DA5030.Proj.Lohar

Sachin

11/25/2019

LATEX

CRISP-DM Methodology

Business Understanding

Diabetes is considered as one of the complex disease in the world and according to American Diabetes Association in 2015, around 9.4% of US population had diabetes this contribute to approximately 30 million people. However, according to World Health Organization the number of people with diabetes in the whole world has risen from 108 million in 1980 to 422 million in 2014 and the health expenditure in 2019 for diabetes caused at least USD 760 billion dollars.

For diabetes, most diagnostic methods present today are like black-box models. These models are unable to provide the reasons for underlying diagnosis to physicians; therefore, algorithms that can provide further insight are needed. So I would try to build a machine learning models on the available data to predict diabetes.

Personnel: I am going to work on this as a course project for DA5030 (Fall 2019).

Data: Data is freely available on Kaggle (https://www.kaggle.com/uciml/pima-indians-diabetes-database) and can be downloaded locally.

Computing resources: I have MacOS High Sierra (version 10.13.6) and can be used to perform all the tasks.

Software: To analyze the data, I need freely availble software like RStudio (R version 3.6.1) which is downloaded on the above computing resourse.

Business success criteria: This project would provides several machine learning models with its accuracy to predict diabetes on the provided data.

Data Understanding

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases and can be found on https://www.kaggle.com/uciml/pima-indians-diabetes-database

This dataset contains 9 columns and 768 observations. Columns are the medical predictors which includes Blood glucose level, blood pressure, skin thikness, insulin level, BMI, diabetes pedigree function, age and pregnancies and the last column is outcome which is target binary variable for diabetes (0 for no diabetes and 1 for diabetes). All 768 observations are from females at least 21 years old of Pima Indian heritage.

Details of the columns and its units:

Preganancies are the number of times pregnant (NP)

Glucose is a plasma glucose concentration after 2 h in an OGTT

BloodPressure Diastolic blood pressure (mmHg) (DBP)

SkinThickness Triceps skinfold thickness (mm) (TSFT)

Insulin Two-hour serum insulin (μU/mL) (2HSI)

BMI is Body Mass Index

DiabetesPedigreeFunction is a function for a genetic history of diabetes

Age in years

Outcomes are 0s and 1s which means patient has diabetes or not

Data Preparation

I would explore the data to check its attributes and its structure. I also need to clean the data to increse its quality required by selected machine learning algorithms. Data cleaning would include finding outliers and missing values. I would decide inputation strategy for missing values depending on how much missing data is there and structure of the column.

Modeling

As I have binary dependent varible to be predicted, I would choose logistic regression, K-Nearest Neighbours, Support Vector Machine and Naive Bayes. I would perform these tasks for each model seperately. For test design, I would split data randomly for training and testing, however I would keep the same ration of the dependent variable (Diabetes YES or NO) as my original data. Training data would have 75% of the original data and I would test the model on remaininig 25% of the data to access its accuracy and performance. I would also perfom a k-fold validation to validate their performance.

Evaluation

In evaluation I would review if there is any important factor or task that has somehow been overlooked and I would describe the decision as to how to proceed, along with the rationale.

Deployment

As this is a course project I actually can not deploy in a real-life scenario. However, if I get chance to use this models in future I would use them to predict a real-life scenario. For now I would use this model on my testing data only and monitor its performance.

Data Acquisition

Acquisition of data (e.g., CSV or flat file)

```r
# set working directory so I can upload all required files and documents
# and can save any output files.
setwd("/Users/sachinlohar/Desktop/DA5030/Project")
diab.data <- read.csv ( "diabetes.csv", header = TRUE, stringsAsFactors = FALSE)
dim(diab.data)
```

```
## [1] 768   9
```

Data Exploratioin

Exploratory data plots

```r
# get the structure of the data and see how data has been organized
str ( diab.data )
```

```
## 'data.frame':    768 obs. of  9 variables:
##  $ Pregnancies             : int  6 1 8 1 0 5 3 10 2 8 ...
##  $ Glucose                 : int  148 85 183 89 137 116 78 115 197 125 ...
##  $ BloodPressure           : int  72 66 64 66 40 74 50 0 70 96 ...
##  $ SkinThickness           : int  35 29 0 23 35 0 32 0 45 0 ...
##  $ Insulin                 : int  0 0 0 94 168 0 88 0 543 0 ...
##  $ BMI                     : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
##  $ DiabetesPedigreeFunction: num  0.627 0.351 0.672 0.167 2.288 ...
##  $ Age                     : int  50 31 32 21 33 30 26 29 53 54 ...
##  $ Outcome                 : int  1 0 1 0 1 0 1 0 1 1 ...
```

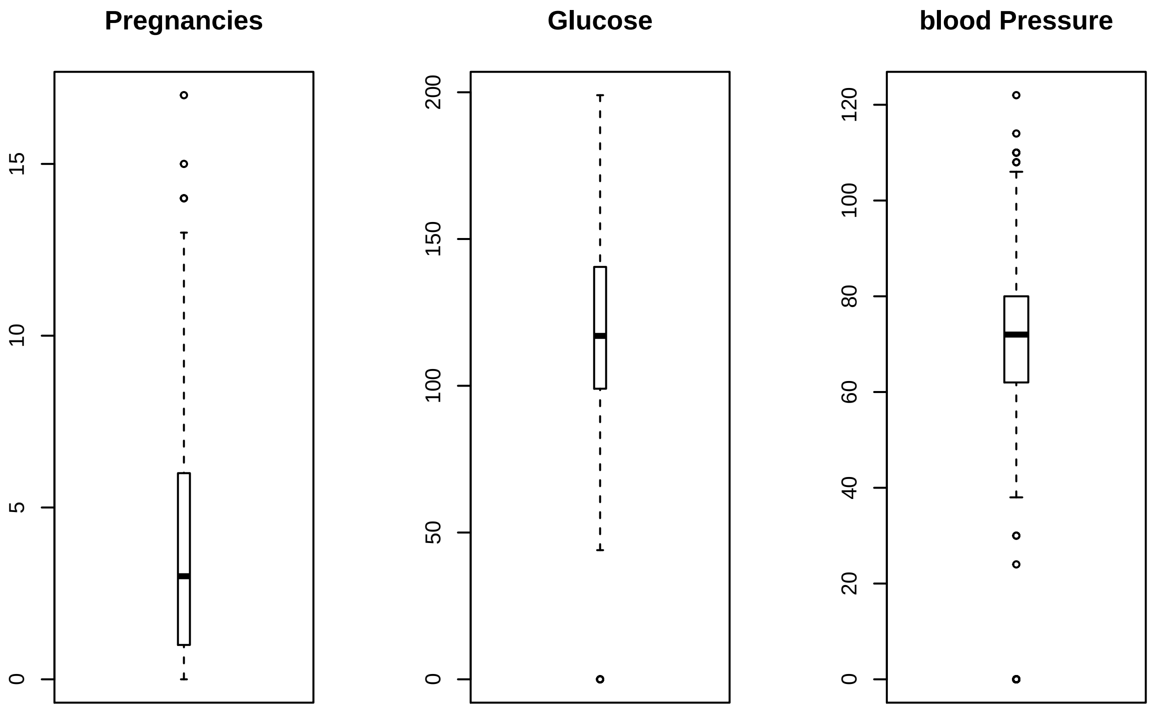
This shows that all the data has numeric variables.

Lets rename some of the variables so it would be visible clearly in plots.

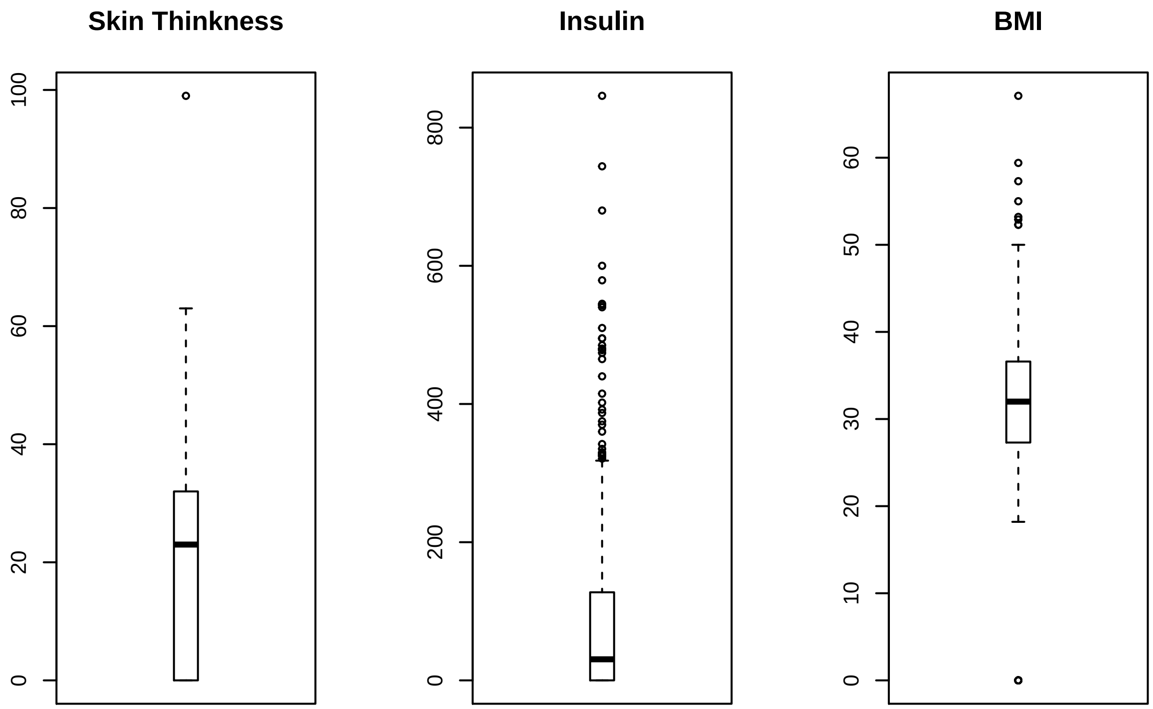
```
colnames(diab.data)[1] <- "Pregn"
colnames(diab.data)[3] <- "BldPrss"
colnames(diab.data)[4] <- "SknThknss"
colnames(diab.data)[7] <- "DPF"
colnames(diab.data)[9] <- "diabetes"
diab.data$diabetes <- as.factor(diab.data$diabetes)
levels(diab.data$diabetes) <- c ( "No", "Yes" )
```

Overall distribution of data.

