STA545 Statistical Data Mining I Project

Estimation of Obesity levels
Priyanka Bhoite Akhilesh Nampalli Pradeepsurya Rajendran
Team 16

2022-12-16

Abstract

Obesity is a serious public health issue that affects a large portion of global population. It is contributed by various lifestyle factors, which if analyzed correctly can control obesity levels. In this project, the dataset containing data for the estimation of obesity levels in individuals from the countries of Mexico, Peru and Colombia, based on their eating habits and physical condition was analyzed. Different statistical analysis, visualization techniques and preprocessing were performed to derive insights from the data. Moreover, different supervised learning algorithms/methods such as Decision Tree, Support Vector Machine, Random Forest, and Bagging were fitted to the data and the relationship of predictors with the response was studied. The results show that Weight and BMI are the most influential predictors followed by Vegetable consumption and Age. Stacked models resulted in the lowest misclassification error rate of 2.5%.

Introduction:

Obesity is a major public health and economic problem of global significance. The problem statement includes understanding the dataset and building a predictive model that is capable of classifying someone into different health categories like obese or normal (health) range. The analysis and modeling of this dataset would give us the relationship between a person's eating habits, physical activity level , lifestyle and the body fat levels. More than any other time in the past few years, the current Covid 19 epidemic has demonstrated the value of leading a healthy lifestyle.

Data Description:

This dataset include data for the estimation of obesity levels in individuals based on their eating habits and physical condition. The data contains 17 attributes and 2111 records, the records are labeled with the class variable NObesity (Obesity Level), that allows classification of the data using the values of Insufficient Weight, Normal Weight, Overweight Level I, Overweight Level II, Obesity Type I, Obesity Type II and Obesity Type III.

Attributes related to eating habits: Frequent consumption of high caloric food (FAVC), Frequency of consumption of vegetables (FCVC), Number of main meals (NCP), Consumption of food between meals (CAEC), Consumption of water daily (CH20), and Consumption of alcohol(CALC).

The attributes related with the physical condition are: Calories consumption monitoring (SCC), Physical activity frequency (FAF), Time using technology devices (TUE), Transportation used (MTRANS), other variables obtained were: Gender, Age, Height and Weight

Materials & Methods:

-Importing data -Data Cleaning & Data Preparation -Exploratory Data Analysis -Data Encoding -Correlation of Variables -Decision Tree Modelling:with BMI -Bagging: with BMI -Random Forest:with BMI -SVM:with BMI -Decision tree: without BMI -SVM:without BMI -Bagging:without BMI -Random Forest-:without BMI

Importing Packages

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0      v purrr   0.3.5
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(easyreg)
library(tidyverse)
library(dplyr)
library(plyr)
```

```
## ------------------------------------------------------------------------------
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)
## ------------------------------------------------------------------------------
##
## Attaching package: 'plyr'
##
## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
##
## The following object is masked from 'package:purrr':
##
##     compact
```

```
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(PerformanceAnalytics)
```

```
## Loading required package: xts
## Loading required package: zoo
##
```

```
## Attaching package: 'zoo'
##
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
##
##
## Attaching package: 'xts'
##
## The following objects are masked from 'package:dplyr':
##
##     first, last
##
##
## Attaching package: 'PerformanceAnalytics'
##
## The following object is masked from 'package:graphics':
##
##     legend
```

```
library(rpart)
library(rpart.plot)
library(zoom)
library(pROC) # ROC curve
```

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(e1071) # SVM
```

```
##
## Attaching package: 'e1071'
##
## The following objects are masked from 'package:PerformanceAnalytics':
##
##     kurtosis, skewness
```

```
library(ipred)
library(randomForest)
```

```
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
## The following object is masked from 'package:ggplot2':
##
```

```
##     margin
```

```r
library(gbm)
```

```
## Loaded gbm 2.1.8.1
```

Importing data

```r
# importing dataset - csv file

data <- read.csv("ObesityData.csv")
#data
summary(data)
```

```
##     Gender                Age            Height          Weight      
##  Length:2111        Min.   :14.00   Min.   :1.450   Min.   : 39.00  
##  Class :character   1st Qu.:19.95   1st Qu.:1.630   1st Qu.: 65.47  
##  Mode  :character   Median :22.78   Median :1.700   Median : 83.00  
##                     Mean   :24.31   Mean   :1.702   Mean   : 86.59  
##                     3rd Qu.:26.00   3rd Qu.:1.768   3rd Qu.:107.43  
##                     Max.   :61.00   Max.   :1.980   Max.   :173.00  
##  family_history_with_overweight     FAVC                FCVC      
##  Length:2111                    Length:2111        Min.   :1.000  
##  Class :character               Class :character   1st Qu.:2.000  
##  Mode  :character               Mode  :character   Median :2.386  
##                                                    Mean   :2.419  
##                                                    3rd Qu.:3.000  
##                                                    Max.   :3.000  
##       NCP            CAEC              SMOKE               CH2O      
##  Min.   :1.000   Length:2111        Length:2111        Min.   :1.000  
##  1st Qu.:2.659   Class :character   Class :character   1st Qu.:1.585  
##  Median :3.000   Mode  :character   Mode  :character   Median :2.000  
##  Mean   :2.686                                         Mean   :2.008  
##  3rd Qu.:3.000                                         3rd Qu.:2.477  
##  Max.   :4.000                                         Max.   :3.000  
##      SCC                 FAF              TUE              CALC          
##  Length:2111        Min.   :0.0000   Min.   :0.0000   Length:2111       
##  Class :character   1st Qu.:0.1245   1st Qu.:0.0000   Class :character  
##  Mode  :character   Median :1.0000   Median :0.6253   Mode  :character  
##                     Mean   :1.0103   Mean   :0.6579                     
##                     3rd Qu.:1.6667   3rd Qu.:1.0000                     
##                     Max.   :3.0000   Max.   :2.0000                     
##     MTRANS           NObeyesdad       
##  Length:2111        Length:2111       
##  Class :character   Class :character  
##  Mode  :character   Mode  :character  
##                                       
##                                       
## 
```

Data Cleaning & Data Preparation

```r
# Checking for null values
any(is.na(data))
```

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

Dataset doesn't contain any null values.

```r
# Numeric predictors
num.cols <- c("Age", "Height", "Weight")

# Categorical Predictors
cat.cols <- c("Gender", "family_history_with_overweight", "FAVC", "FCVC",
              "NCP", "CAEC", "SMOKE", "CH2O", "SCC", "FAF", "TUE", "CALC",
              "MTRANS")
```

Rounding categorical variables like 'NCP','FCVC','CH20','FAF','TUT'

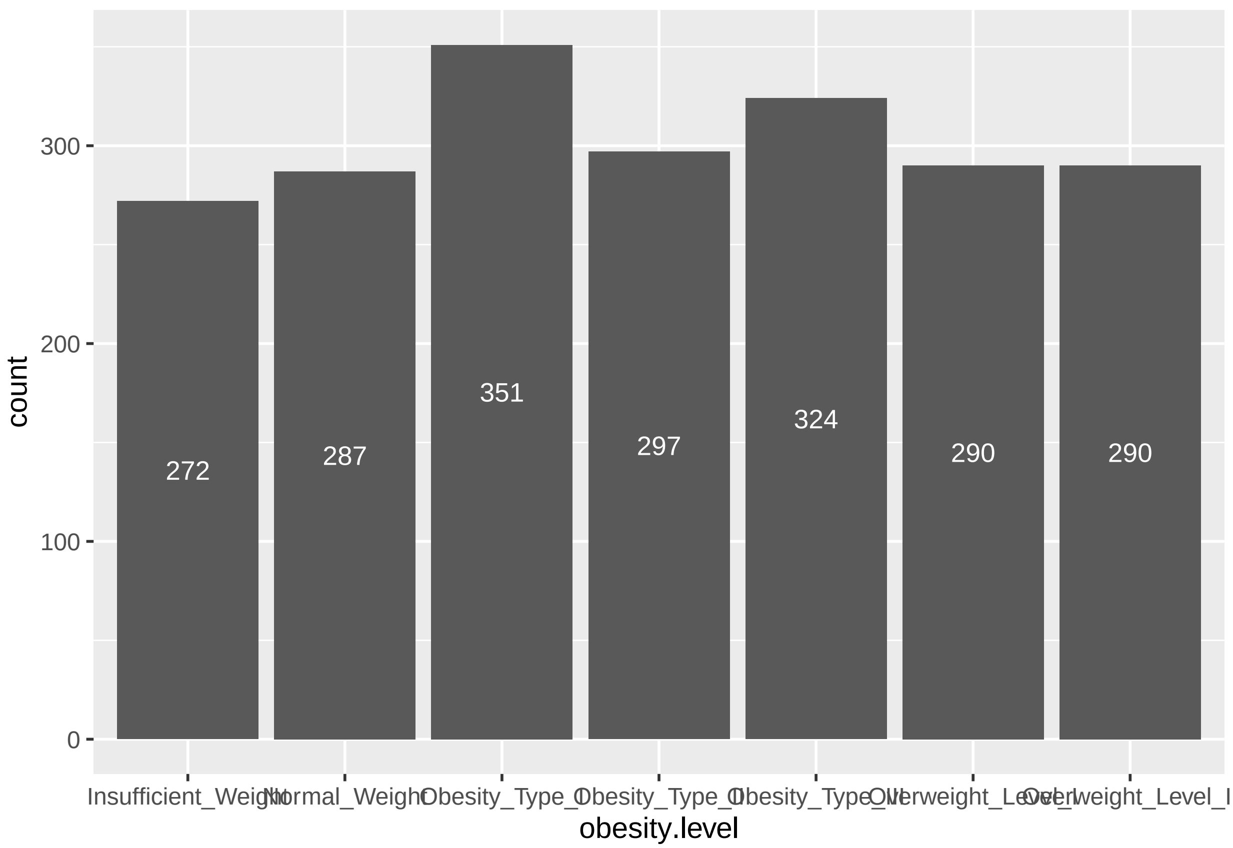
```r
data$FCVC <- round(data$FCVC) # Round off the column to integer
data$NCP <- round(data$NCP) # Round off the column to integer
data$CH2O <- round(data$CH2O) # Round off the column to integer
data$FAF <- round(data$FAF) # Round off the column to integer
data$TUE <- round(data$TUE) # Round off the column to integer

#unique(data[("FCVC")])
```

Exploratory Data Analysis

1- Analyzing the target variable

```r
# Analyzing the target variable
obesity.level <- data$NObeyesdad

ggplot(data = data, aes(x=obesity.level)) + geom_bar(stat='count') +
  stat_count(geom = "text", colour = "white", size = 3.5, aes(label = ..count..), position=position_stac
```

```
## Warning: The dot-dot notation (`..count..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(count)` instead.
```



The above bar graph and the distribution of the data shows that Obesity_Type_I is the most common among the respondents and Insufficient_Weight is the least common one

2- Data Summary

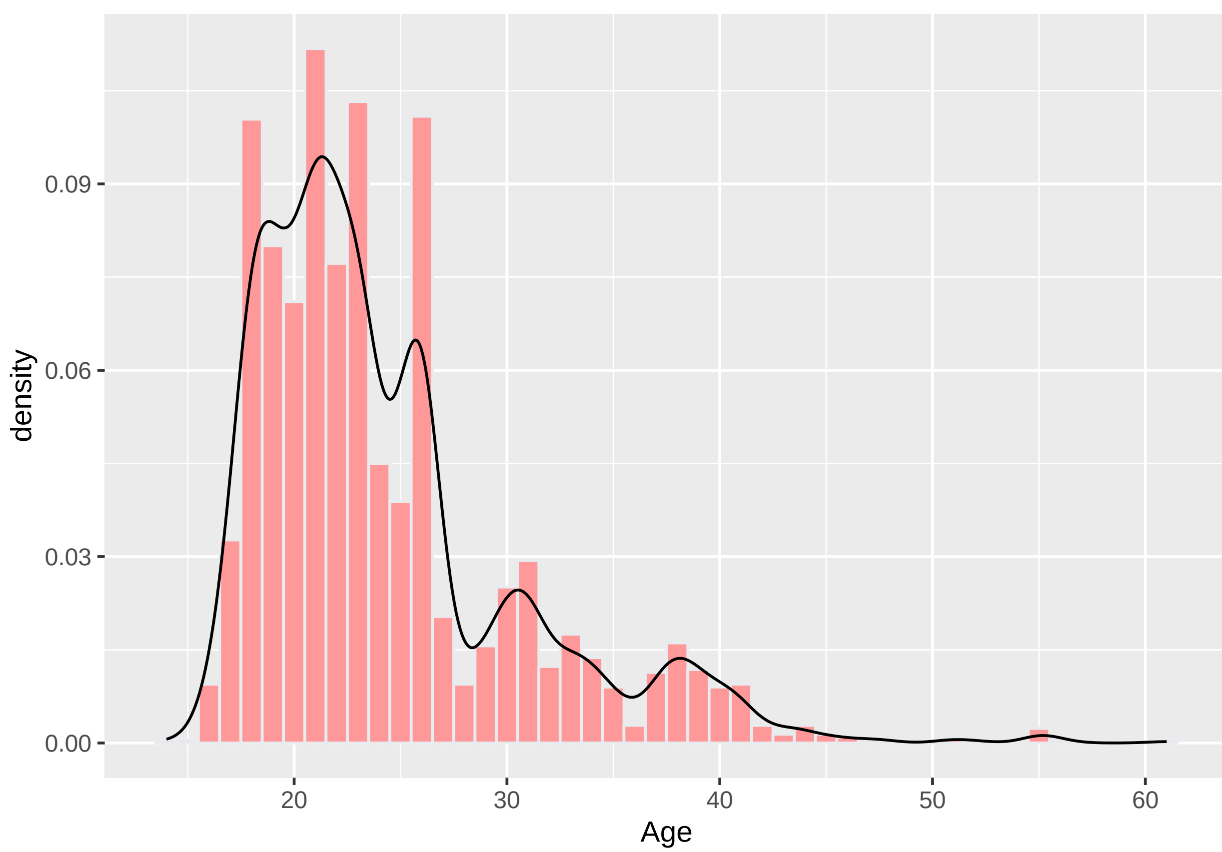
```
summary(data)
```

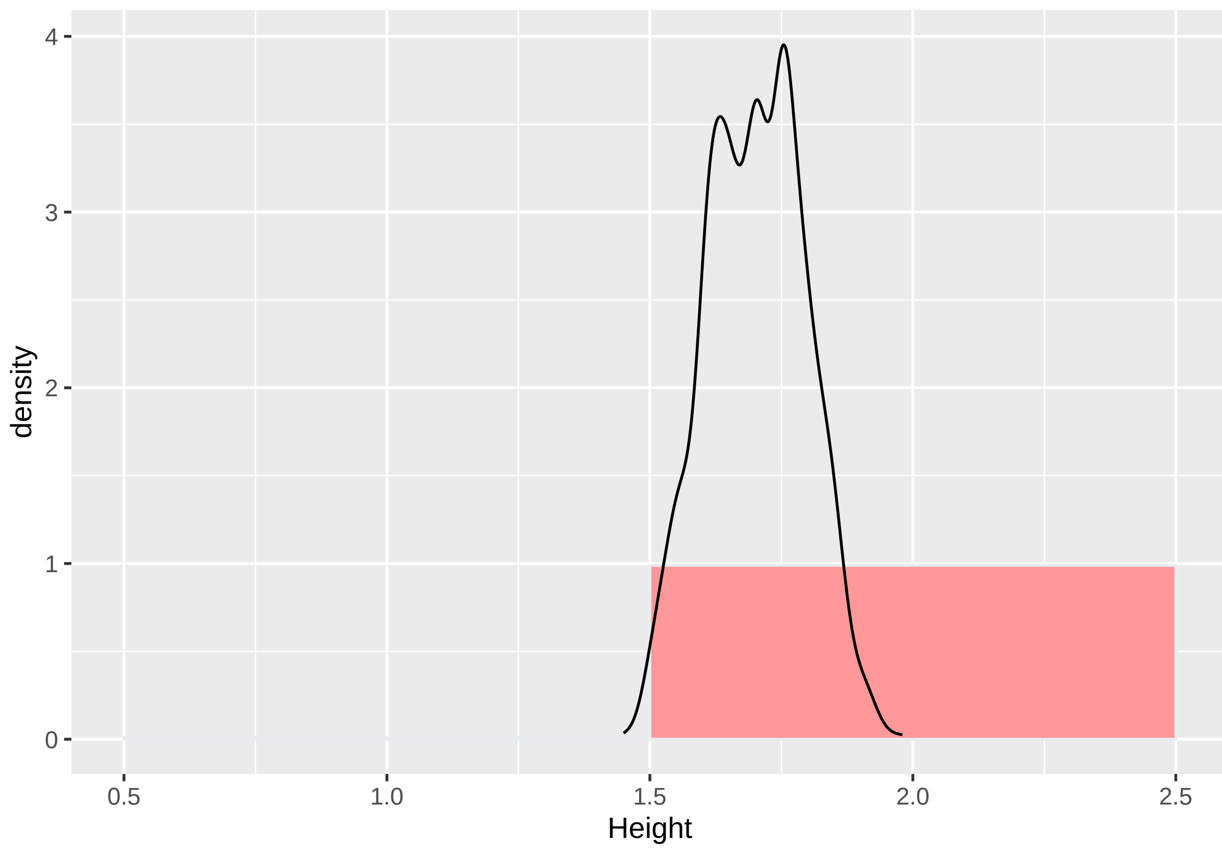
```
##     Gender               Age            Height          Weight      
##  Length:2111        Min.   :14.00   Min.   :1.450   Min.   : 39.00  
##  Class :character   1st Qu.:19.95   1st Qu.:1.630   1st Qu.: 65.47  
##  Mode  :character   Median :22.78   Median :1.700   Median : 83.00  
##                     Mean   :24.31   Mean   :1.702   Mean   : 86.59  
##                     3rd Qu.:26.00   3rd Qu.:1.768   3rd Qu.:107.43  
##                     Max.   :61.00   Max.   :1.980   Max.   :173.00  
##  family_history_with_overweight     FAVC                FCVC      
##  Length:2111                    Length:2111        Min.   :1.000  
##  Class :character               Class :character   1st Qu.:2.000  
##  Mode  :character               Mode  :character   Median :2.000  
##                                                    Mean   :2.423  
##                                                    3rd Qu.:3.000  
##                                                    Max.   :3.000  
##       NCP            CAEC              SMOKE                CH2O      
##  Min.   :1.000   Length:2111        Length:2111        Min.   :1.000  
##  1st Qu.:3.000   Class :character   Class :character   1st Qu.:2.000  
##  Median :3.000   Mode  :character   Mode  :character   Median :2.000  
##  Mean   :2.688                                         Mean   :2.015  
##  3rd Qu.:3.000                                         3rd Qu.:2.000  
```

```
##  Max.   :4.000                                          Max.   :3.000
##      SCC                 FAF              TUE              CALC
##  Length:2111        Min.   :0.000   Min.   :0.0000   Length:2111
##  Class :character   1st Qu.:0.000   1st Qu.:0.0000   Class :character
##  Mode  :character   Median :1.000   Median :1.0000   Mode  :character
##                     Mean   :1.007   Mean   :0.6646
##                     3rd Qu.:2.000   3rd Qu.:1.0000
##                     Max.   :3.000   Max.   :2.0000
##    MTRANS            NObeyesdad
##  Length:2111        Length:2111
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

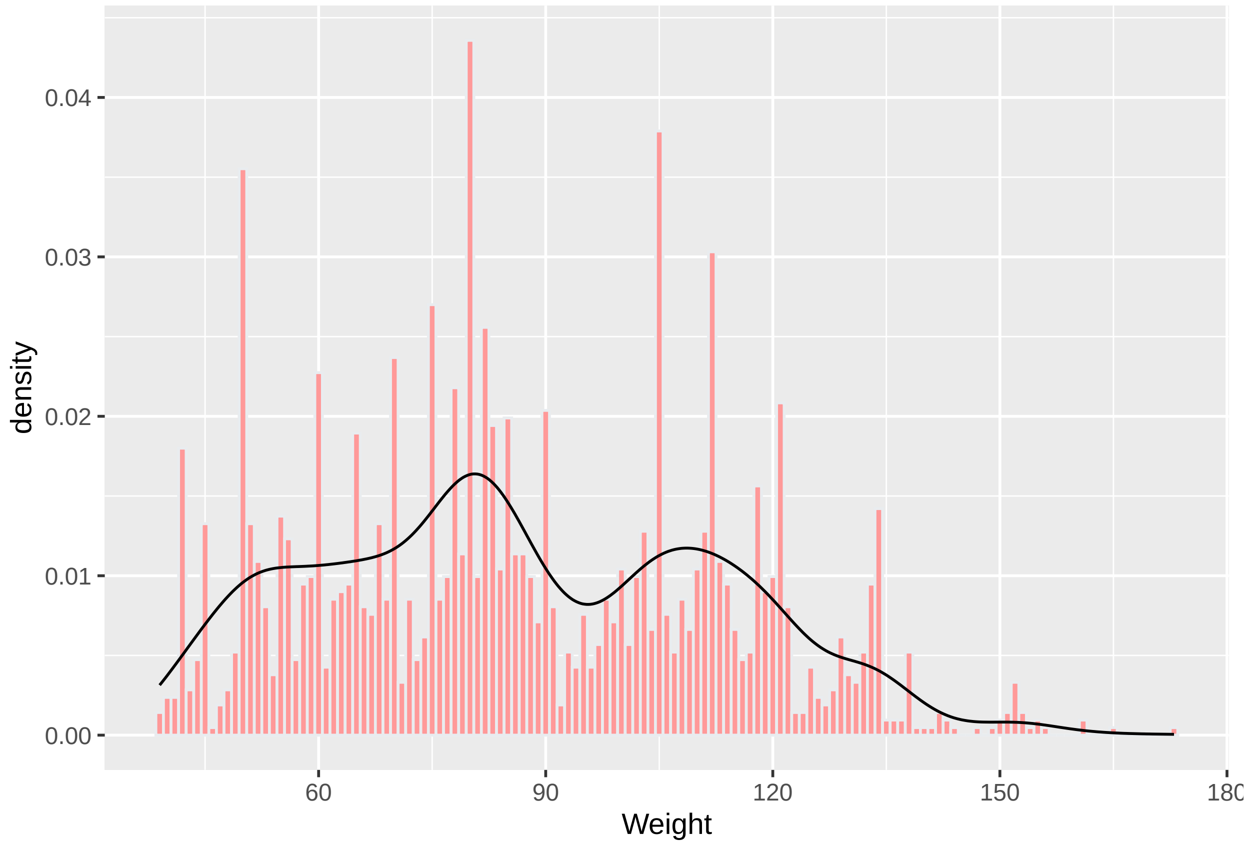
3- Distribution of Weight, Age and Height of all the respondents?

```r
for (var in num.cols) {
  #col <- eval(as.name(paste(var)))
  print(ggplot(data, aes(x=eval(as.name(paste(var))),y=after_stat(density))) + xlab(var) +
  geom_histogram(position='dodge', binwidth=1, fill="#FF9999", color="#e9ecef") +
  geom_density(alpha=0.25))
}
```





4
3
2
1
0
density
0.5
1.0
1.5
2.0
2.5
Height



The weight data is almost bimodal and has an average around the 80kg mark, while the height data has more of a symmetric, normal curve and has an average around the 1.7 meters mark.

4-Analyzing the categorical data and count

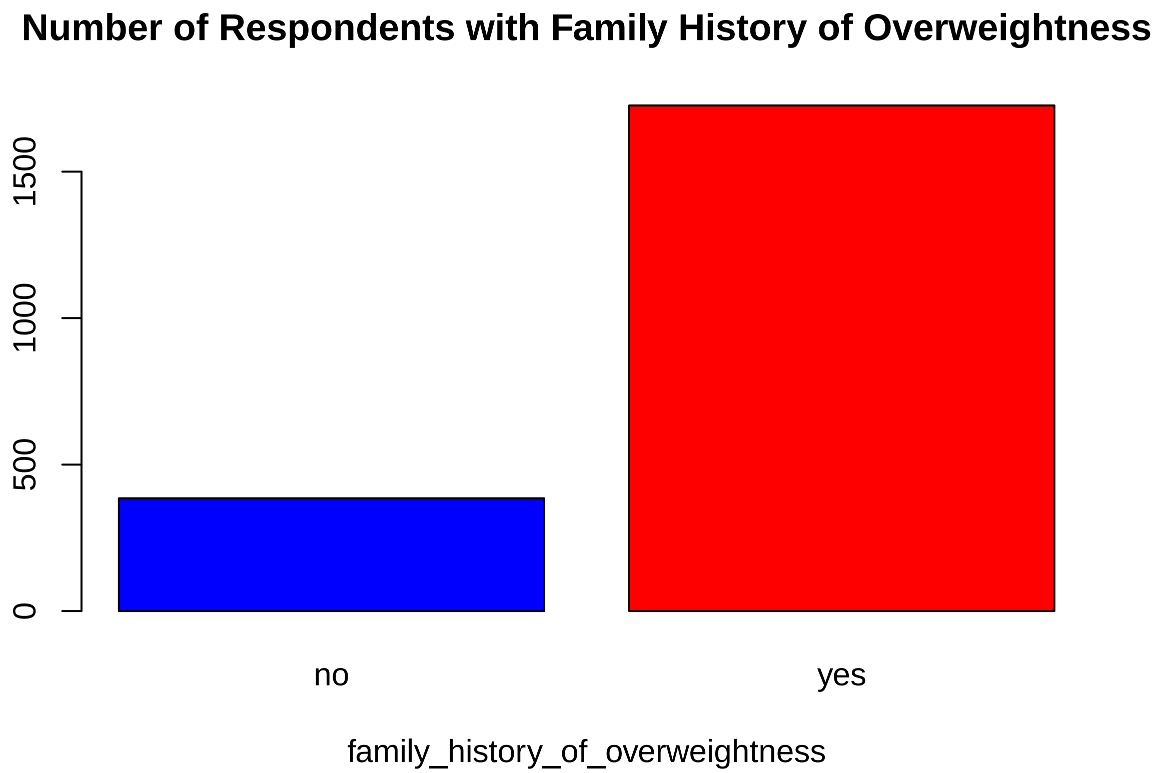
```
# Categorical predictors count
for (var in cat.cols) {
  #print(var)
  count(data[var]) %>% print()
}
```

