Coronary Heart Disease

Submitted towards partial fulfillment of the criteria for award of

PGP – BABI
By
Great Lakes Institute of Management

Submitted by:
Padma Radhika Upadhyayula (Batch: Apr19_B)

Great Lakes Institute of Management

Abstract

This project focuses on a class of "Machine Learning" techniques and their potential impact in the domain of Health Care Analytics.

This document will focus on providing insights on the Coronary Heart Risk dataset, its subjective limitations and our approach towards collection of data on which the Machine Learning algorithms will be applied.

The tools and techniques used to substantiate our findings include working on R Studio and using Data Mining, Predictive Modeling and Machine Learning Techniques limited to use of K Nearest Neighbors (KNN), Logistic Regression, Decision Trees(CART), Random Forest, Artificial Neural Networks and Bagging.
This study is limited to the ~4200 patients in the given dataset and one decision variable: TenYearCHD (A Ten year risk of Coronary Heart Disease).

The findings of this study give us the insight that KNN is one of the judicious techniques for the treatment of missing values, in case of multi-dimensional data.
Also this study found out that Random Forest Algorithm has better performance than other models of Logistic Regression, CART, Artificial Neural Networks and Bagging.
This study uses the performance measures of KS, AUC, Gini Index, Accuracy and Classification Error to evaluate the models developed.
At the end, this study has been successful to find the most important features that influence the Coronary Heart Disease.

Acknowledgements

I am very much thankful for all the knowledge, guidance and support imparted by Mr. Abhay Poddar who gave me invaluable knowledge during the course of the capstone.

In addition, I would also like to convey my deep gratitude towards my Mentor Mr. Animesh Tiwari for helping me gain knowledge and understanding the Machine Learning Techniques used in this project.

Lastly, I would like to thank my family members for their continuous support.

Certificate of Authenticity

I, the undersigned, hereby declare that the capstone report entitled, Coronary Heart Risk submitted by me to the Great Lakes Institute of Management, in partial fulfillment of the criteria for award of PGP-BABI under the guidance of Mr. Abhay Poddar, is my original work and the conclusions drawn therein are based on work done by me.

The final Capstone report is my own work and has not been duplicated from any other source.

Place: Hyderabad

Date: 5th April 2020

Padma Radhika Upadhyayula

Certificate of Completion

This is to certify that Mrs.Padma Radhika Upadhyayula of Great Lakes Institute of Management has successfully completed the capstone report titled “Coronary Heart Risk” in partial fulfillment of the criteria for award of PGP-BABI as prescribed by Great Lakes Institute of Management.

This capstone report is the record of authentic work carried by her during Feb 2020 till Apr 2020.

She has worked under my guidance.

Signature

(Mr. Abhay Poddar)

Date:

Contents

Introduction

Coronary heart disease is a type of heart disease that develops when the arteries of the heart cannot deliver enough oxygen-rich blood to the heart. It is the leading cause of death in the United States and in many other countries. Hence, building a model which can predict the probability of a patient suffering Coronary Heart disease in the next 10 years help patients and health care providers to focus on the important factors that influence the heart disease and to take preventive actions to reduce the risk of getting Heart disease.

Need and Objective of the study

The need of this project is to develop a prediction system which can predict the probability of a patient suffering Coronary Heart Disease in the next 10 years along with identifying the most important factors that influence the heart disease. Using Machine Learning Algorithms, we can identify the random patterns and insights which will help the healthcare professionals to predict the risk of getting Coronary Heart Disease in the next 10 years as well as in identifying the most important risk factors to be focused on in order to prevent or minimize the risk of getting Coronary Heart Disease.

For this project, the data set Coronary_heart_risk_study.csv has been considered to build the predictive model.

The objective of this study is to explore the effectiveness of various Machine Learning Techniques (especially Classification) to predict the risk of getting Heart Disease.

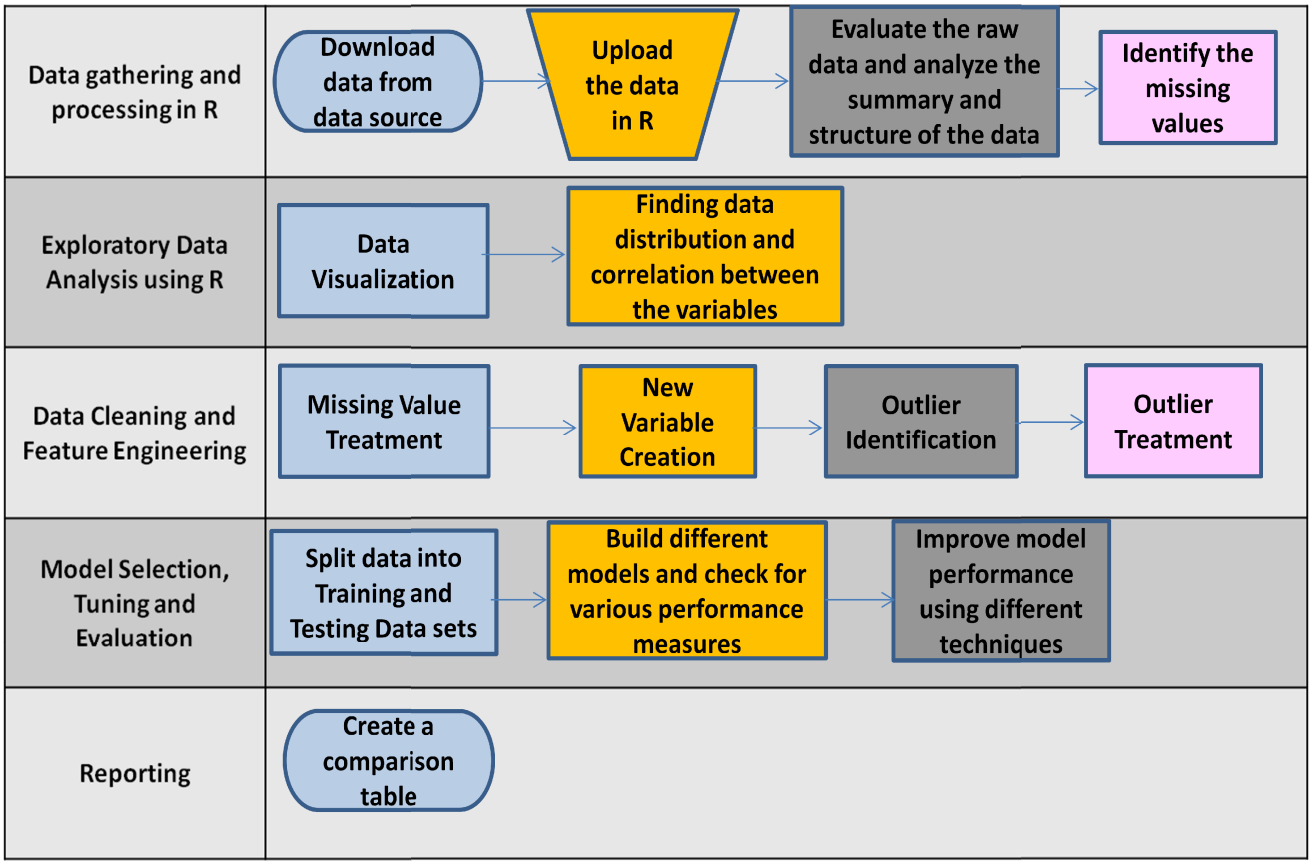
The approach to getting the solution involves:

1) Using predictor/target variable.
2) Use of Classification Techniques:
    a) To predict the risk of getting Coronary Heart Disease in next 10 years.
    b) And to Identify the most important risk factors that influence the heart disease.
3) Conclusion would be which Classification technique helps us to predict the outcome with greater degree of accuracy.

Project Workflow

The work flow used for this project is as below:

Figure 1 ***Detailed workflow followed to reach the solution for this study***



Data Report

Data set consists of 4240 rows and 16 columns of data representing various risk factors.

The sample data is as shown below

male	age	education	currentSm	cigsPerDa	BPMeds	prevalent	prevalent	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	TenYearCHD
1	39	4	0	0	0	0	0	0	195	106	70	26.97	80	77	0
0	46	2	0	0	0	0	0	0	250	121	81	28.73	95	76	0
1	48	1	1	20	0	0	0	0	245	127.5	80	25.34	75	70	0
0	61	3	1	30	0	0	1	0	225	150	95	28.58	65	103	1
0	46	3	1	23	0	0	0	0	285	130	84	23.1	85	85	0
0	43	2	0	0	0	0	1	0	228	180	110	30.3	77	99	0
0	63	1	0	0	0	0	0	0	205	138	71	33.11	60	85	1
0	45	2	1	20	0	0	0	0	313	100	71	21.68	79	78	0
1	52	1	0	0	0	0	1	0	260	141.5	89	26.36	76	79	0
1	43	1	1	30	0	0	1	0	225	162	107	23.61	93	88	0
0	50	1	0	0	0	0	0	0	254	133	76	22.91	75	76	0

The internal structure of the data is :

```
> str(heartRiskData)
'data.frame':   4240 obs. of  16 variables:
 $ male           : int  1 0 1 0 0 0 0 0 1 1 ...
 $ age            : int  39 46 48 61 46 43 63 45 52 43 ...
 $ education      : int  4 2 1 3 3 2 1 2 1 1 ...
 $ currentSmoker  : int  0 0 1 1 1 0 0 1 0 1 ...
 $ cigsPerDay     : int  0 0 20 30 23 0 0 20 0 30 ...
 $ BPMeds         : int  0 0 0 0 0 0 0 0 0 0 ...
 $ prevalentStroke: int  0 0 0 0 0 0 0 0 0 0 ...
 $ prevalentHyp   : int  0 0 0 1 0 1 0 0 1 1 ...
 $ diabetes       : int  0 0 0 0 0 0 0 0 0 0 ...
 $ totChol        : int  195 250 245 225 285 228 205 313 260 225 ...
 $ sysBP          : num  106 121 128 150 130 ...
 $ diaBP          : num  70 81 80 95 84 110 71 71 89 107 ...
 $ BMI            : num  27 28.7 25.3 28.6 23.1 ...
 $ heartRate      : int  80 95 75 65 85 77 60 79 76 93 ...
 $ glucose        : int  77 76 70 103 85 99 85 78 79 88 ...
 $ TenYearCHD     : int  0 0 0 1 0 0 1 0 0 0 ...
```

- Data set is a combination of Categorical variables and Continuous variables.
- Male, education, current smoker, BPMeds, prevalentStroke, prevalentHyp, diabetes are the Categorical variables.
- Age, cigsPerDay, totChol, sysBP, diaBP, BMI, heartRate, glucose are the Continuous variables.
- TenYearCHD is a binary variable which is also the dependant variable.
- Remaing 15 columns of Categorical and Continuous variables are the independent/predictor variables.

Please refer Appendix A for Source Code.

The Summary of the descriptive statistics of the data is:

```
> summary(heartRiskData)
      male             age          education       currentSmoker      cigsPerDay        BPMeds       prevalentStroke
 Min.   :0.0000   Min.   :32.00   Min.   :1.000   Min.   :0.0000   Min.   : 0.000   Min.   :0.00000   Min.   :0.000000
 1st Qu.:0.0000   1st Qu.:42.00   1st Qu.:1.000   1st Qu.:0.0000   1st Qu.: 0.000   1st Qu.:0.00000   1st Qu.:0.000000
 Median :0.0000   Median :49.00   Median :2.000   Median :0.0000   Median : 0.000   Median :0.00000   Median :0.000000
 Mean   :0.4292   Mean   :49.58   Mean   :1.979   Mean   :0.4941   Mean   : 9.006   Mean   :0.02962   Mean   :0.005896
 3rd Qu.:1.0000   3rd Qu.:56.00   3rd Qu.:3.000   3rd Qu.:1.0000   3rd Qu.:20.000   3rd Qu.:0.00000   3rd Qu.:0.000000
 Max.   :1.0000   Max.   :70.00   Max.   :4.000   Max.   :1.0000   Max.   :70.000   Max.   :1.00000   Max.   :1.000000
                                  NA's   :105                      NA's   :29       NA's   :53
  prevalentHyp       diabetes         totChol          sysBP           diaBP            BMI          heartRate        glucose
 Min.   :0.0000   Min.   :0.00000   Min.   :107.0   Min.   : 83.5   Min.   : 48.0   Min.   :15.54   Min.   : 44.00   Min.   : 40.00
 1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:206.0   1st Qu.:117.0   1st Qu.: 75.0   1st Qu.:23.07   1st Qu.: 68.00   1st Qu.: 71.00
 Median :0.0000   Median :0.00000   Median :234.0   Median :128.0   Median : 82.0   Median :25.40   Median : 75.00   Median : 78.00
 Mean   :0.3106   Mean   :0.02571   Mean   :236.7   Mean   :132.4   Mean   : 82.9   Mean   :25.80   Mean   : 75.88   Mean   : 81.96
 3rd Qu.:1.0000   3rd Qu.:0.00000   3rd Qu.:263.0   3rd Qu.:144.0   3rd Qu.: 90.0   3rd Qu.:28.04   3rd Qu.: 83.00   3rd Qu.: 87.00
 Max.   :1.0000   Max.   :1.00000   Max.   :696.0   Max.   :295.0   Max.   :142.5   Max.   :56.80   Max.   :143.00   Max.   :394.00
                                    NA's   :50                                      NA's   :19      NA's   :1        NA's   :388
   TenYearCHD
 Min.   :0.0000
 1st Qu.:0.0000
 Median :0.0000
 Mean   :0.1519
 3rd Qu.:0.0000
 Max.   :1.0000
```

- There are about 645 null values present in the data set.

Exploratory Data Analysis

The following approach is followed for Exploratory Data Analysis.

I. Missing Value Treatment
II. New Variable Creation
III. Outlier Treatment
IV. Univariate Analysis
V. Bivariate Analysis

I. Missing Value Treatment:

There are 645 total null values present in the given data set. As this number is large, we need to impute the null values with appropriate values.

Used KNN algorithm to impute the missing values.

The Summary of the descriptive statistics after missing value treatment is:

```
> summary(impute.data)
```

male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes
Min. :0.0000	Min. :32.00	Min. :1.000	Min. :0.0000	Min. : 0.000	Min. :0.00000	Min. :0.000000	Min. :0.0000	Min. :0.00000
1st Qu.:0.0000	1st Qu.:42.00	1st Qu.:1.000	1st Qu.:0.0000	1st Qu.: 0.000	1st Qu.:0.00000	1st Qu.:0.000000	1st Qu.:0.0000	1st Qu.:0.00000
Median :0.0000	Median :49.00	Median :2.000	Median :0.0000	Median : 0.000	Median :0.00000	Median :0.000000	Median :0.0000	Median :0.00000
Mean :0.4292	Mean :49.58	Mean :1.978	Mean :0.4941	Mean : 9.075	Mean :0.02948	Mean :0.005896	Mean :0.3106	Mean :0.02571
3rd Qu.:1.0000	3rd Qu.:56.00	3rd Qu.:3.000	3rd Qu.:1.0000	3rd Qu.:20.000	3rd Qu.:0.00000	3rd Qu.:0.000000	3rd Qu.:1.0000	3rd Qu.:0.00000
Max. :1.0000	Max. :70.00	Max. :4.000	Max. :1.0000	Max. :70.000	Max. :1.00000	Max. :1.000000	Max. :1.0000	Max. :1.00000

totChol	sysBP	diaBP	BMI	heartRate	glucose	TenYearCHD
Min. :107.0	Min. : 83.5	Min. : 48.0	Min. :15.54	Min. : 44.00	Min. : 40.00	Min. :0.0000
1st Qu.:206.0	1st Qu.:117.0	1st Qu.: 75.0	1st Qu.:23.07	1st Qu.: 68.00	1st Qu.: 72.00	1st Qu.:0.0000
Median :234.0	Median :128.0	Median : 82.0	Median :25.40	Median : 75.00	Median : 78.00	Median :0.0000
Mean :236.7	Mean :132.4	Mean : 82.9	Mean :25.80	Mean : 75.88	Mean : 81.64	Mean :0.1519
3rd Qu.:263.0	3rd Qu.:144.0	3rd Qu.: 90.0	3rd Qu.:28.04	3rd Qu.: 83.00	3rd Qu.: 86.00	3rd Qu.:0.0000
Max. :696.0	Max. :295.0	Max. :142.5	Max. :56.80	Max. :143.00	Max. :394.00	Max. :1.0000

II. New Variable Creation:

- As sysBP and diaBP represents the BP readings of a person, we combine these variables to get a new variable called BPDifference, which is the difference of these two values.

