Data Mining Project

Data Description

This dataset contains comprehensive information on 2,392 high school students, detailing their demographics, study habits, parental involvement, extracurricular activities, and academic performance. The target variable, `GradeClass`, classifies students' grades into distinct categories.

Variables:

- **StudentID**: A unique identifier assigned to each student (1001 to 3392).
- **Age**: The age of the students ranges from 15 to 18 years.
- **Gender**: Gender of the students, where 0 represents Male and 1 represents Female.
- **Ethnicity**: The ethnicity of the students, coded as follows:
    - 0: Caucasian
    - 1: African American
    - 2: Asian
    - 3: Other
- **ParentalEducation**: The education level of the parents, coded as follows:
    - 0: None
    - 1: High School
    - 2: Some College
    - 3: Bachelor's
    - 4: Higher
- **StudyTimeWeekly**: Weekly study time in hours, ranging from 0 to 20.
- **Absences**: Number of absences during the school year, ranging from 0 to 30.
- **Tutoring**: Tutoring status, where 0 indicates No and 1 indicates Yes.

- **ParentalSupport**: The level of parental support, coded as follows:
  - 0: None
  - 1: Low
  - 2: Moderate
  - 3: High
  - 4: Very High
- **Extracurricular**: Participation in extracurricular activities, where 0 indicates No and 1 indicates Yes.
- **Sports**: Participation in sports, where 0 indicates No and 1 indicates Yes.
- **Music**: Participation in music activities, where 0 indicates No and 1 indicates Yes.
- **Volunteering**: Participation in volunteering, where 0 indicates No and 1 indicates Yes.
- **GPA**: Grade Point Average on a scale from 2.0 to 4.0, influenced by study habits, parental involvement, and extracurricular activities.
- **GradeClass**: Classification of students' grades based on GPA:
  - 0: 'A' (GPA >= 3.5)
  - 1: 'B' (3.0 <= GPA < 3.5)
  - 2: 'C' (2.5 <= GPA < 3.0)
  - 3: 'D' (2.0 <= GPA < 2.5)
  - 4: 'F' (GPA < 2.0)

Data Importing:

```
mydata0=read.csv("C:\\Users\\Lenovo\\Downloads\\archive (1)\\Student_performa
nce_data _.csv")
#View(mydata0)
mydata0$GradeClass=ifelse(mydata0$GradeClass==0,"A",
                      ifelse(mydata0$GradeClass==1,"B",
                        ifelse(mydata0$GradeClass==2,"C",
                            ifelse(mydata0$GradeClass==3,"D",
                               ifelse(mydata0$GradeClass==4,"F",NA)))))
mydata=mydata0[,-c(1,14)]
dim(mydata)
```

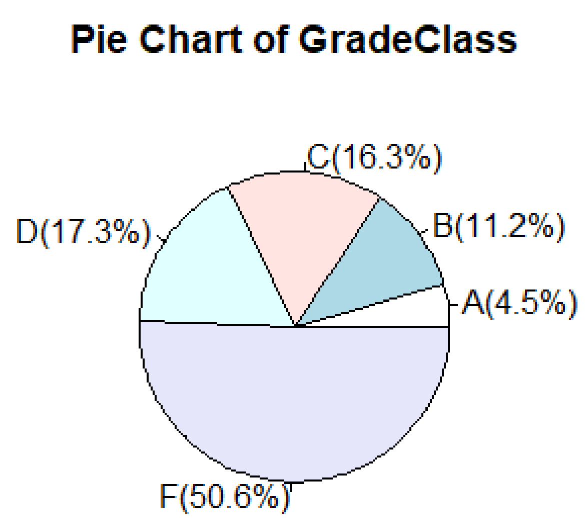
```
## [1] 2392   13

Y=mydata[,"GradeClass"]
X=mydata[,-13]
```

Exploratory Data Analysis

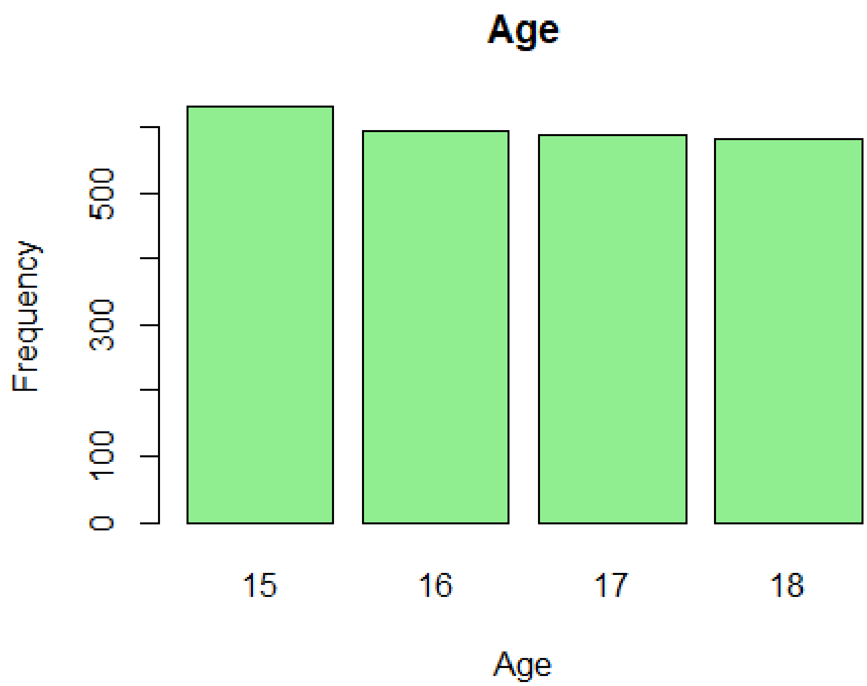
```
sum(is.na(mydata))

## [1] 0
```

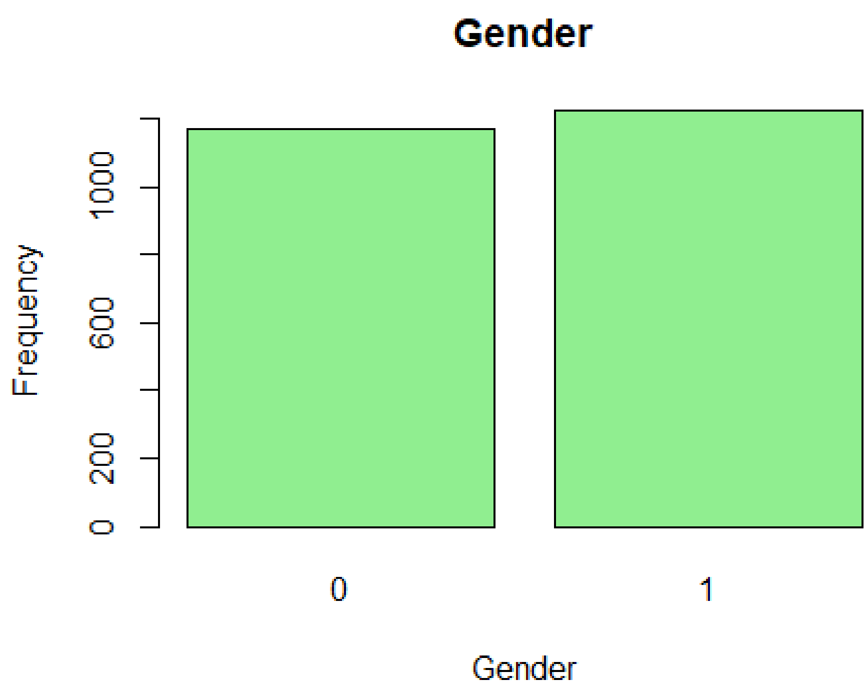
We confirm from the info that there are no nulls in this DataFrame.

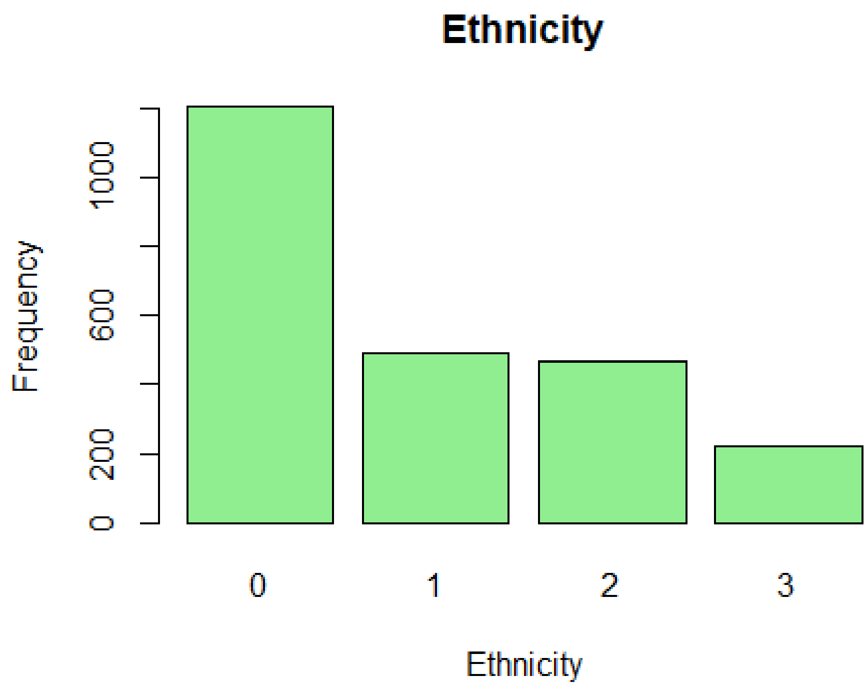
```
a=table(mydata$GradeClass)
label=c("A","B","C","D","F")
pie(a,labels=paste0(label,"(",round(a/ sum(a)*100,1),"%)"),
    main = "Pie Chart of GradeClass")
```

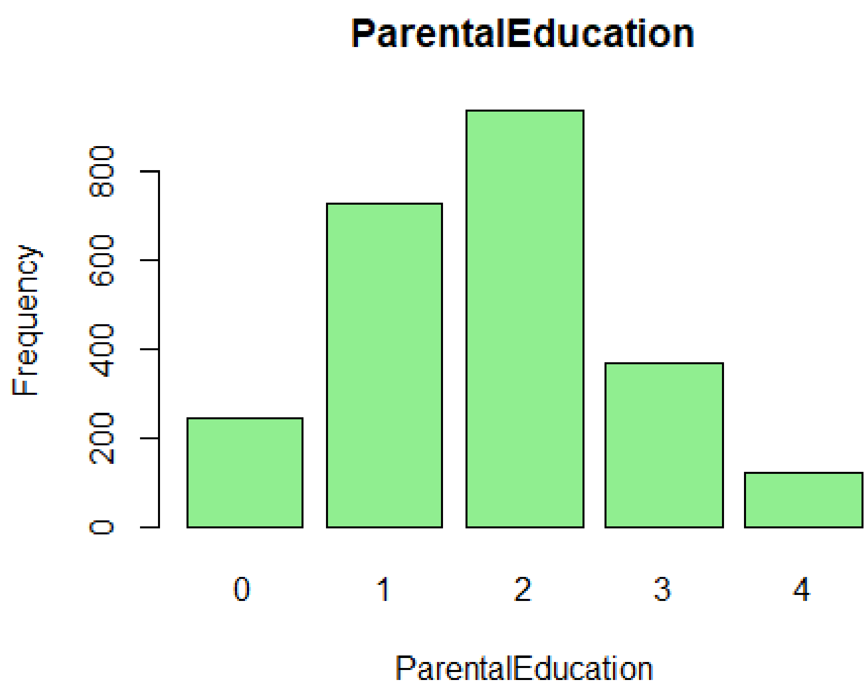


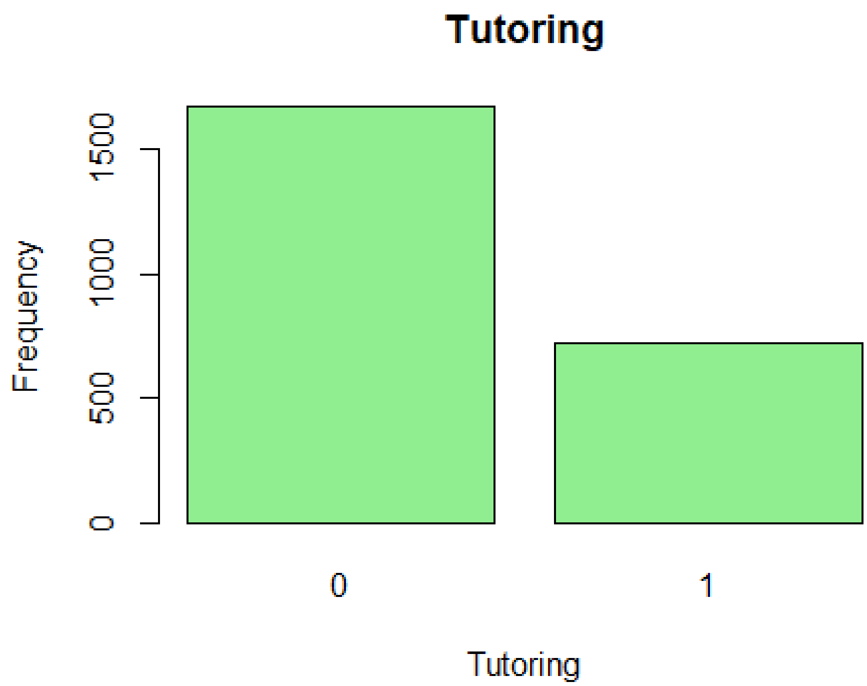

```
categorical=mydata[,-c(5,6,13)]
for(i in 1:ncol(categorical))
{
  a=table(categorical[,i])
  barplot(a,main=colnames(categorical)[i],
          xlab=colnames(categorical)[i],
          ylab="Frequency",col="lightgreen")
}
```

