IS-777 Data Analytics: Applications in Healthcare using R

Team Project

Topic - Heart Disease Risk Prediction

Team C

Sambhaw Sharma
Akshay Negi
Sai Manikanta Yerram
Rohit Bhosale
Vibhor Vaidya

Index

Descriptive Analysis

Descriptive analysis

Descriptive analysis is an essential first step for conducting statistical analysis. This involves

1. Descriptive analysis for each variable
2. Descriptive analysis for combinations of variables

First importing the dataset into R-studio and removing Id, we get our first look at data. Now we have 12 variables.

The total number of observations (n) -- 68586

If we go with logistic regression, we will have 11+1 parameters, Weight associated with each variable, and W0.

Predictor variables -- 11

Response variable -- 1

Variables Names	**Description**	**Type**	**predictor/response**
Age	Age in days	Numeric	p
Gender	1 represents Women, 2 represents Men	Categorical	p
Height	Height in centimeters	Numeric	p
Weight	Weight in Kilograms	Numeric	p
Ap_hi(Systolic blood pressure)	the average systolic blood pressure is below 120. A reading of 140 or above is considered high blood pressure.	Numeric	p
Ap_lo - Diastolic blood pressure	The normal diastolic blood pressure is below 80. A reading of 90 or	Numeric	p

	above is considered high blood pressure.		
Cholesterol	1: normal, 2: above normal, 3: well above normal	Categorical	p
Gluc(Glucose)	1: normal, 2: above normal, 3: well above normal	Categorical	p
Smoke	This tells if the person smokes or not. 0: Do not smoke 1: Smoke	Categorical	p
Alcohol intake	This tells if the person drinks or not. 0: Do not Drink 1: Drinks	Categorical	p
Physical activity	This tells if the person's cardio problem is active or inactive 0: Inactive 1: Active	Categorical	p
cardio	Presence or absence of cardiovascular disease.	Categorical	R

1. age: in days.
2. gender: 1 - women, 2 – men.
3. height: in cm.
4. weight: in kgs.
5. ap_hi: Systolic blood pressure
6. ap_lo: Diastolic blood pressure.
7. cholesterol: 1: normal, 2: above normal, 3: well above normal.
8. gluc: 1: normal, 2: above normal, 3: well above normal.
9. smoke: whether the patient smokes or not.
10. alco: Alcohol intake
11. active: Physical activity.
12. cardio: Presence or absence of cardiovascular disease.

Importing the data from csv into a data frame using read.csv() method

```
##   id   age gender height weight ap_hi ap_lo cholesterol gluc smoke alco
active
## 1  0 18393      2    168     62   110    80           1    1     0    0
1
## 2  1 20228      1    156     85   140    90           3    1     0    0
1
## 3  2 18857      1    165     64   130    70           3    1     0    0
0
## 4  3 17623      2    169     82   150   100           1    1     0    0
1
## 5  4 17474      1    156     56   100    60           1    1     0    0
0
## 6  8 21914      1    151     67   120    80           2    2     0    0
0
##   cardio
## 1      0
## 2      1
## 3      1
## 4      1
## 5      0
## 6      0
```

Displaying the number of columns in the dataframe Displaying the number of rows in the dataframe

```
print(ncol(data))
```

```
## [1] 13
```

```
print(nrow(data))
```

```
## [1] 70000
```

Numeric Data-

we have a total of five predictor variable that is of the numeric type as shown below, collecting all the numeric values in a separate data frame

```
numtab <- data[c(1,3:6)]
head(numtab)
```

```
##     age height weight ap_hi ap_lo
## 1 18393    168     62   110    80
## 2 20228    156     85   140    90
## 3 18857    165     64   130    70
## 4 17623    169     82   150   100
## 5 17474    156     56   100    60
## 6 21914    151     67   120    80
```

Categorical data

we have a total of six predictor variable that is of the categorical type as shown below as well as our response variable is also categorical type, collecting all the categorical data in a separate data frame

```
catdata <- data[-c(1,3:6)]
head(catdata)
```

```
##   gender cholesterol gluc smoke alco active cardio
## 1      2           1    1     0    0      1      0
## 2      1           3    1     0    0      1      1
## 3      1           3    1     0    0      0      1
## 4      2           1    1     0    0      1      1
## 5      1           1    1     0    0      0      0
## 6      1           2    2     0    0      0      0
```

Histograms-

Data Type -

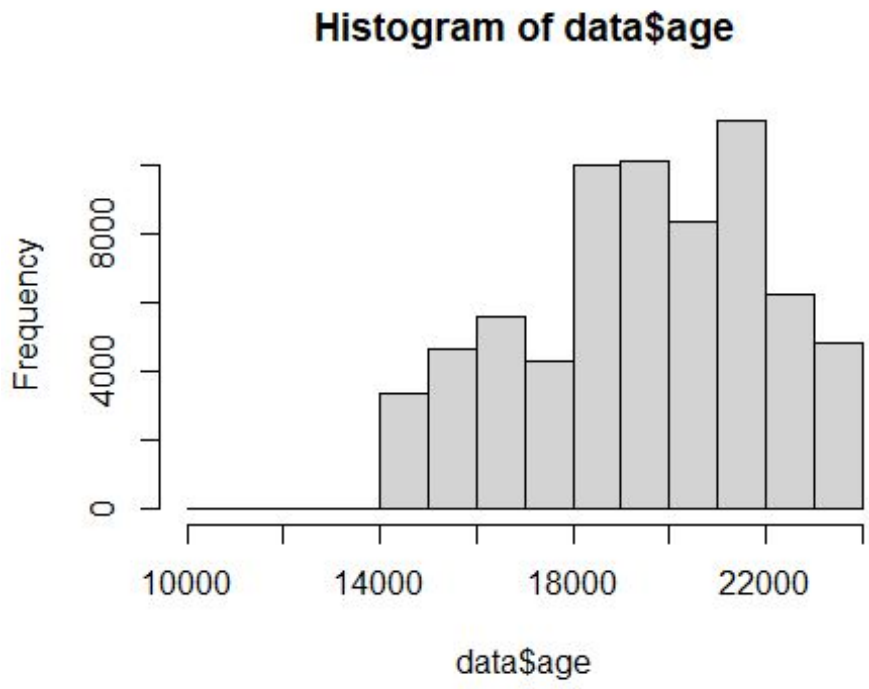
The datatypes of columns currently present in the dataframe This displays a plot of the datatype of features Reference -

Plotting histograms for numeric data to show their distribution

Age

There are 68586 rows with no missing value in this column, here age is given in number of days and Gini's difference is 2824, minimum = 10798, Q1 = 17657 , mean age is 19464, Q3 = 21324, Max = 23713.

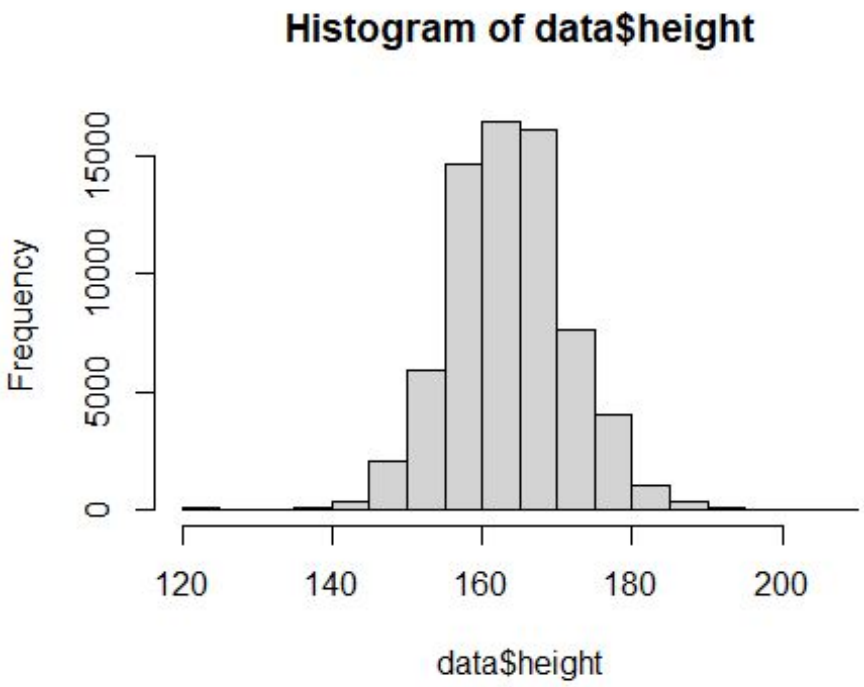
```
hist(data$age)
```



Height

There are 68586 rows with zero missing value in this column. Here height is given in centimeters, and Gini's difference is 8.847, minimum = 120, Q1 = 159, mean is 164.4, Q3 = 170, Max = 207

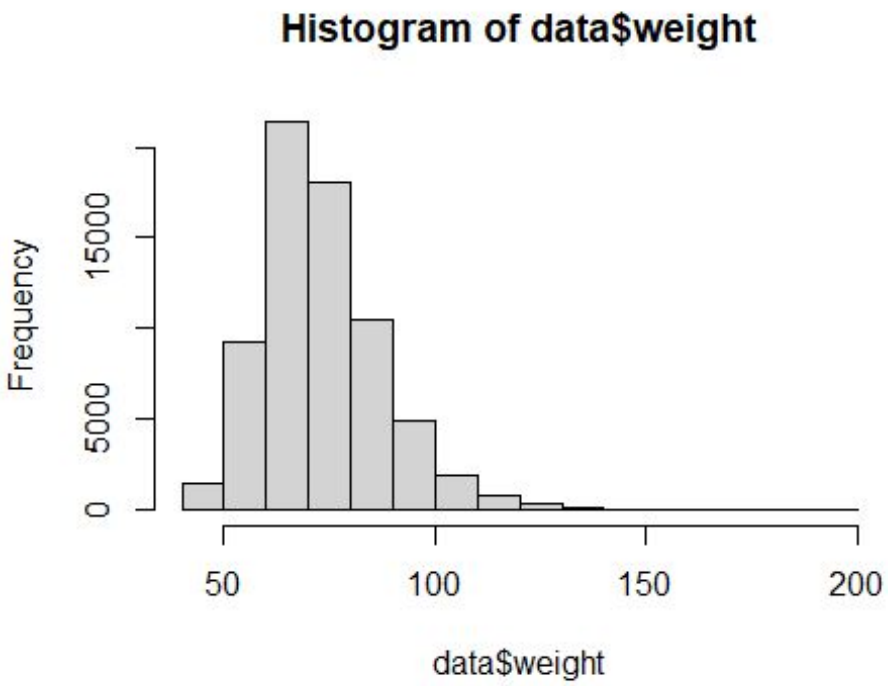
```
hist(data$height)
```



Weight

There are 68586 rows with zero missing value in this column, here weight is given in Kilograms and Gini's difference is 15.5, minimum = 40.0, Q1 = 65 , mean is 74.14, Q3 = 82, Max = 200.0

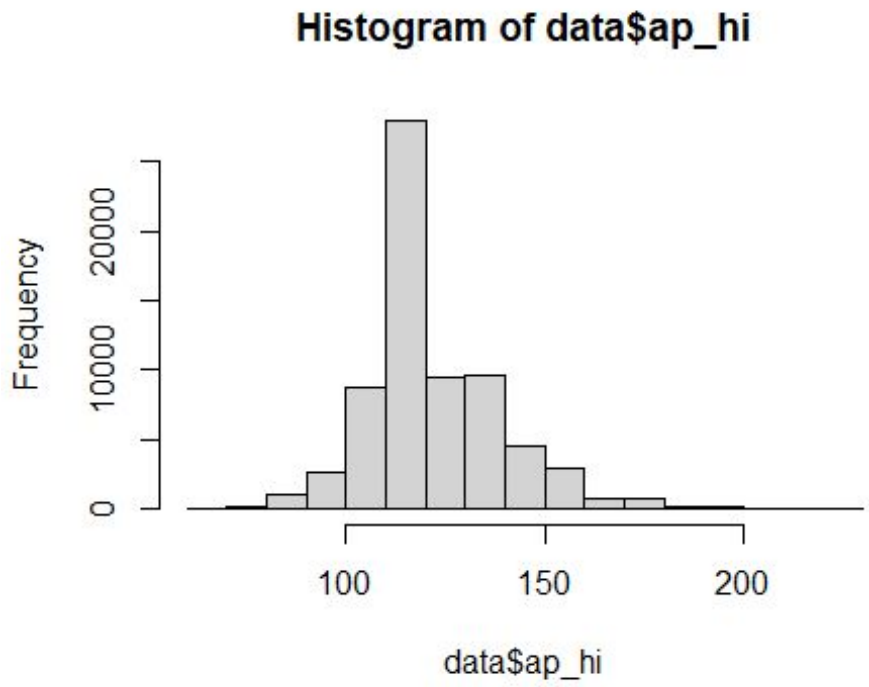
```
hist(data$weight)
```



Systolic blood pressure

A normal systolic blood pressure is below 120. A reading of 140 or above is considered as high blood pressure. There are 68586 rows with zero missing value in this column, Gini's difference is 17.48, minimum = 60, Q1 = 120 , mean is 126.7, Q3 = 140, Max = 230.

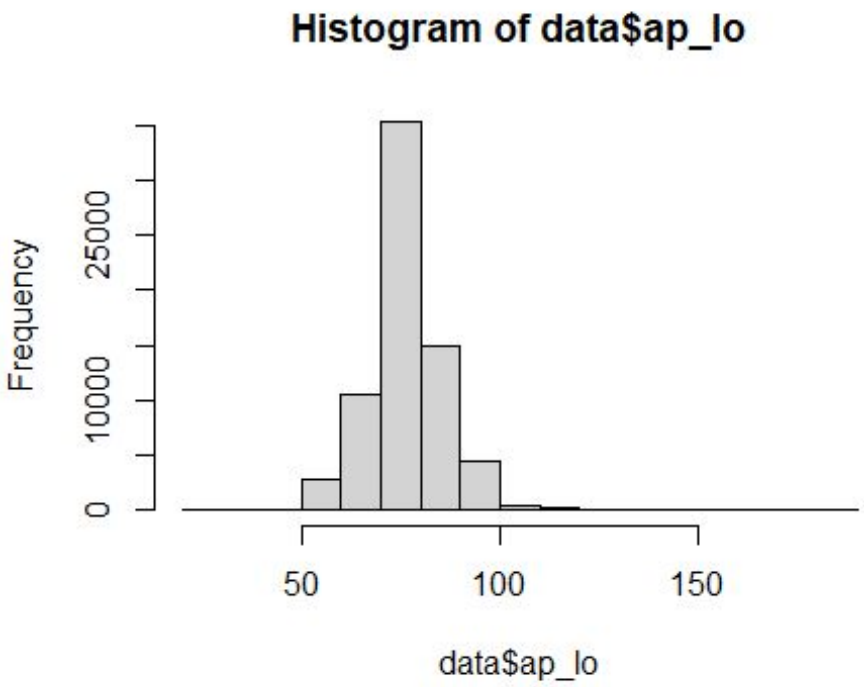
```
hist(data$ap_hi)
```



Diastolic blood pressure

A normal diastolic blood pressure is below 80. A reading of 90 or above is considered as high blood pressure. There are 68586 rows with zero missing value in this column, Gini's difference is 81.29, minimum = 20, Q1 = 80, mean is 81.29, Q3 = 90, Max = 182.

```
hist(data$ap_lo)
```



Barplots -Displaying the bar plots for categorical data representing the number of observations in each category

Gender

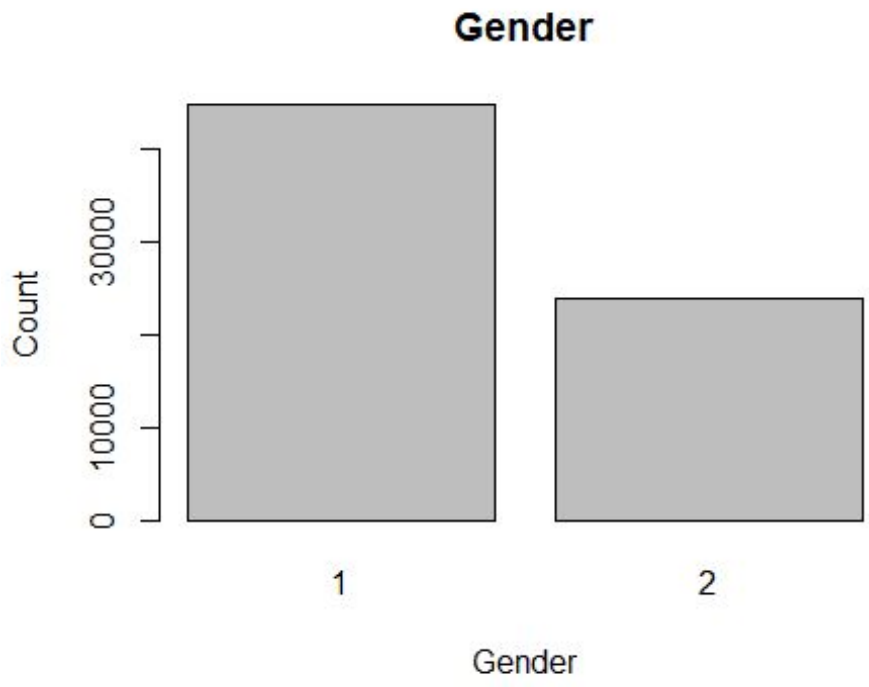
This column has no missing data, two distinct values 1, 2 (1 - women, 2 – men) with a total of 68586 observations, frequency and percent proportion are given below.

gender

n	missing	distinct
68586	0	2

Value	1	2
Frequency	44664	23922
Proportion	0.651	0.349

```
barplot(table(data$gender),main="Gender",
        xlab="Gender",
        ylab="Count")
```