```
##   Gender freq
## 1 Female 1043
## 2   Male 1068
##   family_history_with_overweight freq
## 1                             no  385
## 2                            yes 1726
##   FAVC freq
## 1   no  245
## 2  yes 1866
##   FCVC freq
## 1    1  102
## 2    2 1013
## 3    3  996
##   NCP freq
## 1   1  316
## 2   2  176
## 3   3 1470
```

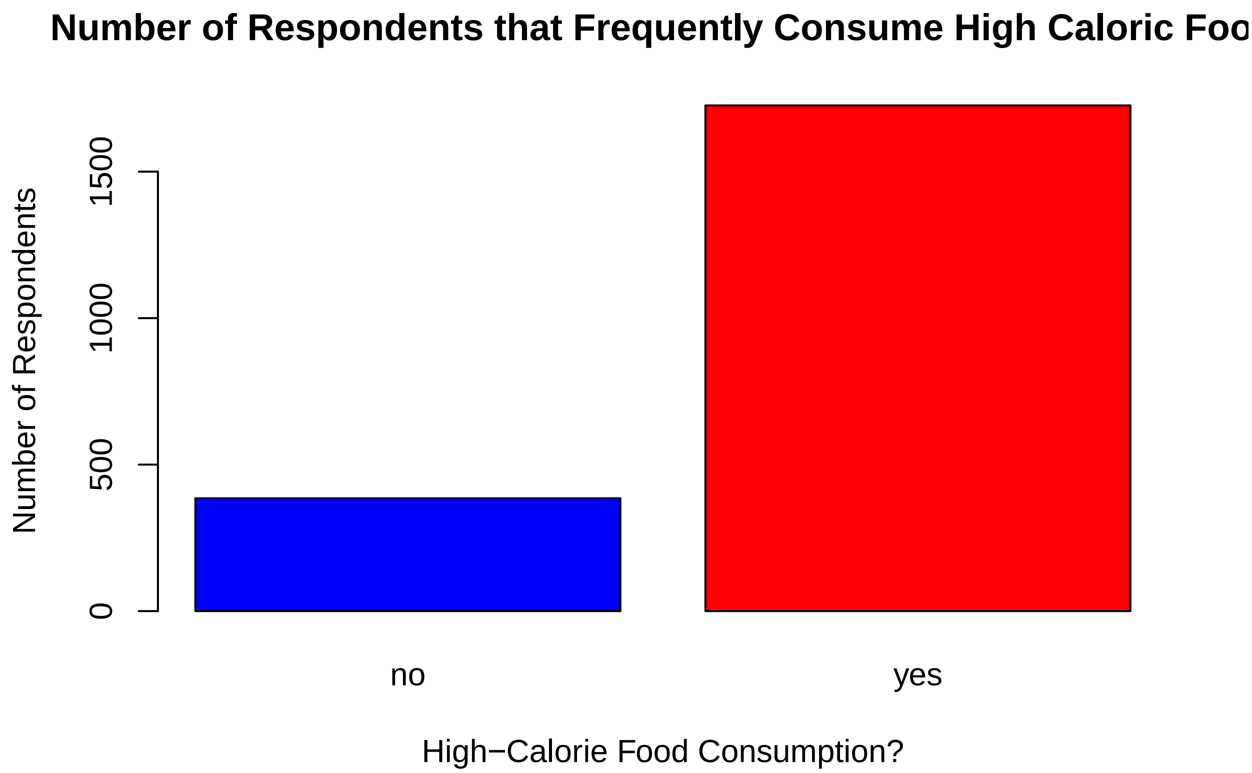
```
## 4   4  149
##         CAEC freq
## 1     Always   53
## 2 Frequently  242
## 3         no   51
## 4  Sometimes 1765
##   SMOKE freq
## 1    no 2067
## 2   yes   44
##   CH2O freq
## 1    1  485
## 2    2 1110
## 3    3  516
##   SCC freq
## 1  no 2015
## 2 yes   96
##   FAF freq
## 1   0  720
## 2   1  776
## 3   2  496
## 4   3  119
##   TUE freq
## 1   0  952
## 2   1  915
## 3   2  244
##         CALC freq
## 1     Always    1
## 2 Frequently   70
## 3         no  639
## 4  Sometimes 1401
##                  MTRANS freq
## 1            Automobile  457
## 2                  Bike    7
## 3             Motorbike   11
## 4 Public_Transportation 1580
## 5               Walking   56
```

5-How are respondents responding to yes/no questions?

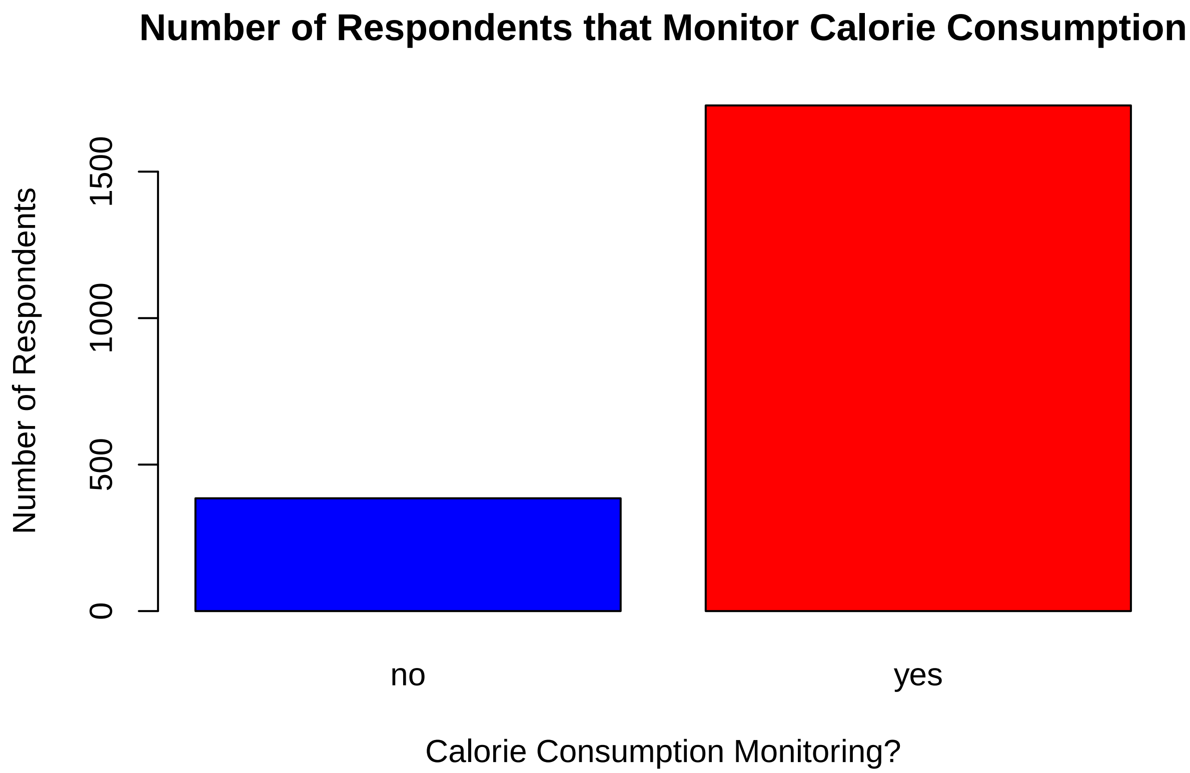
```
counts <- table(data$family_history_with_overweight)
barplot(counts, main="Number of Respondents with Family History of Overweightness",
xlab="family_history_of_overweightness",col=c("blue","red"))
```



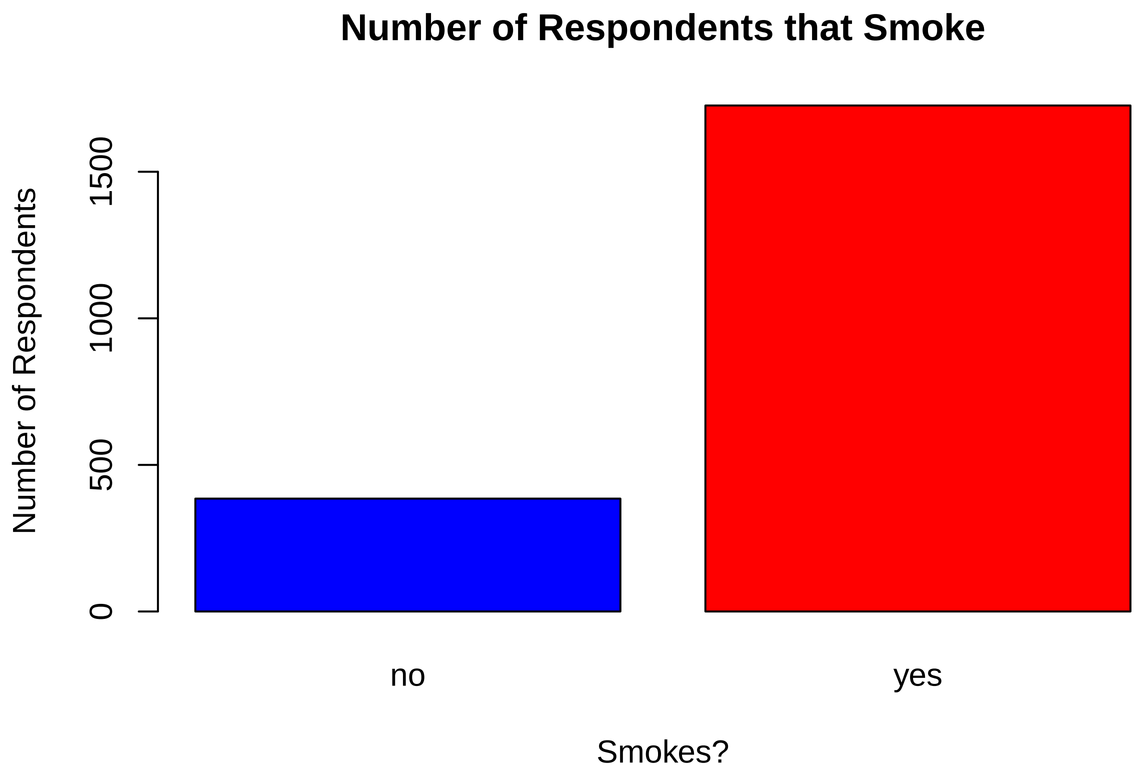

```
counts_1 <- table(data$FAVC)
barplot(counts, main="Number of Respondents that Frequently Consume High Caloric Food",
xlab="High-Calorie Food Consumption?",ylab = "Number of Respondents", col=c("blue","red"))
```




```
counts_2 <- table(data$SCC)
barplot(counts, main="Number of Respondents that Monitor Calorie Consumption",
xlab="Calorie Consumption Monitoring?",ylab = "Number of Respondents", col=c("blue","red"))
```




```
counts_3 <- table(data$SMOKE)
barplot(counts, main="Number of Respondents that Smoke",
xlab="Smokes?",ylab = "Number of Respondents", col=c("blue","red"))
```



Approx 98% of all respondents said "yes" to smoking, 81.76% responded yes to family history of being overweight, 88.35% responded yes to consumption of high calorific food (FAVC),95.4% said they don't monitor their calorie consumption.

6-Relationship between weight & height specifically , since they are used in calculating BMI Filtering the data based on genders

```
# Male data
library(easyreg)
data1 = data.frame(filter(data,Gender == 'Male'))
#data %>% filter()
data1=data.frame(data1$Weight,data1$Height)
#regplot(data1,model=1,digits=3, position=3, ylab="height", xlab="weight",col="red",
#        main = "Relationship between Weight & Height for Females")
```

```
# Female data

data2=data.frame(filter(data,Gender == "Female"))
#data %>% filter()
data2=data2[c("Weight","Height")]
#regplot(data2,model=1,digits=3, position=3, ylab="height", xlab="weight",col="red",
 #       main = "Relationship between Weight & Height for Females")
```

This graph shows us there is a trending upwards relationship between weight and height with both genders, with the regression line for females slightly steeper than that of males, meaning that the same increase in weight for females corresponds to a slightly larger increase in height. We can also see that data points corresponding to weights of male are more clustered than females.

Categorical Data Encoding

```r
# Encoding categorical to numeric
dat <- cbind(data)

# Label encoding categorical predictors with two levels into binary
dat$Gender <- ifelse(dat$Gender == "Male", 1, 0)
dat$FAVC <- ifelse(dat$FAVC == "yes", 1, 0)
dat$SMOKE <- ifelse(dat$SMOKE == "yes", 1, 0)
dat$SCC <- ifelse(dat$SCC == "yes", 1, 0)
#dat$CALC <- ifelse(dat$CALC == "yes", 1, 0)
dat$CALC <- mapvalues(dat$CALC,
          from=c("Always", "Frequently","Sometimes","no"),
          to=c(4,3,2,1))

dat$family_history_with_overweight <- ifelse(dat$family_history_with_overweight == "yes", 1, 0)
```

2 - One hot encoding categorical predictors with more than two levels

```r
#

one.hot <- dummyVars(~ CAEC + MTRANS, data = dat, fullRank = T)
dat_encoded <- data.frame(predict(one.hot, newdata = dat))
```

3-Replacing categorical values in response column with numeric values

```r
# Ordinal Encoding
dat$NObeyesdad <- mapvalues(dat$NObeyesdad,
          from=c("Insufficient_Weight","Normal_Weight","Obesity_Type_I", "Obesity_Type_II",
                 "Obesity_Type_III", "Overweight_Level_I", "Overweight_Level_II"),
          to=c(0, 1, 2, 3, 4, 5, 6))
```

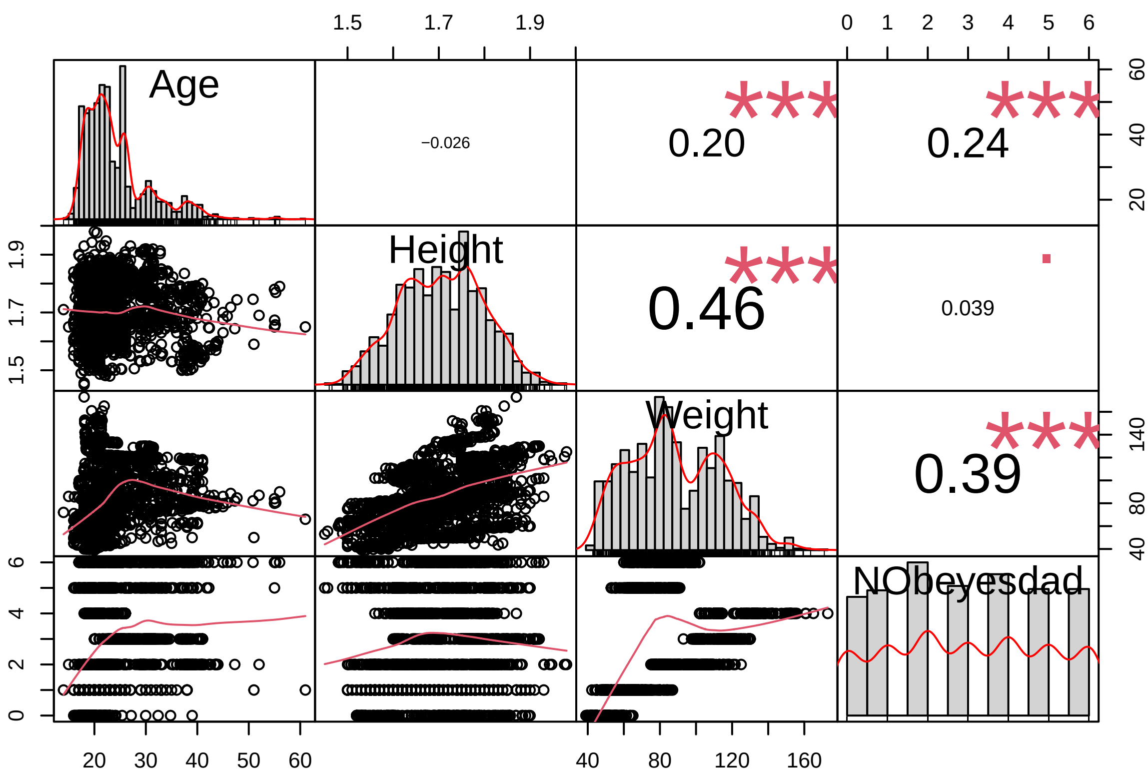
```r
# merging data frame

data.final <- cbind(dat, dat_encoded)

data.final <- data.final[,-9]
data.final <- data.final[,-15]

#data.final <- select(data.final, -CAEC, -MTRANS)
```

Correlation of Variables

```r
library(PerformanceAnalytics)

# converting datatype to numeric
df <- sapply(data.final, as.numeric)

chart.Correlation(df[,c('Age', 'Height', 'Weight', 'NObeyesdad')], histogram=TRUE, pch=19)
```

```
## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter
```

```
## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter
```



Predictive Models

Train Test Stratified Split (75-25)

```r
# train test split
set.seed(545)
# stratified split; train: 75%, test: 25%
indices <- createDataPartition(data.final$NObeyesdad, p = 0.75, list = FALSE)

train <- data.final[indices,]
test <- data.final[-indices,]
```

Decision Tree

Decision trees use a divide-and-conquer approach to make predictions. The goal is to split the training data into subsets based on certain features, with each split resulting in a more homogeneous subset. The splits are chosen to maximize the subsets' homogeneity, with the ultimate goal of producing leaf nodes that are as pure as possible, meaning that they contain a single class label. To measure the homogeneity of a subset, decision trees use impurity measures such as entropy or misclassification error or the Gini index.

These measures calculate the degree of disorderness or randomness in the data. If the data is completely pure, the impurity measure will be 0. The impurity measure will be maximal if the data is equally distributed among different classes.

At each step in the tree-building process, the algorithm selects the feature and the split point that result in the lowest impurity of the subsets. The process continues until the leaf nodes are pure or until a pre-specified stopping criterion is reached.

The primary reason for choosing a Decision tree as the base model is the ease of interpretation and also to have a better understanding of the predictors that better splits the response variable. Moreover, the tree is robust to outliers and will not produce a biased result.

```r
# Decision Tree

#train.df <- data.frame(sapply(train, as.numeric))
#train.df['NObeyesdad'] <- data['NObeyesdad']

tree.fit <- rpart(NObeyesdad ~ . , data = train, method='class')
tree.fit
```

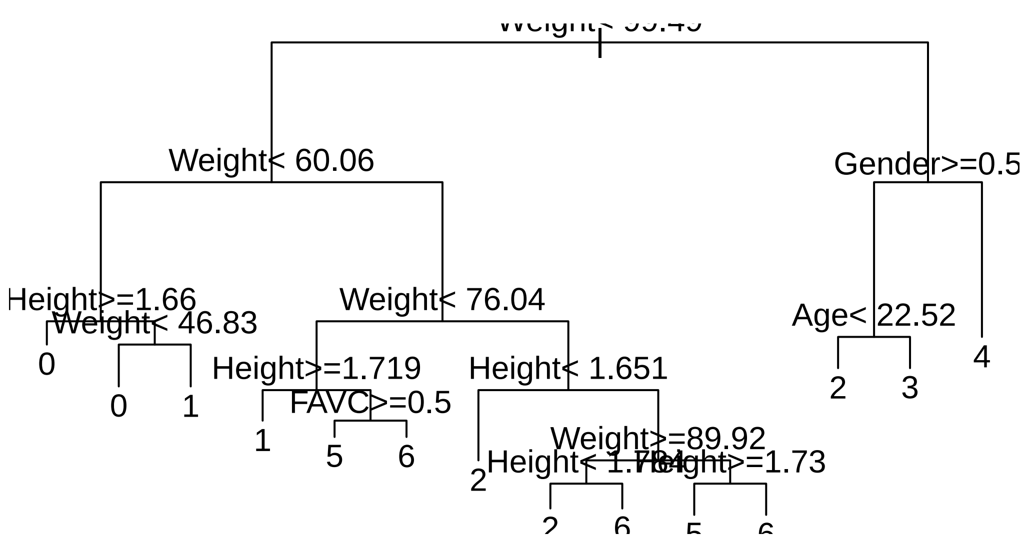
```
## n= 1586
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 1586 1322 2 (0.13 0.14 0.17 0.14 0.15 0.14 0.14)
##    2) Weight< 99.48537 1040  822 5 (0.2 0.21 0.18 0.0029 0 0.21 0.21)
##      4) Weight< 60.059 313  114 0 (0.64 0.33 0 0 0 0.029 0.0032)
##        8) Height>=1.660223 145   15 0 (0.9 0.1 0 0 0 0 0) *
##        9) Height< 1.660223 168   79 1 (0.41 0.53 0 0 0 0.054 0.006)
##         18) Weight< 46.82781 73    7 0 (0.9 0.096 0 0 0 0 0) *
##         19) Weight>=46.82781 95   13 1 (0.032 0.86 0 0 0 0.095 0.011) *
##      5) Weight>=60.059 727  512 6 (0.0069 0.15 0.25 0.0041 0 0.29 0.3)
##       10) Weight< 76.04126 266  136 5 (0.019 0.36 0.011 0 0 0.49 0.12)
##         20) Height>=1.719469 60   11 1 (0.083 0.82 0 0 0 0.1 0) *
##         21) Height< 1.719469 206   82 5 (0 0.22 0.015 0 0 0.6 0.16)
##           42) FAVC>=0.5 154   40 5 (0 0.24 0.019 0 0 0.74 0) *
##           43) FAVC< 0.5 52   19 6 (0 0.17 0 0 0 0.19 0.63) *
##       11) Weight>=76.04126 461  279 6 (0 0.037 0.39 0.0065 0 0.17 0.39)
##         22) Height< 1.650932 126   15 2 (0 0 0.88 0.024 0 0 0.095) *
##         23) Height>=1.650932 335  165 6 (0 0.051 0.21 0 0 0.24 0.51)
##           46) Weight>=89.92407 102   41 2 (0 0 0.6 0 0 0.029 0.37)
##             92) Height< 1.783661 60    1 2 (0 0 0.98 0 0 0 0.017) *
##             93) Height>=1.783661 42    5 6 (0 0 0.048 0 0 0.071 0.88) *
##           47) Weight< 89.92407 233  101 6 (0 0.073 0.034 0 0 0.33 0.57)
##             94) Height>=1.729672 123   48 5 (0 0.14 0 0 0 0.61 0.25) *
##             95) Height< 1.729672 110    9 6 (0 0 0.073 0 0 0.0091 0.92) *
##    3) Weight>=99.48537 546  303 4 (0 0 0.15 0.4 0.45 0 0.0037)
##      6) Gender>=0.5 303   84 3 (0 0 0.27 0.72 0.0033 0 0.0066)
##       12) Age< 22.52469 65   11 2 (0 0 0.83 0.15 0.015 0 0) *
##       13) Age>=22.52469 238   29 3 (0 0 0.11 0.88 0 0 0.0084) *
##      7) Gender< 0.5 243    1 4 (0 0 0 0.0041 1 0 0) *
```

```r
tree.fit$variable.importance
```

```
##                    Weight                    Height
##               610.2502243               395.8226669
##                    Gender                       Age
```

```
##                   253.8260925                     208.6427099
##                          FCVC                             FAF
##                   128.6183963                      83.5285146
## family_history_with_overweight                          NCP
##                    41.8523189                      41.3131342
##                 CAECFrequently                           FAVC
##                    39.7071379                      36.8561879
##                          CALC                     CAECSometimes
##                    30.5450940                      20.3820454
##                        CAECno    MTRANSPublic_Transportation
##                     6.6701194                       4.7872149
##                           TUE                    MTRANSWalking
##                     2.1849740                       1.4795921
##                    MTRANSBike
##                     0.9863947
```

```r
plot(tree.fit)
text(tree.fit, pretty=0)
```




```r
#zm()
```

In the fitted decision tree, Weight was identified as the most significant predictor that best splits the response variable. Additionally, Gender, Height, Age, FAVC were also selected in classifying the input values. As per the constructed tree, if the persons weigh more than 100 kg and are male, they are more likely to have Obesity type III. If the persons weigh less than 61 kg, they are more likely to be in Insufficient weight category. The tree contains 14 terminal nodes. The complexity of the tree can be reduced by pruning.

```r
# Decision tree Train Test set prediction result

tree.predtrain <- predict(tree.fit, train, type = "class")
tree.predtest <- predict(tree.fit, test, type = "class")

train.error <- mean(tree.predtrain != train$NObeyesdad)
test.error <- mean(tree.predtest != test$NObeyesdad)

print(paste("Misclassification error rate in train = ", train.error))
```

```
## [1] "Misclassification error rate in train =  0.141235813366961"
```

```r
print(paste("Misclassification error rate in test = ", test.error))
```

```
## [1] "Misclassification error rate in test =  0.165714285714286"
```

```r
confusionMatrix(tree.predtest,
                as.factor(test$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6
##          0 63  8  0  0  0  0  0
##          1  5 48  0  0  0  3  0
##          2  0  0 67  8  0  0  2
##          3  0  0 14 66  0  0  3
##          4  0  0  0  0 81  0  0
##          5  0 12  1  0  0 66 20
##          6  0  3  5  0  0  3 47
##
## Overall Statistics
##
##                Accuracy : 0.8343
##                  95% CI : (0.7997, 0.8651)
##     No Information Rate : 0.1657
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8066
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9265  0.67606   0.7701   0.8919   1.0000   0.9167
## Specificity            0.9825  0.98238   0.9772   0.9623   1.0000   0.9272
## Pos Pred Value         0.8873  0.85714   0.8701   0.7952   1.0000   0.6667
## Neg Pred Value         0.9890  0.95096   0.9554   0.9819   1.0000   0.9859
## Prevalence             0.1295  0.13524   0.1657   0.1410   0.1543   0.1371
## Detection Rate         0.1200  0.09143   0.1276   0.1257   0.1543   0.1257
## Detection Prevalence   0.1352  0.10667   0.1467   0.1581   0.1543   0.1886
## Balanced Accuracy      0.9545  0.82922   0.8736   0.9271   1.0000   0.9219
##                      Class: 6
## Sensitivity           0.65278
```

```
## Specificity           0.97572
## Pos Pred Value        0.81034
## Neg Pred Value        0.94647
## Prevalence            0.13714
## Detection Rate        0.08952
## Detection Prevalence  0.11048
## Balanced Accuracy     0.81425
```

```
confusionMatrix(tree.predtrain,
                as.factor(train$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6
##          0 196  22   0   0   0   0   0
##          1   8 131   0   0   0  15   1
##          2   0   0 224  13   1   0  13
##          3   0   0  27 209   0   0   2
##          4   0   0   0   1 242   0   0
##          5   0  54   3   0   0 189  31
##          6   0   9  10   0   0  14 171
##
## Overall Statistics
##
##                Accuracy : 0.8588
##                  95% CI : (0.8406, 0.8755)
##     No Information Rate : 0.1665
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8351
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9608  0.60648   0.8485   0.9372   0.9959   0.8670
## Specificity            0.9841  0.98248   0.9796   0.9787   0.9993   0.9357
## Pos Pred Value         0.8991  0.84516   0.8924   0.8782   0.9959   0.6823
## Neg Pred Value         0.9942  0.94060   0.9700   0.9896   0.9993   0.9778
## Prevalence             0.1286  0.13619   0.1665   0.1406   0.1532   0.1375
## Detection Rate         0.1236  0.08260   0.1412   0.1318   0.1526   0.1192
## Detection Prevalence   0.1375  0.09773   0.1583   0.1501   0.1532   0.1747
## Balanced Accuracy      0.9724  0.79448   0.9140   0.9580   0.9976   0.9013
##                      Class: 6
## Sensitivity            0.7844
## Specificity            0.9759
## Pos Pred Value         0.8382
## Neg Pred Value         0.9660
## Prevalence             0.1375
## Detection Rate         0.1078
## Detection Prevalence   0.1286
## Balanced Accuracy      0.8801
```

```r
best_cp <- tree.fit$cptable[which.min(tree.fit$cptable[, "xerror"]), "CP"]
best_cp
```

```
## [1] 0.01
```

```r
tree.fit$cptable
```

```
##           CP nsplit rel error    xerror       xstd
## 1 0.15695915      0 1.0000000 1.0000000 0.01122108
## 2 0.14826021      2 0.6860817 0.7042360 0.01483243
## 3 0.07413011      3 0.5378215 0.5438729 0.01499654
## 4 0.03328290      5 0.3895613 0.3963691 0.01416918
## 5 0.03252648      6 0.3562784 0.3888048 0.01409925
## 6 0.02987897      7 0.3237519 0.3668684 0.01387974
## 7 0.02571861      9 0.2639939 0.2836611 0.01279984
## 8 0.01739788     12 0.1868381 0.2072617 0.01138831
## 9 0.01000000     13 0.1694402 0.1906203 0.01101273
```

Pruning

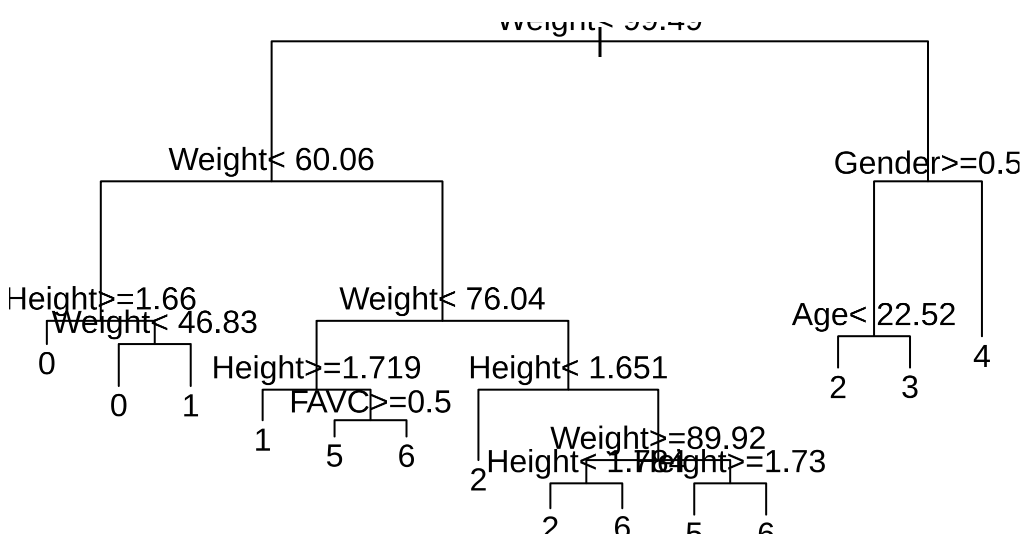
In Decision Tree, pruning is a technique that removes parts of the tree that seems to be redundant in classifying the response categories. Thus, it reduces the complexity of the tree based on the cross validation error.

