Project

```r
library(tidyverse,corrplot)
```

```
## -- Attaching packages --------------------------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   1.0.0
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## -- Conflicts ------------------------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(rpart)
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```r
library(rpart.plot)
library(randomForest)
```

```
## randomForest 4.6-14

## 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(randomForestExplainer)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```r
library(corrr)
library(ggplot2)
#library(Hmisc)
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```r
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
library(gbm)
```

```
## Loaded gbm 2.1.8
```

```r
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```r
df = read.csv("project_data.csv")
```

1 Introduction

In this project we have considered a dataset of 2017 songs from Spotify and each song has 16 features. One of the features is called target which is a categorical variable and tells us whether one particular individual liked or disliked a song. A song is labeled "1" if it is liked and "0" when it is disliked. The other features include acousticness, danceability, durationms (duration in milliseconds), energy, instrumentalness, key, liveness, loudness, mode, speechiness, tempo, timesignature, valence, songname, artist.

The goal of the project is to build several classifier to predict that based on the rest of the features, whether or not that individual would like a song.

But first, we will prepare the data to fit some of these models.

2 Data exploration and feature selection:

Before we try any model, we want to make sure the data is ready for a fit. In particular we are looking for features that have missing values, songs that are duplicated in the dataset, the types of features (if a feature is numerical or categorical), and as well as the variables of importance for the fit.

```r
ncol(df)
```

```
## [1] 17
```

```r
nrow(df)
```

```
## [1] 2017
```

```r
colnames(df)
```

```
##  [1] "X"                "acousticness"     "danceability"     "duration_ms"
##  [5] "energy"           "instrumentalness" "key"              "liveness"
##  [9] "loudness"         "mode"             "speechiness"      "tempo"
## [13] "time_signature"   "valence"          "target"           "song_title"
## [17] "artist"
```

We drop the first column, which is just an indexing column that keeps track of the number of observations and has nothing to do with the data.

```r
df1 = df[,c(2:ncol(df))]
colnames(df1)
```

```
##  [1] "acousticness"     "danceability"     "duration_ms"      "energy"
##  [5] "instrumentalness" "key"              "liveness"         "loudness"
##  [9] "mode"             "speechiness"      "tempo"            "time_signature"
## [13] "valence"          "target"           "song_title"       "artist"
```

Looking for missing values

```r
#summary(df)
is.null(df1)
```

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

Looking for duplicated observations in the dataset.

```r
mean(duplicated(df1))
```

```
## [1] 0.002478929
```

Non-zero mean suggest that there are some duplicated observations. We now delete those.

```r
df2 = unique(df1)
mean(duplicated(df2))
```

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

Number of observations after the removal of duplicated data points.

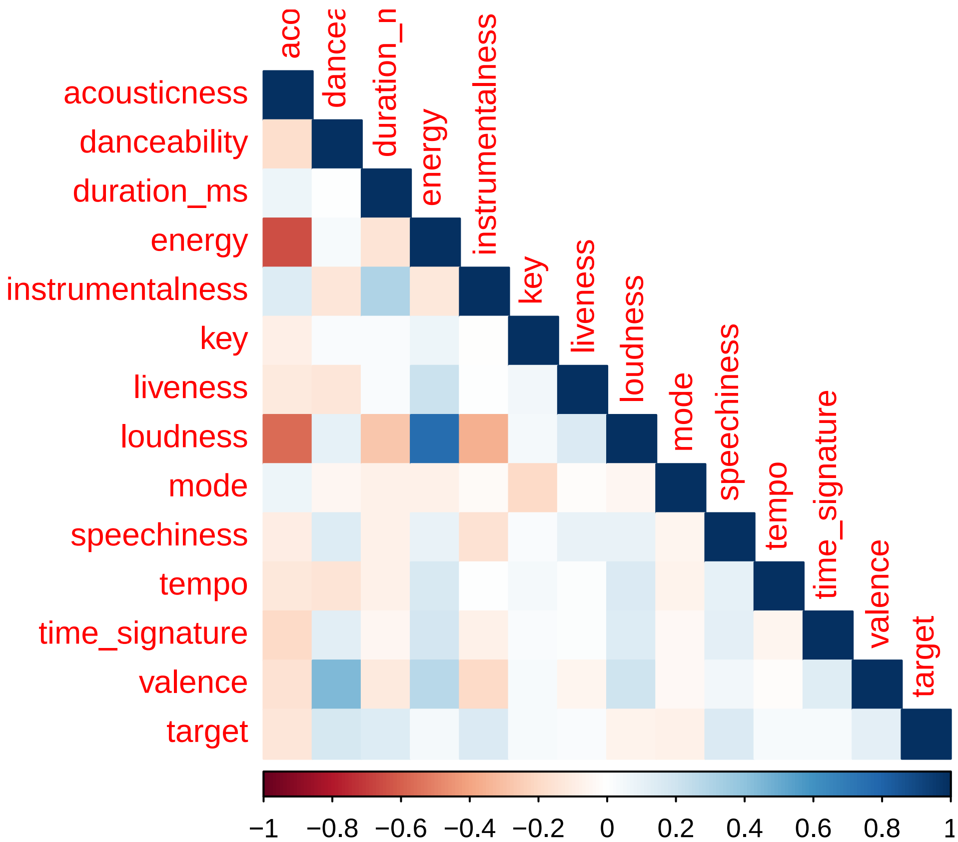
```r
nrow(df2)
```

```
## [1] 2012
```

Originally we had 2017 data points, so this means five of the data points were duplicated.

Looking for correllations (Heatmap of Correlation Matrix): We have drooped the last two features 'song_title' and 'artist_name' as well.

```r
df2_cor = cor(as.matrix(df2[,1:14]))
corrplot(df2_cor, method="color",type = "lower")
```




```
#df2_cor
```

```
df3 = df2[,1:14]

df3_cor <- df3 %>%
  correlate() %>%
  focus(target)
```

Looking for correlation of other feautre with 'target':

```
##
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'
```

```
df3_cor
```

```
## # A tibble: 13 x 2
##    term              target
##    <chr>              <dbl>
##  1 acousticness     -0.130
##  2 danceability      0.177
##  3 duration_ms       0.146
##  4 energy            0.0410
##  5 instrumentalness  0.152
##  6 key               0.0354
##  7 liveness          0.0262
##  8 loudness         -0.0699
```