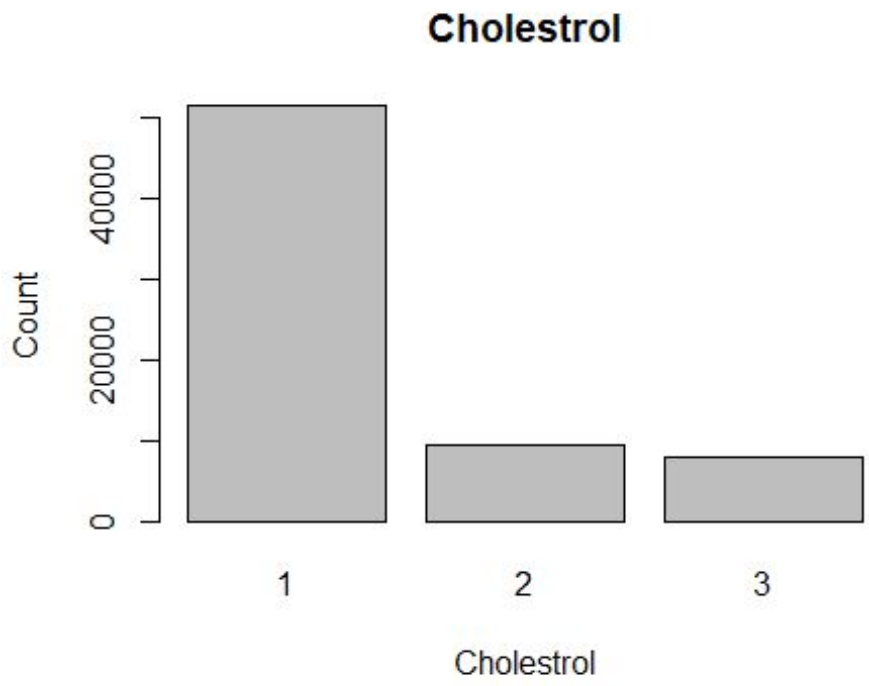
Cholesterol

This column has zero missing data, three distinct values 1, 2, 3 (1 - normal, 2 – above normal, 3-well above normal) with a total of 68586 observations.

cholesterol

n	missing	distinct		
68586	0	3		
Value	1	2	3	
Frequency	51430	9295	7861	
Proportion	0.750	0.136	0.11	

```
barplot(table(data$cholesterol),main="Cholestrol",
        xlab="Cholestrol",
        ylab="Count")
```



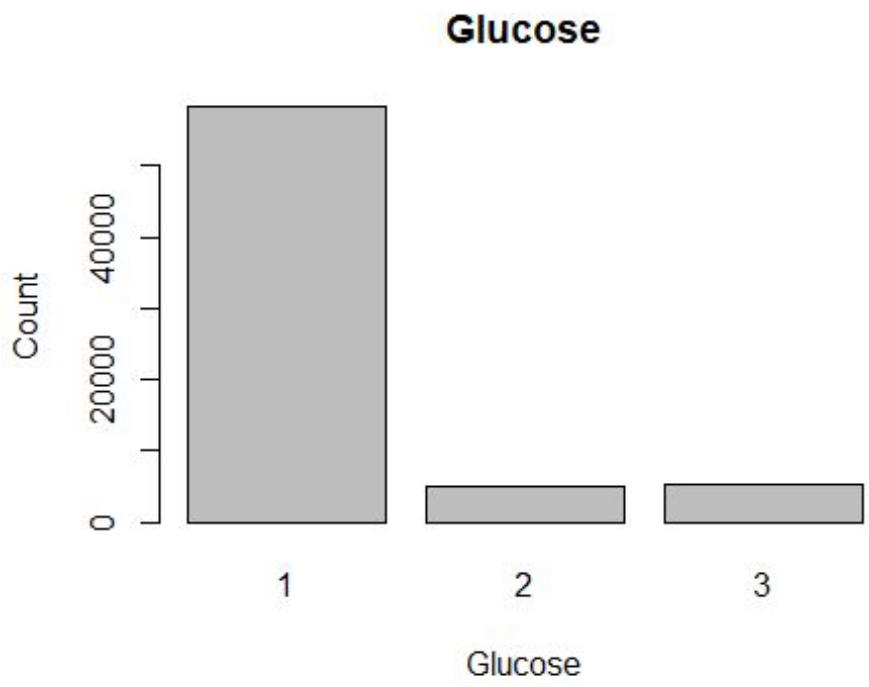
Glucose

This column has zero missing data, three distinct values 1, 2, 3 (1 - normal, 2 – above normal, 3-well above normal) with a total of 68586 observations. Frequency and percent proportion are given below.

gluc

n	missing	distinct			
68586	0	3			
Value	1	2		3	
Frequency	58310		5065		5211
Proportion	0.850	0.074		0.076	

```
barplot(table(data$gluc),main="Glucose",
        xlab="Glucose",
        ylab="Count")
```



Smoke

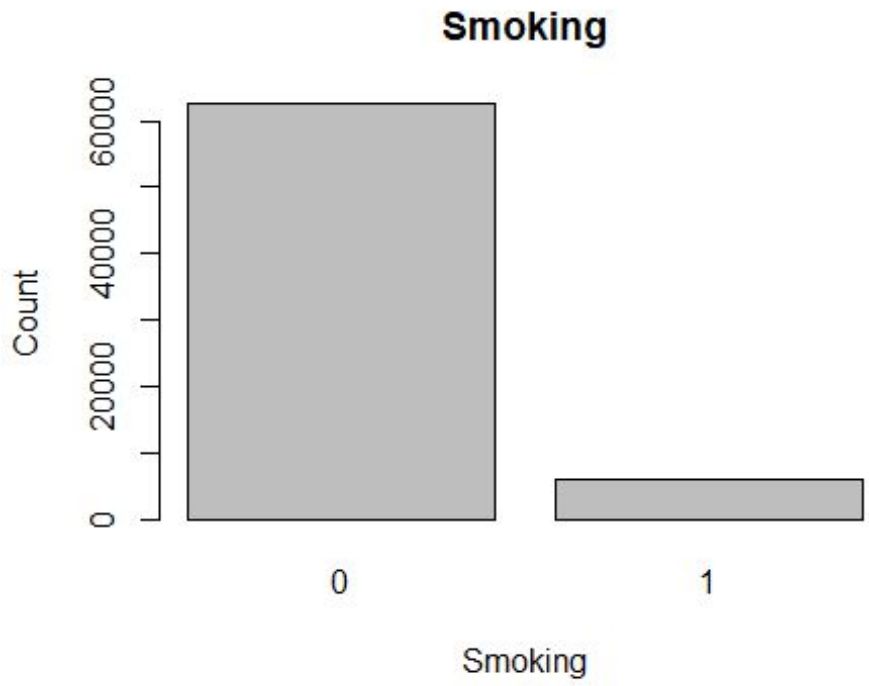
This column has no missing data, two distinct binary values (0 - Do not smoke, 1 – Smoke) with a total of 68586 observations, frequency and percent proportion are given below.

smoke

n	missing	distinct
68586	0	2

Value	0	1
Frequency	62552	6034
Proportion	0.912	0.088

```
barplot(table(data$smoke),main="Smoking",
        xlab="Smoking",
        ylab="Count")
```



Alcohol

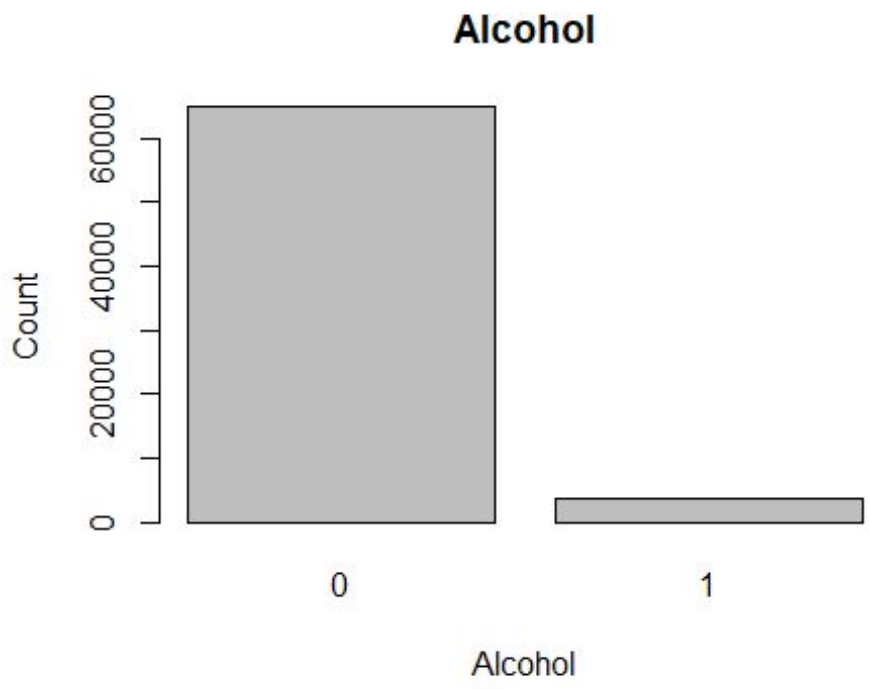
This column has no missing data, two distinct binary values (0 - Do not Drinks, 1 – Drinks) with a total of 68586 observations, frequency and percent proportion are given below.

alco

n	missing	distinct
68586	0	2

Value	0	1
Frequency	64927	3659
Proportion	0.947	0.053

```
barplot(table(data$alco),main="Alcohol",
        xlab="Alcohol",
        ylab="Count")
```



Active

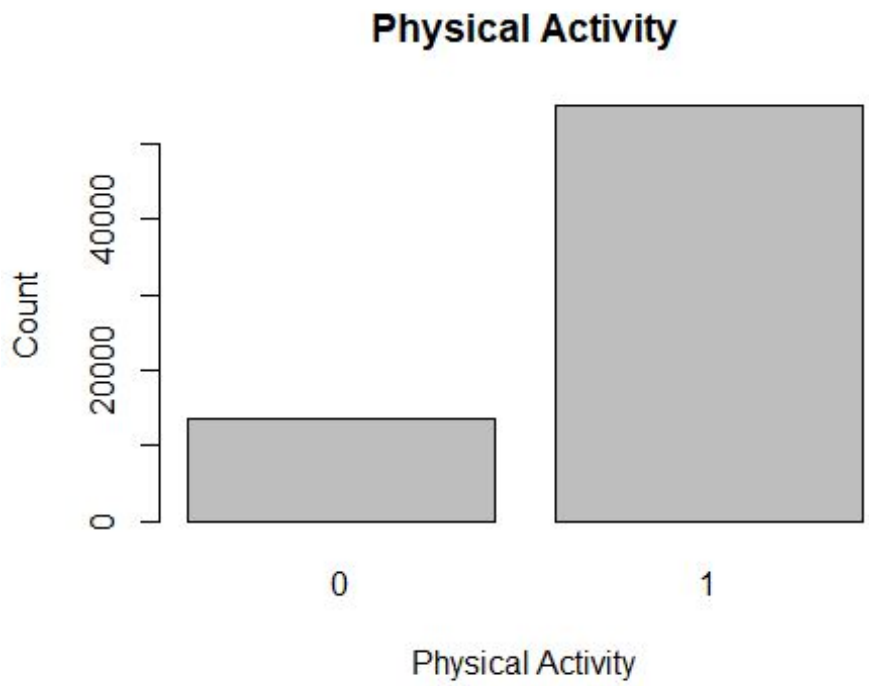
This column has no missing data, two distinct binary values (0 - Inactive, 1 – Active) with a total of 68586 observations, frequency and percent proportion are given below.

active

n	missing	distinct
68586	0	2

Value	0	1
Frequency	13491	55095
Proportion	0.197	0.803

```
barplot(table(data$active),main="Physical Activity",
        xlab="Physical Activity",
        ylab="Count")
```



Cardio

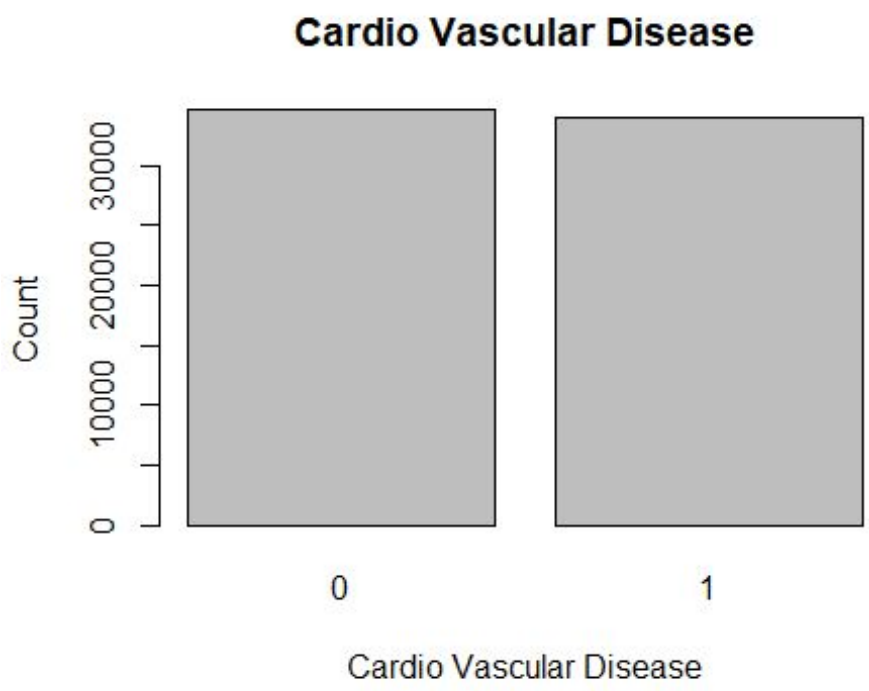
This column has no missing data, two distinct binary values (presence or absence of cardiovascular disease) with a total of 68586 observations, frequency and percent proportion are given below.

cardio

n	missing	distinct
68586	0	2

Value	0	1
Frequency	34650	33936
Proportion	0.505	0.495

```
barplot(table(data$cardio),main="Cardio Vascular Disease",
        xlab="Cardio Vascular Disease",
        ylab="Count")
```



Displaying the summary of the numerical data And distribution of categorical data

5 number summary of numeric data.

```
summary(numtab)
```

```
##       age            height          weight           ap_hi
##  Min.   :10798   Min.   : 55.0   Min.   : 10.00   Min.   : -150.0
##  1st Qu.:17664   1st Qu.:159.0   1st Qu.: 65.00   1st Qu.:  120.0
##  Median :19703   Median :165.0   Median : 72.00   Median :  120.0
##  Mean   :19469   Mean   :164.4   Mean   : 74.21   Mean   :  128.8
##  3rd Qu.:21327   3rd Qu.:170.0   3rd Qu.: 82.00   3rd Qu.:  140.0
##  Max.   :23713   Max.   :250.0   Max.   :200.00   Max.   :16020.0
##      ap_lo
##  Min.   :  -70.00
##  1st Qu.:   80.00
##  Median :   80.00
##  Mean   :   96.63
##  3rd Qu.:   90.00
##  Max.   :11000.00
```

```
Hmisc::describe(data)
```

```
##  12  Variables      68586  Observations
##
--------------------------------------------------------------------------------
---
## age
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##    68586        0     8063        1    19464     2824    15056    15836
##      .25      .50      .75      .90      .95
##    17657    19701    21324    22638    23256
##
## lowest : 10798 10859 10878 10964 14275, highest: 23687 23690 23692 23701
23713
##
--------------------------------------------------------------------------------
---
## gender
##        n  missing distinct
##    68586        0        2
##
## Value          1     2
## Frequency  44664 23922
## Proportion 0.651 0.349
##
--------------------------------------------------------------------------------
---
```

```
## height
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##    68586        0       73    0.998    164.4    8.847      152      155
##      .25      .50      .75      .90      .95
##      159      165      170      175      178
##
## lowest : 120 122 125 130 131, highest: 195 196 197 198 207
##
-------------------------------------------------------------------------
---
## weight
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##    68586        0      263    0.999    74.14     15.5       55       58
##      .25      .50      .75      .90      .95
##       65       72       82       93      100
##
## lowest :  40.0  41.0  42.0  42.2  43.0, highest: 177.0 178.0 180.0 183.0
200.0
##
-------------------------------------------------------------------------
---




## ap_hi
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##    68586        0      107    0.928    126.7    17.48      100      110
##      .25      .50      .75      .90      .95
##      120      120      140      150      160
##
## lowest :  60  70  80  85  90, highest: 202 210 215 220 230
##
-------------------------------------------------------------------------
---
## ap_lo
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##    68586        0       84    0.858    81.29    9.712       70       70
##      .25      .50      .75      .90      .95
##       80       80       90       90      100
##
## lowest :  20  30  40  45  49, highest: 150 160 170 180 182
##
-------------------------------------------------------------------------
---
## cholesterol
##        n  missing distinct
##    68586        0        3
##
## Value          1     2     3
```

```
## Frequency  51430  9295  7861
## Proportion 0.750 0.136 0.115
##
-------------------------------------------------------------------------
---
## gluc
##        n  missing distinct
##    68586        0        3
##
## Value          1     2     3
## Frequency  58310  5065  5211
## Proportion 0.850 0.074 0.076
##
-------------------------------------------------------------------------
---
## smoke
##        n  missing distinct
##    68586        0        2
##
## Value          0     1
## Frequency  62552  6034
## Proportion 0.912 0.088
##
-------------------------------------------------------------------------
---
## alco
##        n  missing distinct
##    68586        0        2
##
## Value          0     1
## Frequency  64927  3659
## Proportion 0.947 0.053
##
-------------------------------------------------------------------------
---
## active
##        n  missing distinct
##    68586        0        2
##
## Value          0     1
## Frequency  13491 55095
## Proportion 0.197 0.803
##
-------------------------------------------------------------------------
---
## cardio
##        n  missing distinct
##    68586        0        2
##
## Value          0     1
```

```
## Frequency  34650 33936
## Proportion 0.505 0.495
##
------------------------------------------------------------------------
```

Correlation matrix

The cor() function produces a matrix that contains all of the pairwise correlations among the predictors in a data set. Here we have selected all the variables with numeric fields and tried to determine the correlation between those variables. As we can see here, table

height and age don't have any correlation because the dataset contains most of the age values from 29 years to 65 years, and height does not change for adult humans. Similarly, weight and age have no relation. Age and ap_hi and ap_low are weakly positive correlations with age means there is some probability that with age, blood pressure increases. Weight and ap_hi and ap_low are weakly positive correlations with age means there is some probability that with weight, blood pressure increases. ap_hi and ap_low have a very strong positive correlation showing that both types of blood pressure rise together.

```
str(data[c(1,3:6)])
```

```
## 'data.frame':    68586 obs. of  5 variables:
##  $ age   : int  18393 20228 18857 17623 17474 21914 22113 22584 17668
19834 ...
##  $ height: int  168 156 165 169 156 151 157 178 158 164 ...
##  $ weight: num  62 85 64 82 56 67 93 95 71 68 ...
##  $ ap_hi : int  110 140 130 150 100 120 130 130 110 110 ...
##  $ ap_lo : int  80 90 70 100 60 80 80 90 70 60 ...
```

```
cor(data[c(1,3:6)], use="pairwise.complete.obs")
```

```
##                age      height     weight     ap_hi      ap_lo
## age     1.00000000 -0.08554315 0.05572175 0.2099602 0.15612991
## height -0.08554315  1.00000000 0.30466972 0.0174550 0.03455806
## weight  0.05572175  0.30466972 1.00000000 0.2703174 0.25128237
## ap_hi   0.20996016  0.01745500 0.27031739 1.0000000 0.73027367
## ap_lo   0.15612991  0.03455806 0.25128237 0.7302737 1.00000000
```

Model Selection

K Nearest Neighbors

We will be fitting a KNN model using the knn() function included in the class library. This function generally works differently from the other fitting models. Rather than a two-step approach in which we first fit the model and then use the model to make predictions, knn() forms predictions using a single command.