```r
tree.prune <- prune(tree.fit, cp = best_cp)
tree.prune
```

```
## n= 1586
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 1586 1322 2 (0.13 0.14 0.17 0.14 0.15 0.14 0.14)
##    2) Weight< 99.48537 1040  822 5 (0.2 0.21 0.18 0.0029 0 0.21 0.21)
##      4) Weight< 60.059 313  114 0 (0.64 0.33 0 0 0 0.029 0.0032)
##        8) Height>=1.660223 145   15 0 (0.9 0.1 0 0 0 0 0) *
##        9) Height< 1.660223 168   79 1 (0.41 0.53 0 0 0 0.054 0.006)
##         18) Weight< 46.82781 73    7 0 (0.9 0.096 0 0 0 0 0) *
##         19) Weight>=46.82781 95   13 1 (0.032 0.86 0 0 0 0.095 0.011) *
##      5) Weight>=60.059 727  512 6 (0.0069 0.15 0.25 0.0041 0 0.29 0.3)
##       10) Weight< 76.04126 266  136 5 (0.019 0.36 0.011 0 0 0.49 0.12)
##         20) Height>=1.719469 60   11 1 (0.083 0.82 0 0 0 0.1 0) *
##         21) Height< 1.719469 206   82 5 (0 0.22 0.015 0 0 0.6 0.16)
##           42) FAVC>=0.5 154   40 5 (0 0.24 0.019 0 0 0.74 0) *
##           43) FAVC< 0.5 52   19 6 (0 0.17 0 0 0 0.19 0.63) *
##       11) Weight>=76.04126 461  279 6 (0 0.037 0.39 0.0065 0 0.17 0.39)
##         22) Height< 1.650932 126   15 2 (0 0 0.88 0.024 0 0 0.095) *
##         23) Height>=1.650932 335  165 6 (0 0.051 0.21 0 0 0.24 0.51)
##           46) Weight>=89.92407 102   41 2 (0 0 0.6 0 0 0.029 0.37)
##             92) Height< 1.783661 60    1 2 (0 0 0.98 0 0 0 0.017) *
##             93) Height>=1.783661 42    5 6 (0 0 0.048 0 0 0.071 0.88) *
##           47) Weight< 89.92407 233  101 6 (0 0.073 0.034 0 0 0.33 0.57)
##             94) Height>=1.729672 123   48 5 (0 0.14 0 0 0 0.61 0.25) *
##             95) Height< 1.729672 110    9 6 (0 0 0.073 0 0 0.0091 0.92) *
##    3) Weight>=99.48537 546  303 4 (0 0 0.15 0.4 0.45 0 0.0037)
```

```
##       6) Gender>=0.5 303   84 3 (0 0 0.27 0.72 0.0033 0 0.0066)
##        12) Age< 22.52469 65   11 2 (0 0 0.83 0.15 0.015 0 0) *
##        13) Age>=22.52469 238   29 3 (0 0 0.11 0.88 0 0 0.0084) *
##       7) Gender< 0.5 243    1 4 (0 0 0 0.0041 1 0 0) *
```

```r
plot(tree.prune)
text(tree.prune, pretty=0)
```




```r
#zm()
```

```r
# Train Test set prediction result

tree.predtrain <- predict(tree.prune, train, type = "class")
tree.predtest <- predict(tree.prune, test, type = "class")

train.error <- mean(tree.predtrain != train$NObeyesdad)
test.error <- mean(tree.predtest != test$NObeyesdad)

print(paste("Misclassification error rate in train = ", train.error))
```

```
## [1] "Misclassification error rate in train =  0.141235813366961"
```

```r
print(paste("Misclassification error rate in test = ", test.error))
```

```
## [1] "Misclassification error rate in test =  0.165714285714286"
```

Pruning didn't reduce the tree complexity without increasing the misclassification error rate in both train and test. Based on the cross validation error, the tree was pruned but it increased the misclassification error rate from 15% to 27% in train data and from 17% to 29% in the test data.

```r
confusionMatrix(tree.predtest,
                as.factor(test$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6
##          0 63  8  0  0  0  0  0
##          1  5 48  0  0  0  3  0
##          2  0  0 67  8  0  0  2
##          3  0  0 14 66  0  0  3
##          4  0  0  0  0 81  0  0
##          5  0 12  1  0  0 66 20
##          6  0  3  5  0  0  3 47
##
## Overall Statistics
##
##                Accuracy : 0.8343
##                  95% CI : (0.7997, 0.8651)
##     No Information Rate : 0.1657
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8066
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9265  0.67606   0.7701   0.8919   1.0000   0.9167
## Specificity            0.9825  0.98238   0.9772   0.9623   1.0000   0.9272
## Pos Pred Value         0.8873  0.85714   0.8701   0.7952   1.0000   0.6667
## Neg Pred Value         0.9890  0.95096   0.9554   0.9819   1.0000   0.9859
## Prevalence             0.1295  0.13524   0.1657   0.1410   0.1543   0.1371
## Detection Rate         0.1200  0.09143   0.1276   0.1257   0.1543   0.1257
## Detection Prevalence   0.1352  0.10667   0.1467   0.1581   0.1543   0.1886
## Balanced Accuracy      0.9545  0.82922   0.8736   0.9271   1.0000   0.9219
##                      Class: 6
## Sensitivity           0.65278
## Specificity           0.97572
## Pos Pred Value        0.81034
## Neg Pred Value        0.94647
## Prevalence            0.13714
## Detection Rate        0.08952
## Detection Prevalence  0.11048
## Balanced Accuracy     0.81425
```

```r
confusionMatrix(tree.predtrain,
                as.factor(train$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6
```

```
##          0 196  22   0   0   0   0   0
##          1   8 131   0   0   0  15   1
##          2   0   0 224  13   1   0  13
##          3   0   0  27 209   0   0   2
##          4   0   0   0   1 242   0   0
##          5   0  54   3   0   0 189  31
##          6   0   9  10   0   0  14 171
##
## Overall Statistics
##
##                Accuracy : 0.8588
##                  95% CI : (0.8406, 0.8755)
##     No Information Rate : 0.1665
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8351
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9608  0.60648   0.8485   0.9372   0.9959   0.8670
## Specificity            0.9841  0.98248   0.9796   0.9787   0.9993   0.9357
## Pos Pred Value         0.8991  0.84516   0.8924   0.8782   0.9959   0.6823
## Neg Pred Value         0.9942  0.94060   0.9700   0.9896   0.9993   0.9778
## Prevalence             0.1286  0.13619   0.1665   0.1406   0.1532   0.1375
## Detection Rate         0.1236  0.08260   0.1412   0.1318   0.1526   0.1192
## Detection Prevalence   0.1375  0.09773   0.1583   0.1501   0.1532   0.1747
## Balanced Accuracy      0.9724  0.79448   0.9140   0.9580   0.9976   0.9013
##                      Class: 6
## Sensitivity            0.7844
## Specificity            0.9759
## Pos Pred Value         0.8382
## Neg Pred Value         0.9660
## Prevalence             0.1375
## Detection Rate         0.1078
## Detection Prevalence   0.1286
## Balanced Accuracy      0.8801
```

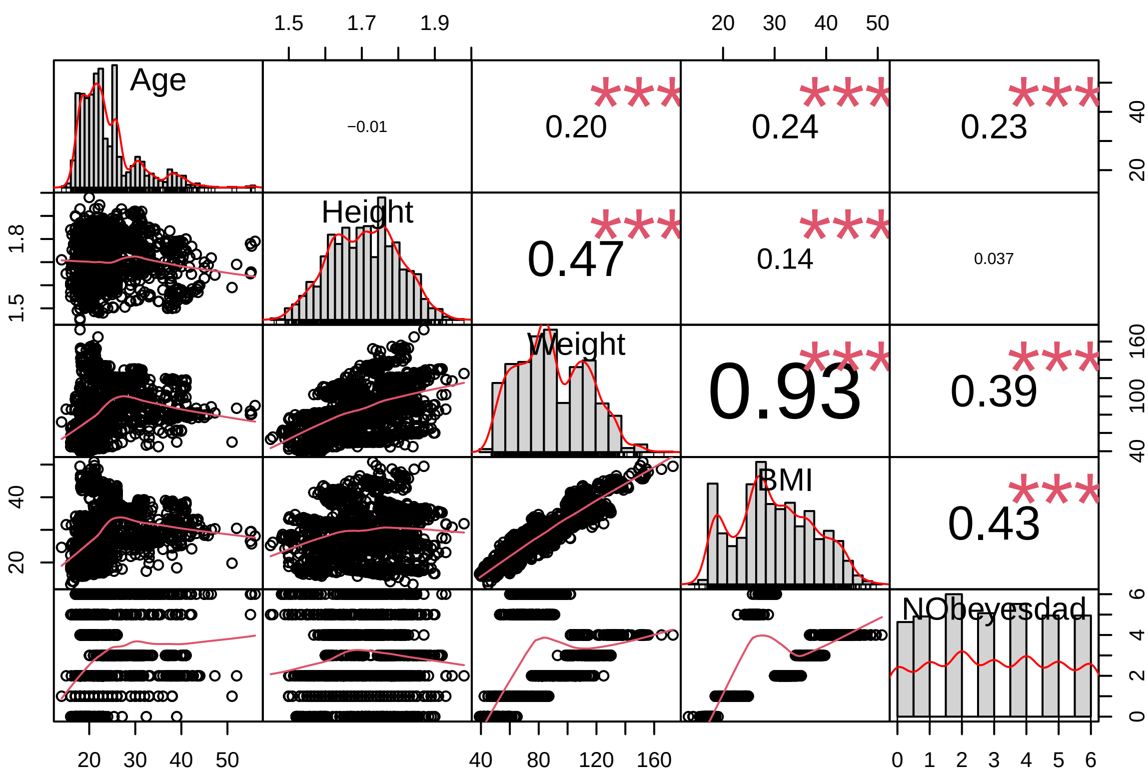
Feature Engineering - BMI

Creating a new feature BMI from Weight and Height predictors. Feature engineering can improve the performance of the model.

```r
# Adding new feature BMI (weight/(height^2))

train <- transform(train, BMI=Weight/(Height^2))
test <- transform(test, BMI=Weight/(Height^2))
```

```r
df <- sapply(train, as.numeric)

chart.Correlation(df[,c('Age', 'Height', 'Weight', 'BMI', 'NObeyesdad')],
                  histogram=TRUE, pch=19)
```

```
## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter
```



BMI can clearly separate the 7 categories of the response variable.

Decision Tree - data with BMI

```
tree.fit <- rpart(NObeyesdad ~ . , data = train)
```

```r
tree.predtrain <- predict(tree.fit, train, type = "class")
tree.predtest <- predict(tree.fit, test, type = "class")

train.error <- mean(tree.predtrain != train$NObeyesdad)
test.error <- mean(tree.predtest != test$NObeyesdad)

print(paste("Misclassification error rate in train = ", train.error))
```

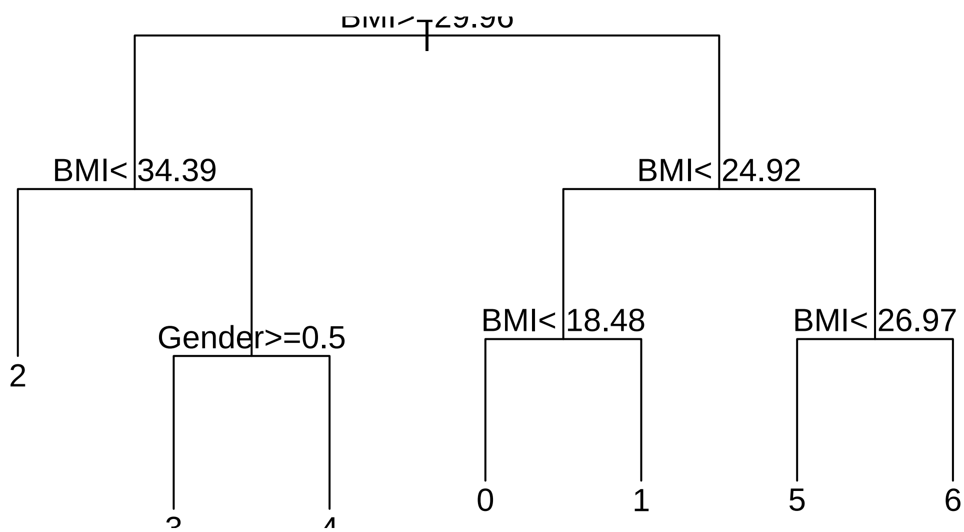
```
## [1] "Misclassification error rate in train =  0.0220680958385876"
```

```r
print(paste("Misclassification error rate in test = ", test.error))
```

```
## [1] "Misclassification error rate in test =  0.0361904761904762"
```

Adding new feature 'BMI' decreased the misclassification error rate in test data from 16.6% to 3.6%.

```r
plot(tree.fit)
text(tree.fit, pretty = 0)
```




```r
#zm()
```

```r
# Confusion matrix
confusionMatrix(tree.predtrain, as.factor(train$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6
##          0 201   0   0   0   0   0   0
```

```
##          1   3 216   0   0   0   7   0
##          2   0   0 258   3   0   0   3
##          3   0   0   5 218   1   0   0
##          4   0   0   1   2 242   0   0
##          5   0   0   0   0   0 205   4
##          6   0   0   0   0   0   6 211
##
## Overall Statistics
##
##                Accuracy : 0.9779
##                  95% CI : (0.9694, 0.9846)
##     No Information Rate : 0.1665
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9742
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9853   1.0000   0.9773   0.9776   0.9959   0.9404
## Specificity            1.0000   0.9927   0.9955   0.9956   0.9978   0.9971
## Pos Pred Value         1.0000   0.9558   0.9773   0.9732   0.9878   0.9809
## Neg Pred Value         0.9978   1.0000   0.9955   0.9963   0.9993   0.9906
## Prevalence             0.1286   0.1362   0.1665   0.1406   0.1532   0.1375
## Detection Rate         0.1267   0.1362   0.1627   0.1375   0.1526   0.1293
## Detection Prevalence   0.1267   0.1425   0.1665   0.1412   0.1545   0.1318
## Balanced Accuracy      0.9926   0.9964   0.9864   0.9866   0.9968   0.9687
##                      Class: 6
## Sensitivity            0.9679
## Specificity            0.9956
## Pos Pred Value         0.9724
## Neg Pred Value         0.9949
## Prevalence             0.1375
## Detection Rate         0.1330
## Detection Prevalence   0.1368
## Balanced Accuracy      0.9818
```

```
confusionMatrix(tree.predtest, as.factor(test$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6
##          0 66  0  0  0  0  0  0
##          1  2 71  0  0  0  2  0
##          2  0  0 80  0  0  0  1
##          3  0  0  5 74  0  0  0
##          4  0  0  1  0 81  0  0
##          5  0  0  0  0  0 66  3
##          6  0  0  1  0  0  4 68
##
## Overall Statistics
##
```

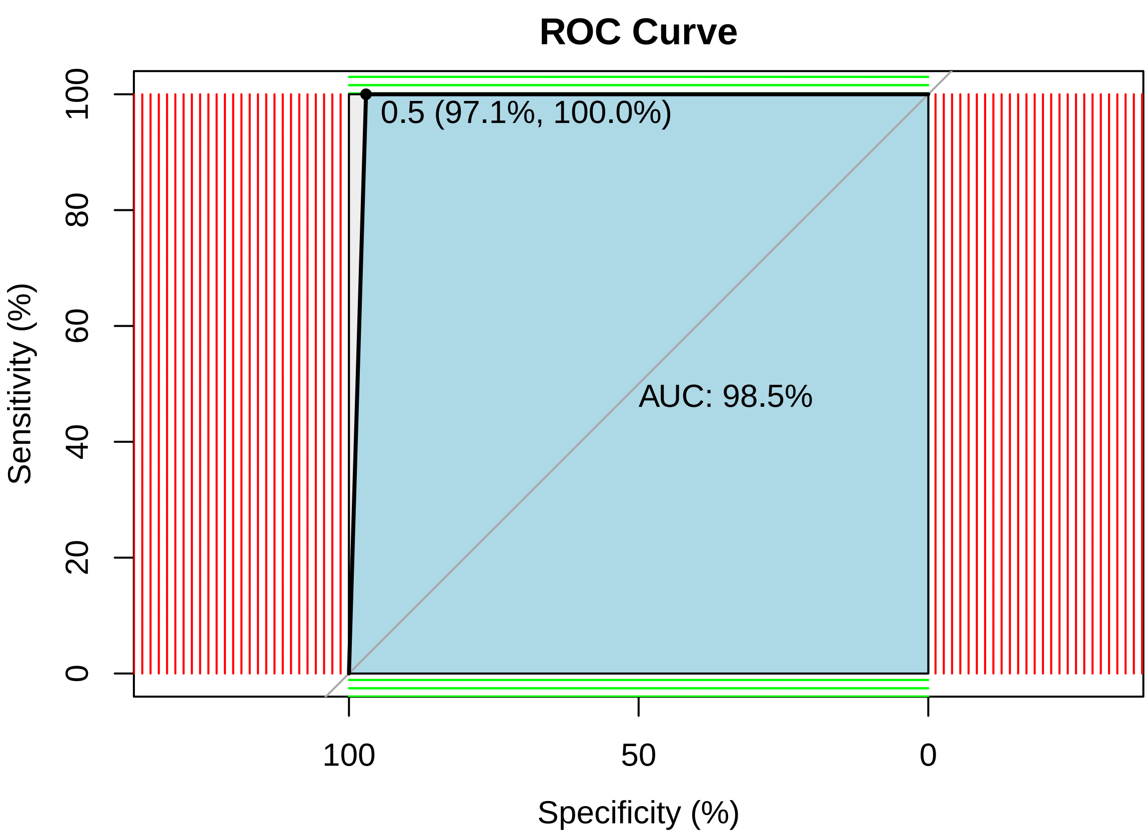
```
##                Accuracy : 0.9638
##                  95% CI : (0.9441, 0.9781)
##     No Information Rate : 0.1657
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9577
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9706   1.0000   0.9195   1.0000   1.0000   0.9167
## Specificity            1.0000   0.9912   0.9977   0.9889   0.9977   0.9934
## Pos Pred Value         1.0000   0.9467   0.9877   0.9367   0.9878   0.9565
## Neg Pred Value         0.9956   1.0000   0.9842   1.0000   1.0000   0.9868
## Prevalence             0.1295   0.1352   0.1657   0.1410   0.1543   0.1371
## Detection Rate         0.1257   0.1352   0.1524   0.1410   0.1543   0.1257
## Detection Prevalence   0.1257   0.1429   0.1543   0.1505   0.1562   0.1314
## Balanced Accuracy      0.9853   0.9956   0.9586   0.9945   0.9989   0.9550
##                      Class: 6
## Sensitivity            0.9444
## Specificity            0.9890
## Pos Pred Value         0.9315
## Neg Pred Value         0.9912
## Prevalence             0.1371
## Detection Rate         0.1295
## Detection Prevalence   0.1390
## Balanced Accuracy      0.9667
```

```r
p1 <- predict(tree.fit, test, type = 'prob')
p1 <- p1[,2]
r <- multiclass.roc(test$NObeyesdad, p1, percent = TRUE)
```

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

## Setting direction: controls > cases
## Setting direction: controls > cases
## Setting direction: controls > cases
## Setting direction: controls > cases
## Setting direction: controls > cases

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

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

```
roc <- r[['rocs']]
r1 <- roc[[1]]
plot.roc(r1,
         print.auc=TRUE,
         auc.polygon=TRUE,
         grid=c(0.1, 0.2),
         grid.col=c("green", "red"),
         max.auc.polygon=TRUE,
         auc.polygon.col="lightblue",
         print.thres=TRUE,
         main= 'ROC Curve')
```




```
# Only using BMI predictor

tree.fit <- rpart(NObeyesdad ~ BMI , data = train)

tree.predtrain <- predict(tree.fit, train['BMI'], type = "class")
tree.predtest <- predict(tree.fit, test['BMI'], type = "class")

train.error <- mean(tree.predtrain != train$NObeyesdad)
test.error <- mean(tree.predtest != test$NObeyesdad)

print(paste("Misclassification error rate in train = ", train.error))
```

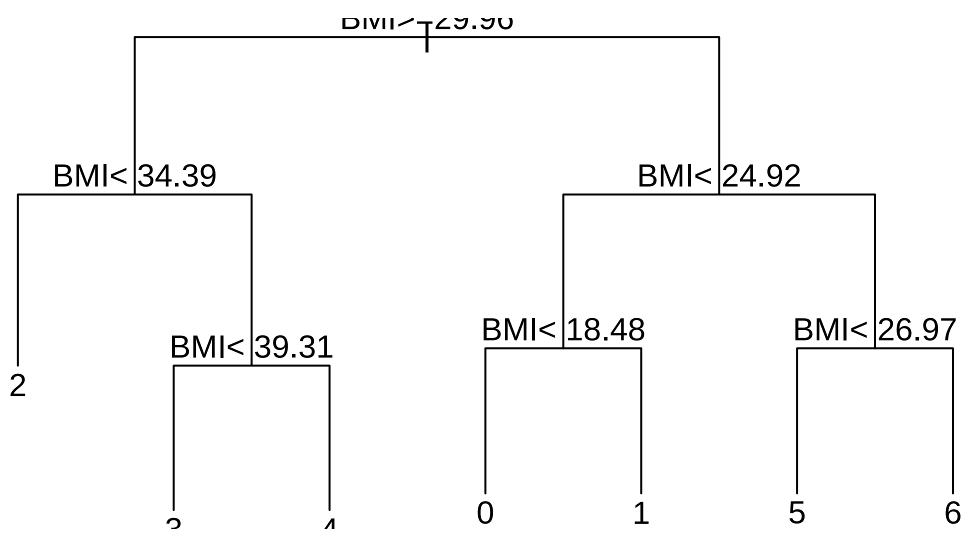
```
## [1] "Misclassification error rate in train =  0.0327868852459016"
```

```r
print(paste("Misclassification error rate in test = ", test.error))
```

```
## [1] "Misclassification error rate in test =  0.0628571428571429"
```

Only with BMI predictor, the model performs pretty good.