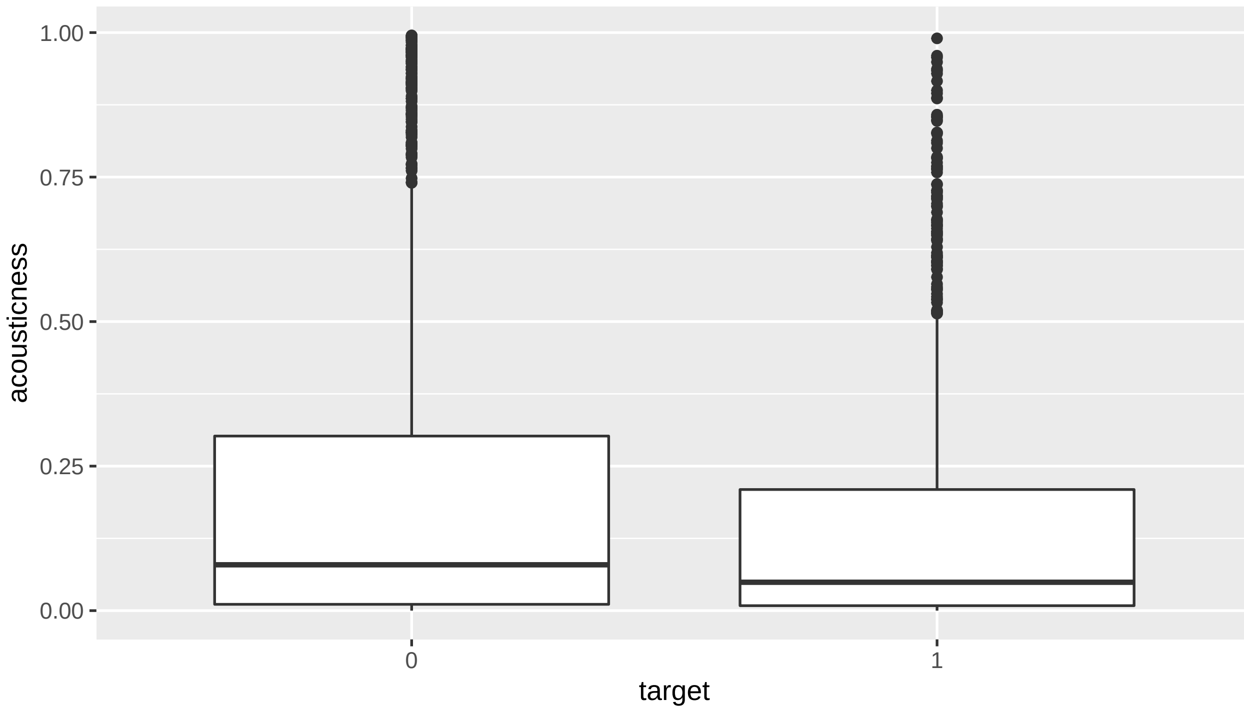
```
##   9 mode              -0.0725
## 10 speechiness        0.154
## 11 tempo              0.0348
## 12 time_signature     0.0399
## 13 valence            0.110
```

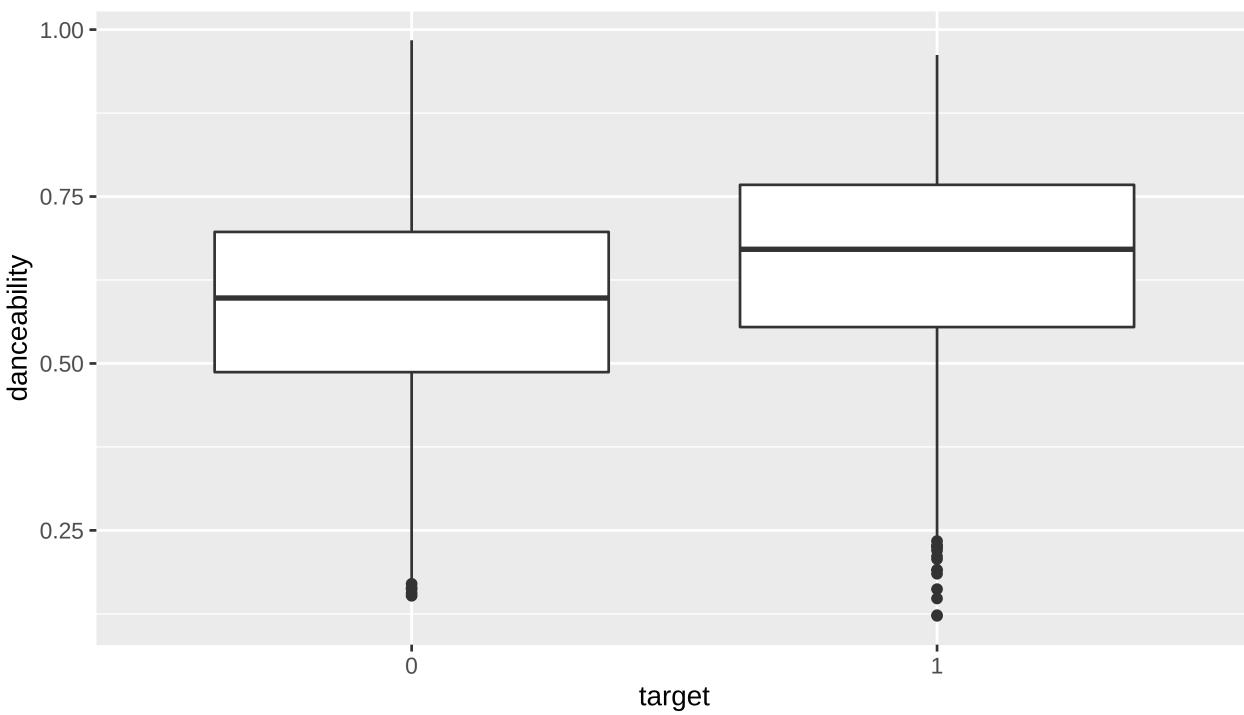
Variation in the data Let's take a look at our response variable `target`. The proportion of 1s and 0s are almost half and half so we do not need to worry about balance the data for fitting.

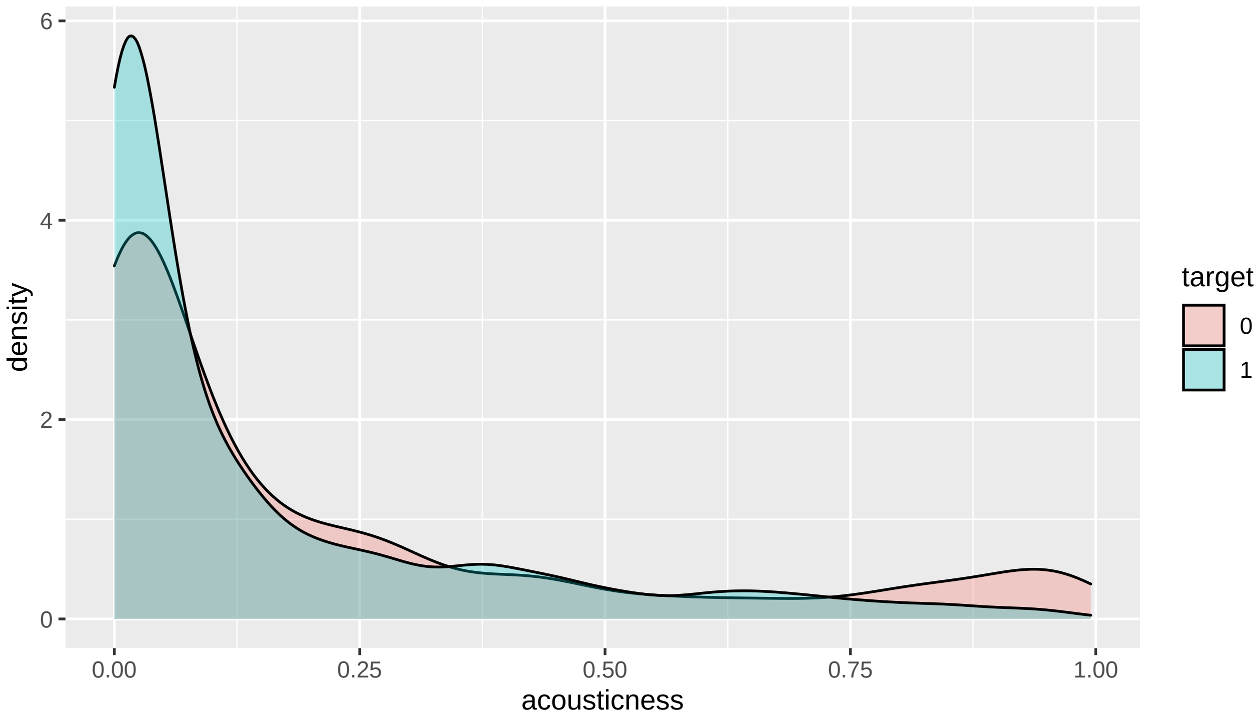
```
table(df3$target)
##
##    0    1
##  997 1015
```

