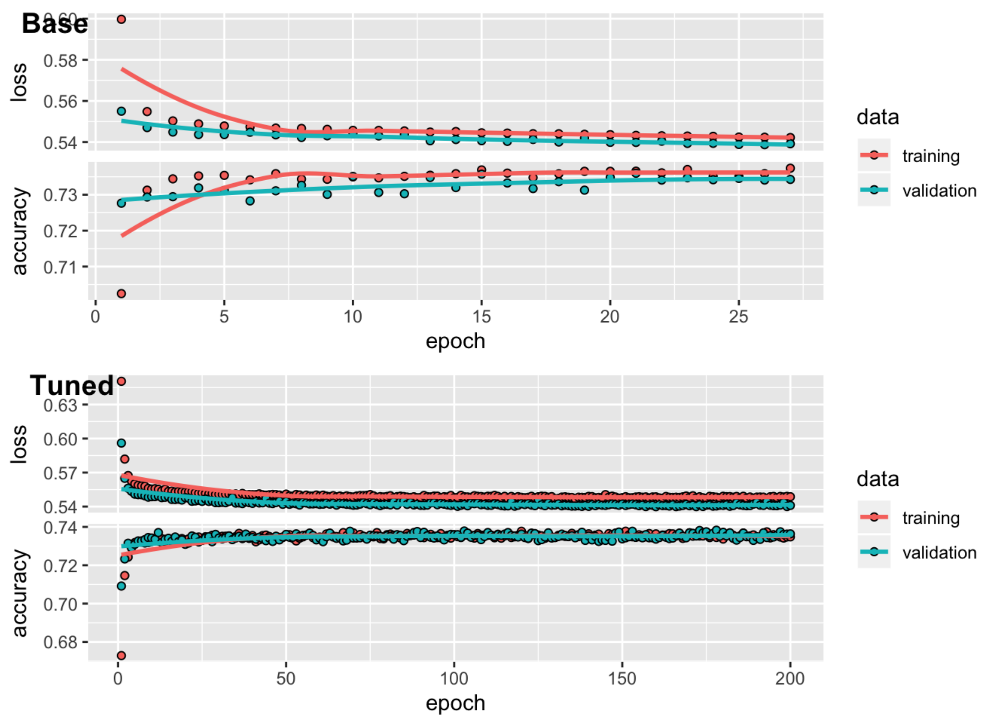
```
##    .metric .estimator .estimate .metric.1 .estimator.1 .estimate.1
## 1 accuracy     binary 0.7297739   roc_auc       binary   0.7953029
## 2      kap     binary 0.4595437   roc_auc       binary   0.7953029
```

Hyperparameter tuning 1 hidden layer

```r
#hyperparameter tuning 1
set.seed(123)
model_keras1_tune <- keras_model_sequential()
model_keras1_tune %>%
  layer_dense(units = 76,
              kernel_initializer = "uniform",
              activation = "relu",kernel_regularizer = regularizer_l2(l = 0.001),
              input_shape = ncol(x_train_tbl))%>%
  layer_dropout(0.4)%>%
  layer_dense(units = 1,
              kernel_initializer = "uniform",
              activation = "sigmoid") %>%
  compile(optimizer = "adam",
          loss = "binary_crossentropy",
          metrics = c("accuracy"))

history1_tune <- model_keras1_tune %>% fit(
  as.matrix(x_train_tbl),
  y_train_vec,
  #optimizer = optimizer_rmsprop(lr = 0.0001),
  batch_size = 32,
  epochs = 50,
  validation_split = 0.3,
  verbose=0)

#hyperparameter tuning 2
set.seed(123)
model_keras1_tune <- keras_model_sequential()
model_keras1_tune %>%
  layer_dense(units = 76,
              kernel_initializer = "uniform",
              activation = "relu",kernel_regularizer = regularizer_l2(l = 0.001),
              input_shape = ncol(x_train_tbl))%>%
  layer_dropout(0.4)%>%
  layer_dense(units = 1,
              kernel_initializer = "uniform",
              activation = "sigmoid") %>%
  compile(optimizer = "adam",
          loss = "binary_crossentropy",
          metrics = c("accuracy"))
history1_tune <- model_keras1_tune %>% fit(
  as.matrix(x_train_tbl),
  y_train_vec,
  batch_size = 64,
  epochs = 100,
  validation_split = 0.25,
  verbose=0)

#hyperparameter tuning 3
set.seed(123)
model_keras1_tune <- keras_model_sequential()
model_keras1_tune %>%
  layer_dense(units = 76,
              kernel_initializer = "uniform",
              activation = "relu",kernel_regularizer = regularizer_l2(l = 0.001),
              input_shape = ncol(x_train_tbl))%>%
  layer_dropout(0.4)%>%
  layer_dense(units = 1,
              kernel_initializer = "uniform",
              activation = "sigmoid") %>%
  compile(optimizer = "adam",
          loss = "binary_crossentropy",
          metrics = c("accuracy"))
history1_tune <- model_keras1_tune %>% fit(
  as.matrix(x_train_tbl),
  y_train_vec,
  batch_size = 256,
  epochs = 200,
  validation_split = 0.2,
  verbose=0)
yhat(model_keras1_tune)
```

```
##    .metric .estimator .estimate .metric.1 .estimator.1 .estimate.1
## 1 accuracy     binary 0.7309984   roc_auc       binary   0.7964259
## 2      kap     binary 0.4619940   roc_auc       binary   0.7964259
```

```
tv_plot1 <- ggarrange(plot(history1),plot(history1_tune),
                    labels = c("Base", "Tuned"),
                    ncol = 1, nrow = 2)
tv_plot1
```



Training loss and validation loss both are reducing, it proves that our data is not overfitting. Since training and validation accuracy is almost equal this model is better.

```
base_tune1<- c("Base","Tuned")
Accuracy_of_ANN1<- c(73.68,73.91)
a1_models<-data.frame(base_tune1,Accuracy_of_ANN1)
a1_models
```

```
##   base_tune1 Accuracy_of_ANN1
## 1       Base            73.68
## 2      Tuned            73.91
```

Accuracy has increased.