```r
plot(tree.fit)
text(tree.fit,pretty=0)
```




```r
#zm()
```

SVM Modeling

The SVM algorithm finds the decision boundary by searching for the hyperplane that has the largest distance (also known as the margin) to the nearest points from each class. The points that are closest to the decision boundary and determine the position and orientation of the hyperplane are called support vectors. The SVM algorithm maximizes the margin by finding the hyperplane that has the largest distance to the nearest points from each class.

In addition, we have also used SVM regularization parameters such as gamma and cost, which controls the balance between maximizing the margin and minimizing the number of support vectors which enhances the model performance to a new query point. The regularization parameter can be used to prevent overfitting.

Moreover we developed 3 types of SVM Models based on predictors relation to the target variable. 1 - Linear SVM: Assuming the relationship between predictors and the target variable to be linear 2- Radial SVM: Assuming the relationship between predictors and the target variable to be non-linear but samples coming from or conforming to a normal distribution. 3- Polynomial SVM: Assuming the relationship between predictors and the target variable to be non-linear

```r
train$CALC <- as.numeric(as.character(train$CALC))

svm.fit <- svm(NObeyesdad ~ ., data = train,
               type="C-classification",
               kernel = "radial", probability=TRUE)
summary(svm.fit)
```

```
##
## Call:
## svm(formula = NObeyesdad ~ ., data = train, type = "C-classification",
##     kernel = "radial", probability = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  986
##
##  ( 203 178 188 177 93 121 26 )
##
##
## Number of Classes:  7
##
## Levels:
##  0 1 2 3 4 5 6
```

```r
set.seed(4)
train$CALC <- as.numeric(as.character(train$CALC))
test$CALC <- as.numeric(as.character(test$CALC))

 svm1 <- svm(NObeyesdad~., data=train,
          type="C-classification",
          kernal="radial", probability=TRUE)

 summary(svm1)
```

```
##
## Call:
## svm(formula = NObeyesdad ~ ., data = train, type = "C-classification",
##     kernal = "radial", probability = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  986
##
##  ( 203 178 188 177 93 121 26 )
##
##
```

```
## Number of Classes:  7
##
## Levels:
##  0 1 2 3 4 5 6
```

```r
train.pred <- predict(svm.fit, train)
test.pred <- predict(svm.fit, test)
train.error <- mean(train.pred != train$NObeyesdad)
test.error <- mean(test.pred != test$NObeyesdad)
print(paste("Train Error Rate (Radial kernel) = ", train.error*100, "%"))
```

```
## [1] "Train Error Rate (Radial kernel) =  5.48549810844893 %"
```

```r
print(paste("Test Error Rate (Radial kernel) = ", test.error*100, "%"))
```

```
## [1] "Test Error Rate (Radial kernel) =  10.8571428571429 %"
```

```r
confusionMatrix(train.pred, as.factor(train$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6
##          0 194   3   0   0   0   0   0
##          1  10 200   2   1   0  20   9
##          2   0   0 255   0   0   1   2
##          3   0   0   6 222   0   0   0
##          4   0   0   0   0 243   0   0
##          5   0  11   1   0   0 189  11
##          6   0   2   0   0   0   8 196
##
## Overall Statistics
##
##                Accuracy : 0.9451
##                  95% CI : (0.9328, 0.9558)
##     No Information Rate : 0.1665
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9359
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9510   0.9259   0.9659   0.9955   1.0000   0.8670
## Specificity            0.9978   0.9693   0.9977   0.9956   1.0000   0.9832
## Pos Pred Value         0.9848   0.8264   0.9884   0.9737   1.0000   0.8915
## Neg Pred Value         0.9928   0.9881   0.9932   0.9993   1.0000   0.9789
## Prevalence             0.1286   0.1362   0.1665   0.1406   0.1532   0.1375
## Detection Rate         0.1223   0.1261   0.1608   0.1400   0.1532   0.1192
## Detection Prevalence   0.1242   0.1526   0.1627   0.1438   0.1532   0.1337
## Balanced Accuracy      0.9744   0.9476   0.9818   0.9956   1.0000   0.9251
##                      Class: 6
## Sensitivity            0.8991
## Specificity            0.9927
```

```
## Pos Pred Value         0.9515
## Neg Pred Value         0.9841
## Prevalence             0.1375
## Detection Rate         0.1236
## Detection Prevalence   0.1299
## Balanced Accuracy      0.9459
```

```
confusionMatrix(test.pred, as.factor(test$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6
##          0 63  3  0  0  0  0  0
##          1  5 57  1  0  0  6  4
##          2  0  1 79  2  0  0  0
##          3  0  0  4 72  1  0  0
##          4  0  0  0  0 80  0  0
##          5  0  8  2  0  0 53  4
##          6  0  2  1  0  0 13 64
##
## Overall Statistics
##
##                Accuracy : 0.8914
##                  95% CI : (0.8616, 0.9167)
##     No Information Rate : 0.1657
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8732
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9265   0.8028   0.9080   0.9730   0.9877   0.7361
## Specificity            0.9934   0.9648   0.9932   0.9889   1.0000   0.9691
## Pos Pred Value         0.9545   0.7808   0.9634   0.9351   1.0000   0.7910
## Neg Pred Value         0.9891   0.9690   0.9819   0.9955   0.9978   0.9585
## Prevalence             0.1295   0.1352   0.1657   0.1410   0.1543   0.1371
## Detection Rate         0.1200   0.1086   0.1505   0.1371   0.1524   0.1010
## Detection Prevalence   0.1257   0.1390   0.1562   0.1467   0.1524   0.1276
## Balanced Accuracy      0.9600   0.8838   0.9506   0.9809   0.9938   0.8526
##                      Class: 6
## Sensitivity            0.8889
## Specificity            0.9647
## Pos Pred Value         0.8000
## Neg Pred Value         0.9820
## Prevalence             0.1371
## Detection Rate         0.1219
## Detection Prevalence   0.1524
## Balanced Accuracy      0.9268
```

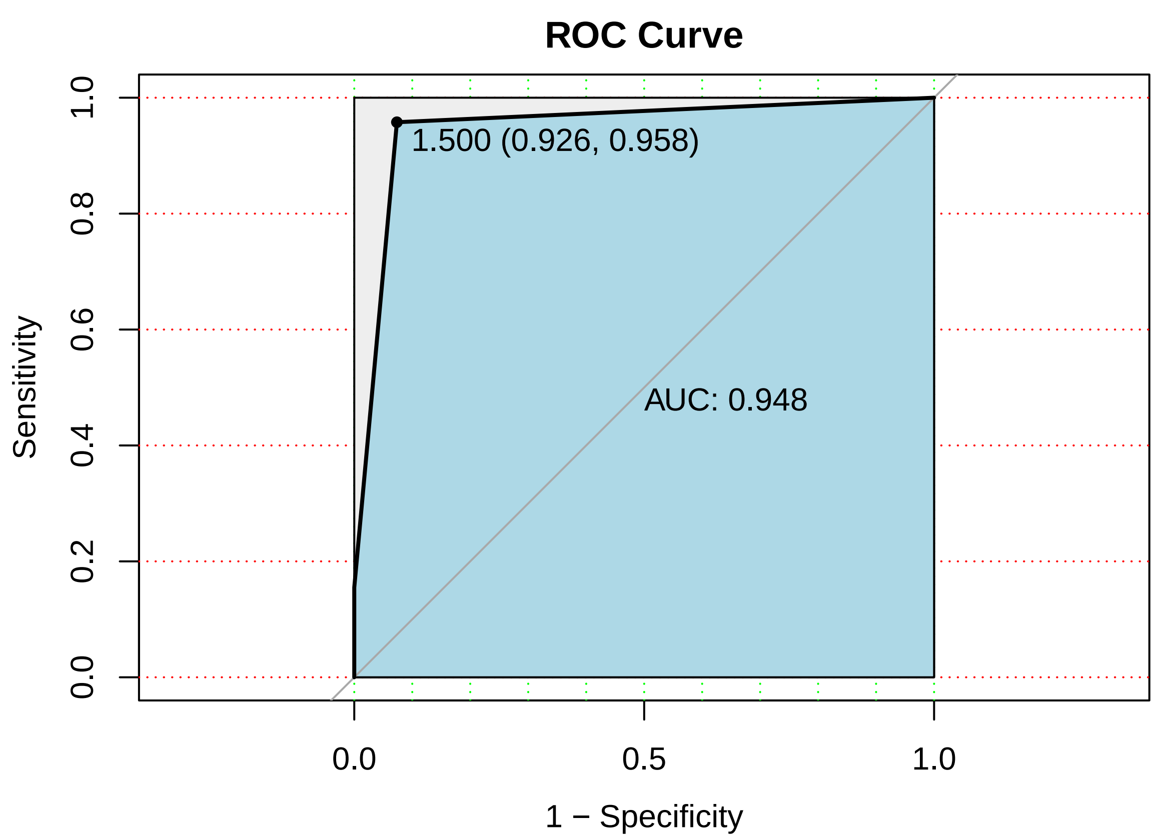
```
p1 <- predict(svm.fit, test, type="prob")
roccurve <- plot(
```

```
    roc(
        response = test$NObeyesdad,
        predictor = as.numeric(p1)
    ),
    legacy.axes = TRUE,
    print.auc=TRUE,
    auc.polygon=TRUE,
    grid=c(0.1, 0.2),
    grid.col=c("green", "red"),
    max.auc.polygon=TRUE,
    auc.polygon.col="lightblue",
    print.thres=TRUE,
    main = "ROC Curve"
)
```

```
## Warning in roc.default(response = test$NObeyesdad, predictor = as.numeric(p1)):
## 'response' has more than two levels. Consider setting 'levels' explicitly or
## using 'multiclass.roc' instead
```

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

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



SVM Radial - data with BMI

```r
train$CALC <- as.numeric(as.character(train$CALC))
test$CALC <- as.numeric(as.character(test$CALC))

svm.fit <- svm(NObeyesdad ~ ., data = train,
               type="C-classification",
               kernel = "radial")

train.pred <- predict(svm.fit, train)
test.pred <- predict(svm.fit, test)
train.error <- mean(train.pred != train$NObeyesdad)
test.error <- mean(test.pred != test$NObeyesdad)
print(paste("Train Error Rate (Radial kernel) = ", train.error*100, "%"))
```

```
## [1] "Train Error Rate (Radial kernel) =  5.48549810844893 %"
```

```r
print(paste("Test Error Rate (Radial kernel) = ", test.error*100, "%"))
```

```
## [1] "Test Error Rate (Radial kernel) =  10.8571428571429 %"
```

```r
confusionMatrix(train.pred, as.factor(train$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6
##          0 194   3   0   0   0   0   0
##          1  10 200   2   1   0  20   9
##          2   0   0 255   0   0   1   2
##          3   0   0   6 222   0   0   0
##          4   0   0   0   0 243   0   0
##          5   0  11   1   0   0 189  11
##          6   0   2   0   0   0   8 196
##
## Overall Statistics
##
##                Accuracy : 0.9451
##                  95% CI : (0.9328, 0.9558)
##     No Information Rate : 0.1665
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9359
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9510   0.9259   0.9659   0.9955   1.0000   0.8670
## Specificity            0.9978   0.9693   0.9977   0.9956   1.0000   0.9832
## Pos Pred Value         0.9848   0.8264   0.9884   0.9737   1.0000   0.8915
## Neg Pred Value         0.9928   0.9881   0.9932   0.9993   1.0000   0.9789
## Prevalence             0.1286   0.1362   0.1665   0.1406   0.1532   0.1375
## Detection Rate         0.1223   0.1261   0.1608   0.1400   0.1532   0.1192
## Detection Prevalence   0.1242   0.1526   0.1627   0.1438   0.1532   0.1337
## Balanced Accuracy      0.9744   0.9476   0.9818   0.9956   1.0000   0.9251
```

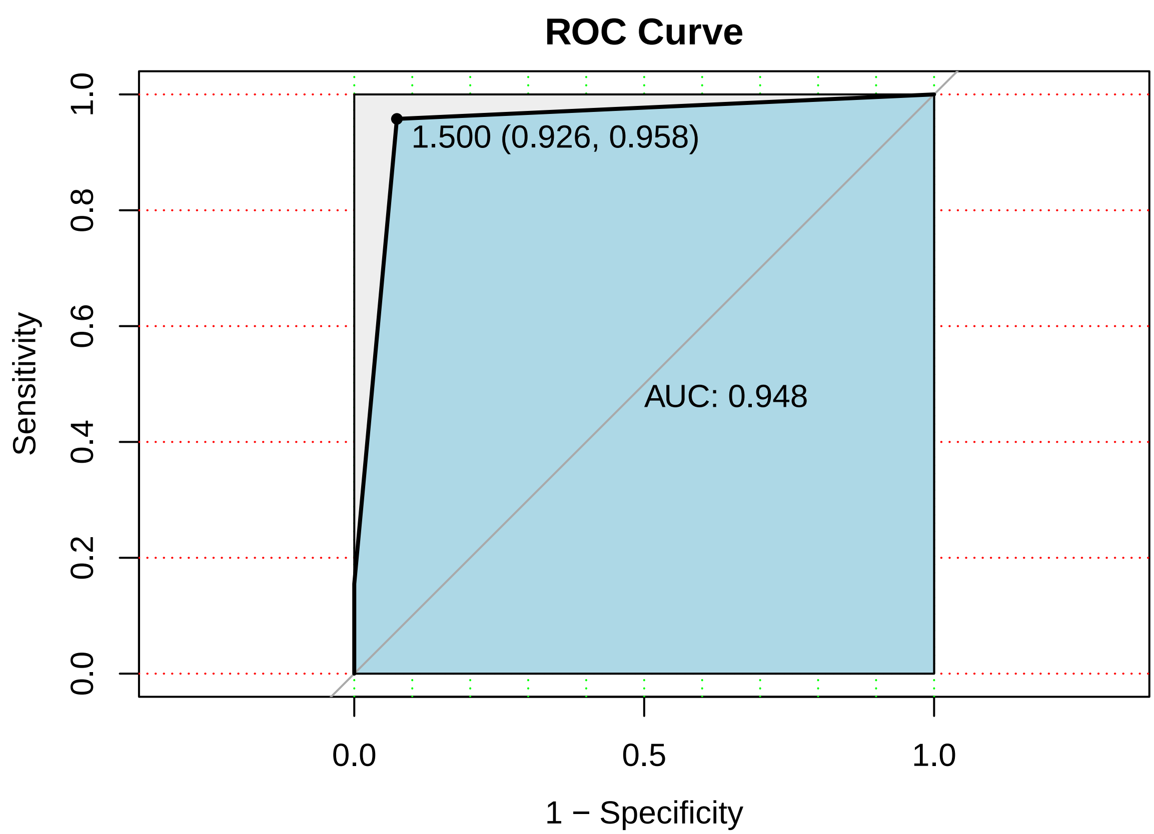
```
##                      Class: 6
## Sensitivity            0.8991
## Specificity            0.9927
## Pos Pred Value         0.9515
## Neg Pred Value         0.9841
## Prevalence             0.1375
## Detection Rate         0.1236
## Detection Prevalence   0.1299
## Balanced Accuracy      0.9459
```

```
confusionMatrix(test.pred, as.factor(test$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6
##          0 63  3  0  0  0  0  0
##          1  5 57  1  0  0  6  4
##          2  0  1 79  2  0  0  0
##          3  0  0  4 72  1  0  0
##          4  0  0  0  0 80  0  0
##          5  0  8  2  0  0 53  4
##          6  0  2  1  0  0 13 64
##
## Overall Statistics
##
##                Accuracy : 0.8914
##                  95% CI : (0.8616, 0.9167)
##     No Information Rate : 0.1657
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8732
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9265   0.8028   0.9080   0.9730   0.9877   0.7361
## Specificity            0.9934   0.9648   0.9932   0.9889   1.0000   0.9691
## Pos Pred Value         0.9545   0.7808   0.9634   0.9351   1.0000   0.7910
## Neg Pred Value         0.9891   0.9690   0.9819   0.9955   0.9978   0.9585
## Prevalence             0.1295   0.1352   0.1657   0.1410   0.1543   0.1371
## Detection Rate         0.1200   0.1086   0.1505   0.1371   0.1524   0.1010
## Detection Prevalence   0.1257   0.1390   0.1562   0.1467   0.1524   0.1276
## Balanced Accuracy      0.9600   0.8838   0.9506   0.9809   0.9938   0.8526
##                      Class: 6
## Sensitivity            0.8889
## Specificity            0.9647
## Pos Pred Value         0.8000
## Neg Pred Value         0.9820
## Prevalence             0.1371
## Detection Rate         0.1219
## Detection Prevalence   0.1524
## Balanced Accuracy      0.9268
```

```
p1 <- predict(svm.fit, test, type="prob")
roccurve <- plot(
    roc(
        response = test$NObeyesdad,
        predictor = as.numeric(p1)
    ),
    legacy.axes = TRUE,
    print.auc=TRUE,
    auc.polygon=TRUE,
    grid=c(0.1, 0.2),
    grid.col=c("green", "red"),
    max.auc.polygon=TRUE,
    auc.polygon.col="lightblue",
    print.thres=TRUE,
    main = "ROC Curve"
)
```

```
## Warning in roc.default(response = test$NObeyesdad, predictor = as.numeric(p1)):
## 'response' has more than two levels. Consider setting 'levels' explicitly or
## using 'multiclass.roc' instead

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

## Setting direction: controls < cases
```



SVM Linear - data with BMI

```
svm.fit <- svm(NObeyesdad ~ ., data = train,
               type="C-classification",
               kernel = "linear")

train.pred <- predict(svm.fit, train)
test.pred <- predict(svm.fit, test)
train.error <- mean(train.pred != train$NObeyesdad)
test.error <- mean(test.pred != test$NObeyesdad)
print(paste("Train Error Rate (Linear kernel) = ", train.error*100, "%"))
```

```
## [1] "Train Error Rate (Linear kernel) =  2.52206809583859 %"
```

```
print(paste("Test Error Rate (Linear kernel) = ", test.error*100, "%"))
```

```
## [1] "Test Error Rate (Linear kernel) =  3.42857142857143 %"
```

```
confusionMatrix(train.pred, as.factor(train$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6
##          0 204   7   0   0   0   0   0
##          1   0 202   0   0   0   2   0
##          2   0   0 259   1   0   0   1
##          3   0   0   1 222   0   0   0
##          4   0   0   0   0 243   0   0
##          5   0   7   0   0   0 211  12
##          6   0   0   4   0   0   5 205
##
## Overall Statistics
##
##                Accuracy : 0.9748
##                  95% CI : (0.9658, 0.9819)
##     No Information Rate : 0.1665
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9705
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            1.0000   0.9352   0.9811   0.9955   1.0000   0.9679
## Specificity            0.9949   0.9985   0.9985   0.9993   1.0000   0.9861
## Pos Pred Value         0.9668   0.9902   0.9923   0.9955   1.0000   0.9174
## Neg Pred Value         1.0000   0.9899   0.9962   0.9993   1.0000   0.9948
## Prevalence             0.1286   0.1362   0.1665   0.1406   0.1532   0.1375
## Detection Rate         0.1286   0.1274   0.1633   0.1400   0.1532   0.1330
## Detection Prevalence   0.1330   0.1286   0.1646   0.1406   0.1532   0.1450
## Balanced Accuracy      0.9975   0.9669   0.9898   0.9974   1.0000   0.9770
##                      Class: 6
## Sensitivity            0.9404
```

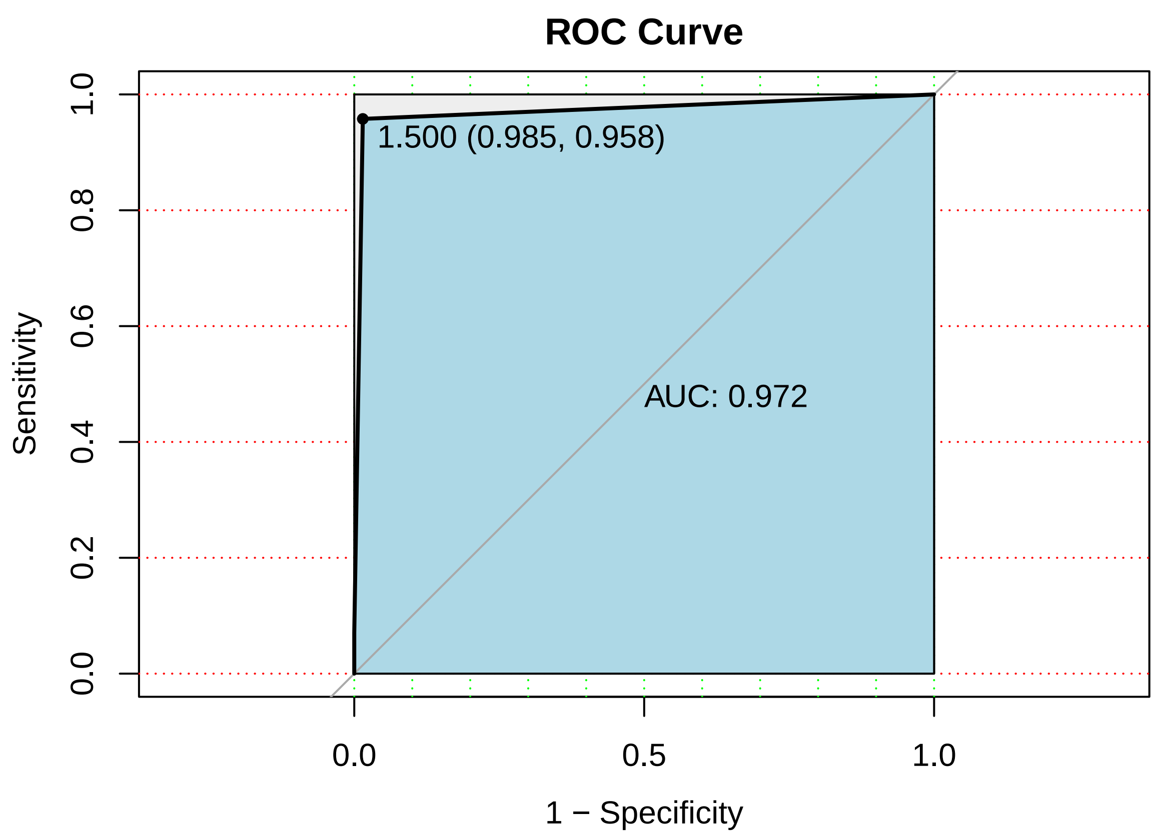
```
## Specificity            0.9934
## Pos Pred Value         0.9579
## Neg Pred Value         0.9905
## Prevalence             0.1375
## Detection Rate         0.1293
## Detection Prevalence   0.1349
## Balanced Accuracy      0.9669
```

```
confusionMatrix(test.pred, as.factor(test$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6
##          0 67  3  0  0  0  0  0
##          1  1 63  0  0  0  0  0
##          2  0  0 85  0  0  0  1
##          3  0  0  2 74  0  0  0
##          4  0  0  0  0 81  0  0
##          5  0  5  0  0  0 68  2
##          6  0  0  0  0  0  4 69
##
## Overall Statistics
##
##                Accuracy : 0.9657
##                  95% CI : (0.9464, 0.9796)
##     No Information Rate : 0.1657
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.96
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9853   0.8873   0.9770   1.0000   1.0000   0.9444
## Specificity            0.9934   0.9978   0.9977   0.9956   1.0000   0.9845
## Pos Pred Value         0.9571   0.9844   0.9884   0.9737   1.0000   0.9067
## Neg Pred Value         0.9978   0.9826   0.9954   1.0000   1.0000   0.9911
## Prevalence             0.1295   0.1352   0.1657   0.1410   0.1543   0.1371
## Detection Rate         0.1276   0.1200   0.1619   0.1410   0.1543   0.1295
## Detection Prevalence   0.1333   0.1219   0.1638   0.1448   0.1543   0.1429
## Balanced Accuracy      0.9894   0.9426   0.9874   0.9978   1.0000   0.9645
##                      Class: 6
## Sensitivity            0.9583
## Specificity            0.9912
## Pos Pred Value         0.9452
## Neg Pred Value         0.9934
## Prevalence             0.1371
## Detection Rate         0.1314
## Detection Prevalence   0.1390
## Balanced Accuracy      0.9748
```

```
p1 <- predict(svm.fit, test, type="prob")
roccurve <- plot(
    roc(
        response = test$NObeyesdad,
        predictor = as.numeric(p1)
    ),
    legacy.axes = TRUE,
    print.auc=TRUE,
    auc.polygon=TRUE,
    grid=c(0.1, 0.2),
    grid.col=c("green", "red"),
    max.auc.polygon=TRUE,
    auc.polygon.col="lightblue",
    print.thres=TRUE,
    main = "ROC Curve"
)
```

```
## Warning in roc.default(response = test$NObeyesdad, predictor = as.numeric(p1)):
## 'response' has more than two levels. Consider setting 'levels' explicitly or
## using 'multiclass.roc' instead

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

## Setting direction: controls < cases
```



SVM Polynomial - data with BMI

```
svm.fit <- svm(NObeyesdad ~ ., data = train,
               type="C-classification",
               kernel = "polynomial", degree=2)

train.pred <- predict(svm.fit, train)
test.pred <- predict(svm.fit, test)
train.error <- mean(train.pred != train$NObeyesdad)
test.error <- mean(test.pred != test$NObeyesdad)
print(paste("Train Error Rate (Radial kernel) = ", train.error*100, "%"))
```

```
## [1] "Train Error Rate (Radial kernel) =  11.7276166456494 %"
```

```
print(paste("Test Error Rate (Radial kernel) = ", test.error*100, "%"))
```

```
## [1] "Test Error Rate (Radial kernel) =  18.2857142857143 %"
```

```
confusionMatrix(train.pred, as.factor(train$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6
##          0 200   3   0   0   0   0   0
##          1   4 176   2   3   2  12   7
##          2   0   8 242   0   0  24  31
##          3   0   2  14 218   0   0   0
##          4   0   0   0   1 241   0   0
##          5   0  15   5   0   0 165  22
##          6   0  12   1   1   0  17 158
##
## Overall Statistics
##
##                Accuracy : 0.8827
##                  95% CI : (0.8659, 0.8982)
##     No Information Rate : 0.1665
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8629
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9804   0.8148   0.9167   0.9776   0.9918   0.7569
## Specificity            0.9978   0.9781   0.9523   0.9883   0.9993   0.9693
## Pos Pred Value         0.9852   0.8544   0.7934   0.9316   0.9959   0.7971
## Neg Pred Value         0.9971   0.9710   0.9828   0.9963   0.9985   0.9616
## Prevalence             0.1286   0.1362   0.1665   0.1406   0.1532   0.1375
## Detection Rate         0.1261   0.1110   0.1526   0.1375   0.1520   0.1040
## Detection Prevalence   0.1280   0.1299   0.1923   0.1475   0.1526   0.1305
## Balanced Accuracy      0.9891   0.8965   0.9345   0.9829   0.9955   0.8631
##                      Class: 6
## Sensitivity           0.72477
```

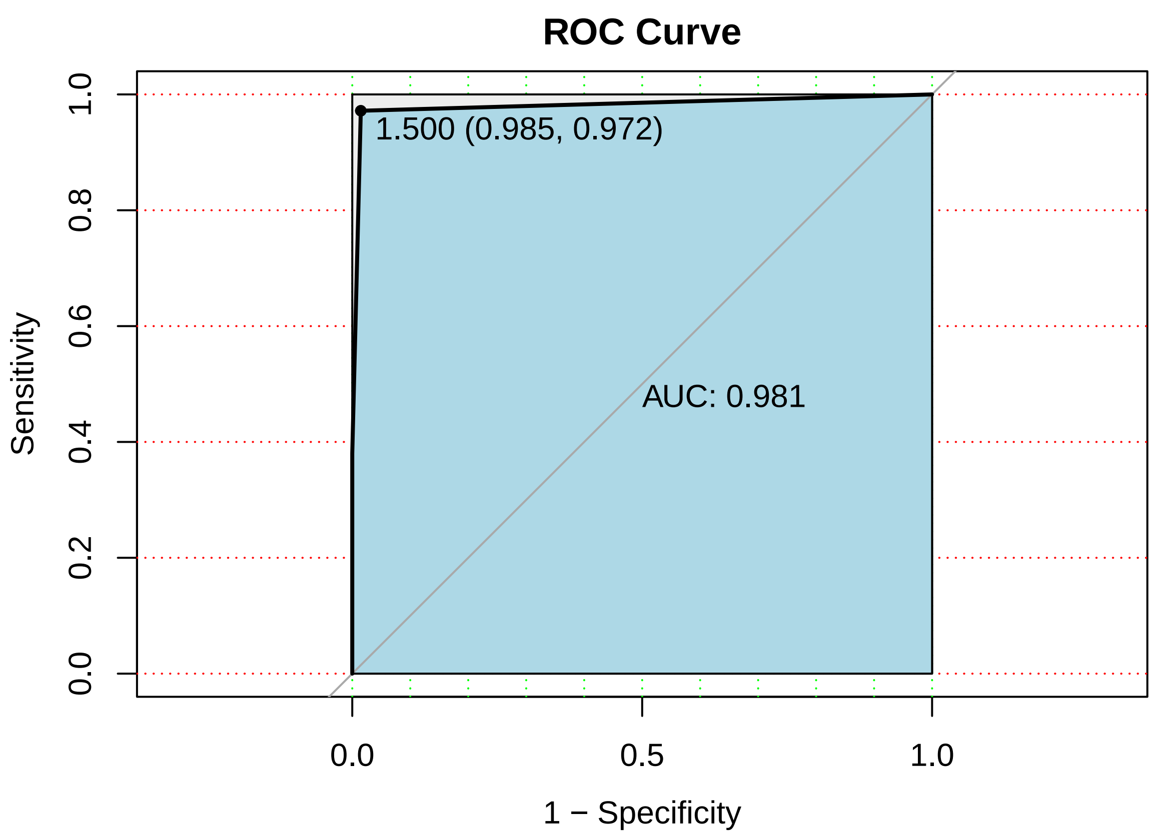
```
## Specificity           0.97734
## Pos Pred Value        0.83598
## Neg Pred Value        0.95705
## Prevalence            0.13745
## Detection Rate        0.09962
## Detection Prevalence  0.11917
## Balanced Accuracy     0.85105
```

```
confusionMatrix(test.pred, as.factor(test$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6
##          0 67  2  0  0  1  0  0
##          1  1 42  3  2  0  4  3
##          2  0  8 74  1  0 16  9
##          3  0  1  6 71  0  0  0
##          4  0  2  0  0 80  0  0
##          5  0  8  1  0  0 39  4
##          6  0  8  3  0  0 13 56
##
## Overall Statistics
##
##                Accuracy : 0.8171
##                  95% CI : (0.7814, 0.8493)
##     No Information Rate : 0.1657
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7861
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9853   0.5915   0.8506   0.9595   0.9877  0.54167
## Specificity            0.9934   0.9714   0.9224   0.9845   0.9955  0.97130
## Pos Pred Value         0.9571   0.7636   0.6852   0.9103   0.9756  0.75000
## Neg Pred Value         0.9978   0.9383   0.9688   0.9933   0.9977  0.93023
## Prevalence             0.1295   0.1352   0.1657   0.1410   0.1543  0.13714
## Detection Rate         0.1276   0.0800   0.1410   0.1352   0.1524  0.07429
## Detection Prevalence   0.1333   0.1048   0.2057   0.1486   0.1562  0.09905
## Balanced Accuracy      0.9894   0.7815   0.8865   0.9720   0.9916  0.75648
##                      Class: 6
## Sensitivity            0.7778
## Specificity            0.9470
## Pos Pred Value         0.7000
## Neg Pred Value         0.9640
## Prevalence             0.1371
## Detection Rate         0.1067
## Detection Prevalence   0.1524
## Balanced Accuracy      0.8624
```

```
p1 <- predict(svm.fit, test, type="prob")
roccurve <- plot(
    roc(
        response = test$NObeyesdad,
        predictor = as.numeric(p1)
    ),
    legacy.axes = TRUE,
    print.auc=TRUE,
    auc.polygon=TRUE,
    grid=c(0.1, 0.2),
    grid.col=c("green", "red"),
    max.auc.polygon=TRUE,
    auc.polygon.col="lightblue",
    print.thres=TRUE,
    main = "ROC Curve"
)
```

```
## Warning in roc.default(response = test$NObeyesdad, predictor = as.numeric(p1)):
## 'response' has more than two levels. Consider setting 'levels' explicitly or
## using 'multiclass.roc' instead

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

## Setting direction: controls < cases
```



Tuning SVM - Radial

Tuning the hyperparameters of the SVM Model i.e. Gamma & Cost to get the best model parameters

```
# Create data copy
tune_data <- cbind(train)

# SVM Tuning
tune_data$NObeyesdad <- as.factor(tune_data$NObeyesdad)

# Tuning took 4 hours to run. So, commenting for knitting the markdown file.
#tune.svm <- tune(svm, NObeyesdad~., data=tune_data,
                 #kernel="radial",type="C-classification",
                 #ranges=list(cost=2^(-3:2),gamma=2^(-25:1)),
                 #tunecontrol = tune.control(nrepeat = 5,
                  #                          sampling = "cross",
                  #                          cross = 5))
```

```
#Best model parameters

#print(paste("Model Parameters: best cost value:", tune.svm$best.parameters[1]))
#print(paste("Model Parameters: best gamma value:", tune.svm$best.parameters[2]))
```

Gamma and Cost value that resulted in the lowest cross validation error are 0.03125 and 4 respectively.

Fitting SVM model using tuned hyperparameters.