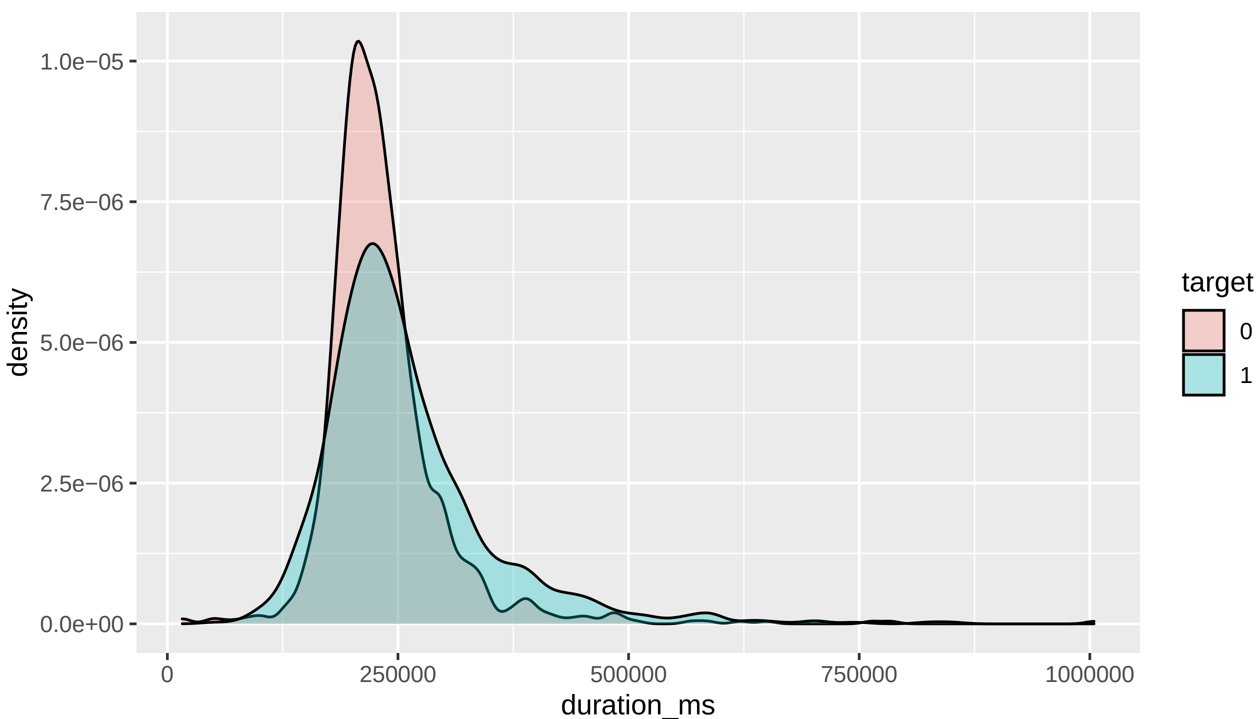
```
df3$target <- as.character(df3$target)
summary(df3)
##   acousticness        danceability     duration_ms          energy
##  Min.   :0.0000028   Min.   :0.1220   Min.   :  16042   Min.   :0.0148
##  1st Qu.:0.0095900   1st Qu.:0.5140   1st Qu.: 200004   1st Qu.:0.5637
##  Median :0.0635000   Median :0.6310   Median : 229120   Median :0.7155
##  Mean   :0.1875135   Mean   :0.6185   Mean   : 246261   Mean   :0.6818
##  3rd Qu.:0.2650000   3rd Qu.:0.7380   3rd Qu.: 270356   3rd Qu.:0.8460
##  Max.   :0.9950000   Max.   :0.9840   Max.   :1004627   Max.   :0.9980
##  instrumentalness         key            liveness         loudness
##  Min.   :0.0000000   Min.   : 0.000   Min.   :0.0188   Min.   :-33.097
##  1st Qu.:0.0000000   1st Qu.: 2.000   1st Qu.:0.0922   1st Qu.: -8.392
##  Median :0.0000738   Median : 6.000   Median :0.1265   Median : -6.247
##  Mean   :0.1329797   Mean   : 5.349   Mean   :0.1908   Mean   : -7.077
##  3rd Qu.:0.0539250   3rd Qu.: 9.000   3rd Qu.:0.2462   3rd Qu.: -4.744
##  Max.   :0.9760000   Max.   :11.000   Max.   :0.9690   Max.   : -0.307
##       mode          speechiness          tempo        time_signature
##  Min.   :0.0000   Min.   :0.02310   Min.   : 47.86   Min.   :1.000
##  1st Qu.:0.0000   1st Qu.:0.03750   1st Qu.:100.16   1st Qu.:4.000
##  Median :1.0000   Median :0.05490   Median :121.41   Median :4.000
##  Mean   :0.6123   Mean   :0.09257   Mean   :121.60   Mean   :3.968
##  3rd Qu.:1.0000   3rd Qu.:0.10800   3rd Qu.:137.70   3rd Qu.:4.000
##  Max.   :1.0000   Max.   :0.81600   Max.   :219.33   Max.   :5.000
##     valence           target
##  Min.   :0.0348   Length:2012
##  1st Qu.:0.2960   Class :character
##  Median :0.4930   Mode  :character
##  Mean   :0.4973
##  3rd Qu.:0.6920
##  Max.   :0.9920
```

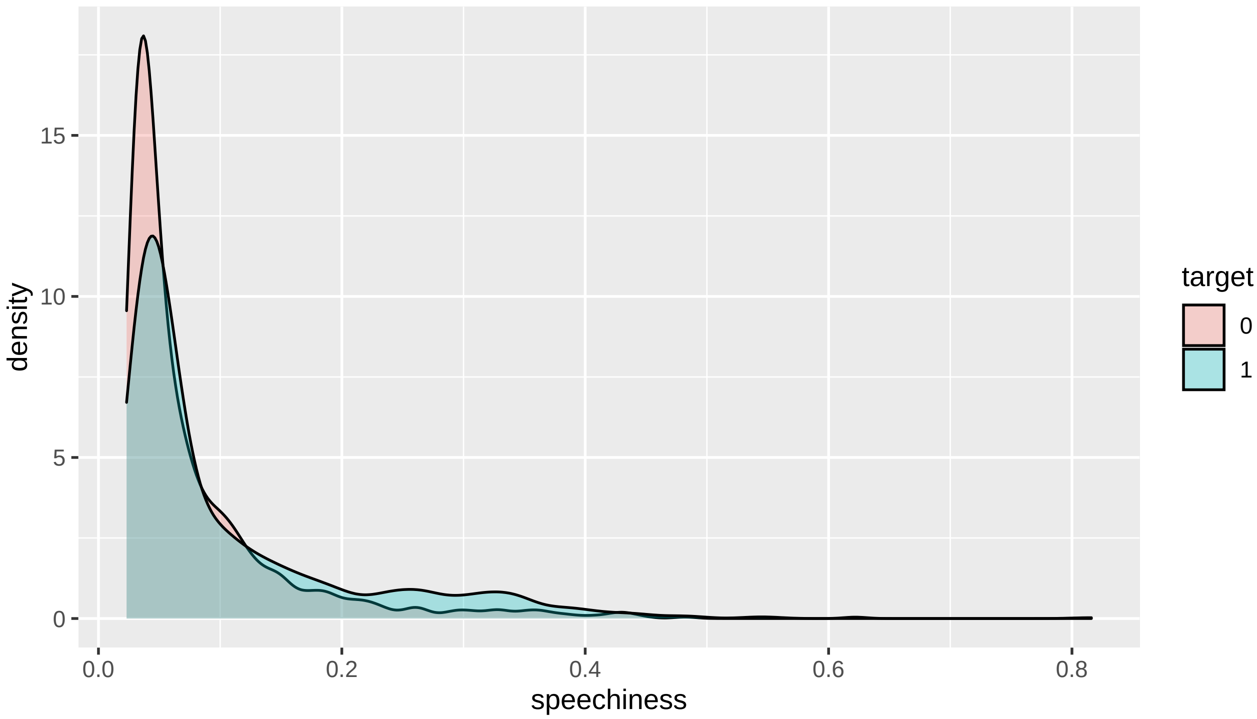
Covariation between variables We also want to have a look at how other variables may vary with `target`: There is some interesting covariation between some features and target, such as acounstincness, duration time, where the mode of density plots show some correlations with `target`.