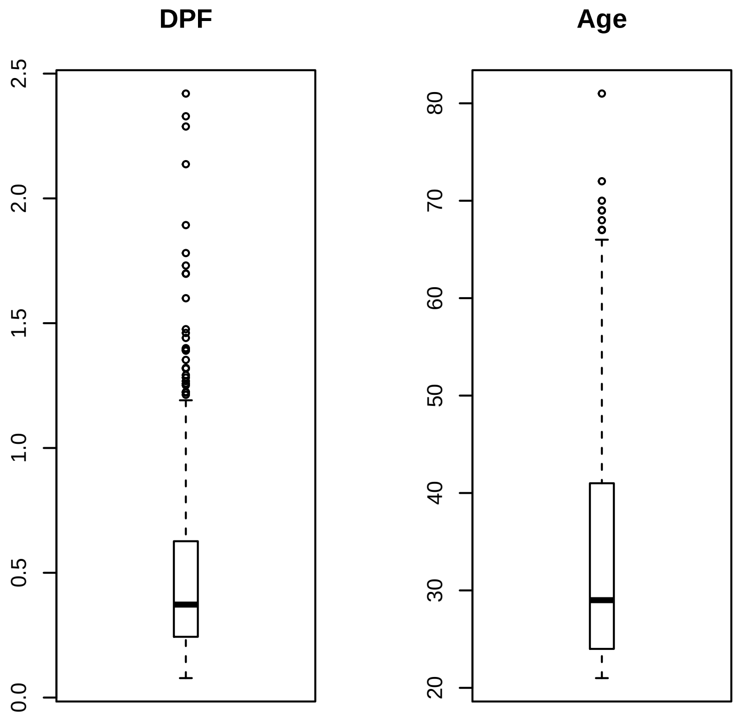
```
par ( mfrow = c ( 1 , 3 ) )
boxplot(diab.data$Pregn, main = "Pregnancies", boxwex = 0.1)
boxplot(diab.data$Glucose, main = "Glucose", boxwex = 0.1)
boxplot(diab.data$BldPrss, main = "blood Pressure", boxwex = 0.2)
```



```
boxplot(diab.data$SknThknss, main = "Skin Thinkness", boxwex = 0.2)
boxplot(diab.data$Insulin, main = "Insulin", boxwex = 0.2)
boxplot(diab.data$BMI, main = "BMI", boxwex = 0.2)
```

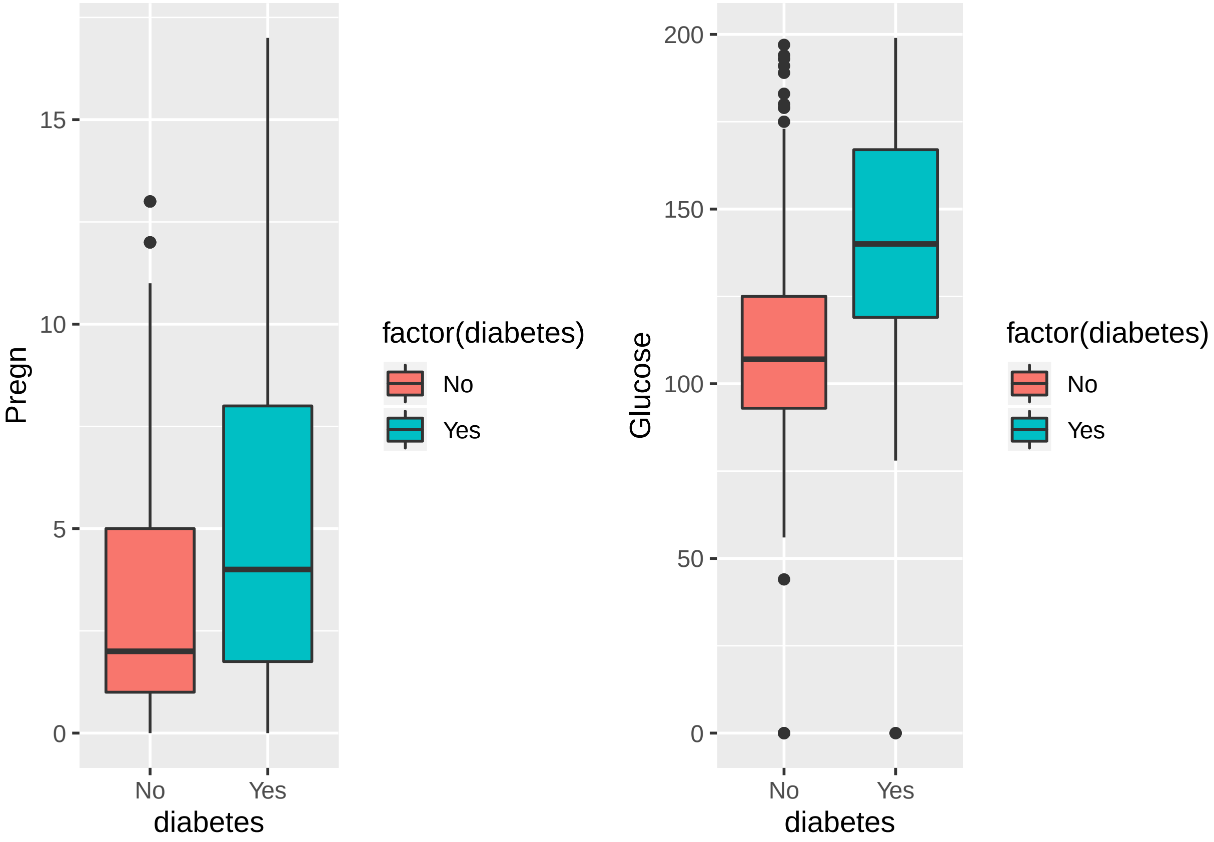


```r
boxplot(diab.data$DPF, main = "DPF", boxwex = 0.2)
boxplot(diab.data$Age, main = "Age", boxwex = 0.2)
```

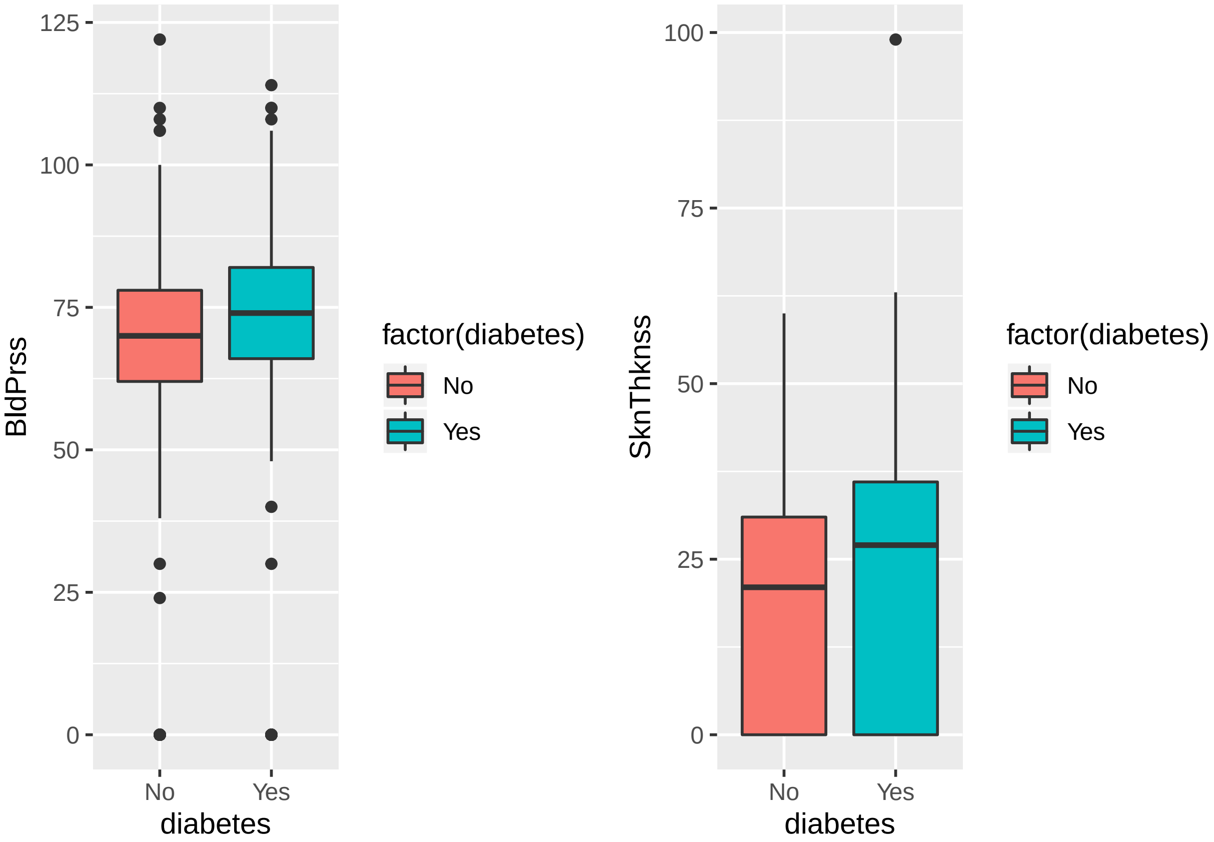


Distribution of the data respective to diabetes outcomes.

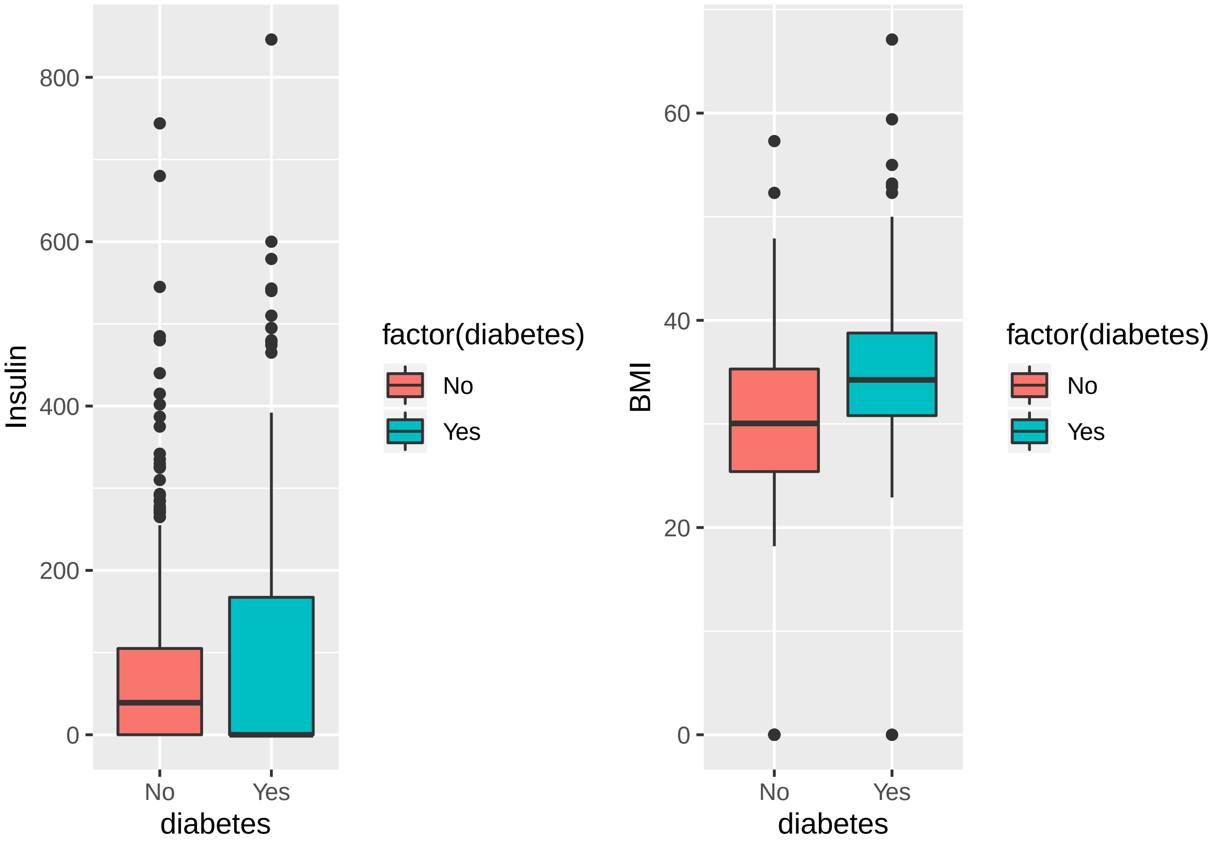
```r
# check each variable distribution respective to diabetes outcome
preg <- ggplot( data = diab.data, aes_string ( x = "diabetes", y = "Pregn" ) ) +
    geom_boxplot ( aes ( fill = factor ( diabetes ) ) )
gluco <- ggplot( data = diab.data, aes_string ( x = "diabetes", y = "Glucose" ) ) +
    geom_boxplot ( aes ( fill = factor ( diabetes ) ) )
bldPre <- ggplot( data = diab.data, aes_string ( x = "diabetes", y = "BldPrss" ) ) +
    geom_boxplot ( aes ( fill = factor ( diabetes ) ) )
sknThk <- ggplot( data = diab.data, aes_string ( x = "diabetes", y = "SknThknss" ) ) +
    geom_boxplot ( aes ( fill = factor ( diabetes ) ) )
insu <- ggplot( data = diab.data, aes_string ( x = "diabetes", y = "Insulin" ) ) +
    geom_boxplot ( aes ( fill = factor ( diabetes ) ) )
bmi <- ggplot( data = diab.data, aes_string ( x = "diabetes", y = "BMI" ) ) +
    geom_boxplot ( aes ( fill = factor ( diabetes ) ) )
dpf <- ggplot( data = diab.data, aes_string ( x = "diabetes", y = "DPF" ) ) +
    geom_boxplot ( aes ( fill = factor ( diabetes ) ) )
age <- ggplot( data = diab.data, aes_string ( x = "diabetes", y = "Age" ) ) +
    geom_boxplot ( aes ( fill = factor ( diabetes ) ) )

grid.arrange(preg, gluco, ncol = 2)
```