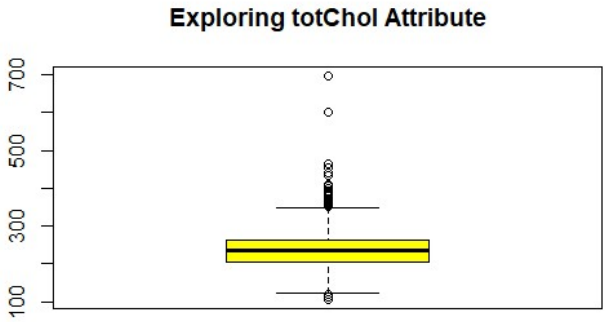
- sysBP and diaBP variables will be discarded from the data set and BPDifference variable will be included in the data set to build the model.
- The summary of BPDifference variable is:

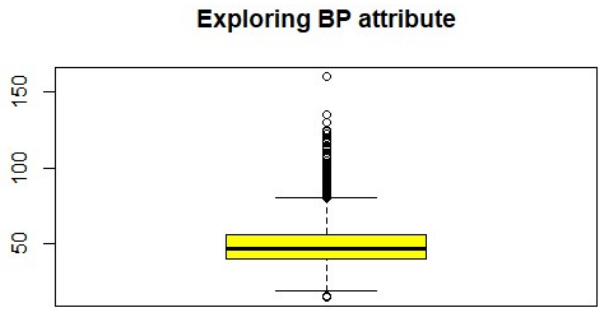
```
summary(BPDifference)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  15.00   40.00   47.00   49.46   56.00  160.00
```

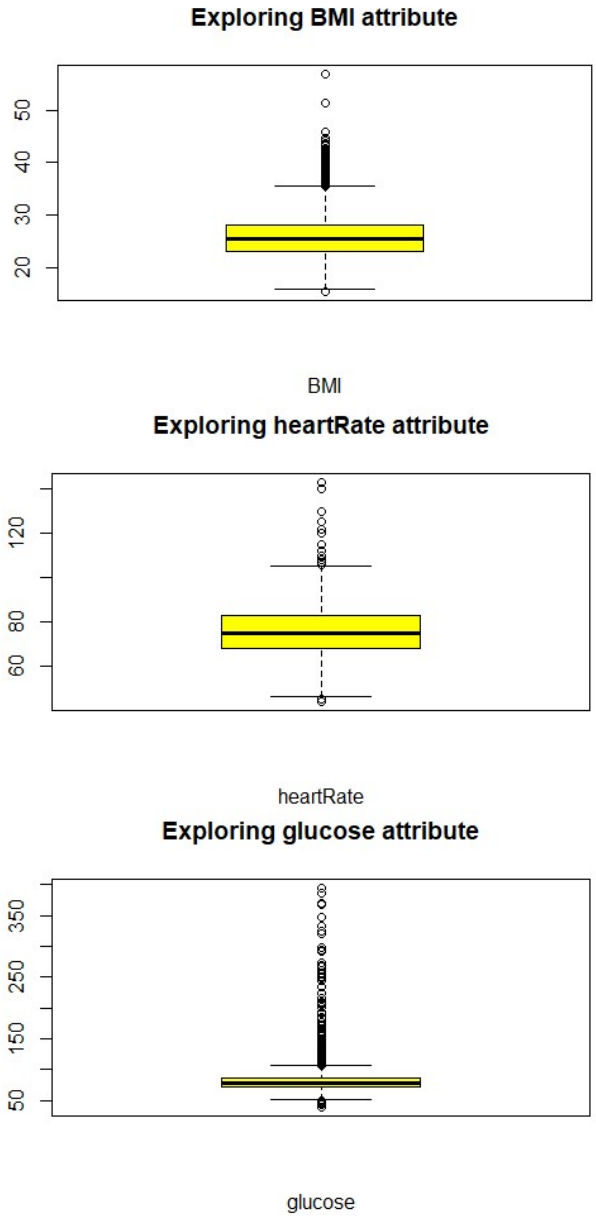
- Any value less than 40 is considered as low BP value and >= 56 is considered as high BP value.

III. Outlier Treatment:

The following box plots show that outliers are present in the variables: “totChol”, “BPDifference”, “BMI”, “heartRate” and “glucose”.

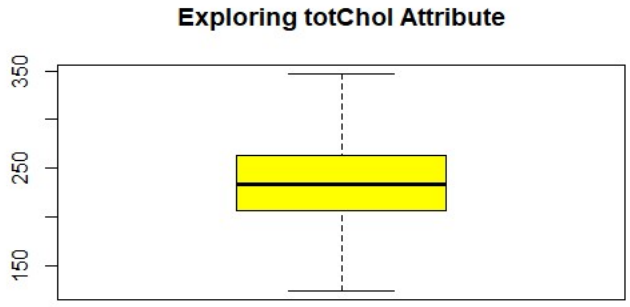


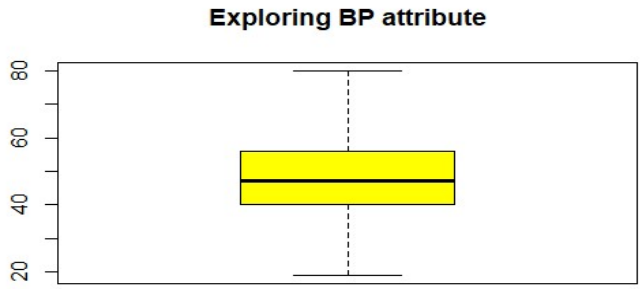


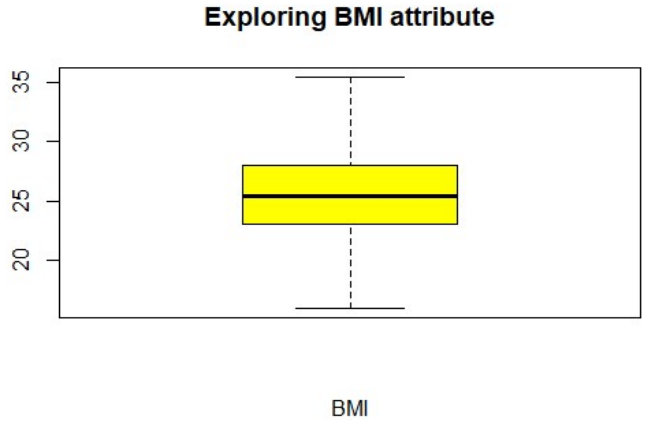


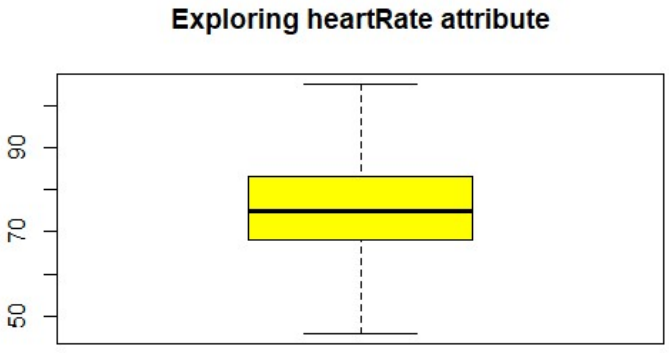
Outliers have been replaced by capping the outlier values.

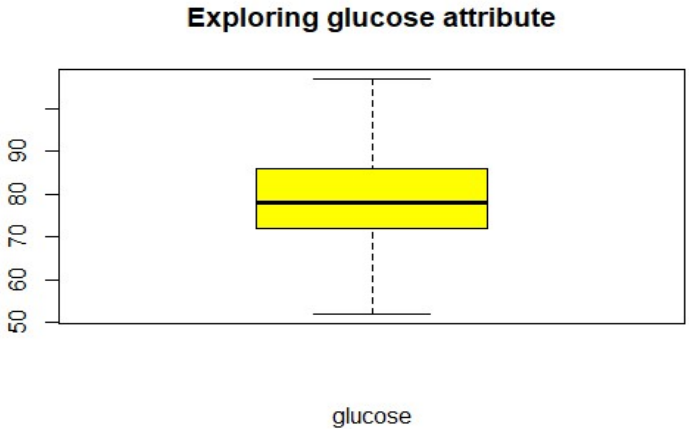
The following boxplots show that there are no outliers in the data.


Exploring totChol Attribute
350
250
150
Total Cholestrol


Exploring BP attribute
80
60
40
20
Difference in BP values


Exploring BMI attribute
35
30
25
20
BMI

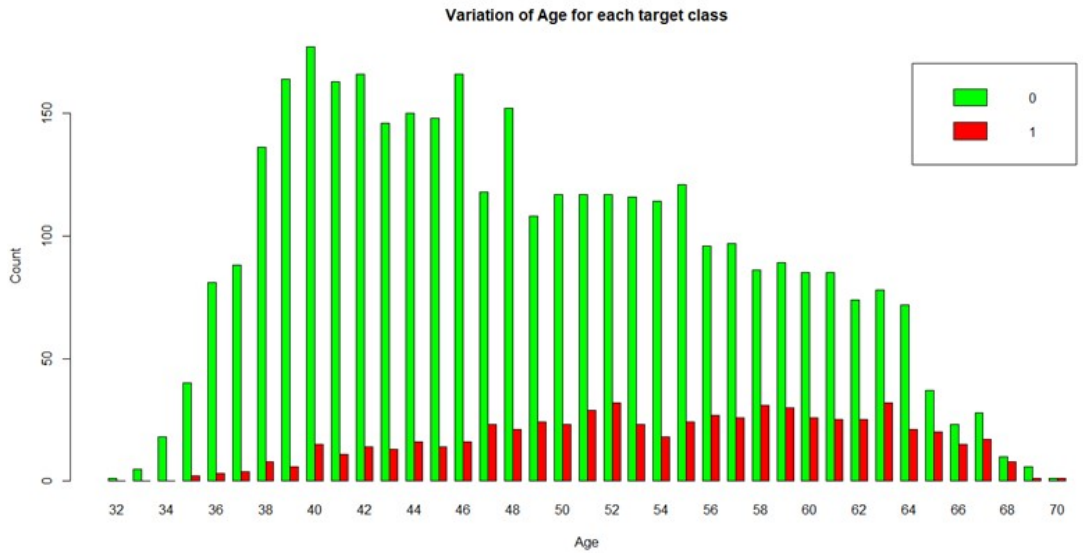




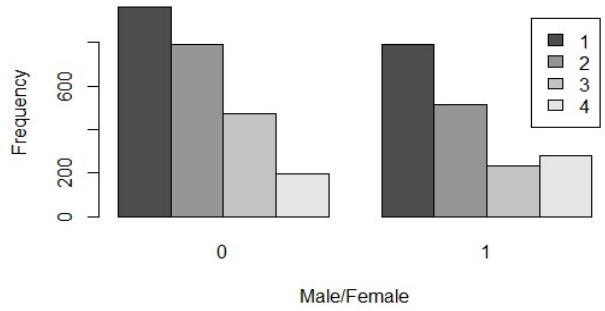
IV. Univariate Analysis:

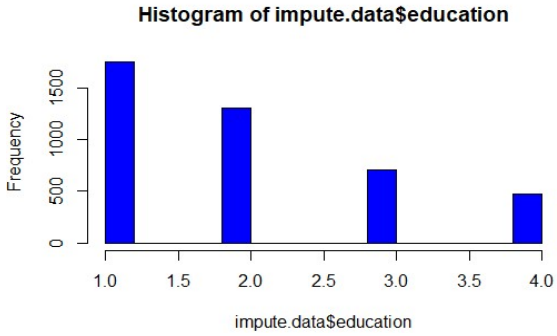
Univariate analysis has been done for both Categorical variables and Continuous variables.

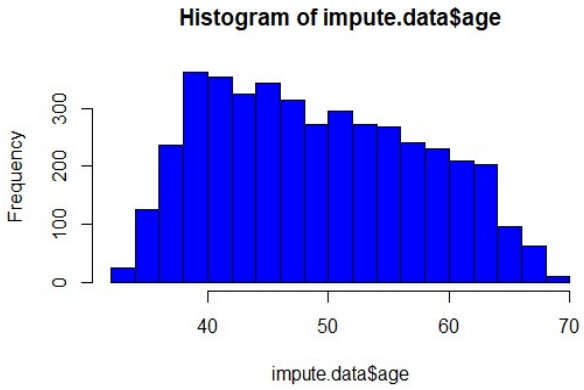
The following graphs show the distribution of these variables.

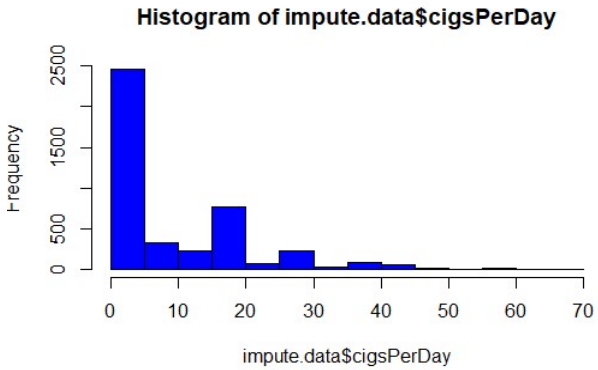

Variation of Age for each target class
Count
0
50
100
150
32
34
36
38
40
42
44
46
48
50
52
54
56
58
60
62
64
66
68
70
Age
0
1

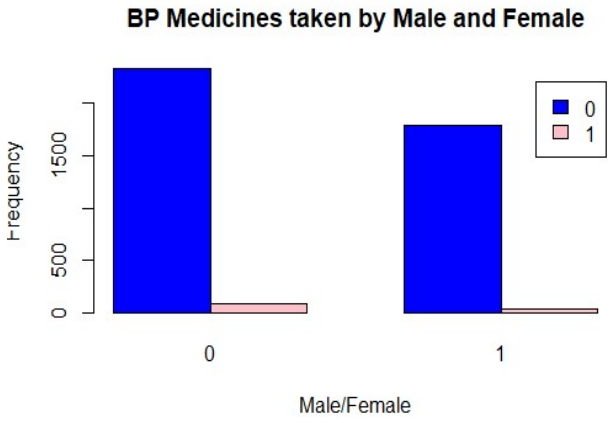

Variation of Gender for each target class
Count
0
500
1000
1500
2000
0
1
Male/Female
0
1

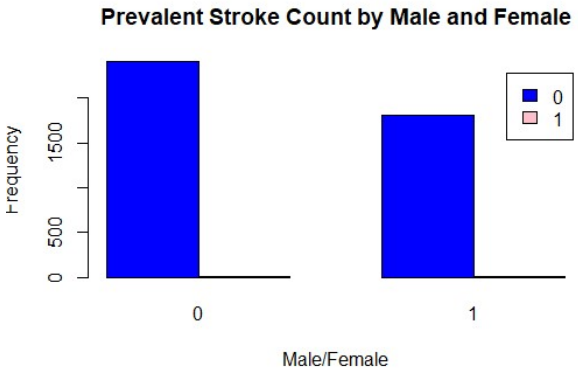

Frequency
0
200
600
1
2
3
4
0
1
Male/Female

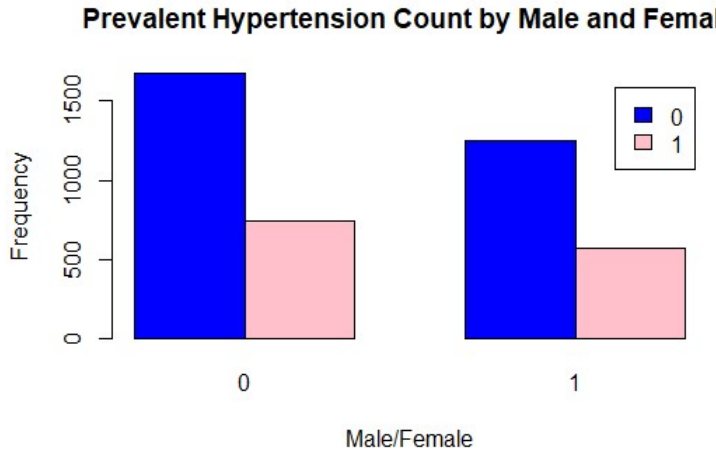

Histogram of impute.data$education
Frequency
0
500
1000
1500
1.0
1.5
2.0
2.5
3.0
3.5
4.0
impute.data$education

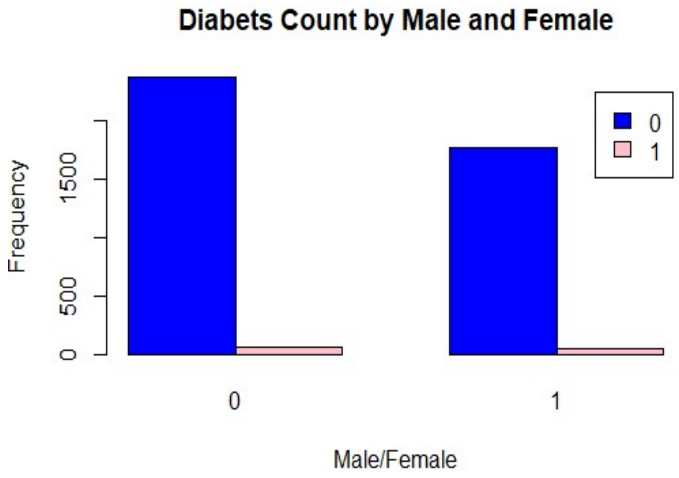

Histogram of impute.data$age
Frequency
0
100
200
300
40
50
60
70
impute.data$age

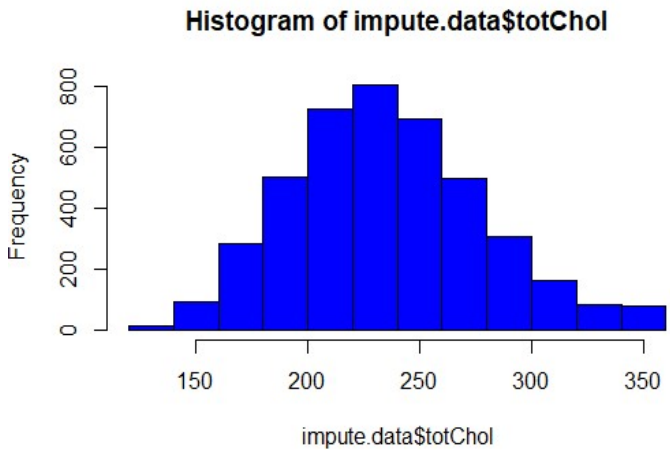

Histogram of impute.data$cigsPerDay
Frequency
0
500
1500
2500
0
10
20
30
40
50
60
70
impute.data$cigsPerDay