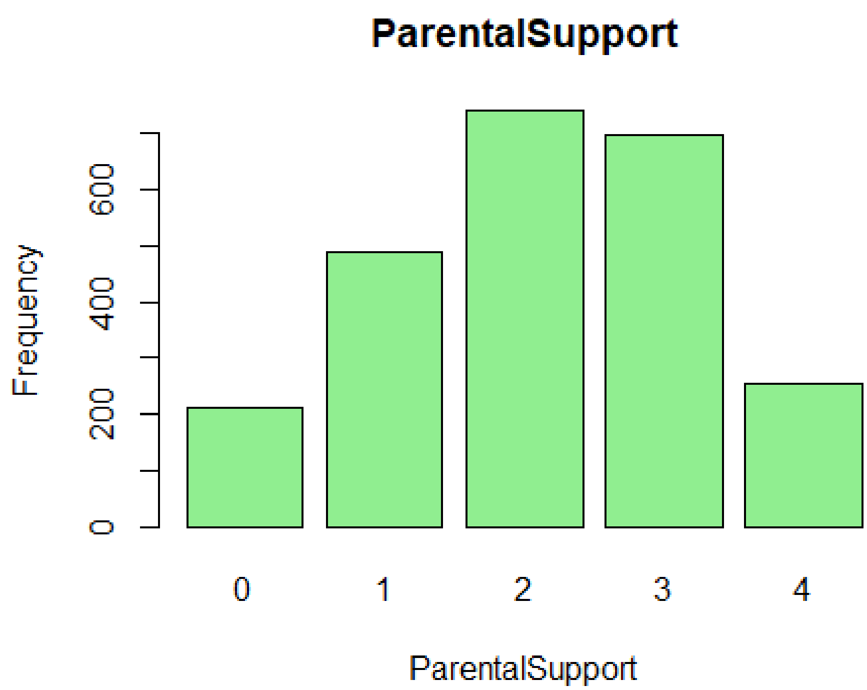

Age
Frequency
0
100
300
500
15
16
17
18
Age

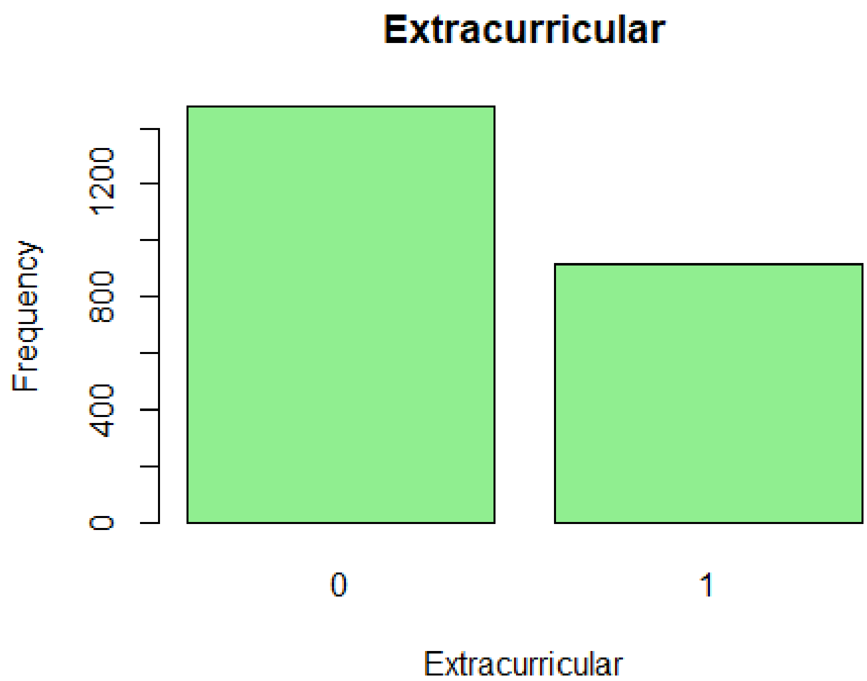

Gender
Frequency
0
200
600
1000
0
1
Gender

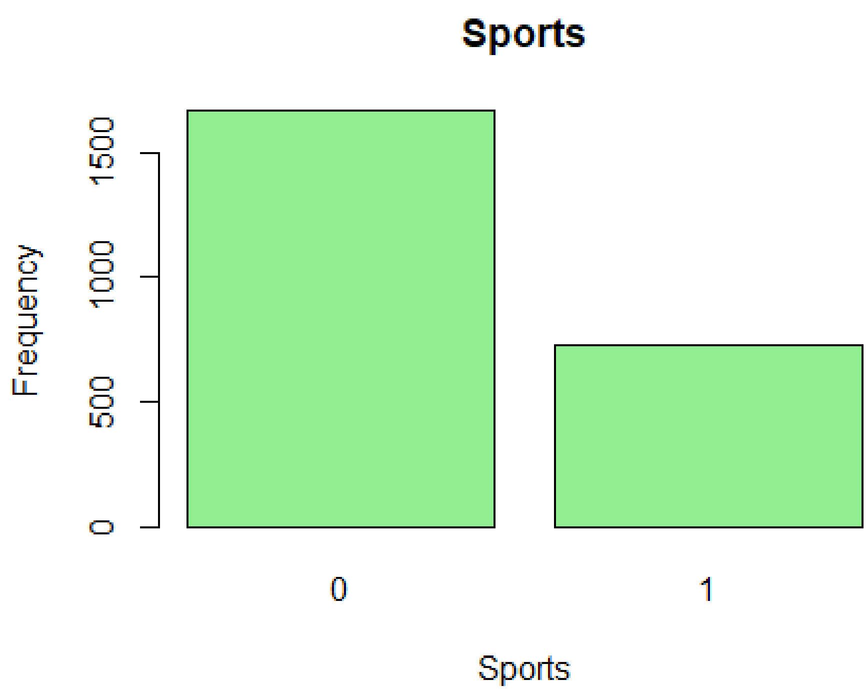

Ethnicity
Frequency
0
200
600
1000
0
1
2
3
Ethnicity

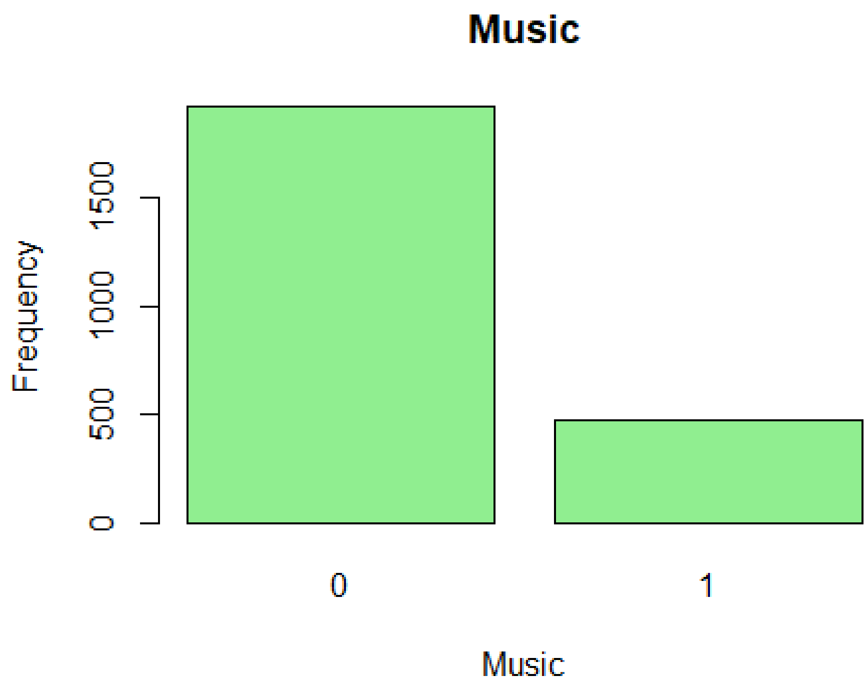

ParentalEducation
Frequency
0
200
400
600
800
0
1
2
3
4
ParentalEducation

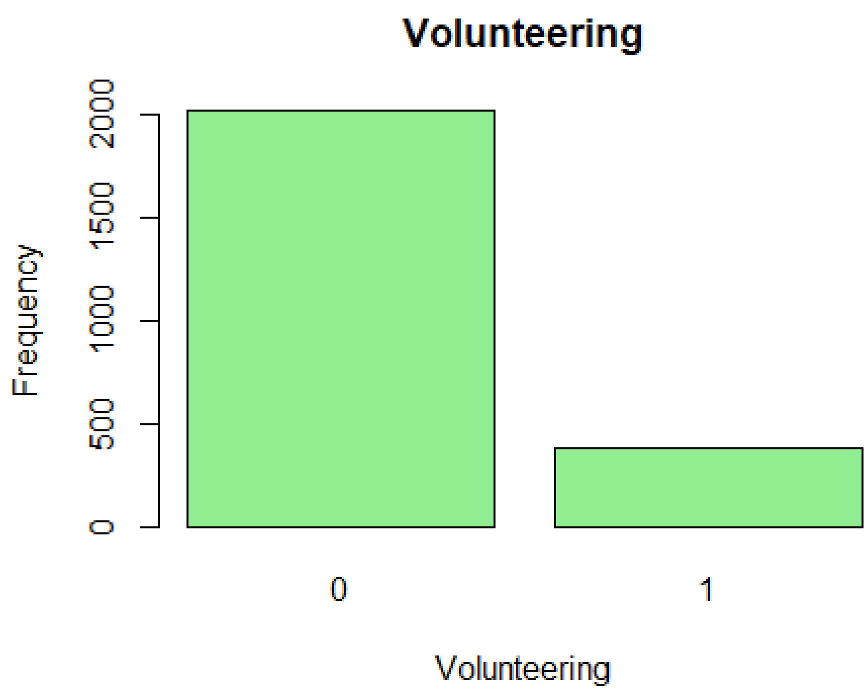

Tutoring
Frequency
0
500
1000
1500
0
1
Tutoring


ParentalSupport
Frequency
0
200
400
600
0
1
2
3
4
ParentalSupport


Extracurricular
Frequency
0
400
800
1200
0
1
Extracurricular

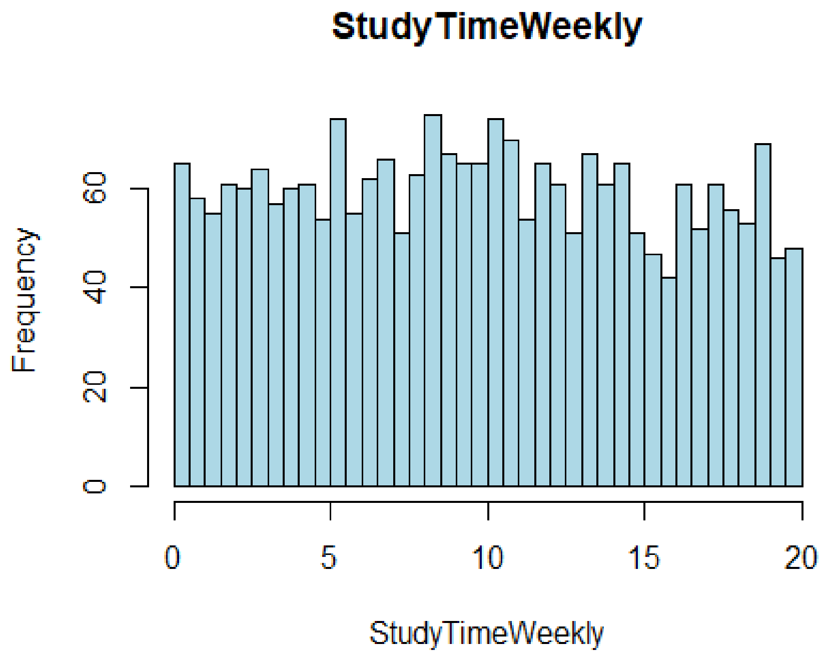

Sports
Frequency
0
500
1000
1500
0
1
Sports

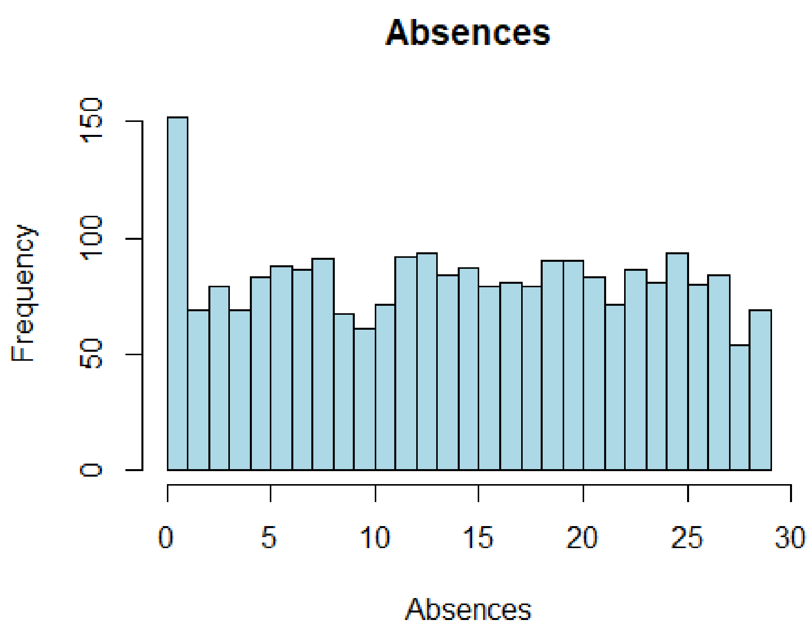


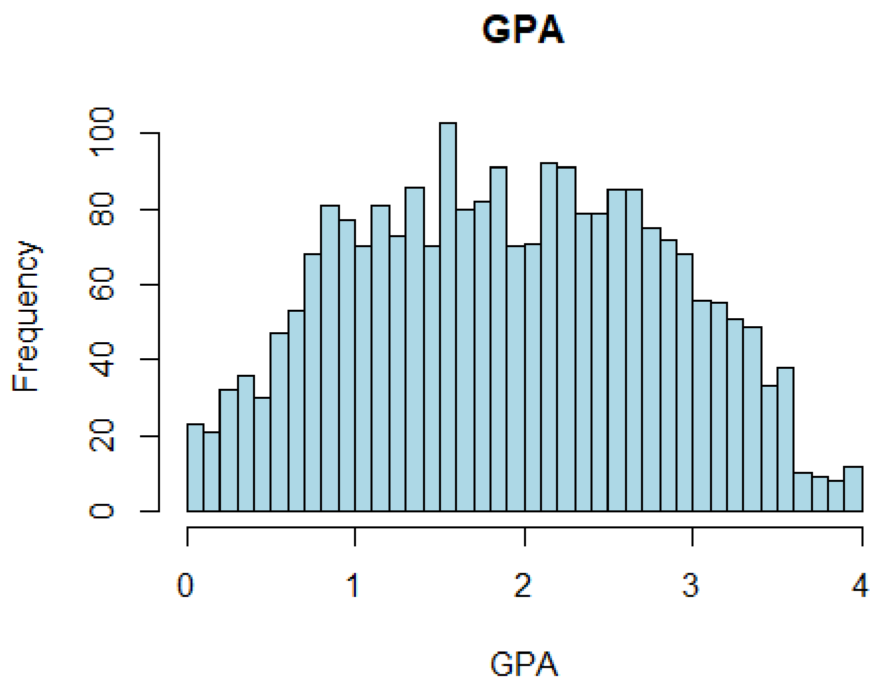



```
numerical=mydata0[,c(6,7,14)]
for(i in 1:ncol(numerical))
{
  hist(numerical[,i],main=colnames(numerical)[i],nclass=30,
```

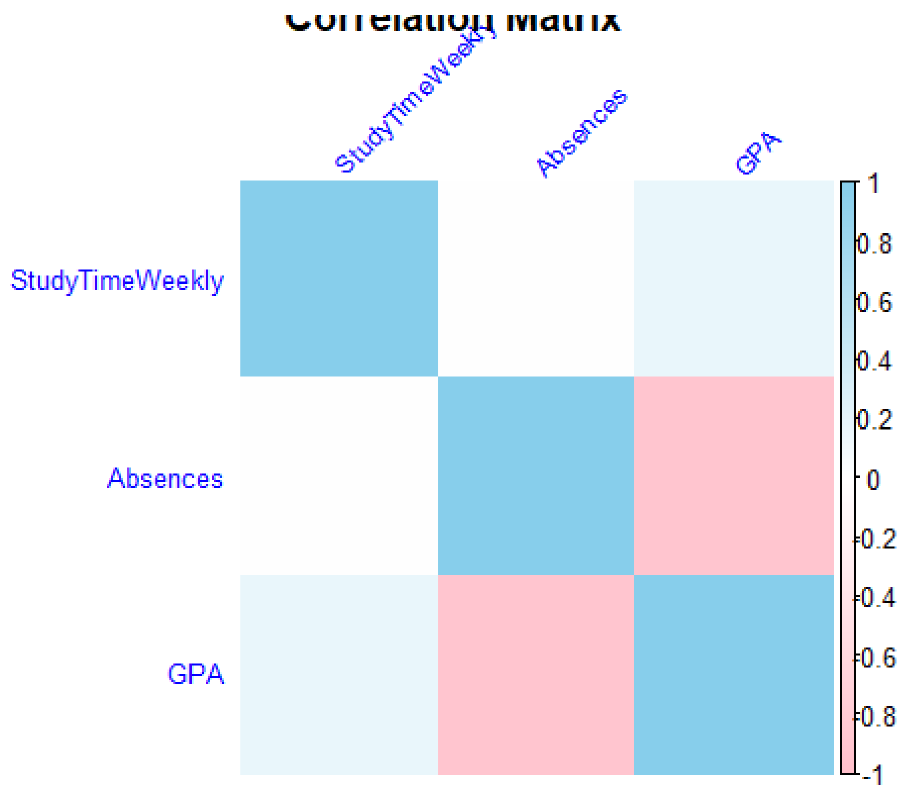
```
           xlab=colnames(numerical)[i],
           ylab="Frequency",col="lightblue")
```


StudyTimeWeekly
Frequency
0
20
40
60
0
5
10
15
20
StudyTimeWeekly

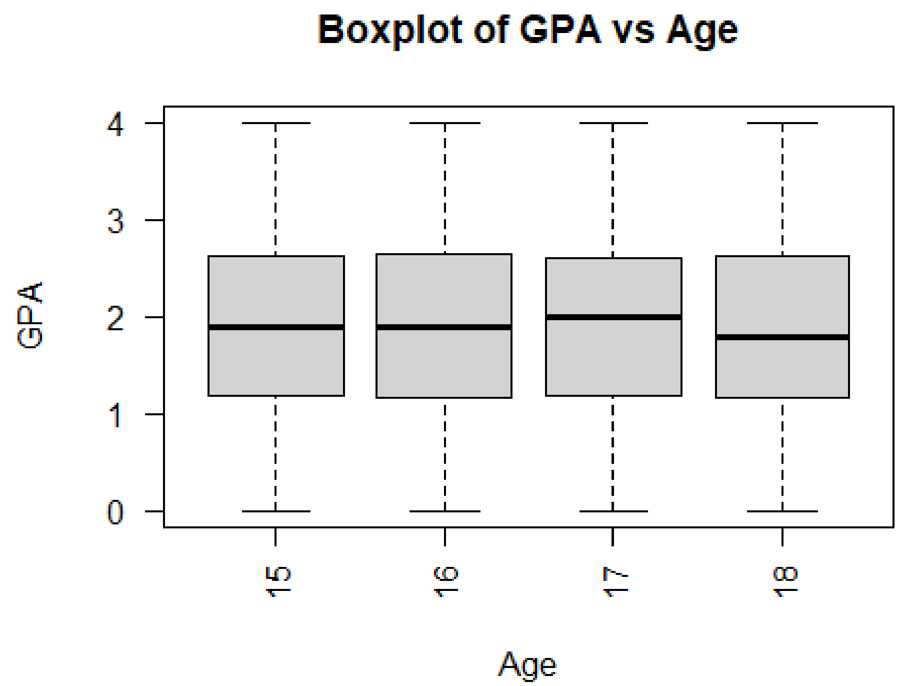

Absences
Frequency
0
50
100
150
0
5
10
15
20
25
30
Absences



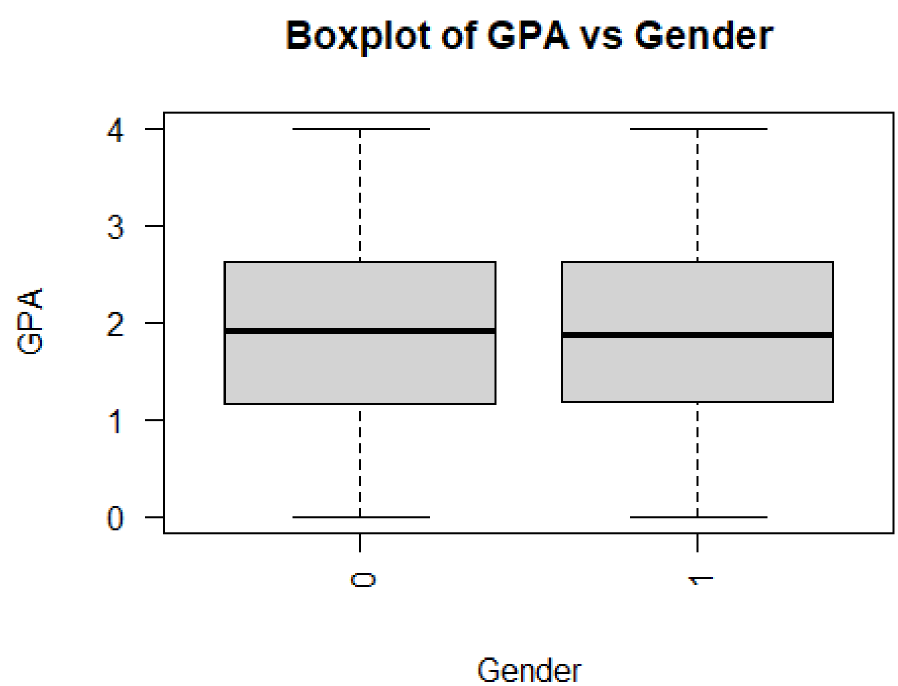

```
library(corrplot)

## Warning: package 'corrplot' was built under R version 4.4.2

## corrplot 0.95 loaded

correlation_matrix=cor(numerical, use = "complete.obs")
corrplot(correlation_matrix, method = "color",
         col = colorRampPalette(c("pink","white", "skyblue"))(200),
         tl.col = "blue",tl.cex = 0.8, tl.srt = 45, title = "Correlation Matr
ix")

## Warning in ind1:ind2: numerical expression has 2 elements: only the first
used
```