1.00
0.75
0.50
0.25
0.00
acousticness
0
1
target

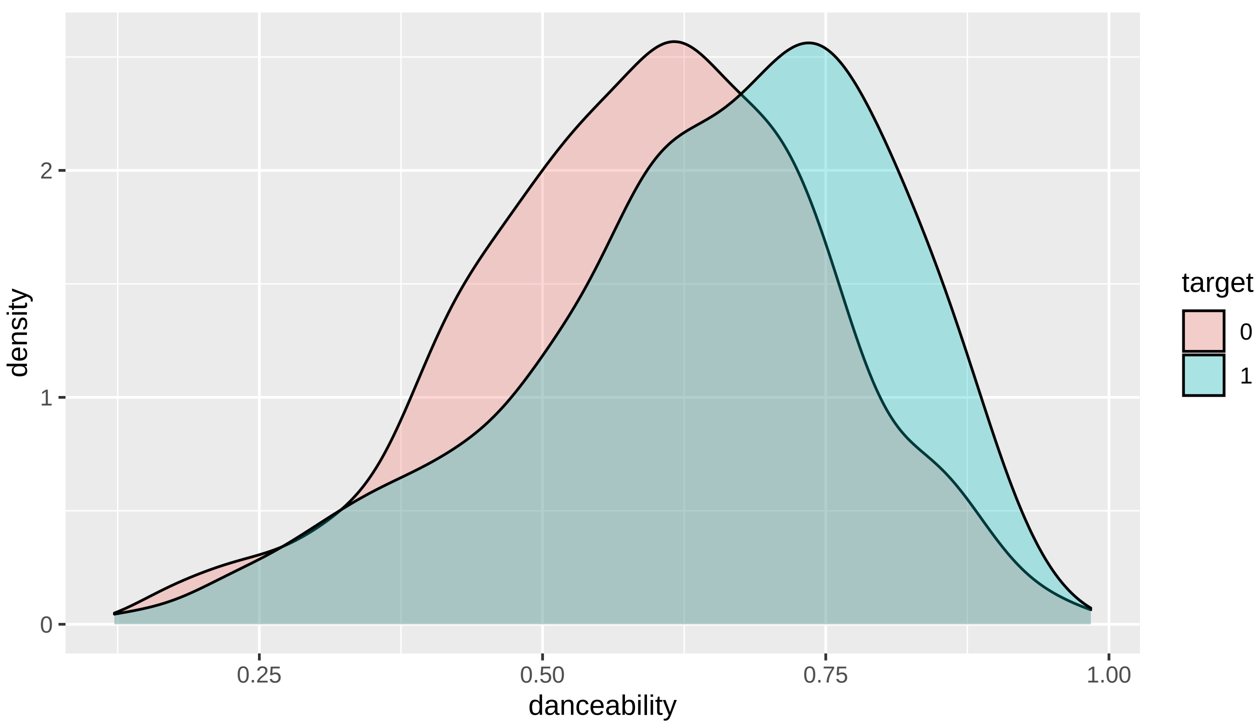

1.00
0.75
0.50
0.25
danceability
0
1
target

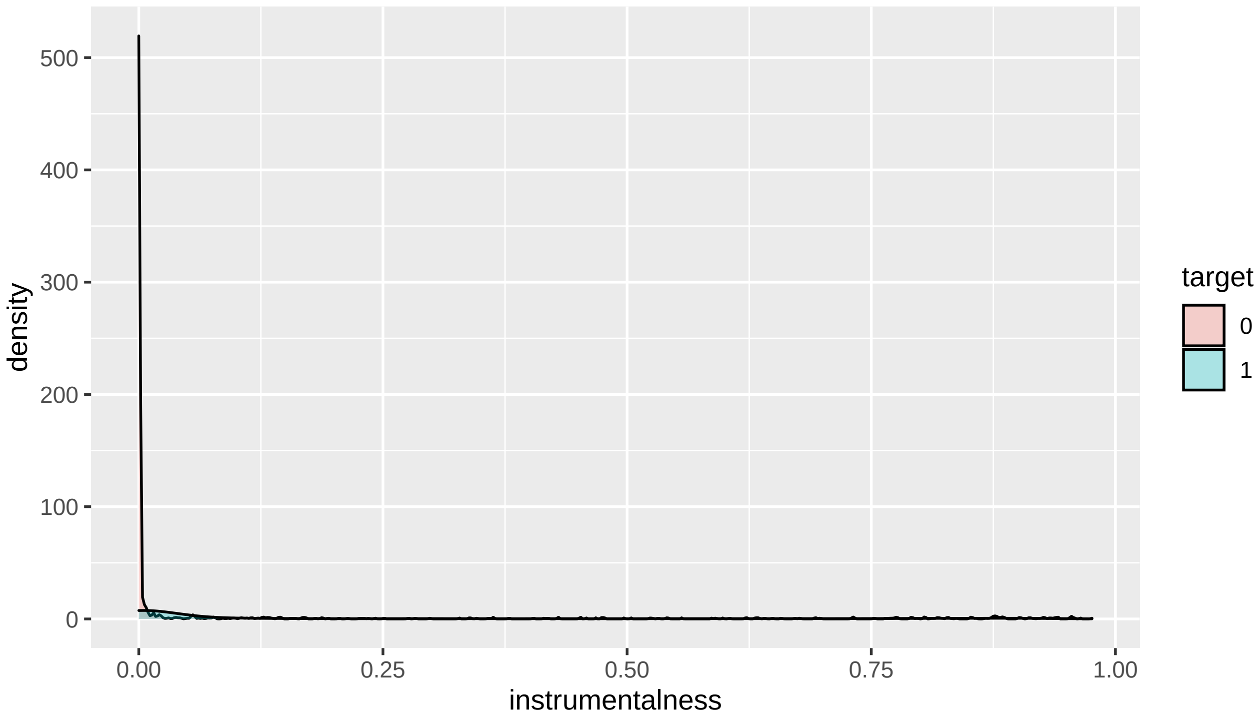

density
acousticness
target
0
1
0.00
0.25
0.50
0.75
1.00
0
2
4
6

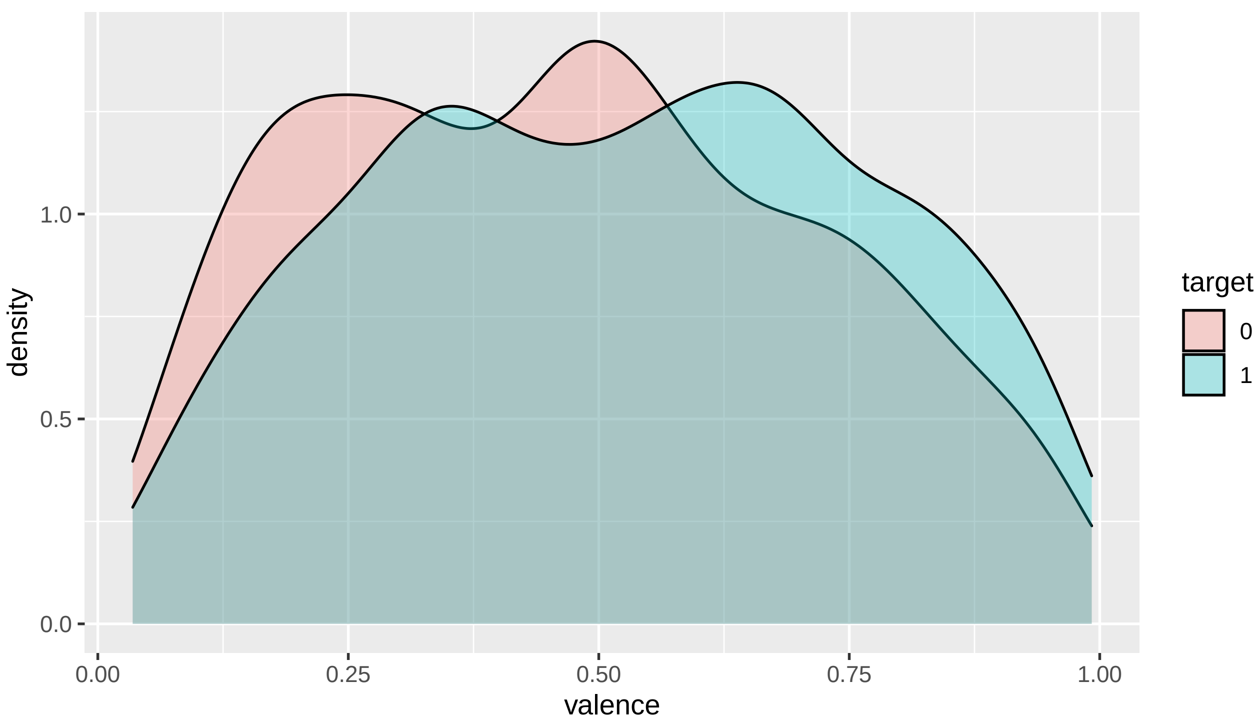

density
duration_ms
target
0
1
0
250000
500000
750000
1000000
0.0e+00
2.5e−06
5.0e−06
7.5e−06
1.0e−05



variables that don't show much interesting variation against target







3 Features selection based on correlations with 'target':

acousticness

danceability

duration_ms

instrumentalness

speechiness

valence

We also rescale the varibale `duration_ms` so that it's on the same scale as other variables.

```
df4 = df3[,c(14,1,2,3,5,10,13)]
df4$duration_ms <- (df4$duration_ms-min(df4$duration_ms))/(max(df4$duration_ms-min(df4$duration_ms)))
head(df4)
```

```
##   target acousticness danceability duration_ms instrumentalness speechiness
## 1      1      0.01020        0.833   0.1907352         0.021900      0.4310
## 2      1      0.19900        0.743   0.3144808         0.006110      0.0794
## 3      1      0.03440        0.838   0.1716241         0.000234      0.2890
## 4      1      0.60400        0.494   0.1854883         0.510000      0.0261
## 5      1      0.18000        0.678   0.3812024         0.512000      0.0694
## 6      1      0.00479        0.804   0.2380079         0.000000      0.1850
##   valence
## 1   0.286
## 2   0.588
## 3   0.173
## 4   0.230
## 5   0.904
## 6   0.264
```