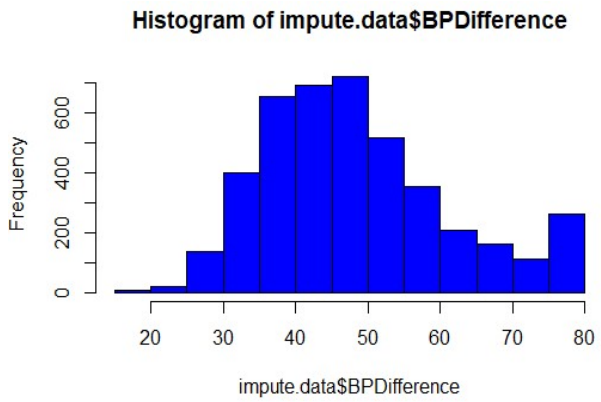

BP Medicines taken by Male and Female
Frequency
0
500
1500
0
1
0
1
Male/Female

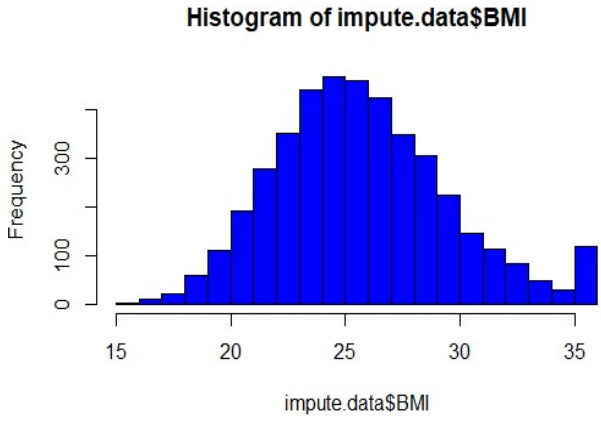

Prevalent Stroke Count by Male and Female
Frequency
0
500
1500
0
1
0
1
Male/Female

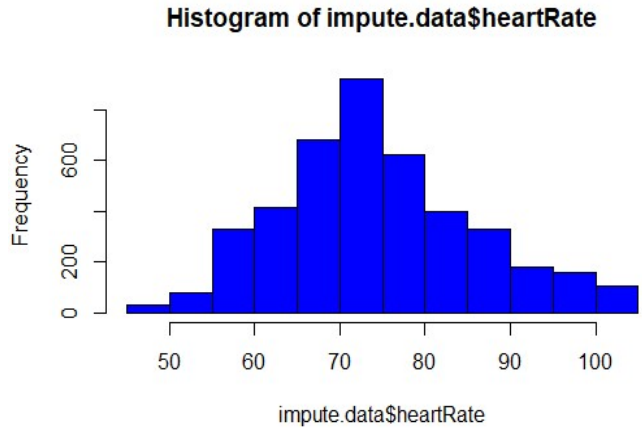

Prevalent Hypertension Count by Male and Female
Frequency
0
500
1000
1500
0
1
Male/Female

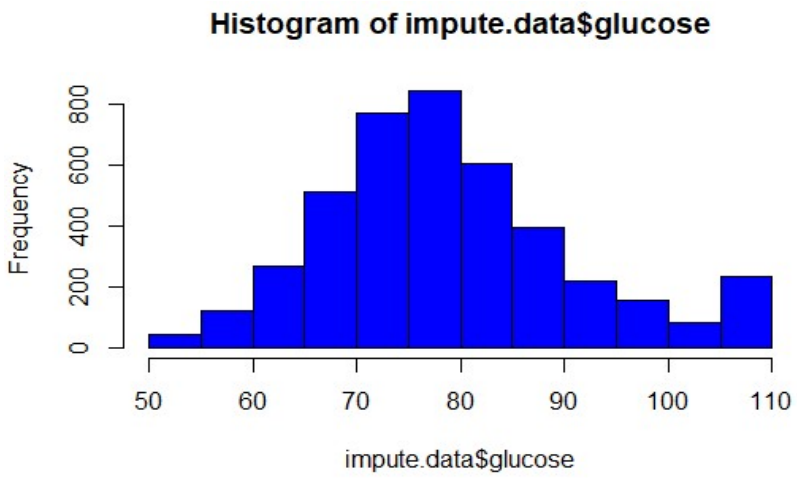

Diabets Count by Male and Female
Frequency
0
500
1500
0
1
Male/Female


Histogram of impute.data$totChol
Frequency
0
200
400
600
800
150
200
250
300
350
impute.data$totChol


Histogram of impute.data$BPDifference
Frequency
0
200
400
600
20
30
40
50
60
70
80
impute.data$BPDifference


Histogram of impute.data$BMI
Frequency
0
100
300
15
20
25
30
35
impute.data$BMI


Histogram of impute.data$heartRate
Frequency
0
200
600
50
60
70
80
90
100
impute.data$heartRate

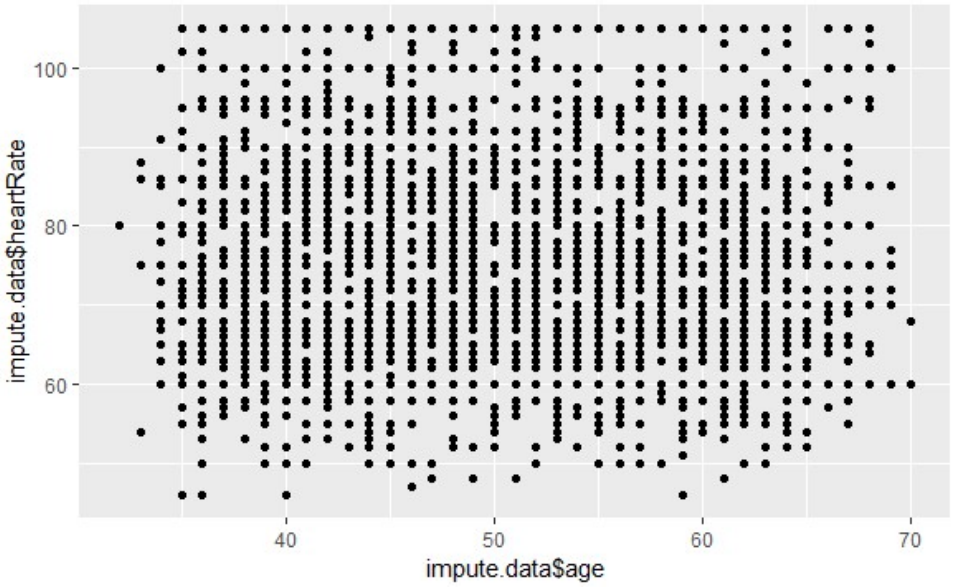


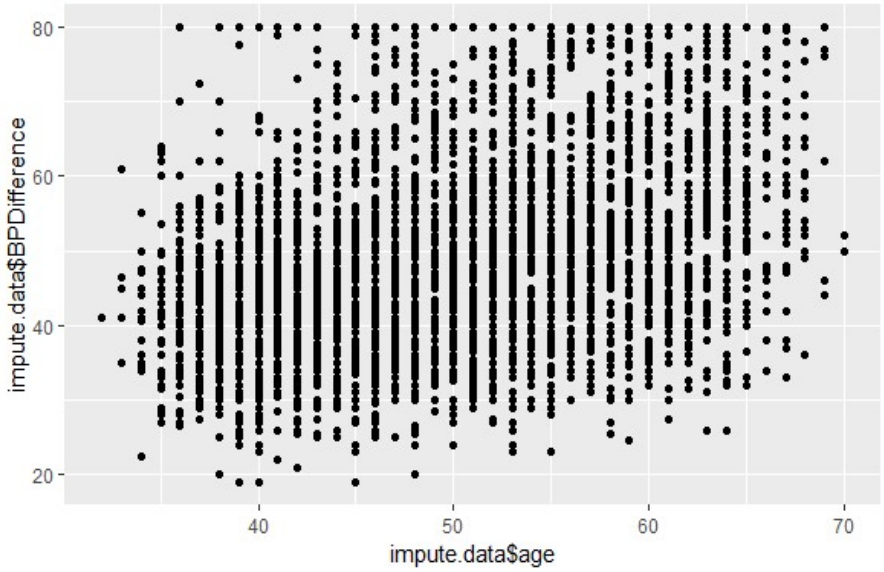
- The graphs of the categorical variables explain the frequency of their allowed categories.
- The graph of totChol variable shows that it is slightly right skewed and tailed at the end indicates the possibility of outliers.
- The graphs of BPDifference, BMI, heartRate and glucose are tailed at the end indicating the possibility of outliers.

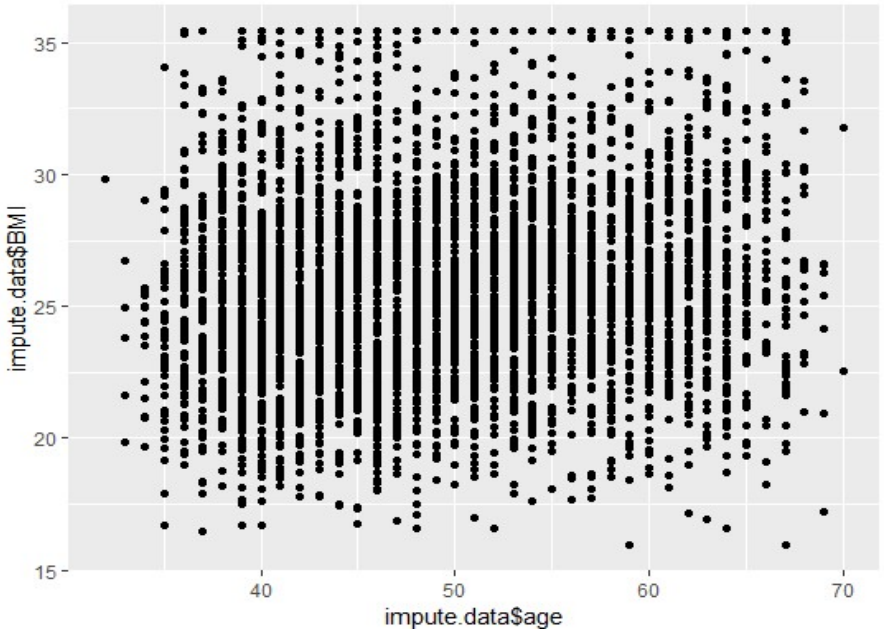
V. Bivariate Analysis:

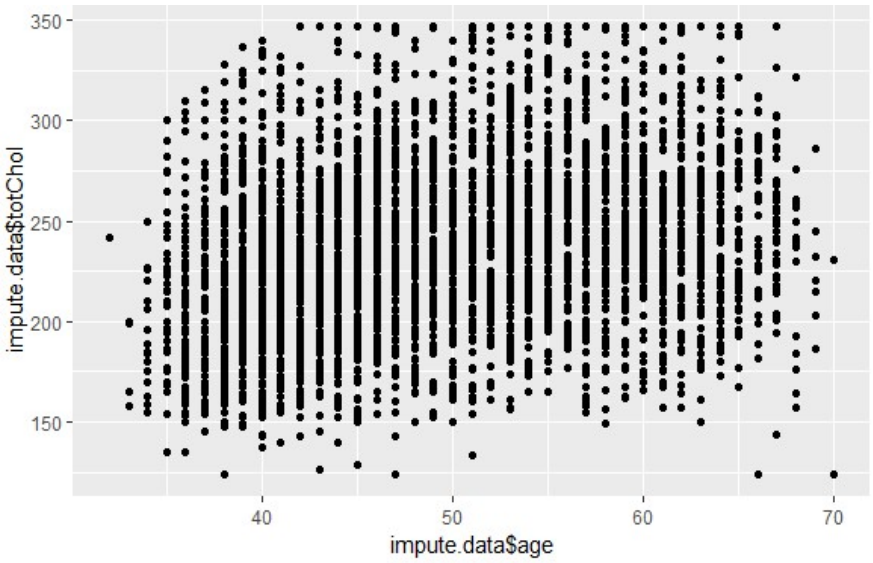
- The following scatter plots and correlation matrix shows if there exists any correlation between the independent variables.
- As per the graphs, age is not correlated with any of the continuous variables of totChol, BMI, heartRate, glucose and BPDifference.
- A box plot between BPMeds and BPDifference shows that, those with high BPDifference values are using BPMedicines.

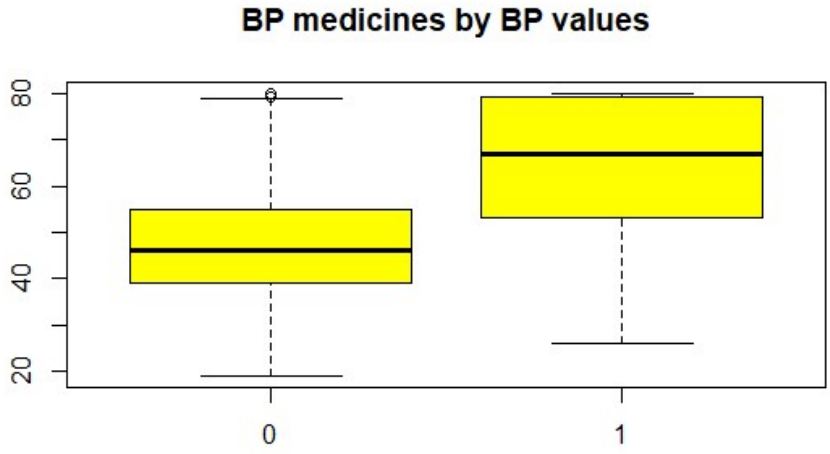

impute.data$glucose
100
90
80
70
60
50
40
50
60
70
impute.data$age

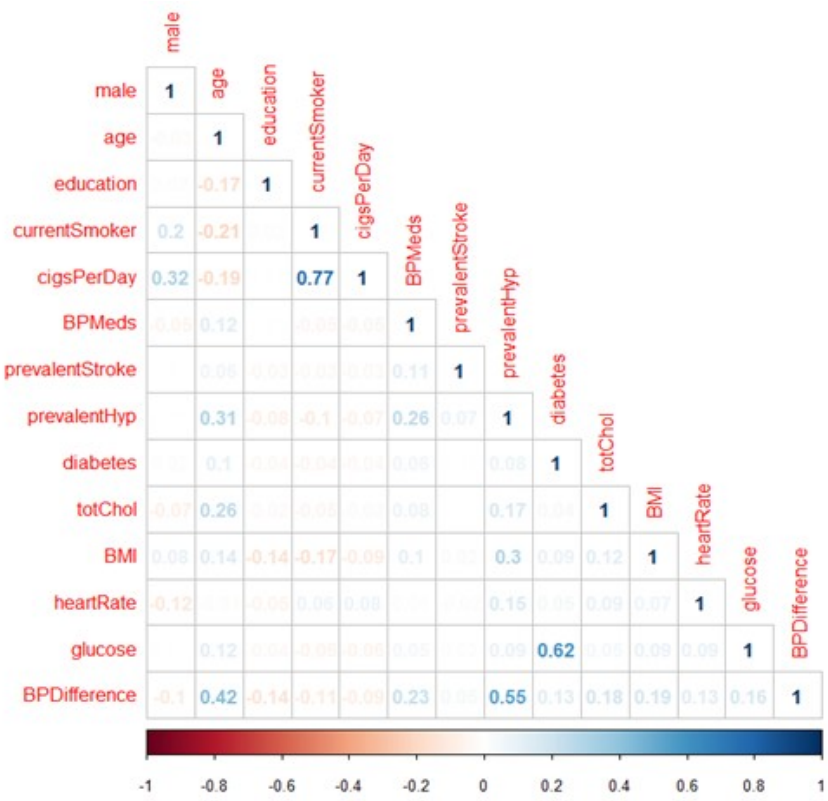

impute.data$heartRate
100
80
60
40
50
60
70
impute.data$age


80
60
40
20
impute.data$BPDifference
40
50
60
70
impute.data$age


35
30
25
20
15
impute.data$BMI
40
50
60
70
impute.data$age


350
300
250
200
150
impute.data$totChol
40
50
60
70
impute.data$age


BP medicines by BP values
80
60
40
20
0
1



- The correlation matrix shows that cgsPerDay and currentSmoker are highly correlated.
- Hence, currentSmoker variable is removed from the dataset.

Please refer Appendix A for Source Code.

Insights from EDA:

- Data contains considerable number of Null values and Outliers. Hence, null value treatment and outlier treatment have been done on the Data.
- Highly correlated variable of "currentSmoker" has been removed from data.
- A new variable "BPDifference" has been created and existing "sysBP" and "diaBP" have been removed from data.

After EDA, data is ready for modeling.

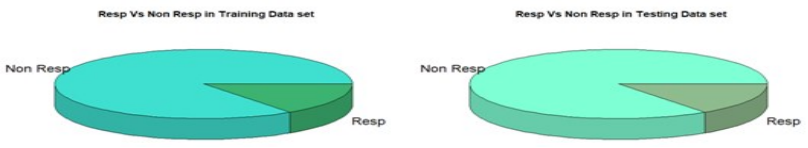
Also, data looks balanced as the ratio of getting coronary heart disease and not getting the disease is about 85% : 15%.

Creating Training and Test Data sets:

- The given data set is divided into Training and Test data sets, with 70:30 proportion.
- The distribution of Responder and Non Responder Class is verified in both the data sets, and ensured it's close to equal.

Following Table summarizes the total number of records as well as percentage distribution of Responder Vs Non Responder Records.