It takes four arguments: TrainX, which is a matrix containing the predictors associated with the training data. TestX is a matrix containing the predictors associated with the testing data. TrainDirection is a vector containing the class labels for the training observations. K is the number of neighbors to be used for clustering.

Contrary to logistic regression, for categorical data kNN cannot be applied directly since it is based on the Euclidean distances. To define the distance metrics for categorical variables, the first step of preprocessing the dataset is to use dummy variables to represent the categorical variables.

dummy <- dummyVars(" ~ .", data=data1) newdata <- data.frame(predict(dummy, newdata = data1))

Using the above code, I have converted the categorical variable into multiple categories, each with a 1 or 0. Except for the response variable, I have made all other categorical variables into multiple variables.

ran <- sample(1:nrow(newdata), 0.5 * nrow(newdata)) trainX <- newdata[c(1:19)][ran,] testX <- newdata[c(1:19)][-ran,] trainY <- newdata[ran,20] trainY <- trainY $cardiotestY <- newdata[-ran, 20] testY <- testY$ cardio

Using the above code, I have split the data into 50% for training and 50% for testing. Where trainX is training predictors and trainY is class labels for training observations. Similarly, testX is the predictor for test data, and testY are their associated labels.

pred <- knn(trainX,testX,trainY,k=5)

Using the above line of code, we have fit the model. Here we have chosen k as five, which means the classification algorithm uses five nearest neighbors to classify a new point to a cluster.

Creating confusion matrix:

table(pred,testY)

Where 1 represents Presence, and 0 represents the absence of cardiovascular disease.

accuracy <- function(x){sum(diag(x)/(sum(Rowsums(x)))) * 100}

This function divides the correct predictions by the total number of predictions that tell us how accurate the model is.

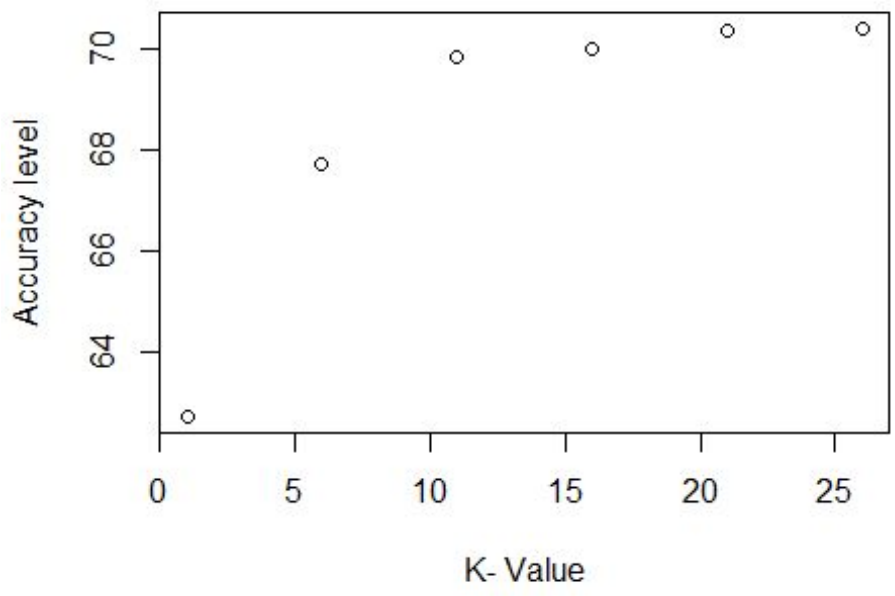
```
pred <- knn(trainX,testX,trainY,k=26)
print(table(pred,testY))
```

```
##     testY
## pred     0     1
##    0 13230  6119
##    1  4037 10907
```

```
#accuracy
print(accuracy(table(pred,testY)))
```

```
## [1] 70.3846
```

If we plot the k - value graph, we will get a graph like blow we can assume k = 26 because accuracy is highest at that point = 70.23299



Logistic regression

Next, we will fit a logistic regression model to predict cardio using columns 1 to 11

Now using codes from above to split the data into two halves for training and testing. The glm() function fits generalized glm() linear models, a class of models that includes logistic regression for which we must pass the argument family=binomial to tell R to run logistic regression.

Here we can see we have a total 15 number of coefficients, and P-value is very high, 0.54410 for gender means the statistical significance of gender to predict the cardio is not significant. Similarly, the statistical significance of gluc2 is also not significant because the p-value is high, hence when one can either use forward selection or backward elimination, choose the right attributes to create a better prediction model.

We can now run the anova() function on the model to analyze the table of deviance. The difference between the null deviance and the residual deviance shows how our model is doing against the null model (a model with only the intercept). The wider this gap, the better we can see age, weight, ap-hi, and cholesterol have higher significance in calculating the response variable.

We use the predict() function to fit the model into the test data and predict the response variable as we have done in the below code block. First, we created an object and assigned response variable value to the that then we can use the if-else statement to that will change probability to the class label

If the probability is greater than 0.5, then the class is '1,' which means he/she has cardiovascular disease. If the probability is less than 0.5, then the class is '0' means he/she doesn't have cardiovascular disease

Here The mean() function can be used to compute the fraction of days for which the prediction was correct. That is equal to the accuracy of the model, as we can in the below code block. We ran the model on both halves first on test data and received an accuracy of 72.89% and again on the training dataset and received the accuracy of 72.75%, which means there is very low variance. We will be able to create a more accurate model if we use K-fold cross-validation, and we might be able to minimize the error rate.

```
# logistic regression

glm.fits=glm(cardio~ ., data = train1 ,family =binomial )
summary (glm.fits)

##
## Call:
## glm(formula = cardio ~ ., family = binomial, data = train1)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.0237  -0.9167  -0.3440   0.9315   2.5641
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.069e+01  3.495e-01 -30.573  < 2e-16 ***
## age           1.402e-04  5.242e-06  26.740  < 2e-16 ***
## gender2       2.952e-02  3.130e-02   0.943   0.3456
## height       -4.841e-03  1.901e-03  -2.546   0.0109 *
## weight        1.041e-02  9.869e-04  10.550  < 2e-16 ***
```

```
## ap_hi         5.829e-02  1.325e-03  44.005  < 2e-16 ***
## ap_lo         8.381e-03  2.061e-03   4.067 4.76e-05 ***
## cholesterol2  3.661e-01  3.909e-02   9.365  < 2e-16 ***
## cholesterol3  1.107e+00  5.108e-02  21.668  < 2e-16 ***
## gluc2         6.569e-02  5.168e-02   1.271   0.2037
## gluc3        -3.828e-01  5.608e-02  -6.826 8.72e-12 ***
## smoke1       -1.960e-01  4.934e-02  -3.972 7.13e-05 ***
## alco1        -1.658e-01  6.011e-02  -2.759   0.0058 **
## active1      -2.346e-01  3.108e-02  -7.548 4.42e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 47534  on 34292  degrees of freedom
## Residual deviance: 38315  on 34279  degrees of freedom
## AIC: 38343
##
## Number of Fisher Scoring iterations: 4
```

```
anova(glm.fits , test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: cardio
##
## Terms added sequentially (first to last)
##
##
##             Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                        34292      47534
## age          1   2017.6     34291      45516 < 2.2e-16 ***
## gender       1     13.0     34290      45503  0.000319 ***
## height       1      0.7     34289      45502  0.416311
## weight       1   1105.1     34288      44397 < 2.2e-16 ***
## ap_hi        1   5391.7     34287      39006 < 2.2e-16 ***
## ap_lo        1     22.8     34286      38983 1.839e-06 ***
## cholesterol  2    527.5     34284      38455 < 2.2e-16 ***
## gluc         2     47.5     34282      38408 4.790e-11 ***
## smoke        1     27.4     34281      38381 1.646e-07 ***
## alco         1      8.5     34280      38372  0.003541 **
## active       1     57.0     34279      38315 4.344e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
## [1] "Accuracy 0.726766395474295"
```

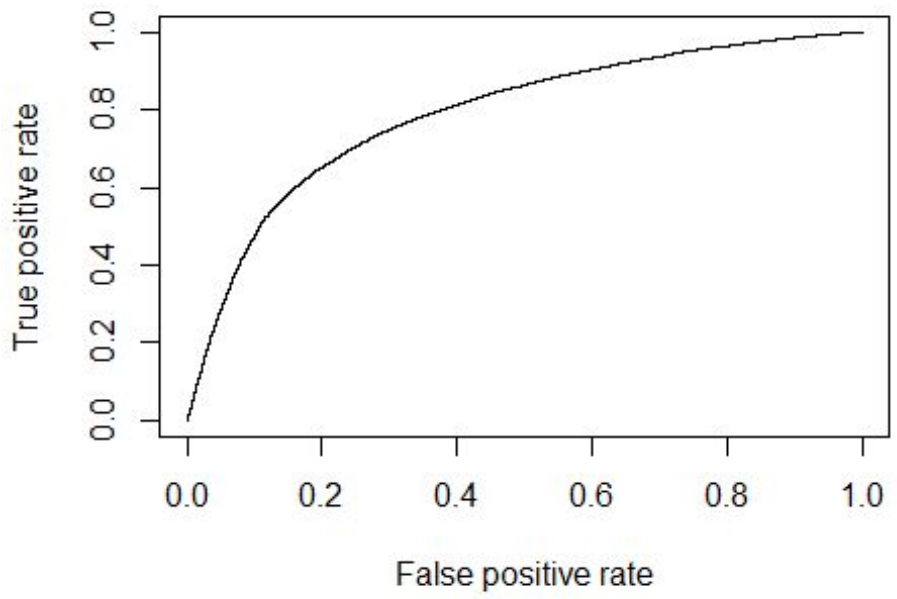
Roc curve

Using the below code block, we can draw the roc curve between true positive rate and false positive rate using the below code block.

As we can see, the ROC curve is above the diagonal means the model is accurately predicting the true positives. We can also calculate the ROC - area under the curve using the below block of code.

Roc- AUC that we received is 79.28% means the model has correctly predicted 79% true positive our future aim is to increase this value.

```
p <- predict(glm.fits, newdata=subset(test1,select=c(1:11)),type='response')
pr <- prediction(p , test1$cardio)
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
plot(prf)
```




```
auc <- performance(pr, measure = "auc")
auc <- auc@y.values[[1]]
auc

## [1] 0.7905714
```

Naive Bayes Classifier

Naive Bayes Classifier Compute the conditional a-posterior probabilities of a categorical class variable given independent predictor variables using the Bayes rule. The standard naive Bayes classifier assumes independence of the predictor variables and Gaussian distribution given the target class of metric predictors.

Using the above code to split the data as training and testing data and Creating confusion matrix:

Now creating objects x, which holds the predictor variables, and y, which holds the response variable cardio

x = training[1:11] y = training$cardio

Applying the Naive Bayes model to predict cardio using columns 1 to 11

model <- naiveBayes(x , y)

Using predict () function to fit the model into the test and training data to predict the response variable.

Using Naive Bayes Classification accuracy of the presence or absence of cardiovascular disease results in 72.04% on test data and 71.91% on training data.

```
Predict <- predict(model,newdata = testing )
misClasificError <- mean(Predict != testing$cardio)
print(paste('Accuracy',1-misClasificError))
```

```
## [1] "Accuracy 0.717660095649131"
```

```
Predict <- predict(model,newdata = training )
misClasificError <- mean(Predict != training$cardio)
print(paste('Accuracy',1-misClasificError))
```

```
## [1] "Accuracy 0.720781493001555"
```

```
print(confusionMatrix(Predict , training$cardio))
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction     0     1
##          0 21197  9572
##          1  4791 15880
##
##                Accuracy : 0.7208
##                  95% CI : (0.7169, 0.7247)
##     No Information Rate : 0.5052
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4404
##
```

```
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8156
##             Specificity : 0.6239
##          Pos Pred Value : 0.6889
##          Neg Pred Value : 0.7682
##              Prevalence : 0.5052
##          Detection Rate : 0.4121
##    Detection Prevalence : 0.5982
##       Balanced Accuracy : 0.7198
##
##        'Positive' Class : 0
##
```

Random forest

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 6722 2754
##          1 1940 5730
##
##                Accuracy : 0.7262
##                  95% CI : (0.7195, 0.7329)
##     No Information Rate : 0.5052
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4519
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7760
##             Specificity : 0.6754
##          Pos Pred Value : 0.7094
##          Neg Pred Value : 0.7471
##              Prevalence : 0.5052
##          Detection Rate : 0.3920
##    Detection Prevalence : 0.5527
##       Balanced Accuracy : 0.7257
##
##        'Positive' Class : 0
```

Resampling

Resampling is the method that consists of drawing repeated samples from the original data samples. The process of Resampling is a nonparametric method of statistical inference. Resampling involves selecting randomized cases with replacement from the original data sample in such a manner that each number of the sample drawn has several cases similar to the original data sample. Due to replacement, the illustrated number of samples used by the Resampling method consists of repetitive cases.

We have used 2 out of 3 cross-validation techniques; those are validation set approach and K-fold cross-validation. We left The leave one out because we have 68000 rows in our data; hence, it is impractical to use the leave one out method because it will run 68000 multiplied by 68000; therefore, we used the k fold and validation set approach only.

The Validation set Approach

The validation set approach consists of randomly splitting the data into two sets: one set is used to train the model, and the remaining other set is used to test the model.

The process works as follow:

1. Build (train) the model on the training data set
2. Apply the model to the test data set to predict the outcome of new unseen observations
3. Quantify the prediction error as the mean squared difference between the observed and the predicted outcome values.

Leave one out cross-validation - LOOCV

This method works as follow:

1. Leave out one data point and build the model on the rest of the data set
2. Test the model against the data point that is left out at step 1 and record the test error associated with the prediction
3. Repeat the process for all data points
4. Compute the overall prediction error by taking the average of all these test error estimates recorded at step 2.

K-fold cross-validation

The k-fold cross-validation method evaluates the model performance on different subsets of the training data and calculates the average prediction error rate. The algorithm is as follow:

1. Randomly split the data set into k-subsets (or k-fold) (for example, five subsets)
2. Reserve one subset and train the model on all other subsets
3. Test the model on the reserved subset and record the prediction error
4. Repeat this process until each of the k subsets has served as the test set.
5. Compute the average of the k recorded errors. This is called the cross-validation error serving as the performance metric for the model.

Applying polynomial regression on our logistic regression model

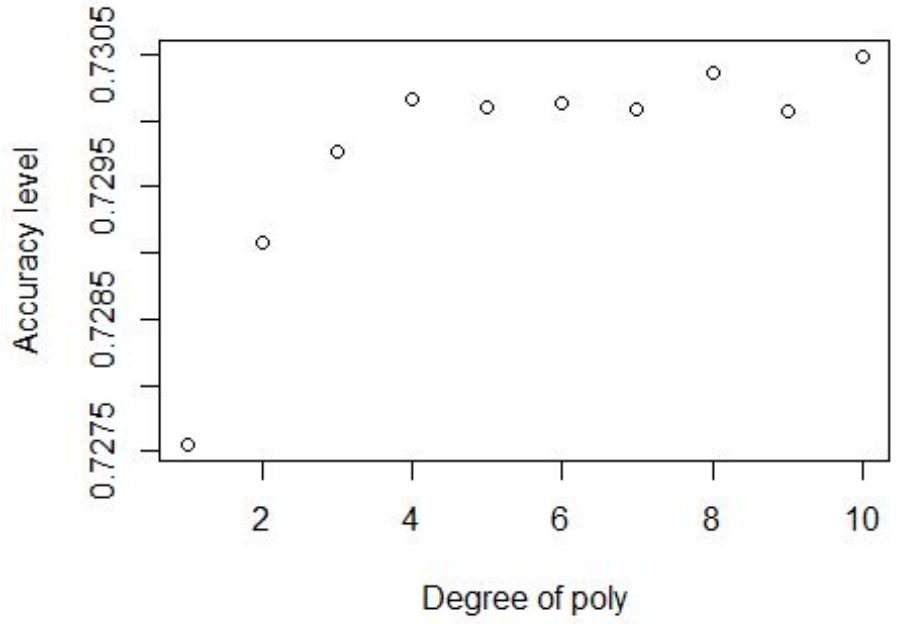
logistic regression with the whole dataset

In the below code, we tried to implement polynomial regression by varying the polynomial degree from 1 -10 in the entire data set. We can see this in the graph below. First, accuracy increased from polynomial degree 1 to 4. Then, it became constant from 4 to 10 with slight variation.