```
# SVM Radial - Tuned params
svm.bestparam <- svm(NObeyesdad~.,
                     data=tune_data,
                     type="C-classification",
                     kernal="radial",
                     gamma=0.03125, #tune.svm$best.parameters[2],
                     cost=4 #as.numeric(tune.svm$best.parameters[1])
                     )

 summary(svm.bestparam)
```

```
##
## Call:
## svm(formula = NObeyesdad ~ ., data = tune_data, type = "C-classification",
##     kernal = "radial", gamma = 0.03125, cost = 4)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  4
##
## Number of Support Vectors:  834
##
##  ( 191 168 158 143 66 94 14 )
##
##
## Number of Classes:  7
##
## Levels:
```

```
## 0 1 2 3 4 5 6
```

```r
#training prediction
svm.bestparam.predtrain <- predict(svm.bestparam, train)
xtab.train <- table(train$NObeyesdad, svm.bestparam.predtrain)
print("Confussion matrix for train data")
```

```
## [1] "Confussion matrix for train data"
```

```r
xtab.train
```

```
##    svm.bestparam.predtrain
##       0   1   2   3   4   5   6
##   0 204   0   0   0   0   0   0
##   1   2 210   0   0   0   3   1
##   2   0   0 261   1   0   0   2
##   3   0   0   0 223   0   0   0
##   4   0   0   0   0 243   0   0
##   5   0   2   0   0   0 215   1
##   6   0   0   1   0   0   2 215
```

```r
#test prediction
svm.bestparam.predtest <- predict(svm.bestparam,test)
xtab.test <- table(test$NObeyesdad, svm.bestparam.predtest)
print("Confusion matrix for test data")
```

```
## [1] "Confusion matrix for test data"
```

```r
xtab.test
```

```
##    svm.bestparam.predtest
##      0  1  2  3  4  5  6
##   0 65  3  0  0  0  0  0
##   1  4 59  1  0  0  6  1
##   2  0  0 86  1  0  0  0
##   3  0  0  2 72  0  0  0
##   4  0  0  0  1 80  0  0
##   5  0  3  0  0  0 61  8
##   6  0  1  0  0  0  0 71
```

```r
svm.bestparam.train.error <- mean(svm.bestparam.predtrain != train$NObeyesdad)
svm.bestparam.test.error <- mean(svm.bestparam.predtest != test$NObeyesdad)

print(paste("Misclassification error rate in train = ", round(svm.bestparam.train.error,4),"%"))
```

```
## [1] "Misclassification error rate in train =  0.0095 %"
```

```r
print(paste("Misclassification error rate in test = ", round(svm.bestparam.test.error,4),"%"))
```

```
## [1] "Misclassification error rate in test =  0.059 %"
```

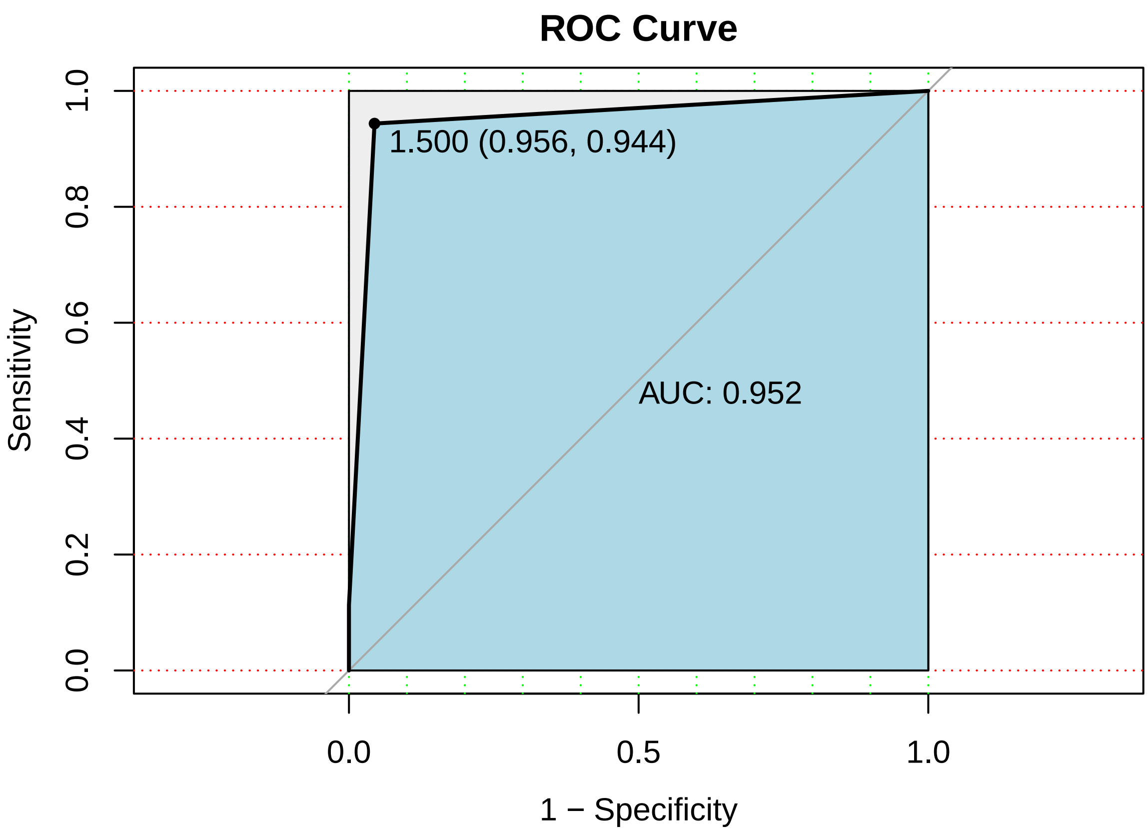
```r
p1 <- predict(svm.bestparam, test, type="prob")
roccurve <- plot(
    roc(
        response = test$NObeyesdad,
        predictor = as.numeric(p1)
    ),
    legacy.axes = TRUE,
    print.auc=TRUE,
```

```
    auc.polygon=TRUE,
    grid=c(0.1, 0.2),
    grid.col=c("green", "red"),
    max.auc.polygon=TRUE,
    auc.polygon.col="lightblue",
    print.thres=TRUE,
    main = "ROC Curve"
)
```

```
## Warning in roc.default(response = test$NObeyesdad, predictor = as.numeric(p1)):
## 'response' has more than two levels. Consider setting 'levels' explicitly or
## using 'multiclass.roc' instead
```

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

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



Bagging

Bagging is mainly focused on reducing the variance and minimizing the overfitting in the data. By using Bagging method we can predict the predicted errors in the dataset with the confusion matrix, which explains how variance can be manipulated by changing the original dataset. Initially we have considered 15 bootstrap samples i.e, 15 samples and predicted the OOB error.With BMI included in the dataset the predictor variables are highly correlated to the response variable, so our results show the that we have a miscalssification error rate of 2.59% which resembles the accuracy of data.

```r
#bagging
set.seed(1)
options(warn=-1)
#train1<-train

#train$NObeyesdad <- mapvalues(train$NObeyesdad,
#                            from=c(0, 1, 2, 3, 4, 5, 6),
#          to=c("Insufficient_Weight","Normal_Weight","Obesity_Type_I", #"Obesity_Type_II", "Obesity_Ty

gbag <- bagging(as.factor(NObeyesdad) ~ .,
                data = train,
                coob=T,
                nbag=15)
print(gbag)
```

```
##
## Bagging classification trees with 15 bootstrap replications
##
## Call: bagging.data.frame(formula = as.factor(NObeyesdad) ~ ., data = train,
##     coob = T, nbag = 15)
##
## Out-of-bag estimate of misclassification error:  0.0259
```

```r
#training prediction
bag.predtrain <- predict(gbag, train)
xtab.train <- table(train$NObeyesdad, bag.predtrain)
print("Confussion matrix for train data")
```

```
## [1] "Confussion matrix for train data"
```

```r
xtab.train
```

```
##    bag.predtrain
##       0   1   2   3   4   5   6
##   0 204   0   0   0   0   0   0
##   1   0 216   0   0   0   0   0
##   2   0   0 264   0   0   0   0
##   3   0   0   0 221   2   0   0
##   4   0   0   0   0 243   0   0
##   5   0   1   0   0   0 216   1
##   6   0   0   0   0   0   0 218
```

```r
#test prediction
test1<-test
#$NObeyesdad <- mapvalues(test1$NObeyesdad,
#                            from=c(0, 1, 2, 3, 4, 5, 6),
#          to=c("Insufficient_Weight","Normal_Weight","Obesity_Type_I", #"Obesity_Type_II", "Obesity_Ty

bag.predtest <- predict(gbag, test1)
xtab.test <- table(test$NObeyesdad, bag.predtest)
print("Confussion matrix for test data")
```

```
## [1] "Confussion matrix for test data"
```

```r
xtab.test
```

```
##    bag.predtest
```

```
##      0  1  2  3  4  5  6
##   0 66  2  0  0  0  0  0
##   1  1 70  0  0  0  0  0
##   2  0  0 80  5  1  0  1
##   3  0  0  2 72  0  0  0
##   4  0  0  0  0 81  0  0
##   5  0  1  0  0  0 67  4
##   6  0  0  0  0  0  2 70
```

```r
bag.train.error <- mean(bag.predtrain != train$NObeyesdad)
bag.test.error <- mean(bag.predtest != test1$NObeyesdad)

print(paste("Misclassification error rate in train = ", bag.train.error,"%"))
```

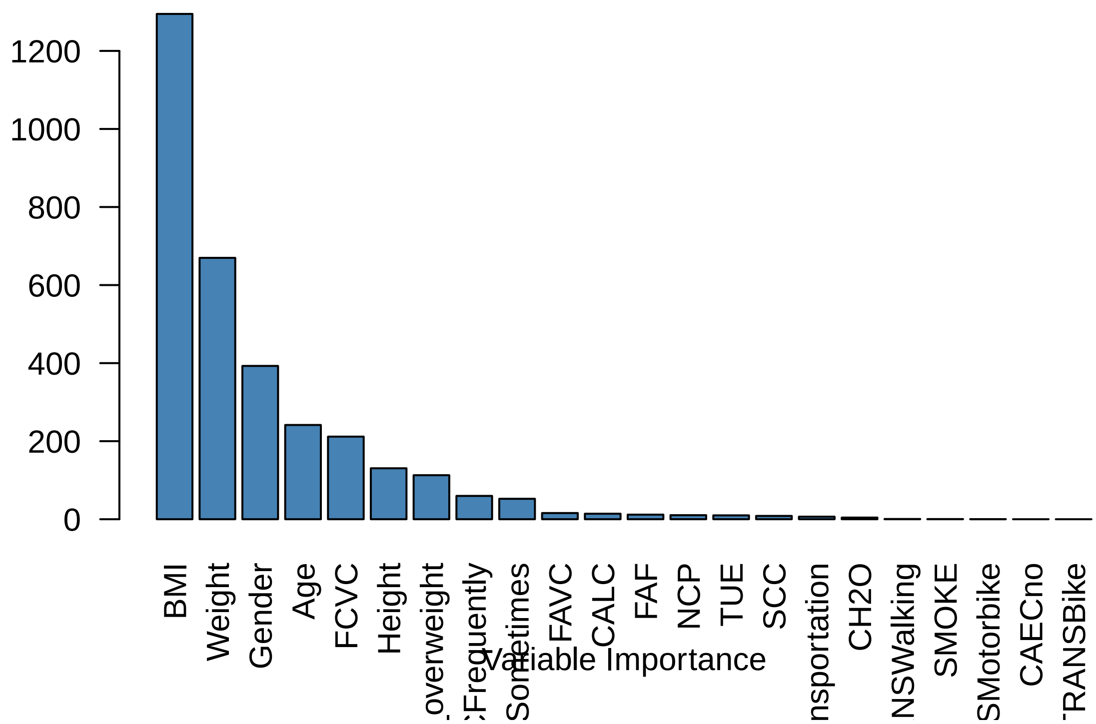
```
## [1] "Misclassification error rate in train =  0.00252206809583859 %"
```

```r
print(paste("Misclassification error rate in test = ", bag.test.error,"%"))
```

```
## [1] "Misclassification error rate in test =  0.0361904761904762 %"
```

Another importance of Bagging is to find the Important predictors thats affecting the increase in variance, such as in this case if we look at below graph, we can see BMI, weight and Gender are most important predictors.

```r
#calculate variable importance
varimp.data <- subset(train, select=-c(NObeyesdad))
VI <- data.frame(var=names(varimp.data), imp=varImp(gbag))

#sort variable importance descending
VI_plot <- VI[order(VI$Overall, decreasing=TRUE),]

#visualize variable importance with horizontal bar plot
barplot(VI_plot$Overall,
        names.arg=rownames(VI_plot),
        horiz=FALSE,
        col='steelblue',
        xlab='Variable Importance',
        las=2,
        srt=45)
```

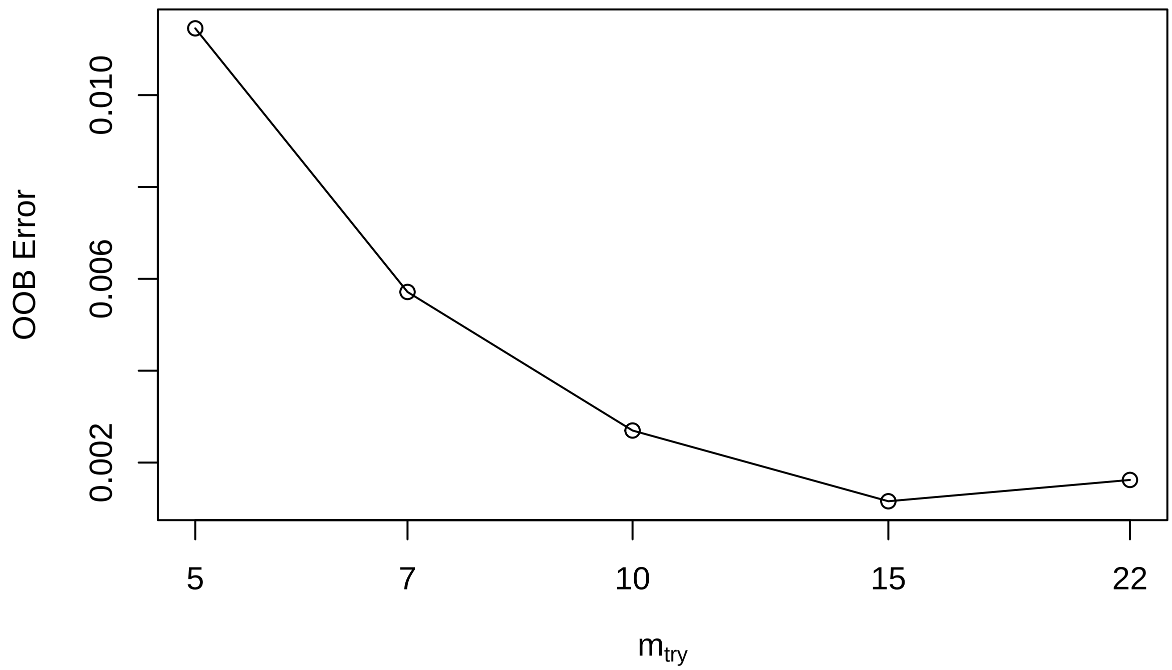


Random forest

Random Forest Modelling is done to show an improvement in Bagging or boosting model. Random Forest Modelling with BMI shows how correlated are our predictors with response variable and with the outputs obtained we can see that OOB error is 2.3 % with mtry=15 which is obtained by tuning RF. From the VarIMpplot we can see that the variance is mostly affect by three top predictors BMI, Gender, weight.

```r
set.seed(1)
bestmtry <-tuneRF(sapply(train, as.numeric),sapply(train$NObeyesdad, as.numeric),improve = 0.01, stepFa
```

```
## mtry = 7  OOB error = 0.005714271
## Searching left ...
## mtry = 5     OOB error = 0.01145374
## -1.004409 0.01
## Searching right ...
## mtry = 10    OOB error = 0.002697978
## 0.5278527 0.01
## mtry = 15    OOB error = 0.001158514
## 0.570599 0.01
## mtry = 22    OOB error = 0.001622058
## -0.4001193 0.01
```




```
print(bestmtry)
```

```
##    mtry    OOBError
## 5     5 0.011453738
## 7     7 0.005714271
## 10   10 0.002697978
## 15   15 0.001158514
## 22   22 0.001622058
```

```
set.seed(545)

rf.fit <- randomForest(
  NObeyesdad ~ .,
  data = sapply(train, as.numeric),
  importance = TRUE,
  mtry = 15,
  ntree = 5000
)

rf.pred <- predict(rf.fit, train, type='class')
rf.predtest <- predict(rf.fit, test, type='class')

print("--- Training Error - Random Forest ---")
```

```
## [1] "--- Training Error - Random Forest ---"
```

```
print(mean(round(rf.pred) != train$NObeyesdad))
```

```
## [1] 0.01008827
```

```
print("--- Test Error - Random Forest ---")
```

```
## [1] "--- Test Error - Random Forest ---"
```

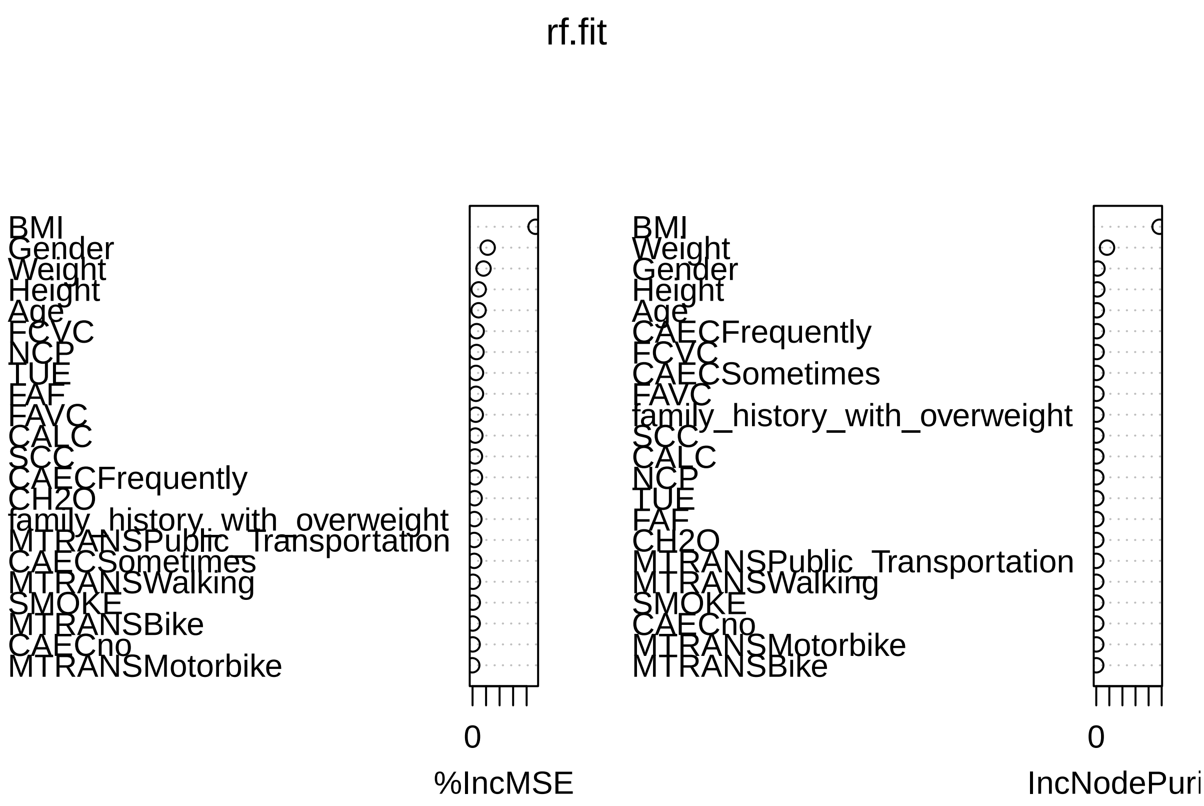
```
print(mean(round(rf.predtest) != test$NObeyesdad))
```

```
## [1] 0.05714286
```

```
rf.fit
```

```
##
## Call:
##  randomForest(formula = NObeyesdad ~ ., data = sapply(train, as.numeric),      importance = TRUE, mt
##                Type of random forest: regression
##                      Number of trees: 5000
## No. of variables tried at each split: 15
##
##           Mean of squared residuals: 0.08699868
##                     % Var explained: 97.72
```

```
varImpPlot(rf.fit)
```




```
importance(rf.fit)
```

```
##                                    %IncMSE IncNodePurity
```

```
## Gender                          111.7678143  9.511443e+01
## Age                              44.6605454  4.266393e+01
## Height                           45.3892503  8.139974e+01
## Weight                           81.1968426  8.248905e+02
## family_history_with_overweight  13.8267235  1.118870e+01
## FAVC                             23.7457245  1.345473e+01
## FCVC                             30.6231534  3.038406e+01
## NCP                              29.2454056  4.366582e+00
## SMOKE                             1.8933613  4.713621e-01
## CH2O                             15.3995044  1.939242e+00
## SCC                              17.8457483  8.074145e+00
## FAF                              25.2447498  3.286274e+00
## TUE                              25.4984518  3.755332e+00
## CALC                             20.6403180  6.189276e+00
## CAECFrequently                   17.7535605  4.211299e+01
## CAECno                            0.7220675  2.391157e-01
## CAECSometimes                    12.2291714  1.351817e+01
## MTRANSBike                        1.1533316  3.489652e-02
## MTRANSMotorbike                  -2.2414953  1.673223e-01
## MTRANSPublic_Transportation      12.2805411  1.609130e+00
## MTRANSWalking                     2.9343287  6.848289e-01
## BMI                             466.1170471  4.837369e+03
```

We can notice that we have improved our model from bagging by using random forest model which reduced the OOB error from 2.59% to 2.28%.

Stacking (Decision Tree + SVM) –> Decision Tree –> Output

Stacking, also known as a stacked generalization, is an ensemble learning technique that combines the predictions of multiple models to make a more accurate prediction. It involves training a learning algorithm to combine the predictions of several other learning algorithms. In stacking, the base models are trained on the original training dataset, and the outputs of the base models (predictions) are used as input to a higher level or meta-model, which is trained to make a final prediction. The decision tree and support vector machine (linear kernel) were used as base models in this project. The main idea behind stacking is that the base models may have different strengths and weaknesses, and by combining their predictions, we can obtain a more accurate overall prediction. Stacking can be a powerful technique, but it can also be computationally expensive since it requires training multiple models. It is often used in competitions where the goal is to achieve the highest possible prediction accuracy.

There are several variations of stacking, including: • Homogeneous stacking: In this approach, all base models are the same type. • Heterogeneous stacking: In this approach, the base models are different types. • Single-level stacking: In this approach, only one meta-model combines the base models' predictions. • Multi-level stacking: In this approach, multiple levels of meta-models are used to combine the predictions of the base models. Here, single-level stacking with heterogenous base models was tried.