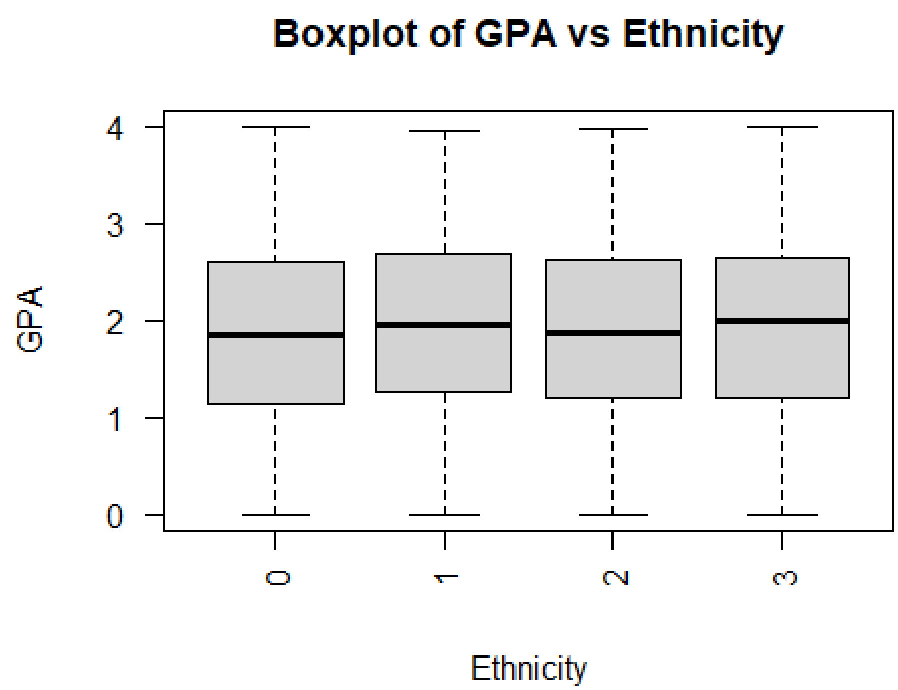

```
x=mydata0[,-c(1,6,7,15)]
for(i in 1:(ncol(x)-1))
  {
  boxplot(GPA~x[,i],data=x,main=paste("Boxplot of GPA vs",colnames(x)[i]),
          xlab=colnames(x)[i],ylab="GPA",
          las = 2,cex.names = 0.6)
  }
```

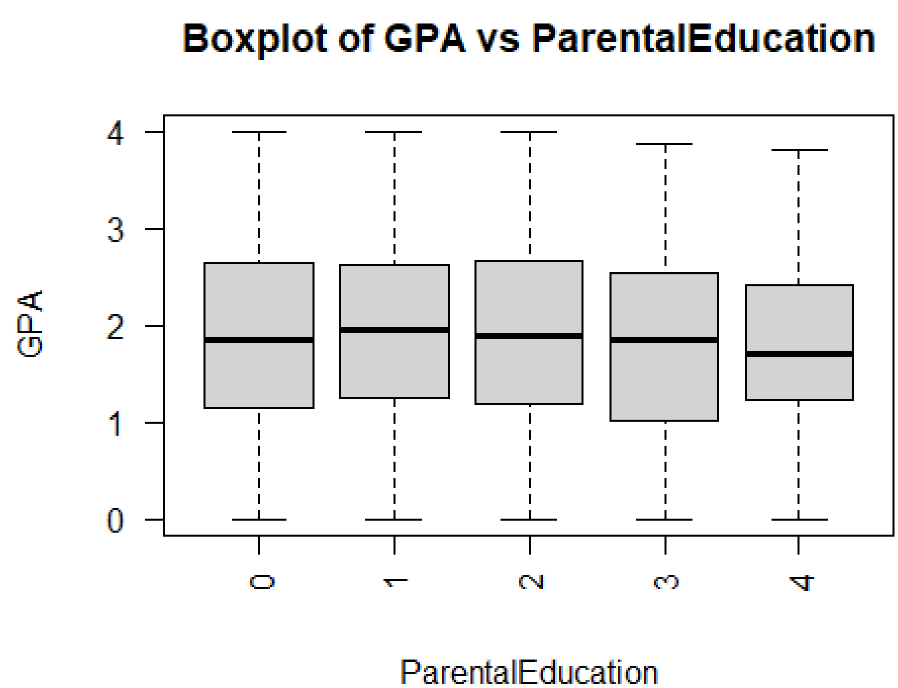


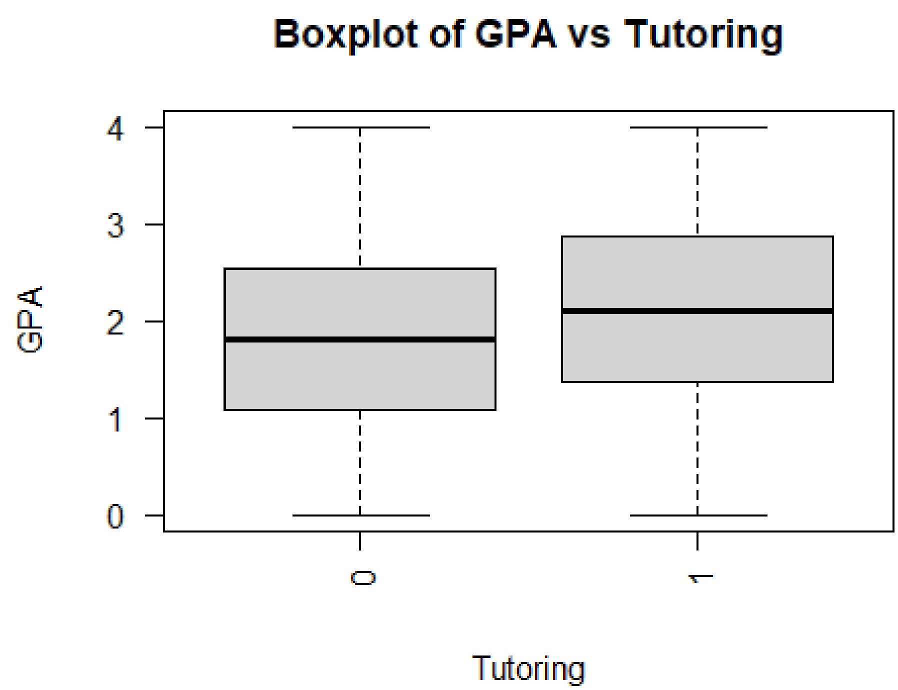
Boxplot of GPA vs Age
GPA
Age
15
16
17
18
0
1
2
3
4

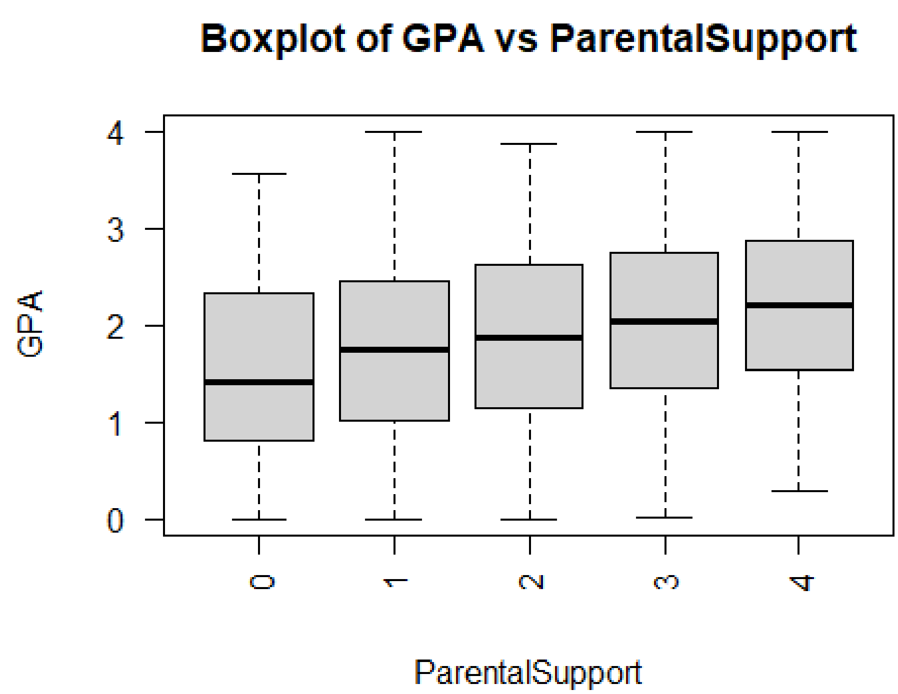


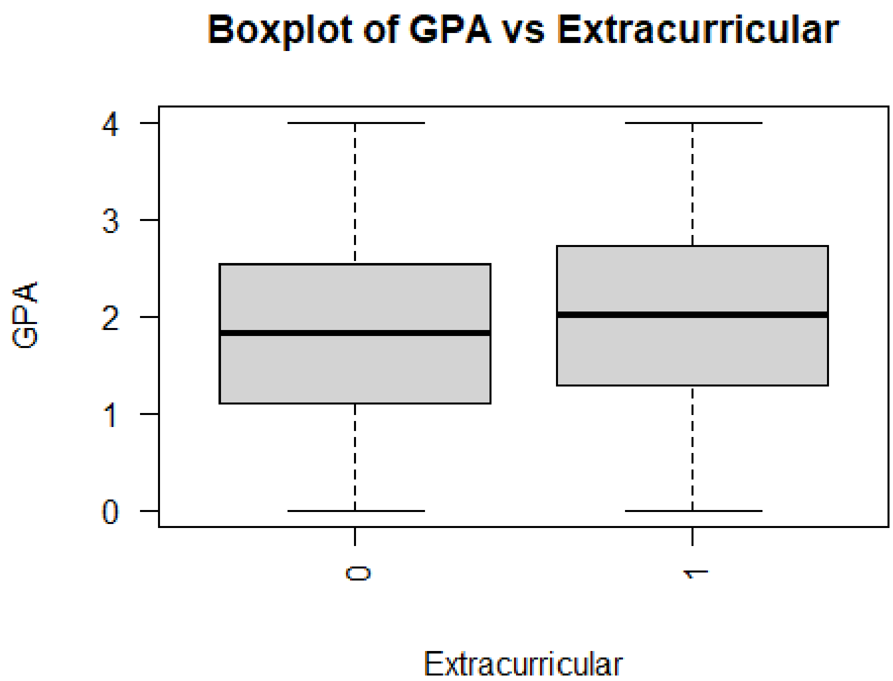
Boxplot of GPA vs Gender
GPA
Gender
0
1
0
1
2
3
4