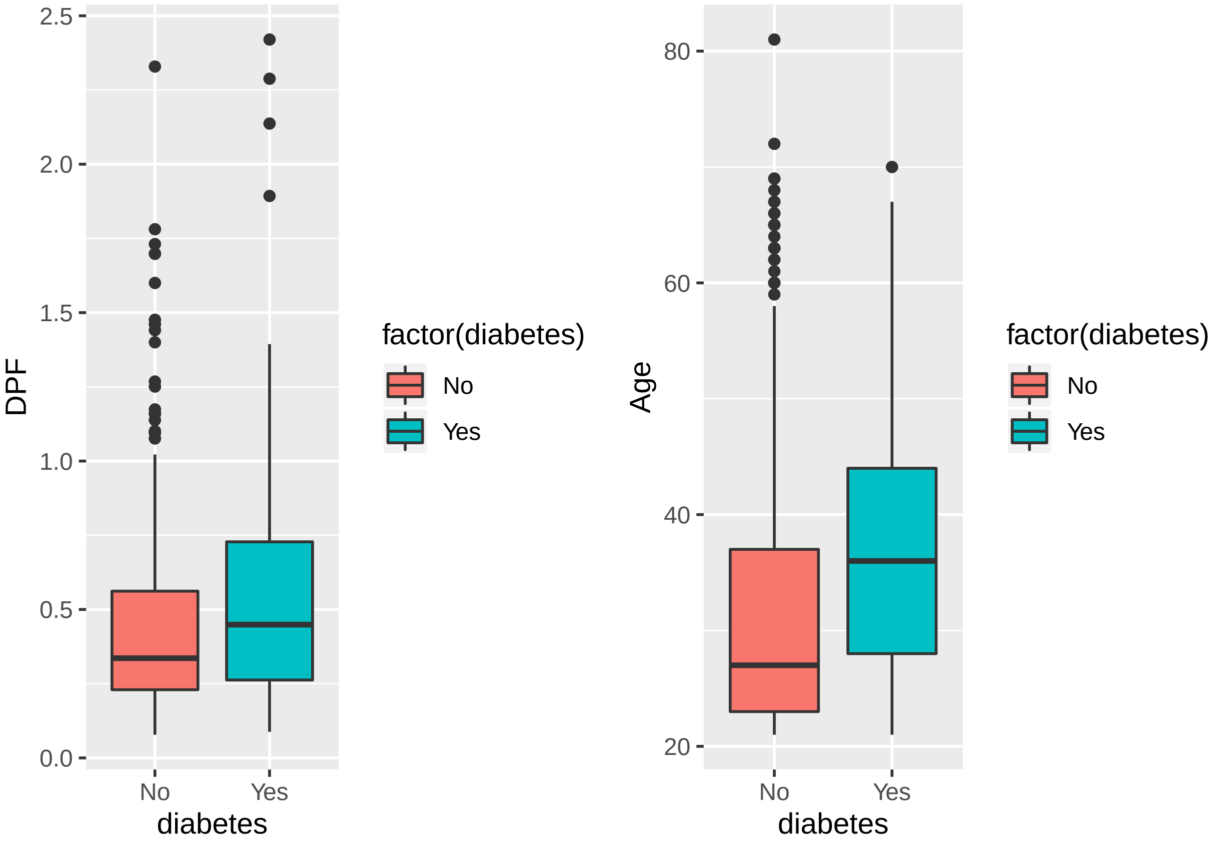
```
grid.arrange( bldPre ,sknThk, ncol = 2)
```



```
grid.arrange( insu, bmi, ncol = 2)
```



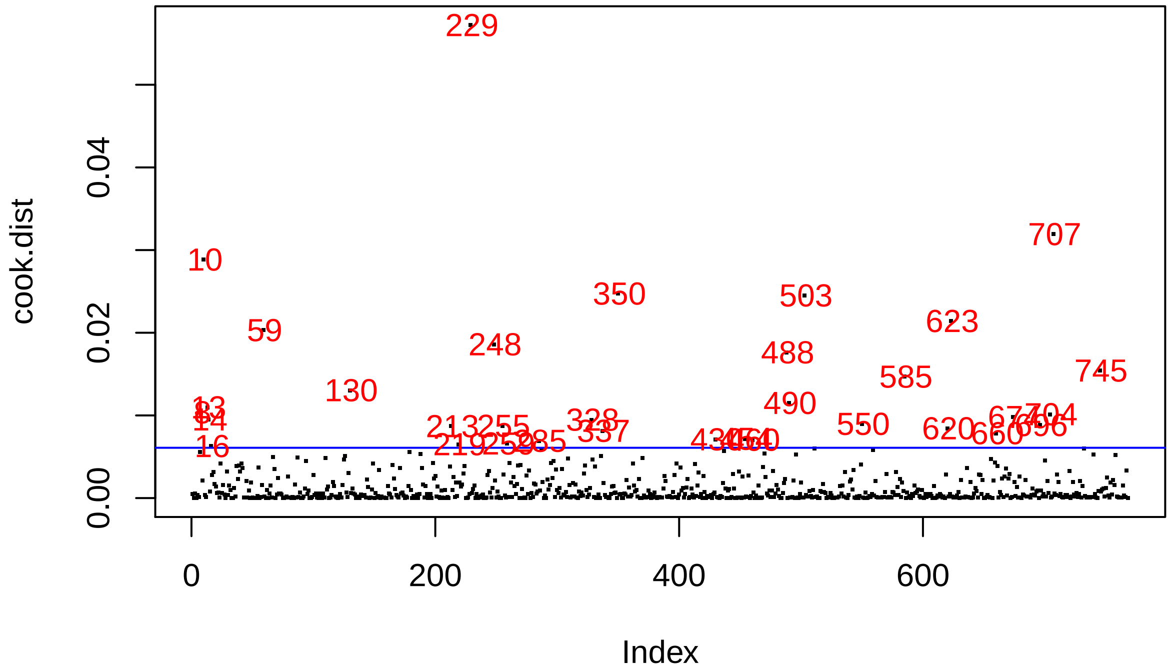
```
grid.arrange( dpf, age, ncol = 2)
```



Detection of outliers

Rather finding individual variable's outlier, I would prefer to collectively consider features that matter in model building. For this I can apply cook's distance method where I would build a linear model on our data to detect the outliers. Generally variables that have cook's distance greater than 4 times a mean will be considered having an outliers and would affect our model performance.

```
set.seed(123)
# build a linear model on our data
out.detect.model <-  glm(diabetes ~ . , data = diab.data, family = "binomial")

# get the cook's distance
cook.dist <- cooks.distance(out.detect.model)

# plot the row numbers for which a variable has outliers.
plot(cook.dist , pch = ".", cex = 2)
abline ( h = 4 * mean ( cook.dist, na.rm = T ), col = "blue" )
text ( x = 1 : length ( cook.dist ) + 1 , y = cook.dist,
       labels = ifelse ( cook.dist > 4 * mean ( cook.dist, na.rm = T),
                         names ( cook.dist ), "" ), col = "red" )
```



```
# get a list of number of rows
outl.row.numbers <- as.numeric ( names ( cook.dist ) [ ( cook.dist > 4 *
                                                    mean ( cook.dist,
                                                           na.rm = T))])
length(outl.row.numbers)
```

```
## [1] 33
```

```
diab.data <- diab.data[-outl.row.numbers,]
```

```
dim(diab.data)
```

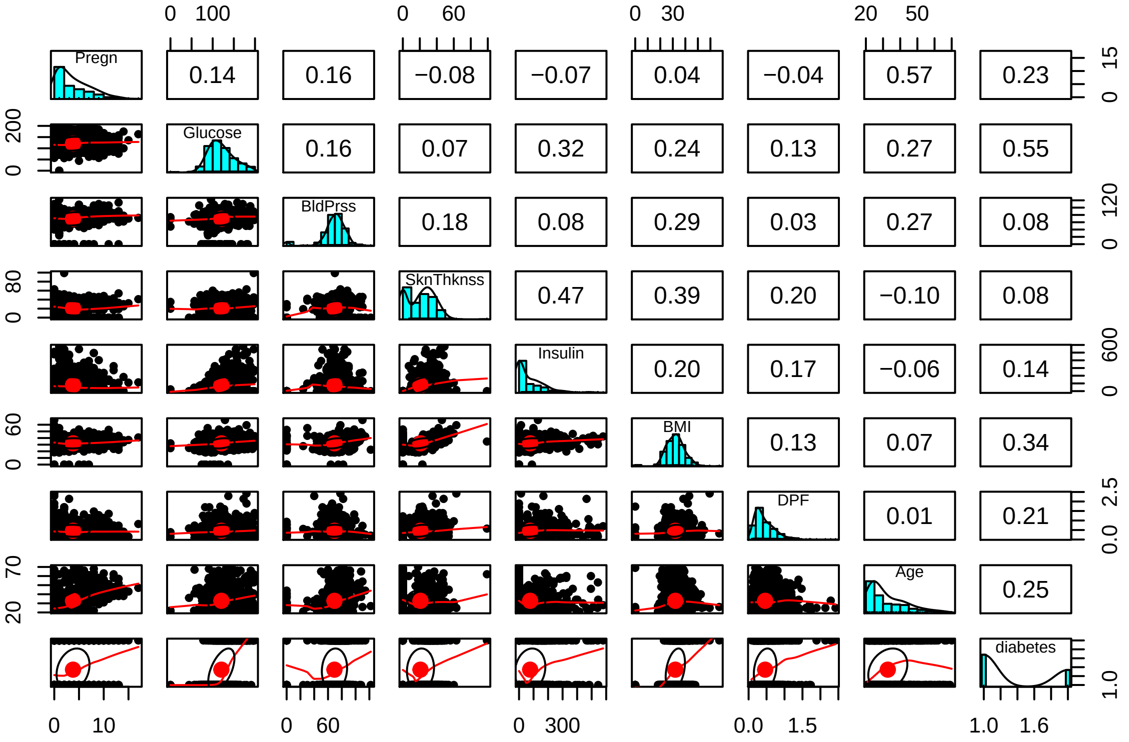
```
## [1] 735   9
```

This shows that there are 33 rows which has outliers which would affect the performance of our model. So, modified our data with outliers replaced. However, there may be some outlier present in an individual variables. I would proceed with this.

correlation/collinearity analysis

To check correlation of all varibles to each other in one plot, I can use pairs.panel function from psych package.

```
pairs.panels(diab.data)
```



This plot shows that non of the variables has very strong correlation to each other. Only glucose has a little bit correlation with diabetes as compared to other variables which makes sense that higher glucose level is considered as diabetic condition. Apart from that preganancy and age has correlation while skin thickness shows correlation with insulin level and BMI. However, none of these correlations are very strong. Rest all varibles do not show any strong correlation with each other.