2 hidden layer

```
library(keras)
set.seed(123)
model_keras2 <- keras_model_sequential()
model_keras2 %>%
  layer_dense(units = 176,
              kernel_initializer = "uniform",
              activation = "relu",
              input_shape = ncol(x_train_tbl)) %>%
  layer_dense(units = 150,
              kernel_initializer = "uniform",
              activation = "relu") %>%
  layer_dense(units = 1,
              kernel_initializer = "uniform",
              activation = "sigmoid") %>%
  compile(optimizer = "adam",
          loss = "binary_crossentropy",
          metrics = c("accuracy"))
history2 <- model_keras2 %>% fit(
  as.matrix(x_train_tbl),
  y_train_vec,
  batch_size = 32,
  epochs = 30,
  validation_split = 0.3
)
yhat(model_keras2)
```

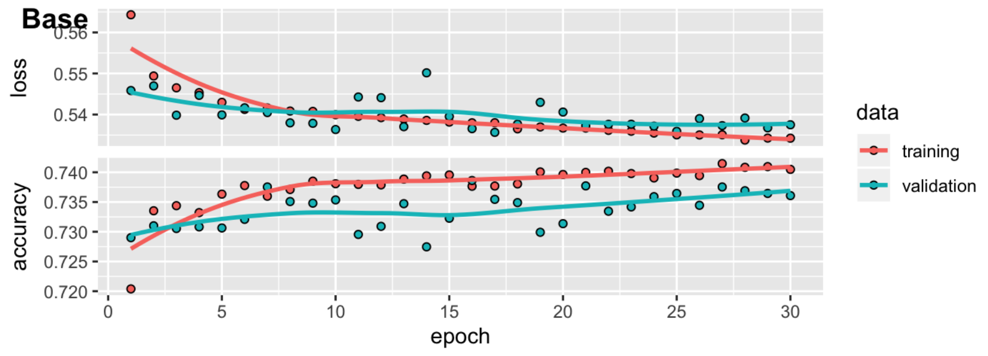
```
##    .metric .estimator .estimate .metric.1 .estimator.1 .estimate.1
## 1 accuracy     binary 0.7306719   roc_auc       binary   0.7941462
## 2      kap     binary 0.4613406   roc_auc       binary   0.7941462
```

Hyperparameter tuning for ANN2

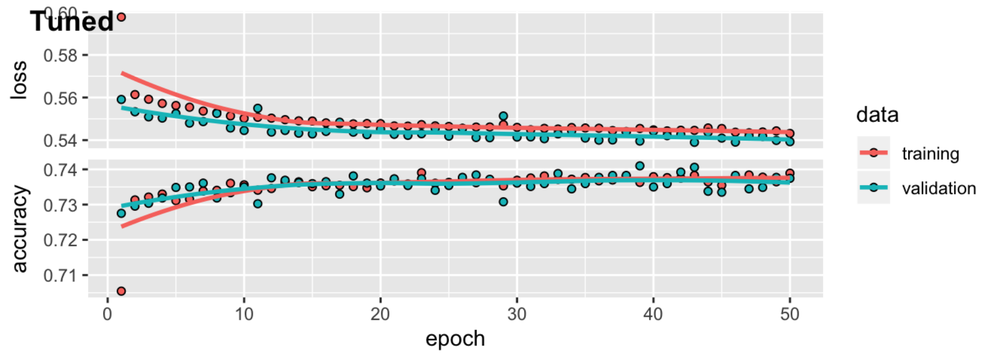
```
#hyperparameter tuning 1
set.seed(123)
model_keras2_tune <- keras_model_sequential()
model_keras2_tune %>%
  layer_dense(units = 116,
              kernel_initializer = "uniform",
              activation = "relu",kernel_regularizer = regularizer_l2(l = 0.001),
              input_shape = ncol(x_train_tbl)) %>%
  layer_dense(units = 110,
              kernel_initializer = "uniform",
              activation = "relu",kernel_regularizer = regularizer_l2(l = 0.001)) %>%
  layer_dense(units = 1,
              kernel_initializer = "uniform",
              activation = "sigmoid") %>%
  compile(optimizer = "adam",
          loss = "binary_crossentropy",
          metrics = c("accuracy"))

history2_tune <- model_keras2_tune %>% fit(
  as.matrix(x_train_tbl),
  y_train_vec,
  #optimizer = optimizer_rmsprop(lr = 0.0001),
  batch_size = 32,
  epochs = 10,
  validation_split = 0.3,
  verbose=0)


#hyperparameter tuning 2
set.seed(123)
model_keras2_tune <- keras_model_sequential()
model_keras2_tune %>%
  layer_dense(units = 116,
              kernel_initializer = "uniform",
              activation = "relu",kernel_regularizer = regularizer_l2(l = 0.001),
              input_shape = ncol(x_train_tbl)) %>%
  layer_dense(units = 110,
              kernel_initializer = "uniform",
              activation = "relu",kernel_regularizer = regularizer_l2(l = 0.001)) %>%
  layer_dense(units = 1,
              kernel_initializer = "uniform",
              activation = "sigmoid") %>%
  compile(optimizer = "adam",
          loss = "binary_crossentropy",
          metrics = c("accuracy"))
history2_tune <- model_keras2_tune %>% fit(
  as.matrix(x_train_tbl),
  y_train_vec,
  #optimizer = optimizer_rmsprop(lr = 0.0001),
  batch_size = 64,
  epochs = 30,
  validation_split = 0.25,
  verbose=0)


#hyperparameter tuning 3
set.seed(123)
model_keras2_tune <- keras_model_sequential()
model_keras2_tune %>%
  layer_dense(units = 116,
              kernel_initializer = "uniform",
              activation = "relu",kernel_regularizer = regularizer_l2(l = 0.001),
              input_shape = ncol(x_train_tbl)) %>%
  layer_dense(units = 110,
              kernel_initializer = "uniform",
              activation = "relu",kernel_regularizer = regularizer_l2(l = 0.001)) %>%
  layer_dense(units = 1,
              kernel_initializer = "uniform",
              activation = "sigmoid") %>%
  compile(optimizer = "adam",
          loss = "binary_crossentropy",
          metrics = c("accuracy"))
history2_tune <- model_keras2_tune %>% fit(
  as.matrix(x_train_tbl),
  y_train_vec,
  #optimizer = optimizer_rmsprop(lr = 0.0001),
  batch_size = 128,
  epochs =50,
  validation_split = 0.2,
  verbose=0)

yhat(model_keras2_tune)
```

```
##    .metric .estimator .estimate .metric.1 .estimator.1 .estimate.1
## 1 accuracy     binary 0.7295289   roc_auc       binary    0.796704
## 2      kap     binary 0.4590539   roc_auc       binary    0.796704
```

```
tv_plot2 <- ggarrange(plot(history2),plot(history2_tune),
                    labels = c("Base", "Tuned"),
                    ncol = 1, nrow = 2)
tv_plot2
```



The loss and both training and



validation accuracies have slightly decreased.

```
base_tune2<- c("Base","Tuned")
Accuracy_of_ANN2<- c(73.14,73.49)
a2_models<-data.frame(base_tune2,Accuracy_of_ANN2)
a2_models
```

```
##   base_tune2 Accuracy_of_ANN2
## 1       Base            73.14
## 2      Tuned            73.49
```

Decision Tree

```
#install.packages("MLmetrics")
#library(MLmetrics)
set.seed(123)
dt_model = train(Disease ~ ., data = training,
                  method="rpart")
predict_dt <- predict(dt_model,newdata = testing)
confusionMatrix(predict_dt, testing$Disease)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  4833 2140
##      True   1292 3984
##
##                Accuracy : 0.7198
##                  95% CI : (0.7118, 0.7278)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4396
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7891
##             Specificity : 0.6506
##          Pos Pred Value : 0.6931
##          Neg Pred Value : 0.7551
##              Prevalence : 0.5000
##          Detection Rate : 0.3946
##    Detection Prevalence : 0.5693
##       Balanced Accuracy : 0.7198
##
##        'Positive' Class : False
##
```