Parameter	Training Data	Testing Data
No of Records	2968	1272
No of features	14	14
% of Responder	84.80%	84.82%
% of Non Responder	15.16%	15.17%



Modeling Techniques and Findings

As per the given data set, it appears to be a classification problem. Hence the following classification Algorithms have been tried on the given data set to build classification models, compare the models and identify the best model which can predict the risk of having coronary heart disease in the next 10 years.

i. Logistic Regression
ii. CART
iii. Random Forest
iv. Artificial Neural Networks
v. Bagging

Model Building – Logistic Regression:

Rationale: Logistic Regression has been considered appropriate regression technique as the dependant variable is binary and explains the relationship between dependant and independent variables. That is, whether the independent variables have an influence on the probability of having coronary heart disease in the next 10 years.

Please refer Appendix A for Source Code for implementing Logistic Regression on Training and Test data sets.

Logistic Regression Output:

Summary of the Logistic Regression model is:

```
> summary(model)

Call:
glm(formula = Train$TenYearCHD ~ ., family = binomial, data = Train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.4483  -0.6003  -0.4324  -0.2923   2.7827

Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)     -7.559738   0.785935  -9.619  < 2e-16 ***
male             0.516611   0.120217   4.297 1.73e-05 ***
age              0.063239   0.007457   8.480  < 2e-16 ***
education        0.002465   0.054875   0.045  0.96417
cigsPerDay       0.020634   0.004684   4.405 1.06e-05 ***
BPMeds           0.301345   0.262326   1.149  0.25066
prevalentStroke  1.322094   0.513181   2.576  0.00999 **
prevalentHyp     0.403189   0.138434   2.913  0.00359 **
diabetes         0.539633   0.278535   1.937  0.05270 .
totChol          0.002899   0.001318   2.199  0.02786 *
BMI              0.011855   0.015035   0.788  0.43042
heartRate       -0.001313   0.004849  -0.271  0.78658
glucose          0.004192   0.004595   0.912  0.36164
BPDifference     0.013414   0.005086   2.637  0.00836 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2529.2  on 2967  degrees of freedom
Residual deviance: 2259.8  on 2954  degrees of freedom
AIC: 2287.8

Number of Fisher Scoring iterations: 5
```

- It can be seen that 7 out of 14 variables are significantly associated to the outcome.
- Out of these 7 variables, 3 variables: male, age and cigsPerDay are highly significant.
- The coefficient of estimate for age is 0.063 which is positive, indicating, as age increases, risk of getting coronary heart disease also increases.

- Similarly, the coefficient of estimates for cigsPerDay, BPDifference are positive indicating, as they increase in value, risk of getting Coronary heart disease also increases.
- The difference between Null deviance and Residual deviance tells us that the model is a good fit. Greater the difference, better the model.

Model Validation – Confusion Matrix:

A confusion matrix describes the performance of a classification model.

Measures of Accuracy of Confusion matrix are:

- Accuracy: How often the model is correct
- Specificity: When it is no, how often does it predict no
- Sensitivity: when it is actually yes, how often it predicts yes.

Confusion Matrix for Training Data set:

```
          Reference
Prediction    0    1
         0 2509    8
         1  426   25

               Accuracy : 0.8538
                 95% CI : (0.8405, 0.8663)
    No Information Rate : 0.9889
    P-Value [Acc > NIR] : 1

                  Kappa : 0.0843

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.85486
            Specificity : 0.75758
         Pos Pred Value : 0.99682
         Neg Pred Value : 0.05543
             Prevalence : 0.98888
         Detection Rate : 0.84535
   Detection Prevalence : 0.84805
      Balanced Accuracy : 0.80622

       'Positive' Class : 0
```

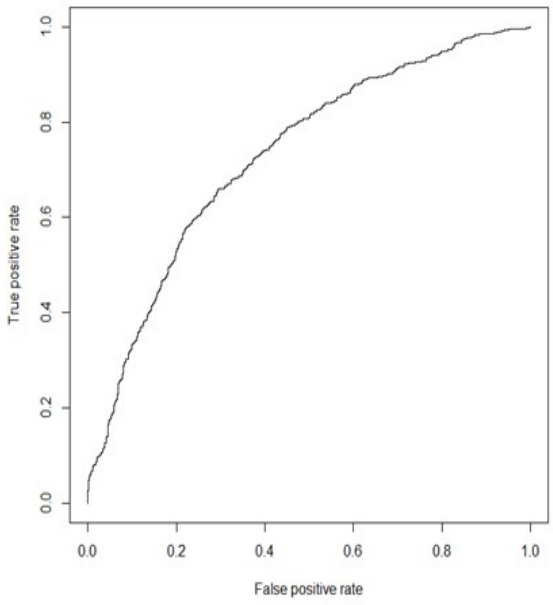
Confusion matrix for Test Data set:

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1074    5
         1  179   14

               Accuracy : 0.8553
                 95% CI : (0.8348, 0.8742)
    No Information Rate : 0.9851
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1078

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.85714
            Specificity : 0.73684
         Pos Pred Value : 0.99537
         Neg Pred Value : 0.07254
             Prevalence : 0.98506
         Detection Rate : 0.84434
   Detection Prevalence : 0.84827
      Balanced Accuracy : 0.79699

       'Positive' Class : 0
```

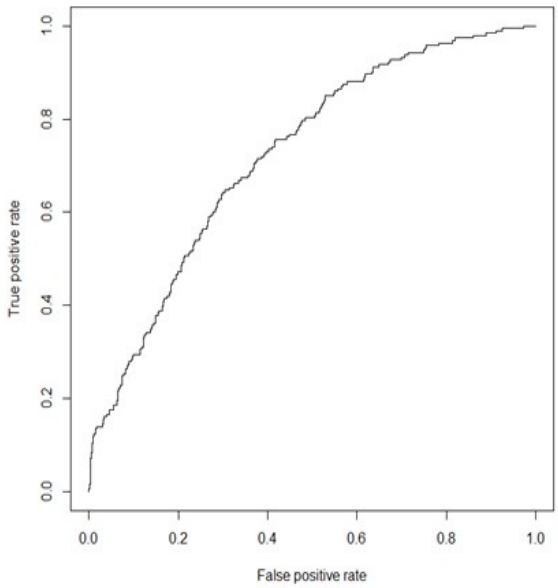
- Accuracy of the training data set is 85%, which means, 85% times, the model can predict the outcome accurately.
- Specificity of the training data set is 76%, which means, 76% times, the model predicts True negative, when it actually is negative.
- Sensitivity of the training data set is 85%,which means, 85% times, the model can predict true positive when it actually is positive.
- Accuracy, Specificity and Sensitivity of the Test data set are 86%, 74% and 86% respectively.

ROC Curve: The ROC curve shows the trade-off between sensitivity (or TPR) and specificity (1 – FPR). Classifiers that give curves closer to the top-left corner indicate a better performance.

ROC Curve for Training Data set



ROC Curve for Test Data set



- As the curves are above the diagonal line, indicating the better performance of the model.

Model Performance measures: The summary of the various model performance measures for both Training and Test Data sets are as shown below.

	Logistic Regression		
Measure	Train DS	Test DS	% Dev
KS	37%	34%	3%
Area under curve	73%	72%	1%
Gini Coefficient	39%	39%	0%
Accuracy	85%	86%	1%
Classification Error Rate	15%	14%	1%

- The AUC values , that is the probability that a randomly chosen positive instance is ranked higher than a randomly chosen negative instance for Training and Test data sets are 73% and 72% respectively. Which indicates the better performance of the model.
- Also the % deviation between Performance measures of Train and Test data sets is also less than 10% indicating the model is a good fit.

Model Building – CART: Decision Trees are commonly used with the objective of creating a model that predicts the value of a target (or dependent variable) based on the values of several input (or independent variables).

If the target variable is a categorical , then the decision tree will be a classification tree, which is used to identify he “class” within which the target variable would likely fall into.

The CART algorithm is structured as a sequence of questions, the answers to which determine what the next question, if any should be. The result of these questions is a tree like structure where the ends are terminal nodes at which point there are no more questions.

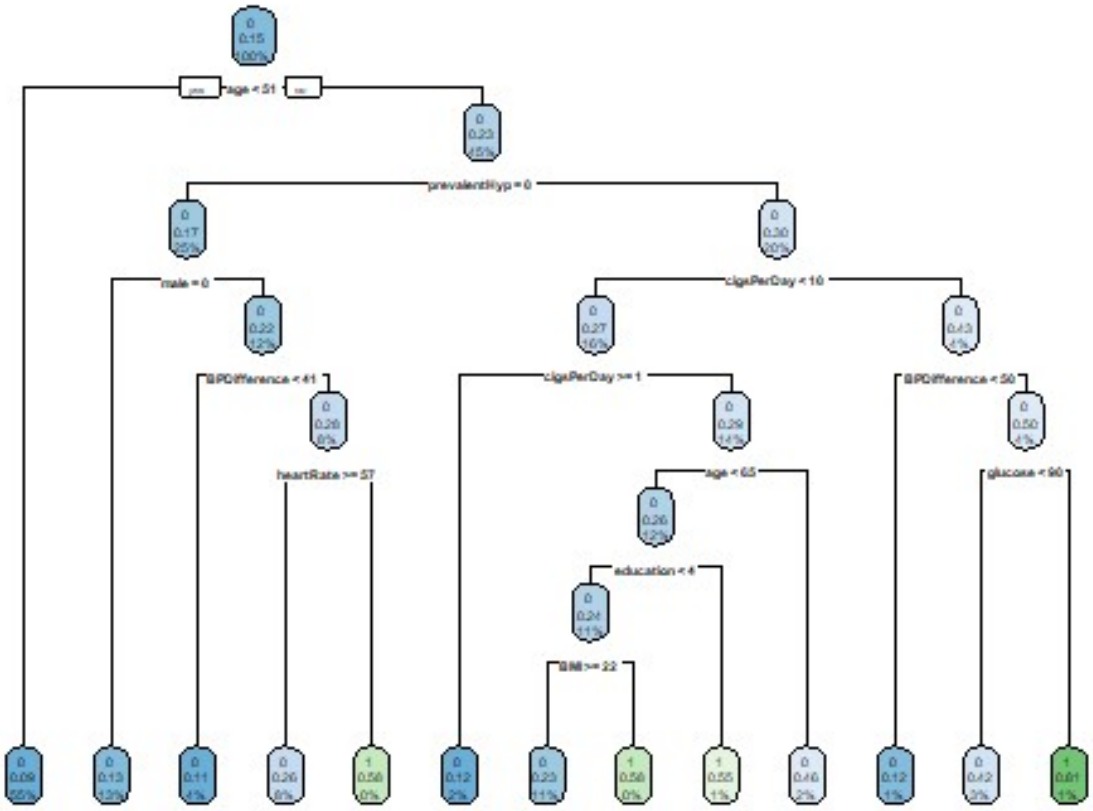
CART Model – Output: The model outcome of Train data set is as follows.

```
node), split, n, loss, yval, (yprob)
      * denotes terminal node

  1) root 2968 451 0 (0.84804582 0.15195418)
    2) age< 50.5 1633 144 0 (0.91181874 0.08818126) *
    3) age>=50.5 1335 307 0 (0.77003745 0.22996255)
      6) prevalentHyp< 0.5 744 129 0 (0.82661290 0.17338710)
       12) male< 0.5 392  50 0 (0.87244898 0.12755102) *
       13) male>=0.5 352  79 0 (0.77556818 0.22443182)
         26) BPDifference< 40.75 109  12 0 (0.88990826 0.11009174) *
         27) BPDifference>=40.75 243  67 0 (0.72427984 0.27572016)
           54) heartRate>=56.5 231  60 0 (0.74025974 0.25974026) *
           55) heartRate< 56.5 12   5 1 (0.41666667 0.58333333) *
      7) prevalentHyp>=0.5 591 178 0 (0.69881557 0.30118443)
       14) cigsPerDay< 9.5 463 123 0 (0.73434125 0.26565875)
         28) cigsPerDay>=0.5 58   7 0 (0.87931034 0.12068966) *
         29) cigsPerDay< 0.5 405 116 0 (0.71358025 0.28641975)
           58) age< 64.5 351  91 0 (0.74074074 0.25925926)
            116) education< 3.5 329  79 0 (0.75987842 0.24012158)
              232) BMI>=21.575 317  72 0 (0.77287066 0.22712934) *
              233) BMI< 21.575 12   5 1 (0.41666667 0.58333333) *
            117) education>=3.5 22  10 1 (0.45454545 0.54545455) *
           59) age>=64.5 54  25 0 (0.53703704 0.46296296) *
       15) cigsPerDay>=9.5 128  55 0 (0.57031250 0.42968750)
         30) BPDifference< 49.75 24   3 0 (0.87500000 0.12500000) *
         31) BPDifference>=49.75 104  52 0 (0.50000000 0.50000000)
           62) glucose< 89.5 83  35 0 (0.57831325 0.42168675) *
           63) glucose>=89.5 21   4 1 (0.19047619 0.80952381) *
```

- For Example: The first part of the output is the total number of 0's and 1's in Train data set.
- The first question is, if the age < 50.5, then 1633people have the chance of not getting coronary heart disease(91%) and 144 people have been wrongly identified as 0(9%).

- The second question is, if age > =50.5, then , 1335people have chance of not getting the disease(77%) and 307people have been wrongly identified as 0(22%).
- Likewise, the question and answers of each predictor variable explains the probabilities of getting the disease or not.

The graphical output of CART model is shown below.



- At the top, it is the overall probability of getting the coronary heart disease. It shows, 15% of the population has the chance of getting Coronary heart disease.

- First node asks, if age < 51. If yes, then, go down to the root's left child node which shows the probability of getting the heart disease is 9% for 50% of the population.
- If, age >= 51, then the probability of getting heart disease is 22% for 50% of the population.
- Now, under the population of age >= 51, if prevalentHyp < 0, then the probability of getting heart disease is 17% for 25% of this population.

Likewise, the decision tree explains the probability of getting the heart disease or not by assigning the probabilities to the nodes of the tree.

The CP table of the decision tree shows there is no need to prune this tree as there is no xerror less than the root node.

```
> CARTModel$cptable
           CP nsplit rel error   xerror       xstd
1 0.005764967      0 1.0000000 1.000000 0.04336321
2 0.002217295      5 0.9711752 1.026608 0.04383145
3 0.001478197      9 0.9623060 1.046563 0.04417584
4 0.000000000     12 0.9578714 1.057650 0.04436470
>
```

CART Model Validation – Confusion Matrix:

Confusion matrices for Training and Test data sets are as shown below.

Confusion Matrix for Train Data set

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 2475   42
         1  386   65

               Accuracy : 0.8558
                 95% CI : (0.8426, 0.8682)
    No Information Rate : 0.9639
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1855

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.8651
            Specificity : 0.6075
         Pos Pred Value : 0.9833
         Neg Pred Value : 0.1441
             Prevalence : 0.9639
         Detection Rate : 0.8339
   Detection Prevalence : 0.8480
      Balanced Accuracy : 0.7363

       'Positive' Class : 0
```

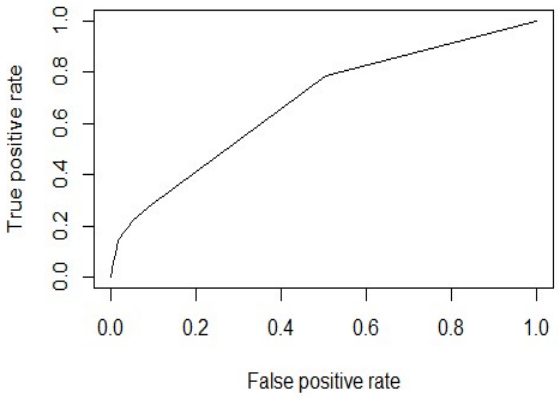
Confusion Matrix for Test Data set

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1056   23
         1  184    9

               Accuracy : 0.8373
                 95% CI : (0.8158, 0.8571)
    No Information Rate : 0.9748
    P-Value [Acc > NIR] : 1

                  Kappa : 0.0385

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.85161
            Specificity : 0.28125
         Pos Pred Value : 0.97868
         Neg Pred Value : 0.04663
             Prevalence : 0.97484
         Detection Rate : 0.83019
   Detection Prevalence : 0.84827
      Balanced Accuracy : 0.56643