```r
# with BMI and weight
train <- data.final[indices,]
test <- data.final[-indices,]
train <- transform(train, BMI=Weight/(Height^2))
test <- transform(test, BMI=Weight/(Height^2))

# Decision Tree
tree.fit <- rpart(NObeyesdad ~ . , data = train)
tree.predtrain <- predict(tree.fit, train, type = "class")
tree.predtest <- predict(tree.fit, test, type = "class")
```

```r
train$CALC <- as.numeric(as.character(train$CALC))
test$CALC <- as.numeric(as.character(test$CALC))

# SVM
svm.fit <- svm(NObeyesdad ~ ., data = train,
               type="C-classification",
               kernel = "linear")

svm.train.pred <- predict(svm.fit, train)
svm.test.pred <- predict(svm.fit, test)


new_df <- data.frame(svm.train.pred, tree.predtrain, train$NObeyesdad)
entree.fit <- rpart(train.NObeyesdad ~ . , data = new_df)

entree.pred <- predict(entree.fit, new_df, type = "class")
entree.error <- mean(entree.pred != new_df$train.NObeyesdad)

print(paste("Ensemble tree error (train) = ", entree.error))
```

```
## [1] "Ensemble tree error (train) =  0.0170239596469105"
```

```r
confusionMatrix(entree.pred, as.factor(train$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6
##          0 201   0   0   0   0   0   0
##          1   3 216   0   0   0   7   0
##          2   0   0 259   1   0   0   1
##          3   0   0   1 222   0   0   0
##          4   0   0   0   0 243   0   0
##          5   0   0   0   0   0 205   4
##          6   0   0   4   0   0   6 213
##
## Overall Statistics
##
##                Accuracy : 0.983
##                  95% CI : (0.9753, 0.9888)
##     No Information Rate : 0.1665
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9801
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9853   1.0000   0.9811   0.9955   1.0000   0.9404
## Specificity            1.0000   0.9927   0.9985   0.9993   1.0000   0.9971
## Pos Pred Value         1.0000   0.9558   0.9923   0.9955   1.0000   0.9809
## Neg Pred Value         0.9978   1.0000   0.9962   0.9993   1.0000   0.9906
```

```
## Prevalence             0.1286   0.1362   0.1665   0.1406   0.1532   0.1375
## Detection Rate         0.1267   0.1362   0.1633   0.1400   0.1532   0.1293
## Detection Prevalence   0.1267   0.1425   0.1646   0.1406   0.1532   0.1318
## Balanced Accuracy      0.9926   0.9964   0.9898   0.9974   1.0000   0.9687
##                      Class: 6
## Sensitivity            0.9771
## Specificity            0.9927
## Pos Pred Value         0.9552
## Neg Pred Value         0.9963
## Prevalence             0.1375
## Detection Rate         0.1343
## Detection Prevalence   0.1406
## Balanced Accuracy      0.9849
```

```
new_df <- data.frame(svm.test.pred, tree.predtest, test$NObeyesdad)
entree.fit <- rpart(test.NObeyesdad ~ . , data = new_df)

entree.pred <- predict(entree.fit, new_df, type = "class")
entree.error <- mean(entree.pred != new_df$test.NObeyesdad)

print(paste("Ensemble tree error (test) = ", entree.error))
```

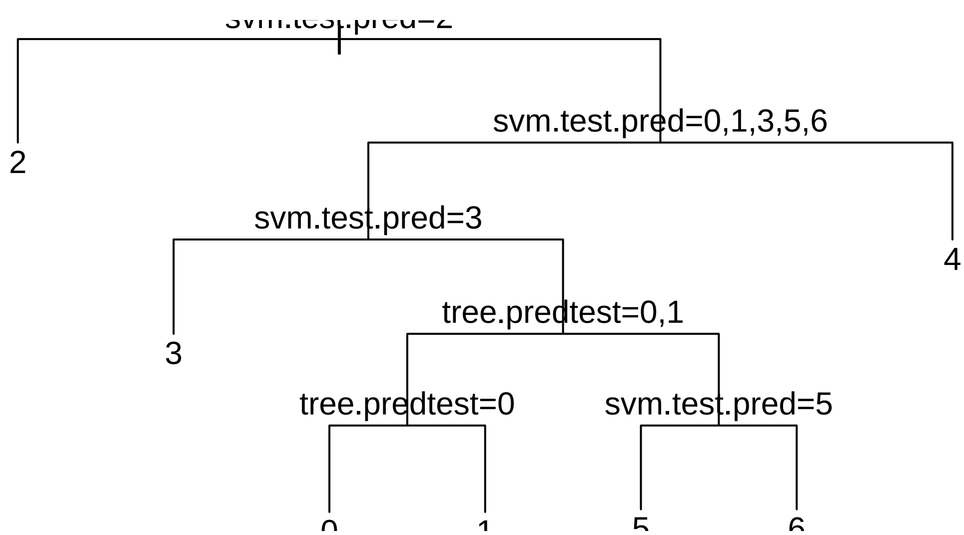
```
## [1] "Ensemble tree error (test) =  0.0247619047619048"
```

```
confusionMatrix(entree.pred, as.factor(test$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6
##          0 66  0  0  0  0  0  0
##          1  2 71  0  0  0  2  0
##          2  0  0 85  0  0  0  1
##          3  0  0  2 74  0  0  0
##          4  0  0  0  0 81  0  0
##          5  0  0  0  0  0 66  2
##          6  0  0  0  0  0  4 69
##
## Overall Statistics
##
##                Accuracy : 0.9752
##                  95% CI : (0.958, 0.9868)
##     No Information Rate : 0.1657
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9711
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9706   1.0000   0.9770   1.0000   1.0000   0.9167
## Specificity            1.0000   0.9912   0.9977   0.9956   1.0000   0.9956
## Pos Pred Value         1.0000   0.9467   0.9884   0.9737   1.0000   0.9706
```

```
## Neg Pred Value          0.9956   1.0000   0.9954   1.0000   1.0000   0.9869
## Prevalence              0.1295   0.1352   0.1657   0.1410   0.1543   0.1371
## Detection Rate          0.1257   0.1352   0.1619   0.1410   0.1543   0.1257
## Detection Prevalence    0.1257   0.1429   0.1638   0.1448   0.1543   0.1295
## Balanced Accuracy       0.9853   0.9956   0.9874   0.9978   1.0000   0.9561
##                       Class: 6
## Sensitivity             0.9583
## Specificity             0.9912
## Pos Pred Value          0.9452
## Neg Pred Value          0.9934
## Prevalence              0.1371
## Detection Rate          0.1314
## Detection Prevalence    0.1390
## Balanced Accuracy       0.9748
```

```r
plot(entree.fit)
text(entree.fit, pretty =0)
```



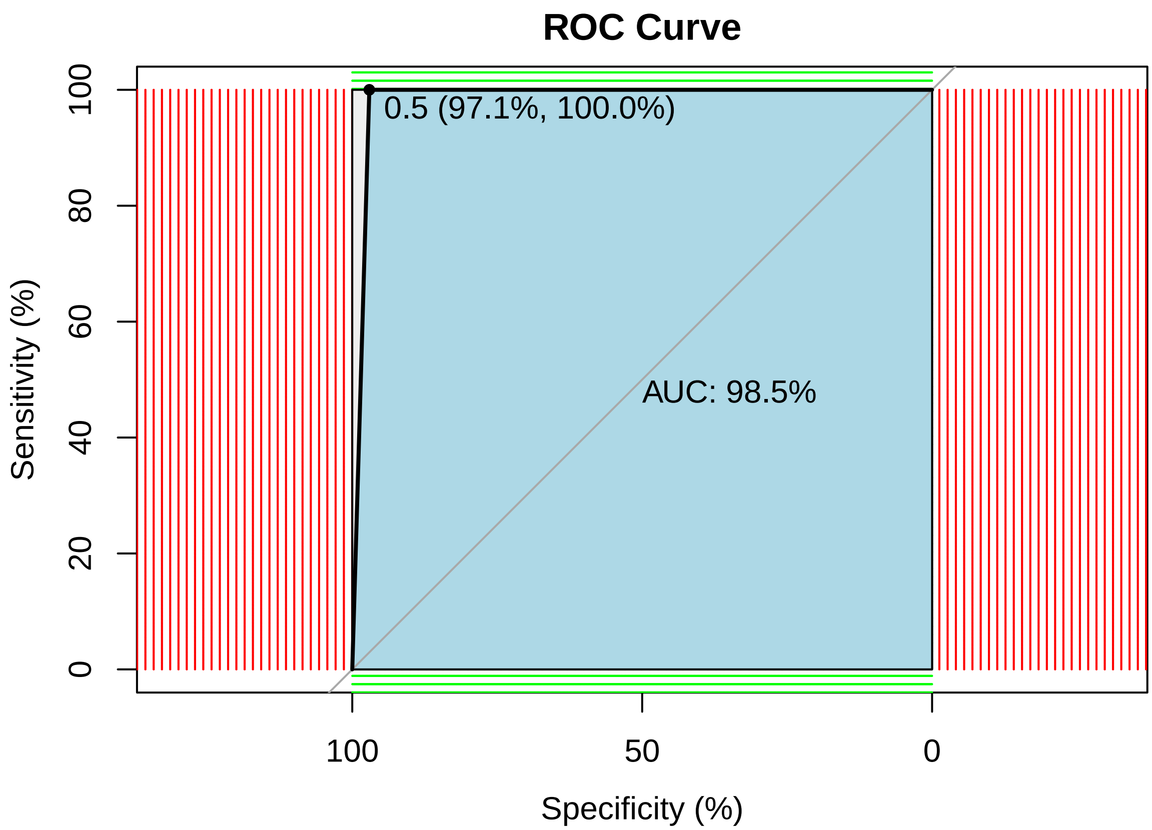

```r
#zm()
```

```r
p1 <- predict(entree.fit, test, type = 'prob')
p1 <- p1[,2]
r <- multiclass.roc(test$NObeyesdad, p1, percent = TRUE)
```

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

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

```r
roc <- r[['rocs']]
r1 <- roc[[1]]
plot.roc(r1,
         print.auc=TRUE,
         auc.polygon=TRUE,
         grid=c(0.1, 0.2),
         grid.col=c("green", "red"),
         max.auc.polygon=TRUE,
         auc.polygon.col="lightblue",
         print.thres=TRUE,
         main= 'ROC Curve')
```



Presentation Feedback

Since BMI is a derived predictor from Weight & Height and has high correlation with the response variable, we were suggested to drop BMI and weight in order to analyze the influence of other predictors on the response variable. In the subsequent sub sections, without BMI and Weight predictors, different models were fitted to the data and their results were analyzed.

SVM Modeling (without Weight & BMI)

```
set.seed(560)
data_updated <- subset(data.final, select=-c(Weight))

# stratified split; train: 75%, test: 25%
indices_updated <- createDataPartition(data_updated$NObeyesdad, p = 0.75, list = FALSE)

train_new <- data_updated[indices_updated,]
test_new <- data_updated[-indices_updated,]
```

```
train_new$CALC <- as.numeric(as.character(train_new$CALC))

svm_new.fit <- svm(NObeyesdad ~ ., data = train_new,
               type="C-classification",
               kernel = "radial")
summary(svm_new.fit)
```

```
##
```

```
## Call:
## svm(formula = NObeyesdad ~ ., data = train_new, type = "C-classification",
##     kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  1135
##
##  ( 205 198 209 191 142 142 48 )
##
##
## Number of Classes:  7
##
## Levels:
##  0 1 2 3 4 5 6
```

```
set.seed(104)
train_new$CALC <- as.numeric(as.character(train_new$CALC))
test_new$CALC <- as.numeric(as.character(test_new$CALC))

 svm1_new <- svm(NObeyesdad~., data=train_new,
          type="C-classification",
          kernal="radial")

 summary(svm1_new)
```

```
##
## Call:
## svm(formula = NObeyesdad ~ ., data = train_new, type = "C-classification",
##     kernal = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  1135
##
##  ( 205 198 209 191 142 142 48 )
##
##
## Number of Classes:  7
##
## Levels:
##  0 1 2 3 4 5 6
```

```
train_new.pred <- predict(svm_new.fit, train_new)
test_new.pred <- predict(svm_new.fit, test_new)
train.error <- mean(train_new.pred != train_new$NObeyesdad)
test.error <- mean(test_new.pred != test_new$NObeyesdad)
```

```r
print(paste("Train Error Rate (Radial kernel) = ", train.error*100, "%"))
```

```
## [1] "Train Error Rate (Radial kernel) =  18.3480453972257 %"
```

```r
print(paste("Test Error Rate (Radial kernel) = ", test.error*100, "%"))
```

```
## [1] "Test Error Rate (Radial kernel) =  25.1428571428571 %"
```

```r
confusionMatrix(train_new.pred, as.factor(train_new$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6
##          0 175  11   3   0   0   4   3
##          1  19 168   9   5   2  19  12
##          2   1  11 206   1   0  25  21
##          3   0   2  21 212   0   4  16
##          4   0   0   2   0 240   4   3
##          5   9  13  12   0   0 150  19
##          6   0  11  11   5   1  12 144
##
## Overall Statistics
##
##                Accuracy : 0.8165
##                  95% CI : (0.7966, 0.8353)
##     No Information Rate : 0.1665
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7857
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.8578   0.7778   0.7803   0.9507   0.9877  0.68807
## Specificity            0.9848   0.9518   0.9554   0.9685   0.9933  0.96126
## Pos Pred Value         0.8929   0.7179   0.7774   0.8314   0.9639  0.73892
## Neg Pred Value         0.9791   0.9645   0.9561   0.9917   0.9978  0.95083
## Prevalence             0.1286   0.1362   0.1665   0.1406   0.1532  0.13745
## Detection Rate         0.1103   0.1059   0.1299   0.1337   0.1513  0.09458
## Detection Prevalence   0.1236   0.1475   0.1671   0.1608   0.1570  0.12799
## Balanced Accuracy      0.9213   0.8648   0.8678   0.9596   0.9905  0.82467
##                      Class: 6
## Sensitivity           0.66055
## Specificity           0.97076
## Pos Pred Value        0.78261
## Neg Pred Value        0.94722
## Prevalence            0.13745
## Detection Rate        0.09079
## Detection Prevalence  0.11602
## Balanced Accuracy     0.81566
```

```r
confusionMatrix(test_new.pred, as.factor(test_new$NObeyesdad))
```

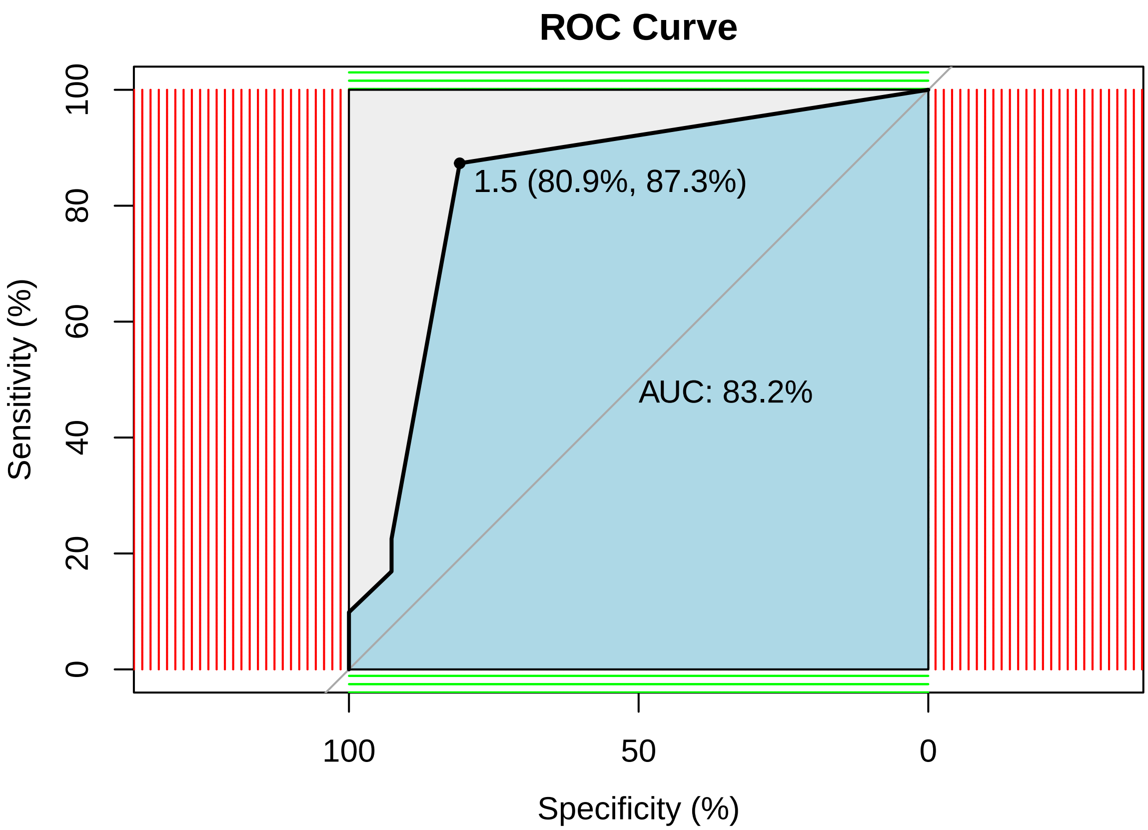
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6
##          0 55  9  0  0  0  1  0
##          1  8 46  8  0  0  8  9
##          2  0  4 58  0  0 10 12
##          3  1  1  8 73  0  0 12
##          4  0  0  3  0 81  0  0
##          5  4  4  7  0  0 47  6
##          6  0  7  3  1  0  6 33
##
## Overall Statistics
##
##                Accuracy : 0.7486
##                  95% CI : (0.7092, 0.7851)
##     No Information Rate : 0.1657
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7063
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.8088  0.64789   0.6667   0.9865   1.0000  0.65278
## Specificity            0.9781  0.92731   0.9406   0.9512   0.9932  0.95364
## Pos Pred Value         0.8462  0.58228   0.6905   0.7684   0.9643  0.69118
## Neg Pred Value         0.9717  0.94395   0.9342   0.9977   1.0000  0.94530
## Prevalence             0.1295  0.13524   0.1657   0.1410   0.1543  0.13714
## Detection Rate         0.1048  0.08762   0.1105   0.1390   0.1543  0.08952
## Detection Prevalence   0.1238  0.15048   0.1600   0.1810   0.1600  0.12952
## Balanced Accuracy      0.8935  0.78760   0.8037   0.9689   0.9966  0.80321
##                      Class: 6
## Sensitivity           0.45833
## Specificity           0.96247
## Pos Pred Value        0.66000
## Neg Pred Value        0.91789
## Prevalence            0.13714
## Detection Rate        0.06286
## Detection Prevalence  0.09524
## Balanced Accuracy     0.71040
```

```r
p1 <- predict(svm_new.fit, test_new, type = 'prob')
#p1 <- p1[,2]
r <- multiclass.roc(as.numeric(as.character(test_new$NObeyesdad)), as.numeric(p1), percent = TRUE)
```

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

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

```r
roc <- r[['rocs']]
r1 <- roc[[1]]
plot.roc(r1,
         print.auc=TRUE,
         auc.polygon=TRUE,
         grid=c(0.1, 0.2),
         grid.col=c("green", "red"),
         max.auc.polygon=TRUE,
         auc.polygon.col="lightblue",
         print.thres=TRUE,
         main= 'ROC Curve')
```

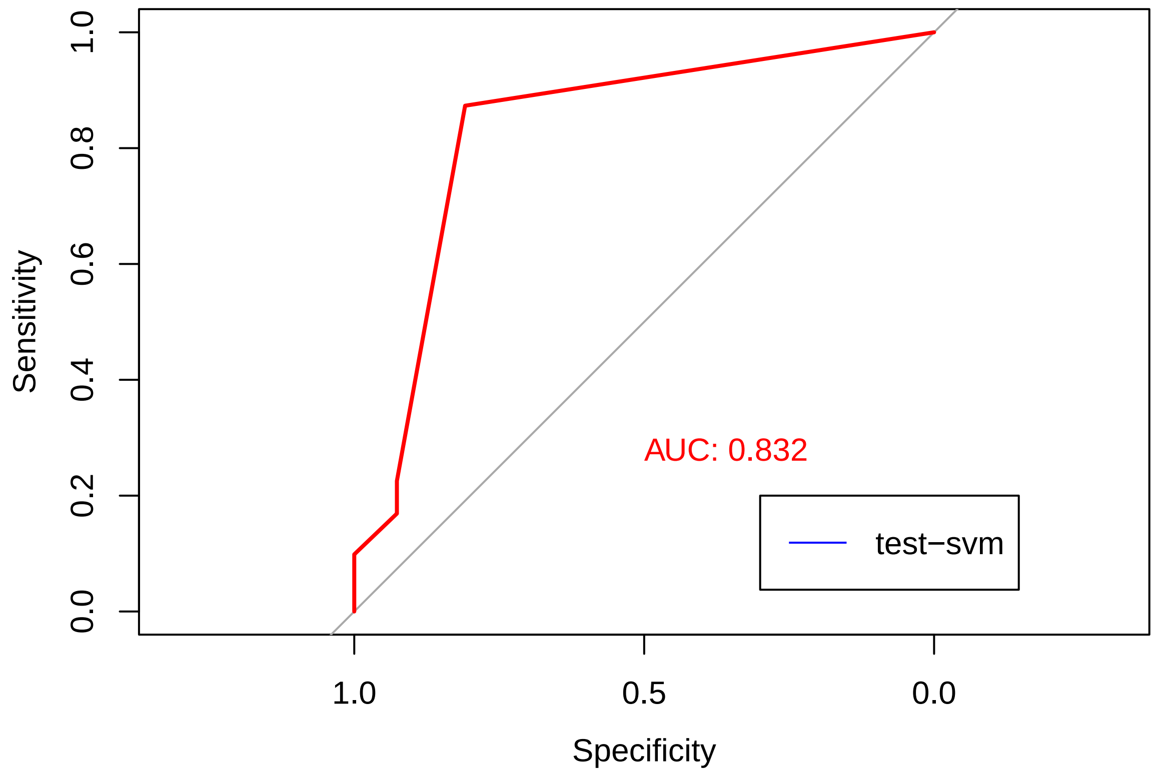



```
roc_svm_test_new <- roc(response = test_new$NObeyesdad, predictor =as.numeric(test_new.pred))
```

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

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

```
plot(roc_svm_test_new,
     #add = TRUE,
     col = "red",
     print.auc=TRUE,
     print.auc.x = 0.5,
     print.auc.y = 0.3
     )
legend(0.3, 0.2, legend = c("test-svm"), lty = c(1), col = c("blue"))
```



SVM Linear

```
svm.fit_linear <- svm(NObeyesdad ~ ., data = train_new,
               type="C-classification",
               kernel = "linear")

train.pred <- predict(svm.fit_linear, train_new)
test.pred <- predict(svm.fit_linear, test_new)
train.error <- mean(train.pred != train_new$NObeyesdad)
test.error <- mean(test.pred != test_new$NObeyesdad)
print(paste("Train Error Rate (Linear kernel) = ", train.error*100, "%"))
```

```
## [1] "Train Error Rate (Linear kernel) =  30.8322824716267 %"
```

```
print(paste("Test Error Rate (Linear kernel) = ", test.error*100, "%"))
```

```
## [1] "Test Error Rate (Linear kernel) =  35.8095238095238 %"
```

```
confusionMatrix(train.pred, as.factor(train_new$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction    0   1   2   3   4   5   6
##            0 163  44   4   1   1   9   4
##            1  25 118  10   2   0  29  13
##            2   8  18 185  12   0  53  61
```

```
##          3   0   3  35 198   0   8  35
##          4   0   0   2   0 241   1   1
##          5   7  22  10   0   0 106  18
##          6   1  11  18  10   1  12  86
##
## Overall Statistics
##
##                Accuracy : 0.6917
##                  95% CI : (0.6683, 0.7143)
##     No Information Rate : 0.1665
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6393
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.7990   0.5463   0.7008   0.8879   0.9918  0.48624
## Specificity            0.9544   0.9423   0.8850   0.9406   0.9970  0.95833
## Pos Pred Value         0.7212   0.5990   0.5490   0.7097   0.9837  0.65031
## Neg Pred Value         0.9699   0.9294   0.9367   0.9809   0.9985  0.92129
## Prevalence             0.1286   0.1362   0.1665   0.1406   0.1532  0.13745
## Detection Rate         0.1028   0.0744   0.1166   0.1248   0.1520  0.06683
## Detection Prevalence   0.1425   0.1242   0.2125   0.1759   0.1545  0.10277
## Balanced Accuracy      0.8767   0.7443   0.7929   0.9142   0.9944  0.72229
##                      Class: 6
## Sensitivity           0.39450
## Specificity           0.96126
## Pos Pred Value        0.61871
## Neg Pred Value        0.90878
## Prevalence            0.13745
## Detection Rate        0.05422
## Detection Prevalence  0.08764
## Balanced Accuracy     0.67788
```

```
confusionMatrix(test.pred, as.factor(test_new$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6
##          0 48 17  0  0  0  1  0
##          1 10 30  6  0  0  9  4
##          2  3  6 48  5  0 18 20
##          3  0  1 10 65  0  1 17
##          4  0  0  3  0 81  0  0
##          5  5  7  6  0  0 40  6
##          6  2 10 14  4  0  3 25
##
## Overall Statistics
##
##                Accuracy : 0.6419
##                  95% CI : (0.5992, 0.683)
```

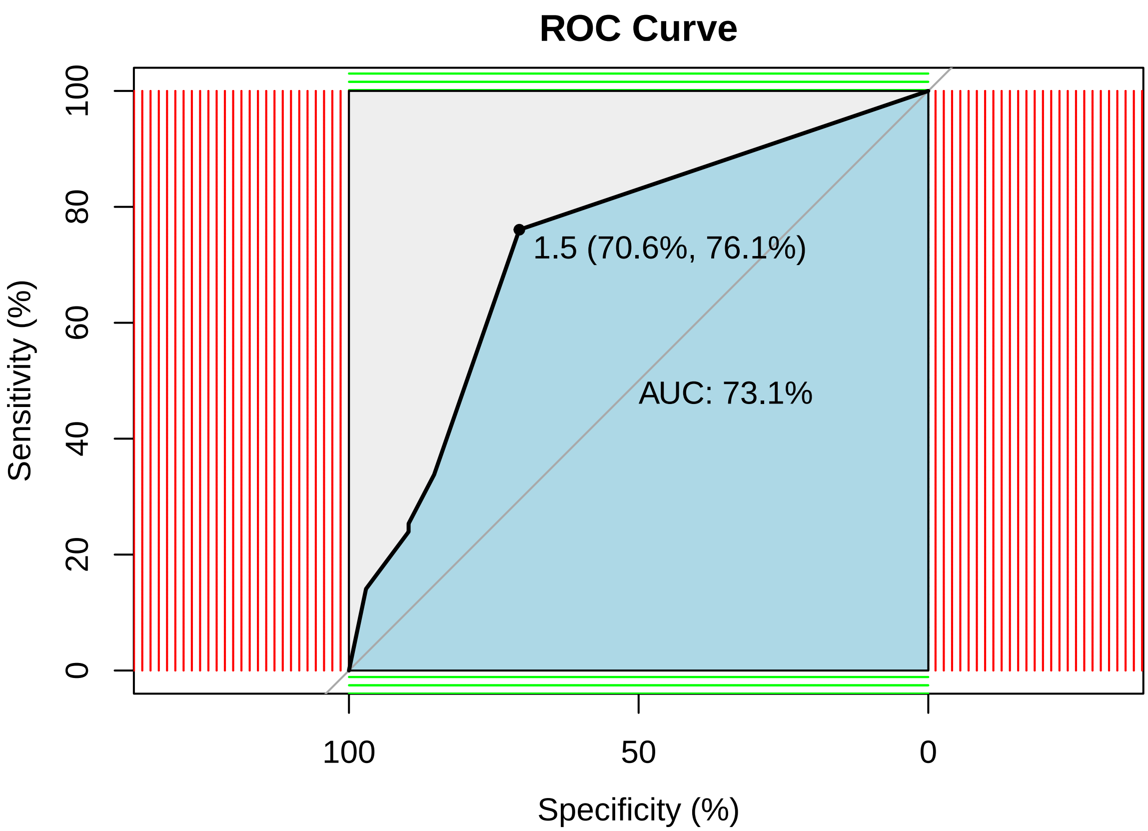
```
##     No Information Rate : 0.1657
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5813
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.70588  0.42254  0.55172   0.8784   1.0000  0.55556
## Specificity           0.96061  0.93612  0.88128   0.9357   0.9932  0.94702
## Pos Pred Value        0.72727  0.50847  0.48000   0.6915   0.9643  0.62500
## Neg Pred Value        0.95643  0.91202  0.90824   0.9791   1.0000  0.93059
## Prevalence            0.12952  0.13524  0.16571   0.1410   0.1543  0.13714
## Detection Rate        0.09143  0.05714  0.09143   0.1238   0.1543  0.07619
## Detection Prevalence  0.12571  0.11238  0.19048   0.1790   0.1600  0.12190
## Balanced Accuracy     0.83325  0.67933  0.71650   0.9070   0.9966  0.75129
##                      Class: 6
## Sensitivity           0.34722
## Specificity           0.92715
## Pos Pred Value        0.43103
## Neg Pred Value        0.89936
## Prevalence            0.13714
## Detection Rate        0.04762
## Detection Prevalence  0.11048
## Balanced Accuracy     0.63719
```

```
p1 <- predict(svm.fit_linear, test_new, type = 'prob')
#p1 <- p1[,2]
r <- multiclass.roc(as.numeric(as.character(test_new$NObeyesdad)), as.numeric(p1), percent = TRUE)
```