Hyperparameter tuning Decision Tree

```
library(caret)
library(rpart)
set.seed(123)
dt_model_tune <- train(Disease ~ ., data = training,
                  metric = "Accuracy",
                  method = "rpart",
                  tuneLength = 8,
                  trControl=trainControl("repeatedcv",number=10,repeats=10,classProbs=TRUE),
                  control =rpart.control(minsplit=20,minbucket=round(20/3),maxdepth=15,
                                       tuneGrid = expand.grid(cp = seq(0, 0.1, 0.02))))
```

Hyper-parameters: minsplit: The minimum number of observations that must exist in a node in order for a split to be attempted minbucket: The minimum number of observations in any terminal node. maxdepth: The maxdepth parameter prevents the tree from growing past a certain depth / height.. The root node is treated a depth 0 cp:The complexity parameter (cp) is the minimum improvement in the model needed at each node.

```
predict_dt_tune <- predict(dt_model_tune,newdata = testing)
confusionMatrix(predict_dt_tune, testing$Disease)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  4822 1986
##      True   1303 4138
##
##                Accuracy : 0.7315
##                  95% CI : (0.7235, 0.7393)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.463
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7873
##             Specificity : 0.6757
##          Pos Pred Value : 0.7083
##          Neg Pred Value : 0.7605
##              Prevalence : 0.5000
##          Detection Rate : 0.3937
##    Detection Prevalence : 0.5558
##       Balanced Accuracy : 0.7315
##
##        'Positive' Class : False
##
```

```
base_tune_dt<- c("Base","Tuned")
Accuracy_of_dt<- c(72.31,73.88)
a_dt_models<-data.frame(base_tune_dt,Accuracy_of_dt)
a_dt_models
```

```
##   base_tune_dt Accuracy_of_dt
## 1         Base          72.31
## 2        Tuned          73.88
```

The accuracy is increased for the tuned model.

Building a model : SVM-Linear *Hyperparameter Tuning for SVM-Linear:*

```
trctrl <- trainControl(method = "cv",number=3,repeats = 2)
set.seed(3233)
unwantedoutput1 <- capture.output(model_svm_linear_tune <- train(Disease ~ ., data =training,
                          method = "svmLinear",
                          metric="Accuracy",
                          trControl=trctrl,
                          tuneLength = 10,
                          tuneGrid = expand.grid(C = seq(0.5, 1, 2))))
```

```
predict_svm_linear_tune <- predict(model_svm_linear_tune, newdata = testing)
confusionMatrix(predict_svm_linear_tune, testing$Disease)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  4982 2184
##      True   1143 3940
##
##                Accuracy : 0.7284
##                  95% CI : (0.7204, 0.7362)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4568
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8134
##             Specificity : 0.6434
##          Pos Pred Value : 0.6952
##          Neg Pred Value : 0.7751
##              Prevalence : 0.5000
##          Detection Rate : 0.4067
##    Detection Prevalence : 0.5850
##       Balanced Accuracy : 0.7284
##
##        'Positive' Class : False
##
```

1.For a linear kernel, the choice of C does not seem to affect performance very much.

Building a model : Random forest *Hyperparameter Tuning for Random forest:*

```
control <- trainControl(method="cv", number=3, repeats=1)
mtry <- c(1,2,5,10)
tunegrid <- expand.grid(.mtry=mtry)
```

Each axis of the grid is an algorithm parameter, and points in the grid are specific combinations of parameters. Because we are only tuning one parameter, the grid search is a linear search through a vector of candidate values.mtry parameter is available in caret for tuning. 1. **mtry:** Number of variables randomly sampled as candidates at each split.

```
set.seed(124)
model_rf_tune <- train(Disease~., data=training, method="rf",metric='Accuracy',tuneGrid=tunegrid, trControl=
control)
model_rf_tune
```

```
## Random Forest
##
## 36751 samples
##    16 predictor
##     2 classes: 'False', 'True'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 24501, 24500, 24501
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    1    0.7050691  0.4101248
##    2    0.7344561  0.4689062
##    5    0.7278984  0.4557935
##   10    0.7096132  0.4192253
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
predict_rf_tune <- predict(model_rf_tune, newdata = testing)
confusionMatrix(predict_rf_tune, testing$Disease)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction False True
##      False  4862 2063
##      True   1263 4061
##
##                Accuracy : 0.7285
##                  95% CI : (0.7205, 0.7363)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4569
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7938
##             Specificity : 0.6631
##          Pos Pred Value : 0.7021
##          Neg Pred Value : 0.7628
##              Prevalence : 0.5000
##          Detection Rate : 0.3969
##    Detection Prevalence : 0.5654
##       Balanced Accuracy : 0.7285
##
##        'Positive' Class : False
##
```

Building a model : Gradient Boosting *Hyperparameter Tuning for Gradient Boosting:*

```
#install.packages("doParallel")
control <- trainControl(method="repeatedcv", number=3, repeats=3)
tgrid<- expand.grid(n.trees =c(1080:1100),
                    interaction.depth=c(1:3),
                    shrinkage=c(0.001,0.2,0.3),
                    n.minobsinnode=15)
```

1.**n.trees:** The total number of trees in the sequence or ensemble.Since they can easily overfit if there are many number of trees,we must find the optimal number of trees that minimize the loss function of interest with cross validation. 2.**shrinkage:** Determines the contribution of each tree on the final outcome and controls how quickly the algorithm proceeds down the gradient descent.Generally, the smaller this value, the more accurate the model can be but also will require more trees in the sequence. 3.**interaction.depth:** Controls the depth of the individual trees.Higher depth trees allow the algorithm to capture unique interactions but also increase the risk of over-fitting. 4.**n.minobsinnode:** Controls the complexity of each tree.Higher values help prevent a model from learning relationships which might be highly specific to the particular sample selected for a tree (overfitting) but smaller values can help with imbalanced target classes in classification problems.

```
set.seed(124) #for reproducability
unwantedoutput <- capture.output(model_gbm_tune <- train(Disease ~ ., data = training, method = "gbm",metric
="Accuracy",tuneGrid=tgrid,trControl=control))
```

```
predict_gbm_tune <- predict(model_gbm_tune, newdata = testing)
confusionMatrix(predict_gbm_tune, testing$Disease)
```