Data Cleaning and Shaping

data imputation

For this I need to get the summary of the data.

```
summary(diab.data)
```

```
##      Pregn          Glucose          BldPrss          SknThknss
##  Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00
##  1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 64.00   1st Qu.: 0.00
##  Median : 3.000   Median :116.0   Median : 72.00   Median :23.00
##  Mean   : 3.795   Mean   :120.5   Mean   : 69.36   Mean   :20.76
##  3rd Qu.: 6.000   3rd Qu.:139.5   3rd Qu.: 80.00   3rd Qu.:32.00
##  Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00
##     Insulin            BMI             DPF              Age
##  Min.   :  0.00   Min.   : 0.00   Min.   :0.0780   Min.   :21.00
##  1st Qu.:  0.00   1st Qu.:27.30   1st Qu.:0.2420   1st Qu.:24.00
##  Median : 36.00   Median :32.00   Median :0.3660   Median :29.00
##  Mean   : 76.66   Mean   :31.99   Mean   :0.4609   Mean   :32.77
##  3rd Qu.:126.00   3rd Qu.:36.50   3rd Qu.:0.6060   3rd Qu.:40.00
##  Max.   :579.00   Max.   :67.10   Max.   :2.4200   Max.   :70.00
##  diabetes
##  No :482
##  Yes:253
```

```
##
##
##
##
```

There are no missing values in this data set as such, however 0 value does not make any sense in glucose, blood pressure, skin thikness, insulin and BMI for alive person. 0 value in preganancy is possible as it denotes no preganancy.

So, 0s in these variables are considered as a missing values.

Lets check how many 0s are in this variables.

```
cat("Glucose: ", sum(diab.data$Glucose == 0),
    "(", round((sum(diab.data$Glucose == 0)/length(diab.data$Glucose))*100, 2),"%)\n")
```

```
## Glucose:  3 ( 0.41 %)
```

```
cat("Blood Pressure: ", sum(diab.data$BldPrss == 0),
    "(", round((sum(diab.data$BldPrss == 0)/length(diab.data$BldPrss))*100, 2),"%)\n")
```

```
## Blood Pressure:  28 ( 3.81 %)
```

```
cat("Skin Thinkness: ", sum(diab.data$SknThknss == 0),
    "(", round((sum(diab.data$SknThknss == 0)/length(diab.data$SknThknss))*100, 2),"%)\n")
```

```
## Skin Thinkness:  212 ( 28.84 %)
```

```
cat("Insulin: ", sum(diab.data$Insulin == 0),
    "(", round((sum(diab.data$Insulin == 0)/length(diab.data$Insulin))*100, 2),"%)\n")
```

```
## Insulin:  354 ( 48.16 %)
```

```
cat("BMI: ", sum(diab.data$BMI == 0),
    "(", round((sum(diab.data$BMI == 0)/length(diab.data$BMI))*100, 2),"%)\n")
```

```
## BMI:  9 ( 1.22 %)
```

```
# percentage of missing values
```

So the missing values and its percentage in each variable is as follow.

Glucose: 3 (0.41%)

Blood Pressure: 28 (3.81%)

Skin Thinkness: 212 (28.84%)

Insulin: 354 (48.16%)

BMI: 9 (1.22%)

I would not delete the missing value rows as removing these rows would affect valuable information in other variables like glucose level, blood pressure, BMI and Diabetes Pedigree Function. As the number of missing values are very high I would impute these values using multivariate imputation by chained equations (mice package). However, I would do impite missing values after dividing data into training and test set.

I also have noticed that there is one an unusual observation in skin thinkness. Lets check this and impute with its median value.

```
# count the unusual 99 value in skin thikness
sum(diab.data[, 4] == 99)
```

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

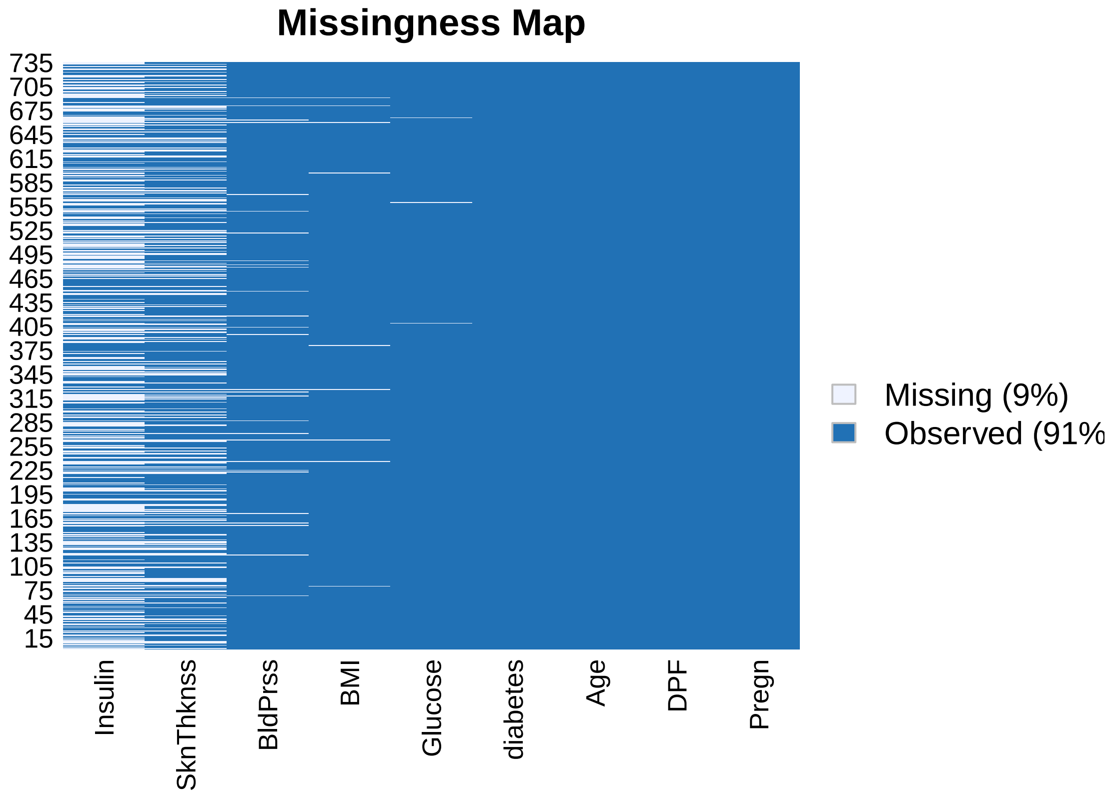
```
# impute this with its median value
diab.data[, 4][diab.data[,4] == 99] <- 23
# check its imputation
sum(diab.data[, 4] == 99)
```

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

Lets replace all 0s with NA values which is required by our imputation algorithm.

We can see missing values in plot.

```
missmap(diab.data)
```



creation of training and validation subsets

I think it is good to split the data first before imputaion of all 0 values. I would use 75% data for training the model while 25% for testing.

We know that 500 observations are NO and 268 observations are YES for diabetes varible which is 65.1% and 34.9% respectively. I want to split the data in the same proportion for diabetes variable.

```
set.seed(123)
# split the data with 75/25 proportion
split.data <- createDataPartition ( diab.data$diabetes, p = 0.75, list = FALSE )
# assign to training and test dataset
train.data <- diab.data[split.data,]
test.data <- diab.data[-split.data,]
# check its proportion for diabetes variable.
round(prop.table(table(diab.data$diabetes)) * 100, digits = 2)
```

```
##
##    No   Yes
## 65.58 34.42
```

```
round(prop.table(table(train.data$diabetes)) * 100, digits = 2)
```

```
##
##    No   Yes
## 65.58 34.42
```

```
round(prop.table(table(test.data$diabetes)) * 100, digits = 2)
```

```
##
##    No   Yes
## 65.57 34.43
```

This shows that we have the same proportion of diabetes factors in training and test dataset as in our original dataset.

Impute data on training dataset

```
set.seed(123)
# apply multivariate imputation by chained equation with
# random forest method on these 2 varibles with NA values
imputed.values.train <- mice ( train.data[ , c ("SknThknss", "Insulin")], method = 'rf')
```

```
##
##  iter imp variable
##   1   1  SknThknss  Insulin
##   1   2  SknThknss  Insulin
##   1   3  SknThknss  Insulin
##   1   4  SknThknss  Insulin
##   1   5  SknThknss  Insulin
##   2   1  SknThknss  Insulin
##   2   2  SknThknss  Insulin
##   2   3  SknThknss  Insulin
##   2   4  SknThknss  Insulin
##   2   5  SknThknss  Insulin
##   3   1  SknThknss  Insulin
##   3   2  SknThknss  Insulin
##   3   3  SknThknss  Insulin
##   3   4  SknThknss  Insulin
##   3   5  SknThknss  Insulin
##   4   1  SknThknss  Insulin
##   4   2  SknThknss  Insulin
##   4   3  SknThknss  Insulin
##   4   4  SknThknss  Insulin
##   4   5  SknThknss  Insulin
##   5   1  SknThknss  Insulin
##   5   2  SknThknss  Insulin
##   5   3  SknThknss  Insulin
##   5   4  SknThknss  Insulin
##   5   5  SknThknss  Insulin
```

```
# Extracts the completed data from a 'mids' object
extract.values.train <- mice::complete ( imputed.values.train )

# replace the NAs with imouted values

train.data$SknThknss <- extract.values.train$SknThknss
train.data$Insulin <- extract.values.train$Insulin
```

Lets impute glucose with its mean and blood pressure and BMI with its median as number of missing values are not very high in these varibles.

```
train.data[, 2][is.na(train.data[,2])] <- 120
train.data[, 3][is.na(train.data[,3])] <- 72
train.data[, 6][is.na(train.data[,6])] <- 32
```

Impute data on test dataset

```
set.seed(123)
# apply multivariate imputation by chained equation with
# random forest method on these 2 varibles with NA values
imputed.values.test <- mice ( test.data[ , c ("SknThknss", "Insulin")], method = 'rf')
```