```
## logistic regression with whole dataset
acc
```

```
## [[1]]
## [1] 0.7275537
##
## [[2]]
## [1] 0.7290847
##
## [[3]]
## [1] 0.7297699
##
## [[4]]
## [1] 0.7301636
##
## [[5]]
## [1] 0.7301053
##
## [[6]]
## [1] 0.7301344
##
## [[7]]
## [1] 0.7300907
##
## [[8]]
## [1] 0.7303531
##
## [[9]]
## [1] 0.7300761
##
## [[10]]
## [1] 0.7304844
```

```
plot(unlist(acc), xlab="Degree of poly",ylab="Accuracy level")
```



Applying polynomial regression on our logistic regression model, with - validation set approach

We split the data into two halves of training and testing datasets for the validation set approach. We tried to fit the model on training data and using the predict function. Then We attempted to test model performance on the testing dataset.

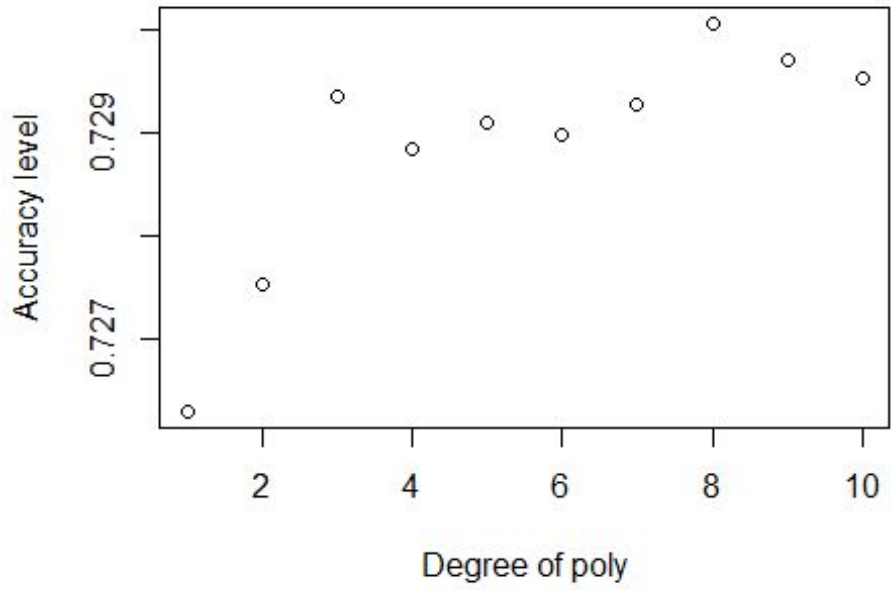
In the code below, we tried to implement polynomial regression with varying the degree of polynomial from 1 -10 in whole data set as we can see in graph blew at first accuracy increased from polynomial degree 1 to 3 then it became almost constant from 4 to 10 with slight variation.

1. Randomly divide the available set of observations into two parts, a training set and a validation set or hold-out set.
2. Fit the model on the training set.
3. Use the resulting fitted model to predict the responses for the observations in the validation set.
4. The resulting validation set error rate is typically assessed using the MSE in the case of a quantitative response. This provides an estimate of the test error rate.

```
#---------logistic regression with - validation set approach

acc


## [1] 0.7262998
```

```
## [1] 0.7275246
## [1] 0.7293617
## [1] 0.7288368
## [1] 0.7290992
## [1] 0.7289826
## [1] 0.7292742
## [1] 0.7300615
## [1] 0.7297116
## [1] 0.7295366
```

```
plot(unlist(acc), xlab="Degree of poly",ylab="Accuracy level")
```



Applying polynomial regression on our logistic regression model, with - k- fold

For the k- fold cross-validation approach, we used the cv.glm() function for providing k value(10) as our data set has around 70krows; hence it is better to have standard ten-fold cross-validation and to get error rate.

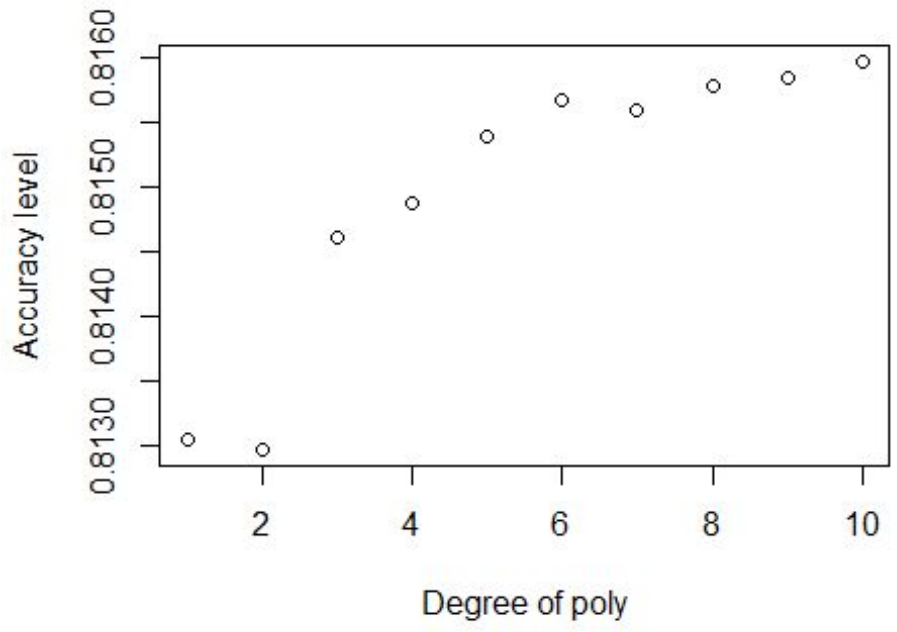
In the code below, we tried to implement polynomial regression with varying the degree of polynomial from 1 -10 in whole data set as we can see in graph blew at first accuracy increased from polynomial degree 1 to 6 then it became almost constant from 6 to 10 with slight variation.

```
#---------logistic regression with - k- fold---------------------------
```

```
## [1] 0.1869547
## [1] 0.1870325
## [1] 0.1853901
```

```
## [1] 0.1851219
## [1] 0.1846109
## [1] 0.1843316
## [1] 0.1843971
## [1] 0.1842164
## [1] 0.1841497
## [1] 0.1840271
```

```
plot(unlist(acc), xlab="Degree of poly",ylab="Accuracy level")
```



Naive Bayes with the Whole dataset

Naive Bayes is a probabilistic model. We can't introduce the polynomials; hence we made the standard approach of applying naive Bayes on the whole dataset, and we received an Accuracy of 71.99%.

```
NAIVE BAYES WITH WHOLE dataset
model <- naiveBayes(data$cardio~ .,
                    data = data)
```

```
## [1] "Accuracy 0.719913685008602"
```

Naive Bayes with validation set dataset

We split the data into 75% and 25% for this cross-validation approach, fitted the model on training data, and then tested it on testing data and received slightly less 71.87%. This may be because the validation set approach overestimates the error rate.

```
NAIVE BAYES WITH validation set dataset
```

```
## [1] "Accuracy 0.714568995684125"
```

```
## [1] "Accuracy 0.72148133748056"
```

```
confusionMatrix(Predict , training$cardio)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 21233  9572
##          1  4755 15880
##
##                Accuracy : 0.7215
##                  95% CI : (0.7176, 0.7254)
##     No Information Rate : 0.5052
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4418
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8170
##             Specificity : 0.6239
##          Pos Pred Value : 0.6893
##          Neg Pred Value : 0.7696
##              Prevalence : 0.5052
##          Detection Rate : 0.4128
##    Detection Prevalence : 0.5989
##       Balanced Accuracy : 0.7205
##
##        'Positive' Class : 0
##
```

Naive Bayes with K - FOLD

For the k- fold cross-validation approach, we used trainControl() function for providing k value(10) as our data set has around 70krows; hence it is better to have standard ten-fold cross-validation to get error rate.

Using a confusion matrix, we got Accuracy (average):71.98%, similar to the whole dataset approach and higher than the validation set approach.

```
NAIVE BAYES WITH K - FOLD
print(paste('Accuracy',1-misClasificError))

## [1] "Accuracy 0.719913685008602"

confusionMatrix(nb.m1)

## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    0    1
##          0 41.2 18.7
##          1  9.3 30.8
##
##  Accuracy (average) : 0.72
```

Cross-validation with KNN

KNN with whole dataset approach

KNN is a probabilistic model. We can't introduce the polynomials; hence we made the standard approach of applying knn on the whole dataset and we received Accuracy 73.08%

```
#pred <- knn(trainX,testX,trainY,k=26)
#confusion matrix
print(table(pred,testY))

##     testY
## pred     0     1
##    0 27128 10947
##    1  7522 22989

#accuracy

print(accuracy(table(pred,testY)))

## [1] 73.07176
```

KNN with validation set dataset

For this cross validation approach we split the data into 50% and 50% and fitted the model on training data and then tested it on testing data and received slightly less 70.51% this may be because the validation set approach overestimates the error rate.

```r
#fitting the model using k=26
pred <- knn(trainX,testX,trainY,k=26)
#confusion matrix
print(table(pred,testY))
```

```
##     testY
## pred     0     1
##    0 13293  6220
##    1  4006 10774
```

```r
#accuracy

print(accuracy(table(pred,testY)))
```

```
## [1] 70.1805
```

K-fold cross validation for KNN-

Cross-validation is a widely-used approach of machine learning, which addresses this training and test data problem, while also using all the data for checking the predictive accuracy. This is achieved by breaking the data into a variety of folds. If we have N folds, then the first step of the algorithm is to train the algorithm using (N−1) of the folds, and test the algorithm's accuracy on the single left-out fold. This is then repeated N times until each fold has been used as in the *test* set. If we have M parameter settings to try out, then this is accomplished in an outer loop, so we have to fit the algorithm a total of N×M times.

We will use the createFolds function from the caret package to make 5 folds and 10 folds of our data. Here users may be confused by the fact that the createFolds function uses the same letter 'k' as the 'k' in k-nearest neighbors. These 'k' are totally unrelated. The caret function createFolds is asking for how many folds to create the N from above. The 'k' in the knn function is for how many closest observations to use in classifying a new sample. Here we will create 5 and 10 folds:

```
> sapply(idx, length)
Fold1 Fold2 Fold3 Fold4 Fold5
13718 13717 13717 13717 13717
> idx[2]
$Fold2
   [1]     2    26    31    38    40    58    60    64    67    73    91    93   101   108   113
  [16]   121   129   137   138   142   143   144   151   165   166   167   170   173   184   187
  [31]   195   197   201   210   212   219   220   222   223   226   231   234   239   242   245
  [46]   251   255   262   266   269   282   285   291   296   297   300   302   325   335   341
  [61]   342   344   347   348   349   351   363   367   371   372   375   376   377   380   383
  [76]   390   402   407   408   413   416   426   432   438   440   441   447   451   452   455
  [91]   456   460   462   465   466   472   473   476   482   489   490   499   507   511   515
 [106]   516   537   548   550   552   556   558   565   575   578   581   584   587   589   592
 [121]   597   602   606   613   614   629   643   653   660   661   662   663   670   672   676
 [136]   677   678   682   684   689   692   697   700   704   705   707   717   725   729   736
```

Fig: Displays how the folds are divided and a sample of the second fold when using 5 folds.

```
#### Cross Validation for KNN

##   iteration error5Fold accuracy5Fold
## 1         1      0.293      70.69544
## 2         2      0.295      70.45272
## 3         3      0.289      71.09426
## 4         4      0.291      70.89014
## 5         5      0.296      70.43085

ggplot(Fold5, aes(x = iteration, y = accuracy5Fold)) +
  geom_line()
```

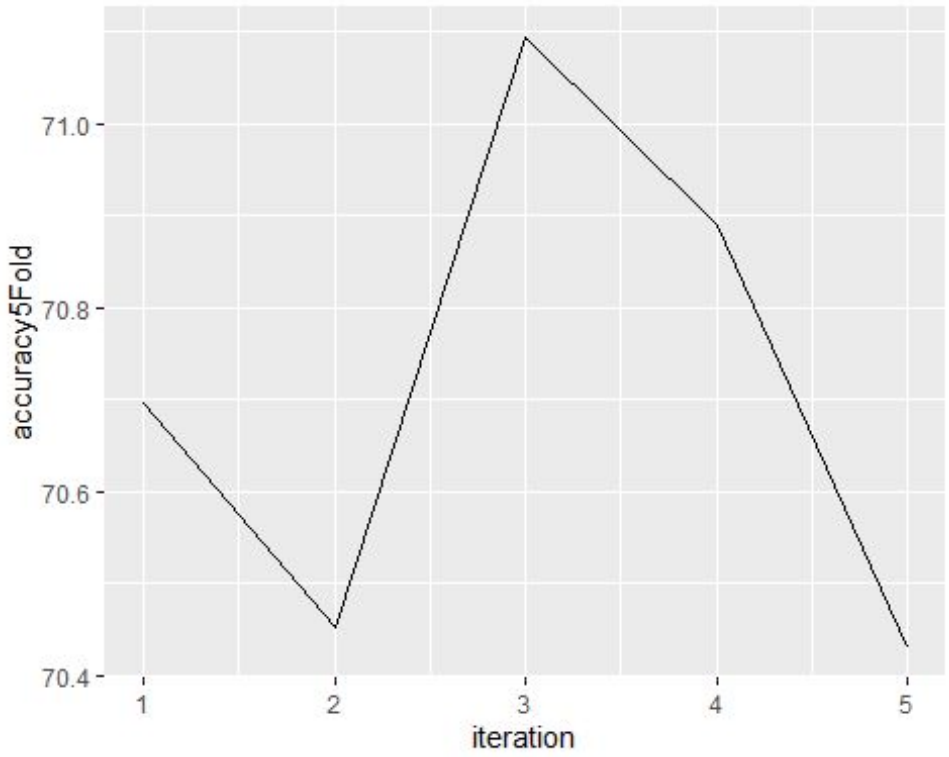


Fig: The variation in accuracy for each fold.

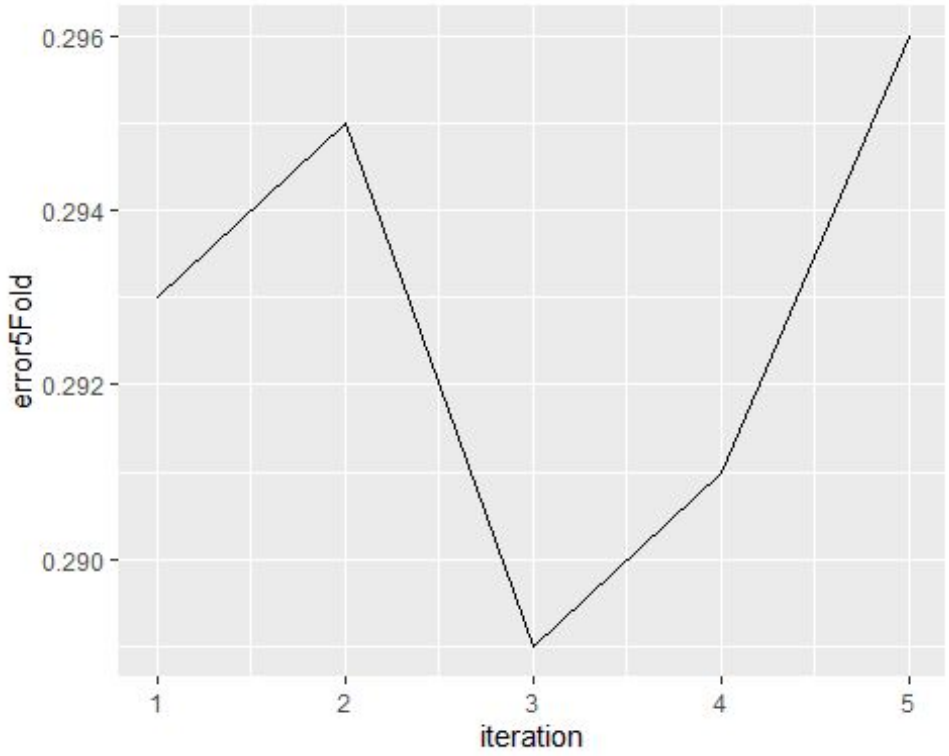


Fig: The variation in error rate for each fold

So we can see there is some variation for each fold, with errors rates hovering around 0.28 - 0.29 and accuracy around 70-71.