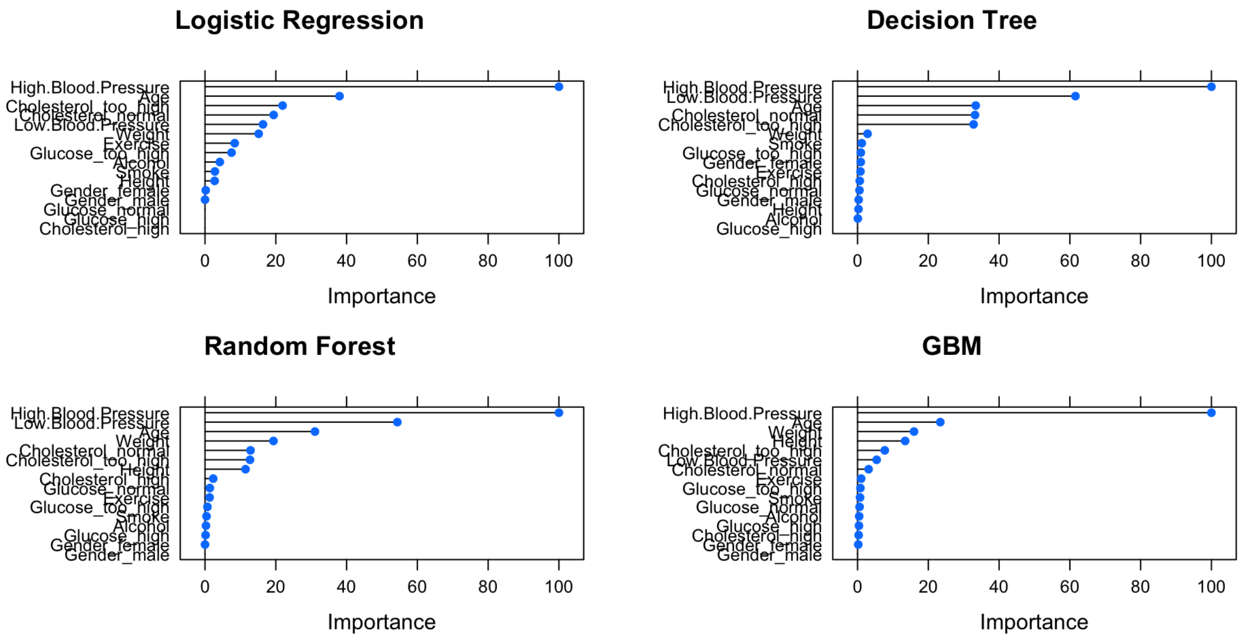
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False  4692 1864
##      True   1433 4260
##
##                Accuracy : 0.7308
##                  95% CI : (0.7229, 0.7387)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4617
##
##  Mcnemar's Test P-Value : 6.954e-14
##
##             Sensitivity : 0.7660
##             Specificity : 0.6956
##          Pos Pred Value : 0.7157
##          Neg Pred Value : 0.7483
##              Prevalence : 0.5000
##          Detection Rate : 0.3831
##    Detection Prevalence : 0.5352
##       Balanced Accuracy : 0.7308
##
##        'Positive' Class : False
##
```

Feature Importance of four models