```
#ncol(df4)
```

#4 Models
The potential methods to build a calcification model for this project include:

Logistic Regression (Penny)

Naive Bayes (Sinta)

Decision Trees (Atta)

Random Forests (Sinta)

Boosting (Penny)

Neural Nets (Antonio)

Deep Neural Nets (Antonio)

Test Train split

```
n = nrow(df4)
set.seed(199)
pin = .65
ii = sample(1:n,floor(pin*n))
cdtrain = df4[ii,]
```

```
cdtest = df4[-ii,]
cat("dimension of train data:",dim(cdtrain),"\n")
```

```
## dimension of train data: 1307 7
```

```
## dimension of train data: 750 3
cat("dimension of test data:",dim(cdtest),"\n")
```

```
## dimension of test data: 705 7
```

```
## dimension of test data: 250 3
```

```
fit = rpart(target~., data = cdtrain, method = 'class')
#rpart.plot(fit, extra = 106)
```

```
predict_unseen = predict(fit, cdtest, type = 'class')
table_mat = table(cdtest$target, predict_unseen)
table_mat
```

```
##    predict_unseen
##       0   1
##   0 220 132
##   1  97 256
```

```
accuracy_Test = sum(diag(table_mat)) / sum(table_mat)
print(paste('Accuracy for test data is: ', accuracy_Test))
```

```
## [1] "Accuracy for test data is:  0.675177304964539"
```