```
#10 fold cross validation
```

```
##    iteration error 10 Fold accuracy 10 Fold
## 1          1       0.288       71.20572
## 2          2       0.290       71.04534
## 3          3       0.287       71.33275
## 4          4       0.289       71.05992
## 5          5       0.294       70.57880
## 6          6       0.295       70.50160
## 7          7       0.295       70.48702
## 8          8       0.286       71.40566
## 9          9       0.292       70.79749
## 10        10       0.296       70.38927
```

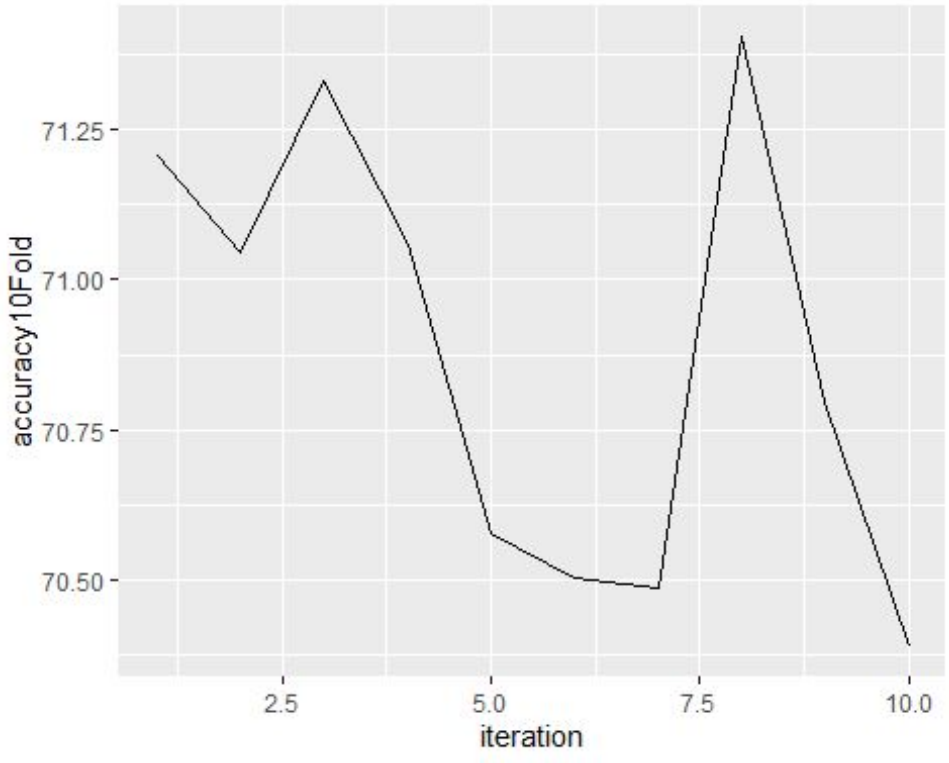


Fig: The variation in accuracy for each fold

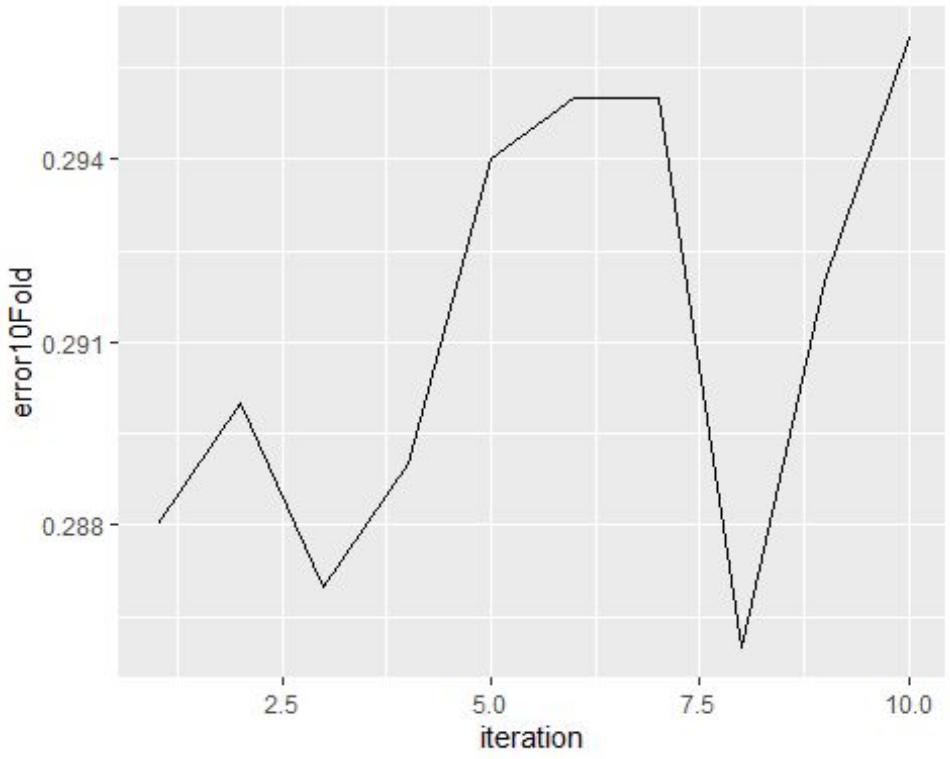


Fig: The variation in error rate for each fold

So we can see there is some variation for each fold, with errors rates hovering around 0.28 - 0.29 and accuracy around 70-71.

Quantifying parameter uncertainty using bootstrapping on logistic regression model

Creating function for computing the statistic by using the boot() function, which is part of the boot library, to perform the bootstrap by repeatedly sampling observations from the data set with replacement.

```
lr.mod <- glm(cardio ~ age + height +
                  weight + ap_hi + ap_lo,
            data = data ,family = "binomial")
summary(lr.mod)
##
## Call:
## glm(formula = cardio ~ age + height + weight + ap_hi + ap_lo,
```

```
##    family = "binomial", data = data)
##
## Deviance Residuals:
##    Min         1Q   Median       3Q    Max
## -3.8050  -0.9492  -0.3372   0.9629   2.7448
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.070e+01  2.214e-01 -48.308  < 2e-16 ***
## age            1.517e-04  3.662e-06  41.420  < 2e-16 ***
## height   -9.008e-03  1.168e-03  -7.715 1.21e-14 ***
## weight         1.318e-02  6.832e-04  19.293  < 2e-16 ***
## ap_hi        5.812e-02  9.221e-04  63.035  < 2e-16 ***
## ap_lo          1.114e-02  1.439e-03   7.740 9.94e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##    Null deviance: 95073  on 68585  degrees of freedom
## Residual deviance: 78162  on 68580  degrees of freedom
## AIC: 78174
##
## Number of Fisher Scoring iterations: 4
```

Function returns, or outputs, an estimate for α based on applying to the observations indexed by the argument index.
estimate α using all 68586 observations.

```
# Setting up the non-parametric bootstrap
```

```r
logit.bootstrap <- function(data, indices) {
  d <- data[indices, ]
  fit <- glm(cardio ~ age + height +
             weight + ap_hi + ap_lo,
             data = d ,family = "binomial")

  return(coef(fit))
}
nrow(data)
## [1] 68586
logit.bootstrap(data=data, 1:68586)
##   (Intercept)         age         height      weight      ap_hi
## -10.696553453   0.000151674  -0.009007561   0.013180920   0.058123835
##          ap_lo
##   0.011138790
set.seed(956)
# using a sample to randomly select 100 observations from the range 1 to 100,
with replacement.
logit.bootstrap(data=data, sample(100, 68586, replace = TRUE))
##   (Intercept)         age         height      weight      ap_hi
## -1.041020e+01  1.277317e-04 -4.405855e-02  4.081500e-02  9.757447e-02
##          ap_lo
## -4.681876e-03
logit.boot <- boot(data=data, logit.bootstrap, R= 1000) # 1000 samples
logit.boot
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
```

```
## Call:
## boot(data = data, statistic = logit.bootstrap, R = 1000)
## Bootstrap Statistics :
##          original          bias  std. error
## t1* -10.696553453  8.612412e-04 0.2354005713
## t2*   0.000151674  7.631675e-08 0.0000035816
## t3*  -0.009007561 -5.039779e-05 0.0012125733
## t4*   0.013180920  1.938322e-05 0.0007064244
## t5*   0.058123835  2.148106e-05 0.0010254974
## t6*   0.011138790  1.771166e-05 0.0015139121
```

The confidence intervals with boot.ci

```
# Calculate confidence intervals for each coefficient


boot.ci(logit.boot, type="basic", index=1)  # intercept
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicate
##
## CALL :
## boot.ci(boot.out = logit.boot, type = "basic", index = 1)
##
## Intervals :
## Level    Basic
## 95%   (-11.16, -10.26 )
## Calculations and Intervals on Original Scale
boot.ci(logit.boot, type="basic", index=2)  # age
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
```

```
##
## CALL :
## boot.ci(boot.out = logit.boot, type = "basic", index = 2)
##
## Intervals :
## Level    Basic
## 95%   ( 0.0001,  0.0002 )
## Calculations and Intervals on Original Scale
boot.ci(logit.boot, type="basic", index=3)  # height
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = logit.boot, type = "basic", index = 3)
##
## Intervals :
## Level    Basic
## 95%   (-0.0112, -0.0065 )
## Calculations and Intervals on Original Scale
boot.ci(logit.boot, type="basic", index=4)  # weight
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = logit.boot, type = "basic", index = 4)
##
## Intervals :
## Level    Basic
```

```
## 95%   ( 0.0118,  0.0145 )
## Calculations and Intervals on Original Scale
boot.ci(logit.boot, type="basic", index=5)  # ap_hi
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = logit.boot, type = "basic", index = 5)
##
## Intervals :
## Level    Basic
## 95%   ( 0.056,  0.060 )
## Calculations and Intervals on Original Scale
boot.ci(logit.boot, type="basic", index=6)  # ap_lo
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
## CALL :
## boot.ci(boot.out = logit.boot, type = "basic", index = 6)
## Intervals :
## Level      Basic
## 95%   ( 0.0081,  0.0139 )
## Calculations and Intervals on Original Scale
```

To estimate the sampling distribution of regression coefficients and calculate the standard errors/confidence intervals of regression coefficients bootstrapping is used on logistic models.

The nonparametric bootstrap resamples more than once and arbitrarily draws observations with substitution (for example a few observations are drawn just a single time, others on different occasions and some never), and stores the coefficients. So, R = 1000 regression coefficients is taken. These 1000 coefficients would then be able to be utilized to figure their confidence intervals. As a pseudo-irregular number generator is utilized, seed is set to a discretionary number to guarantee the very same outcomes each time. Higher the replications more stable will

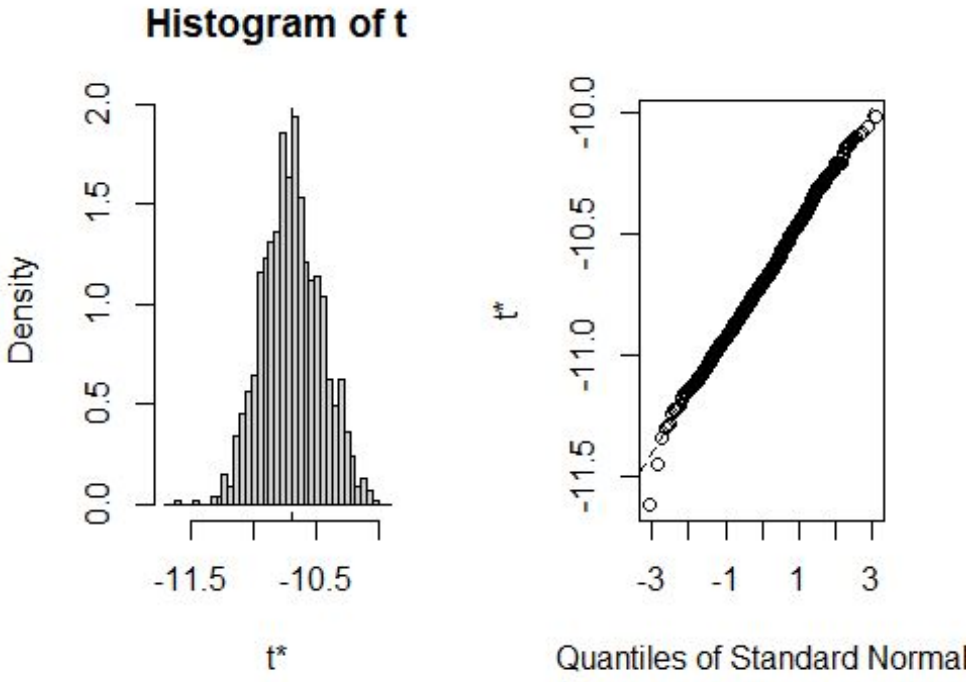
be the estimates. The boot () function gives the "bias" which is the difference between regression coefficients of the model and the mean of the bootstrap samples.

When performing the bootstrap, you are not interested in a single bootstrap sample, but in the distribution of statistics (e.g., regression coefficients) over, say, 1000 bootstrap samples. The bigger your sample size, the higher the number of iterations should be to ensure that every observation is taken into account.

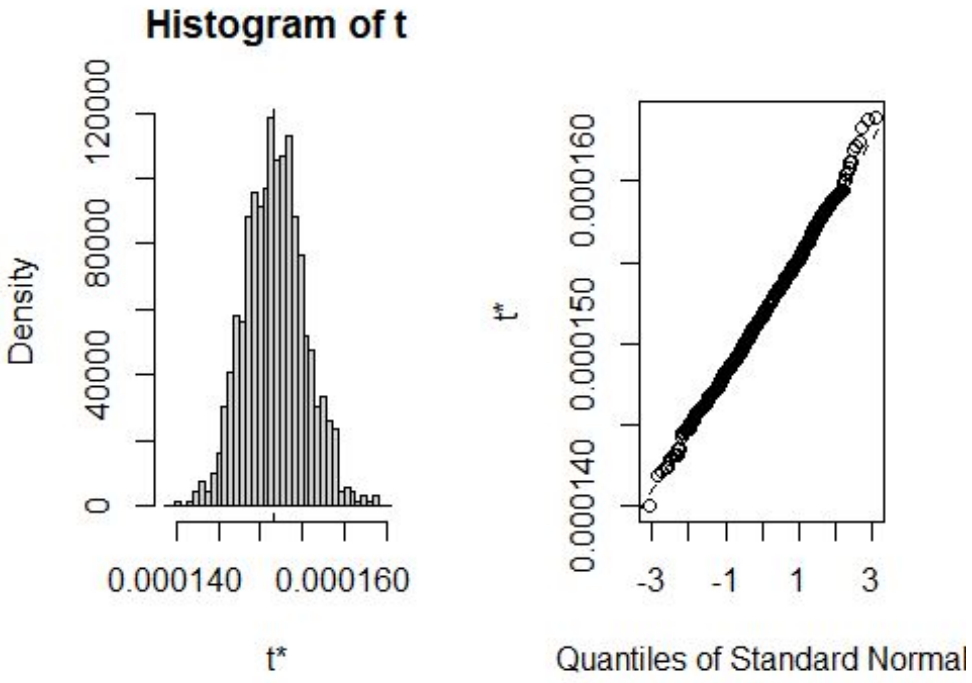
When implementing bootstrap, a single bootstrap sample is not of use, 1000 bootstrap samples will give more stable estimates. The greater the sample size, the higher the number of R iterations should be for considering all the observations. Also, when the number of replications is high, the bootstrap will yield very similar confidence intervals and point estimates.

The bootstrap outputs give the original regression coefficients ("original") and their bias. Values of the original coefficients and bootstrapped has a difference which is known as bias value. Standard errors which are larger than the original standard errors can be displayed. It also gives the confidence intervals on the original scale.

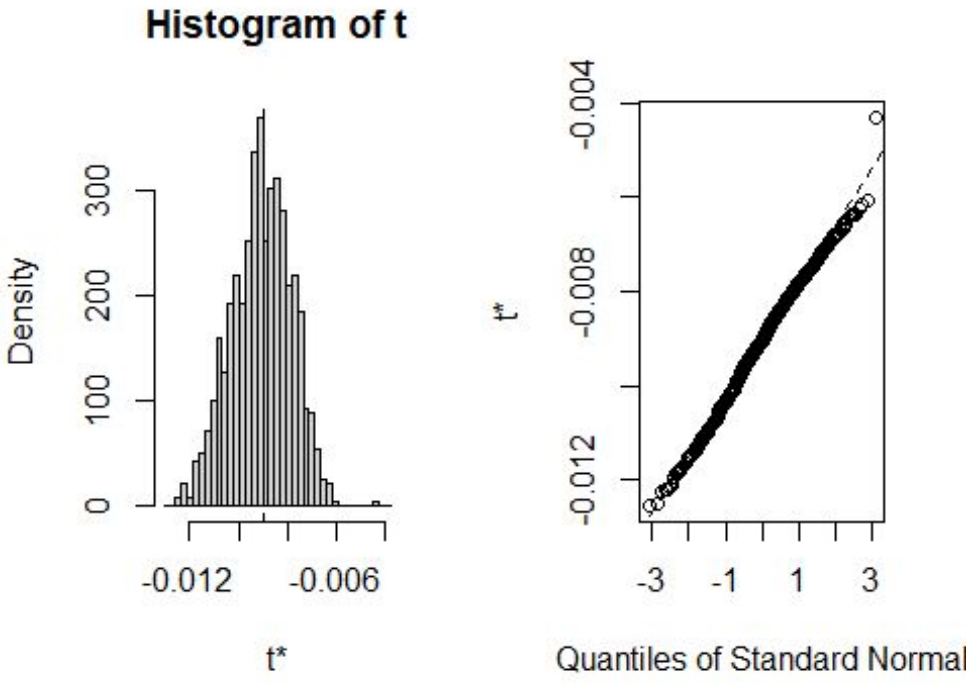
```
plot(logit.boot, index=1) # intercept
```



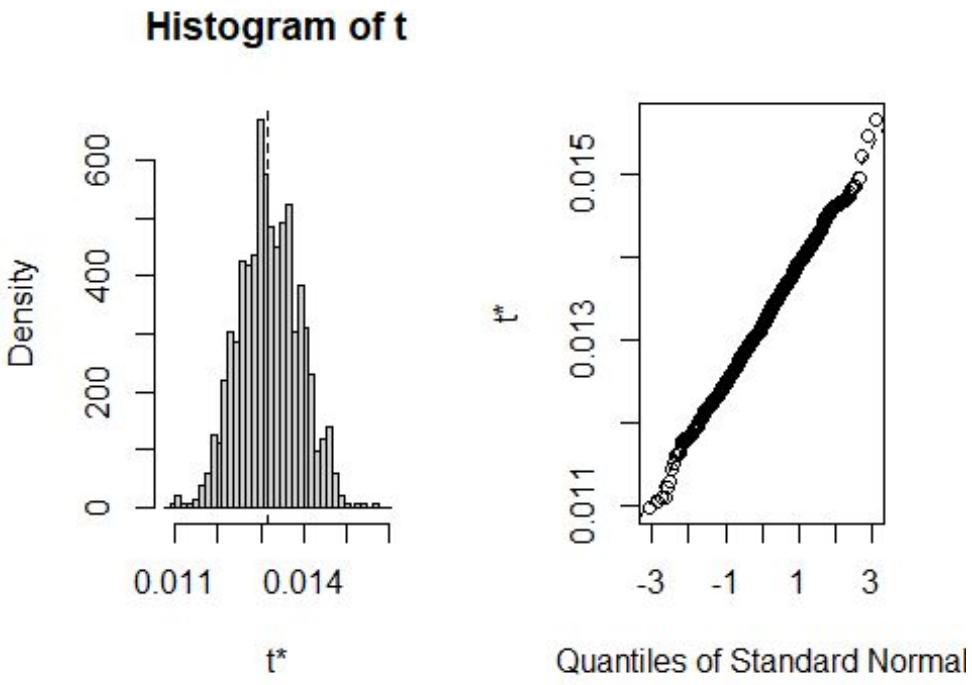