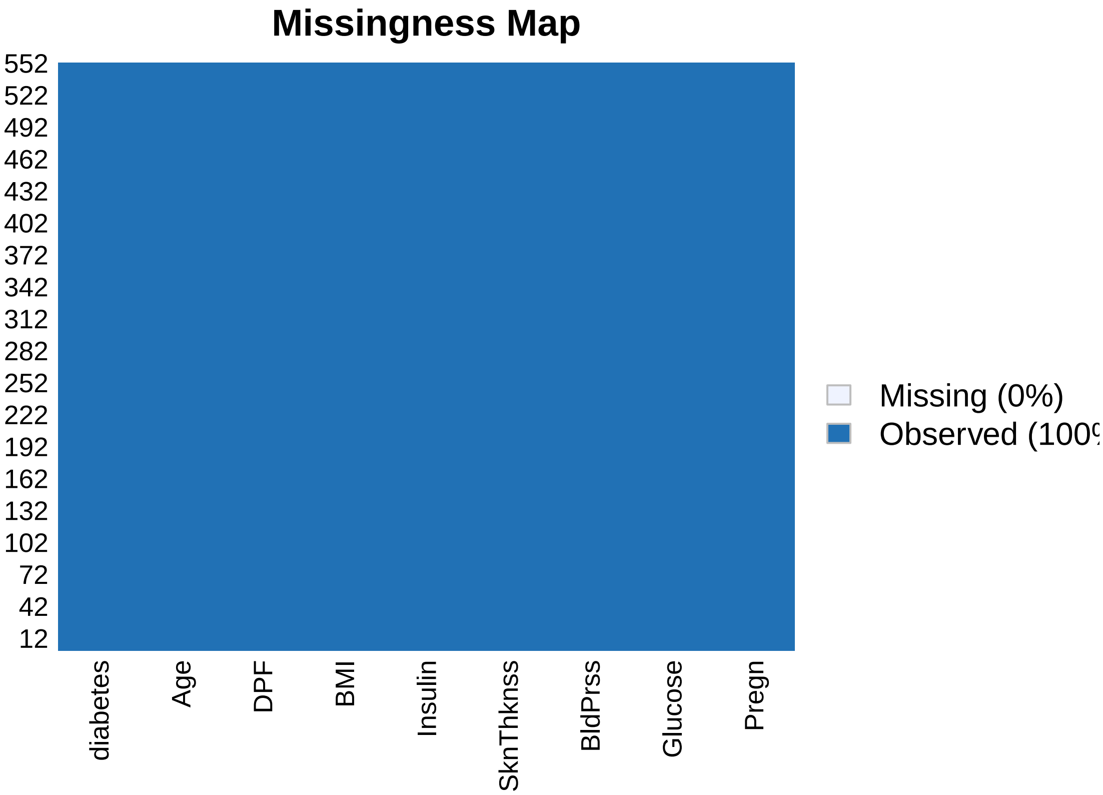
```
##
##  iter imp variable
##   1   1  SknThknss  Insulin
##   1   2  SknThknss  Insulin
##   1   3  SknThknss  Insulin
##   1   4  SknThknss  Insulin
##   1   5  SknThknss  Insulin
##   2   1  SknThknss  Insulin
##   2   2  SknThknss  Insulin
##   2   3  SknThknss  Insulin
##   2   4  SknThknss  Insulin
##   2   5  SknThknss  Insulin
##   3   1  SknThknss  Insulin
##   3   2  SknThknss  Insulin
##   3   3  SknThknss  Insulin
##   3   4  SknThknss  Insulin
##   3   5  SknThknss  Insulin
##   4   1  SknThknss  Insulin
##   4   2  SknThknss  Insulin
##   4   3  SknThknss  Insulin
##   4   4  SknThknss  Insulin
##   4   5  SknThknss  Insulin
##   5   1  SknThknss  Insulin
##   5   2  SknThknss  Insulin
##   5   3  SknThknss  Insulin
##   5   4  SknThknss  Insulin
##   5   5  SknThknss  Insulin
```

```
# Extracts the completed data from a 'mids' object
extract.values.test <- mice::complete ( imputed.values.test )

# replace the NAs with imouted values

test.data$SknThknss <- extract.values.test$SknThknss
test.data$Insulin <- extract.values.test$Insulin
```

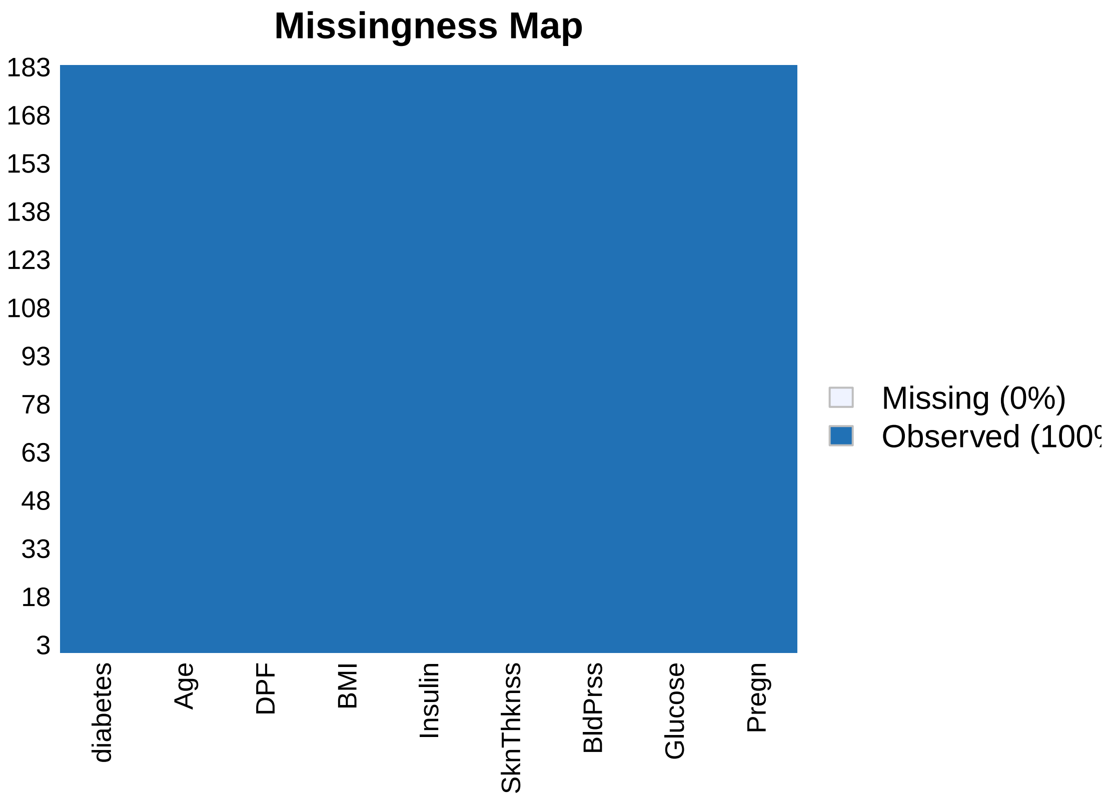
Lets impute glucose with its mean and blood pressure and BMI with its median as number of missing values are not very high in these varibles.

```
test.data[, 2][is.na(test.data[,2])] <- 120
test.data[, 3][is.na(test.data[,3])] <- 70
test.data[, 6][is.na(test.data[,6])] <- 32
```

```
missmap(train.data)
```



```
missmap(test.data)
```



This shows that we don't have any missing values in our train and test data set.

```
summary(train.data)
```

```
##      Pregn           Glucose          BldPrss          SknThknss
##  Min.   : 0.00   Min.   : 44.0   Min.   : 24.00   Min.   : 7.00
##  1st Qu.: 1.00   1st Qu.: 99.0   1st Qu.: 64.00   1st Qu.:21.75
##  Median : 3.00   Median :115.0   Median : 72.00   Median :30.00
##  Mean   : 3.81   Mean   :120.8   Mean   : 72.07   Mean   :29.44
##  3rd Qu.: 6.00   3rd Qu.:140.0   3rd Qu.: 80.00   3rd Qu.:37.00
##  Max.   :17.00   Max.   :199.0   Max.   :122.00   Max.   :63.00
##     Insulin            BMI             DPF              Age
##  Min.   : 14.0   Min.   :18.20   Min.   :0.0780   Min.   :21.00
##  1st Qu.: 76.0   1st Qu.:27.50   1st Qu.:0.2377   1st Qu.:24.00
##  Median :123.5   Median :32.00   Median :0.3505   Median :29.00
##  Mean   :146.6   Mean   :32.41   Mean   :0.4540   Mean   :33.12
##  3rd Qu.:185.8   3rd Qu.:36.50   3rd Qu.:0.5940   3rd Qu.:41.00
##  Max.   :545.0   Max.   :67.10   Max.   :2.4200   Max.   :69.00
##  diabetes
##  No :362
##  Yes:190
##
##
##
##
```

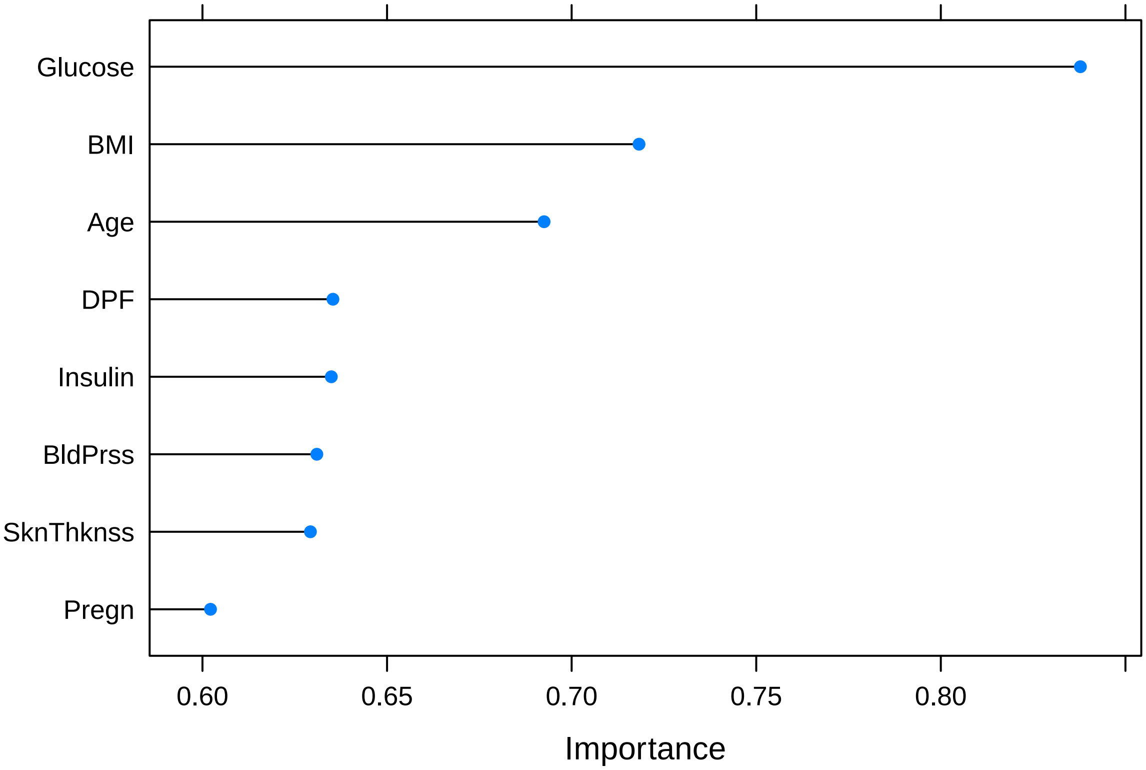
```
summary(test.data)
```

```
##      Pregn            Glucose          BldPrss          SknThknss
##  Min.   : 0.000   Min.   : 68.0   Min.   : 30.00   Min.   :10.00
##  1st Qu.: 1.000   1st Qu.:100.0   1st Qu.: 64.00   1st Qu.:22.00
##  Median : 3.000   Median :120.0   Median : 72.00   Median :29.00
##  Mean   : 3.749   Mean   :121.5   Mean   : 72.13   Mean   :28.98
##  3rd Qu.: 6.000   3rd Qu.:139.0   3rd Qu.: 80.00   3rd Qu.:35.00
##  Max.   :14.000   Max.   :197.0   Max.   :114.00   Max.   :60.00
##     Insulin            BMI             DPF                Age        diabetes
##  Min.   : 18.0   Min.   :18.40   Min.   :0.0840   Min.   :21.0   No :120
##  1st Qu.: 65.0   1st Qu.:27.50   1st Qu.:0.2570   1st Qu.:24.0   Yes: 63
##  Median :120.0   Median :32.00   Median :0.4020   Median :27.0
##  Mean   :143.4   Mean   :32.31   Mean   :0.4817   Mean   :31.7
##  3rd Qu.:175.0   3rd Qu.:36.35   3rd Qu.:0.6570   3rd Qu.:36.0
##  Max.   :579.0   Max.   :55.00   Max.   :1.6980   Max.   :70.0
```

```r
# define min-max normalization function
normalize.f <- function ( x ) {
  return ( ( x - min ( x ) ) / ( max ( x ) - min( x ) ) )
}
# normalize imputed training dataset
train.data[,1:8] <- apply(train.data[ , 1:8], 2, normalize.f)
# normalize imputed test dataset
test.data[ , 1:8] <- apply ( test.data[ , 1:8] , 2 , normalize.f )
```

The Learning Vector Quantization (LVQ) will be used in all examples because of its simplicity.