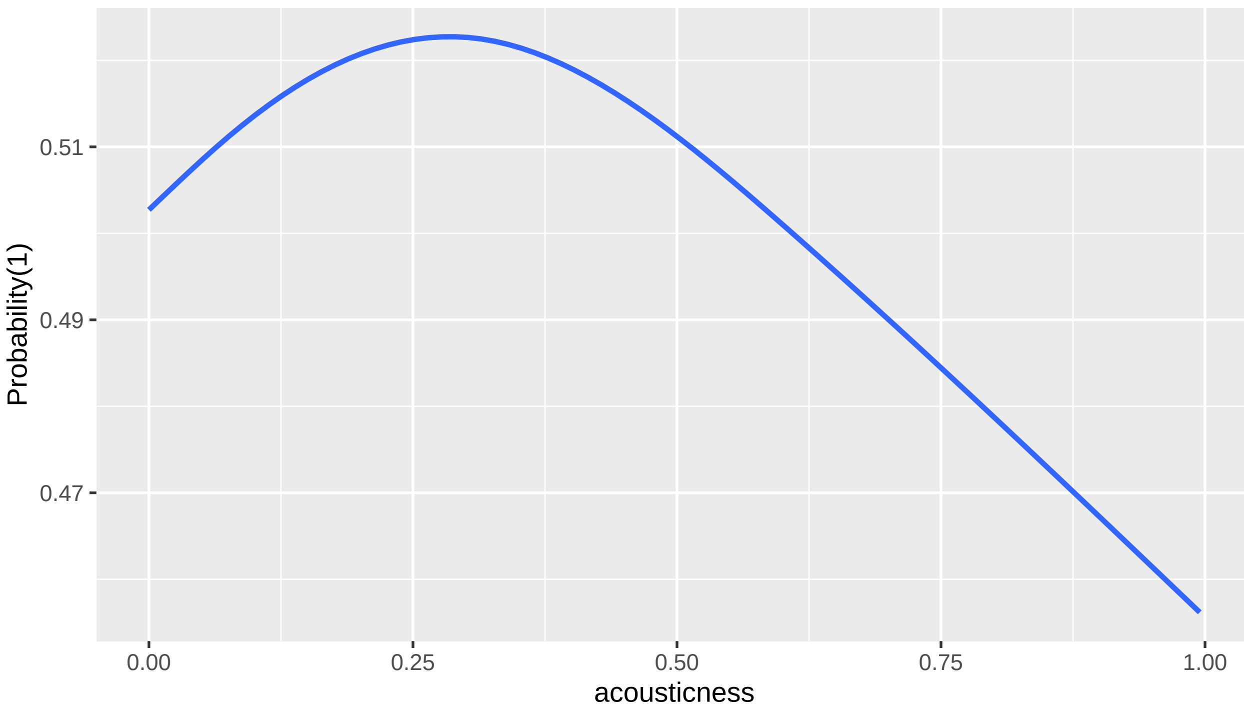
Logistic regression

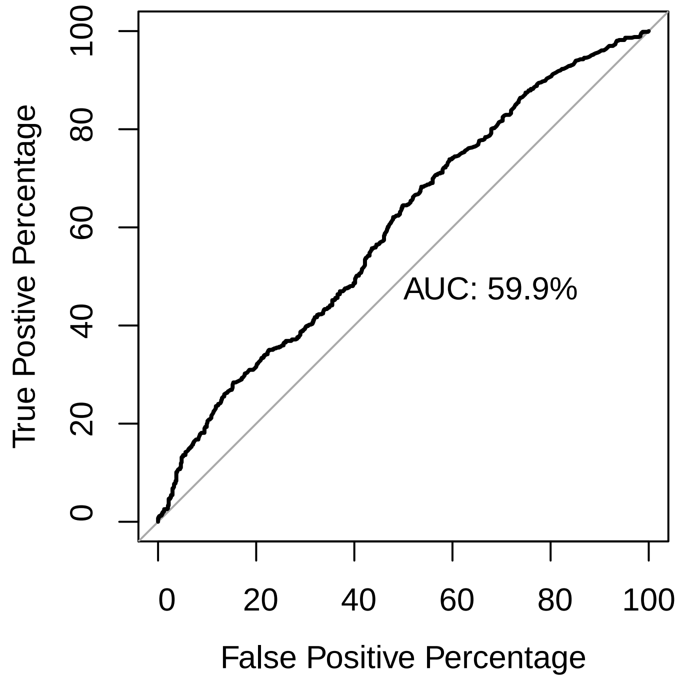
We want to start out simple, so let's use only one predictor to fit the training set.

```
      glm.fits=glm(as.factor(target)~speechiness,
                   data=cdtrain,family=binomial)
      contrasts(as.factor(cdtrain$target))
##   1
## 0 0
## 1 1
      summary(glm.fits)
##
## Call:
## glm(formula = as.factor(target) ~ speechiness, family = binomial,
##     data = cdtrain)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8517  -1.1190   0.7007   1.2197   1.2786
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.32295    0.08266  -3.907 9.35e-05 ***
## speechiness  3.81500    0.68844   5.541 3.00e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1811.7  on 1306  degrees of freedom
## Residual deviance: 1777.5  on 1305  degrees of freedom
## AIC: 1781.5
##
## Number of Fisher Scoring iterations: 4
```

Evaluation of the model
We first plot the ROC curve on the train set and the AUC is about 60%, but the AUC of the test set decreased a little, which is 57.5%.






```
##
## Call:
## roc.default(response = cdtrain$target, predictor = glm.fits$fitted.values,     percent = TRUE, plot =
##
## Data: glm.fits$fitted.values in 645 controls (cdtrain$target 0) < 662 cases (cdtrain$target 1).
## Area under the curve: 59.95%
```

Confusion matrix and performance statistics on the test set:

```
glm.probs=predict(glm.fits,newdata=cdtest,type="response")
str(glm.probs)
```

```
##  Named num [1:705] 0.789 0.444 0.731 0.466 0.528 ...
##  - attr(*, "names")= chr [1:705] "1" "4" "9" "12" ...
```

```
glm.pred=rep('0',length(glm.probs))
glm.pred[glm.probs >.5]='1'

#confusion matrix
#table(glm.pred,cdtest$target)
# accuracy
#(176+83)/length(glm.probs)

confusionMatrix(as.factor(glm.pred),as.factor(cdtest$target))
```

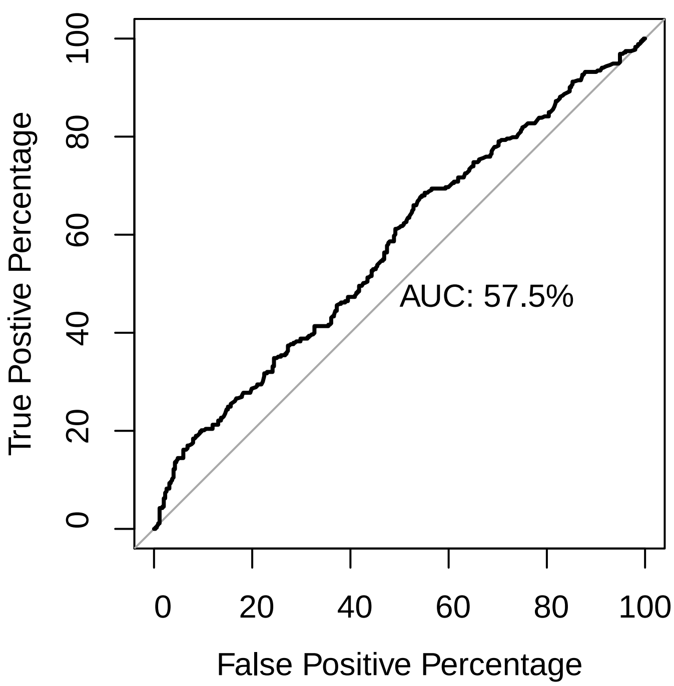
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 256 225
##          1  96 128
##
##                Accuracy : 0.5447
##                  95% CI : (0.5071, 0.5819)
```

```
##     No Information Rate : 0.5007
##     P-Value [Acc > NIR] : 0.01076
##
##                   Kappa : 0.0898
##
##  Mcnemar's Test P-Value : 9.048e-13
##
##             Sensitivity : 0.7273
##             Specificity : 0.3626
##          Pos Pred Value : 0.5322
##          Neg Pred Value : 0.5714
##              Prevalence : 0.4993
##          Detection Rate : 0.3631
##    Detection Prevalence : 0.6823
##       Balanced Accuracy : 0.5449
##
##        'Positive' Class : 0
##
```

Here is the ROC for the test results:

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

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




```
##
## Call:
## roc.default(response = cdtest$target, predictor = glm.probs,     percent = TRUE, plot = TRUE, legacy
##
## Data: glm.probs in 352 controls (cdtest$target 0) < 353 cases (cdtest$target 1).
## Area under the curve: 57.46%
```

The performance of one-predictor logistic regression is not very ideal, with 54% accuracy on the test set. The AUC is also fairly low, 57.5%.

Logistic regression with multiple predictors
Next we will try to use more than one predictors to fit logistic regression.