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

```
## Setting direction: controls > cases
## Setting direction: controls > cases
```

```r
roc <- r[['rocs']]
r1 <- roc[[1]]
plot.roc(r1,
         print.auc=TRUE,
         auc.polygon=TRUE,
         grid=c(0.1, 0.2),
         grid.col=c("green", "red"),
         max.auc.polygon=TRUE,
         auc.polygon.col="lightblue",
         print.thres=TRUE,
         main= 'ROC Curve')
```



SVM Polynomial

```r
svm.fit_poly <- svm(NObeyesdad ~ ., data = train_new,
               type="C-classification",
               kernel = "polynomial", degree=2)

train.pred <- predict(svm.fit_poly, train_new)
test.pred <- predict(svm.fit_poly, test_new)
train.error <- mean(train.pred != train_new$NObeyesdad)
test.error <- mean(test.pred != test_new$NObeyesdad)
print(paste("Train Error Rate (Polynomial kernel) = ", train.error*100, "%"))
```

```
## [1] "Train Error Rate (Polynomial kernel) =  22.3203026481715 %"
```

```r
print(paste("Test Error Rate (Polynomial kernel) = ", test.error*100, "%"))
```

```
## [1] "Test Error Rate (Polynomial kernel) =  28.5714285714286 %"
```

```r
confusionMatrix(train.pred, as.factor(train_new$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6
##          0 159  14   1   0   0   7   4
##          1  18 159   6   4   1  11  12
##          2  21  19 220   7   0  59  43
##          3   2   7  26 209   0  13  18
##          4   0   0   2   0 241   5   3
##          5   4   9   5   0   0 111   5
##          6   0   8   4   3   1  12 133
##
## Overall Statistics
##
##                Accuracy : 0.7768
##                  95% CI : (0.7555, 0.7971)
##     No Information Rate : 0.1665
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7386
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.7794   0.7361   0.8333   0.9372   0.9918  0.50917
## Specificity            0.9812   0.9620   0.8873   0.9516   0.9926  0.98319
## Pos Pred Value         0.8595   0.7536   0.5962   0.7600   0.9602  0.82836
## Neg Pred Value         0.9679   0.9585   0.9638   0.9893   0.9985  0.92631
## Prevalence             0.1286   0.1362   0.1665   0.1406   0.1532  0.13745
## Detection Rate         0.1003   0.1003   0.1387   0.1318   0.1520  0.06999
## Detection Prevalence   0.1166   0.1330   0.2327   0.1734   0.1583  0.08449
## Balanced Accuracy      0.8803   0.8491   0.8603   0.9444   0.9922  0.74618
##                      Class: 6
## Sensitivity           0.61009
## Specificity           0.97953
## Pos Pred Value        0.82609
## Neg Pred Value        0.94035
## Prevalence            0.13745
## Detection Rate        0.08386
## Detection Prevalence  0.10151
## Balanced Accuracy     0.79481
```

```r
confusionMatrix(test.pred, as.factor(test_new$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

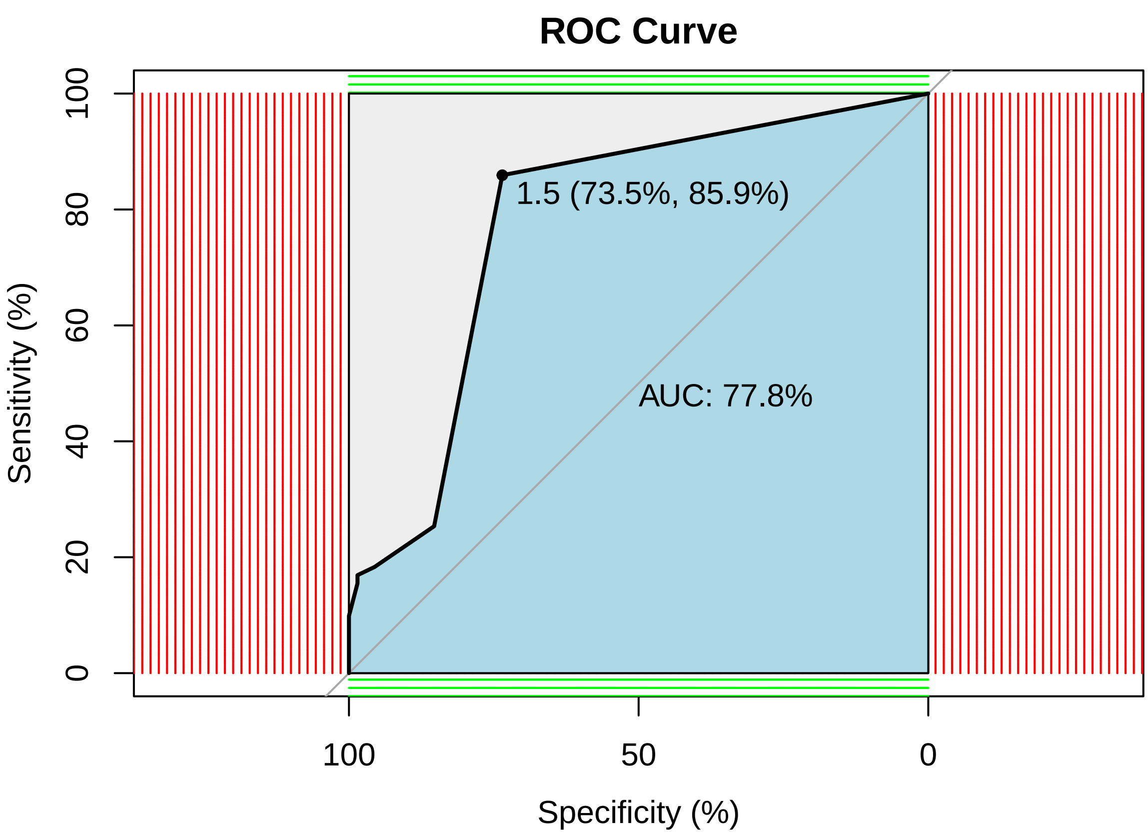
```
## Prediction  0  1  2  3  4  5  6
##          0 50 10  0  0  0  1  0
##          1  8 43  7  1  0  4  8
##          2  7  5 63  2  0 22 18
##          3  2  1 11 70  0  5 12
##          4  0  1  3  0 81  1  0
##          5  1  4  2  0  0 36  2
##          6  0  7  1  1  0  3 32
##
## Overall Statistics
##
##                Accuracy : 0.7143
##                  95% CI : (0.6736, 0.7526)
##     No Information Rate : 0.1657
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6655
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.73529   0.6056   0.7241   0.9459   1.0000  0.50000
## Specificity           0.97593   0.9383   0.8767   0.9313   0.9887  0.98013
## Pos Pred Value        0.81967   0.6056   0.5385   0.6931   0.9419  0.80000
## Neg Pred Value        0.96121   0.9383   0.9412   0.9906   1.0000  0.92500
## Prevalence            0.12952   0.1352   0.1657   0.1410   0.1543  0.13714
## Detection Rate        0.09524   0.0819   0.1200   0.1333   0.1543  0.06857
## Detection Prevalence  0.11619   0.1352   0.2229   0.1924   0.1638  0.08571
## Balanced Accuracy     0.85561   0.7720   0.8004   0.9386   0.9944  0.74007
##                      Class: 6
## Sensitivity           0.44444
## Specificity           0.97351
## Pos Pred Value        0.72727
## Neg Pred Value        0.91684
## Prevalence            0.13714
## Detection Rate        0.06095
## Detection Prevalence  0.08381
## Balanced Accuracy     0.70898
```

```
p1 <- predict(svm.fit_poly, test_new, type = 'prob')
#p1 <- p1[,2]
r <- multiclass.roc(as.numeric(as.character(test_new$NObeyesdad)), as.numeric(p1), percent = TRUE)
```

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

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

## Setting direction: controls > cases
## Setting direction: controls > cases
```

```r
roc <- r[['rocs']]
r1 <- roc[[1]]
plot.roc(r1,
         print.auc=TRUE,
         auc.polygon=TRUE,
         grid=c(0.1, 0.2),
         grid.col=c("green", "red"),
         max.auc.polygon=TRUE,
         auc.polygon.col="lightblue",
         print.thres=TRUE,
         main= 'ROC Curve')
```



SVM Summary (data excluding the BMI & Weight)

After eliminating the BMI & Weight predictor , the linear kernal svm train error increased from 10.4% to 35.3% and the test error increased from 10.85 to 40%. For radial kernal svm, the train dataset error increased from 8.39% to 24.89% and test error increased from 11.61% to 30.85%.

Decision Tree (Without Weight and BMI)

```r
tree.fit <- rpart(NObeyesdad ~ . , data = train_new, method='class')
tree.fit
```

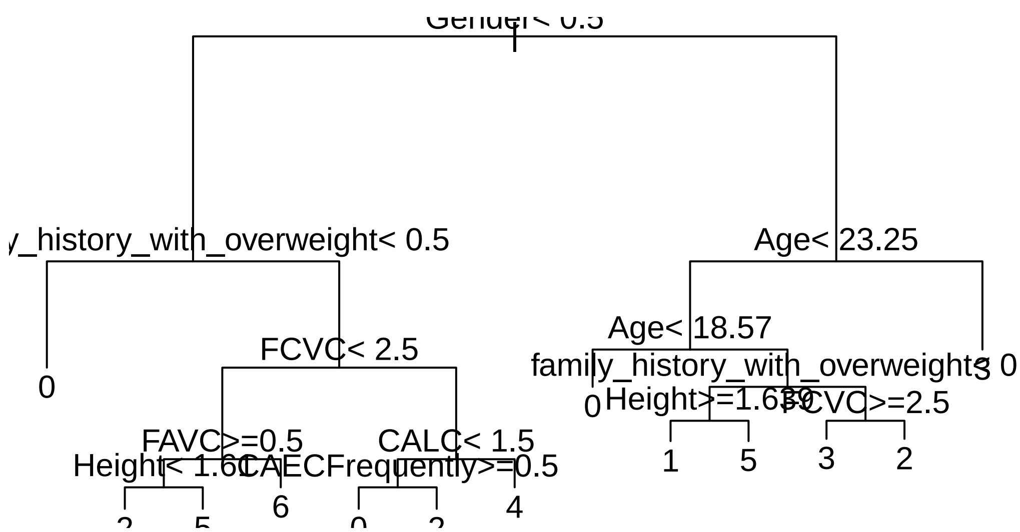
```
## n= 1586
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 1586 1322 2 (0.13 0.14 0.17 0.14 0.15 0.14 0.14)
##    2) Gender< 0.5 790  548 4 (0.16 0.14 0.15 0.0025 0.31 0.14 0.1)
##      4) family_history_with_overweight< 0.5 172   78 0 (0.55 0.28 0.0058 0.0058 0 0.15 0.012) *
##      5) family_history_with_overweight>=0.5 618  376 4 (0.053 0.099 0.18 0.0016 0.39 0.14 0.13)
##       10) FCVC< 2.5 233  152 2 (0.052 0.14 0.35 0 0 0.27 0.19)
##         20) FAVC>=0.5 203  122 2 (0.054 0.14 0.4 0 0 0.3 0.11)
##           40) Height< 1.60966 78   26 2 (0.026 0.15 0.67 0 0 0.15 0) *
##           41) Height>=1.60966 125   77 5 (0.072 0.13 0.23 0 0 0.38 0.18) *
##         21) FAVC< 0.5 30    8 6 (0.033 0.17 0 0 0 0.067 0.73) *
##       11) FCVC>=2.5 385  143 4 (0.055 0.073 0.086 0.0026 0.63 0.065 0.091)
##         22) CALC< 1.5 76   54 2 (0.28 0.16 0.29 0 0.013 0.013 0.25)
##           44) CAECFrequently>=0.5 25    5 0 (0.8 0.12 0.04 0 0 0 0.04) *
##           45) CAECFrequently< 0.5 51   30 2 (0.02 0.18 0.41 0 0.02 0.02 0.35) *
##         23) CALC>=1.5 309   68 4 (0 0.052 0.036 0.0032 0.78 0.078 0.052) *
##    3) Gender>=0.5 796  575 3 (0.097 0.13 0.19 0.28 0.0013 0.13 0.17)
##      6) Age< 23.25392 437  335 2 (0.18 0.2 0.23 0.055 0.0023 0.18 0.16)
##       12) Age< 18.56852 130   73 0 (0.44 0.18 0.18 0 0.0077 0.046 0.14) *
##       13) Age>=18.56852 307  229 2 (0.065 0.2 0.25 0.078 0 0.24 0.16)
##         26) family_history_with_overweight< 0.5 60   29 1 (0 0.52 0.017 0 0 0.4 0.067)
##           52) Height>=1.639042 35    7 1 (0 0.8 0.029 0 0 0.086 0.086) *
##           53) Height< 1.639042 25    4 5 (0 0.12 0 0 0 0.84 0.04) *
##         27) family_history_with_overweight>=0.5 247  170 2 (0.081 0.13 0.31 0.097 0 0.2 0.19)
##           54) FCVC>=2.5 64   45 3 (0.16 0.19 0.047 0.3 0 0.27 0.047) *
##           55) FCVC< 2.5 183  109 2 (0.055 0.1 0.4 0.027 0 0.17 0.23) *
##      7) Age>=23.25392 359  162 3 (0 0.056 0.13 0.55 0 0.075 0.19) *
```

```r
tree.fit$variable.importance
```

```
##                            Age                           FCVC
##                    111.3338970                    110.6476874
##                         Height family_history_with_overweight
##                    104.5012089                     79.1952890
##                         Gender                           CALC
##                     71.0485530                     61.0560753
##    MTRANSPublic_Transportation                   CAECSometimes
##                     33.8376523                     28.9906974
##                            NCP                  CAECFrequently
##                     27.2769030                     26.5035799
##                            FAF                         CAECno
```

```
##                   19.3322594                 16.1655782
##                         FAVC                       CH2O
##                   15.6120750                  9.3461472
##                          TUE                        SCC
##                    6.9322989                  2.4325533
##                   MTRANSBike              MTRANSWalking
##                    0.2502343                  0.1320575
```

```r
plot(tree.fit)
text(tree.fit, pretty=0)
```




```r
#zm()
```

From this tree result, Frequency of consumption of vegetables (FCVC) is the most significant predictor that best differentiates the response categories. People who consumes more vegetable are highly likely to have 'Normal_Weight'. Next, Age and Time of Using Device (TUE) are significant in splitting the data further into respective obesity categories.

```r
# Decision tree Train Test set prediction result

tree.predtrain <- predict(tree.fit, train_new, type = "class")
tree.predtest <- predict(tree.fit, test_new, type = "class")

train.error <- mean(tree.predtrain != train_new$NObeyesdad)
test.error <- mean(tree.predtest != test_new$NObeyesdad)

print(paste("Misclassification error rate in train = ", train.error))
```

```
## [1] "Misclassification error rate in train =  0.436317780580076"
```

```r
print(paste("Misclassification error rate in test = ", test.error))
```

```
## [1] "Misclassification error rate in test =  0.478095238095238"
```

The misclassification error rate in both train and test data are higher without Weight and BMI predictors.

```r
confusionMatrix(tree.predtest,
                as.factor(test$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6
##          0 52 18  9  0  1 14  8
##          1  1  9  2  0  0  2  0
##          2  2 16 32  4  0 22 16
##          3  6 14 25 66  2 14 31
##          4  2  7  6  4 78  5  5
##          5  5  2 13  0  0 15  7
##          6  0  5  0  0  0  0  5
##
## Overall Statistics
##
##                Accuracy : 0.4895
##                  95% CI : (0.446, 0.5332)
##     No Information Rate : 0.1657
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4028
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.76471  0.12676  0.36782   0.8919   0.9630  0.20833
## Specificity           0.89059  0.98899  0.86301   0.7960   0.9347  0.94040
## Pos Pred Value        0.50980  0.64286  0.34783   0.4177   0.7290  0.35714
## Neg Pred Value        0.96217  0.87867  0.87298   0.9782   0.9928  0.88199
## Prevalence            0.12952  0.13524  0.16571   0.1410   0.1543  0.13714
## Detection Rate        0.09905  0.01714  0.06095   0.1257   0.1486  0.02857
## Detection Prevalence  0.19429  0.02667  0.17524   0.3010   0.2038  0.08000
## Balanced Accuracy     0.82765  0.55787  0.61541   0.8440   0.9488  0.57437
##                      Class: 6
## Sensitivity          0.069444
## Specificity          0.988962
## Pos Pred Value       0.500000
## Neg Pred Value       0.869903
## Prevalence           0.137143
## Detection Rate       0.009524
## Detection Prevalence 0.019048
## Balanced Accuracy    0.529203
```

```r
confusionMatrix(tree.predtrain,
                as.factor(train$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6
##          0 156  75  30   1   8  36  21
##          1   4  18   4   1   0   3   5
##          2  12  43 138  11   0  48  60
##          3  15  38  47 207   3  45  68
##          4   1  17  14   2 231  25  19
##          5  14  19  30   1   1  61  24
##          6   2   6   1   0   0   0  21
##
## Overall Statistics
##
##                Accuracy : 0.5246
##                  95% CI : (0.4997, 0.5494)
##     No Information Rate : 0.1665
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4438
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.76471  0.08333  0.52273   0.9283   0.9506  0.27982
## Specificity           0.87627  0.98759  0.86838   0.8415   0.9419  0.93494
## Pos Pred Value        0.47706  0.51429  0.44231   0.4894   0.7476  0.40667
## Neg Pred Value        0.96187  0.87234  0.90110   0.9862   0.9906  0.89067
## Prevalence            0.12863  0.13619  0.16646   0.1406   0.1532  0.13745
## Detection Rate        0.09836  0.01135  0.08701   0.1305   0.1456  0.03846
## Detection Prevalence  0.20618  0.02207  0.19672   0.2667   0.1948  0.09458
## Balanced Accuracy     0.82049  0.53546  0.69555   0.8849   0.9463  0.60738
##                      Class: 6
## Sensitivity           0.09633
## Specificity           0.99342
## Pos Pred Value        0.70000
## Neg Pred Value        0.87339
## Prevalence            0.13745
## Detection Rate        0.01324
## Detection Prevalence  0.01892
## Balanced Accuracy     0.54488
```

```r
p1 <- predict(tree.fit, test, type = 'prob')
p1 <- p1[,2]
r <- multiclass.roc(test$NObeyesdad, p1, percent = TRUE)
```

```
## Setting direction: controls > cases
## Setting direction: controls > cases
## Setting direction: controls > cases
```

```
## Setting direction: controls > cases
## Setting direction: controls > cases
## Setting direction: controls > cases
## Setting direction: controls > cases
## Setting direction: controls > cases
## Setting direction: controls > cases
## Setting direction: controls > cases
## Setting direction: controls > cases
## Setting direction: controls > cases
## Setting direction: controls > cases
```

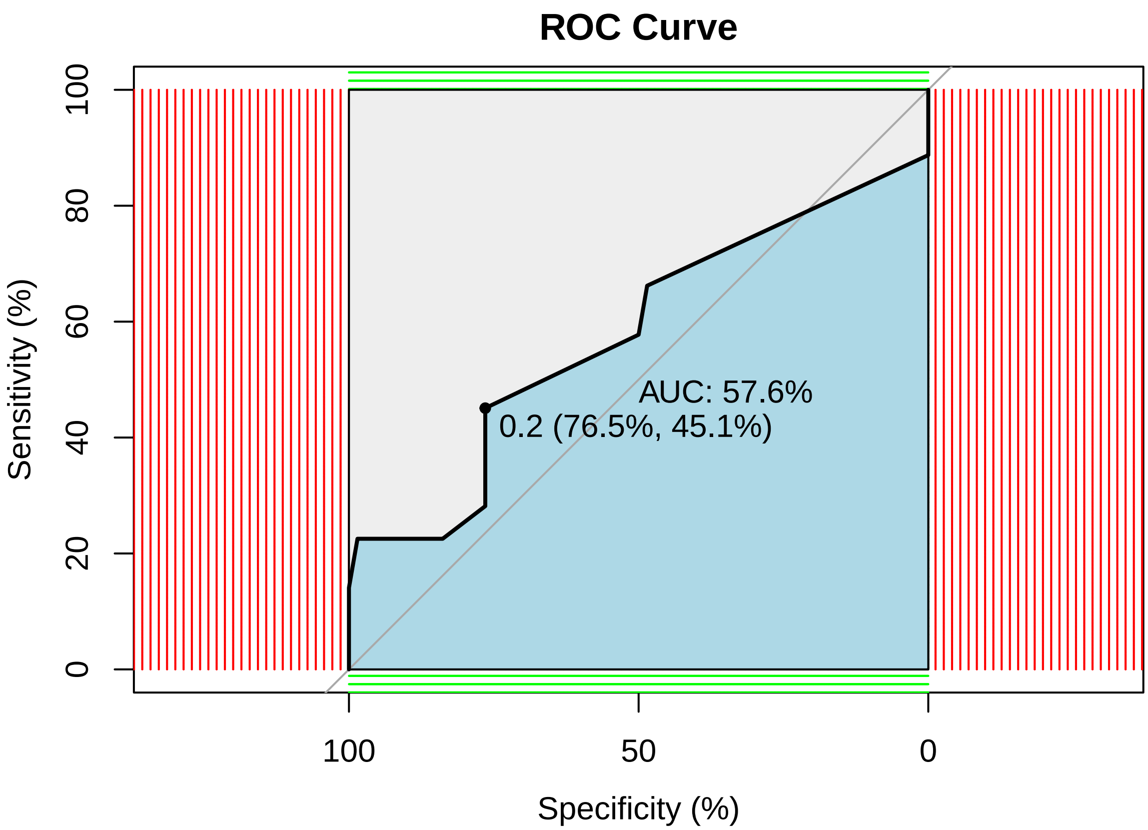
```
## Setting direction: controls < cases
```

```
## Setting direction: controls > cases
## Setting direction: controls > cases
```

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

```
## Setting direction: controls > cases
```

```
roc <- r[['rocs']]
r1 <- roc[[1]]
plot.roc(r1,
         print.auc=TRUE,
         auc.polygon=TRUE,
         grid=c(0.1, 0.2),
         grid.col=c("green", "red"),
         max.auc.polygon=TRUE,
         auc.polygon.col="lightblue",
         print.thres=TRUE,
         main= 'ROC Curve')
```



Pruning

```
best_cp <- tree.fit$cptable[which.min(tree.fit$cptable[, "xerror"]), "CP"]
best_cp
```

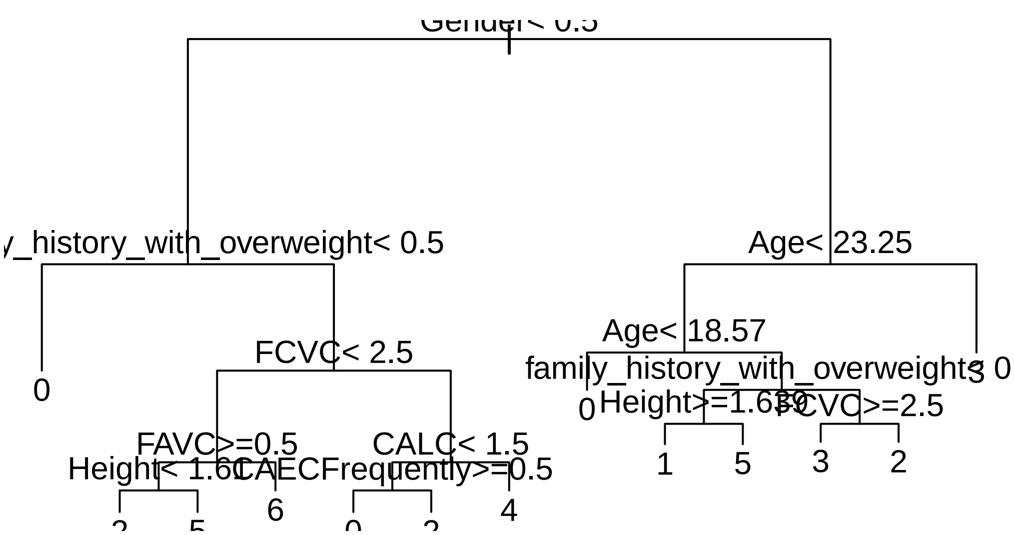
```
## [1] 0.01
```

```
tree.prune <- prune(tree.fit, cp = best_cp)
tree.prune
```

```
## n= 1586
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 1586 1322 2 (0.13 0.14 0.17 0.14 0.15 0.14 0.14)
##    2) Gender< 0.5 790  548 4 (0.16 0.14 0.15 0.0025 0.31 0.14 0.1)
##      4) family_history_with_overweight< 0.5 172   78 0 (0.55 0.28 0.0058 0.0058 0 0.15 0.012) *
##      5) family_history_with_overweight>=0.5 618  376 4 (0.053 0.099 0.18 0.0016 0.39 0.14 0.13)
##       10) FCVC< 2.5 233  152 2 (0.052 0.14 0.35 0 0 0.27 0.19)
##         20) FAVC>=0.5 203  122 2 (0.054 0.14 0.4 0 0 0.3 0.11)
##           40) Height< 1.60966 78   26 2 (0.026 0.15 0.67 0 0 0.15 0) *
##           41) Height>=1.60966 125   77 5 (0.072 0.13 0.23 0 0 0.38 0.18) *
##         21) FAVC< 0.5 30    8 6 (0.033 0.17 0 0 0 0.067 0.73) *
##       11) FCVC>=2.5 385  143 4 (0.055 0.073 0.086 0.0026 0.63 0.065 0.091)
##         22) CALC< 1.5 76   54 2 (0.28 0.16 0.29 0 0.013 0.013 0.25)
```

```
##           44) CAECFrequently>=0.5 25    5 0 (0.8 0.12 0.04 0 0 0 0.04) *
##           45) CAECFrequently< 0.5 51   30 2 (0.02 0.18 0.41 0 0.02 0.02 0.35) *
##         23) CALC>=1.5 309   68 4 (0 0.052 0.036 0.0032 0.78 0.078 0.052) *
##    3) Gender>=0.5 796  575 3 (0.097 0.13 0.19 0.28 0.0013 0.13 0.17)
##      6) Age< 23.25392 437  335 2 (0.18 0.2 0.23 0.055 0.0023 0.18 0.16)
##       12) Age< 18.56852 130   73 0 (0.44 0.18 0.18 0 0.0077 0.046 0.14) *
##       13) Age>=18.56852 307  229 2 (0.065 0.2 0.25 0.078 0 0.24 0.16)
##         26) family_history_with_overweight< 0.5 60   29 1 (0 0.52 0.017 0 0 0.4 0.067)
##           52) Height>=1.639042 35    7 1 (0 0.8 0.029 0 0 0.086 0.086) *
##           53) Height< 1.639042 25    4 5 (0 0.12 0 0 0 0.84 0.04) *
##         27) family_history_with_overweight>=0.5 247  170 2 (0.081 0.13 0.31 0.097 0 0.2 0.19)
##           54) FCVC>=2.5 64   45 3 (0.16 0.19 0.047 0.3 0 0.27 0.047) *
##           55) FCVC< 2.5 183  109 2 (0.055 0.1 0.4 0.027 0 0.17 0.23) *
##      7) Age>=23.25392 359  162 3 (0 0.056 0.13 0.55 0 0.075 0.19) *
```

```r
plot(tree.prune)
text(tree.prune, pretty=0)
```




```r
#zm()
```

```r
# Train Test set prediction result

tree.predtrain <- predict(tree.prune, train_new, type = "class")
tree.predtest <- predict(tree.prune, test_new, type = "class")

train.error <- mean(tree.predtrain != train_new$NObeyesdad)
test.error <- mean(tree.predtest != test_new$NObeyesdad)
```

```r
print(paste("Misclassification error rate in train = ", train.error))
```

```
## [1] "Misclassification error rate in train =  0.436317780580076"
```

```r
print(paste("Misclassification error rate in test = ", test.error))
```

```
## [1] "Misclassification error rate in test =  0.478095238095238"
```

```r
confusionMatrix(tree.predtest,
                as.factor(test$NObeyesdad))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6
##          0 52 18  9  0  1 14  8
##          1  1  9  2  0  0  2  0
##          2  2 16 32  4  0 22 16
##          3  6 14 25 66  2 14 31
##          4  2  7  6  4 78  5  5
##          5  5  2 13  0  0 15  7
##          6  0  5  0  0  0  0  5
##
## Overall Statistics
##
##                Accuracy : 0.4895
##                  95% CI : (0.446, 0.5332)
##     No Information Rate : 0.1657
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4028
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.76471  0.12676  0.36782   0.8919   0.9630  0.20833
## Specificity           0.89059  0.98899  0.86301   0.7960   0.9347  0.94040
## Pos Pred Value        0.50980  0.64286  0.34783   0.4177   0.7290  0.35714
## Neg Pred Value        0.96217  0.87867  0.87298   0.9782   0.9928  0.88199
## Prevalence            0.12952  0.13524  0.16571   0.1410   0.1543  0.13714
## Detection Rate        0.09905  0.01714  0.06095   0.1257   0.1486  0.02857
## Detection Prevalence  0.19429  0.02667  0.17524   0.3010   0.2038  0.08000
## Balanced Accuracy     0.82765  0.55787  0.61541   0.8440   0.9488  0.57437
##                      Class: 6
## Sensitivity          0.069444
## Specificity          0.988962
## Pos Pred Value       0.500000
## Neg Pred Value       0.869903
## Prevalence           0.137143
## Detection Rate       0.009524
## Detection Prevalence 0.019048
## Balanced Accuracy    0.529203
```

```r
confusionMatrix(tree.predtrain,
                as.factor(train$NObeyesdad))
```