       'Positive' Class : 0
```

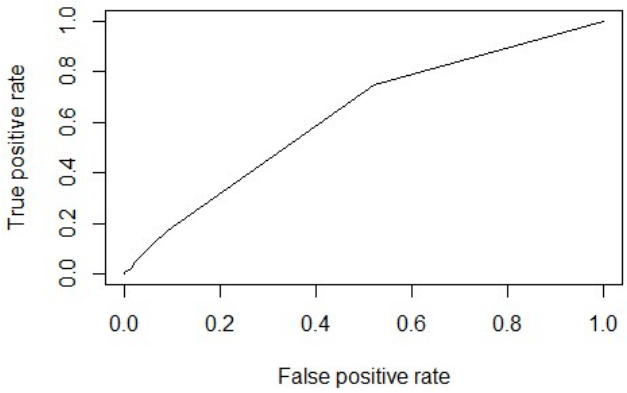
- Accuracy of the training data set is 86%, which means, 86% times, the model can predict the outcome accurately.
- Specificity of the training data set is 61%, which means, 61% times, the model predicts True negative, when it actually is negative.
- Sensitivity of the training data set is 87%,which means, 87% times, the model can predict true positive when it actually is positive.
- Accuracy, Specificity and Sensitivity of the Test data set are 84%, 28% and 85% respectively.

ROC Curve:

ROC Curve for Training data set



ROC Curve for Test data set



As the curves are above the diagonal line, indicating the better performance of the model.

Model Performance measures:

The rank order table for Training data set is :

	deciles	cnt	cnt_resp	cnt_non_resp	rrate	cum_resp	cum_non_resp	cum_rel_resp	cum_rel_non_resp	ks
1	10	366	128	238	35.0%	128	238	28.4%	9.5%	18.92
2	9	1256	227	1029	18.1%	355	1267	78.7%	50.3%	28.37
3	5	1346	96	1250	7.1%	451	2517	100.0%	100.0%	0.00

The rank order table for Test data set is:

	deciles	cnt	cnt_resp	cnt_non_resp	rrate	cum_resp	cum_non_resp	cum_rel_resp	cum_rel_non_resp	ks
1	10	130	33	97	25.4%	33	97	17.1%	9.0%	8.11
2	9	576	111	465	19.3%	144	562	74.6%	52.1%	22.52
3	5	566	49	517	8.7%	193	1079	100.0%	100.0%	0.00

- The response rate in the top decile is 35% for Training dataset and 25% for Test dataset.
- The KS for the top decile is 18.92% for Training dataset and 8% for Test dataset. This indicates, the model is not a good fit.

The summary of the various model performance measures for both Training and Test Data sets are as shown below.

	CART		
Measure	Train DS	Test DS	% Dev
KS	28%	23%	5%
Area under curve	68%	62%	6%
Gini Coefficient	31%	28%	3%
Accuracy	86%	84%	2%
Classification Error Rate	14%	16%	2%

- Although, Accuracy is 86%, other model performance measures of KS, AUC and Gini values are very low , indicating the model is not a good fit.

Please refer Appendix A for Source Code for implementing CART Algorithm on Training and Test data sets.

Model Building – Random Forest: A Supervised Classification Algorithm, as the name suggests, this algorithm creates the forest with a number of trees in random order. In general, the more trees in the forest the more robust the forest looks like. In the same way in the random forest classifier, the higher the number of trees in the forest gives the high accuracy results.

Rationale:

- The same random forest algorithm or the random forest classifier can use for both classification and the regression task.
- Random forest classifier will handle the missing values.

- When we have more trees in the forest, random forest classifier won't over fit the model.
- Can model the random forest classifier for categorical values also.

Random Forest Output:

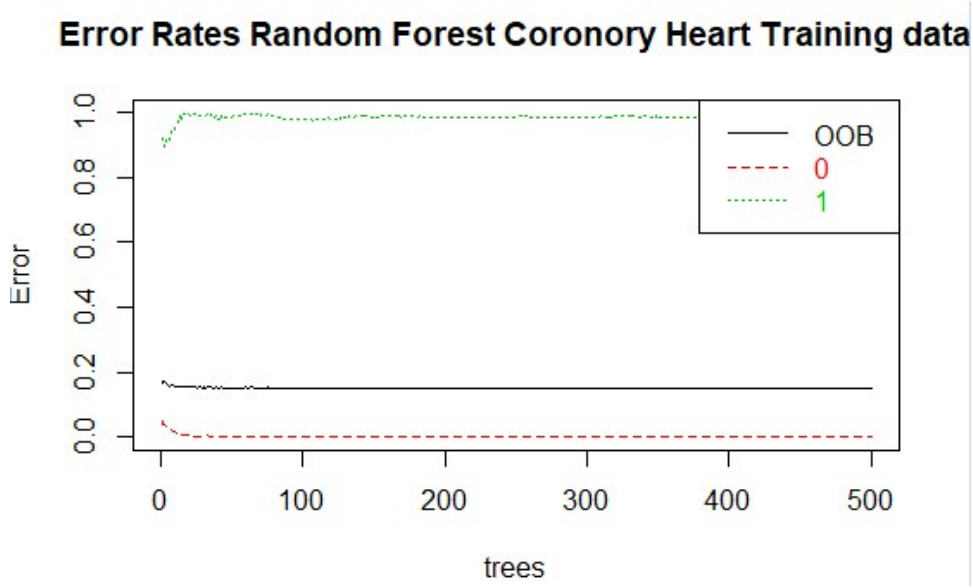
```
Call:
 randomForest(formula = as.factor(TenYearCHD) ~ ., data = rf.train,      ntree = 500, mtry = 7, nodes
ize = 100, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 7

        OOB estimate of  error rate: 15.2%
Confusion matrix:
     0 1 class.error
0 2512 5 0.001986492
1  446 5 0.988913525
```

Out of Bag Error Rate: Out-of-bag (OOB) error, also called out-of-bag estimate, is a method of measuring the prediction error of random forests, boosted decision trees, and other machine learning models utilizing bootstrap aggregating to sub-sample data samples used for training.

Out-of-bag estimates help in avoiding the need for an independent validation dataset.

The Out-of-Bag Estimate of Error Rate for the above Random Forest is 15.2%.

The graphical output for the OOB estimate of error rate is provided below:



The output in tabular form for the OOB estimate of error rate is provided below:

```
> RF$err.rate
             OOB           0         1
  [1,] 0.1724754 0.0499479709 0.9177215
  [2,] 0.1669386 0.0364217252 0.9124088
  [3,] 0.1696509 0.0390913893 0.8944282
  [4,] 0.1673991 0.0320452403 0.9212598
  [5,] 0.1599402 0.0259454705 0.9179104
  [6,] 0.1572913 0.0227752003 0.9166667
  [7,] 0.1594406 0.0201895344 0.9399538
  [8,] 0.1604555 0.0199511401 0.9411765
  [9,] 0.1587573 0.0193158954 0.9391892
 [10,] 0.1574830 0.0156438026 0.9485459
 [11,] 0.1556460 0.0127897682 0.9552573
 [12,] 0.1557211 0.0091816367 0.9732739
 [13,] 0.1558485 0.0091706539 0.9733333
 [14,] 0.1566509 0.0079617834 0.9866667
 [15,] 0.1554806 0.0075576770 0.9800443
```

…

```
[145,] 0.1512803 0.0015891935 0.9866962
[146,] 0.1516173 0.0015891935 0.9889135
[147,] 0.1516173 0.0019864919 0.9866962
[148,] 0.1509434 0.0019864919 0.9822616
[149,] 0.1512803 0.0019864919 0.9844789
[150,] 0.1516173 0.0019864919 0.9866962
[151,] 0.1516173 0.0019864919 0.9866962
[152,] 0.1512803 0.0023837902 0.9822616
[153,] 0.1512803 0.0023837902 0.9822616
```

```
[314,] 0.1516173 0.0023837902 0.9844789
[315,] 0.1519542 0.0023837902 0.9866962
[316,] 0.1516173 0.0023837902 0.9844789
[317,] 0.1516173 0.0023837902 0.9844789
[318,] 0.1512803 0.0019864919 0.9844789
[319,] 0.1512803 0.0019864919 0.9844789
[320,] 0.1516173 0.0019864919 0.9866962
[321,] 0.1516173 0.0019864919 0.9866962
[322,] 0.1516173 0.0019864919 0.9866962
[323,] 0.1516173 0.0019864919 0.9866962
[324,] 0.1516173 0.0019864919 0.9866962
[325,] 0.1516173 0.0019864919 0.9866962
[326,] 0.1516173 0.0019864919 0.9866962
[327,] 0.1516173 0.0019864919 0.9866962
[328,] 0.1516173 0.0019864919 0.9866962
[329,] 0.1516173 0.0019864919 0.9866962
[330,] 0.1516173 0.0019864919 0.9866962
[331,] 0.1512803 0.0019864919 0.9844789
[332,] 0.1516173 0.0019864919 0.9866962
[333,] 0.1516173 0.0019864919 0.9866962
```

It is observed that as the number of tress increases, the OOB error rate starts decreasing till it reaches around 150th tree with OOB = 0.1516 (the minimum value). After this, the OOB doesn't decrease further and remains largely steady. Hence, the optimal number of trees would be 150.

Variable Importance:

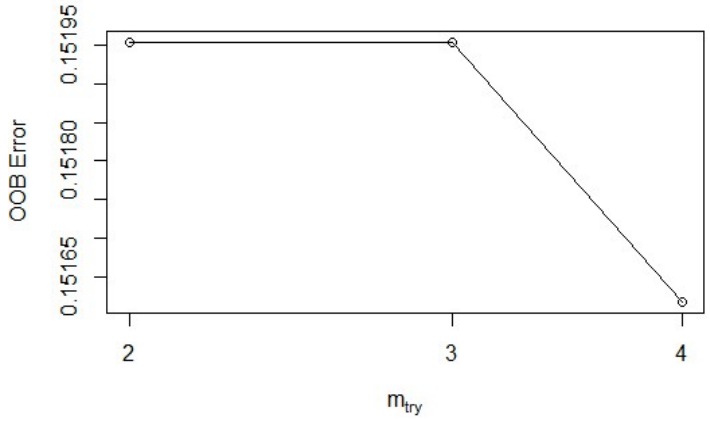
The important variables in Random Forest are:

```
> impVar[order(impVar[,1], decreasing=TRUE),]
                    0     1 MeanDecreaseAccuracy MeanDecreaseGini
cigsPerDay      10.33 -2.57                 9.94             10.43
male             7.87 -4.02                 6.34              6.27
totChol          7.15 -4.75                 5.44             13.91
diabetes         4.42  4.41                 6.46              2.82
BMI              4.15 -0.99                 3.59             20.61
heartRate        4.03 -1.70                 3.01             10.94
BPMeds           3.16  1.86                 4.08              2.62
education       -0.05  1.19                 0.55              2.80
age             -0.09 13.90                 7.44             34.89
BPDifference    -1.09 15.55                 8.14             36.29
prevalentHyp    -1.71  5.93                 1.89              6.82
prevalentStroke -1.91  2.47                -0.36              0.93
glucose         -4.98 10.57                 3.56             13.51
>
```

Optimal mtry Value: In the random forests literature, the number of variables available for splitting at each tree node is referred to as the mtry parameter.

The optimum number of variables is obtained using tuneRF function as follows:

The optimum number of variables is: 4



Random Forest Model Validation – Confusion Matrix:

Confusion Matrix for Train Data

```
          Reference
Prediction    0    1
         0 2517    0
         1  451    0

               Accuracy : 0.848
                 95% CI : (0.8346, 0.8608)
    No Information Rate : 1
    P-Value [Acc > NIR] : 1

                  Kappa : 0

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.848
            Specificity :    NA
         Pos Pred Value :    NA
         Neg Pred Value :    NA
             Prevalence : 1.000
         Detection Rate : 0.848
   Detection Prevalence : 0.848
      Balanced Accuracy :    NA

       'Positive' Class : 0
```

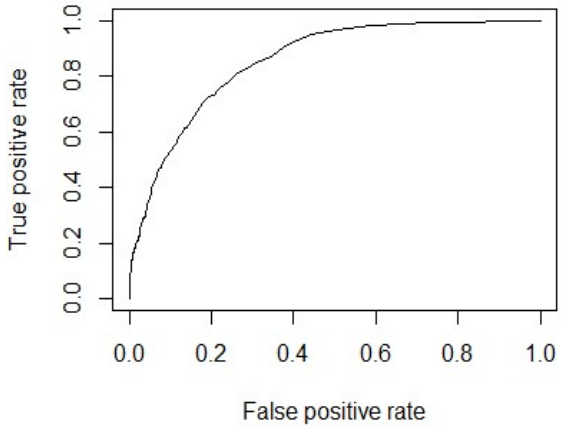
Confusion Matrix for Test Data

```
          Reference
Prediction    0    1
         0 1079    0
         1  193    0

               Accuracy : 0.8483
                 95% CI : (0.8274, 0.8676)
    No Information Rate : 1
    P-Value [Acc > NIR] : 1

                  Kappa : 0

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.8483
            Specificity :     NA
         Pos Pred Value :     NA
         Neg Pred Value :     NA
             Prevalence : 1.0000
         Detection Rate : 0.8483
   Detection Prevalence : 0.8483
      Balanced Accuracy :     NA

       'Positive' Class : 0
```

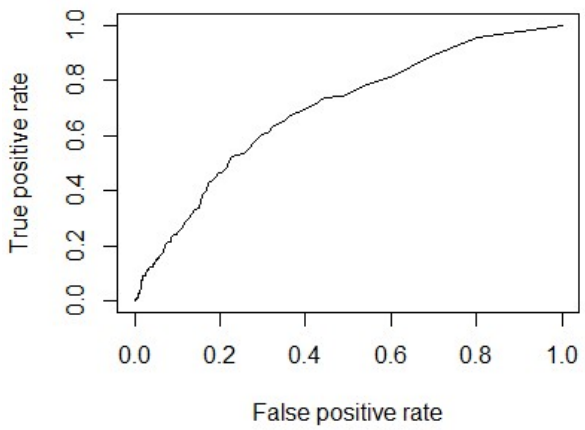
- Accuracy of Training and Test data sets is 85%.

ROC Curve:

ROC Curve of Train Data



ROC Curve of Test Data



- As the curves are above the diagonal line, indicating the better performance of the model.

Model Performance measures:

The rank order table for Training data set is :

	deciles	cnt	cnt_resp	cnt_non_resp	rrate	cum_resp	cum_non_resp	cum_rel_resp	cum_rel_non_resp	ks
1	10	300	169	131	56.0%	169	131	37.0%	5.0%	0.32
2	9	298	101	197	34.0%	270	328	60.0%	13.0%	0.47
3	8	332	78	254	23.0%	348	582	77.0%	23.0%	0.54
4	7	338	45	293	13.0%	393	875	87.0%	35.0%	0.52
5	6	277	35	242	13.0%	428	1117	95.0%	44.0%	0.51
6	5	385	14	371	4.0%	442	1488	98.0%	59.0%	0.39
7	4	364	7	357	2.0%	449	1845	100.0%	73.0%	0.27
8	3	674	2	672	0.0%	451	2517	100.0%	100.0%	0.00

The rank order table for Test data set is:

	deciles	cnt	cnt_resp	cnt_non_resp	rrate	cum_resp	cum_non_resp	cum_rel_resp	cum_rel_non_resp	ks
1	10	128	41	87	32.0%	41	87	21.0%	8.0%	0.13
2	9	129	36	93	28.0%	77	180	40.0%	17.0%	0.23
3	8	137	29	108	21.0%	106	288	55.0%	27.0%	0.28
4	7	130	24	106	18.0%	130	394	67.0%	37.0%	0.30
5	6	144	13	131	9.0%	143	525	74.0%	49.0%	0.25
6	5	144	15	129	10.0%	158	654	82.0%	61.0%	0.21
7	4	103	13	90	13.0%	171	744	89.0%	69.0%	0.20
8	3	133	13	120	10.0%	184	864	95.0%	80.0%	0.15
9	2	224	9	215	4.0%	193	1079	100.0%	100.0%	0.00

- The base line response rate is 15% for both Training and Test data sets, where as response rate in the top two deciles for training data is 56% and 34%, and for Test data is 32% and 28%.
- The KS for the top decile is 47% for Training dataset and 23% for Test dataset.

This indicates, the model is a good fit.

The summary of the various model performance measures for both Training and Test Data sets are as shown below.

	Random Forest		
Measure	Train DS	Test DS	% Dev
KS	55%	31%	24%
Area under curve	86%	69%	17%
Gini Coefficient	73%	69%	4%
Accuracy	85%	85%	0%
Classification Error Rate	15%	15%	0%

- The model observed to perform above expectations on majority of the model performance measures, indicating it to be a good model.

Please refer Appendix A for Source Code for implementing Random Forest Algorithm on Training and Test data sets.

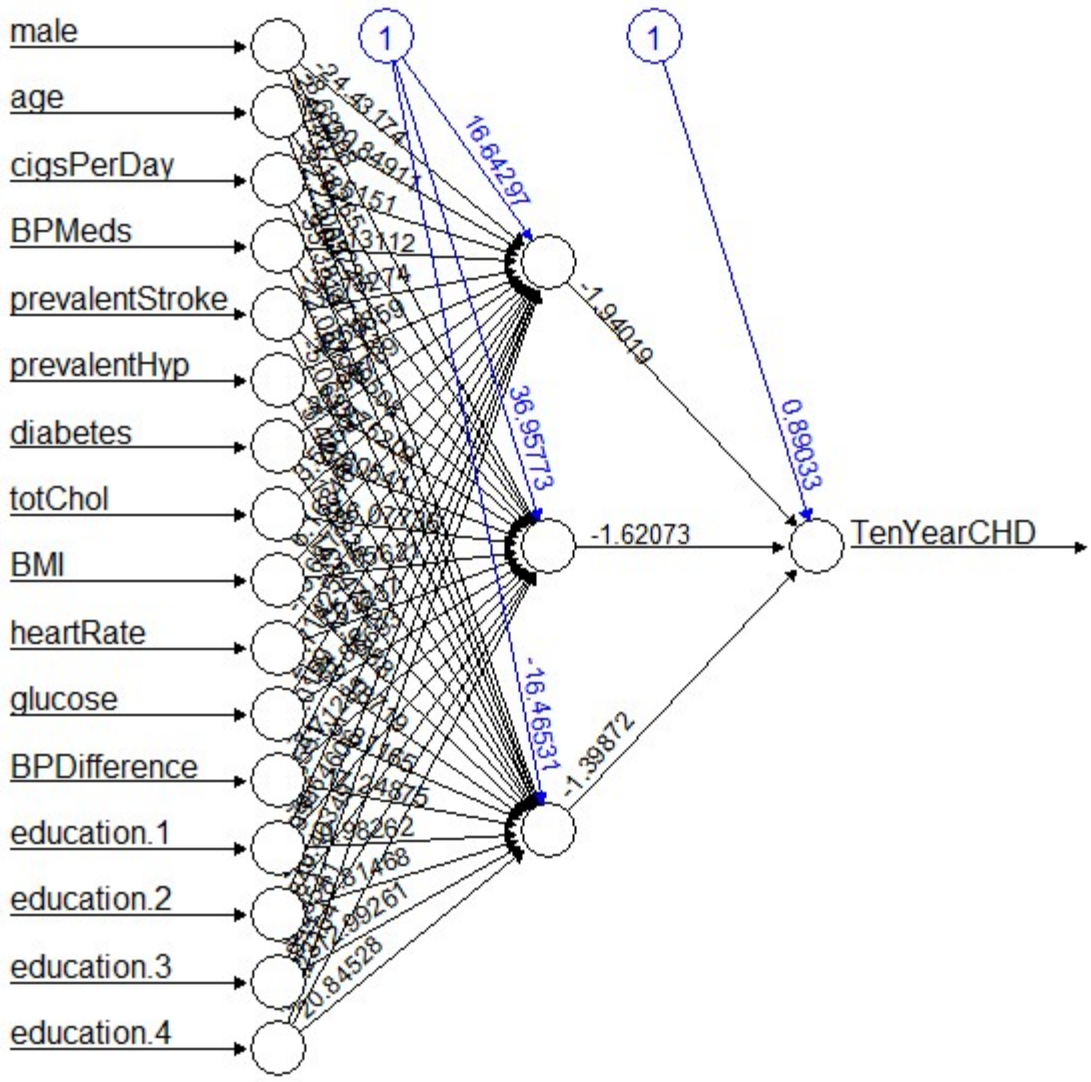
Model Building – Artificial Neural Network: Artificial neural networks are characterized by containing adaptive weights along paths between neurons that can be tuned by a learning algorithm that learns from observed data in order to improve the model. In addition to the learning algorithm itself, one must choose an appropriate cost function.

Rationale: The cost function is what's used to learn the optimal solution to the problem being solved. This involves determining the best values for all of the tunable model parameters, with neuron path adaptive weights being the primary target, along with algorithm tuning parameters such as the learning rate. It's usually done through optimization techniques such as gradient descent or stochastic gradient descent.

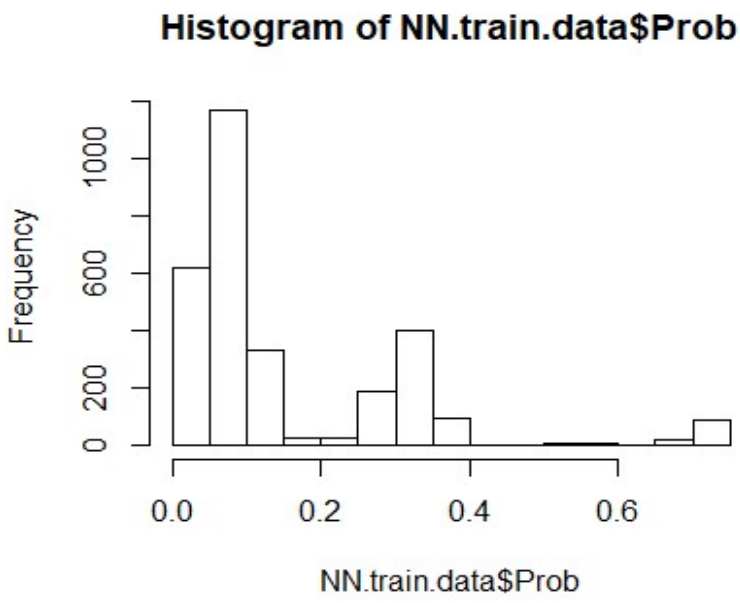
These optimization techniques basically try to make the ANN solution be as close as possible to the optimal solution, which when successful means that the ANN is able to solve the intended problem with high performance.

Artificial Neural Network – Output:

Graphical output of Artificial Neural Network is:



Histogram of Probabilities in Training Data Set:



Artificial Neural Network Model Validation – Confusion Matrix:

Confusion Matrix for Train data set