```
library(gbm)
```

```
## Loaded gbm 2.1.5
```

```
grid.arrange(plot(varImp(elastic_reg),main="Logistic Regression"),
             plot(varImp(dt_model_tune),main="Decision Tree"),
             plot(varImp(model_rf_tune),main="Random Forest"),
             plot(varImp(model_gbm_tune),
                  top=16 ,main="GBM"),nrow = 2, ncol = 2)
```

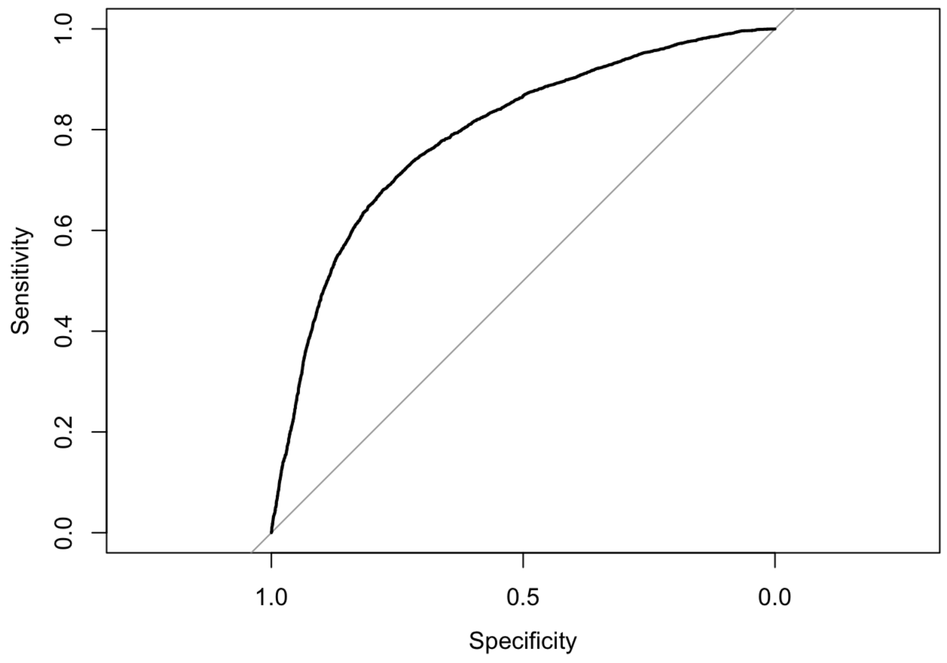


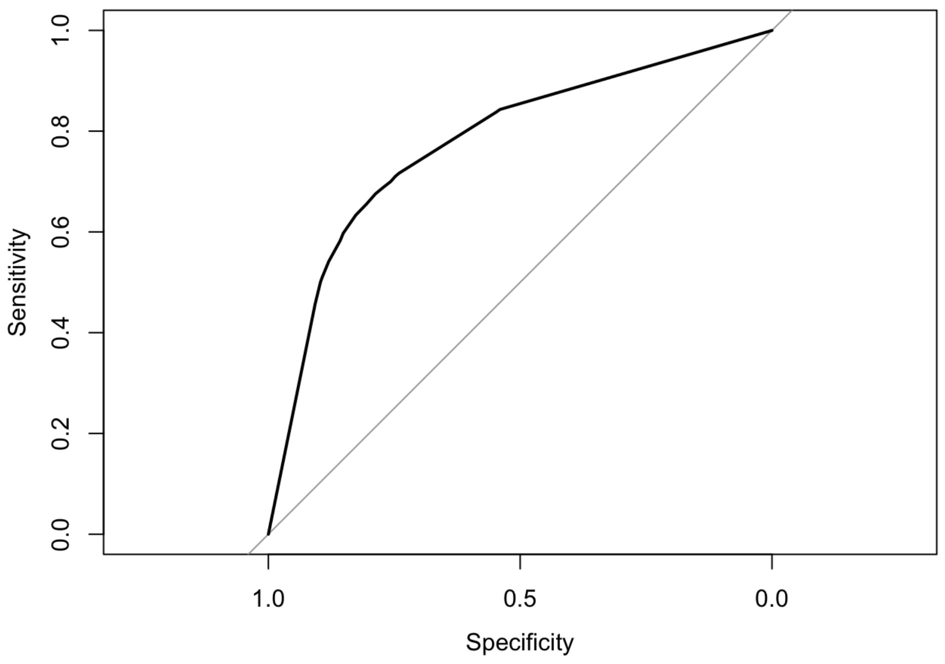
Feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node. The node probability can be calculated by the number of samples that reach the node, divided by the total number of samples. The higher the value the more important the feature. A decision tree is a simple, decision making-diagram. Random forests are a large number of trees, combined (using averages or "majority rules") at the end of the process. Gradient boosting machines also combine decision trees, but start the combining process at the beginning, instead of at the end. After comparing the feature importances of the above four models, The top 4 features are:

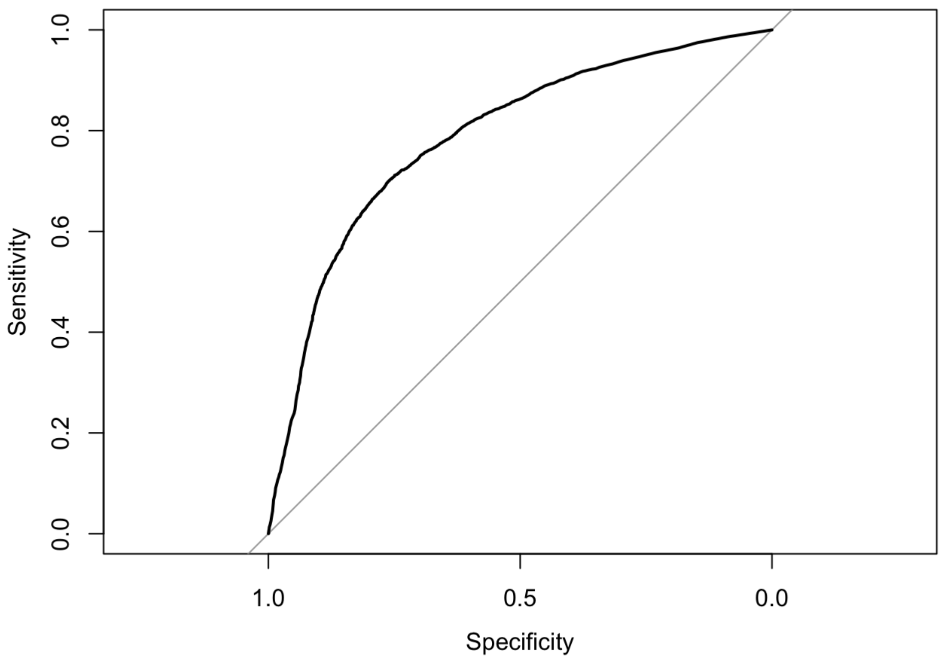
High Blood Pressure (occurs as the top feature in all models) Age (is in the top 4 features in all models) Cholesterol too high & Weight (occurs twice in top 4 features) Cholesterol_normal, Height, Low Blood Pressure are also important features for all models.

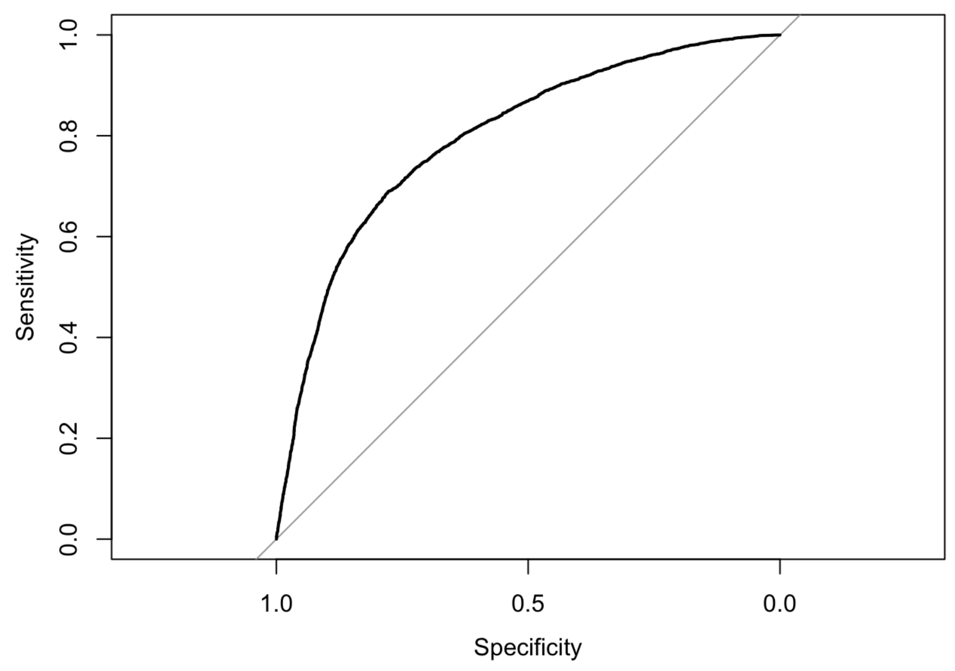
ROC AND AUC An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds.

```
roc_func<- function(mo){
probs <- predict(mo,testing,type="prob")
head(probs)
colnames(probs)[1]="False"
colnames(probs)[2]="True"
roc_curve <- roc(testing$Disease,probs$True)
plot(roc_curve)
}

listee<-c(roc_func(elastic_reg),
          roc_func(dt_model_tune),
          roc_func(model_rf_tune),
          roc_func(model_gbm_tune))
```




Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
1.0
0.5
0.0
Specificity


Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
1.0
0.5
0.0
Specificity



Area under the curve AUC provides an aggregate measure of performance across all possible classification thresholds. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0.

```
auc_func<- function(mo){
probs <- predict(mo,testing,type="prob")
head(probs)
colnames(probs)[1]="False"
colnames(probs)[2]="True"
auc_mo<-auc(testing$Disease,probs$True)
auc_mo
}
auc_ANN0<-0.7964934
auc_ANN1<-0.8050782
auc_ANN2<-0.8055849
auc_lr<-auc_func(elastic_reg)
auc_dt<-auc_func(dt_model_tune)
auc_rf<-auc_func(model_rf_tune)
auc_gbm<-auc_func(model_gbm_tune)
```

Comparing Areas under curve:

```
Model <- c("Logistic Regression","Decision Trees","Random_Forest","Gradient_boosting","ANN0","ANN1","ANN2")
AUC <- c(auc_lr,auc_dt,auc_rf,auc_gbm,auc_ANN0,auc_ANN1,auc_ANN2)

auc<-data.frame(Model,AUC)
auc<-auc[with(auc, order(-AUC)), ]
auc
```

```
##                 Model       AUC
## 7                ANN2 0.8055849
## 6                ANN1 0.8050782
## 4   Gradient_boosting 0.7966766
## 5                ANN0 0.7964934
## 1 Logistic Regression 0.7899298
## 3       Random_Forest 0.7875738
## 2      Decision Trees 0.7786245
```

Gradient Boosting model has the highest AUC.

Comparing performance of three models specifically: linear SVM, logistic regression, and single layer perceptron (with ZERO hidden layer).