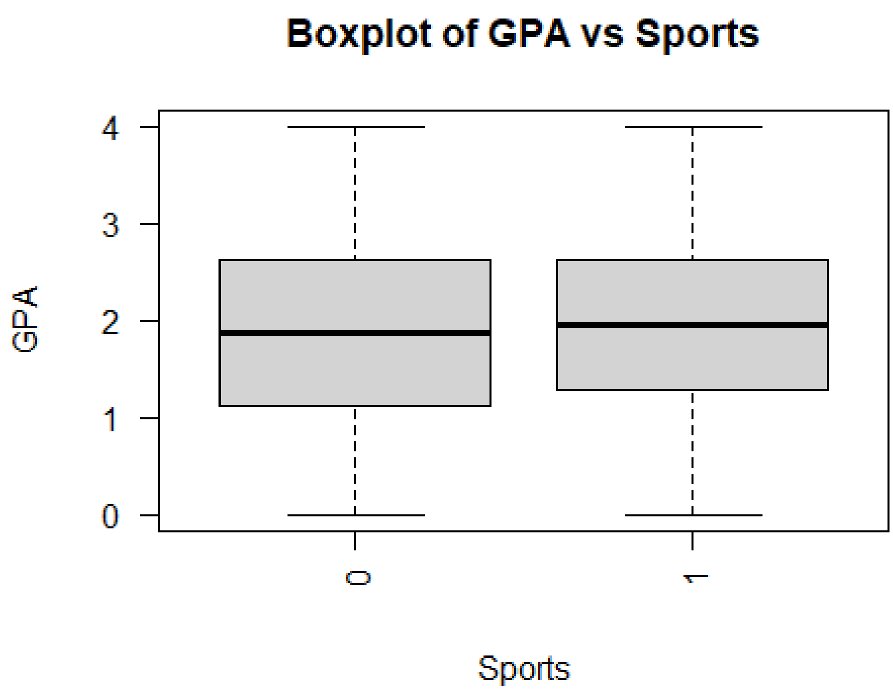

Boxplot of GPA vs Ethnicity
GPA
0
1
2
3
4
Ethnicity

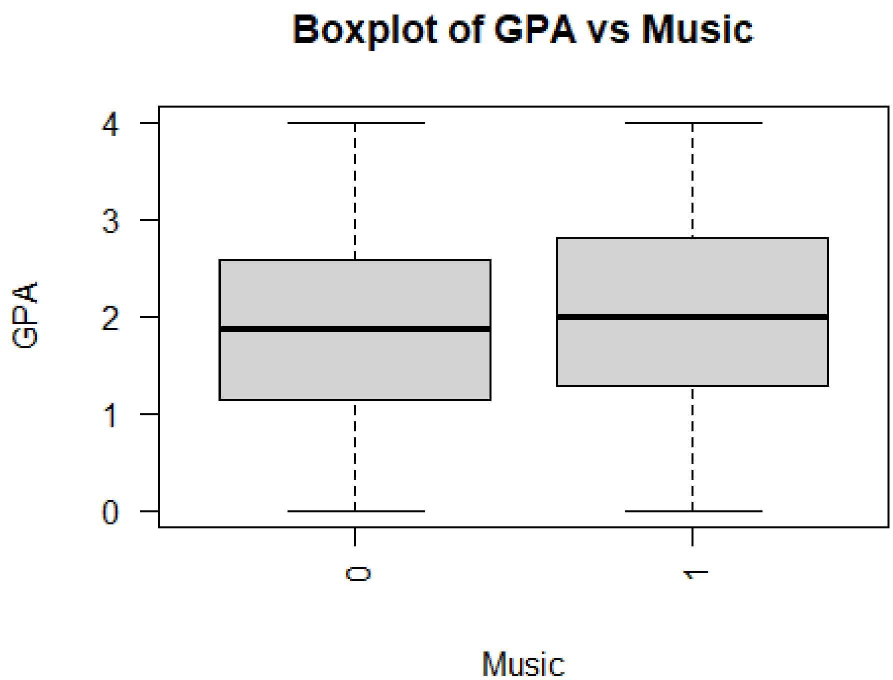

Boxplot of GPA vs ParentalEducation
GPA
0
1
2
3
4
ParentalEducation

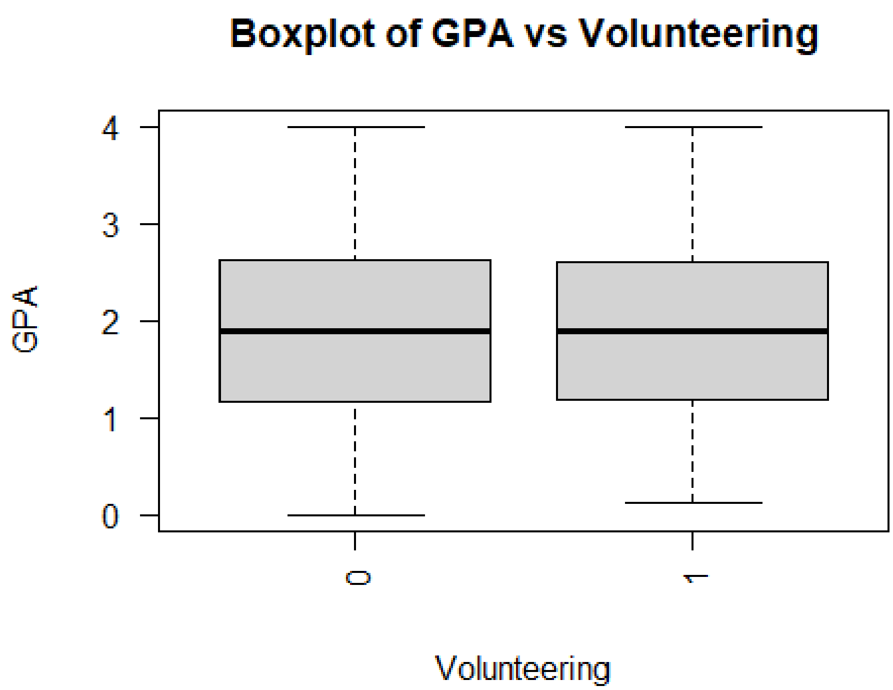

Boxplot of GPA vs Tutoring
GPA
0
1
2
3
4
Tutoring


Boxplot of GPA vs ParentalSupport
GPA
0
1
2
3
4
ParentalSupport


Boxplot of GPA vs Extracurricular
GPA
4
3
2
1
0
0
1
Extracurricular


Boxplot of GPA vs Sports
GPA
4
3
2
1
0
0
1
Sports


Boxplot of GPA vs Music
GPA
4
3
2
1
0
0
1
Music


Boxplot of GPA vs Volunteering
GPA
4
3
2
1
0
0
1
Volunteering

Model Fitting

1. Multinomial Logistic Regression

Data Preparation: Creating dummy variables and splitting dataset as train and test.

```
data1=mydata

data1$eth_caucasian=ifelse(data1$Ethnicity==0,1,0)
data1$eth_African_American=ifelse(data1$Ethnicity==1,1,0)
data1$eth_Asian=ifelse(data1$Ethnicity==2,1,0)

data1$par_ed_High_school=ifelse(data1$ParentalEducation==1,1,0)
data1$par_ed_college=ifelse(data1$ParentalEducation==2,1,0)
data1$par_ed_bachelor=ifelse(data1$ParentalEducation==3,1,0)
data1$par_ed_higher=ifelse(data1$ParentalEducation==4,1,0)

data1$Par_sup_low=ifelse(data1$ParentalSupport==1,1,0)
data1$Par_sup_moderate=ifelse(data1$ParentalSupport==2,1,0)
data1$Par_sup_high=ifelse(data1$ParentalSupport==3,1,0)
data1$Par_sup_very_high=ifelse(data1$ParentalSupport==4,1,0)

data2=data1[,-c(3,4,8)]

idx=sample(nrow(data2), size = 0.7*nrow(data2), replace = FALSE)
train=data2[idx,]
test=data2[-idx,]
```

Fitting model on train data set.

```
library(nnet)
logistic_model = multinom(GradeClass ~ ., data =data.frame(train))

## # weights:  110 (84 variable)
## initial  value 2694.199065
## iter  10 value 1722.430367
## iter  20 value 1521.268520
## iter  30 value 1379.307227
## iter  40 value 1352.495807
## iter  50 value 1342.955388
## iter  60 value 1340.478248
## iter  70 value 1339.823053
## iter  80 value 1339.676179
## iter  90 value 1339.643717
## final  value 1339.643635
## converged

summary(logistic_model)
```

```
## Call:
## multinom(formula = GradeClass ~ ., data = data.frame(train))
##
## Coefficients:
##   (Intercept)          Age     Gender StudyTimeWeekly    Absences   Tutori
ng
## B    4.002384 -0.084482272 0.09350458     -0.03530022 -0.04951313 -0.44952
46
## C    4.791461 -0.010500704 0.16533453     -0.09164578  0.11515031 -1.03544
36
## D    4.546972 -0.051529922 0.13797502     -0.13920546  0.30682392 -1.56977
40
## F    2.051251 -0.009533597 0.13112417     -0.23082973  0.63605777 -2.19351
90
##   Extracurricular     Sports      Music Volunteering eth_caucasian
## B     -0.07108413  0.1697829  0.2617197  -0.09002078    -0.7862314
## C     -0.25228041 -0.1874577 -0.2074666  -0.14942263    -0.8118651
## D     -0.79481638 -0.3136178 -0.4355552  -0.14661753    -0.8209159
## F     -1.12212270 -0.7580979 -0.5016813   0.02597506    -0.3164758
##   eth_African_American  eth_Asian par_ed_High_school par_ed_college
## B           -0.7693074 -1.4027435        0.032084705     0.08827789
## C           -0.7683193 -1.2126765        0.003273188    -0.03481707
## D           -0.5564450 -0.9027420        0.350282129     0.03850479
## F           -0.4231849 -0.6325834       -0.044366137    -0.20864999
##   par_ed_bachelor par_ed_higher Par_sup_low Par_sup_moderate Par_sup_high
## B     -0.33019108     0.8699894   0.3384141        0.7484593   -0.3003643
## C     -0.26436657     0.6646912  -0.6398722       -0.7408678   -2.1904080
## D     -0.04156643     1.2895607  -0.4163825       -0.9736310   -2.5452884
## F     -0.26916788     1.3801192  -1.1272261       -2.1247821   -4.2760893
##   Par_sup_very_high
## B        -0.9464278
## C        -2.8299110
## D        -3.1116559
## F        -5.2919763
##
## Std. Errors:
##   (Intercept)       Age    Gender StudyTimeWeekly   Absences  Tutoring
## B    2.410577 0.1278449 0.2851014      0.02653595 0.03308449 0.2887667
## C    2.329792 0.1247437 0.2797657      0.02625735 0.03134548 0.2875915
## D    2.414245 0.1299794 0.2927455      0.02774533 0.03369122 0.3085216
## F    2.603644 0.1413947 0.3180574      0.03025997 0.03851930 0.3382356
##   Extracurricular    Sports     Music Volunteering eth_caucasian
## B       0.2878931 0.3111430 0.3404571    0.3768517     0.5971815
## C       0.2825833 0.3086548 0.3408288    0.3727588     0.5914160
## D       0.2990582 0.3206466 0.3602217    0.3899449     0.6128225
## F       0.3257401 0.3478853 0.3915714    0.4252097     0.6601993
##   eth_African_American eth_Asian par_ed_High_school par_ed_college
## B            0.6455340 0.6392624          0.5123817      0.4933839
## C            0.6387088 0.6251921          0.4956124      0.4786742
## D            0.6610988 0.6460518          0.5203162      0.5061917
```

```
## F            0.7142868 0.6998570          0.5591249      0.5417717
##   par_ed_bachelor par_ed_higher Par_sup_low Par_sup_moderate Par_sup_high
## B       0.5738513      1.180251   0.9700283        0.9098058    0.8818479
## C       0.5530251      1.158186   0.8763875        0.8204472    0.7918183
## D       0.5818256      1.162634   0.9012985        0.8490933    0.8223991
## F       0.6258513      1.198893   0.9335442        0.8822694    0.8597359
##   Par_sup_very_high
## B         0.9117061
## C         0.8266716
## D         0.8612576
## F         0.9127389
##
## Residual Deviance: 2679.287
## AIC: 2847.287
```

The objective of multinomial logistic regression model is to minimize negative log likelihood function which is attained at value 1410.87

```
training_pred=predict(logistic_model, newdata = train)
training_conf_matrix=table(Predicted = training_pred, Actual = train$GradeCla
ss)
print(training_conf_matrix)
```

```
##          Actual
## Predicted   A   B   C   D   F
##         A   9   9   5   1   0
##         B  40 126  30   5   2
##         C   8  38 186  56   4
##         D   6   4  48 161  32
##         F  12  16  13  72 791
```

```
training_accuracy =sum(diag(training_conf_matrix)) / sum(training_conf_matrix
)
training_error_rate=1-training_accuracy
print(paste("training Error Rate:", round(training_error_rate*100,2),"%"))
```

```
## [1] "training Error Rate: 23.95 %"
```

```
#training_accuracy = mean(training_pred == train$GradeClass)
cat("Training_Accuracy:", training_accuracy, "\n")
```

```
## Training_Accuracy: 0.760454
```

```
# Make predictions
testing_pred=predict(logistic_model, newdata = test)
testing_conf_matrix=table(Predicted = testing_pred, Actual = test$GradeClass)
print(testing_conf_matrix)
```

```
##          Actual
## Predicted   A   B   C   D   F
##         A   0   5   2   1   0
##         B  21  34  12   1   3
```

```
##         C   3  18  66  22   2
##         D   3   5  20  57  20
##         F   5  14   9  38 357
```

```
testing_accuracy =sum(diag(testing_conf_matrix)) / sum(testing_conf_matrix)
testing_error_rate= 1-testing_accuracy
print(paste("Testing Error Rate:", round(testing_error_rate*100,2),"%"))
```

```
## [1] "Testing Error Rate: 28.41 %"
```

```
#testing_accuracy = mean(testing_pred == test$GradeClass)
cat("Testing_Accuracy:", testing_accuracy, "\n")
```

```
## Testing_Accuracy: 0.7158774
```

Variable selection:

```
library(MASS)
step=stepAIC(logistic_model,direction="both")
```

```
summary(step)$call
```

```
## multinom(formula = GradeClass ~ StudyTimeWeekly + Absences +
##     Tutoring + Extracurricular + Sports + Music + Par_sup_moderate +
##     Par_sup_high + Par_sup_very_high, data = data.frame(train))
```

```
new_logistic=multinom(formula = GradeClass ~ StudyTimeWeekly + Absences +
    Tutoring + Extracurricular + Sports + Music + par_ed_higher +
    Par_sup_low + Par_sup_moderate + Par_sup_high + Par_sup_very_high,
    data = data.frame(train))
```

```
## # weights:  65 (48 variable)
## initial  value 2694.199065
## iter  10 value 1591.913859
## iter  20 value 1373.516149
## iter  30 value 1352.901289
## iter  40 value 1349.899197
## iter  50 value 1349.208017
## final  value 1349.194235
## converged
```

```
summary(new_logistic)
```

```
## Call:
## multinom(formula = GradeClass ~ StudyTimeWeekly + Absences +
##     Tutoring + Extracurricular + Sports + Music + par_ed_higher +
##     Par_sup_low + Par_sup_moderate + Par_sup_high + Par_sup_very_high,
##     data = data.frame(train))
##
## Coefficients:
##   (Intercept) StudyTimeWeekly    Absences  Tutoring Extracurricular     Sp
orts
```

```
## B    1.580843     -0.02780384 -0.04833114 -0.449087     -0.07368919  0.108
0934
## C    3.603522     -0.08462679  0.11611936 -1.032889     -0.24925407 -0.260
5872
## D    2.984008     -0.13404452  0.30641826 -1.557989     -0.77318497 -0.373
5747
## F    1.259938     -0.22623241  0.63495833 -2.184170     -1.11666009 -0.814
5800
##        Music par_ed_higher Par_sup_low Par_sup_moderate Par_sup_high
## B  0.2742091     0.8763296   0.4542169        0.8384799   -0.1640848
## C -0.2119239     0.6776981  -0.5225228       -0.6590190   -2.0686972
## D -0.4344324     1.1252143  -0.3154435       -0.8974957   -2.4355269
## F -0.5119093     1.4275320  -0.9682236       -2.0106828   -4.1347214
##   Par_sup_very_high
## B        -0.7770625
## C        -2.6619055
## D        -2.9579594
## F        -5.0779217
##
## Std. Errors:
##   (Intercept) StudyTimeWeekly   Absences  Tutoring Extracurricular    Spor
ts
## B   0.9318091      0.02601207 0.03276366 0.2842735       0.2845906 0.30642
62
## C   0.8413955      0.02579585 0.03120948 0.2842015       0.2798943 0.30438
22
## D   0.8658642      0.02733678 0.03358725 0.3051598       0.2965241 0.31655
37
## F   0.9075125      0.02987222 0.03838097 0.3349066       0.3229807 0.34380
04
##       Music par_ed_higher Par_sup_low Par_sup_moderate Par_sup_high
## B 0.3354433      1.092915   0.9594989        0.9002224    0.8699449
## C 0.3373349      1.076004   0.8643480        0.8090939    0.7783818
## D 0.3572102      1.070625   0.8888717        0.8370161    0.8088038
## F 0.3888730      1.098757   0.9206762        0.8697888    0.8460421
##   Par_sup_very_high
## B         0.9005099
## C         0.8137335
## D         0.8481894
## F         0.8980391
##
## Residual Deviance: 2698.388
## AIC: 2794.388

training_pred=predict(new_logistic, newdata = train)
training_conf_matrix=table(Predicted = training_pred, Actual = train$GradeCla
ss)
print(training_conf_matrix)
```

```
##          Actual
## Predicted   A   B   C   D   F
##         A   6   6   6   0   0
##         B  43 127  30   5   2
##         C   8  40 189  54   6
##         D   6   6  46 161  32
##         F  12  14  11  75 789
```

```
training_accuracy =sum(diag(training_conf_matrix)) / sum(training_conf_matrix
)
training_error_rate=1-training_accuracy
print(paste("training Error Rate:", round(training_error_rate*100,2),"%"))
```

```
## [1] "training Error Rate: 24.01 %"
```

```
#training_accuracy = mean(training_pred == train$GradeClass)
cat("Training_Accuracy:", training_accuracy, "\n")
```

```
## Training_Accuracy: 0.7598566
```

```
# Make predictions
testing_pred=predict(new_logistic, newdata = test)
testing_conf_matrix=table(Predicted = testing_pred, Actual = test$GradeClass)
print(testing_conf_matrix)
```

```
##          Actual
## Predicted   A   B   C   D   F
##         A   0   2   2   1   0
##         B  20  37  12   1   3
##         C   4  18  66  20   2
##         D   3   5  20  62  16
##         F   5  14   9  35 361
```

```
testing_accuracy =sum(diag(testing_conf_matrix)) / sum(testing_conf_matrix)
testing_error_rate=1-testing_accuracy
print(paste("Testing Error Rate:", round(testing_error_rate*100,2),"%"))
```

```
## [1] "Testing Error Rate: 26.74 %"
```

```
#testing_accuracy = mean(testing_pred == test$GradeClass)
cat("Testing_Accuracy:", testing_accuracy, "\n")
```

```
## Testing_Accuracy: 0.7325905
```

```
summary(logistic_model)$AIC
```

```
## [1] 2847.287
```

```
summary(new_logistic)$AIC
```

```
## [1] 2794.388
```

Model	Training accuracy	Testing accuracy	AIC	Residual deviance
Logistic model	0.7605	0.7158	2847.287	2679.287
New Logistic model	0.7598	0.7325	2794.388	2698.388

Conclusion:

- AIC value of new model is less than model fitted using all variables. Hence the new model is better.
- Training and testing accuracy also increases for new model.