```
          Reference
Prediction    0    1
         0 2480   26
         1  369   93

               Accuracy : 0.8669
                 95% CI : (0.8542, 0.8789)
    No Information Rate : 0.9599
    P-Value [Acc > NIR] : 1

                  Kappa : 0.2738

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.8705
            Specificity : 0.7815
         Pos Pred Value : 0.9896
         Neg Pred Value : 0.2013
             Prevalence : 0.9599
         Detection Rate : 0.8356
   Detection Prevalence : 0.8443
      Balanced Accuracy : 0.8260

       'Positive' Class : 0
```

Confusion Matrix for Test data set

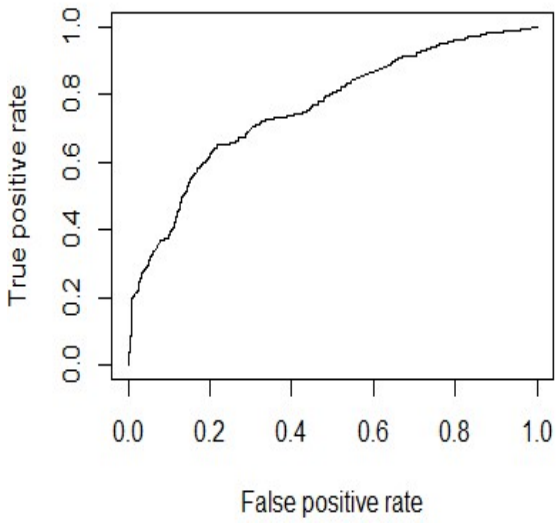
```
          Reference
Prediction    0    1
         0 1052   38
         1  166   16

               Accuracy : 0.8396
                 95% CI : (0.8183, 0.8594)
    No Information Rate : 0.9575
    P-Value [Acc > NIR] : 1

                  Kappa : 0.075

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.86371
            Specificity : 0.29630
         Pos Pred Value : 0.96514
         Neg Pred Value : 0.08791
             Prevalence : 0.95755
         Detection Rate : 0.82704
   Detection Prevalence : 0.85692
      Balanced Accuracy : 0.58000

       'Positive' Class : 0
```

- Accuracy of the Neural Network model for Training and Test data sets are 87% and 84% respectively.
- Specificity for Training and Test Data sets are 78% and 30% respectively.
- Sensitivity for Training and Test data sets are 87% and 86%.

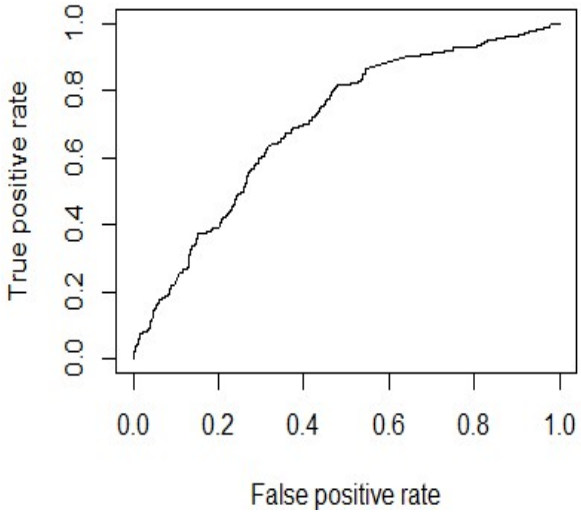
As per the above measures, this model is a good fit.

ROC Curve:

ROC Curve for Train Data set



ROC Curve for Test Data set



- As the curves are above the diagonal line, indicating the better performance of the model.

Model Performance measures:

The rank order table for Training data set is :

	deciles	cnt	cnt_resp	cnt_non_resp	rrate	cum_resp	cum_non_resp	cum_rel_resp	cum_rel_non_resp	ks
1	10	297	151	146	51.0%	151	146	33.0%	6.0%	0.27
2	9	297	86	211	29.0%	237	357	51.0%	14.0%	0.37
3	8	297	63	234	21.0%	300	591	65.0%	24.0%	0.41
4	7	296	35	261	12.0%	335	852	73.0%	34.0%	0.39
5	6	297	19	278	6.0%	354	1130	77.0%	45.0%	0.32
6	5	528	56	472	11.0%	410	1602	89.0%	64.0%	0.25
7	4	65	9	56	14.0%	419	1658	91.0%	66.0%	0.25
8	3	297	19	278	6.0%	438	1936	95.0%	77.0%	0.18
9	2	297	16	281	5.0%	454	2217	98.0%	88.0%	0.10
10	1	297	8	289	3.0%	462	2506	100.0%	100.0%	0.00

The rank order table for Test data set is:

	deciles	cnt	cnt_resp	cnt_non_resp	rrate	cum_resp	cum_non_resp	cum_rel_resp	cum_rel_non_resp	ks
1	10	128	35	93	27.0%	35	93	19.0%	9.0%	0.10
2	9	127	33	94	26.0%	68	187	37.0%	17.0%	0.20
3	8	127	27	100	21.0%	95	287	52.0%	26.0%	0.26
4	7	127	26	101	20.0%	121	388	66.0%	36.0%	0.30
5	6	127	20	107	16.0%	141	495	77.0%	45.0%	0.32
6	5	228	23	205	10.0%	164	700	90.0%	64.0%	0.26
7	4	26	0	26	0.0%	164	726	90.0%	67.0%	0.23
8	3	127	5	122	4.0%	169	848	93.0%	78.0%	0.15
9	2	127	6	121	5.0%	175	969	96.0%	89.0%	0.07
10	1	128	7	121	5.0%	182	1090	100.0%	100.0%	0.00

- The base line response rate is 16% for both Training and Test data sets, where as response rate in the top two deciles for training data is 51% and 29%, and for Test data is 27% and 26%.
- The KS for the top decile is 37% for Training dataset and 20% for Test dataset.

This indicates, the model is a good fit.

The summary of the various model performance measures for both Training and Test Data sets are as shown below.

	Artificial Neural Networks		
Measure	Train DS	Test DS	% Dev
KS	43%	34%	9%
Area under curve	76%	70%	6%
Gini Coefficient	52%	51%	1%
Accuracy	87%	84%	3%
Classification Error Rate	13%	16%	3%

- The model observed to perform above expectations on majority of the model performance measures, indicating it to be a good model.

Please refer Appendix A for Source Code for implementing Artificial Neural Network Algorithm on Training and Test data sets.

Model Building – Bagging: Ensembling is a technique of combining two or more algorithms of similar or dissimilar types called base learners. This is done to make a more robust system which incorporates the predictions from all the base learners. Bagging is one of the ensembling modeling and has been implemented on the given Coronary Heart Risk data set.

Rationale: Bagging is also referred to as bootstrap aggregation. To understand bagging, we first need to understand bootstrapping. Bootstrapping is a sampling technique in which we choose 'n' observations or rows out of the original dataset of 'n' rows as well. But the key is that each row is selected with replacement from the original dataset so that each row is equally likely to be selected in each iteration.

Bagging is mainly done to reduce the variance.

Bagging Model Validation - Confusion Matrix:

Confusion Matrix for Train data

```
> with(train.data, table(TenYearCHD, predict.class))
           predict.class
TenYearCHD    0
         0 2506
         1  462
```

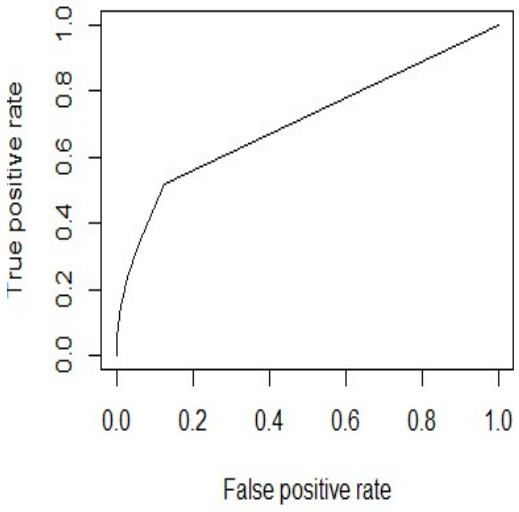
Confusion Matrix for Test data

```
> with(test.data, table(TenYearCHD, predict.class))
           predict.class
TenYearCHD    0
         0 1090
         1  182
```

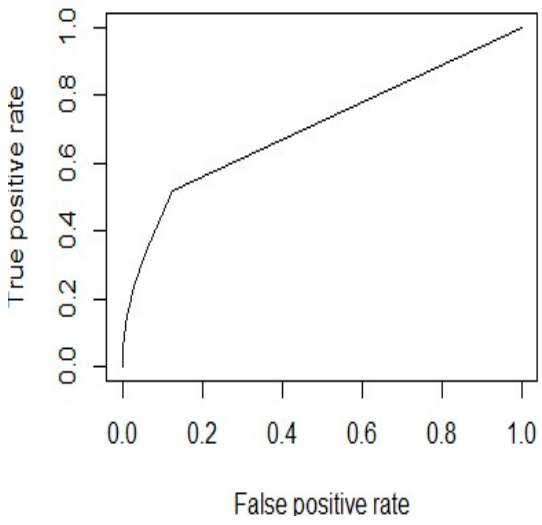
- Accuracy of Bagging model for Train and Test data sets are 84% and 86% respectively.

ROC Curve:

ROC Curve for Train data



ROC Curve for Test data



- As the curves are above the diagonal line, indicating the better performance of the model.

Model Performance measures:

The rank order table for Training data set is :

	deciles	cnt	cnt_resp	cnt_non_resp	rrate	cum_resp	cum_non_resp	cum_rel_resp	cum_rel_non_resp	ks
1	10	551	240	311	43.6%	240	311	52.0%	12.4%	39.54
2	9	2417	222	2195	9.2%	462	2506	100.0%	100.0%	0.00

The rank order table for Test data set is:

	deciles	cnt	cnt_resp	cnt_non_resp	rrate	cum_resp	cum_non_resp	cum_rel_resp	cum_rel_non_resp	ks
1	10	131	37	94	28.2%	37	94	20.3%	8.6%	11.71
2	9	1141	145	996	12.7%	182	1090	100.0%	100.0%	0.00

- The response rate in the top two deciles for training data is 44% and 9%, and for Test data is 28% and 12%.
- The KS for the top decile is 40% for Training dataset and 12% for Test dataset.

This indicates, the model is a good fit.

The summary of the various model performance measures for both Training and Test Data sets are as shown below.

	Bagging		
Measure	Train DS	Test DS	% Dev
KS	40%	18%	22%
Area under curve	70%	60%	10%
Gini Coefficient	50%	50%	0%
Accuracy	84%	86%	2%
Classification Error Rate	16%	14%	2%

- The model observed to perform above expectations on majority of the model performance measures, indicating it to be a good model.

Please refer Appendix A for Source Code for implementing Bagging Algorithm on Training and Test data sets.

Model Comparison

Summary of the all implemented models is as shown below.

	Logistic Regression			CART			Random Forest			Artificial Neural Networks			Bagging		
Measure	Train DS	Test DS	% Dev	Train DS	Test DS	% Dev	Train DS	Test DS	% Dev	Train DS	Test DS	% Dev	Train DS	Test DS	% Dev
KS	37%	34%	3%	28%	23%	5%	55%	31%	24%	43%	34%	9%	40%	18%	22%
Area under curve	73%	72%	1%	68%	62%	6%	86%	69%	17%	76%	70%	6%	70%	60%	10%
Gini Coefficient	39%	39%	0%	31%	28%	3%	73%	69%	4%	52%	51%	1%	50%	50%	0%
Accuracy	85%	86%	1%	86%	84%	2%	85%	85%	0%	87%	84%	3%	84%	86%	2%
Classification Error Rate	15%	14%	1%	14%	16%	2%	15%	15%	0%	13%	16%	3%	16%	14%	2%

Conclusion:

- Logistic Regression and CART method have given poorer performance compared to Random Forest, Artificial Neural Networks and Bagging.
- The Random Forest method has the best performance among all the three models. The percentage deviation between Training and Testing Dataset also is reasonably under control, suggesting a robust model.
- Neural Network has given relatively secondary performance compared to Random Forest, however, better than CART. And Logistic Regression.
- Bagging did not show much improvement over the best performer Random Forest.

Hence, Random Forest is the best performer for predicting Coronary Heart disease.

Insights from Analysis

- Age, gender, cigsperday and BPDifference are the significant features in the given data set.
- Consuming more cigarettes has high risk of getting heart disease.
- Males have high risk of getting heart disease compared to females.
- As the age increases, risk of getting heart disease also increases.
- The higher the BPDifference, more risk of getting the heart disease.

Business Recommendations:

- People getting older must screen themselves often.
- Avoid smoking.
- Control BP.

Appendix A – Source Code

```
#=============================================================================
# Data Analysis - Coronary Heart Risk
#=============================================================================
rm(list = ls(all.names = TRUE))
#Environment Set up and Data Import
#Set up working Directory
setwd("C:/Users/Radhika/Desktop/R Programming/Project_Capstone")
getwd()
#install.packages("readxl")
library(readxl)
#Read the input file
heartRiskData=read.csv("Coronary_heart_risk_study.csv")
attach(heartRiskData)
str(heartRiskData)
summary(heartRiskData)
dim(heartRiskData)
head(heartRiskData)
original = heartRiskData
#Null Value treatment
sum(is.na(heartRiskData))
#install.packages("VIM")
library(VIM)
impute.data = kNN(heartRiskData)
summary(impute.data)
dim(impute.data)
impute.data = subset(impute.data, select = male:TenYearCHD)
head(impute.data)
#Creating new BP Column
BPDifference = impute.data$sysBP - impute.data$diaBP
summary(BPDifference)
#Add the new column in the imputed data
impute.data = cbind(impute.data, BPDifference)
#Remove existing columns of sysBP and diaBP
impute.data = subset(impute.data, select = -c(sysBP,diaBP) )
dim(impute.data)
head(impute.data)
```