```
three_m<- c("SVM_Linear","Logistic Regression","ANN0")
Acc<- c(73.25,73.48,73.22)
acc_three_models<-data.frame(three_m,Acc)
acc_three_models<-acc_three_models[with(acc_three_models, order(-Acc)), ]
acc_three_models
```

```
##                three_m   Acc
## 2 Logistic Regression 73.48
## 1          SVM_Linear 73.25
## 3                ANN0 73.22
```

The accuracies obtained from these three models are quite similar. If we look at the optimization problems of linear SVM and (regularized) LR, they are very similar: That is, they only differ in the loss function — SVM minimizes hinge loss while logistic regression minimizes logistic loss. A network without a hidden layer is actually identical to a logistic regression model if the logistic (sigmoidal) activation function is used.SVMs separates data without using kernal trick, hyperplanes are linear which are roughly equivalent to neural networks without an activation function.

Master table with Models,Accuracies, Hyperparameters and time taken

```
Model_name <- c("SVM_Linear",
                "Logistic Regression",
                "ANN0",
                "Random Forest",
                "Gradient Boosting",
                "KNN",
                "Naive Bayes",
                "SVM Non_linear",
                "Decision Trees",
                "ANN1",
                "ANN2")
Hyperparameters <- c("C",
                     "alpha & lambda",
                     "(epochs,batch_size,kernel regularizer,no of hidden nodes,validation split)",
                     "(mtry)",
                     "(n.trees,interaction.depth,shrinkage,n.minobsinnode)",
                     "(k)",
                     "(laplace tuning)",
                     "(sigma,C)",
                     "(minsplit,minbucket,maxdepth,cp)",
                     "(epochs,batch_size,kernel regularizer,no of hidden nodes,validation split)",
                     "(epochs,batch_size,kernel regularizer,no of hidden nodes,validation split)"
                     )
Accuracies <- c(73.25,
                73.48,
                73.22,
                73.3,
                73.97,
                63.42,
                60.49,
                71.58,
                72.88,
                73.90,
                73.49)
Time <- c("2 hrs","5 mins" ,"15 mins","20 mins","35 mins","15 mins","2 mins","8 hrs","15 mins","15 mins","15
mins")



acc_models<-data.frame(Model_name,Hyperparameters,Accuracies,Time)
acc_models <- acc_models[with(acc_models, order(-Accuracies)), ]
acc_models
```

```
##              Model_name
## 5    Gradient Boosting
## 10                ANN1
## 11                ANN2
## 2  Logistic Regression
## 4        Random Forest
## 1           SVM_Linear
## 3                 ANN0
## 9       Decision Trees
## 8       SVM Non_linear
## 6                  KNN
## 7          Naive Bayes
##                                                           Hyperparameters
## 5                        (n.trees,interaction.depth,shrinkage,n.minobsinnode)
## 10 (epochs,batch_size,kernel regularizer,no of hidden nodes,validation split)
## 11 (epochs,batch_size,kernel regularizer,no of hidden nodes,validation split)
## 2                                                          alpha & lambda
## 4                                                                  (mtry)
## 1                                                                       C
## 3  (epochs,batch_size,kernel regularizer,no of hidden nodes,validation split)
## 9                                         (minsplit,minbucket,maxdepth,cp)
## 8                                                               (sigma,C)
## 6                                                                     (k)
## 7                                                         (laplace tuning)
##    Accuracies    Time
## 5       73.97 35 mins
## 10      73.90 15 mins
## 11      73.49 15 mins
## 2       73.48  5 mins
## 4       73.30 20 mins
## 1       73.25   2 hrs
## 3       73.22 15 mins
## 9       72.88 15 mins
## 8       71.58   8 hrs
## 6       63.42 15 mins
## 7       60.49  2 mins
```

The performance metric used is Accuracy. Gradient Boosting has the highest accuracy followed by ANN1.Since the data is balanced, accuracy is used. Used cross validation method.

SECTION 3: TESTING DATA

1. Prediction & Interpretation on Test Data Reading the Testing data csv and storing it into a dataframe

```
#Loading Weather Forecasting training dataset.

d_test <- read.csv("/Users/juilee81/Desktop/DA/DA_HW_03/Disease\ Prediction\ Testing.csv",header = TRUE)
#View(d_test)
```

Checking for missing values column-wise and visualize the data:

```
colSums(is.na(d_test))
```

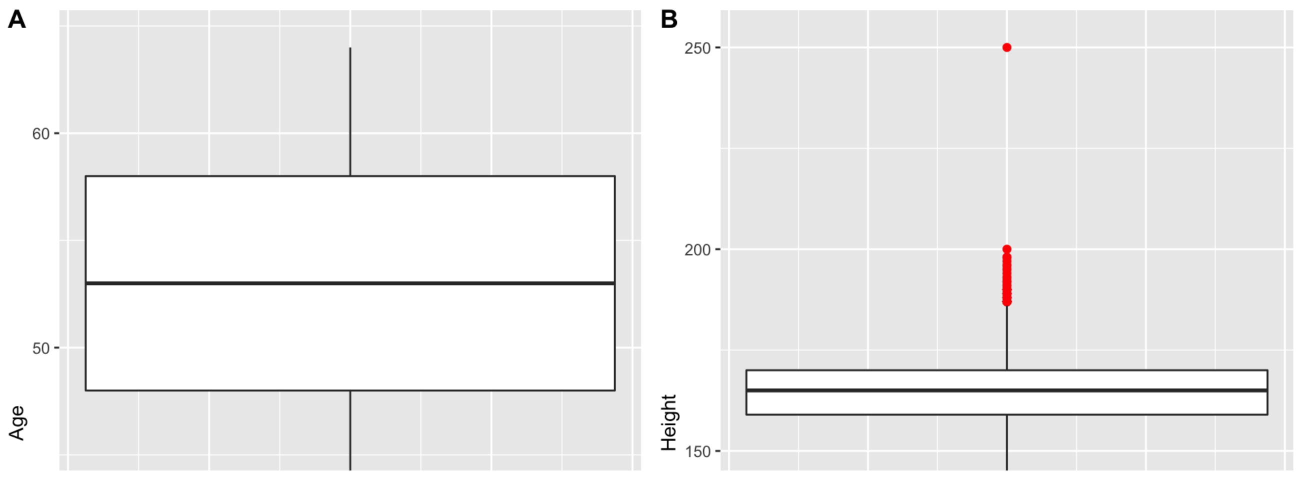
```
##                  ID                 Age              Gender
##                   0                   0                   0
##              Height              Weight High.Blood.Pressure
##                   0                   0                   0
##   Low.Blood.Pressure         Cholesterol             Glucose
##                   0                   0                   0
##               Smoke             Alcohol            Exercise
##                   0                   0                   0
```

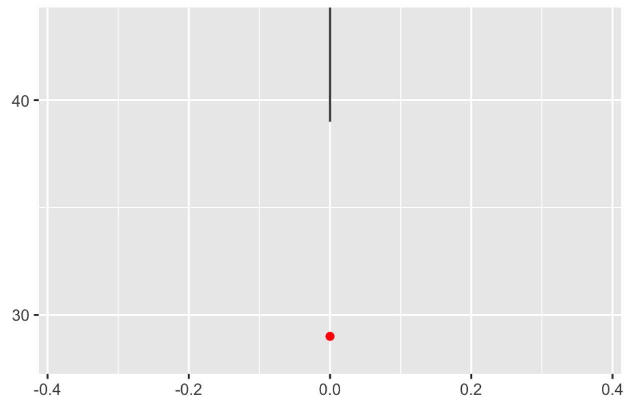
Outlier detection

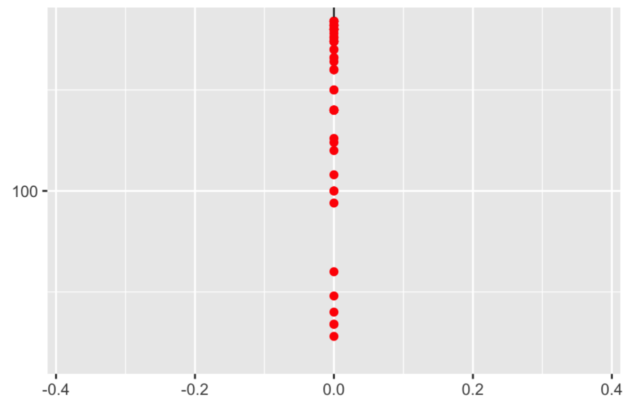
```
summary(d_test)
```

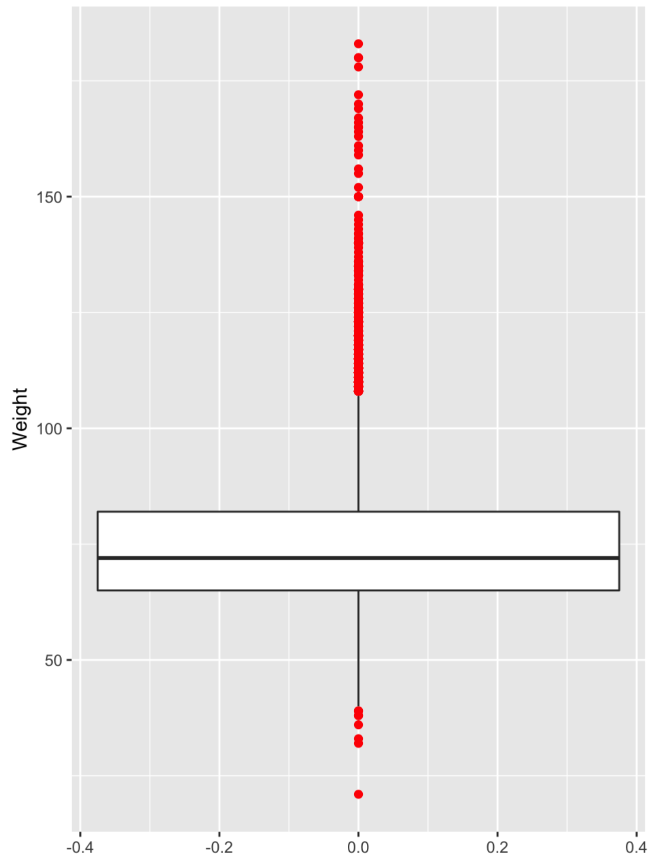
```
##        ID              Age           Gender          Height     
##  Min.   :    0   Min.   :29.00   female:13667   Min.   : 64.0  
##  1st Qu.: 5250   1st Qu.:48.00   male  : 7333   1st Qu.:159.0  
##  Median :10500   Median :53.00                  Median :165.0  
##  Mean   :10500   Mean   :52.81                  Mean   :164.3  
##  3rd Qu.:15749   3rd Qu.:58.00                  3rd Qu.:170.0  
##  Max.   :20999   Max.   :64.00                  Max.   :250.0  
##      Weight       High.Blood.Pressure Low.Blood.Pressure   Cholesterol   
##  Min.   : 21.00   Min.   :   10.0     Min.   : -70.00    high     : 2844  
##  1st Qu.: 65.00   1st Qu.:  120.0     1st Qu.:  80.00    normal   :15709  
##  Median : 72.00   Median :  120.0     Median :  80.00    too high: 2447  
##  Mean   : 74.24   Mean   :  129.1     Mean   :  95.96                    
##  3rd Qu.: 82.00   3rd Qu.:  140.0     3rd Qu.:  90.00                    
##  Max.   :183.00   Max.   :16020.0     Max.   :8500.00                    
##      Glucose            Smoke             Alcohol            Exercise    
##  high    : 1563   Min.   :0.00000   Min.   :0.00000   Min.   :0.000  
##  normal  :17827   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:1.000  
##  too high: 1610   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
```

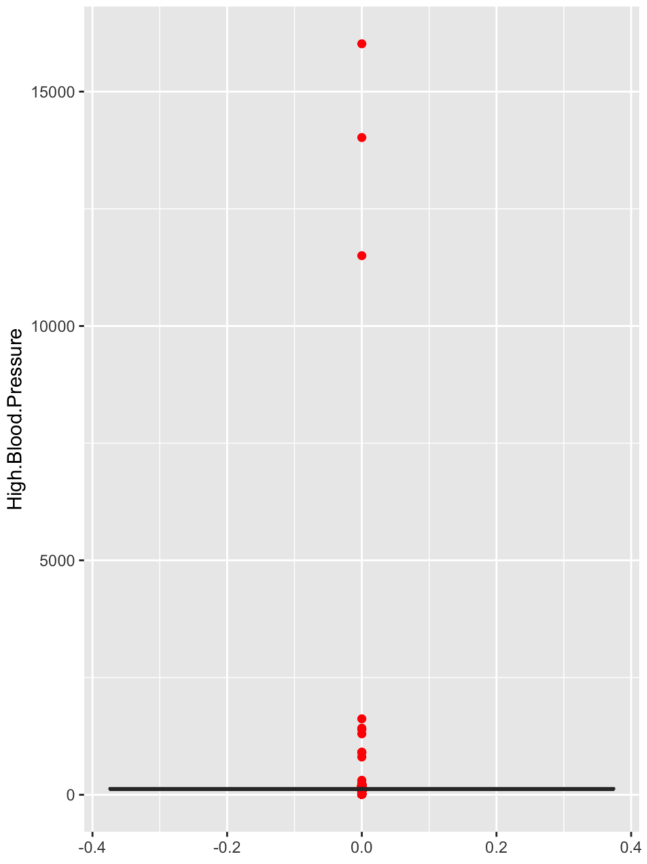
```r
bxp_Age<- ggplot(d_test, aes(y=Age))+
              geom_boxplot(outlier.colour="red",
              outlier.shape=16,
              outlier.size=2, notch=FALSE)

bxp_Height<- ggplot(d_test, aes(y=Height))+
              geom_boxplot(outlier.colour="red",
              outlier.shape=16,
              outlier.size=2, notch=FALSE)