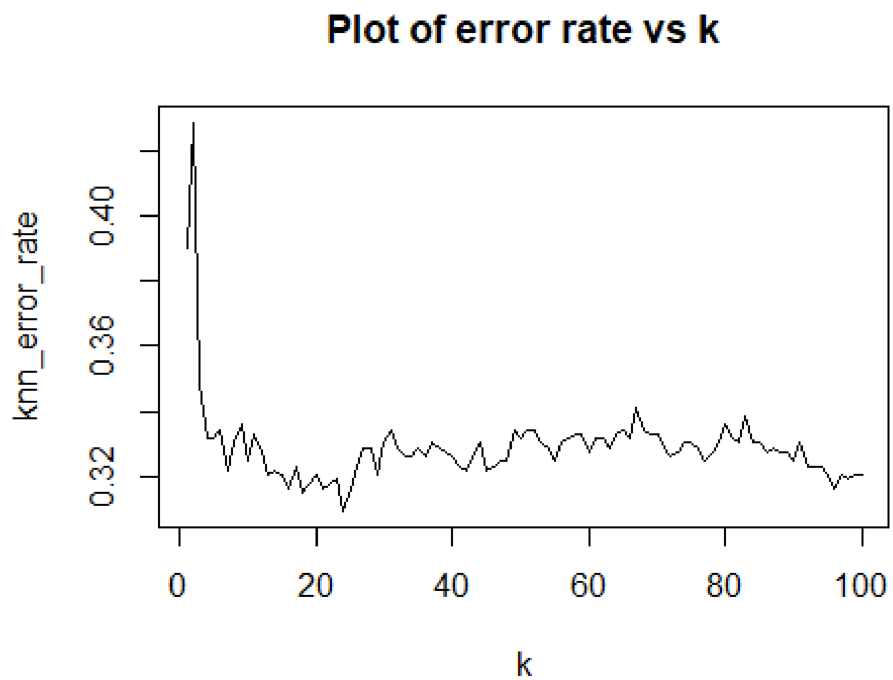
2. K- nearest neighbor classifier

data preparation and splitting

```
mydata=mydata0[,-c(1,14)]
idx=sample(nrow(mydata), size = 0.7*nrow(mydata), replace = FALSE)
train=mydata[idx,]
test=mydata[-idx,]
dim(train)

## [1] 1674   13

y_train=train[,13]
y_test=test[,13]
x_train=train[,-13]
x_test=test[,-13]
```

fitting KNN model for different values of k

```
library(class)
k=1:100
knn_accuracy=knn_error_rate=c()
for(kn in 1:100){
  knn_pred=knn(train = x_train,test = x_test,
               cl=train$GradeClass,k=kn)
  conf_matrix=table(Predicted = knn_pred, Actual = test$GradeClass)
  knn_accuracy[kn]=sum(diag(conf_matrix)) / sum(conf_matrix)
  knn_error_rate[kn]=1-knn_accuracy[kn]
}
plot(k,knn_error_rate,type='l',main="Plot of error rate vs k")
```



Tuning for best value of k

```
library(e1071)
kn=1:100
tune_knn=tune.knn(train[,-13],as.factor(train[,13]),k=kn)
summary(tune_knn)

##
## Parameter tuning of 'knn.wrapper':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   k
##  22
##
## - best performance: 0.2943201
##
## - Detailed performance results:
##       k     error dispersion
## 1     1 0.3879480 0.02753620
## 2     2 0.3871130 0.02451922
## 3     3 0.3411280 0.03593381
## 4     4 0.3285983 0.04155377
## 5     5 0.3135617 0.02625928
## 6     6 0.3177563 0.03969032
## 7     7 0.3156450 0.03362630
```

```
## 8      8 0.3160739 0.03181845
## 9      9 0.3160652 0.03037944
## 10    10 0.3093759 0.02628831
## 11    11 0.3093672 0.02400584
## 12    12 0.3139592 0.02362528
## 13    13 0.3152092 0.02633696
## 14    14 0.3101953 0.02391954
## 15    15 0.3118724 0.01659872
## 16    16 0.3068654 0.02626206
## 17    17 0.3051883 0.03155836
## 18    18 0.3072891 0.03257256
## 19    19 0.3022681 0.02682674
## 20    20 0.3081224 0.02806074
## 21    21 0.3014296 0.02713119
## 22    22 0.2943201 0.02745624
## 23    23 0.2943201 0.02766036
## 24    24 0.3022577 0.02859997
## 25    25 0.3005927 0.03241555
## 26    26 0.3018445 0.03297950
## 27    27 0.2993358 0.02567518
## 28    28 0.2993515 0.03299333
## 29    29 0.2989331 0.03029602
## 30    30 0.3006084 0.02814749
## 31    31 0.3014400 0.02843462
## 32    32 0.3014400 0.02870696
## 33    33 0.3043689 0.03468679
## 34    34 0.3043602 0.03148821
## 35    35 0.3077040 0.02979397
## 36    36 0.3097960 0.03637566
## 37    37 0.3064435 0.03832237
## 38    38 0.3093759 0.03610164
## 39    39 0.3102075 0.03566634
## 40    40 0.3072821 0.03245561
## 41    41 0.3093741 0.03055379
## 42    42 0.3072856 0.03139352
## 43    43 0.3047751 0.03272813
## 44    44 0.3047751 0.03052203
## 45    45 0.3064470 0.03386608
## 46    46 0.3051953 0.03289509
## 47    47 0.3102127 0.03270456
## 48    48 0.3077092 0.03355284
## 49    49 0.3056172 0.03378178
## 50    50 0.3068706 0.03683568
## 51    51 0.3093863 0.03574668
## 52    52 0.3052057 0.03537916
## 53    53 0.3110600 0.03647930
## 54    54 0.3102249 0.03642444
## 55    55 0.3089714 0.03806960
## 56    56 0.3089679 0.04079832
## 57    57 0.3072960 0.03695758
```

```
## 58   58 0.3056224 0.03782423
## 59   59 0.3068741 0.03573872
## 60   60 0.3072943 0.03698451
## 61   61 0.3068776 0.03741497
## 62   62 0.3068776 0.03657368
## 63   63 0.3077127 0.04006486
## 64   64 0.3018602 0.03551804
## 65   65 0.3031189 0.03569613
## 66   66 0.3081311 0.03713742
## 67   67 0.3110617 0.03425082
## 68   68 0.3064575 0.03503136
## 69   69 0.3072960 0.03673852
## 70   70 0.3085495 0.03791314
## 71   71 0.3064627 0.03695978
## 72   72 0.3035356 0.03764228
## 73   73 0.3089662 0.03421975
## 74   74 0.3093846 0.03431098
## 75   75 0.3089662 0.03329782
## 76   76 0.3085478 0.03383634
## 77   77 0.3064557 0.03290158
## 78   78 0.3043654 0.03186833
## 79   79 0.3085478 0.03337327
## 80   80 0.3060321 0.03312588
## 81   81 0.3047786 0.03287091
## 82   82 0.3068724 0.03271590
## 83   83 0.3081259 0.03415159
## 84   84 0.3060356 0.03367267
## 85   85 0.3077075 0.03443101
## 86   86 0.3077040 0.03433289
## 87   87 0.3072856 0.03394396
## 88   88 0.3081224 0.03587928
## 89   89 0.3081241 0.03717707
## 90   90 0.3097978 0.03839372
## 91   91 0.3102144 0.03834502
## 92   92 0.3089575 0.03830882
## 93   93 0.3077022 0.03680184
## 94   94 0.3093759 0.03405876
## 95   95 0.3077040 0.03379030
## 96   96 0.3077022 0.03405667
## 97   97 0.3072856 0.03563075
## 98   98 0.3085391 0.03444624
## 99   99 0.3110513 0.03210595
## 100 100 0.3093724 0.03372986
```

```r
paste("Best value of k is",tune_knn$best.parameters)
```

```
## [1] "Best value of k is 22"
```

Prediction for test data

```r
knn_pred_test=knn(train = train[,-13],test =test[,-13],
             cl=train$GradeClass,k=tune_knn$best.parameters$k)
conf_matrix_test=table(Predicted = knn_pred_test, Actual = test$GradeClass)
conf_matrix_test

##            Actual
## Predicted   A   B   C   D   F
##          A   3   2   0   0   0
##          B  13  39  20   2   1
##          C   7  38  66  29   4
##          D   1   2  23  63  16
##          F   5  13   9  40 322

knn_test_accuracy=sum(diag(conf_matrix_test)) / sum(conf_matrix_test)
knn_test_error_rate=1-knn_test_accuracy
knn_test_error_rate

## [1] 0.3133705
```

Model	Best value of K	Testing Accuracy
KNN	22	0.6866

3. Support Vector Machine

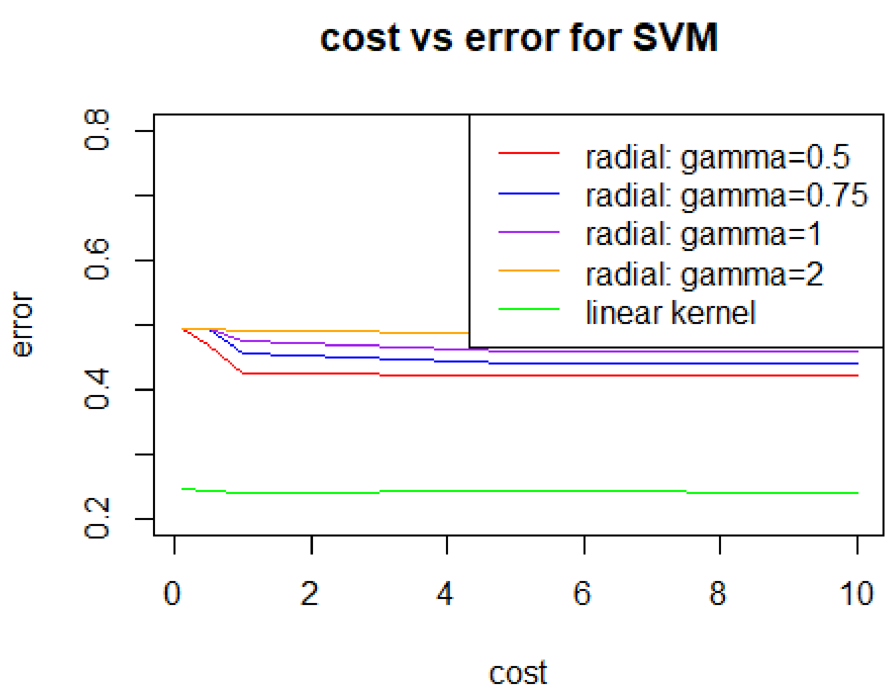
Finding optimum value of parameters (cost, gamma) for radial kernel

```
library(e1071)
### Tuning for radial kernel
set.seed(1)
tune.out_radial=tune(svm, as.factor(GradeClass)~., data = train,
                       kernel = "radial",
                       ranges = list(
                         cost = c(0.1,0.5, 1,5, 10),
                         gamma = c(0.5,0.75,1,2)
                       )
                     )
summary(tune.out_radial)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     5   0.5
##
## - best performance: 0.4209571
##
## - Detailed performance results:
##    cost gamma     error dispersion
## 1   0.1  0.50 0.4937134 0.02800566
## 2   0.5  0.50 0.4698815 0.03363581
## 3   1.0  0.50 0.4238790 0.03582883
## 4   5.0  0.50 0.4209571 0.03187642
## 5  10.0  0.50 0.4217974 0.02836011
## 6   0.1  0.75 0.4937134 0.02800566
## 7   0.5  0.75 0.4928783 0.02739753
## 8   1.0  0.75 0.4564906 0.03565685
## 9   5.0  0.75 0.4410303 0.02753332
## 10 10.0  0.75 0.4414487 0.02718091
## 11  0.1  1.00 0.4937134 0.02800566
## 12  0.5  1.00 0.4937134 0.02800566
## 13  1.0  1.00 0.4748937 0.03089951
## 14  5.0  1.00 0.4602545 0.03191472
## 15 10.0  1.00 0.4602545 0.03129929
## 16  0.1  2.00 0.4937134 0.02800566
## 17  0.5  2.00 0.4937134 0.02800566
## 18  1.0  2.00 0.4899442 0.02885234
```

```
## 19  5.0  2.00 0.4857636 0.02831151
## 20 10.0  2.00 0.4857636 0.02831151

tune.out_radial$best.parameters

##   cost gamma
## 4    5   0.5

tune.out_radial$best.performance

## [1] 0.4209571

### Tuning for linear kernel
tune.out_linear=tune(svm, as.factor(GradeClass)~., data = train,
                       kernel = "linear",
                       ranges = list(cost = c(0.1,0.5, 1,5, 10))
                     )

summary(tune.out_linear)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.2399808
##
## - Detailed performance results:
##   cost     error dispersion
## 1  0.1 0.2462517 0.03300885
## 2  0.5 0.2429079 0.03763766
## 3  1.0 0.2399808 0.03322843
## 4  5.0 0.2433264 0.03047065
## 5 10.0 0.2416527 0.03020263

tune.out_linear$best.parameters

##   cost
## 3    1

tune.out_linear$best.performance

## [1] 0.2399808

### plots
plot(subset(tune.out_radial$performances,tune.out_radial$performances$gamma==
0.5)$cost,subset(tune.out_radial$performances,tune.out_radial$performances$ga
mma==0.5)$error,type="l",ylim=c(0.2,0.8),col="red",lwd=1.5,main="cost vs erro
```

```
r for SVM",xlab="cost",ylab="error")

lines(subset(tune.out_radial$performances,tune.out_radial$performances$gamma=
=0.75)$cost,subset(tune.out_radial$performances,tune.out_radial$performances$
gamma==0.75)$error,type="l",col="blue",lwd=1.5)

lines(subset(tune.out_radial$performances,tune.out_radial$performances$gamma=
=1)$cost,subset(tune.out_radial$performances,tune.out_radial$performances$gam
ma==1)$error,type="l",col="purple",lwd=1.5)

lines(subset(tune.out_radial$performances,tune.out_radial$performances$gamma=
=2)$cost,subset(tune.out_radial$performances,tune.out_radial$performances$gam
ma==2)$error,type="l",col="orange",lwd=1.5)

lines(tune.out_linear$performances$cost,tune.out_linear$performances$error,ty
pe="l",col="green",lwd=1.5)

legend("topright",c("radial: gamma=0.5","radial: gamma=0.75","radial: gamma=1
","radial: gamma=2","linear kernel"),col=c("red","blue","purple","orange","gr
een"),lty=rep(1,4))
```



Kernel	Best Cost	Best Gamma	Performance (Error)
Radial	5	0.5	0.4209
Linear	1	-	0.23998

```
### svm for validation
svmfit=svm(as.factor(GradeClass)~., data = train,
           kernel = "linear",
           cost = 1)
summary(svmfit)
```

```
##
## Call:
## svm(formula = as.factor(GradeClass) ~ ., data = train, kernel = "linear",
##     cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  962
##
##  ( 169 78 280 266 169 )
##
##
## Number of Classes:  5
##
## Levels:
##  A B C D F
```

```
train_err=1-sum(diag(table(svmfit$fitted, train$GradeClass)))/sum(table(svmfi
t$fitted,train$GradeClass))
test_conf_matrix=table(predict(svmfit,test),test$GradeClass)
test_err=1-sum(diag(test_conf_matrix))/sum(test_conf_matrix)
paste("training error:",train_err)
```

```
## [1] "training error: 0.228793309438471"
```

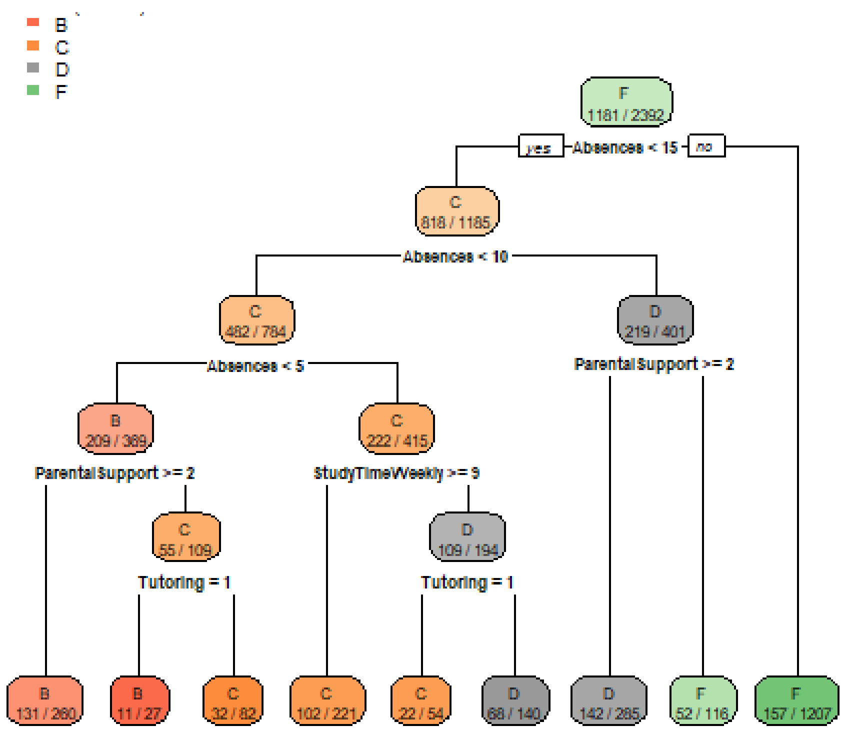
```
paste("testing error:",test_err)
```

```
## [1] "testing error: 0.268802228412256"
```

4.Decision Tree Classifier

Fitting a decision tree

```
library(tree)

library(rpart)
library(rpart.plot)

tree=rpart(as.factor(mydata$GradeClass)~.,data=mydata)
rpart.plot(tree,extra=3)
```

