```
setwd("/Users/kirsten 1/Documents/Masters Programs/MIS 545 Data Mining - UofA/Project/fma_metadata_full")

#M_index = c('track_genre_top', 'track_bitrate', 'chroma_cens', "chroma_cqt", 'chroma_stft', 'mfcc', 'rmse', 'spectral_bandwidth', 'spectral_centroid', 'spectral_contrast', 'spectral_rolloff', 'tonnetz', 'zcr')
#M = Music[ , M_index]
library(MASS)
Index = sample(nrow(M), size = 2000, replace = FALSE)
M2 = M[Index, ]
M2$track_genre_top = droplevels(as.factor(M2$track_genre_top))
M2$track_genre_top = as.numeric(M2$track_genre_top)
head(M2$track_genre_top)
nrow(!complete.cases(M2))

library(party)
if (!require(C50)) {
        install.packages("C50")
  }
###   ROC curve\n
 if (!require(pROC)) {
        install.packages("pROC")
  }
library(C50)
library(pROC)

boxplot(track_bitrate ~ track_genre_top, data = M)
plot(rmse.4 ~ track_genre_top, data = M)

#############################################################################
####################################
##repeat with this file
Song = read.csv("Song.csv", na.strings= c("", " ", "  ", "   "), header=TRUE, blank.lines.skip=TRUE, stringsAsFactors = FALSE)
Song$track_genre_top = droplevels(as.factor(Song$track_genre_top))
#Song$track_genre_top = as.numeric(Song$track_genre_top)
summary(Song)
str(Song)
head(Song)
summary(Song$track_genre_top)
names(Song)
```

```
M_index = c('track_genre_top', 'track_bitrate', 'chroma_cens', "chroma_cqt",
'chroma_stft', 'mfcc', 'rmse', 'spectral_bandwidth', 'spectral_centroid',
'spectral_contrast', 'spectral_rolloff', 'tonnetz', 'zcr')

M2_index = c('track_genre_top', 'track_bitrate', 'chroma_cens', 'chroma_stft', 'mfcc',
'rmse', 'spectral_bandwidth', 'spectral_centroid', 'spectral_contrast', 'spectral_rolloff',
'zcr')  ##removed tonnetz and cqt

M3_index = c('track_genre_top', 'track_bitrate', 'chroma_stft', 'spectral_bandwidth',
'spectral_centroid', 'spectral_rolloff')  ##removed tonnetz and cqt, cens, contrast, mfcc,
zcr, rmse

Test = Song [3:4]
head (Test)


S = Song[ , M3_index]

SBackup = S
names(S)
head(S)
str(S)
S$track_genre_top = as.numeric(S$track_genre_top)

Top <- ggplot(data = S, aes(, y = track_genre_top))
Top

Top + geom_point(alpha = .25, size = 5) +
# Cluster centers, colored black:
geom_point(data = centers, aes(, y = track_genre_top), size = 4,
color = 'black') +
# Cool colors for each cluster:
#scale_color_gradientn(colours = rainbow(num_cluster)) +
# Add a title, align to the center
theme(plot.title = element_text(hjust = 0.5)) +
ggtitle("GGPLOT")


library(MASS)
fits = lm(track_genre_top ~ . , data = S)
steps = stepAIC(fits, direction="both")
steps$anova
short = names(unlist(steps[[1]]))
short
```

```
library(party)
library(varImp)

cfs = cforest(track_genre_top ~ . , data = S, control=cforest_unbiased(mtry=12,
ntree=50))
varimp(cfs)


str(S$track_genre_top)

if(!require(ggplot2)){
install.packages("ggplot2")
}
library(ggplot2)
if(!require(animation)){
install.packages("animation")
}
 library(animation)
if(!require(fpc)){
install.packages("fpc")
}
library(fpc)



####regression. Do I need to normalize data first or ??