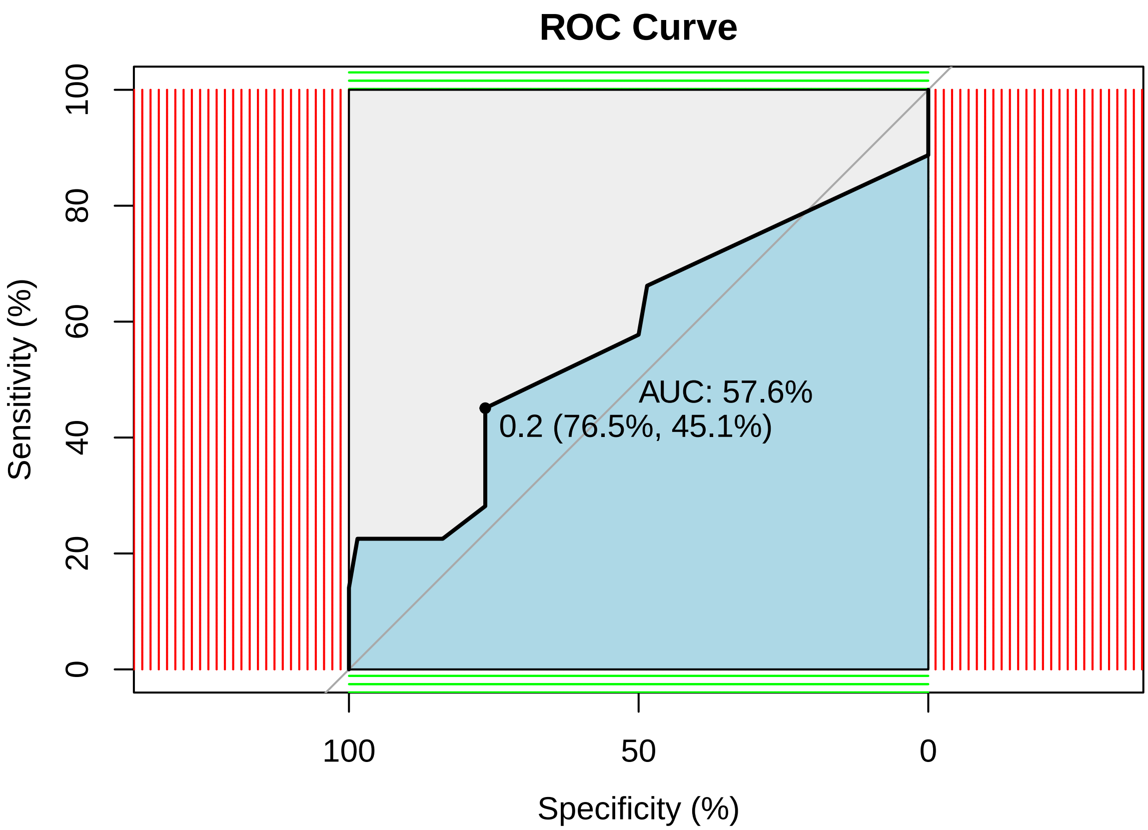
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6
##          0 156  75  30   1   8  36  21
##          1   4  18   4   1   0   3   5
##          2  12  43 138  11   0  48  60
##          3  15  38  47 207   3  45  68
##          4   1  17  14   2 231  25  19
##          5  14  19  30   1   1  61  24
##          6   2   6   1   0   0   0  21
##
## Overall Statistics
##
##                Accuracy : 0.5246
##                  95% CI : (0.4997, 0.5494)
##     No Information Rate : 0.1665
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4438
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.76471  0.08333  0.52273   0.9283   0.9506  0.27982
## Specificity           0.87627  0.98759  0.86838   0.8415   0.9419  0.93494
## Pos Pred Value        0.47706  0.51429  0.44231   0.4894   0.7476  0.40667
## Neg Pred Value        0.96187  0.87234  0.90110   0.9862   0.9906  0.89067
## Prevalence            0.12863  0.13619  0.16646   0.1406   0.1532  0.13745
## Detection Rate        0.09836  0.01135  0.08701   0.1305   0.1456  0.03846
## Detection Prevalence  0.20618  0.02207  0.19672   0.2667   0.1948  0.09458
## Balanced Accuracy     0.82049  0.53546  0.69555   0.8849   0.9463  0.60738
##                      Class: 6
## Sensitivity           0.09633
## Specificity           0.99342
## Pos Pred Value        0.70000
## Neg Pred Value        0.87339
## Prevalence            0.13745
## Detection Rate        0.01324
## Detection Prevalence  0.01892
## Balanced Accuracy     0.54488
```

```r
p1 <- predict(tree.prune, test, type = 'prob')
p1 <- p1[,2]
r <- multiclass.roc(test$NObeyesdad, p1, percent = TRUE)
```

```
## Setting direction: controls > cases
## Setting direction: controls > cases
## Setting direction: controls > cases
```

```
## Setting direction: controls > cases
## Setting direction: controls > cases
## Setting direction: controls > cases
## Setting direction: controls > cases
## Setting direction: controls > cases
## Setting direction: controls > cases
## Setting direction: controls > cases
## Setting direction: controls > cases
## Setting direction: controls > cases
## Setting direction: controls > cases

## Setting direction: controls < cases

## Setting direction: controls > cases
## Setting direction: controls > cases

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

## Setting direction: controls > cases
```

```r
roc <- r[['rocs']]
r1 <- roc[[1]]
plot.roc(r1,
         print.auc=TRUE,
         auc.polygon=TRUE,
         grid=c(0.1, 0.2),
         grid.col=c("green", "red"),
         max.auc.polygon=TRUE,
         auc.polygon.col="lightblue",
         print.thres=TRUE,
         main= 'ROC Curve')
```



Bagging model with the updated data

```
options(warn=-1)
set.seed(545)

train1_new<-train_new

train1_new$NObeyesdad <- mapvalues(train1_new$NObeyesdad,
                                   from=c(0, 1, 2, 3, 4, 5, 6),
          to=c("Insufficient_Weight","Normal_Weight","Obesity_Type_I", "Obesity_Type_II",
                 "Obesity_Type_III", "Overweight_Level_I", "Overweight_Level_II"))

gbag_new <- bagging(as.factor(NObeyesdad) ~ ., data = train1_new,coob=T,nbag=40)
print(gbag_new)
```

```
##
## Bagging classification trees with 40 bootstrap replications
##
## Call: bagging.data.frame(formula = as.factor(NObeyesdad) ~ ., data = train1_new,
##     coob = T, nbag = 40)
##
## Out-of-bag estimate of misclassification error:  0.1923
```

The Out-of-bag estimate error increased from 4% to 19.23% by eliminating BMI & Weight from the dataset.

```r
bag.predtrain_new <- predict(gbag_new, train1_new)
xtab.train <- table(train_new$NObeyesdad, bag.predtrain_new)
print("Confussion matrix for train data")
```

```
## [1] "Confussion matrix for train data"
```

```r
xtab.train
```

```
##    bag.predtrain_new
##     Insufficient_Weight Normal_Weight Obesity_Type_I Obesity_Type_II
##   0                 204             0              0               0
##   1                   0           216              0               0
##   2                   0             0            263               0
##   3                   0             0              0             223
##   4                   0             0              0               0
##   5                   0             0              0               0
##   6                   0             0              0               0
##    bag.predtrain_new
##     Obesity_Type_III Overweight_Level_I Overweight_Level_II
##   0                0                  0                   0
##   1                0                  0                   0
##   2                0                  0                   1
##   3                0                  0                   0
##   4              243                  0                   0
##   5                0                218                   0
##   6                0                  0                 218
```

```r
#test prediction
test1_new<-test_new
test1_new$NObeyesdad <- mapvalues(test1_new$NObeyesdad,
                           from=c(0, 1, 2, 3, 4, 5, 6),
          to=c("Insufficient_Weight","Normal_Weight","Obesity_Type_I", "Obesity_Type_II",
                "Obesity_Type_III", "Overweight_Level_I", "Overweight_Level_II"))

bag.predtest_new <- predict(gbag_new, test1_new)
xtab.test <- table(test_new$NObeyesdad, bag.predtest_new)
print("Confussion matrix for test data")
```

```
## [1] "Confussion matrix for test data"
```

```r
xtab.test
```

```
##    bag.predtest_new
##     Insufficient_Weight Normal_Weight Obesity_Type_I Obesity_Type_II
##   0                  59             5              0               0
##   1                  10            40              5               2
##   2                   0             7             66               3
##   3                   0             0              1              70
##   4                   0             0              0               0
##   5                   1             8              3               1
##   6                   1             3              5               5
##    bag.predtest_new
##     Obesity_Type_III Overweight_Level_I Overweight_Level_II
##   0                0                  3                   1
##   1                1                  7                   6
##   2                2                  6                   3
```

```
##   3                0                  0                    3
##   4               81                  0                    0
##   5                0                 57                    2
##   6                0                  1                   57
```

```
bag.train.error_new <- mean(bag.predtrain_new != train1_new$NObeyesdad)
bag.test.error_new <- mean(bag.predtest_new != test1_new$NObeyesdad)

print(paste("Misclassification error rate in train = ",round(bag.train.error_new,4),"%"))
```

```
## [1] "Misclassification error rate in train =  6e-04 %"
```

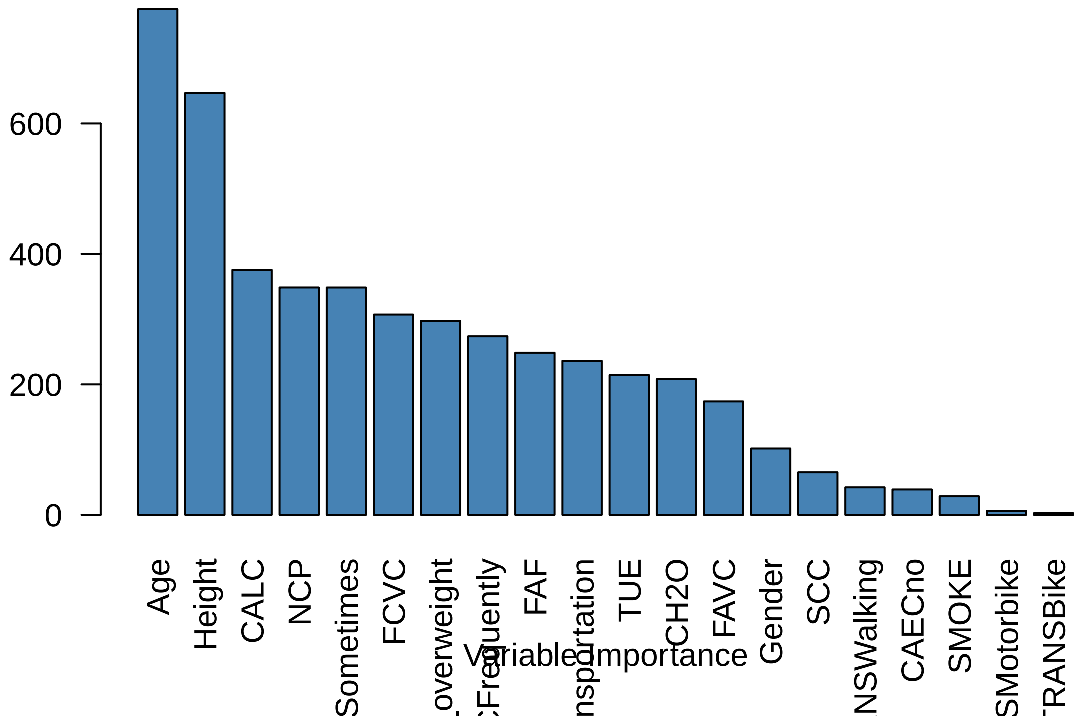
```
print(paste("Misclassification error rate in test = ", round(bag.test.error_new,4),"%"))
```

```
## [1] "Misclassification error rate in test =  0.181 %"
```

The misclassification error in test data increased from 3% to 18% with the new dataset.

Feature Importance

```
#calculate variable importance
varimp.data_new <- subset(train1_new, select=-c(NObeyesdad))
VI <- data.frame(var=names(varimp.data_new), imp=varImp(gbag_new))

#sort variable importance descending
VI_plot <- VI[order(VI$Overall, decreasing=TRUE),]

#visualize variable importance with horizontal bar plot
barplot(VI_plot$Overall,
        names.arg=rownames(VI_plot),
        horiz=FALSE,
        col='steelblue',
        xlab='Variable Importance',
        las=2,
        srt=45)
```



After eliminating Weight & BMI predictors, Age, Height NCP (Consumption of high calorific food) are highly correlated with the predictor variable

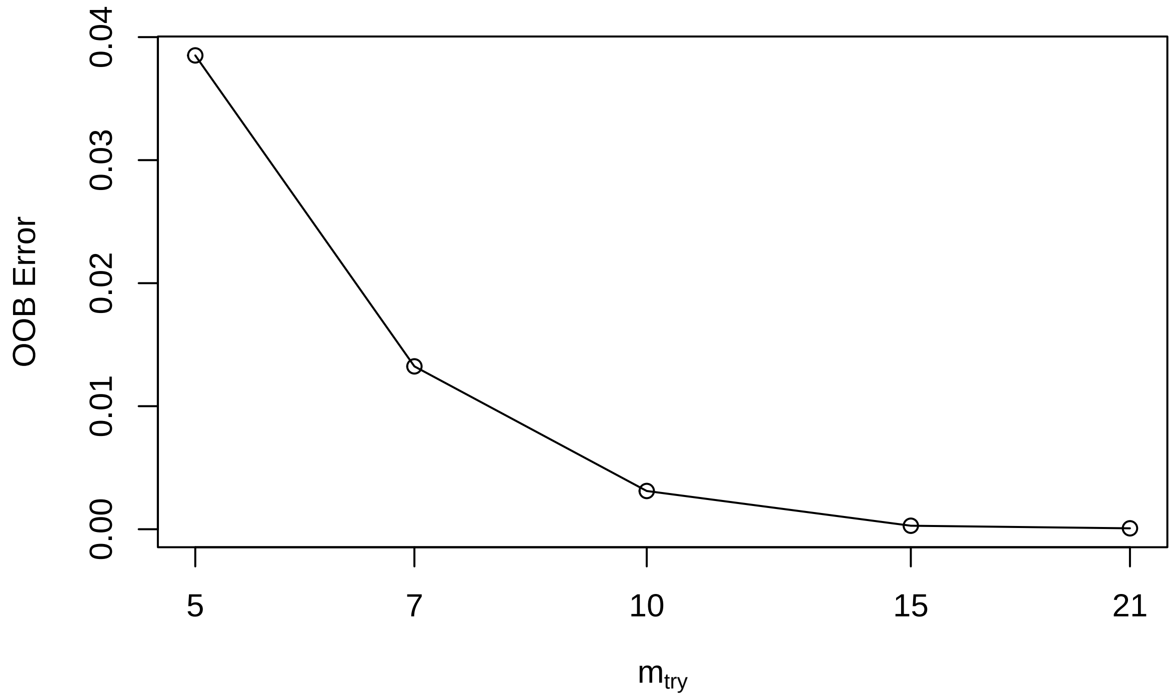
Random Forest-without BMI

Random Forest without BMI: In order to improve the random forest model we have to tune the model to obtain the best mtry value which reduces the OOB error. OOB error is important because when choosing the datasets for random forest or bagging or boosting it takes only 2/3 rds of data and we have to make sure to reduce the error with remaining 3rd data. Below we have obtained at mtry value 21 we can produce less OOB error. Reason to remove BMI is to understand how well the predictors are correlated with the response variable. Now Comparing the results from random Forest- with BMI and without BMI dataset, we can see that the training error and test error has increased, having training error as 23% and test error as 45% without BMI.Random Forest also gives information on which predictors have the most importance like in this case top predictors without BMI are Gender,Age,Height. OOB error is recorded as 35.98% without BMI and 2.28% with BMI. So we say that Modelling with BMI includes highly correlated variables and modelling without BMI has highly uncorrelated variables which explains why our outputs are higher in case of OOB error and test,training errors.

```
#TuneRF-to find best mtry
set.seed(1)
bestmtry_new <-tuneRF(sapply(train_new, as.numeric),sapply(train_new$NObeyesdad, as.numeric),improve = 0
```

```
## mtry = 7  OOB error = 0.01323653
## Searching left ...
## mtry = 5     OOB error = 0.03851146
## -1.909483 0.01
## Searching right ...
```

```
## mtry = 10     OOB error = 0.003106982
## 0.7652721 0.01
## mtry = 15     OOB error = 0.0002828541
## 0.9089618 0.01
## mtry = 21     OOB error = 7.395923e-05
## 0.7385252 0.01
```




```r
print(bestmtry_new)
```

```
##    mtry     OOBError
## 5     5 3.851146e-02
## 7     7 1.323653e-02
## 10   10 3.106982e-03
## 15   15 2.828541e-04
## 21   21 7.395923e-05
```

```r
#Random forest model
set.seed(545)
library(randomForest)

rf.fit_new <- randomForest(
  NObeyesdad ~ .,
  data = sapply(train_new, as.numeric),
  importance = TRUE,
  mtry = 21,
  ntree = 5000
)
```

```r
rf.pred_new <- predict(rf.fit_new, train_new, type='class')
rf.predtest_new <- predict(rf.fit_new, test_new, type='class')

print("--- Training Error - Random Forest ---")
```

```
## [1] "--- Training Error - Random Forest ---"
```

```r
print(mean(round(rf.pred_new) != train_new$NObeyesdad))
```

```
## [1] 0.221942
```

```r
print("--- Test Error - Random Forest ---")
```

```
## [1] "--- Test Error - Random Forest ---"
```

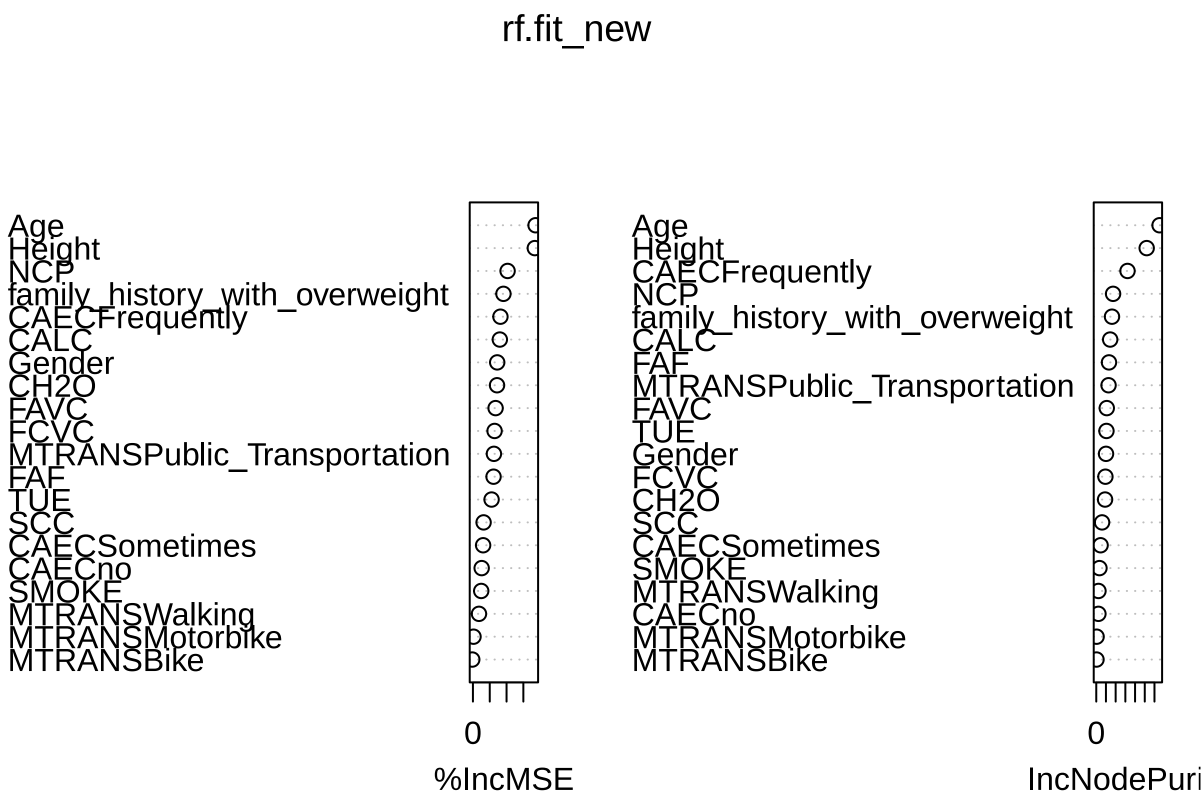
```r
print(mean(round(rf.predtest_new) != test_new$NObeyesdad))
```

```
## [1] 0.4342857
```

```r
rf.fit_new
```

```
##
## Call:
##  randomForest(formula = NObeyesdad ~ ., data = sapply(train_new,      as.numeric), importance = TRUE
##                Type of random forest: regression
##                      Number of trees: 5000
## No. of variables tried at each split: 20
##
##           Mean of squared residuals: 1.29891
##                     % Var explained: 65.9
```

```r
varImpPlot(rf.fit_new)
```




```
importance(rf.fit_new)
```

```
##                                   %IncMSE IncNodePurity
## Gender                         144.433107    201.411660
## Age                            372.596236   1304.974127
## Height                         368.142030   1039.692729
## family_history_with_overweight 181.409637    324.257913
## FAVC                           134.776295    216.716394
## FCVC                           128.779804    188.961710
## NCP                            206.175868    347.281713
## SMOKE                           49.295548     66.281790
## CH2O                           143.389589    181.473390
## SCC                             63.427941    120.201365
## FAF                            122.509582    262.549140
## TUE                            110.859832    210.219258
## CALC                           160.054518    286.876517
## CAECFrequently                 163.556439    644.374385
## CAECno                          51.895379     44.507623
## CAECSometimes                   60.547591     89.680683
## MTRANSBike                      -4.106655      5.112035
## MTRANSMotorbike                  3.354598      7.634782
## MTRANSPublic_Transportation    124.803690    253.625824
## MTRANSWalking                   35.993042     45.657681
```

Results Summary

The comprehensive results were presented in the previous section with explanation and reasoning for the performance of different models. Decision tree was the simplest and easy to interpret model that also performs well with misclassification error rate of approximately 3% in test data. In terms of performance, stacking performs well. Moreover, the error rate for bagging, random forest and stacking are almost closer that may not be impactful in the real world. Thus depending on the business requirement and budget constraint, just BMI and weight predictors or all predictors can be used. Also, either performance oriented model or simple interpretable model can be preferred

Conclusion

The Obesity level dataset was analyzed, preprocessed and different supervised models were fitted to the data and the results were analyzed. In terms of performance and lowest misclassification error rate, Stacking (SVM + Decision Tree –> Decision Tree –> Output) is the best model. In terms of simplicity and ease of interpretation, Decision tree performs the best. The difference in test error rate of these two models are less than 1.5%. BMI and weight play significant roles in determining the obesity category of an individual. Without these two, the influence of other predictors on the response variable is minimal and model fitted on them resulted in high misclassification error. Thus, depending on the business requirement and budget constraints, one predictors over the other or one model over the other can be preferred.

Authors Contributions Summary:

Priyanka Bhoite:

Data Cleaning & Preparation Exploratory Data Analysis

1. Analysis of Target Variable Distribution
2. Analyzing Distributions of Numeric Variables
3. Analyzing the categorical data and count
4. Relationship between Weight & Height for both genders Modeling - Support Vector Machines & Bagging (for dataset with BMI) Support Vector Machines & Bagging (for dataset without BMI)
5. Team Discussion

Akhilesh Nampalli:

1. Initial Project selection, Guiding on materials and methods to use, Validating EDA and Modelling methods.

2. Bagging

3. Randon forest- with BMI, without BMI

4. Team Discussion

Pradeepsurya Rajendran:

1. Complete Exploratory Data Analysis in Python. Created an interactive visualization plots and rendered it in HTML, presentable to the clients.

- Descriptive statistics measure
- Quantile statistics measure
- Variable Interactions
- Correlation Analysis (Pearson, Spearman, KendallTau)
- Missing values check

2. Categorical Encoding
3. Feature Engineering (Experimented with different features and chose BMI)
4. Decision Tree

5. Tree Pruning
6. SVM Model Tuning
7. Stacking
8. Model Assessments using ROC curve and Confusion Matrix
9. Code Review
10. Team Discussion

Github Repository

https://github.com/rpradeepsurya/sta545_statistical_data_mining_project

References

1. Dataset - Estimation of Obesity Levels
2. Lecture Slides
3. R Documentation