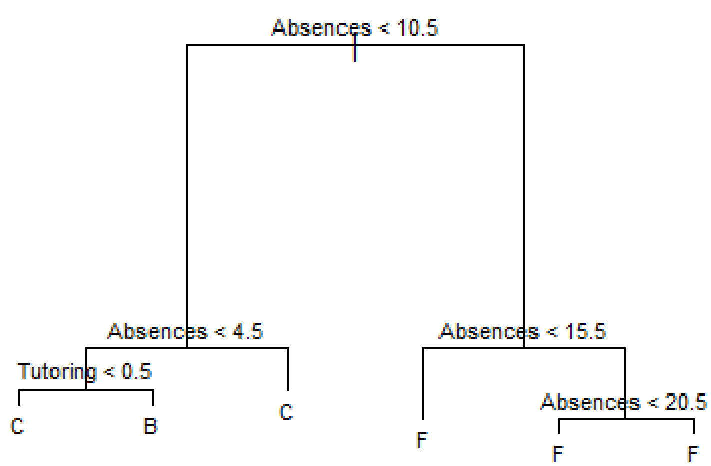



```
print(tree)

## n= 2392
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 2392 1181 F (0.045 0.11 0.16 0.17 0.51)
##    2) Absences< 14.5 1185  818 C (0.077 0.21 0.31 0.27 0.14)
##      4) Absences< 9.5 784  482 C (0.11 0.29 0.39 0.18 0.033)
##        8) Absences< 4.5 369  209 B (0.19 0.43 0.3 0.06 0.019)
##         16) ParentalSupport>=1.5 260  131 B (0.23 0.5 0.21 0.038 0.019) *
##         17) ParentalSupport< 1.5 109   55 C (0.092 0.28 0.5 0.11 0.018)
##           34) Tutoring>=0.5 27   11 B (0.26 0.59 0.15 0 0) *
##           35) Tutoring< 0.5 82   32 C (0.037 0.18 0.61 0.15 0.024) *
```

```
##        9) Absences>=4.5 415  222 C (0.031 0.17 0.47 0.29 0.046)
##         18) StudyTimeWeekly>=8.957582 221  102 C (0.036 0.24 0.54 0.15 0.0
27) *
##         19) StudyTimeWeekly< 8.957582 194  109 D (0.026 0.088 0.38 0.44 0.
067)
##           38) Tutoring>=0.5 54   22 C (0.019 0.15 0.59 0.24 0) *
##           39) Tutoring< 0.5 140   68 D (0.029 0.064 0.3 0.51 0.093) *
##      5) Absences>=9.5 401  219 D (0.017 0.03 0.16 0.45 0.34)
##       10) ParentalSupport>=1.5 285  142 D (0.018 0.039 0.19 0.5 0.25) *
##       11) ParentalSupport< 1.5 116   52 F (0.017 0.0086 0.086 0.34 0.55) *
##    3) Absences>=14.5 1207  157 F (0.013 0.022 0.02 0.075 0.87) *
```

fitting decision tree on train data

```
#Training tree model
train_tree=tree(as.factor(train$GradeClass)~.,data = train)
plot(train_tree)
text(train_tree,cex=0.7)
```



prediction and accuracy:

```
#Tree prediction on test
pr_test_tree=predict(train_tree,newdata=test,type ='class')
pr_train_tree=predict(train_tree,newdata=train,type="class")

#confusion matrix
a=table(pr_test_tree,test$GradeClass)
```

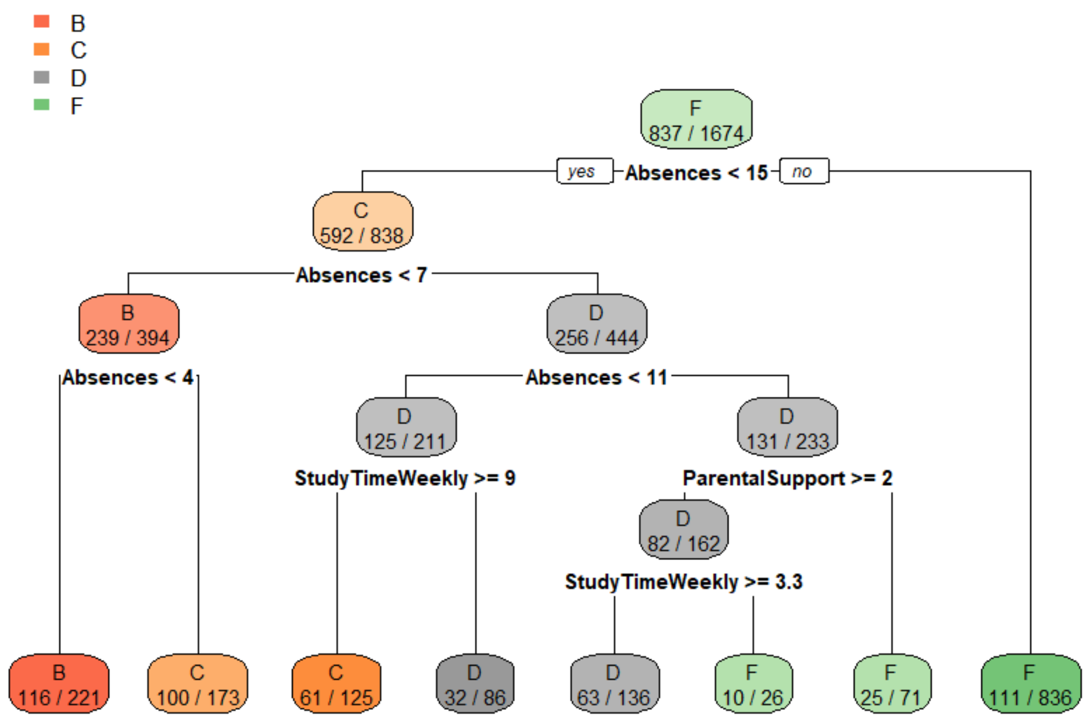
```
test_accuracy=sum(diag(a))/sum(a)
paste("test accuracy:",test_accuracy)

## [1] "test accuracy: 0.61142061281337"

b=table(pr_train_tree,train$GradeClass)
train_accuracy=sum(diag(b))/sum(b)
paste("train accuracy:",train_accuracy)

## [1] "train accuracy: 0.663679808841099"
```

Fitting models using gini index and entropy as splitting criteria.

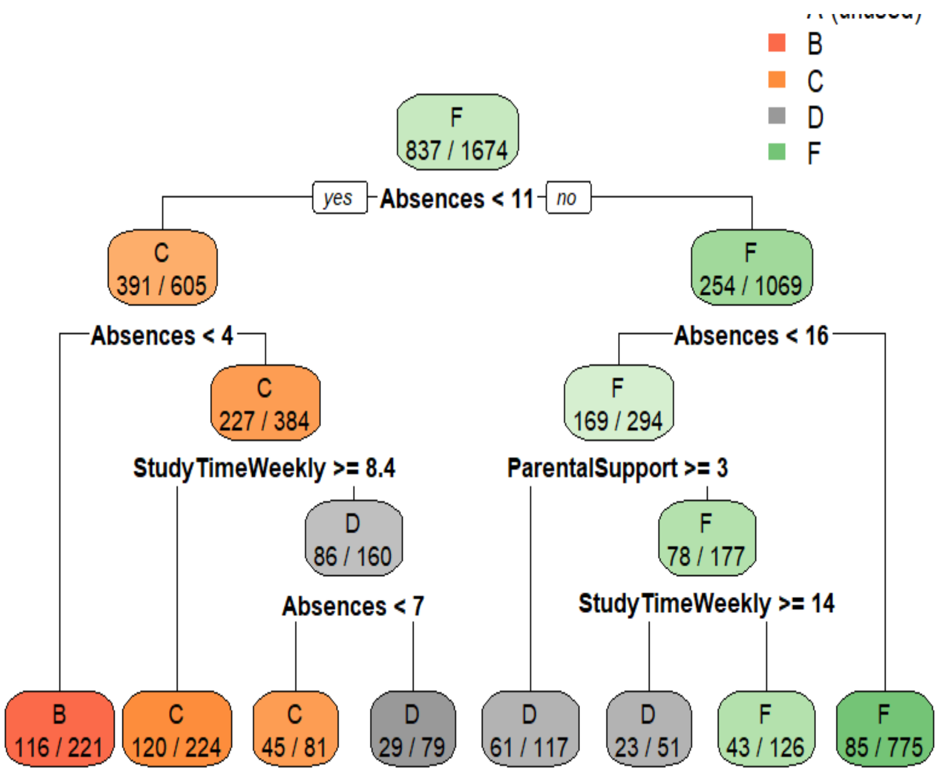
```
tree_model_gini=rpart(as.factor(mydata$GradeClass) ~ ., data = train, parms =
list(split = "gini"))


tree_model_entropy=rpart(as.factor(GradeClass) ~ ., data = train, parms = lis
t(split = "information"))
```

Pruning tree

```
prune.train_tree_gini=prune(tree_model_gini,cp=0.010000)
rpart.plot(prune.train_tree_gini,extra=3)
```



```r
prune.train_tree_entropy=prune(tree_model_entropy,cp=0.010000)
rpart.plot(prune.train_tree_entropy,extra=3)
```




```r
prune.test_tree_gini=predict(prune.train_tree_gini,test,type ='class')
confusion_matrix_gini=as.matrix(table(prune.test_tree_gini,test$GradeClass))
accuracy_gini=sum(diag(confusion_matrix_gini))/sum(confusion_matrix_gini);acc
uracy_gini

## [1] 0.6824513

prune.test_tree_entropy=predict(prune.train_tree_entropy,test,type ='class')
confusion_matrix_entropy=as.matrix(table(prune.test_tree_entropy,test$GradeCl
ass))
accuracy_entropy=sum(diag(confusion_matrix_entropy))/sum(confusion_matrix_ent
ropy);accuracy_entropy

## [1] 0.6713092
```

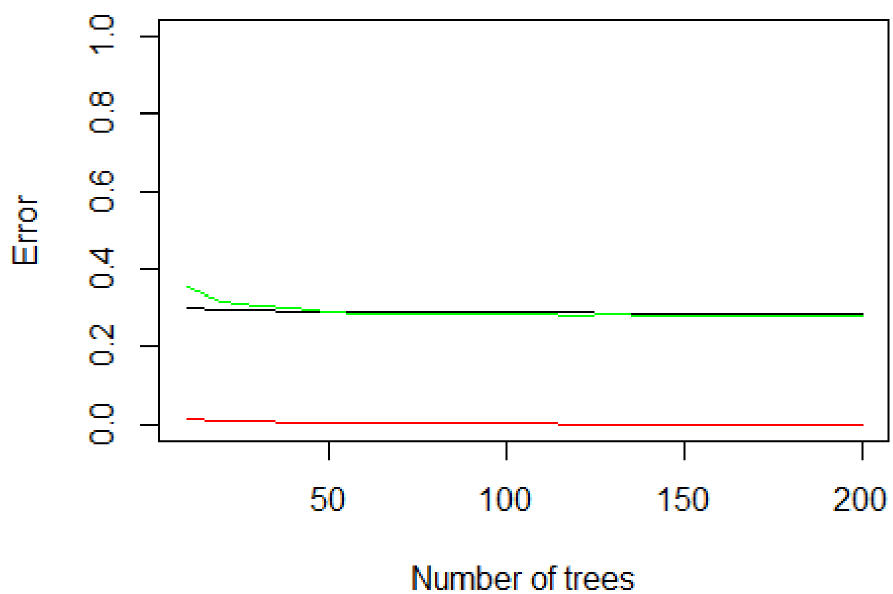
Splitting criterion	Cp for pruning	Accuracy
Gini index	0.01	0.6824
Entropy	0.01	0.6712

5.Bagging

K fold cv for bagging.

```
library(caret)

library(randomForest)
k = 10
set.seed(123)
folds=createFolds(mydata$GradeClass, k = k, list = TRUE, returnTrain = FALSE)
train_error_rates=test_error_rates=c()
Mean_train_Error_Rate=Mean_test_Error_Rate=c()
oob=c()
nseq=seq(10,200,by=10)
# Perform k-fold cross-validation
for (p in 1:length(nseq))
  {
  for(i in 1:k )
    {
    test_indices = folds[[i]]
    x_train_cv = mydata[-test_indices, ]
    x_test_cv = mydata[test_indices, ]
    bag = randomForest(as.factor(GradeClass) ~ ., data = x_train_cv,
                        mtry = ncol(x_train_cv) - 1, ntree = nseq[p])
    #train
    train_pred_bag=predict(bag, newdata = x_train_cv)
    train_conf_matrix_bag=table(x_train_cv$GradeClass, train_pred_bag)
    train_error_rate=1 -sum(diag(train_conf_matrix_bag)) / sum(train_conf_mat
rix_bag)
    train_error_rates=c(train_error_rates, train_error_rate)
    #test
    test_pred_bag=predict(bag, newdata = x_test_cv)
    test_conf_matrix_bag=table(x_test_cv$GradeClass, test_pred_bag)
    test_error_rate=1 -sum(diag(test_conf_matrix_bag)) / sum(test_conf_matrix
_bag)
    test_error_rates=c(test_error_rates, test_error_rate)

  }
  Mean_train_Error_Rate[p]= mean(train_error_rates)
  Mean_test_Error_Rate[p]= mean(test_error_rates)

  bag_oob=randomForest(as.factor(GradeClass) ~ ., data = mydata,
                        mtry = ncol(mydata) - 1, ntree = nseq[p])
  oob[p]=bag_oob$err.rate[bag_oob$ntree,1]
}

plot(nseq,Mean_train_Error_Rate,type="l",col="red",ylim=c(0,1),lwd=1.5,xlab="
```

```r
Number of trees",ylab="Error")
lines(nseq,Mean_test_Error_Rate,type="l",col="black",lwd=1.5)
lines(nseq,oob,type="l",col="green",lwd=1.5)
```



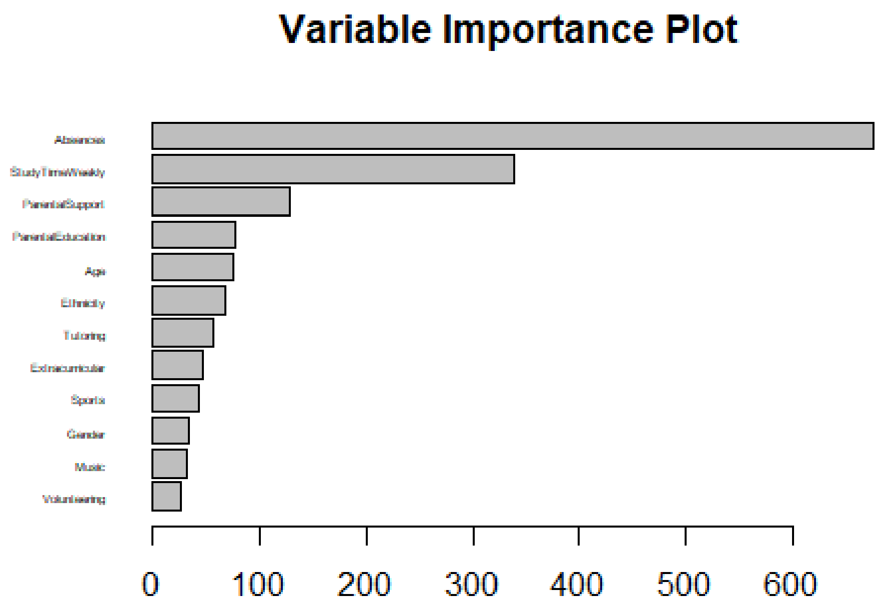
To get optimal number of trees.

```r
tune_bag=tune.randomForest(as.factor(GradeClass)~.,data=train,
                    ntree=seq(10,200,by=10),mtry=ncol(train)-1)
summary(tune_bag)

##
## Parameter tuning of 'randomForest':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  mtry ntree
##    12   190
##
## - best performance: 0.2742591
##
## - Detailed performance results:
##    mtry ntree     error dispersion
## 1    12    10 0.2993602 0.03424495
## 2    12    20 0.2847106 0.02783673
## 3    12    30 0.2813720 0.03362080
## 4    12    40 0.2847089 0.02897221
```

```
## 5    12    50 0.2822106 0.03433976
## 6    12    60 0.2817957 0.03486036
## 7    12    70 0.2792852 0.02869607
## 8    12    80 0.2863842 0.03180385
## 9    12    90 0.2788563 0.02329375
## 10   12   100 0.2830457 0.03464706
## 11   12   110 0.2817957 0.03445816
## 12   12   120 0.2838755 0.02940360
## 13   12   130 0.2801203 0.03322604
## 14   12   140 0.2830370 0.02820180
## 15   12   150 0.2817904 0.02787687
## 16   12   160 0.2780230 0.02518271
## 17   12   170 0.2809554 0.02784100
## 18   12   180 0.2859693 0.02119490
## 19   12   190 0.2742591 0.02855679
## 20   12   200 0.2801116 0.02468123
```

For validation

```
bag_final_train=randomForest(as.factor(GradeClass)~.,data=train,
                             mtry=12,ntree=tune_bag$best.parameters$ntree)
print(bag_final_train)

##
## Call:
##  randomForest(formula = as.factor(GradeClass) ~ ., data = train,      mtry
= 12, ntree = tune_bag$best.parameters$ntree)
##                Type of random forest: classification
##                      Number of trees: 190
## No. of variables tried at each split: 12
##
##         OOB estimate of  error rate: 27.78%
## Confusion matrix:
##    A  B   C   D   F class.error
## A 22 32   4   7  13  0.71794872
## B 13 85  52   9  16  0.51428571
## C  0 28 164  66  15  0.39926740
## D  1  8  63 137  71  0.51071429
## F  2  3   6  56 801  0.07718894

test_pred_bag_final=predict(bag_final_train, newdata =test)
test_conf_matrix_bag_final=table(test$GradeClass, test_pred_bag_final)
test_error_rate_bag_final=1 -sum(diag(test_conf_matrix_bag_final)) / sum(test
_conf_matrix_bag_final)
test_error_rate_bag_final

## [1] 0.2980501

barplot(sort(importance(bag_final)[,1]),horiz=T,
        las=1,cex.names = 0.4,main = "Variable Importance Plot")
```



Method	Best n tree	OOB error	Testing error
Bagging	190	0.2778	0.2980

6.Random Forest

For different number of trees, plot of error rate of random forest model with different mtry values.