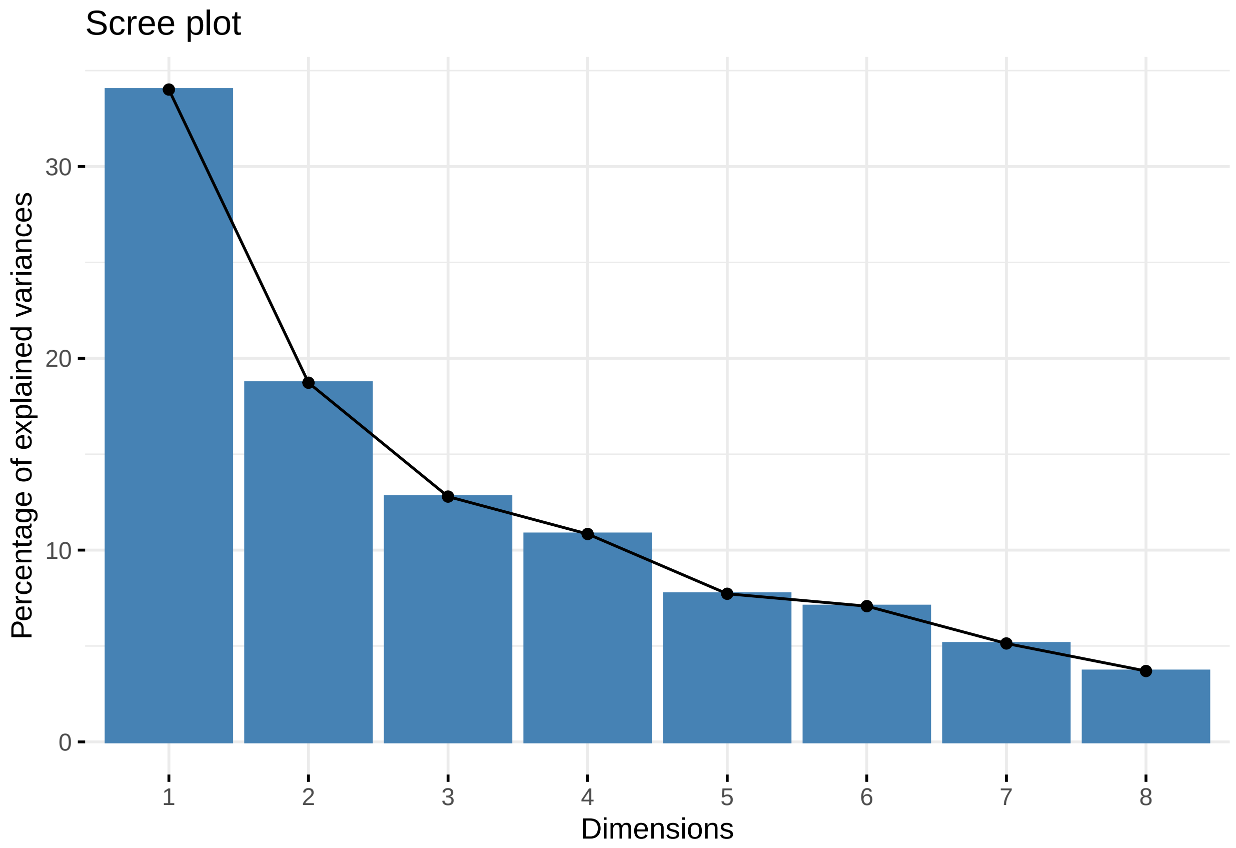
```r
set.seed(123)
# set the control structure for feature selection
control <- trainControl( method = "repeatedcv", number = 10, repeats = 3)
# train the model
model <- train ( diabetes ~ . , data = train.data,
                 method = "lvq",
                 preProcess = "scale",
                 trControl = control )
# estimate variable importance
importance <- varImp( model, scale = FALSE)
plot ( importance )
```



```r
set.seed(123)
# Lets check principle components
PCA.train <- prcomp(train.data[, 1:8])
summary(PCA.train)
```

```
## Importance of components:
##                           PC1    PC2    PC3    PC4     PC5     PC6     PC7
## Standard deviation     0.2950 0.2189 0.1809 0.1666 0.14059 0.13459 0.11461
## Proportion of Variance 0.3401 0.1873 0.1279 0.1084 0.07723 0.07079 0.05133
## Cumulative Proportion  0.3401 0.5274 0.6553 0.7637 0.84092 0.91171 0.96303
##                            PC8
## Standard deviation     0.09726
## Proportion of Variance 0.03697
## Cumulative Proportion  1.00000
```

```r
fviz_eig(PCA.train)
```



This shows that around 66% of the data explained by first three principle components.

Model construction and evaluation

Model 1: Logistic Regression

```
# Build the logistic regression model with all variables
logit.regres.model.1 <- glm(diabetes ~ . , data = train.data, family = binomial(link='logit'))
summary(logit.regres.model.1)
```

```
##
## Call:
## glm(formula = diabetes ~ ., family = binomial(link = "logit"),
##     data = train.data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3589  -0.6126  -0.2758   0.4546   2.4154
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -7.90921    0.76722 -10.309  < 2e-16 ***
## Pregn        2.31889    0.73155   3.170  0.00153 **
## Glucose      7.59621    0.80815   9.399  < 2e-16 ***
## BldPrss      0.13771    1.10114   0.125  0.90048
## SknThknss    0.64667    0.74929   0.863  0.38811
## Insulin      0.09855    0.67467   0.146  0.88386
## BMI          5.28419    1.10427   4.785 1.71e-06 ***
## DPF          4.16654    0.94579   4.405 1.06e-05 ***
## Age          0.27490    0.58863   0.467  0.64049
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 710.74  on 551  degrees of freedom
## Residual deviance: 432.17  on 543  degrees of freedom
## AIC: 450.17
##
## Number of Fisher Scoring iterations: 5
```

SknThknss has highest p-value, so drop it and build model on rest of the variables.

```
logit.regres.model.2 <- glm(diabetes ~  . -SknThknss, data = train.data, family = binomial(link='logit'
summary(logit.regres.model.2)
```

```
##
## Call:
## glm(formula = diabetes ~ . - SknThknss, family = binomial(link = "logit"),
##     data = train.data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3446  -0.6121  -0.2827   0.4597   2.3927
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -7.7649     0.7428 -10.453  < 2e-16 ***
## Pregn         2.3661     0.7322   3.232  0.00123 **
## Glucose       7.5594     0.8066   9.372  < 2e-16 ***
## BldPrss       0.1871     1.1001   0.170  0.86493
## Insulin       0.1654     0.6681   0.248  0.80442
## BMI           5.6168     1.0403   5.399 6.69e-08 ***
## DPF           4.1118     0.9411   4.369 1.25e-05 ***
## Age           0.2791     0.5884   0.474  0.63520
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 710.74  on 551  degrees of freedom
## Residual deviance: 432.92  on 544  degrees of freedom
## AIC: 448.92
##
## Number of Fisher Scoring iterations: 5
```

BldPrss has highest p-value, so drop it and build model on rest of the variables.

```
logit.regres.model.3 <- glm(diabetes ~  . -SknThknss -BldPrss, data = train.data, family = binomial(lin
summary(logit.regres.model.3)
```

```
##
## Call:
## glm(formula = diabetes ~ . - SknThknss - BldPrss, family = binomial(link = "logit"),
##     data = train.data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3460  -0.6139  -0.2851   0.4601   2.3879
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -7.7003     0.6369 -12.091  < 2e-16 ***
## Pregn         2.3702     0.7318   3.239   0.0012 **
## Glucose       7.5719     0.8036   9.422  < 2e-16 ***
## Insulin       0.1603     0.6674   0.240   0.8102
## BMI           5.6723     0.9883   5.739 9.51e-09 ***
## DPF           4.1032     0.9395   4.367 1.26e-05 ***
## Age           0.3046     0.5691   0.535   0.5925
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 710.74  on 551  degrees of freedom
## Residual deviance: 432.94  on 545  degrees of freedom
## AIC: 446.94
##
## Number of Fisher Scoring iterations: 5
```

Age has highest p-value, so drop it and build model on rest of the variables.

```
logit.regres.model.4 <- glm(diabetes ~  . -SknThknss -BldPrss -Age , data = train.data, family = binomi
summary(logit.regres.model.4)
```

```
##
## Call:
## glm(formula = diabetes ~ . - SknThknss - BldPrss - Age, family = binomial(link = "logit"),
##     data = train.data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3794  -0.6228  -0.2882   0.4662   2.3849
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -7.7005     0.6364 -12.099  < 2e-16 ***
## Pregn         2.5713     0.6308   4.076 4.57e-05 ***
## Glucose       7.6593     0.7892   9.705  < 2e-16 ***
## Insulin       0.1630     0.6695   0.243    0.808
## BMI           5.6511     0.9881   5.719 1.07e-08 ***
## DPF           4.1069     0.9390   4.373 1.22e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
```

```
##
##     Null deviance: 710.74  on 551  degrees of freedom
## Residual deviance: 433.23  on 546  degrees of freedom
## AIC: 445.23
##
## Number of Fisher Scoring iterations: 5
```

Insulin has highest p-value, so drop it and build model on rest of the variables.

After dropping 4 high p-value variables, model can be build on pregnancy, glucose, BMI and DPF.

```
logit.regres.model.5 <- glm(diabetes ~ Pregn + Glucose + BMI + DPF,
                            data = train.data,
                            family = binomial(link='logit'))
summary(logit.regres.model.5)
```

```
##
## Call:
## glm(formula = diabetes ~ Pregn + Glucose + BMI + DPF, family = binomial(link = "logit"),
##     data = train.data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3855  -0.6254  -0.2892   0.4687   2.3910
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -7.6828     0.6314 -12.167  < 2e-16 ***
## Pregn         2.5731     0.6308   4.079 4.52e-05 ***
## Glucose       7.6998     0.7727   9.965  < 2e-16 ***
## BMI           5.6673     0.9866   5.744 9.23e-09 ***
## DPF           4.0964     0.9377   4.368 1.25e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 710.74  on 551  degrees of freedom
## Residual deviance: 433.29  on 547  degrees of freedom
## AIC: 443.29
##
## Number of Fisher Scoring iterations: 5
```

Now we have regression model with all variable with p-value smaller than 0.05. Lets use this model to predict train data set. I am doing to check its accuracy on its own dataset. Then I will check its accuracy on test dataset and if accuracy on train dataset is higher than test then our model is overfitting. I would decide prediction power as 0.5 that if its probability is more that half then I would say the prediction is correct.

```
# Prediction for train dataset
train.pred <- predict(logit.regres.model.5, train.data, type = "response")

# Classification table - train dataset
traintable <- table(Predicted = train.pred >= 0.5, Actual = train.data$diabetes)
traintable
```

```
##          Actual
## Predicted  No Yes
##     FALSE 320  63
##     TRUE   42 127
```

```
# Accuracy of the model - train dataset
accuracy.train <- round(sum(diag(traintable))/sum(traintable),2)
cat("Accuracy is: ",accuracy.train)
```

```
## Accuracy is:  0.81
```

Model has 79% accuracy on training dataset. Lets check this on test dataset.

```
# Prediction for train dataset
test.pred <- predict(logit.regres.model.5, test.data, type = "response")

# Classification table - train dataset
test.table <- table(Predicted = test.pred >= 0.5, Actual = test.data$diabetes)
test.table
```

```
##          Actual
## Predicted No Yes
##     FALSE 95  22
##     TRUE  25  41
```