normIt <- function(feature){
            normalized <- ((feature - min(feature)) / (max(feature) - min(feature)))
            return (normalized)
   }
   head(S)
nor_S <- apply(S[-1], 2, FUN = normIt)
nor_S <- as.data.frame(nor_S)
head(nor_S)
Nor_S = cbind(nor_S, S$track_genre_top)
head(Nor_S)
summary(Nor_S)
Nor_S$track_genre_top= as.numeric(Nor_S$track_genre_top)

Nor_S$genre_nor <- apply(Nor_S[13], 2, FUN = normIt)
names(Nor_S)[13] = "track_genre_top"
names(Nor_S)[6] = "track_genre_top"
```

```
names(Nor_S)[13] = "track_bitrate"
head(Nor_S)

#Nor_S$norm_bitrate <- apply(Nor_S[13], 2, FUN = normIt)
############################################################
try hierarchical clustering

head(Nor_S)
head(S)
install.packages("cluster")
library(cluster)
HC = cbind(Nor_S$spectrall_rolloff, Nor_S$track_genre_top)
HC = daisy(HC, metric = "gower")
hc = hclust(HC, method = "complete")
hc
Sm = daisy(S, metric = "gower")
summary(hc)
Sm
plot(hc, hang = .1)


hc = hclust(daisy(nor_S, metric = "gower"))

############################################################
Clustering with Kmeans - using 2 values
SR = cbind(Nor_S$spectral_rolloff, Nor_S$genre_nor)
SR <- as.data.frame(SR)
head(SR)
names(SR)[1] = "Spectral_Rolloff"
names(SR)[2] = "Genre"

K5 = kmeans(SR,5)
class(K5)
str(K5)
kmeans.totwith <- function(dataset, number_of_centers){
            km <- kmeans(dataset, number_of_centers)
            km$tot.withinss
      }
K2 = kmeans.totwith(SR, 2)
K2
K3 = kmeans.totwith(SR, 3)
K3
K4 = kmeans.totwith(SR, 4)
```

```
K4
K5 = kmeans.totwith(SR, 5)
K5

kmeans.distortion <- function(dataset, maxk){
            vec <- as.vector(1:maxk)
   vec[1:maxk] <- sapply(1:maxk, kmeans.totwith, dataset = dataset)
         return (vec)
       }
maxk <- 7
dis_vct <- kmeans.distortion(SR, maxk)
#   Elbow Curve
plot(1:maxk,
     dis_vct,
     type = 'b',
     col = 'blue',
     xlab = "Number of cluster",
     ylab = "Distortion",
     main = "Elbow Curve for Spectral Rolloff"
      )
num_cluster = 3

result <- kmeans.ani(SR, num_cluster)


dis_vct <- kmeans.distortion(SR, maxk)
#   Elbow Curve
plot(1:maxk,
     dis_vct,
     type = 'b',
     col = 'blue',
     xlab = "Number of cluster",
     ylab = "Distortion"
      )
result <- kmeans.ani(SR, num_cluster)
centers <- as.data.frame(result$centers)
counts <- aggregate(SR, by = list(result$cluster), FUN = length)[, 2]
S2 = S
S2$cluster = result$cluster
head(S2$cluster)
plot2 <- ggplot(data = SR, aes(x = Spectral_Rolloff, y = Genre, color = result$cluster))
plot2

plot2 + geom_point(alpha = .25, size = 5) +
```

```
# Cluster centers, colored black:
geom_point(data = centers, aes(x = Spectral_Rolloff, y = Genre), size = 4,
color = 'black') +
# Cool colors for each cluster:
scale_color_gradientn(colours = rainbow(num_cluster)) +
# Add a title, align to the center
theme(plot.title = element_text(hjust = 0.5)) +
ggtitle("K-means clusters")

#ds <- dbscan(SR[,c(1,2)], eps = .4, MinPts = 4, scale = TRUE, showplot = 1, seeds = TRUE,
method = "hybrid")
ds
#table(S2$cluster, SR$cluster)   #dbscan is only showing 2 clusters vs 3
######################################################################
clustering..
head(Nor_S)
head(nor_S)
MF = cbind(Nor_S$spectral_rolloff, Nor_S$track_bitrate)
MF <- as.data.frame(MF)
head(MF)
#names(S3)[1] = "Spectral_Rolloff"
names(MF)[1] = "Rolloff"
names(MF)[2] = "Bitrate"

K = kmeans(MF, centers = 3, nstart = 25)
str(K)
K
install.packages("factoextra")
library(factoextra)
fviz_cluster(K, data=nor_S, choose.var = c("spectral_rolloff", "spectral_bandwidth"))

K5 = kmeans(MF,5)
class(K5)
str(K5)
kmeans.totwith <- function(dataset, number_of_centers){
            km <- kmeans(dataset, number_of_centers)
            km$tot.withinss
      }
kmeans.distortion <- function(dataset, maxk){
           vec <- as.vector(1:maxk)
   vec[1:maxk] <- sapply(1:maxk, kmeans.totwith, dataset = dataset)
        return (vec)
      }
maxk <- 7
```

```
dis_vct <- kmeans.distortion(MF, maxk)
#   Elbow Curve
plot(1:maxk,
     dis_vct,
     type = 'b',
     col = 'blue',
     xlab = "Number of cluster",
     ylab = "Distortion",
     main = "Elbow Curve for Spectral Rolloff and Bitrate"
      )
num_cluster = 3

result <- kmeans.ani(MF, num_cluster)
centers <- as.data.frame(result$centers)
counts <- aggregate(MF, by = list(result$cluster), FUN = length)[, 2]
S4 = S
MF$cluster = result$cluster
head(S2$cluster)
plot2 <- ggplot(data = MF, aes(x = Rolloff, y = Bitrate, color = result$cluster))
plot2

plot2 + geom_point(alpha = .25, size = 5) +
# Cluster centers, colored black:
geom_point(data = centers, aes(x = Rolloff, y = Bitrate), size = 4,
color = 'black') +
# Cool colors for each cluster:
scale_color_gradientn(colours = rainbow(num_cluster)) +
# Add a title, align to the center
theme(plot.title = element_text(hjust = 0.5)) +
ggtitle("K-means clusters")




############################################################
##graphs.....