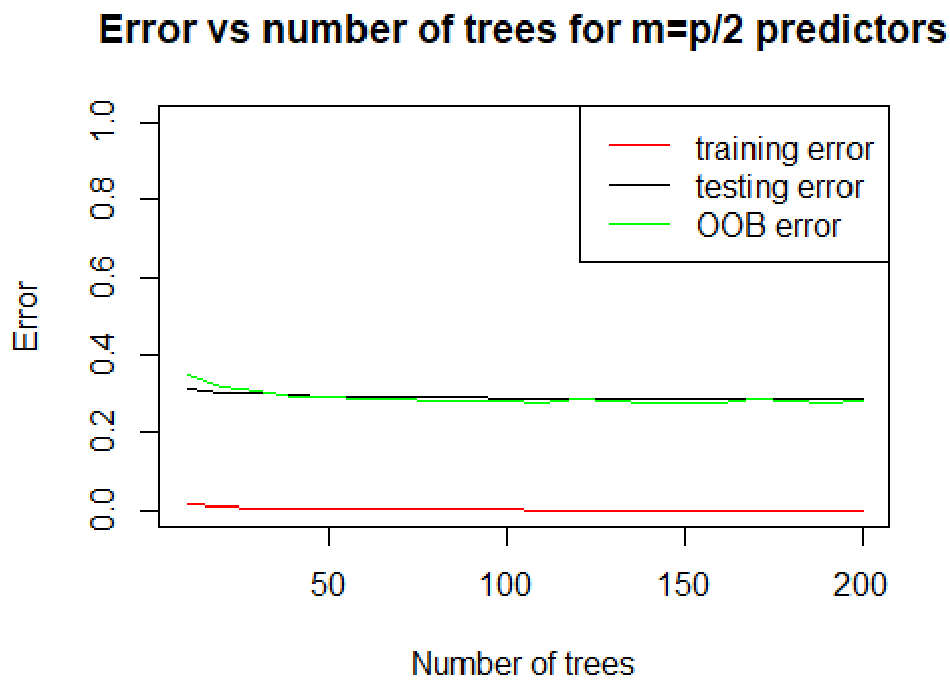
```
k = 5
set.seed(123)
folds=createFolds(mydata$GradeClass, k = k, list = TRUE, returnTrain = FALSE)
train_error_rates_rf1=train_error_rates_rf2=train_error_rates_rf3=train_error
_rates_rf4=c()
test_error_rates_rf1=test_error_rates_rf2=test_error_rates_rf3=test_error_rat
es_rf4=c()
Mean_train_Error_Rate_rf1=Mean_train_Error_Rate_rf2=Mean_train_Error_Rate_rf3
=Mean_train_Error_Rate_rf4=c()
Mean_test_Error_Rate_rf1=Mean_test_Error_Rate_rf2=Mean_test_Error_Rate_rf3=Me
an_test_Error_Rate_rf4=c()
oob_rf1=oob_rf2=oob_rf3=oob_rf4=c()
nseq=seq(10,200,by=10)
# Perform k-fold cross-validation
for (p in 1:length(nseq))
{
  for(i in 1:k )
  {
    test_indices = folds[[i]]
    x_train_cv = mydata[-test_indices, ]
    x_test_cv = mydata[test_indices, ]

    #*** model1
    rf1 = randomForest(as.factor(GradeClass) ~ ., data = x_train_cv,
                       mtry = (ncol(x_train_cv)-1)/2, ntree = nseq[p])
    #train
    train_pred_rf1=predict(rf1, newdata = x_train_cv)
    train_conf_matrix_rf1=table(x_train_cv$GradeClass, train_pred_rf1)
    train_error_rate_rf1=1 -sum(diag(train_conf_matrix_rf1)) / sum(train_conf
_matrix_rf1)
    train_error_rates_rf1=c(train_error_rates_rf1, train_error_rate_rf1)
    #test
    test_pred_rf1=predict(rf1, newdata = x_test_cv)
    test_conf_matrix_rf1=table(x_test_cv$GradeClass, test_pred_rf1)
    test_error_rate_rf1=1 -sum(diag(test_conf_matrix_rf1)) / sum(test_conf_ma
trix_rf1)
    test_error_rates_rf1=c(test_error_rates_rf1, test_error_rate_rf1)


    #*** model2
    rf2 = randomForest(as.factor(GradeClass) ~ ., data = x_train_cv,
                       mtry = sqrt(ncol(x_train_cv)-1), ntree = nseq[p])
    #train
    train_pred_rf2=predict(rf2, newdata = x_train_cv)
```

```
    train_conf_matrix_rf2=table(x_train_cv$GradeClass, train_pred_rf2)
    train_error_rate_rf2=1 -sum(diag(train_conf_matrix_rf2)) / sum(train_conf
_matrix_rf2)
    train_error_rates_rf2=c(train_error_rates_rf2, train_error_rate_rf2)
    #test
    test_pred_rf2=predict(rf2, newdata = x_test_cv)
    test_conf_matrix_rf2=table(x_test_cv$GradeClass, test_pred_rf2)
    test_error_rate_rf2=1 -sum(diag(test_conf_matrix_rf2)) / sum(test_conf_ma
trix_rf2)
    test_error_rates_rf2=c(test_error_rates_rf2, test_error_rate_rf2)


    #*** model3
    rf3 = randomForest(as.factor(GradeClass) ~ ., data = x_train_cv,
                       mtry = sqrt(ncol(x_train_cv)-1)+2, ntree = nseq[p])
    #train
    train_pred_rf3=predict(rf3, newdata = x_train_cv)
    train_conf_matrix_rf3=table(x_train_cv$GradeClass, train_pred_rf3)
    train_error_rate_rf3=1 -sum(diag(train_conf_matrix_rf3)) / sum(train_conf
_matrix_rf3)
    train_error_rates_rf3=c(train_error_rates_rf3, train_error_rate_rf3)
    #test
    test_pred_rf3=predict(rf3, newdata = x_test_cv)
    test_conf_matrix_rf3=table(x_test_cv$GradeClass, test_pred_rf3)
    test_error_rate_rf3=1 -sum(diag(test_conf_matrix_rf3)) / sum(test_conf_ma
trix_rf3)
    test_error_rates_rf3=c(test_error_rates_rf3, test_error_rate_rf3)


    #*** model4
    rf4 = randomForest(as.factor(GradeClass) ~ ., data = x_train_cv,
                       mtry = sqrt(ncol(x_train_cv)-1)-2, ntree = nseq[p])
    #train
    train_pred_rf4=predict(rf4, newdata = x_train_cv)
    train_conf_matrix_rf4=table(x_train_cv$GradeClass, train_pred_rf4)
    train_error_rate_rf4=1 -sum(diag(train_conf_matrix_rf4)) / sum(train_conf
_matrix_rf4)
    train_error_rates_rf4=c(train_error_rates_rf4, train_error_rate_rf4)
    #test
    test_pred_rf4=predict(rf4, newdata = x_test_cv)
    test_conf_matrix_rf4=table(x_test_cv$GradeClass, test_pred_rf4)
    test_error_rate_rf4=1 -sum(diag(test_conf_matrix_rf4)) / sum(test_conf_ma
trix_rf4)
    test_error_rates_rf4=c(test_error_rates_rf4, test_error_rate_rf4)


  }
  Mean_train_Error_Rate_rf1[p]= mean(train_error_rates_rf1)
  Mean_train_Error_Rate_rf2[p]= mean(train_error_rates_rf2)
```

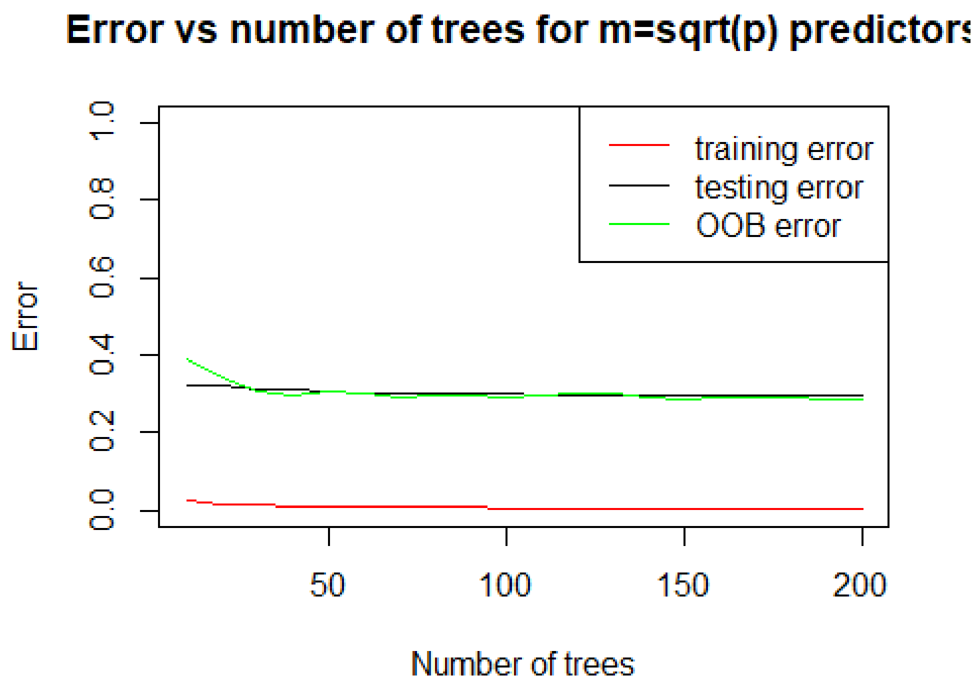
```
  Mean_train_Error_Rate_rf3[p]= mean(train_error_rates_rf3)
  Mean_train_Error_Rate_rf4[p]= mean(train_error_rates_rf4)

  Mean_test_Error_Rate_rf1[p]= mean(test_error_rates_rf1)
  Mean_test_Error_Rate_rf2[p]= mean(test_error_rates_rf2)
  Mean_test_Error_Rate_rf3[p]= mean(test_error_rates_rf3)
  Mean_test_Error_Rate_rf4[p]= mean(test_error_rates_rf4)



  ### full data models for oob
  rf1_oob=randomForest(as.factor(GradeClass) ~ ., data = mydata,
                       mtry =(ncol(x_train_cv)-1)/2, ntree = nseq[p])
  oob_rf1[p]=rf1_oob$err.rate[rf1_oob$ntree,1]

  rf2_oob=randomForest(as.factor(GradeClass) ~ ., data = mydata,
                       mtry =sqrt(ncol(x_train_cv)-1), ntree = nseq[p])
  oob_rf2[p]=rf2_oob$err.rate[rf2_oob$ntree,1]

  rf3_oob=randomForest(as.factor(GradeClass) ~ ., data = mydata,
                       mtry =sqrt(ncol(x_train_cv)-1)+2, ntree = nseq[p])
  oob_rf3[p]=rf3_oob$err.rate[rf3_oob$ntree,1]

  rf4_oob=randomForest(as.factor(GradeClass) ~ ., data = mydata,
                       mtry =sqrt(ncol(x_train_cv)-1)-2, ntree = nseq[p])
  oob_rf4[p]=rf4_oob$err.rate[rf4_oob$ntree,1]

}
```

Plots of error rates of random forests with different mtry values.

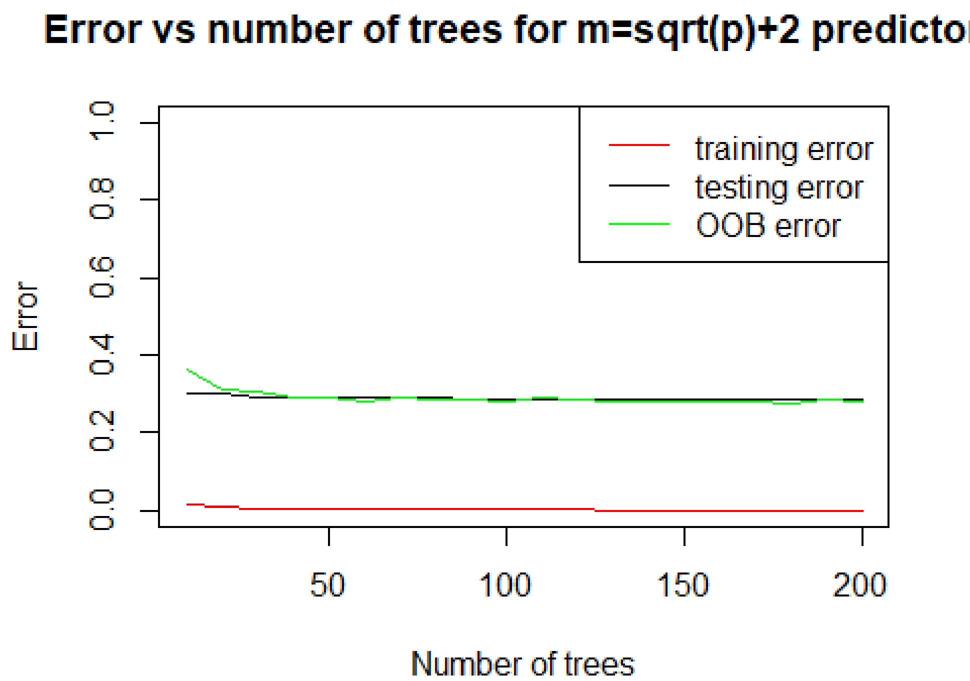
```
plot(nseq,Mean_train_Error_Rate_rf1,type="l",col="red",ylim=c(0,1),lwd=1.5,main="Error vs number of trees for m=p/2 predictors",xlab="Number of trees",ylab="Error")
lines(nseq,Mean_test_Error_Rate_rf1,type="l",col="black",lwd=1.5)
lines(nseq,oob_rf1,type="l",col="green",lwd=1.5)
legend("topright",c("training error","testing error","OOB error"),col=c("red","black","green"),lty=c(1,1,1))
```



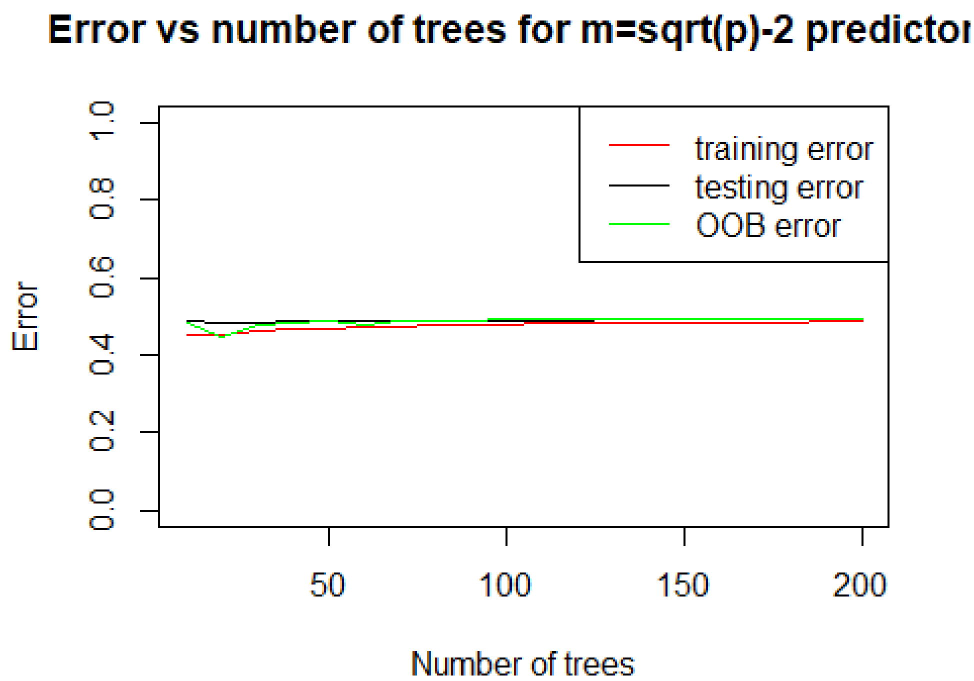

```r
plot(nseq,Mean_train_Error_Rate_rf2,type="l",col="red",ylim=c(0,1),lwd=1.5,main="Error vs number of trees for m=sqrt(p) predictors",xlab="Number of trees",ylab="Error")
lines(nseq,Mean_test_Error_Rate_rf2,type="l",col="black",lwd=1.5)
lines(nseq,oob_rf2,type="l",col="green",lwd=1.5)
legend("topright",c("training error","testing error","OOB error"),col=c("red","black","green"),lty=c(1,1,1))
```




```
plot(nseq,Mean_train_Error_Rate_rf3,type="l",col="red",ylim=c(0,1),lwd=1.5,ma
in="Error vs number of trees for m=sqrt(p)+2 predictors",xlab="Number of tree
s",ylab="Error")
lines(nseq,Mean_test_Error_Rate_rf3,type="l",col="black",lwd=1.5)
lines(nseq,oob_rf3,type="l",col="green",lwd=1.5)
legend("topright",c("training error","testing error","OOB error"),col=c("red"
,"black","green"),lty=c(1,1,1))
```