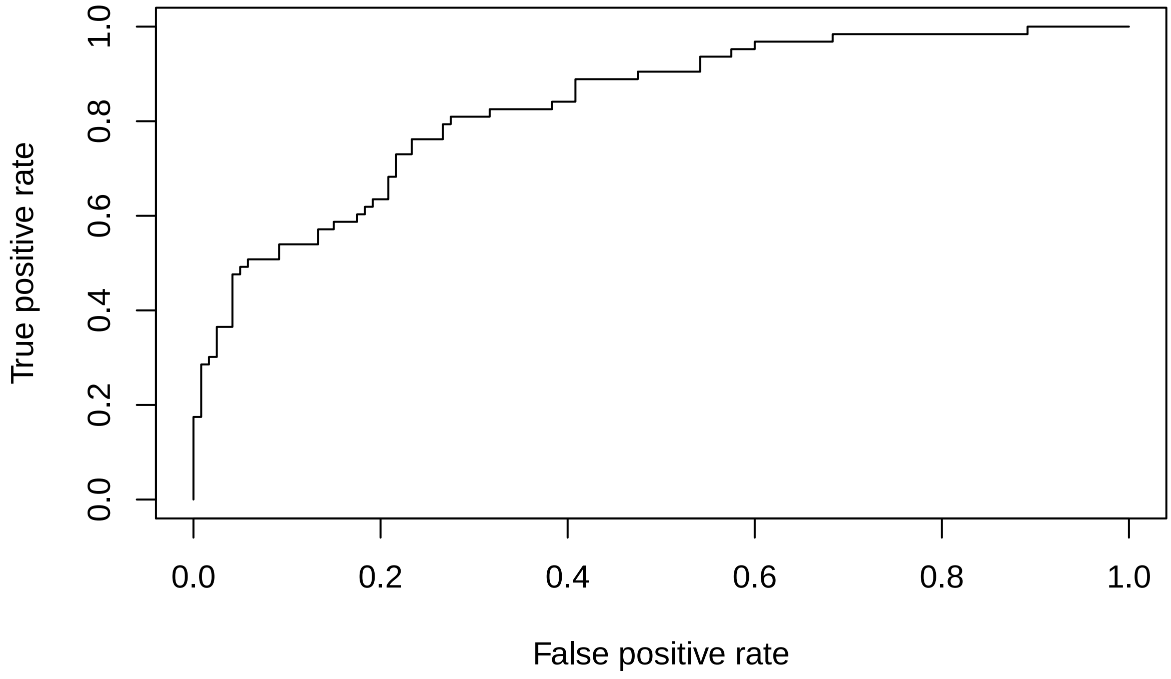
```
# Accuracy of the model - train dataset
accuracy.test <- round(sum(diag(test.table))/sum(test.table),2)
cat("Accuracy is: ",accuracy.test)
```

```
## Accuracy is:  0.74
```

Accuracy of our model on test dataset is 80% which is not that bad.

Lets evaluate predictor performance by using performance function with ROC curve with true positive rate (tpr) and false positive rate (fpr) options.

```
predict.test = prediction(test.pred, test.data$diabetes)
ROC.values = performance(predict.test, measure = "tpr", x.measure = "fpr")
plot(ROC.values)
```



ROC curve look good. Lets check its area under the curve percentage.

```r
auc <- performance(predict.test, measure = "auc")
auc <- auc@y.values[[1]]
round(auc, digits = 2)
```

```
## [1] 0.83
```

Area under the curve is 87%. We can say that model performs good. Lets do 10 fold cross-validation of this model.

```r
set.seed(1233)
# set 10 folds for cross validation
ctrl.reg <- trainControl ( method = "repeatedcv", number = 10, repeats = 3)
cv.reg <- train( diabetes ~ . ,
                  data = train.data,
                  method = "glm",
                  trControl = ctrl.reg,
                  preProcess = c("center", "scale"),
                  tuneLength = 10)
cv.reg.pred <- predict(cv.reg, test.data)
confusionMatrix(cv.reg.pred, test.data$diabetes)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  95  22
##        Yes 25  41
##
##                Accuracy : 0.7432
##                  95% CI : (0.6735, 0.8048)
##     No Information Rate : 0.6557
##     P-Value [Acc > NIR] : 0.006969
##
##                   Kappa : 0.4375
##
##  Mcnemar's Test P-Value : 0.770493
##
##             Sensitivity : 0.7917
##             Specificity : 0.6508
##          Pos Pred Value : 0.8120
##          Neg Pred Value : 0.6212
##              Prevalence : 0.6557
##          Detection Rate : 0.5191
##    Detection Prevalence : 0.6393
##       Balanced Accuracy : 0.7212
##
##        'Positive' Class : No
##
```

I did 10 fold cross-validation for logistic regression and now accuracy is 79% which shows that it does not improves its performance.

Model 2: k-Nearest Neighbors

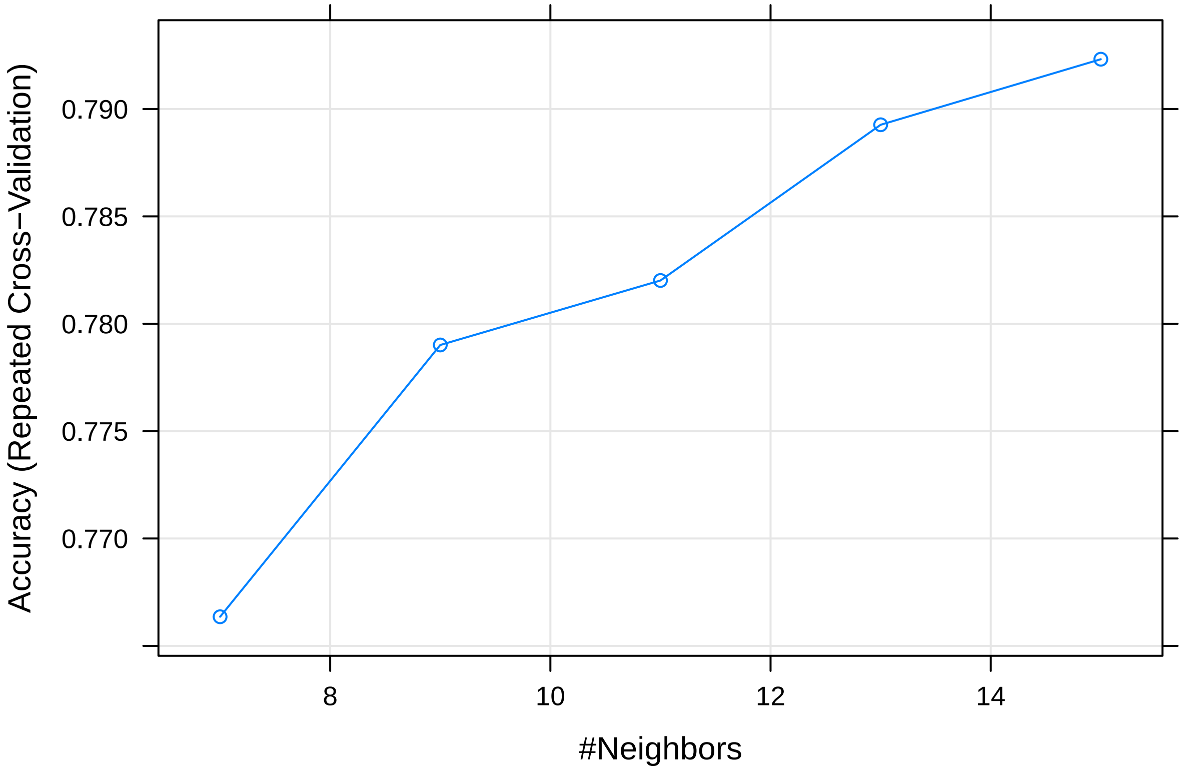
I choose to use this algorithm because this does not make any additional assumptions and simple to implement. The drawback of this algorithm become slower as data increase in size. However, I have relatively small data which has 9 columns and 768 observations. So this algorithm would work fine my data.

K-fold cross-validation to select the best peforming k-NN model.

```r
# cross-validation step
set.seed(123)
ctrl.knn <- trainControl ( method = "repeatedcv", number = 5, repeats = 3)
# define grid, select odd numbers to decide which point to choose
nn_grid <- expand.grid( k = c (7, 9, 11, 13, 15))
# build the best model with appropriate k number
best.knn.1 <- train( diabetes ~ . ,
                  data = train.data,
                  method = "knn",
                  trControl = ctrl.knn,
                  preProcess = c("center", "scale"),
                  tuneGrid = nn_grid)
best.knn.1
```

```
## k-Nearest Neighbors
##
## 552 samples
##   8 predictor
##   2 classes: 'No', 'Yes'
##
## Pre-processing: centered (8), scaled (8)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 442, 442, 441, 442, 441, 442, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    7  0.7663554  0.4534663
##    9  0.7790117  0.4819024
##   11  0.7820147  0.4863644
##   13  0.7892656  0.5001730
##   15  0.7923178  0.5034694
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 15.
```

```r
plot ( best.knn.1 )
```



A 5-fold cross validation shows that k = 13 for nearest neighbors would give us best accuracy model.

```
# train model on train data using optimal k, test with test data
model_knn <- knn ( train = train.data[, 1:8],
                  test = test.data[, 1:8],
                  # class (Outcome) labels
                  cl = train.data$diabetes,
                  k = 13 )
# A summary table will be generated with cell row, column and table
# proportions and marginal totals and proportions
CrossTable(x = test.data$diabetes, y = model_knn , prop.chisq = FALSE )
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  183
##
##
##                    | model_knn
## test.data$diabetes |        No |       Yes | Row Total |
## -------------------|-----------|-----------|-----------|
##                 No |       106 |        14 |       120 |
##                      |     0.883 |     0.117 |     0.656 |
##                      |     0.822 |     0.259 |           |
##                      |     0.579 |     0.077 |           |
## -------------------|-----------|-----------|-----------|
##                  Yes |        23 |        40 |        63 |
##                      |     0.365 |     0.635 |     0.344 |
##                      |     0.178 |     0.741 |           |
##                      |     0.126 |     0.219 |           |
## -------------------|-----------|-----------|-----------|
##         Column Total |       129 |        54 |       183 |
##                      |     0.705 |     0.295 |           |
## -------------------|-----------|-----------|-----------|
##
##
```

```
# get the confusion matrix
confusion_matrix <- table ( test.data$diabetes , model_knn )
# print the accuracy
cat("Test accuracy: ", round(sum(diag(confusion_matrix))/sum(confusion_matrix)*100),"%")
```

```
## Test accuracy:  80 %
```

This model shows 80% accuracy after k-fold validation.