```
glm.fits2=glm(as.factor(target)~speechiness+danceability+duration_ms+
                instrumentalness+acousticness,data=cdtrain,family=binomial)

summary(glm.fits2)
```

```
##
## Call:
## glm(formula = as.factor(target) ~ speechiness + danceability +
##     duration_ms + instrumentalness + acousticness, family = binomial,
##     data = cdtrain)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4101  -1.0485   0.3749   1.0621   2.1025
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -2.5535     0.3254  -7.847 4.27e-15 ***
## speechiness        4.2133     0.7176   5.871 4.32e-09 ***
## danceability       2.0856     0.3934   5.302 1.15e-07 ***
## duration_ms        3.8931     0.8612   4.521 6.17e-06 ***
## instrumentalness   1.8284     0.2574   7.104 1.21e-12 ***
## acousticness      -1.1927     0.2441  -4.885 1.03e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1811.7  on 1306  degrees of freedom
## Residual deviance: 1629.3  on 1301  degrees of freedom
## AIC: 1641.3
##
## Number of Fisher Scoring iterations: 4
```

Predict on the test set and evaluate the performance.
We see that it does do a better job than the one-predictor model above, both accuracy and AUC are increased by significant amount but it is still not what we desire as performance.

```
glm.probs2=predict(glm.fits2,newdata=cdtest,type="response")

glm.pred2=rep('0',length(glm.probs2))
glm.pred2[glm.probs2 >.5]='1'

#confusion matrix
#table(glm.pred2,cdtest$target)
# accuracy
#(176+83)/length(glm.probs2)
confusionMatrix(as.factor(glm.pred2),as.factor(cdtest$target))
```

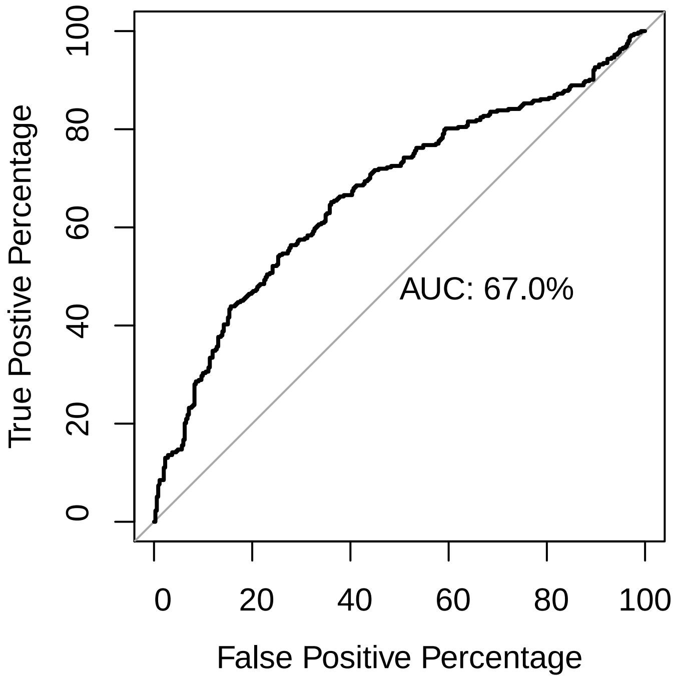
```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction   0   1
##          0 229 137
##          1 123 216
##
##                Accuracy : 0.6312
##                  95% CI : (0.5944, 0.6669)
##     No Information Rate : 0.5007
##     P-Value [Acc > NIR] : 2.076e-12
##
##                   Kappa : 0.2625
##
##  Mcnemar's Test P-Value : 0.4201
##
##             Sensitivity : 0.6506
##             Specificity : 0.6119
##          Pos Pred Value : 0.6257
##          Neg Pred Value : 0.6372
##              Prevalence : 0.4993
##          Detection Rate : 0.3248
##    Detection Prevalence : 0.5191
##       Balanced Accuracy : 0.6312
##
##        'Positive' Class : 0
##
```

```r
#ROC CURVE
par(pty='s')
roc(cdtest$target, glm.probs2,
        plot=TRUE,legacy.axes=TRUE, percent=TRUE,
        xlab="False Positive Percentage", ylab="True Postive Percentage",print.auc=TRUE)
```

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

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



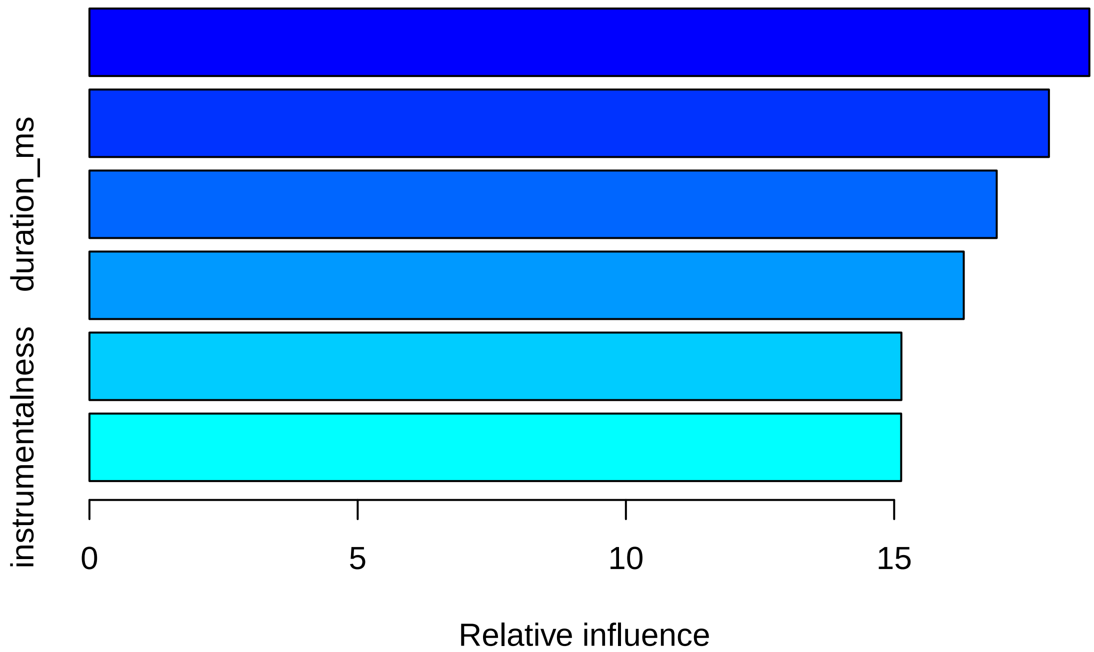

```
##
## Call:
## roc.default(response = cdtest$target, predictor = glm.probs2,     percent = TRUE, plot = TRUE, legacy
##
## Data: glm.probs2 in 352 controls (cdtest$target 0) < 353 cases (cdtest$target 1).
## Area under the curve: 66.96%
```

Boosting

Train, Validation and Test Split In order to use the three set approach, we divide the data into three sets, the train set, the validation set and the test set.