```
plot(logit.boot, index=2) # age
```



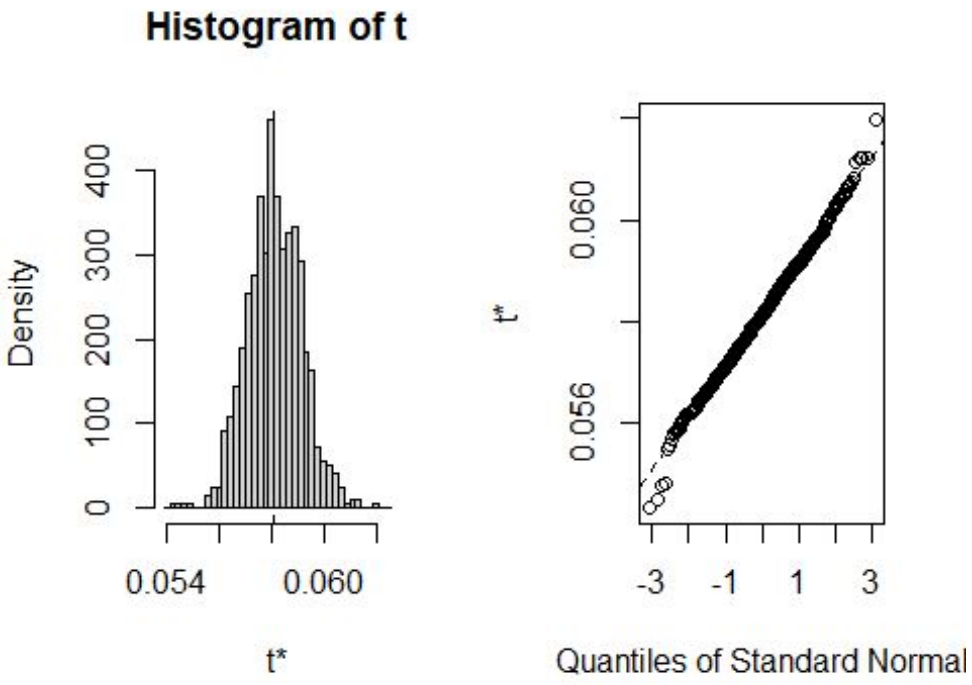

```
plot(logit.boot, index=3) # height
```




```
plot(logit.boot, index=4) # weight
```




```
plot(logit.boot, index=5) # ap_hi
```




```
plot(logit.boot, index=6) # ap_lo
```

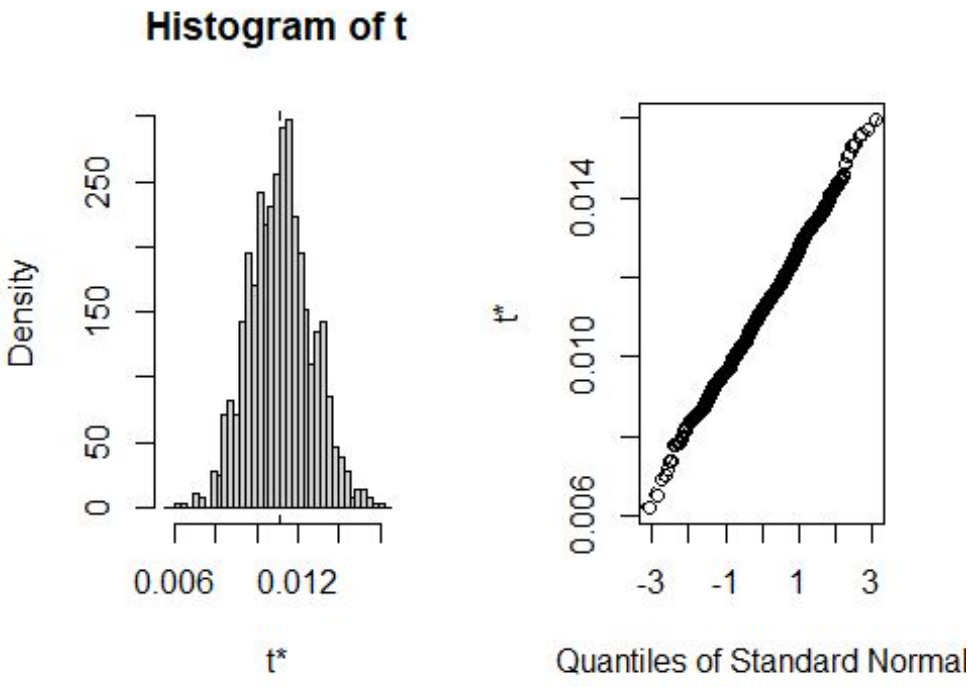


Fig: plot of bootstrap estimates for t

BMI Variable Creation

```
newdata <- data

newdata$bmi <- (newdata$weight)/(newdata$height/100)**2
newdata$height = NULL
newdata$weight = NULL
ndata <- newdata[,c(1:9,11,10)]
```

Subset selection methods

Stepwise Best Subset selection

To perform best subset selection, we fit a separate least squares regression best subset for each possible combination of the p predictors. That is, we fit all p models selection that contain exactly one predictor, all combinations of predictor variables = p(p−1)/2 models that contain exactly two predictors, and so forth. We then look at all of the resulting models, with the goal of identifying the one that is best.

here as we can see that the most important predictor variable is ‘ap_hi’ second is ‘Age’ and least is ‘gender’ means a person has higher chances of having heart related problems if he and she

1. has high blood pressure level

2. has a higher age

3. has higher cholesterol level

4. has obesity

least impactful predictor variables are if he or she has

1. Normal glucose level

2. Gender doesn't affect heart problems

3. smoking is not affecting heart

```
library(leaps)
# Stepwise selection

full.model <-glm(cardio~ ., data = data ,family =binomial )

sub.models <- regsubsets(cardio~ ., data = data , nvmax = 13)
summary(sub.models)
```

```
## Subset selection object
## Call: regsubsets.formula(cardio ~ ., data = data, nvmax = 13)
## 13 Variables  (and intercept)
##              Forced in Forced out
## age              FALSE      FALSE
## gender2          FALSE      FALSE
## height           FALSE      FALSE
## weight           FALSE      FALSE
## ap_hi            FALSE      FALSE
## ap_lo            FALSE      FALSE
## cholesterol2     FALSE      FALSE
## cholesterol3     FALSE      FALSE
## gluc2            FALSE      FALSE
## gluc3            FALSE      FALSE
## smoke1           FALSE      FALSE
## alco1            FALSE      FALSE
## active1          FALSE      FALSE
## 1 subsets of each size up to 13
## Selection Algorithm: exhaustive
##          age gender2 height weight ap_hi ap_lo cholesterol2 cholesterol3
gluc2
## 1  ( 1 )  " " " "     " "    " "    "*"   " "   " "          " "
" "
## 2  ( 1 )  "*" " "     " "    " "    "*"   " "   " "          " "
" "
## 3  ( 1 )  "*" " "     " "    " "    "*"   " "   " "          "*"
" "
## 4  ( 1 )  "*" " "     " "    "*"    "*"   " "   " "          "*"
```

```
" "
## 5  ( 1 )  "*" " "     " "    "*"    "*"   " "   "*"          "*"
" "
## 6  ( 1 )  "*" " "     " "    "*"    "*"   " "   "*"          "*"
" "
## 7  ( 1 )  "*" " "     " "    "*"    "*"   "*"   "*"          "*"
" "
## 8  ( 1 )  "*" " "     " "    "*"    "*"   "*"   "*"          "*"
" "
## 9  ( 1 )  "*" " "     " "    "*"    "*"   "*"   "*"          "*"
" "
## 10  ( 1 ) "*" " "     "*"    "*"    "*"   "*"   "*"          "*"
" "
## 11  ( 1 ) "*" " "     "*"    "*"    "*"   "*"   "*"          "*"
" "
## 12  ( 1 ) "*" "*"     "*"    "*"    "*"   "*"   "*"          "*"
" "
## 13  ( 1 ) "*" "*"     "*"    "*"    "*"   "*"   "*"          "*"
"*"
##           gluc3 smoke1 alco1 active1
## 1  ( 1 )  " "   " "    " "   " "
## 2  ( 1 )  " "   " "    " "   " "
## 3  ( 1 )  " "   " "    " "   " "
## 4  ( 1 )  " "   " "    " "   " "
## 5  ( 1 )  " "   " "    " "   " "
## 6  ( 1 )  " "   " "    " "   "*"
## 7  ( 1 )  " "   " "    " "   "*"
## 8  ( 1 )  "*"   " "    " "   "*"
## 9  ( 1 )  "*"   " "    "*"   "*"
## 10  ( 1 ) "*"   " "    "*"   "*"
## 11  ( 1 ) "*"   "*"    "*"   "*"
## 12  ( 1 ) "*"   "*"    "*"   "*"
## 13  ( 1 ) "*"   "*"    "*"   "*"
```

Forward selection

Forward stepwise selection is a computationally efficient alternative to best subset selection. While the best subset selection procedure considers all 2^p possible models containing subsets of the p predictors, forward stepwise considers a much smaller set of models. Forward stepwise selection begins with a model containing no predictors, and then adds predictors to the model, one-at-a-time, until all of the predictors are in the model.

Here we tried to apply forward selection using 2 function step() and StepAIC() total number of predictor variables in our model without applying forward selection 13 including the the variables with 3 categorical values

As the summary shows that the full model has 13 predictor variables as well as forward model too has 13 predictor variables, means we can not exclude any of the predictor variable from our model.

Delta A vector of length two. The first component is the raw cross-validation estimate of prediction error. The second component is the adjusted cross-validation estimate. The adjustment is designed to compensate for the bias introduced by not using leave-one-out cross-validation.

```
# forward selection

set.seed(123)

#forward model using step function
forward.model <- step(lr.mod, direction = "forward")

## Start:  AIC=76850.42
## cardio ~ age + gender + height + weight + ap_hi + ap_lo + cholesterol +
##     gluc + smoke + alco + active

summary(forward.model)

##
## Call:
## glm(formula = cardio ~ age + gender + height + weight + ap_hi +
##     ap_lo + cholesterol + gluc + smoke + alco + active, family = binomial,
##     data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.0147  -0.9205  -0.3372   0.9328   2.5569
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.078e+01  2.483e-01 -43.433  < 2e-16 ***
## age           1.396e-04  3.717e-06  37.554  < 2e-16 ***
## gender2      -1.536e-02  2.218e-02  -0.693  0.48852
## height       -4.043e-03  1.354e-03  -2.987  0.00282 **
## weight        1.099e-02  6.957e-04  15.800  < 2e-16 ***
## ap_hi         5.662e-02  9.274e-04  61.046  < 2e-16 ***
## ap_lo         1.033e-02  1.452e-03   7.115 1.12e-12 ***
## cholesterol2  3.793e-01  2.738e-02  13.851  < 2e-16 ***
## cholesterol3  1.092e+00  3.577e-02  30.540  < 2e-16 ***
## gluc2         1.829e-02  3.627e-02   0.504  0.61399
## gluc3        -3.470e-01  3.961e-02  -8.759  < 2e-16 ***
## smoke1       -1.453e-01  3.488e-02  -4.168 3.08e-05 ***
## alco1        -2.187e-01  4.249e-02  -5.147 2.64e-07 ***
## active1      -2.297e-01  2.192e-02 -10.477  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 95073  on 68585  degrees of freedom
## Residual deviance: 76822  on 68572  degrees of freedom
## AIC: 76850
##
## Number of Fisher Scoring iterations: 4
```

```
#best forward.model based on AIC value
step.model <- stepAIC(lr.mod, direction = "forward",
                      trace = FALSE)
summary(step.model)
```

```
##
## Call:
## glm(formula = cardio ~ age + gender + height + weight + ap_hi +
##     ap_lo + cholesterol + gluc + smoke + alco + active, family = binomial,
##     data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.0147  -0.9205  -0.3372   0.9328   2.5569
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.078e+01  2.483e-01 -43.433  < 2e-16 ***
## age           1.396e-04  3.717e-06  37.554  < 2e-16 ***
## gender2      -1.536e-02  2.218e-02  -0.693  0.48852
## height       -4.043e-03  1.354e-03  -2.987  0.00282 **
## weight        1.099e-02  6.957e-04  15.800  < 2e-16 ***
## ap_hi         5.662e-02  9.274e-04  61.046  < 2e-16 ***
## ap_lo         1.033e-02  1.452e-03   7.115 1.12e-12 ***
## cholesterol2  3.793e-01  2.738e-02  13.851  < 2e-16 ***
## cholesterol3  1.092e+00  3.577e-02  30.540  < 2e-16 ***
## gluc2         1.829e-02  3.627e-02   0.504  0.61399
## gluc3        -3.470e-01  3.961e-02  -8.759  < 2e-16 ***
## smoke1       -1.453e-01  3.488e-02  -4.168 3.08e-05 ***
## alco1        -2.187e-01  4.249e-02  -5.147 2.64e-07 ***
## active1      -2.297e-01  2.192e-02 -10.477  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 95073  on 68585  degrees of freedom
## Residual deviance: 76822  on 68572  degrees of freedom
## AIC: 76850
##
## Number of Fisher Scoring iterations: 4
```

```r
# applying K fold on stepmodel with K = 10
cv.err =cv.glm(data ,step.model , K = 10)
print(cv.err$delta[1])
```

```
## [1] 0.1868078
```

```r
print(cv.err$delta[2])
```

```
## [1] 0.1868031
```

```r
#finding accuracy of step.model
p <- predict(step.model, data ,type='response')
p <- ifelse(p > 0.5,1,0)
accuracy <- mean(p == data$cardio)
print(paste('Accuracy',accuracy ))
```

```
## [1] "Accuracy 0.727714110751465"
```

Backward selection

It begins with the full least squares model containing all p predictors, and then iteratively removes the least useful predictor, one-at-a-time. It's completely opposite of forward selection where we increase the number of predictor variables one by one and find out the best model.

As the summary shows that the full model has 13 predictor variables but the model has 12 predictor variables, which means we can exclude the 'Gender' variable from our model, it's an unessesy variable, and removing this variable won't affect our model accuracy. here we received similar accuracy for both the full model as well as backward model that is around 72.8%.

Delta A vector of length two. The first component is the raw cross-validation estimate of prediction error. The second component is the adjusted cross-validation estimate. The adjustment is designed to compensate for the bias introduced by not using leave-one-out cross-validation.

```r
set.seed(123)

# backward selection using step function
backward <- step(lr.mod)
```

```
## Start:  AIC=76850.42
## cardio ~ age + gender + height + weight + ap_hi + ap_lo + cholesterol +
##     gluc + smoke + alco + active
##
##               Df Deviance   AIC
## - gender       1    76823 76849
## <none>              76822 76850
## - height       1    76831 76857
## - smoke        1    76840 76866
## - alco         1    76849 76875
```

```
## - ap_lo        1    76873 76899
## - gluc         2    76901 76925
## - active       1    76932 76958
## - weight       1    77075 77101
## - cholesterol  2    77930 77954
## - age          1    78269 78295
## - ap_hi        1    81214 81240
##
## Step:  AIC=76848.9
## cardio ~ age + height + weight + ap_hi + ap_lo + cholesterol +
##     gluc + smoke + alco + active
##
##               Df Deviance   AIC
## <none>             76823 76849
## - height       1    76837 76861
## - smoke        1    76843 76867
## - alco         1    76850 76874
## - ap_lo        1    76873 76897
## - gluc         2    76902 76924
## - active       1    76933 76957
## - weight       1    77076 77100
## - cholesterol  2    77932 77954
## - age          1    78269 78293
## - ap_hi        1    81215 81239
```

```
summary(backward)
```

```
##
## Call:
## glm(formula = cardio ~ age + height + weight + ap_hi + ap_lo +
##     cholesterol + gluc + smoke + alco + active, family = binomial,
##     data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.0125  -0.9206  -0.3371   0.9327   2.5570
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.072e+01  2.269e-01 -47.218  < 2e-16 ***
## age           1.395e-04  3.715e-06  37.550  < 2e-16 ***
## height       -4.472e-03  1.204e-03  -3.716 0.000203 ***
## weight        1.100e-02  6.956e-04  15.809  < 2e-16 ***
## ap_hi         5.660e-02  9.272e-04  61.048  < 2e-16 ***
## ap_lo         1.031e-02  1.452e-03   7.105 1.20e-12 ***
## cholesterol2  3.797e-01  2.738e-02  13.870  < 2e-16 ***
## cholesterol3  1.093e+00  3.577e-02  30.553  < 2e-16 ***
## gluc2         1.833e-02  3.627e-02   0.505 0.613291
## gluc3        -3.468e-01  3.961e-02  -8.755  < 2e-16 ***
## smoke1       -1.514e-01  3.375e-02  -4.487 7.21e-06 ***
```

```
## alco1        -2.202e-01  4.243e-02  -5.189 2.11e-07 ***
## active1      -2.297e-01  2.192e-02 -10.478  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 95073  on 68585  degrees of freedom
## Residual deviance: 76823  on 68573  degrees of freedom
## AIC: 76849
##
## Number of Fisher Scoring iterations: 4
```

```
# backward selection using best AIC value using StepAIC function
step.model <- stepAIC(lr.mod, direction = "backward", trace = FALSE)
summary(step.model)
```

```
##
## Call:
## glm(formula = cardio ~ age + height + weight + ap_hi + ap_lo +
##     cholesterol + gluc + smoke + alco + active, family = binomial,
##     data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.0125  -0.9206  -0.3371   0.9327   2.5570
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.072e+01  2.269e-01 -47.218  < 2e-16 ***
## age           1.395e-04  3.715e-06  37.550  < 2e-16 ***
## height       -4.472e-03  1.204e-03  -3.716 0.000203 ***
## weight        1.100e-02  6.956e-04  15.809  < 2e-16 ***
## ap_hi         5.660e-02  9.272e-04  61.048  < 2e-16 ***
## ap_lo         1.031e-02  1.452e-03   7.105 1.20e-12 ***
## cholesterol2  3.797e-01  2.738e-02  13.870  < 2e-16 ***
## cholesterol3  1.093e+00  3.577e-02  30.553  < 2e-16 ***
## gluc2         1.833e-02  3.627e-02   0.505 0.613291
## gluc3        -3.468e-01  3.961e-02  -8.755  < 2e-16 ***
## smoke1       -1.514e-01  3.375e-02  -4.487 7.21e-06 ***
## alco1        -2.202e-01  4.243e-02  -5.189 2.11e-07 ***
## active1      -2.297e-01  2.192e-02 -10.478  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 95073  on 68585  degrees of freedom
## Residual deviance: 76823  on 68573  degrees of freedom
## AIC: 76849
```