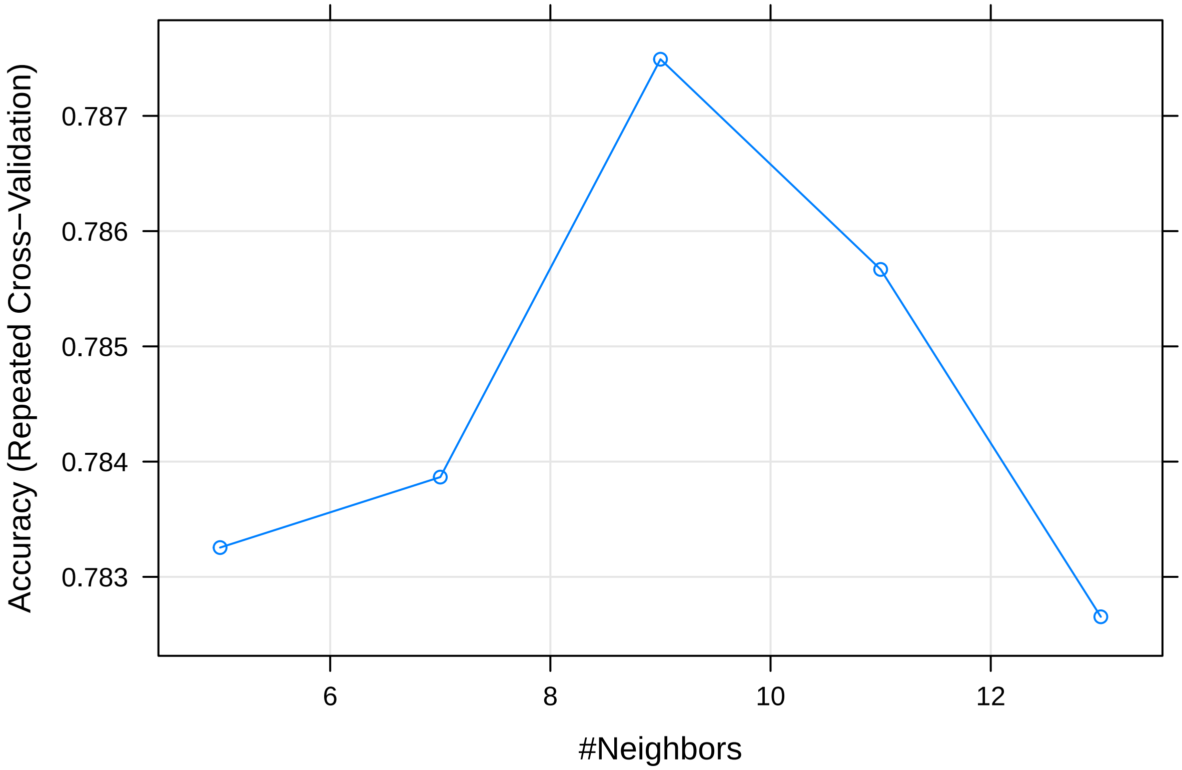
Now lets see how this model would perform on the selected variables. As per our logistic regression performance, variables pregnancy, glucose, BMI and DPF gave us more accuracy.

```
train.knn <- train.data[, c(1, 2, 6, 7, 9)]
test.knn <- test.data[, c(1, 2, 6, 7, 9)]
```

```
# cross-validation step
set.seed(123)
ctrl.knn <- trainControl ( method = "repeatedcv", number = 5, repeats = 3)
# define grid, select odd numbers to decide which point to choose
nn_grid <- expand.grid( k = c ( 5, 7, 9, 11, 13))
# build the best model with appropriate k number
best.knn.2 <- train( diabetes ~ . ,
                  data = train.data[, c(1, 2, 6, 7, 9)],
                  method = "knn",
                  trControl = ctrl.knn,
                  preProcess = c("center", "scale"),
                  tuneGrid = nn_grid)
best.knn.2
```

```
## k-Nearest Neighbors
##
## 552 samples
##   4 predictor
##   2 classes: 'No', 'Yes'
##
## Pre-processing: centered (4), scaled (4)
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 442, 442, 441, 442, 441, 442, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.7832542  0.5025282
##    7  0.7838657  0.4980360
##    9  0.7874911  0.5031468
##   11  0.7856675  0.4960103
##   13  0.7826536  0.4891926
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

```
plot ( best.knn.2 )
```



```
# train model on train data using optimal k, test with test data
model.knn.2 <- knn ( train = train.data[, c(1, 2, 6, 7)],
                    test = test.data[, c(1, 2, 6, 7)],
                    # class (Outcome) labels
                    cl = train.data$diabetes,
                    k = 11 )
# A summary table will be generated with cell row, column and table
# proportions and marginal totals and proportions
CrossTable(x = test.data$diabetes, y = model.knn.2 , prop.chisq = FALSE )
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |           N / Col Total |
## |         N / Table Total |
## |-------------------------|
##
```

```
##
## Total Observations in Table:  183
##
##
##                    | model.knn.2
## test.data$diabetes |        No |       Yes | Row Total |
## -------------------|-----------|-----------|-----------|
##                 No |       104 |        16 |       120 |
##                    |     0.867 |     0.133 |     0.656 |
##                    |     0.819 |     0.286 |           |
##                    |     0.568 |     0.087 |           |
## -------------------|-----------|-----------|-----------|
##                Yes |        23 |        40 |        63 |
##                    |     0.365 |     0.635 |     0.344 |
##                    |     0.181 |     0.714 |           |
##                    |     0.126 |     0.219 |           |
## -------------------|-----------|-----------|-----------|
##       Column Total |       127 |        56 |       183 |
##                    |     0.694 |     0.306 |           |
## -------------------|-----------|-----------|-----------|
##
##
```

```
# get the confusion matrix
confusion_matrix <- table ( test.data$diabetes , model.knn.2 )
# print the accuracy
cat("Test accuracy: ", round(sum(diag(confusion_matrix))/sum(confusion_matrix)*100),"%")
```

```
## Test accuracy:  79 %
```

It looks like selecting only 4 variable from logistic regression model does not improves model performance.

Model 3: Naive Bayes

I choose this algorithm because there is very little correlation between our predictors and Naïve Bayes algorithm makes assumption that all predictors are independent, so I think this model would work better for this data.

```
# build Naive Bayes model on all columns
naive.B.model.1 <- naiveBayes(diabetes ~ . , data = train.data)
# predict on test dataset
naive.pred.1 <- predict(naive.B.model.1, test.data)
# create confusion matrix to see its accuracy
confusionMatrix(naive.pred.1, test.data$diabetes)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  87  19
##        Yes 33  44
##
##                Accuracy : 0.7158
##                  95% CI : (0.6446, 0.7799)
##     No Information Rate : 0.6557
##     P-Value [Acc > NIR] : 0.04956
```

```
##
##                   Kappa : 0.4022
##
##  Mcnemar's Test P-Value : 0.07142
##
##             Sensitivity : 0.7250
##             Specificity : 0.6984
##          Pos Pred Value : 0.8208
##          Neg Pred Value : 0.5714
##              Prevalence : 0.6557
##          Detection Rate : 0.4754
##    Detection Prevalence : 0.5792
##       Balanced Accuracy : 0.7117
##
##        'Positive' Class : No
##
```

Our Naive Bayes model shows that it has 72% accuracy. Lets do 10 fold cross-validation to see if we can improve its performance.

```
naive.B.model.2 <- train ( train.data[ , -9 ],
                           train.data$diabetes,
                           'nb' ,
                           trControl = trainControl( method = 'cv',
                                                     number = 10 ))
```

```
naive.pred.2 <- predict(naive.B.model.2,newdata = test.data )
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 34
```

```
confusionMatrix(naive.pred.2, test.data$diabetes)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  87  19
##        Yes 33  44
##
##                Accuracy : 0.7158
##                  95% CI : (0.6446, 0.7799)
##     No Information Rate : 0.6557
##     P-Value [Acc > NIR] : 0.04956
##
##                   Kappa : 0.4022
##
##  Mcnemar's Test P-Value : 0.07142
##
##             Sensitivity : 0.7250
##             Specificity : 0.6984
##          Pos Pred Value : 0.8208
##          Neg Pred Value : 0.5714
##              Prevalence : 0.6557
##          Detection Rate : 0.4754
##    Detection Prevalence : 0.5792
##       Balanced Accuracy : 0.7117
##
##        'Positive' Class : No
##
```

10 fold cross-validation shows that our Naive Bayes model has improved its performance to 77% accuracy.

Model 4: Support Vector Machine

I choose this algorithm because our dependent variable is a binary and I think support vector machine algorithm would work well on this data.

```r
# build SVM model with radial kernel
svm.model.1 = svm(diabetes ~ . , data = train.data ,
                  kernel = "radial", type = "C-classification")

# Summary
summary(svm.model.1)
```

```
##
## Call:
## svm(formula = diabetes ~ ., data = train.data, kernel = "radial",
##     type = "C-classification")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  297
##
##  ( 146 151 )
##
##
## Number of Classes:  2
##
## Levels:
##  No Yes
```

```r
# predict for training data to check its accuracy
train.pred.1 = predict ( svm.model.1 , newdata = train.data )
# generate confusion matrix
confusionMatrix ( train.pred.1 , train.data$diabetes )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  341  53
##        Yes  21 137
##
##                Accuracy : 0.8659
##                  95% CI : (0.8346, 0.8932)
##     No Information Rate : 0.6558
##     P-Value [Acc > NIR] : < 2.2e-16
```

```
##
##                   Kappa : 0.6907
##
##  Mcnemar's Test P-Value : 0.0003137
##
##             Sensitivity : 0.9420
##             Specificity : 0.7211
##          Pos Pred Value : 0.8655
##          Neg Pred Value : 0.8671
##              Prevalence : 0.6558
##          Detection Rate : 0.6178
##    Detection Prevalence : 0.7138
##       Balanced Accuracy : 0.8315
##
##        'Positive' Class : No
##
```

On training dataset it shows that its accuracy is 82%. Lets see its accuracy on test dataset.

```
# predict on test data set
test.pred.1 <- predict ( svm.model.1, test.data )
confusionMatrix ( test.pred.1 , test.data$diabetes )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  97  24
##        Yes 23  39
##
##                Accuracy : 0.7432
##                  95% CI : (0.6735, 0.8048)
##     No Information Rate : 0.6557
##     P-Value [Acc > NIR] : 0.006969
##
##                   Kappa : 0.429
##
##  Mcnemar's Test P-Value : 1.000000
##
##             Sensitivity : 0.8083
##             Specificity : 0.6190
##          Pos Pred Value : 0.8017
##          Neg Pred Value : 0.6290
##              Prevalence : 0.6557
##          Detection Rate : 0.5301
##    Detection Prevalence : 0.6612
##       Balanced Accuracy : 0.7137
##
##        'Positive' Class : No
##
```

Accuracy has droped to 80% for the test dataset. This model has overfitting problem.

I need to tune this model for better accuracy with cost and gamma options.

```
tuned.svm <- tune.svm(diabetes ~ . ,
                      data = train.data,
                      cost = 10^c(-1, 0, 1),
                      gamma = 10^(-5:-1))
summary(tuned.svm)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  gamma cost
##   0.01   10
##
## - best performance: 0.1883766
##
## - Detailed performance results:
##    gamma cost     error dispersion
## 1  1e-05  0.1 0.3439935 0.05368822
## 2  1e-04  0.1 0.3439935 0.05368822
## 3  1e-03  0.1 0.3439935 0.05368822
## 4  1e-02  0.1 0.3294805 0.05481573
## 5  1e-01  0.1 0.2173052 0.04560736
## 6  1e-05  1.0 0.3439935 0.05368822
## 7  1e-04  1.0 0.3439935 0.05368822
## 8  1e-03  1.0 0.2895455 0.05569227
## 9  1e-02  1.0 0.1920130 0.04689213
## 10 1e-01  1.0 0.2155844 0.03680927
## 11 1e-05 10.0 0.3439935 0.05368822
## 12 1e-04 10.0 0.2841234 0.05902277
## 13 1e-03 10.0 0.1956169 0.04572975
## 14 1e-02 10.0 0.1883766 0.03819006
## 15 1e-01 10.0 0.2336688 0.05156743
```

```
# build SVM model with radial kernel
svm.model.2 = svm(diabetes ~ . , data = train.data ,
                  kernel = "radial", type = "C-classification",
                  gamma = 0.01,
                  cost = 10)
# predict on test data set
test.pred.2 <- predict ( svm.model.2, test.data )
confusionMatrix ( test.pred.2 , test.data$diabetes )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  97  25
##        Yes 23  38
##
##                Accuracy : 0.7377
##                  95% CI : (0.6677, 0.7998)
##     No Information Rate : 0.6557
##     P-Value [Acc > NIR] : 0.01085
##
```

```
##                   Kappa : 0.4146
##
##  Mcnemar's Test P-Value : 0.88523
##
##             Sensitivity : 0.8083
##             Specificity : 0.6032
##          Pos Pred Value : 0.7951
##          Neg Pred Value : 0.6230
##              Prevalence : 0.6557
##          Detection Rate : 0.5301
##    Detection Prevalence : 0.6667
##       Balanced Accuracy : 0.7058
##
##        'Positive' Class : No
##
```

Tunning showed that the accuracy is the same as previous 80% on the test dataset.

Model Comparison

Logistic regression: 80%

k-nearest Neighbours: 80%

Naive Bayes: 77%

Support Vector Machine: 80%

This shows that except Naive Bayes, all other models works good on our data.