```
set.seed(1722)
n=nrow(df4)
n1=floor(n/2)
n2=floor(n/4)
n3=n-n1-n2
ii = sample(1:n,n)
train = df4[ii[1:n1],]
val = df4[ii[n1+1:n2],]
trainval = rbind(train,val)
test = df4[ii[n1+n2+1:n3],]
```

```
set.seed(122)
boostfit1 = gbm(target~.,data = train,distribution="bernoulli",
interaction.depth = 4, n.trees = 1000,shrinkage = 0.2)
summary(boostfit1)
```

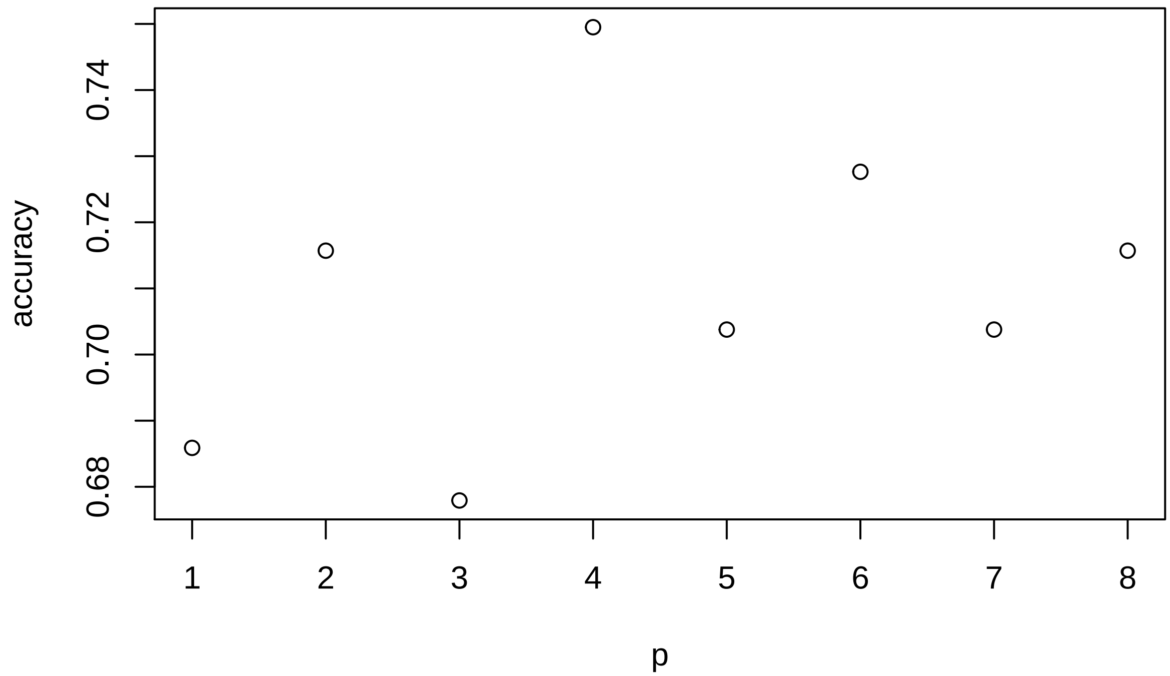



```
##                                 var  rel.inf
## speechiness           speechiness 18.63915
## valence                   valence 17.88512
## duration_ms           duration_ms 16.91190
## acousticness         acousticness 16.29752
## danceability         danceability 15.13541
## instrumentalness instrumentalness 15.13090
```

```
boostfit2 = gbm(target~.,data=train,distribution = "bernoulli",
interaction.depth = 4, n.trees = 1000,shrinkage = 0.001)
boostfit3 = gbm(target~.,data=train,distribution = "bernoulli",
interaction.depth =4, n.trees = 5000,shrinkage = 0.2)
boostfit4 = gbm(target~.,data=train,distribution = "bernoulli",
interaction.depth = 4, n.trees = 5000, shrinkage = 0.001)
boostfit5 = gbm(target~.,data=train,distribution="bernoulli",
interaction.depth = 10, n.trees = 1000,shrinkage = 0.2)
boostfit6 = gbm(target~.,data=train,distribution = "bernoulli",
interaction.depth = 10, n.trees = 1000,shrinkage = 0.001)
boostfit7 = gbm(target~.,data=train,distribution = "bernoulli",
interaction.depth = 10, n.trees = 5000,shrinkage = 0.2)
boostfit8 = gbm(target~.,data=train,distribution = "bernoulli",
interaction.depth = 10, n.trees = 5000,shrinkage = 0.001)
```

Predict on the test. We choose the probability threshold to be 0.5 to classify probability greater than 0.5 as the class '1', '0' otherwise.

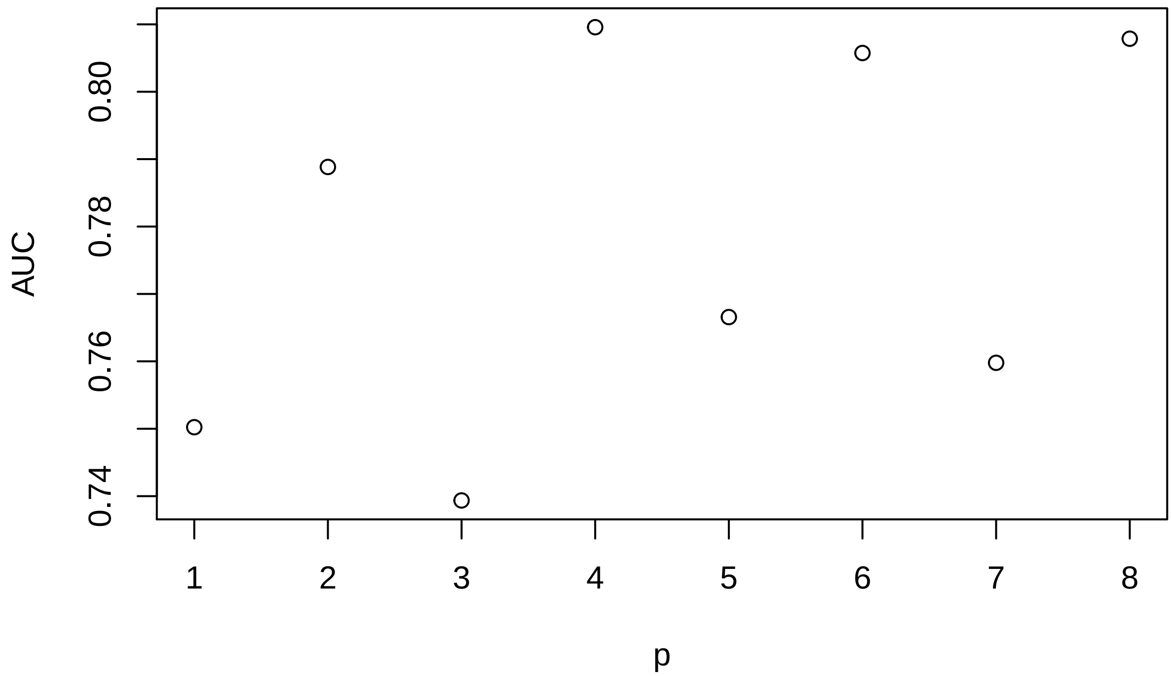
We then calculate the accuracy of all models' prediction on the validation set. According to the accuracy plot, the best result belongs to the fourth model with maximum depth of 4, 5000 trees, and shrinkage of



0.001.

We also calculated the AUC for all 8 models and plotted the AUCs. Agian, we confirm that the fourth model is the best. So we will use the fourth one to predict on the test set.

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

## Setting direction: controls < cases

## Area under the curve: 0.7502

## Setting levels: control = 0, case = 1
## Setting direction: controls < cases

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

## Setting direction: controls < cases

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

## Setting direction: controls < cases

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

## Setting direction: controls < cases

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

## Setting direction: controls < cases

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

## Setting direction: controls < cases

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

## Setting direction: controls < cases

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

## Setting direction: controls < cases
```



Predicting on the test set, we get accuracy of 0.72.

```
boostfit = gbm(target~.,data = trainval ,distribution="bernoulli",
interaction.depth = 4, n.trees = 5000,shrinkage = 0.001)
boostvalpred = predict(boostfit, newdata = test, n.trees = 5000)
pred_test = ifelse(boostvalpred>0.5,1,0)
confusionMatrix(as.factor(pred_test),as.factor(test$target))$overall['Accuracy']
```

```
##  Accuracy
## 0.7017893
```

5 Conclusions