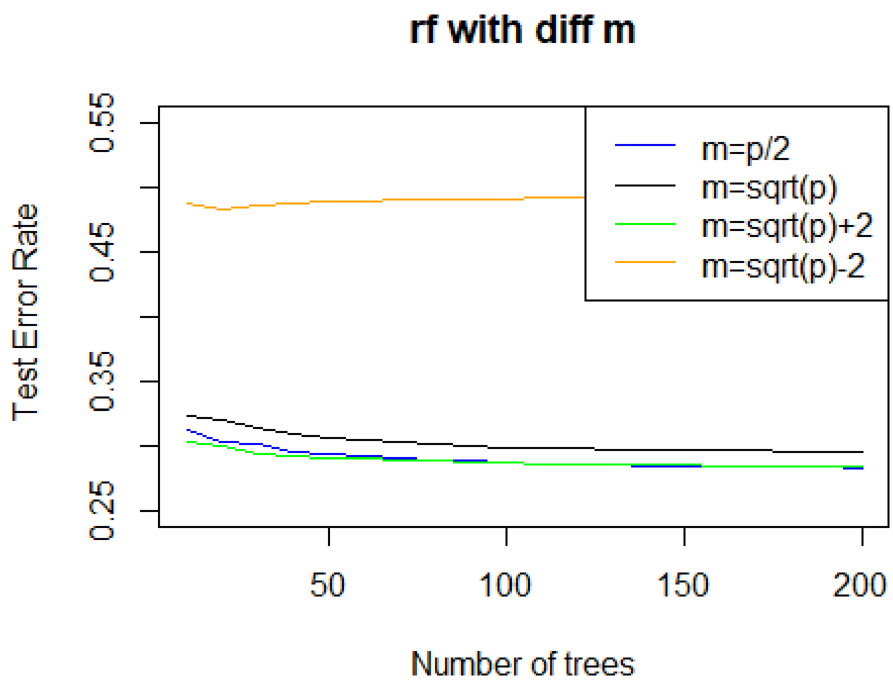



```r
plot(nseq,Mean_train_Error_Rate_rf4,type="l",col="red",ylim=c(0,1),lwd=1.5,main="Error vs number of trees for m=sqrt(p)-2 predictors",xlab="Number of trees",ylab="Error")
lines(nseq,Mean_test_Error_Rate_rf4,type="l",col="black",lwd=1.5)
lines(nseq,oob_rf4,type="l",col="green",lwd=1.5)

legend("topright",c("training error","testing error","OOB error"),col=c("red","black","green"),lty=c(1,1,1))
```

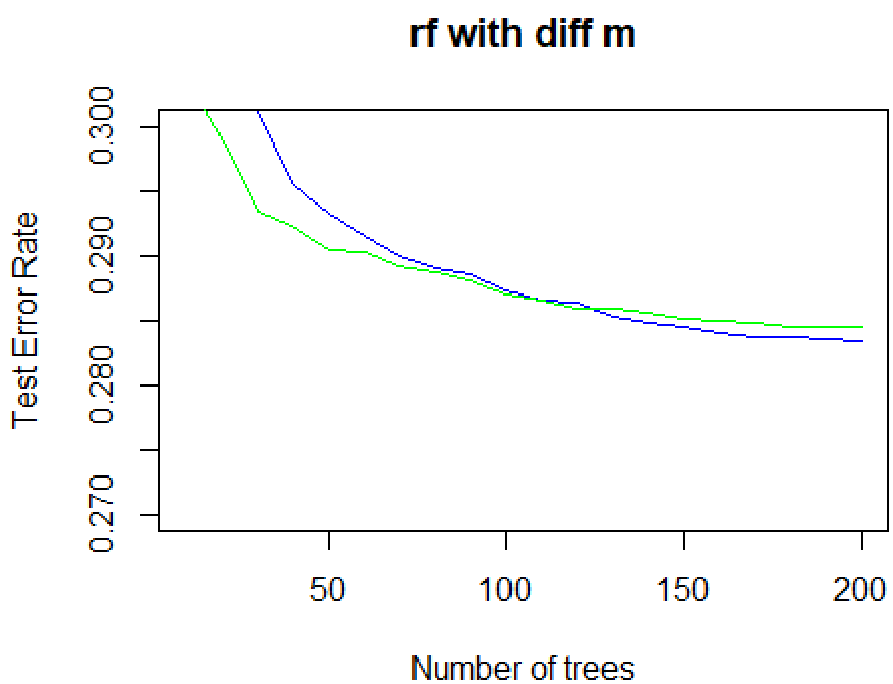


Plot for test error rate of various random forest models.

```
plot(nseq,Mean_test_Error_Rate_rf1,type="l",col="blue",lwd=1.5,ylim=c(0.25,0.
55),main="rf with diff m",xlab="Number of trees",ylab=" Test Error Rate")
lines(nseq,Mean_test_Error_Rate_rf2,type="l",col="black",lwd=1.5)
lines(nseq,Mean_test_Error_Rate_rf3,type="l",col="green",lwd=1.5)
lines(nseq,Mean_test_Error_Rate_rf4,type="l",col="orange",lwd=1.5)
legend("topright",c("m=p/2","m=sqrt(p)","m=sqrt(p)+2","m=sqrt(p)-2"),col=c("b
lue","black","green","orange"),lty=rep(1,4))
```




```
plot(nseq,Mean_test_Error_Rate_rf1,type="l",col="blue",lwd=1.5,ylim=c(0.27,0.
3),main="rf with diff m",xlab="Number of trees",ylab="Test Error Rate")
lines(nseq,Mean_test_Error_Rate_rf3,type="l",col="green",lwd=1.5)
```



To get optimal number of trees

```r
tune_rf=tune.randomForest(as.factor(GradeClass)~.,data=train,
                          ntree=seq(10,200,by=10),
                          mtry=c((ncol(train)-1)/2,sqrt(ncol(train)-1),
                                 sqrt(ncol(train)-1)+2,sqrt(ncol(train)-1)-2))
summary(tune_rf)
```

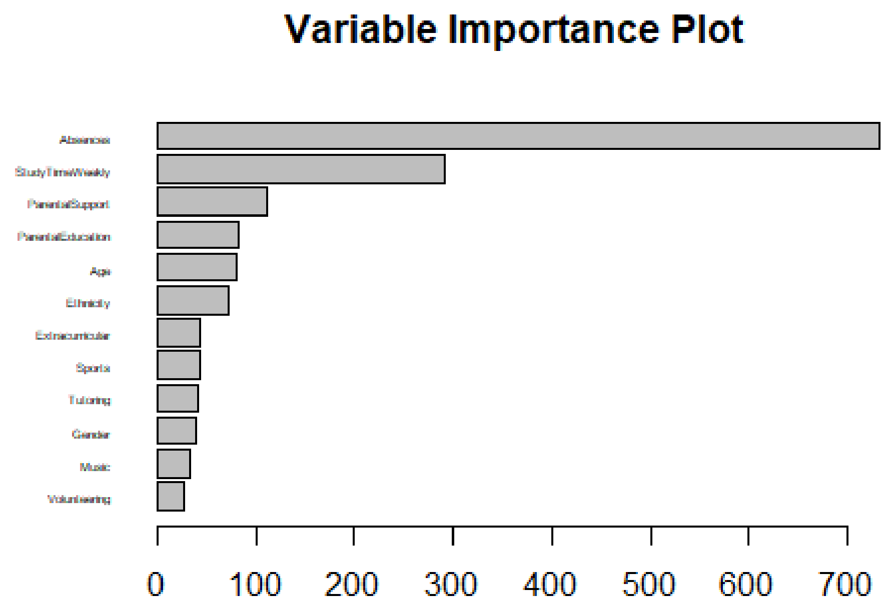
```
##
## Parameter tuning of 'randomForest':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  mtry ntree
##     6   190
##
## - best performance: 0.2733961
##
## - Detailed performance results:
##         mtry ntree     error dispersion
## 1  6.000000    10 0.2959745 0.02625807
## 2  3.464102    10 0.3268985 0.04120148
## 3  5.464102    10 0.3089418 0.02294025
## 4  1.464102    10 0.4757165 0.04288541
## 5  6.000000    20 0.3001447 0.02024327
## 6  3.464102    20 0.2955439 0.03206686
## 7  5.464102    20 0.2934467 0.04041309
## 8  1.464102    20 0.4920171 0.03655764
## 9  6.000000    30 0.2880300 0.02492813
## 10 3.464102    30 0.2922071 0.02858565
## 11 5.464102    30 0.2926151 0.04133638
## 12 1.464102    30 0.4920153 0.03959334
## 13 6.000000    40 0.2834153 0.03613240
## 14 3.464102    40 0.2901046 0.03040444
## 15 5.464102    40 0.2825785 0.03458402
## 16 1.464102    40 0.4936890 0.03720399
## 17 6.000000    50 0.2817381 0.03141690
## 18 3.464102    50 0.2947089 0.02412825
## 19 5.464102    50 0.2792503 0.03129301
## 20 1.464102    50 0.4928539 0.03669745
## 21 6.000000    60 0.2754759 0.03164478
## 22 3.464102    60 0.2901168 0.02549357
## 23 5.464102    60 0.2809205 0.02509367
## 24 1.464102    60 0.4932706 0.03712242
## 25 6.000000    70 0.2805056 0.02731655
## 26 3.464102    70 0.2989017 0.03306396
## 27 5.464102    70 0.2788302 0.03016615
## 28 1.464102    70 0.4936890 0.03720399
## 29 6.000000    80 0.2779934 0.02966311
```

```
## 30 3.464102    80 0.2876046 0.03665327
## 31 5.464102    80 0.2784083 0.02985490
## 32 1.464102    80 0.4932706 0.03790026
## 33 6.000000    90 0.2758996 0.03537375
## 34 3.464102    90 0.2917870 0.02719662
## 35 5.464102    90 0.2800819 0.02728760
## 36 1.464102    90 0.4936890 0.03720399
## 37 6.000000   100 0.2742225 0.03095104
## 38 3.464102   100 0.2942992 0.02692673
## 39 5.464102   100 0.2842591 0.02930796
## 40 1.464102   100 0.4936890 0.03720399
## 41 6.000000   110 0.2767277 0.02927252
## 42 3.464102   110 0.2846810 0.03161181
## 43 5.464102   110 0.2759066 0.03283934
## 44 1.464102   110 0.4936890 0.03720399
## 45 6.000000   120 0.2767347 0.02923655
## 46 3.464102   120 0.2880160 0.02699396
## 47 5.464102   120 0.2775732 0.03291637
## 48 1.464102   120 0.4936890 0.03720399
## 49 6.000000   130 0.2742225 0.03650623
## 50 3.464102   130 0.2951360 0.03055609
## 51 5.464102   130 0.2738075 0.03108318
## 52 1.464102   130 0.4936890 0.03720399
## 53 6.000000   140 0.2746496 0.02836638
## 54 3.464102   140 0.2855265 0.02285123
## 55 5.464102   140 0.2746409 0.03219766
## 56 1.464102   140 0.4936890 0.03720399
## 57 6.000000   150 0.2821653 0.03005785
## 58 3.464102   150 0.2813476 0.02772553
## 59 5.464102   150 0.2784170 0.02802461
## 60 1.464102   150 0.4936890 0.03720399
## 61 6.000000   160 0.2750662 0.02679088
## 62 3.464102   160 0.2821775 0.02501423
## 63 5.464102   160 0.2738128 0.02580860
## 64 1.464102   160 0.4936890 0.03720399
## 65 6.000000   170 0.2742242 0.02717717
## 66 3.464102   170 0.2813389 0.02489467
## 67 5.464102   170 0.2742347 0.02629716
## 68 1.464102   170 0.4936890 0.03720399
## 69 6.000000   180 0.2763145 0.02624992
## 70 3.464102   180 0.2905370 0.02671991
## 71 5.464102   180 0.2788337 0.02461305
## 72 1.464102   180 0.4936890 0.03720399
## 73 6.000000   190 0.2733961 0.02319444
## 74 3.464102   190 0.2855143 0.03017368
## 75 5.464102   190 0.2800837 0.02995617
## 76 1.464102   190 0.4936890 0.03720399
## 77 6.000000   200 0.2763128 0.02688728
## 78 3.464102   200 0.2834222 0.02873554
```

```
## 79 5.464102   200 0.2754881 0.02738155
## 80 1.464102   200 0.4936890 0.03720399
```

Fitting model based on tuned parameters.

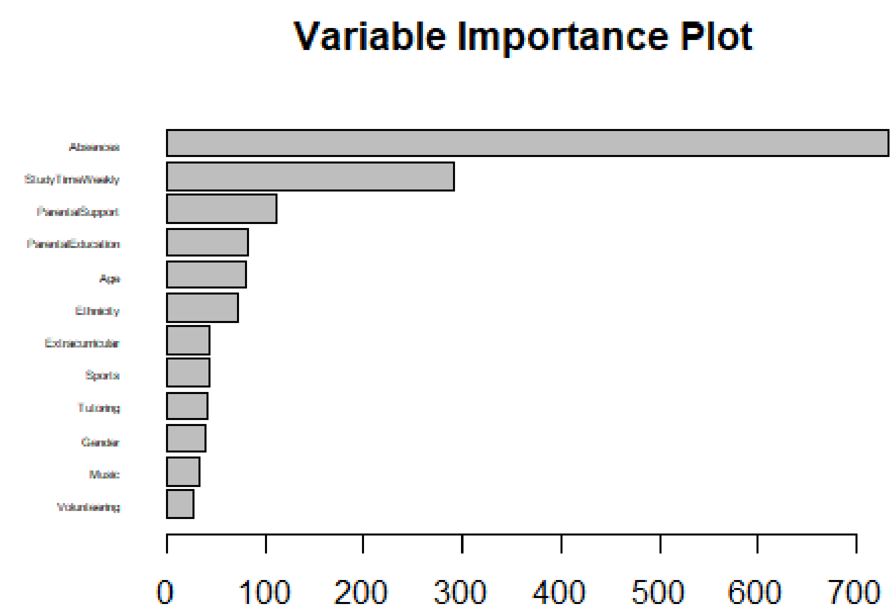
```
rf_final_train=randomForest(as.factor(GradeClass)~.,data=train,
                      mtry=6,ntree=190)
print(rf_final_train)

##
## Call:
##  randomForest(formula = as.factor(GradeClass) ~ ., data = train,      mtry
= 6, ntree = 190)
##                Type of random forest: classification
##                      Number of trees: 190

## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 27.96%
## Confusion matrix:
##    A  B   C   D   F class.error
## A 20 32   6   8  12   0.7435897
## B 16 85  52   7  15   0.5142857
## C  0 34 157  66  16   0.4249084
## D  1  8  58 137  76   0.5107143
## F  1  5   5  50 807   0.0702765

test_pred_rf_final=predict(rf_final_train, newdata =test)
test_conf_matrix_rf_final=table(test$GradeClass, test_pred_rf_final)
test_error_rate_rf_final=1 -sum(diag(test_conf_matrix_rf_final)) / sum(test_c
onf_matrix_rf_final)
test_error_rate_rf_final

## [1] 0.2952646

barplot(sort(importance(rf_final)[,1]),horiz=T,
        las=1,cex.names = 0.4,main="Variable Importance Plot")
```



Method	m	Best n tree	OOB error	Testing error
Random forest	p/2=6	150	0.2796	0.2952

7.Boosting

```
library(gbm)

boost=gbm(as.factor(GradeClass) ~., data =train,
            distribution = "multinomial", n.trees = 500,
            interaction.depth = 2, shrinkage = 0.01)

train_pred_class=apply(predict.gbm(boost,train,type="response"),1,which.max)-
1

## Using 500 trees...

test_pred_class=apply(predict.gbm(boost,test,type="response"),1,which.max)-1

## Using 500 trees...

barplot(sort(importance(rf_final)[,1]),horiz=T,
        las=1,cex.names = 0.4,main="Variable Importance Plot")
```



Train and Test error rate.

```
train_conf_mat=table(train$GradeClass,train_pred_class)
test_conf_mat=table(test$GradeClass,test_pred_class)

test_accuracy=sum(diag(test_conf_mat))/sum(test_conf_mat)
```

```
test_err_rate_boosting=1-test_accuracy
paste("test error:",test_err_rate_boosting)

## [1] "test error: 0.309192200557103"

train_accuracy=sum(diag(train_conf_mat))/sum(train_conf_mat)
train_err_rate_boosting=1-train_accuracy
paste("train error:",train_err_rate_boosting)

## [1] "train error: 0.241935483870968"
```

Conclusion:

Method	Testing Accuracy
Multinomial Logistic Regression	0.7325905
KNN	0.6866295
SVM	0.7311977
Decision tree	0.6824512
Bagging	0.7019499
Random Forest	0.7047354
Boosting	0.6908078