```
##
## Number of Fisher Scoring iterations: 4

# applying K fold on stepmodel with K = 10


cv.err =cv.glm(data ,step.model , K = 10)
print(cv.err$delta[1])

## [1] 0.1868013

print(cv.err$delta[2])

## [1] 0.186797

# finding accuracy of the forward model
p <- predict(step.model, data ,type='response')
p <- ifelse(p > 0.5,1,0)
accuracy <- mean(p == data$cardio)
print(paste('Accuracy',accuracy ))

## [1] "Accuracy 0.727816172396699"
```

Regularisation techniques

Rigged Regression

Ridge regression uses L2 regularisation to weight/penalise residuals when the parameters of a regression model are being learned. In the context of linear regression, it can be compared to Ordinary Least Square, L2 regularisation is a small addition to the Ordinary Least Square function that weights residuals in a particular way to make the parameters more stable. to avoid overfitting. here lr.mod is our logistic regression model, first we used model.matrix creates a model matrix, by expanding factors to a set of dummy variables, then we created 2 sets by dividing predictor and response variables df.x and df.y. creating a list of lambda values to find optimal lambda that is 'lambdas'

step 1 creating a rigged regression model to find out change in coefficient with change in lambda values using glmnet() function this function takes predictor and response variable as input and alpha = 0 represent rigged regression, family = binomial for logistic regression and output shown as cofficents vs lambda plot, we can see as the lambda value is increasing the value of coefficient are moving towards zero.

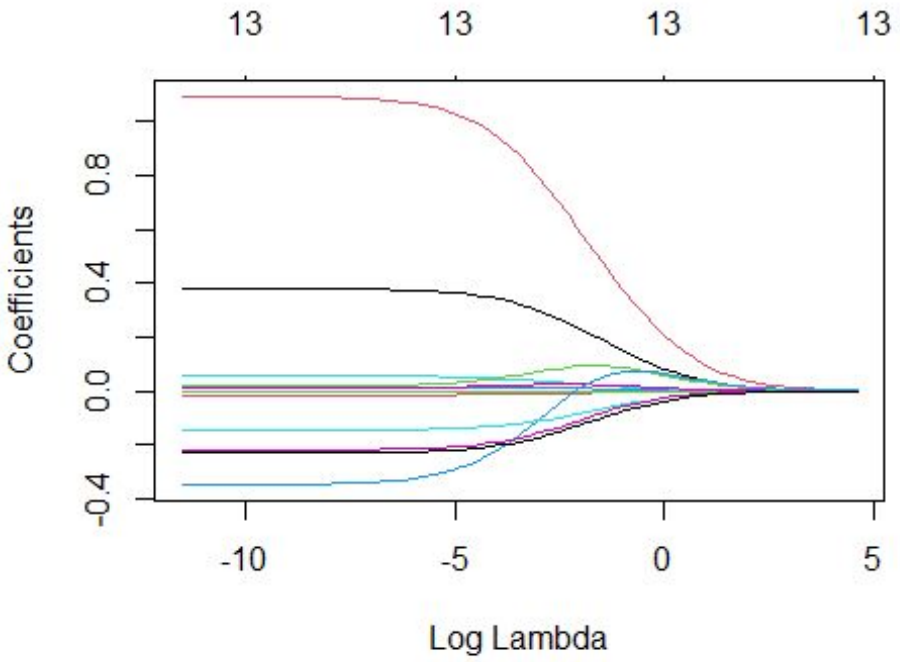
step 2 finding optimal lambda values using cv.glmnet() function, it returns a plot between deviance vs lambda value and based on the plot we find the best lambda value that is 0.0001584893

step 3 creating the best - rigged regression model using optimal lambda value and finding accuracy that is 0.727801592161666, and finding coefficients of ridge_reg2 model these are quite nearer to zero compared to lr.mod the original model.

Deviance A measure that plays the role of RSS for a broader class of models. The deviance is negative two times the maximized log-likelihood, the smaller the deviance, the better the fit.

Mean Square Error The MSE will be small if the predicted responses are very close to the true responses, and will be large if for some of the observations, the predicted and true responses differ substantially.

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 95073  on 68585  degrees of freedom
## Residual deviance: 76822  on 68572  degrees of freedom
## AIC: 76850
##
## Number of Fisher Scoring iterations: 4

#model.matrix creates a  model matrix, by expanding factors to a set of dummy
variables.

df.x = model.matrix(cardio~ .,df )[,-1]
df.y = df$cardio


lambdas <- 10^seq(-5, 2, by = .1)

ridge_reg = glmnet(df.x, df.y, nlambda = 100, alpha = 0 , family = binomial,
lambda = lambdas)
#assess.glmnet(ridge_reg, newx = df.test.x , newy = df.test.y )
plot(ridge_reg, xvar = "lambda")
```

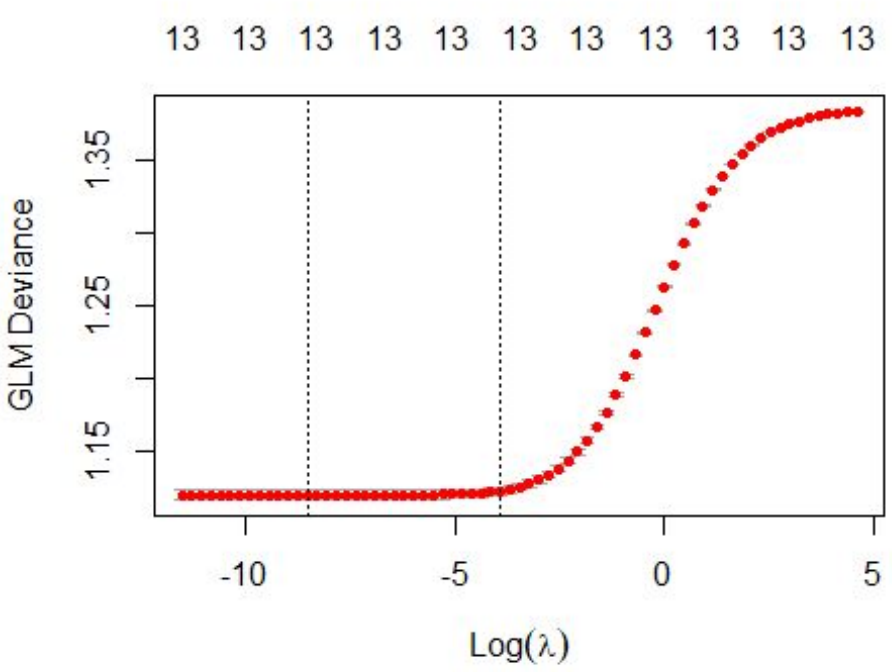


using c.glmnet() to find out optimal lambda, Cross-Validation For Glmnet Does k-fold cross-validation for glmnet, produces a plot, and returns a value for lambda, I have not defined nfold as default is 10

```
cv_ridge <- cv.glmnet(df.x, df.y, nlambda = 100, alpha = 0 , family = binomial, lambda = lambdas)
optimal_lambda <- cv_ridge$lambda.min
print(optimal_lambda)
```

```
## [1] 0.0001995262
```

```
plot(cv_ridge)
```




```
## [1] "Accuracy 0.727816172396699"
```

Lasso Regression

Lasso regression is a parsimonious model that performs L1 regularization. The L1 regularization adds a penalty equivalent to the absolute magnitude of regression coefficients and tries to minimize them.

first we used model.matrix creates a model matrix, by expanding factors to a set of dummy variables, then we created 2 sets by dividing predictor and response variables df.x and df.y. creating a list of lambda values to find optimal lambda that is 'lambdas'

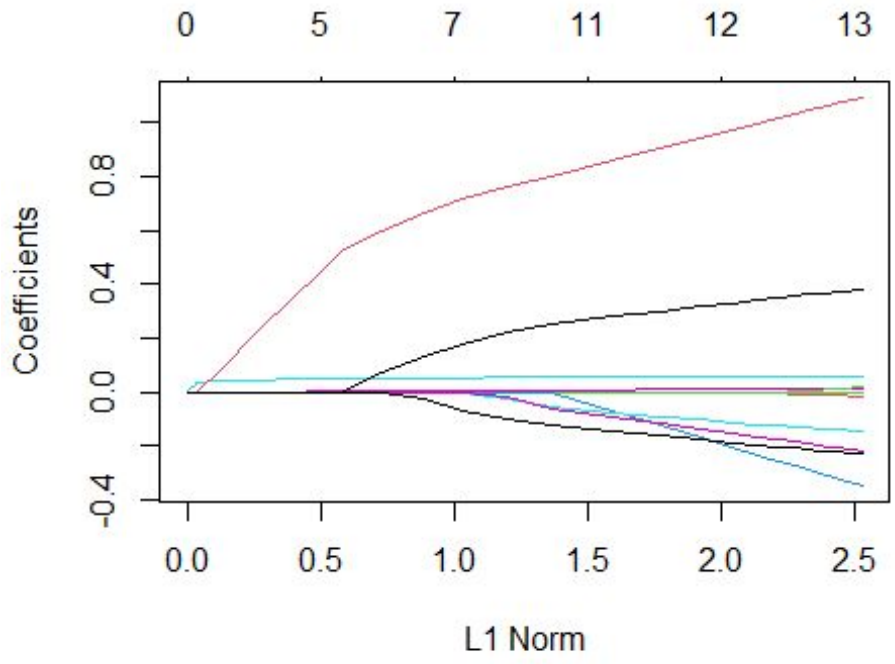
step 1. creating a lasso regression model to find out change in coefficient with change in lambda values using glmnet() function this function takes predictor and response variable as input and alpha = 1 represent lasso regression, family = binomial for logistic regression and output shown as cofficents vs lambda plot, we can see as the lambda value is increasing the value of coefficient are diverging towards zero one by one.

step 2 finding optimal lambda values using cv.glmnet() function, it returns a plot between deviance vs lambda value and based on the plot we find the best lambda value that is 0.0001995262.

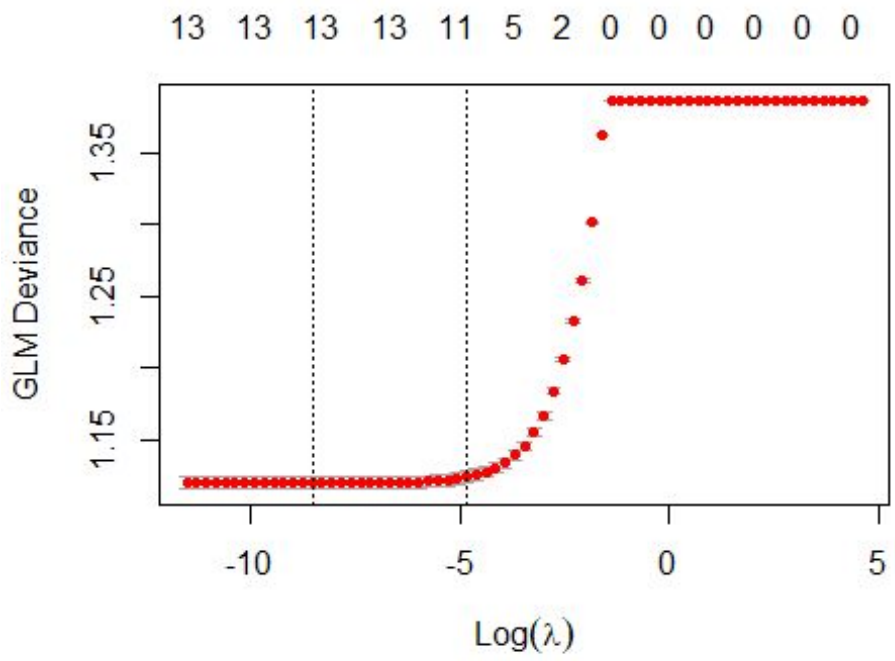
step 3 creating best - lasso regression model using optimal lambda value and finding accuracy that is 0.727816172396699, and finding coefficients of lasso.best we can see all the predictors are present in lasso.best model as none of the coefficient is 0 for optimal lambda, means we need all the predictors and none of this predictors are causing over-fitting.

```
#model.matrix creates a  model matrix, by expanding factors to a set of dummy
variables.
df.x = model.matrix(cardio~ .,df )[,-1]
df.y = df$cardio

#range of lambda values
lambdas <- 10^seq(-5, 2, by = .1)

#creating lasso model with range of lambda values to see change in daviance
lasso.mod = glmnet(df.x,df.y, alpha=1, family = "binomial", lambda= lambdas)
plot(lasso.mod)
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```




```
# founding optimal lambda value With K fold cross validation
set.seed (123)
cv.lasso =cv.glmnet(df.x,df.y, alpha=1, family = binomial, lambda = lambdas)
plot(cv.lasso)
```




```
# finding optimal lambda value with $lambda.min
optimal_lambda = cv.lasso$lambda.min
print(optimal_lambda)

## [1] 0.0001995262

# now with optimal lambda value creating best lasso model
lasso.best=glmnet(df.x,df.y,alpha=1,family = "binomial",lambda=
optimal_lambda)
coef(lasso.best)

## 14 x 1 sparse Matrix of class "dgCMatrix"
##                              s0
## (Intercept)  -1.077864e+01
## age           1.392153e-04
## gender2      -1.440709e-02
## height       -3.931590e-03
## weight        1.090283e-02
## ap_hi         5.655135e-02
## ap_lo         1.026672e-02
## cholesterol2  3.763708e-01
## cholesterol3  1.084569e+00
## gluc2         1.611585e-02
## gluc3        -3.379167e-01
## smoke1       -1.432309e-01
## alco1        -2.145019e-01
## active1      -2.269714e-01
```

```
lasso.pred = predict(lasso.best,s= optimal_lambda , newx = df.x )
fitted.results <- ifelse(lasso.pred > 0,1,0)
error <- mean(fitted.results != df.y)
print(paste('Accuracy',1- error))

## [1] "Accuracy 0.727816172396699"
```

Accommodating Non-Linearity

Linear relationship between the Response and Predictors when gets replaced by many Nonlinear smooth functions to model and capture the Non linearities in the data then these class of statistical Models are called as Generalized Additive Models. Below code is the implementation of Generalized Additive Model using the 'gam' package. In simple terms GAM's are Generalized version of Linear Models in which the Predictors depend Linearly or Nonlinearly on some Smooth Nonlinear functions like Splines, Polynomials or Step functions etc.

To relax the assumption of linearity function s () which is the shorthand for fitting smoothing splines technique is used.

Below code fits a GAM which is Nonlinear in 'age', 'bmi', 'ap_hi' and 'ap_lo' with 4 degrees of freedom respectively as they are fitted using Smoothing Splines, whereas it is Linear in Terms of variable 'cholesterol' and 'gluc'. Logistic Regression Model using GAMs has been applied for predicting the Probabilities of the Binary Response values which is 'cardio'. The summary () function produces a summary of the gam fit.

```
gamdata <- ndata

library(gam)  # To search on the gam function
logitgam.mod1 <-gam(cardio ~ s(age,df=4) + s(bmi,df=4)
                  + s(ap_hi ,df = 4) + s(ap_lo ,df = 4)+cholesterol + gluc
,data=gamdata,family=binomial)

logitgam.mod2 <-gam(cardio ~ s(age,df=4) + s(bmi,df=4)
                  + s(ap_hi ,df = 4) + ap_lo +cholesterol + gluc
,data=gamdata,family=binomial)

logitgam.mod3 <-gam(cardio ~ s(age,df=4) + s(bmi,df=4)
                  + ap_hi + ap_lo +cholesterol + gluc
,data=gamdata,family=binomial)

logitgam.mod4 <-gam(cardio ~ s(age,df=4) + bmi
                  + ap_hi + ap_lo +cholesterol + gluc
,data=gamdata,family=binomial)

summary(logitgam.mod1)

##
## Call: gam(formula = cardio ~ s(age, df = 4) + s(bmi, df = 4) + s(ap_hi,
```

```
##     df = 4) + s(ap_lo, df = 4) + cholesterol + gluc, family = binomial,
##     data = gamdata)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -2.7182 -0.8791 -0.4365  0.8714  2.3297
##
## (Dispersion Parameter for binomial family taken to be 1)
##
##     Null Deviance: 95072.95 on 68585 degrees of freedom
## Residual Deviance: 75827.59 on 68565 degrees of freedom
## AIC: 75869.59
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##                      Df Sum Sq Mean Sq  F value    Pr(>F)
## s(age, df = 4)        1   2135  2135.3 2073.465 < 2.2e-16 ***
## s(bmi, df = 4)        1   1180  1180.0 1145.826 < 2.2e-16 ***
## s(ap_hi, df = 4)      1   7972  7972.4 7741.369 < 2.2e-16 ***
## s(ap_lo, df = 4)      1     63    63.4   61.596 4.277e-15 ***
## cholesterol           2    910   455.0  441.844 < 2.2e-16 ***
## gluc                  2     66    32.8   31.837 1.513e-14 ***
## Residuals         68565  70611     1.0
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                   Npar Df Npar Chisq    P(Chi)
## (Intercept)
## s(age, df = 4)         3     122.07 < 2.2e-16 ***
## s(bmi, df = 4)         3      54.61 8.318e-12 ***
## s(ap_hi, df = 4)       3     916.78 < 2.2e-16 ***
## s(ap_lo, df = 4)       3      89.32 < 2.2e-16 ***
## cholesterol
## gluc
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(logitgam.mod2)
```

```
##
## Call: gam(formula = cardio ~ s(age, df = 4) + s(bmi, df = 4) + s(ap_hi,
##     df = 4) + ap_lo + cholesterol + gluc, family = binomial,
##     data = gamdata)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -2.7223 -0.8818 -0.4321  0.8724  2.3755
##
## (Dispersion Parameter for binomial family taken to be 1)
##
```