bxp_Weight<- ggplot(d_test, aes(y=Weight))+
              geom_boxplot(outlier.colour="red",
              outlier.shape=16,
              outlier.size=2, notch=FALSE)

bxp_High.Blood.Pressure<- ggplot(d_test, aes(y=High.Blood.Pressure))+
              geom_boxplot(outlier.colour="red",
              outlier.shape=16,
              outlier.size=2, notch=FALSE)

bxp_Low.Blood.Pressure<- ggplot(d_test, aes(y=Low.Blood.Pressure))+
              geom_boxplot(outlier.colour="red",
              outlier.shape=16,
              outlier.size=2, notch=FALSE)



figure <- ggarrange(bxp_Age,bxp_Height,bxp_Weight,bxp_High.Blood.Pressure,bxp_Low.Blood.Pressure,
                    labels = c("A", "B"),
                    ncol = 2, nrow = 3)
figure
```

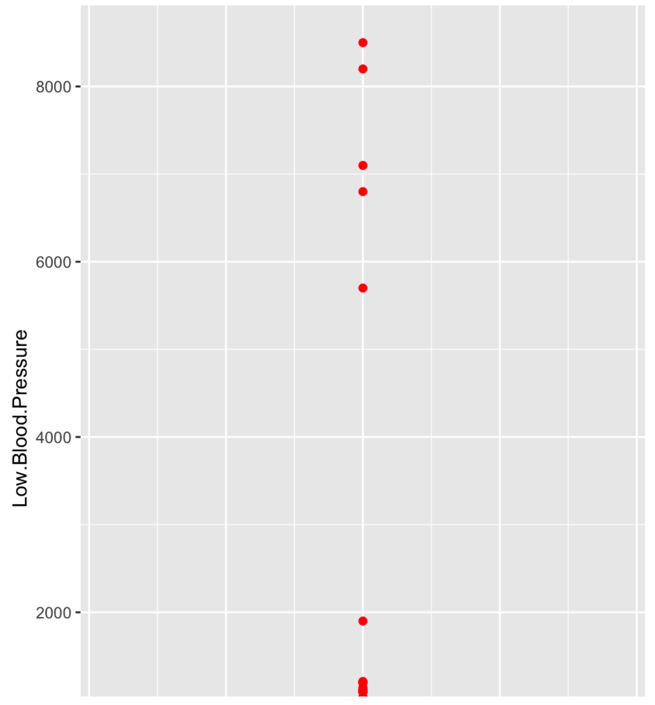


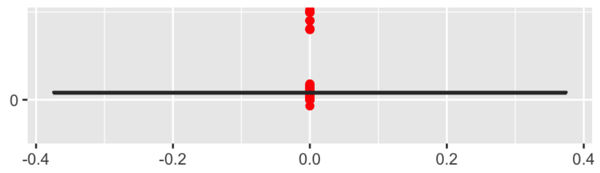

40
30
-0.4
-0.2
0.0
0.2
0.4


100
-0.4
-0.2
0.0
0.2
0.4


Weight
150
100
50
-0.4
-0.2
0.0
0.2
0.4


High.Blood.Pressure
15000
10000
5000
0
-0.4
-0.2
0.0
0.2
0.4


Low.Blood.Pressure
8000
6000
4000
2000



Outliers are detected in the variables of High and low blood Pressure, Weight and height.This can be seen from the summary and boxplots.

Treating Outliers:

```
#install.packages("raster")
out<- function(q1,q3) {
  l<- q1-1.5*(q3-q1)
  u<- q3+1.5*(q3-q1)
  print(paste(l,u))
}
#Clamping using inter-quartile range
out(159,170)
```

```
## [1] "142.5 186.5"
```

```
d_test$Height<-clamp(d_test$Height, lower=142.5, upper=186.5)
out(65,82)
```

```
## [1] "39.5 107.5"
```

```
d_test$Weight<-clamp(d_test$Weight, lower=39.5, upper=107.5)
out(80,90)#low blood pressure
```

```
## [1] "65 105"
```

```
out(120,140)#High blood pressure
```

```
## [1] "90 170"
```

```
#Since low blood pressure cannot be greater than high blood pressure, therefore swapping lower of High blood
pressure and upper of low blood pressure.
d_test$High.Blood.Pressure<-clamp(d_test$High.Blood.Pressure, lower=105, upper=170)
d_test$Low.Blood.Pressure<-clamp(d_test$Low.Blood.Pressure, lower=65, upper=90)
```

Normalising the numeric data

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

```
d_test$Age <- normalize(d_test$Age)
d_test$Height <- normalize(d_test$Height)
d_test$Weight <- normalize(d_test$Weight)
d_test$Low.Blood.Pressure <- normalize(d_test$Low.Blood.Pressure)
d_test$High.Blood.Pressure  <- normalize(d_test$High.Blood.Pressure)
```

Creating dummy variables of all the categorical variables

```
library(fastDummies)
ID <- d_test$ID
d_dummies_test <- fastDummies::dummy_cols(d_test,select_columns=c('Gender','Cholesterol','Glucose'))
d_dummies_test <- d_dummies_test[,c(-1,-3,-8,-9)]
colnames(d_dummies_test)[13] <- "Cholesterol_too_high"
colnames(d_dummies_test)[16] <- "Glucose_too_high"
```

Logistic model

```
predict_lr_final <- predict(elastic_reg,d_dummies_test)
```

ANN0 model

```
predict_ANNO_final <- predict_classes(model_keras0_tune,x = as.matrix(d_dummies_test))
```

ANN1 model

```
predict_ANN1_final <- predict_classes(model_keras1_tune,x = as.matrix(d_dummies_test))
```

ANN2 model

```
predict_ANN2_final <- predict_classes(model_keras2_tune,x = as.matrix(d_dummies_test))
```

Decision Tree model

```
predict_dt_final <- predict(dt_model_tune, newdata = d_dummies_test)
```

INTO CSV

```
Disease_predictions_csv <-data.frame(ID,
                            predict_dt_final,
                            predict_lr_final,
                            predict_ANNO_final,
                            predict_ANN1_final,
                            predict_ANN2_final)

colnames(Disease_predictions_csv) <- c('ID', 'DT', 'LR', 'ANN0', 'ANN1','ANN2')
Disease_predictions_csv$ANN0 <-factor(Disease_predictions_csv$ANN0,labels = c("False", "True"))
Disease_predictions_csv$ANN1 <-factor(Disease_predictions_csv$ANN1,labels = c("False", "True"))
Disease_predictions_csv$ANN2 <-factor(Disease_predictions_csv$ANN2,labels = c("False", "True"))

rownames(Disease_predictions_csv)<-NULL
```

Writing to CSV

```
write.csv(Disease_predictions_csv,"HW_04_Juilee_Salunkhe_Predictions.csv")
```

Conclusion In this study we have explored the data of unspecified disease dataset and gain insights about the key factors that decide the whether or not the person has the disease or not using multiple machine learning algorithms and data analysis.We have also compared various machine learning models in terms of accuracy as a performance metric and found that Gradient Boosting has the highest accuracy.