```r
#check if data is balanced or not
prop.table(table(impute.data$TenYearCHD))
#Identifying outliers
boxplot(impute.data$education)
boxplot(impute.data$age, xlab="Age", main="Exploring Age attribute", col="yellow")
boxplot(impute.data$totChol, xlab="Total Cholestrol",
        main="Exploring totChol Attribute", col="yellow")#outliers present
boxplot(impute.data$BPDifference, xlab="Difference in BP values",
        main="Exploring BP attribute", col="yellow")#Outliers present
boxplot(impute.data$BMI, xlab="BMI",
        main="Exploring BMI attribute", col="yellow")#outliers present
boxplot(impute.data$heartRate, xlab="heartRate",
        main="Exploring heartRate attribute", col="yellow")#outliers present
boxplot(impute.data$glucose, xlab="glucose",
        main="Exploring glucose attribute", col="yellow")#outliers present
#Outlier Treatment
#outliers for totChol
#Identifying the outlier boundaries
IQRage = IQR(impute.data$totChol)
LLage = quantile(impute.data$totChol,0.25) - 1.5*IQRage
ULage = quantile(impute.data$totChol,0.75) + 1.5*IQRage
totCholOut = subset(impute.data, totChol < LLage | totChol > ULage)
totCholOut
dim(totCholOut)
# Outlier treatment for totChol: Capping
totCholWout = subset(impute.data, totChol >= LLage & totChol <= ULage)
dim(totCholWout)
maxVal=max(totCholWout$totChol)
minVal=min(totCholWout$totChol)
summary(impute.data$totChol)
impute.data$totChol[impute.data$totChol > maxVal] = maxVal
impute.data$totChol[impute.data$totChol < minVal] = minVal
summary(impute.data$totChol)
boxplot(impute.data$totChol, xlab="Total Cholestrol",
        main="Exploring totChol Attribute", col="yellow")
```

```r
#outliers for BPDifference
#Identifying the outlier boundaries
IQRage = IQR(impute.data$BPDifference)
LLage = quantile(impute.data$BPDifference,0.25) - 1.5*IQRage
ULage = quantile(impute.data$BPDifference,0.75) + 1.5*IQRage
sysBPOut = subset(impute.data, impute.data$BPDifference < LLage | BPDifference > ULage)
sysBPOut
dim(sysBPOut)
# Outlier treatment for BP: Capping
BPWout = subset(impute.data, impute.data$BPDifference >= LLage & impute.data$BPDifference <= ULage)
dim(BPWout)
maxVal=max(BPWout$BPDifference)
minVal = min(BPWout$BPDifference)
summary(impute.data$BPDifference)
impute.data$BPDifference[impute.data$BPDifference > maxVal] = maxVal
impute.data$BPDifference[impute.data$BPDifference < minVal] = minVal
summary(impute.data$BPDifference)
boxplot(impute.data$BPDifference, xlab="Difference in BP values",
        main="Exploring BP attribute", col="yellow")
#outliers for BMI
#Identifying the outlier boundaries
IQRage = IQR(impute.data$BMI)
LLage = quantile(impute.data$BMI,0.25) - 1.5*IQRage
ULage = quantile(impute.data$BMI,0.75) + 1.5*IQRage
BMIOut = subset(impute.data, impute.data$BMI < LLage | impute.data$BMI > ULage)
head(BMIOut)
dim(BMIOut)
# Outlier treatment for BMI: Capping
BMIWout = subset(impute.data, impute.data$BMI >= LLage & impute.data$BMI <= ULage)
dim(BMIWout)
maxVal = max(BMIWout$BMI)
minVal = min(BMIWout$BMI)
summary(impute.data$BMI)
impute.data$BMI[impute.data$BMI > maxVal] = maxVal
impute.data$BMI[impute.data$BMI < minVal] = minVal
summary(impute.data$BMI)
```

```
boxplot(impute.data$BMI, xlab="BMI",
        main="Exploring BMI attribute", col="yellow")
#outliers for heartrate
#Identifying the outlier boundaries
IQRage = IQR(impute.data$heartRate)
LLage = quantile(impute.data$heartRate,0.25) - 1.5*IQRage
ULage = quantile(impute.data$heartRate,0.75) + 1.5*IQRage
hrOut = subset(impute.data, impute.data$heartRate < LLage | impute.data$heartRate > ULage)
head(hrOut)
dim(hrOut)
# Outlier treatment for heartrate: Capping
hrWout = subset(impute.data, impute.data$heartRate >= LLage & impute.data$heartRate <= ULage)
dim(hrWout)
maxVal = max(hrWout$heartRate)
minVal = min(hrWout$heartRate)
summary(impute.data$heartRate)
impute.data$heartRate[impute.data$heartRate > maxVal] = maxVal
impute.data$heartRate[impute.data$heartRate < minVal] = minVal
summary(impute.data$heartRate)
boxplot(impute.data$heartRate, xlab="heartRate",
        main="Exploring heartRate attribute", col="yellow")
#outliers for glucose
#Identifying the outlier boundaries
IQRage = IQR(impute.data$glucose)
LLage = quantile(impute.data$glucose,0.25) - 1.5*IQRage
ULage = quantile(impute.data$glucose,0.75) + 1.5*IQRage
glucoseOut = subset(impute.data, impute.data$glucose < LLage | impute.data$glucose > ULage)
head(glucoseOut)
dim(glucoseOut)
# Outlier treatment for glucose: Capping
glucoseWout = subset(impute.data, impute.data$glucose >= LLage & impute.data$glucose <= ULage)
dim(glucoseWout)
maxVal = max(glucoseWout$glucose)
minVal = min(glucoseWout$glucose)
summary(impute.data$glucose)
```

```
impute.data$glucose[impute.data$glucose > maxVal] = maxVal
impute.data$glucose[impute.data$glucose < minVal] = minVal
summary(impute.data$glucose)
boxplot(impute.data$glucose, xlab="glucose",
        main="Exploring glucose attribute", col="yellow")
#
#Univariate Analysis
#Age
hist(impute.data$age, col="blue")
#For Education
hist(impute.data$education, col=c("blue"))
counts <- table(impute.data$education, impute.data$male)
barplot(counts, beside=TRUE, legend = rownames(counts), xlab = "Male/Female", ylab = "Frequency")
impute.data%>%count(education)
#Current Smoker
counts = table(impute.data$currentSmoker, impute.data$male)
counts
#cigsPerDay
hist(impute.data$cigsPerDay, col=c("blue"))
#BPMeds
counts = table(impute.data$BPMeds, impute.data$male)
counts
barplot(counts, beside=TRUE, legend = rownames(counts), xlab = "Male/Female",
        ylab = "Frequency", col=c("blue","pink"), main ="BP Medicines taken by Male and Female")
#prevalentStroke
counts = table(impute.data$prevalentStroke, impute.data$male)
counts
barplot(counts, beside=TRUE, legend = rownames(counts), xlab = "Male/Female",
        ylab = "Frequency", col=c("blue","pink"), main = "Prevalent Stroke Count by Male and Female")

#prevalentHyp
counts = table(impute.data$prevalentHyp, impute.data$male)
counts
```

```
barplot(counts, beside=TRUE, legend = rownames(counts), xlab = "Male/Female",
        ylab = "Frequency", col=c("blue","pink"),
        main = "Prevalent Hypertension Count by Male and Female")

#Diabetes
counts = table(impute.data$diabetes, impute.data$male)
counts
barplot(counts, beside=TRUE, legend = rownames(counts), xlab = "Male/Female",
        ylab = "Frequency", col=c("blue","pink"),
        main = "Diabets Count by Male and Female")
#male
hist(impute.data$male, col="blue")
#totchol
hist(impute.data$totChol , col="blue")
#BP
hist(impute.data$BPDifference, col = "blue")
#BMI
hist(impute.data$BMI, col="blue")
#heartrate
hist(impute.data$heartRate, col="blue")
#glucose
hist(impute.data$glucose, col="blue")
hist(impute.data$cigsPerDay, col="blue")
#Bivariate Analysis
library(ggplot2)
ggplot(impute.data, aes(x = impute.data$age, y = impute.data$glucose)) + geom_point()
ggplot(impute.data, aes(x = impute.data$age, y = impute.data$heartRate)) + geom_point()
ggplot(impute.data, aes(x = impute.data$age, y = impute.data$BPDifference)) + geom_point()
ggplot(impute.data, aes(x = impute.data$age, y = impute.data$BMI)) + geom_point()
ggplot(impute.data, aes(x = impute.data$age, y = impute.data$totChol)) + geom_point()
boxplot(impute.data$BPDifference  ~ impute.data$BPMeds, main="BP medicines by BP values", col="yellow")
#checking correlation between variables
library(caTools)
library(caret)
library(ROCR)
library(corrplot)
```

```r
library(car)
impute.data.cor = subset(impute.data, select = -c(TenYearCHD))
length(impute.data.cor)
names(impute.data.cor)
cor.matrix = cor(impute.data.cor)
correlation=corrplot(cor.matrix,method = "number",type = "lower")
highlyCorrelated <- findCorrelation(cor.matrix, cutoff=0.7)
print(highlyCorrelated)
highlyCorCol <- colnames(impute.data.cor)[highlyCorrelated]
highlyCorCol
impute.data.cor = subset(impute.data.cor, select = -c(currentSmoker))
head(impute.data.cor)
```

```r
#Logistic Regression
spl = sample.split(impute.data.cor$TenYearCHD, SplitRatio = 0.7)
Train = subset(impute.data.cor, spl==T)
Test = subset(impute.data.cor, spl==F)
dim(Train)
dim(Test)
#Check if distribution of partition data is correct
prop.table(x, margin = NULL)
ain$TenYearCHD)))
#0.8480458 0.1519542
prop.table((table(Test$TenYearCHD)))
#
# So the data is well distributed in the training and validation sets
install.packages("plotrix")
library(plotrix)
par(mfrow=c(1,2))
pie3D(prop.table((table(Train$TenYearCHD))),
      main='Resp Vs Non Resp in Training Data set',
      #explode=0.1,
      labels=c("Non Resp", "Resp"),
      col = c("Turquoise", "Medium Sea Green")
)
pie3D(prop.table((table(Test$TenYearCHD))),
      main='Resp Vs Non Resp in Testing Data set',
      #explode=0.1,
      labels=c("Non Resp", "Resp"),
      col = c("Aquamarine", "Dark Sea Green")
)

model=glm(Train$TenYearCHD~.,data = Train,family = binomial)
summary(model)
vif(model)
```

```
predTrain=predict(model, newdata = Train,type = "response")
#Creation of confusion matrix to assess model performance measures
Train$Class = ifelse(predTrain>0.5,1,0)
class(Train$Class)
# Confusion matrix for Train data
#install.packages("caret")
library(caret)
confusionMatrix(as.factor(Train$TenYearCHD), as.factor(Train$Class))
#Creation of Confusion matrix for Test data
predTest=predict(model, newdata = Test,type = "response")
Test$Class = ifelse(predTest>0.5,1,0)
head(Test)
confusionMatrix(as.factor(Test$TenYearCHD), as.factor(Test$Class))
#Performance measures for Training Data
pred.Train = ROCR::prediction(predTrain, Train$TenYearCHD)
perf.Train = performance(pred.Train, "tpr", "fpr")
plot(perf.Train)

KS <- max(attr(perf.Train, 'y.values')[[1]]-attr(perf.Train, 'x.values')[[1]])
auc <- performance(pred.Train,"auc");
auc <- as.numeric(auc@y.values)
library(ineq)
gini = ineq(predTrain, type="Gini")
KS
auc
gini
#Performance measures for Testing Data
pred.Test = ROCR::prediction(predTest, Test$TenYearCHD)
perf.Test = performance(pred.Test, "tpr", "fpr")
plot(perf.Test)
KS <- max(attr(perf.Test, 'y.values')[[1]]-attr(perf.Test, 'x.values')[[1]])
auc <- performance(pred.Test,"auc");
```

```
auc <- as.numeric(auc@y.values)
gini = ineq(predTest, type="Gini")
KS
auc
gini
#CART model
#set.seed(50)
#Get Training and Test data
spl = sample.split(impute.data.cor$TenYearCHD, SplitRatio = 0.7)
Train = subset(impute.data.cor, spl==T)
Test = subset(impute.data.cor, spl==F)
dim(Train)
dim(Test)
library(rpart)
#install.packages("rpart.plot")
library(rpart.plot)
r.ctrl = rpart.control(minsplit = 100, minbucket = 10, cp=0, xval=10)
CARTModel = rpart(TenYearCHD~.,data=Train, method="class", control = r.ctrl)
CARTModel
rpart.plot(CARTModel)
attributes(CARTModel)
CARTModel$cptable
plotcp(CARTModel)
CARTModel$cptable[which.min(CARTModel$cptable[,"xerror"]),"CP"]
ptree = prune(CARTModel, 0.005764967, "CP")
rpart.plot(ptree)
#CART Validation on Train Data
predTrain = predict(CARTModel, newdata = Train)
predTest=predict(CARTModel, newdata = Test)
#Confusion matrix for Train data
library(caret)
Train$Class = ifelse(predTrain>0.5,1,0)
class(Train$Class)
confusionMatrix(as.factor(Train$TenYearCHD), as.factor(Train$Class[,2]))
#Confusion matrix for Test data
```

```r
#Confusion matrix for Test data
Test$Class = ifelse(predTest>0.5,1,0)
confusionMatrix(as.factor(Test$TenYearCHD), as.factor(Test$Class[,2]))
#
library(ROCR)
#Validation on Train data
predROC1 = ROCR::prediction(predTrain[,2], Train$TenYearCHD)
perf1 = performance(predROC1, "tpr", "fpr")
plot(perf1)
as.numeric(performance(predROC1, "auc")@y.values)
#Validation on Test Data
predROC2 = ROCR::prediction(predTest[,2], Test$TenYearCHD)
perf = performance(predROC2, "tpr", "fpr")
plot(perf)
as.numeric(performance(predROC2, "auc")@y.values)
# Measure Model Performance
Train$predict.class <- predict(CARTModel, Train, type="class")
Train$predict.score <- predict(CARTModel, Train, type="prob")
 head(Train)
## deciling code
decile <- function(x){
  deciles <- vector(length=10)
  for (i in seq(0.1,1,.1)){
    deciles[i*10] <- quantile(x, i, na.rm=T)
  }
  return (
    ifelse(x<deciles[1], 1,
           ifelse(x<deciles[2], 2,
                  ifelse(x<deciles[3], 3,
                         ifelse(x<deciles[4], 4,
                                ifelse(x<deciles[5], 5,
                                       ifelse(x<deciles[6], 6,
                                              ifelse(x<deciles[7], 7,
                                                     ifelse(x<deciles[8],
                                                            8,
                                                            ifelse(x<deciles[9], 9, 10
                                                            ))))))))))
}
class(Train$predict.score)
## deciling
Train$deciles <- decile(Train$predict.score[,2])
View(Train)
## Ranking code
##install.packages("data.table")
##install.packages("scales")
library(data.table)
library(scales)
tmp_DT = data.table(Train)
rank <- tmp_DT[, list(
  cnt = length(TenYearCHD),
  cnt_resp = sum(TenYearCHD),
  cnt_non_resp = sum(TenYearCHD == 0)) ,
  by=deciles][order(-deciles)]
rank$rrate <- round(rank$cnt_resp / rank$cnt,4);
```

```
rank$cum_resp <- cumsum(rank$cnt_resp)
rank$cum_non_resp <- cumsum(rank$cnt_non_resp)
rank$cum_rel_resp <- round(rank$cum_resp / sum(rank$cnt_resp),4);
rank$cum_rel_non_resp <- round(rank$cum_non_resp /
                                 sum(rank$cnt_non_resp),4);
rank$ks <- abs(rank$cum_rel_resp - rank$cum_rel_non_resp) * 100;
rank$rrate <- percent(rank$rrate)
rank$cum_rel_resp <- percent(rank$cum_rel_resp)
rank$cum_rel_non_resp <- percent(rank$cum_rel_non_resp)
View(rank)
#install.packages("ROCR")
#install.packages("ineq")
library(ROCR)
library(ineq)
pred <- prediction(Train$predict.score[,2], Train$TenYearCHD)
perf <- performance(pred, "tpr", "fpr")
plot(perf)
KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
auc <- performance(pred,"auc");
auc <- as.numeric(auc@y.values)
gini = ineq(Train$predict.score[,2], type="Gini")
with(Train, table(TenYearCHD, predict.class))
auc
KS
gini
## Scoring Holdout sample
Test$predict.class <- predict(CARTModel, Test, type="class")
Test$predict.score <- predict(CARTModel, Test, type="prob")
Test$deciles <- decile(Test$predict.score[,2])
View(Test)
# Ranking code
tmp_DT = data.table(Test)
h_rank <- tmp_DT[, list(
  cnt = length(TenYearCHD),
  cnt_resp = sum(TenYearCHD),
```