```
##     Null Deviance: 95072.95 on 68585 degrees of freedom
## Residual Deviance: 75874.63 on 68568 degrees of freedom
## AIC: 75910.63
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##                     Df Sum Sq Mean Sq  F value    Pr(>F)
## s(age, df = 4)       1   2137  2136.8 2074.379 < 2.2e-16 ***
## s(bmi, df = 4)       1   1187  1186.9 1152.190 < 2.2e-16 ***
## s(ap_hi, df = 4)     1   8130  8129.5 7891.903 < 2.2e-16 ***
## ap_lo                1     54    54.3   52.705 3.917e-13 ***
## cholesterol          2    914   456.8  443.448 < 2.2e-16 ***
## gluc                 2     67    33.3   32.351 9.053e-15 ***
## Residuals        68568  70632     1.0
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                   Npar Df Npar Chisq    P(Chi)
## (Intercept)
## s(age, df = 4)         3     122.49 < 2.2e-16 ***
## s(bmi, df = 4)         3      54.48  8.85e-12 ***
## s(ap_hi, df = 4)       3    1088.86 < 2.2e-16 ***
## ap_lo
## cholesterol
## gluc
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(logitgam.mod3)
```

```
##
## Call: gam(formula = cardio ~ s(age, df = 4) + s(bmi, df = 4) + ap_hi +
##     ap_lo + cholesterol + gluc, family = binomial, data = gamdata)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -3.7906 -0.9162 -0.3395  0.9318  2.6141
##
## (Dispersion Parameter for binomial family taken to be 1)
##
##     Null Deviance: 95072.95 on 68585 degrees of freedom
## Residual Deviance: 76846.39 on 68571 degrees of freedom
## AIC: 76876.39
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##                   Df Sum Sq Mean Sq  F value    Pr(>F)
## s(age, df = 4)     1   2127  2126.5 1806.921 < 2.2e-16 ***
```

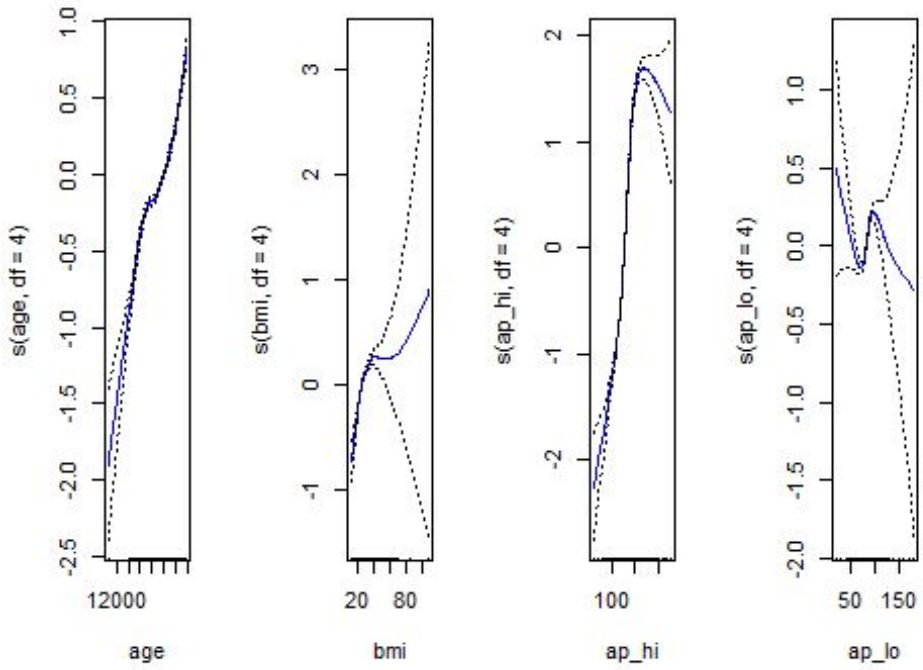
```
## s(bmi, df = 4)     1   1190  1190.1 1011.247 < 2.2e-16 ***
## ap_hi              1   7638  7637.7 6489.836 < 2.2e-16 ***
## ap_lo              1     58    58.3   49.504 1.998e-12 ***
## cholesterol        2   1004   502.0  426.521 < 2.2e-16 ***
## gluc               2     70    35.0   29.738 1.232e-13 ***
## Residuals      68571  80699     1.2
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Anova for Nonparametric Effects
##                Npar Df Npar Chisq    P(Chi)
## (Intercept)
## s(age, df = 4)       3    128.487 < 2.2e-16 ***
## s(bmi, df = 4)       3     52.836 1.987e-11 ***
## ap_hi
## ap_lo
## cholesterol
## gluc
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(logitgam.mod4)
```

```
##
## Call: gam(formula = cardio ~ s(age, df = 4) + bmi + ap_hi + ap_lo +
##     cholesterol + gluc, family = binomial, data = gamdata)
## Deviance Residuals:
##     Min      1Q  Median      3Q     Max
## -3.9145 -0.9137 -0.3483  0.9347  2.6264
##
## (Dispersion Parameter for binomial family taken to be 1)
##
##     Null Deviance: 95072.95 on 68585 degrees of freedom
## Residual Deviance: 76898.52 on 68574 degrees of freedom
## AIC: 76922.52
##
## Number of Local Scoring Iterations: NA
##
## Anova for Parametric Effects
##                   Df Sum Sq Mean Sq  F value    Pr(>F)
## s(age, df = 4)     1   2150  2149.9 1801.808 < 2.2e-16 ***
## bmi                1   1194  1193.9 1000.611 < 2.2e-16 ***
## ap_hi              1   7681  7680.7 6437.077 < 2.2e-16 ***
## ap_lo              1     60    60.5   50.701 1.086e-12 ***
## cholesterol        2   1006   502.9  421.465 < 2.2e-16 ***
## gluc               2     70    35.1   29.429 1.677e-13 ***
## Residuals      68574  81823     1.2
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Anova for Nonparametric Effects
##                 Npar Df Npar Chisq    P(Chi)
## (Intercept)
## s(age, df = 4)       3     128.83 < 2.2e-16 ***
## bmi
## ap_hi
## ap_lo
## cholesterol
## gluc
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

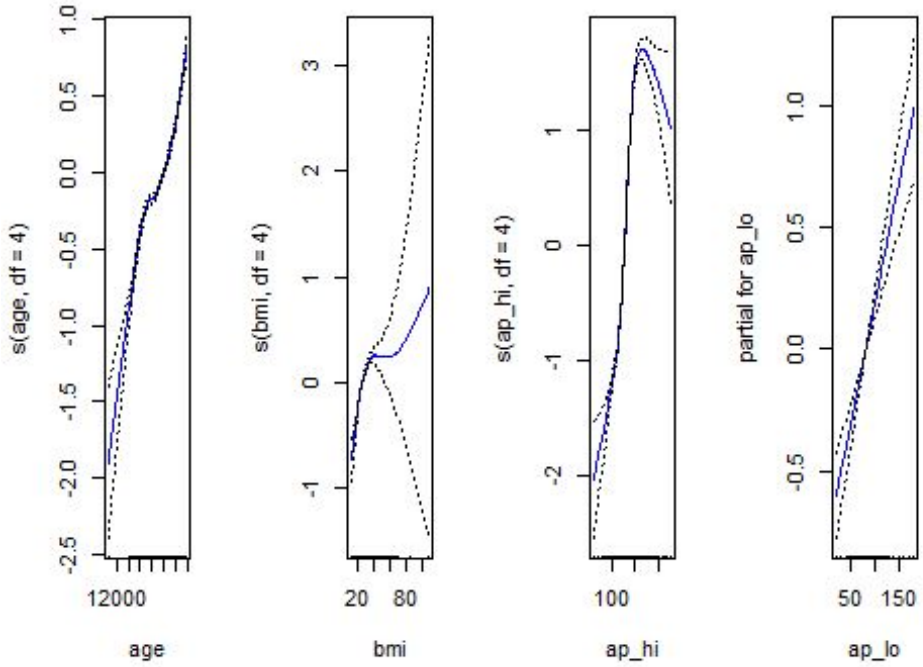
```
par(mfrow=c(1,4))
plot(logitgam.mod1, se=TRUE,col="blue")
```



The curvy shapes for the variables ‘age’ and ‘bmi is due to the Smoothing splines which models the Non linearities in the data. The dotted Lines around the main curve lines are the Standard Error Bands.

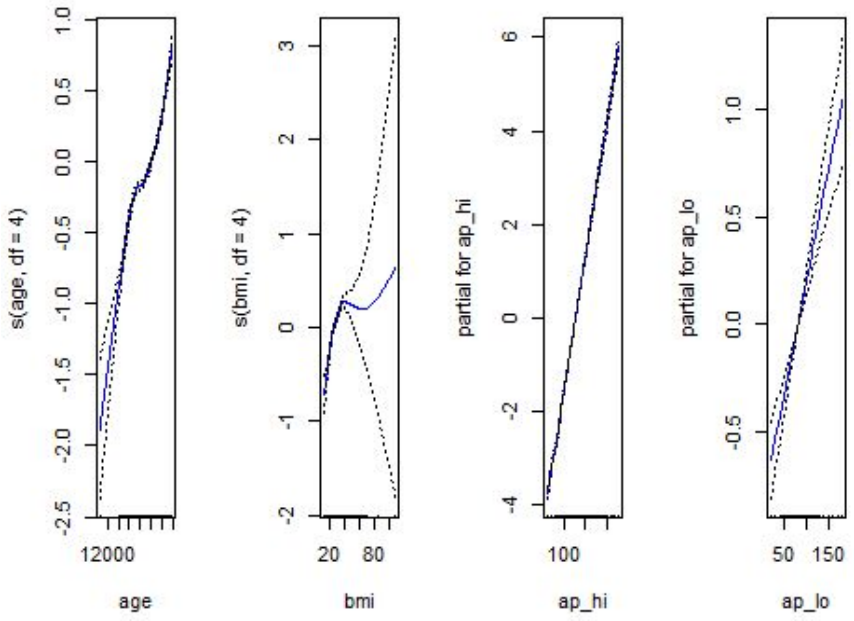
Fitting another model which is Linear in variable ‘ap_lo’.

```
par(mfrow=c(1,4))

plot(logitgam.mod2, se=TRUE,col="blue")
```



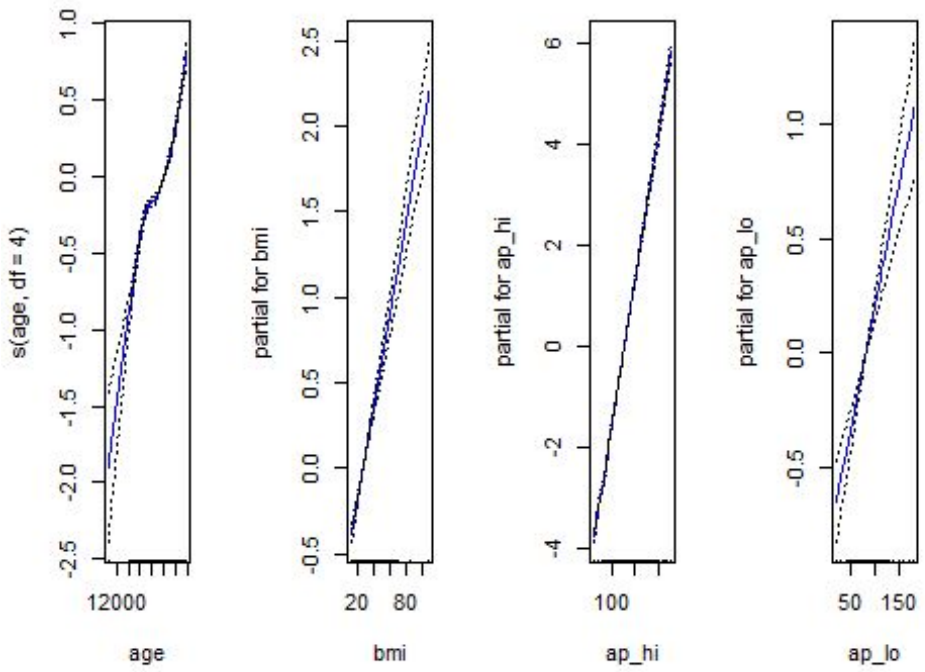
Fitting another model which is Linear in variable 'ap_lo', 'ap_hi'.

```
par(mfrow=c(1,4))

plot(logitgam.mod3, se=TRUE,col="blue")
```



Fitting another model which is Linear in variable ‘ap_lo’, ‘ap_hi’ and ‘bmi’.

```
par(mfrow=c(1,4))

plot(logitgam.mod4, se=TRUE,col="blue")
```



From the plot above for ‘bmi’, it is linear i.e., a straight line (a polynomial of degree 1).

Using anova() function for checking the goodness of fit for the above models, one which is Non Linear in ‘bmi’ and another which is Linear in ‘ap_lo’, ‘ap_hi’ and ‘bmi’.

```
anova(logitgam.mod1,logitgam.mod2,logitgam.mod3,logitgam.mod4 ,test =
"Chisq")

## Analysis of Deviance Table
##
## Model 1: cardio ~ s(age, df = 4) + s(bmi, df = 4) + s(ap_hi, df = 4) +
##     s(ap_lo, df = 4) + cholesterol + gluc
## Model 2: cardio ~ s(age, df = 4) + s(bmi, df = 4) + s(ap_hi, df = 4) +
##     ap_lo + cholesterol + gluc
## Model 3: cardio ~ s(age, df = 4) + s(bmi, df = 4) + ap_hi + ap_lo +
cholesterol +
##     gluc
## Model 4: cardio ~ s(age, df = 4) + bmi + ap_hi + ap_lo + cholesterol +
##     gluc
##   Resid. Df Resid. Dev      Df Deviance  Pr(>Chi)
## 1     68565      75828
## 2     68568      75875 -3.0001   -47.04 3.412e-10 ***
```

```
## 3     68571      76846 -3.0000  -971.76 < 2.2e-16 ***
## 4     68574      76899 -3.0001   -52.13 2.809e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Results indicate model 1 which is Nonlinear in terms of all the numeric variables is significant and better. Hence GAM is required which fits a Nonlinear function for numeric variables with degree of freedom = 4.

Model 1 which is Nonlinear in terms of all the numeric variables is significant and much better. Hence GAM is required which fits a Nonlinear function for variable 'ap_hi', 'ap_lo', 'age' and 'bmi'.

Summary

After selecting the project Heart Disease Risk Prediction from the list of project proposals through various project selection criteria the initial steps included descriptive analysis. Descriptive analysis which is an essential first step for conducting statistical analysis included analysis of each variable as well as combinations of variables. After importing data, categorization of data based on their data types was done. Histograms, bar plots were implemented for the predictor and the response variables. Also, the summary of the numerical data and distribution of categorical data were displayed.

Correlation matrix cor() function produced a matrix that contained all of the pairwise correlations among the predictors in a data set. Results include Weight and ap_hi and ap_low which were weakly positive correlations with age means there was some probability that with weight blood pressure increases. Also, ap_hi and ap_low have very strong positive correlation showing that both types of blood pressure rise together.

Further different types of models were implemented to get the best accuracy for the project. Implementation of K Nearest Neighbors resulted with accuracy which is highest at that point = 70.23299 when the k - value graph is plotted. For Logistic Regression model mean () function was used to compute the fraction of days for which the prediction was correct. After running the model on both halves first on test data and received an accuracy of 72.89% and again on the training dataset and received the accuracy of 72.75%, which means there was very low variance. Also, from Roc- AUC we received is 79.28% means the model has correctly predicted 79% true positive. Using Naive Bayes Classification accuracy of the presence or absence of cardiovascular disease resulted in 72.04% on test data and 71.91% on training data.

We have used 2 out of 3 cross-validation techniques; those are validation set approach and K-fold cross-validation. We left the leave one out because we have 68000 rows in our data; hence, it is impractical to use the leave one out method because it will run 68000 multiplied by 68000; therefore, we used the k fold and validation set approach only. For the k- fold cross-validation approach, we used the cv.glm() function for providing k value(10) as our data set has around 70krows; hence it is better to have standard ten-fold cross-validation and to get error rate. Using a confusion matrix, we got Accuracy (average):71.98%, similar to the whole dataset approach and higher than the validation set approach.

Quantifying parameter uncertainty using bootstrapping on a logistic regression model was done. The bootstrap outputs resulted in the original regression coefficients ("original") and their bias.

Values of the original coefficients and bootstrapped has a difference which is known as bias value. Standard errors which are larger than the original standard errors can be displayed. It also gives the confidence intervals on the original scale.

After Quantifying parameter uncertainty we studied Best subset selection method to find out most important predictor variables and least important predictor variables, and forward and backward subset selection and regularization techniques like Rigged and Lasso regression using glmnet() function to find out best model that and we received similar results for both Rigged and Lasso with "Accuracy 0.727816172396699" and "Accuracy 0.727616172396699" After that we used Using Generalized Additive Models in an exploratory fashion by relaxing the assumption of linearity for our numeric variables like BMI and Age to fit spline and find out best degree of freedom for the fit. Results indicated model 1 which is Nonlinear in terms of all the numeric variables is significant and much better. Hence GAM is required which fits a Nonlinear function for numeric variables with degree of freedom 4.

Reference

http://www.sthda.com/english/articles/38-regression-model-validation/157-cross-validation-essentials-in-r/

https://faculty.marshall.usc.edu/gareth-james/ISL/ISLR%20Seventh%20Printing.pdf

https://stats.libretexts.org/Bookshelves/Computing_and_Modeling/RTG%3A_Classification_Methods/4%3A_Numerical_Experiments_and_Real_Data_Analysis/Preprocessing_of_categorical_predictors_in_SVM%2C_KNN_and_KDC_(contributed_by_Xi_Cheng)

https://datatricks.co.uk/one-hot-encoding-in-r-three-simple-methods

https://www.rdocumentation.org/packages/e1071/versions/1.7-3/topics/naiveBayes

https://www.edureka.co/blog/naive-bayes-in-r/

https://www.pluralsight.com/guides/linear-lasso-and-ridge-regression-with-r