```
  cnt_non_resp = sum(TenYearCHD == 0)) ,
  by=deciles][order(-deciles)]
h_rank$rrate <- round(h_rank$cnt_resp / h_rank$cnt,4);
h_rank$cum_resp <- cumsum(h_rank$cnt_resp)
h_rank$cum_non_resp <- cumsum(h_rank$cnt_non_resp)
h_rank$cum_rel_resp <- round(h_rank$cum_resp / sum(h_rank$cnt_resp),4);
h_rank$cum_rel_non_resp <- round(h_rank$cum_non_resp /
                                   sum(h_rank$cnt_non_resp),4);
h_rank$ks <- abs(h_rank$cum_rel_resp - h_rank$cum_rel_non_resp)*100;
h_rank$rrate <- percent(h_rank$rrate)
h_rank$cum_rel_resp <- percent(h_rank$cum_rel_resp)
h_rank$cum_rel_non_resp <- percent(h_rank$cum_rel_non_resp)
View(h_rank)
pred <- prediction(Test$predict.score[,2], Test$TenYearCHD)
perf <- performance(pred, "tpr", "fpr")
KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
auc <- performance(pred,"auc");
auc <- as.numeric(auc@y.values)
gini = ineq(Test$predict.score[,2], type="Gini")
with(Test, table(TenYearCHD, predict.class))
auc
KS
gini
#
################################################################################
###############Random Forest################
################################################################################
#install.packages("randomForest")
library(randomForest)
#
# Copy datasets for RF
rf.train = subset(impute.data.cor, spl==T)
rf.test = subset(impute.data.cor, spl==F)
dim(rf.train)
```

```
dim(rf.test)
names(rf.train)
str(rf.train)
# Random Forest
RF = randomForest( as.factor(TenYearCHD) ~ .,
                   data = rf.train,
                   ntree = 500, mtry = 7, nodesize = 100,
                   importance = TRUE )
print(RF)
#
dev.off()
plot(RF, main="")
legend("topright", c("OOB", "0", "1"), text.col=1:6, lty=1:3, col=1:3)
title(main="Error Rates Random Forest Coronory Heart Training data")
RF$err.rate
# List the importance of the variables.
impVar <- round(randomForest::importance(RF), 2)
impVar[order(impVar[,3], decreasing=TRUE),]
impVar[order(impVar[,4], decreasing=TRUE),]
impVar[order(impVar[,1], decreasing=TRUE),]
# Tuning Random Forest
tRF <- tuneRF(x = rf.train[,-c(14)],
              y=as.factor(rf.train$TenYearCHD),
              mtryStart = 3, # Approx. Sqrt of Tot. No of Variables
              ntreeTry=150,
              stepFactor = 1.5,
              improve = 0.1,
              trace=TRUE,
              plot = TRUE,
              doBest = TRUE,
              nodesize = 100,
              importance=TRUE
)
# Scoring syntax
rf.train$predict.class <- predict(tRF, rf.train, type="class")
```

```
head(rf.train)
# class(rf.train$predict.score)
rf.train$deciles <- decile(rf.train$predict.score[,2])
View(rf.train)
library(data.table)
tmp_DT = data.table(rf.train)
rank <- tmp_DT[, list(
  cnt = length(TenYearCHD),
  cnt_resp = sum(TenYearCHD),
  cnt_non_resp = sum(TenYearCHD == 0)) ,
  by=deciles][order(-deciles)]
rank$rrate <- round (rank$cnt_resp / rank$cnt,2);
rank$cum_resp <- cumsum(rank$cnt_resp)
rank$cum_non_resp <- cumsum(rank$cnt_non_resp)
rank$cum_rel_resp <- round(rank$cum_resp / sum(rank$cnt_resp),2);
rank$cum_rel_non_resp <- round(rank$cum_non_resp /
                                sum(rank$cnt_non_resp),2);
rank$ks <- abs(rank$cum_rel_resp - rank$cum_rel_non_resp);
library(scales)
rank$rrate <- percent(rank$rrate)
rank$cum_rel_resp <- percent(rank$cum_rel_resp)
rank$cum_rel_non_resp <- percent(rank$cum_rel_non_resp)
View(rank)
# rank
# Baseline Response Rate
sum(rf.train$TenYearCHD) / nrow(rf.train)
library(caret)
confusionMatrix(as.factor(rf.train$TenYearCHD), as.factor(rf.train$predict.class))
library(ROCR)
pred <- prediction(rf.train$predict.score[,2], rf.train$TenYearCHD)
perf <- performance(pred, "tpr", "fpr")
plot(perf)
KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
KS
```

```
auc <- performance(pred,"auc");
auc <- as.numeric(auc@y.values)
auc
## Gini Coefficient
library(ineq)
gini = ineq(rf.train$predict.score[,2], type="Gini")
gini
## Classification Error
with(rf.train, table(TenYearCHD, predict.class))
#
# Model Performance on Testing Data Set
# Rank Order
## Scoring syntax
rf.test$predict.class <- predict(tRF, rf.test, type="class")
rf.test$predict.score <- predict(tRF, rf.test, type="prob")
rf.test$deciles <- decile(rf.test$predict.score[,2])
head(rf.test)
tmp_DT = data.table(rf.test)
h_rank <- tmp_DT[, list(
  cnt = length(TenYearCHD),
  cnt_resp = sum(TenYearCHD),
  cnt_non_resp = sum(TenYearCHD == 0)) ,
  by=deciles][order(-deciles)]
h_rank$rrate <- round (h_rank$cnt_resp / h_rank$cnt,2)
h_rank$cum_resp <- cumsum(h_rank$cnt_resp)
h_rank$cum_non_resp <- cumsum(h_rank$cnt_non_resp)
h_rank$cum_rel_resp <- round(h_rank$cum_resp / sum(h_rank$cnt_resp),2)
h_rank$cum_rel_non_resp <- round(h_rank$cum_non_resp /
                                 sum(h_rank$cnt_non_resp),2)
h_rank$ks <- abs(h_rank$cum_rel_resp - h_rank$cum_rel_non_resp)
library(scales)
h_rank$rrate <- percent(h_rank$rrate)
h_rank$cum_rel_resp <- percent(h_rank$cum_rel_resp)
h_rank$cum_rel_non_resp <- percent(h_rank$cum_rel_non_resp)
View(h_rank)
```

```
484  # Baseline Response Rate
485  sum(rf.test$TenYearCHD) / nrow(rf.test)
486  #Confusion matrix for Test data
487  confusionMatrix(as.factor(rf.test$TenYearCHD), as.factor(rf.test$predict.class
488  pred1 <- prediction(rf.test$predict.score[,2], rf.test$TenYearCHD)
489  perf1 <- performance(pred1, "tpr", "fpr")
490  plot(perf1)
491  KS1 <- max(attr(perf1, 'y.values')[[1]]-attr(perf1, 'x.values')[[1]])
492  KS1
493  ## Area Under Curve
494  auc1 <- performance(pred1,"auc");
495  auc1 <- as.numeric(auc1@y.values)
496  auc1
497  ## Gini Coefficient
498  library(ineq)
499  gini1 = ineq(rf.test$predict.score[,2], type="Gini")
500  gini1
501  ## Classification Error
502  with(rf.test, table(TenYearCHD, predict.class))
503  #
504  #=============================================================================
505  # MODEL BUILDING - NEURAL NETWORK
506  #=============================================================================
507  #
508  library(neuralnet)
509  #
510
511  NNInput=impute.data.cor
512  attach(NNInput)
513  dim(NNInput)
514  str(NNInput)
515  #
516  # Create Dummy Variables for Categorical Variable education
517  NNInput$education.1 <- ifelse(NNInput$education == 1, 1, 0)
518  NNInput$education.2 <- ifelse(NNInput$education == 2, 1, 0)
```

```
NNInput$education.3 <- ifelse(NNInput$education == 3, 1, 0)
NNInput$education.4 <- ifelse(NNInput$education == 4, 1, 0)
head(NNInput)
NNInput = subset(NNInput, select = -c(education) )
#Creating Training and Testing Datasets
set.seed(111)
trainIndex <- createDataPartition(TenYearCHD,
                                  p = .7,
                                  list = FALSE,
                                  times = 1)
NN.train.data <- NNInput[trainIndex,]
NN.test.data <- NNInput[-trainIndex,]
dim(NN.train.data)
dim(NN.test.data)
#Scaling the data sets
scaledTrainData = subset(NN.train.data, select = -c(TenYearCHD) )
scaledTrainData = as.data.frame(scale(scaledTrainData))
scaledTrainData = cbind(NN.train.data[13],scaledTrainData )
temp.data = NN.train.data
temp.data = as.data.frame(scale(temp.data))
head(scaledTrainData)
scaledTestData = subset(NN.test.data, select = -c(TenYearCHD) )
scaledTestData = as.data.frame(scale(scaledTestData))
scaledTestData = cbind(NN.test.data[13],scaledTestData )
head(scaledTestData)
#NN.test.data = as.data.frame(scale(NN.test.data))
#head(NN.test.data)
names(NN.train.data)
nn1 <- neuralnet(formula = TenYearCHD ~
                   male +
                   age +
                   cigsPerDay +
                   BPMeds +
                   prevalentStroke +
                   prevalentHyp +
```

```
                    diabetes +
                    totChol +
                    BMI +
                    heartRate +
                    glucose +
                    BPDifference +
                    education.1 +
                    education.2 +
                    education.3 +
                    education.4 ,
                  data = scaledTrainData,
                  hidden = 3,
                  err.fct = "sse",
                  linear.output = FALSE
                  #lifesign = "full",
                  #lifesign.step = 10,
                  #threshold = 0.01,
                  #stepmax = 1000000
)
plot (nn1)
# Assigning the Probabilities to Dev Sample
NN.train.data$Prob = nn1$net.result[[1]]
head(NN.train.data)
# The distribution of the estimated probabilities
quantile(NN.train.data$Prob, c(0,1,5,10,25,50,75,90,95,98,99,100)/100)

hist(NN.train.data$Prob)
#
## deciling
NN.train.data$deciles <- decile(NN.train.data$Prob)
head(NN.train.data)
View(NN.train.data)
# Ranking code
##install.packages("data.table")
```

```
##install.packages("data.table")
library(data.table)
library(scales)
tmp_DT = data.table(NN.train.data)
rank <- tmp_DT[, list(
  cnt = length(TenYearCHD),
  cnt_resp = sum(TenYearCHD),
  cnt_non_resp = sum(TenYearCHD == 0)) ,
  by=deciles][order(-deciles)]
rank
rank$rrate <- round (rank$cnt_resp / rank$cnt,2);
rank$cum_resp <- cumsum(rank$cnt_resp)
rank$cum_non_resp <- cumsum(rank$cnt_non_resp)
rank$cum_rel_resp <- round(rank$cum_resp / sum(rank$cnt_resp),2);
rank$cum_rel_non_resp <- round(rank$cum_non_resp /
                               sum(rank$cnt_non_resp),2);
rank$cum_rel_non_resp
rank$ks <- abs(rank$cum_rel_resp - rank$cum_rel_non_resp)
library(scales)
rank$rrate <- percent(rank$rrate)
rank$cum_rel_resp <- percent(rank$cum_rel_resp)
rank$cum_rel_non_resp <- percent(rank$cum_rel_non_resp)
View(rank)
# Baseline Response Rate
sum(NN.train.data$TenYearCHD/nrow(NN.train.data))
# Assgining 0 / 1 class based on certain threshold
NN.train.data$Class = ifelse(NN.train.data$Prob>0.5,1,0)
class((NN.train.data$Class))
with( NN.train.data, table(TenYearCHD, as.factor(Class) ))
# We can use the confusionMatrix function of the caret package
#install.packages("caret")
library(caret)
confusionMatrix(as.factor(NN.train.data$TenYearCHD), as.factor(NN.train.data$Class))
## Error Computation
sum((NN.train.data$TenYearCHD - NN.train.data$Prob)^2)/2
## Other Model Performance Measures
```

```
623  library(ROCR)
624  # str(NN.train.data)
625  pred3 = ROCR::prediction(NN.train.data$Prob, NN.train.data$TenYearCHD)
626  perf3 <- performance(pred3, "tpr", "fpr")
627  plot(perf3)
628  KS3 <- max(attr(perf3, 'y.values')[[1]]-attr(perf3, 'x.values')[[1]])
629  KS3
630  auc3 <- performance(pred3,"auc");
631  auc3 <- as.numeric(auc3@y.values)
632  auc3
633  library(ineq)
634  gini3 = ineq(NN.train.data$Prob, type="Gini")
635  auc3
636  KS3
637  gini3
638  # Scoring Test dataset using the Neural Net Model Object
639  # To score we will use the compute function
640  compute.output = compute(nn1, scaledTestData)
641  # compute.output
642  NN.test.data$Predict.score = compute.output$net.result
643  # View(NN.test.data)
644  names(NN.train.data)
645  quantile(NN.test.data$Predict.score,
646           c(0,1,5,10,25,50,75,90,95,99,100)/100)
647  NN.test.data$deciles <- decile(NN.test.data$Predict.score)
648  tmp_DT = data.table(NN.test.data)
649  h_rank <- tmp_DT[, list(
650    cnt = length(TenYearCHD),
651    cnt_resp = sum(TenYearCHD),
652    cnt_non_resp = sum(TenYearCHD == 0)) ,
653    by=deciles][order(-deciles)]
654  h_rank$rrate <- round (h_rank$cnt_resp / h_rank$cnt,2);
655  h_rank$cum_resp <- cumsum(h_rank$cnt_resp)
656  h_rank$cum_non_resp <- cumsum(h_rank$cnt_non_resp)
657  h_rank$cum_rel_resp <- round(h_rank$cum_resp / sum(h_rank$cnt_resp),2);
658  h_rank$cum_rel_non_resp <- round(h_rank$cum_non_resp /
```

```
h_rank$cum_rel_non_resp <- round(h_rank$cum_non_resp /
                                 sum(h_rank$cnt_non_resp),2);
h_rank$ks <- abs(h_rank$cum_rel_resp - h_rank$cum_rel_non_resp);
#
library(scales)
h_rank$rrate <- percent(h_rank$rrate)
h_rank$cum_rel_resp <- percent(h_rank$cum_rel_resp)
h_rank$cum_rel_non_resp <- percent(h_rank$cum_rel_non_resp)
View(h_rank)
# Assigning the Probabilities to Holdout Sample
NN.test.data$Prob = compute.output$net.result
# Assgining 0 / 1 class based on certain threshold
NN.test.data$Class = ifelse(NN.test.data$Prob>0.5,1,0)
with( NN.test.data, table(TenYearCHD, as.factor(Class) ))
# We can use the confusionMatrix function of the caret package
#install.packages("caret")
library(caret)
confusionMatrix(as.factor(NN.test.data$TenYearCHD), as.factor(NN.test.data$Class))
## Error Computation
sum((NN.test.data$TenYearCHD - NN.test.data$Prob)^2)/2
## Other Model Performance Measures
library(ROCR)
# str(NN.test.data)
pred4 <- ROCR::prediction(NN.test.data$Prob, NN.test.data$TenYearCHD)
perf4 <- performance(pred4, "tpr", "fpr")
plot(perf4)
KS4 <- max(attr(perf4, 'y.values')[[1]]-attr(perf4, 'x.values')[[1]])
KS4
auc4 <- performance(pred4,"auc");
auc4 <- as.numeric(auc4@y.values)
auc4
library(ineq)
gini4 = ineq(NN.test.data$Prob, type="Gini")
auc4
KS4
gini4
```

```r
#    Ensamble Methods
#================================================================
#Bagging
#install.packages("xgboost")
#install.packages("caret")
#install.packages("ipred")
#install.packages("rpart")
library(xgboost)
library(caret)
library(ipred)
library(rpart)
#Creating Training and Testing Datasets
set.seed(111)
trainIndex <- createDataPartition(TenYearCHD,
                                  p = .7,
                                  list = FALSE,
                                  times = 1)
train.data <- NNInput[trainIndex,]
test.data <- NNInput[-trainIndex,]
dim(train.data)
dim(test.data)
BAGmodel <- bagging(as.factor(TenYearCHD)~., data = train.data, control = rpart.control(maxdepth = 10, minsplit = 50))
BAGmodel

library(ROCR)

# Measure Model Performance
train.data$predict.class <- predict(BAGmodel, train.data, type="class")
train.data$predict.score <- predict(BAGmodel, train.data, type="prob")
head(train.data)
train.data$deciles = decile(train.data$predict.score[,2])
library(data.table)
library(scales)

# Ranking code
```

```
730 tmp_DT = data.table(train.data)
731 h_rank <- tmp_DT[, list(
732   cnt = length(TenYearCHD),
733   cnt_resp = sum(TenYearCHD),
734   cnt_non_resp = sum(TenYearCHD == 0)) ,
735   by=deciles][order(-deciles)]
736 h_rank$rrate <- round(h_rank$cnt_resp / h_rank$cnt,4);
737 h_rank$cum_resp <- cumsum(h_rank$cnt_resp)
738 h_rank$cum_non_resp <- cumsum(h_rank$cnt_non_resp)
739 h_rank$cum_rel_resp <- round(h_rank$cum_resp / sum(h_rank$cnt_resp),4);
740 h_rank$cum_rel_non_resp <- round(h_rank$cum_non_resp /
741                                  sum(h_rank$cnt_non_resp),4);
742 h_rank$ks <- abs(h_rank$cum_rel_resp - h_rank$cum_rel_non_resp)*100;
743 h_rank$rrate <- percent(h_rank$rrate)
744 h_rank$cum_rel_resp <- percent(h_rank$cum_rel_resp)
745 h_rank$cum_rel_non_resp <- percent(h_rank$cum_rel_non_resp)
746 View(h_rank)
747
748 #install.packages("ROCR")
749 #install.packages("ineq")
750
751 library(ineq)
752 pred <- ROCR::prediction(train.data$predict.score[,2], train.data$TenYearCHD)
753 perf <- performance(pred, "tpr", "fpr")
754 plot(perf)
755 KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
756 auc <- performance(pred,"auc");
757 auc <- as.numeric(auc@y.values)
758 gini = ineq(train.data$predict.score, type="Gini")
759 with(train.data, table(TenYearCHD, predict.class))
760 auc
761 KS
762 gini
```