plot(S$track_bitrate ~ S$track_genre_top)
plot(S$spectral_rolloff ~ S$track_genre_top)
plot(S$spectral_centroid ~ S$track_genre_top)
plot(S$chroma_stft ~ S$track_genre_top)
plot(S$mfcc ~ S$track_genre_top)
plot(S$rmse ~ S$track_genre_top)
```

```
boxplot(spectral_rolloff ~ track_genre_top, data=S, col="pink", main="Music Genre by Spectral Rolloff", ylab = "Spectral Rolloff", xlab = "Music Genre")
boxplot(spectral_centroid ~ track_genre_top, data=S, col="blue", main="Music Genre by Spectral Centroid", ylab = "Spectral Centroid", xlab = "Music Genre")
library(vioplot)
head(S$track_genre_top)
v1 = S$spectral_rolloff[S$track_genre_top == "Folk"]
v2 = S$spectral_rolloff[S$track_genre_top == "Classical"]
v3 = S$spectral_rolloff[S$track_genre_top == "Hip-Hop"]


vioplot(v1, v2, v3, names = c("Folk", "Classical", "Hip-Hop"), col=c("lightgreen", "lightblue", "palevioletred"), main = "Music Genre by Spectral Rolloff")

v1 = S$spectral_centroid[S$track_genre_top == "Folk"]
v2 = S$spectral_centroid[S$track_genre_top == "Classical"]
v3 = S$spectral_centroid[S$track_genre_top == "Hip-Hop"]


vioplot(v1, v2, v3, names = c("Folk", "Classical", "Hip-Hop"), col=c("lightgreen", "lightblue", "palevioletred"), main = "Music Genre by Spectral Centroid")

library(ggplot2)
library(animation)
library(fpc)

g = ggplot(S, aes(spectral_rolloff, track_genre_top)) + geom_point(color = 'firebrick')  +
        # cool colors for each cluster:
        scale_color_gradientn(colours = rainbow(3)) +
        # add a title, align to the center
        theme(plot.title = element_text(hjust = 0.5)) +
        ggtitle("Spectral Rolloff")
 g

 g2 = ggplot(S, aes(spectral_rolloff, spectral_centroid, color = factor(track_genre_top))) + geom_point()  +
        # cool colors for each cluster:
        scale_color_gradientn(colours = rainbow(3)) +
        # add a title, align to the center
        theme(plot.title = element_text(hjust = 0.5)) +
        ggtitle("Spectral Rolloff")
g2

plotR = ggplot(data = S, aes(x = spectral_rolloff, y = track_genre_top))
```

```
plotR + geom_point(alpha = .25, size = 5) +
            # cluster centers, colored black:
            geom_point(data = S, aes(x = spectral_rolloff, y = track_genre_top), size = 5,
color = 'firebrick')  +
            # cool colors for each cluster:
            scale_color_gradientn(colours = rainbow(3)) +
            # add a title, align to the center
            theme(plot.title = element_text(hjust = 0.5)) +
            ggtitle("Spectral Rolloff")


summary(Nor_S)
str(Nor_S)

Nor_S$track_genre_top = as.factor(Nor_S$track_genre_top)

reg2 <- glm(track_genre_top ~ . , data = Nor_S, family = binomial(), control =
glm.control(maxit=50) )

regstep=stepAIC(reg2, direction = "both")
###  model detail
summary(reg)
###   validate test dataset
##library(oddsratio)
##or_glm(data = test, model = reg, incr = list(class2nd=1, ##class3rd=1, classcrew=1,
agechild=1, sexmale=1), CI = 0.95)


library(C50)
library(pROC)

S = Song[ , M_index]  ## get factor back to genre
size <- floor(0.8 * nrow(S))
size
###   randomly decide which ones for training
training_index <- sample(nrow(S), size = size, replace = FALSE)
train <- S[training_index,]
test <- S[-training_index,]
head(S)
###   names of variables that used for prediction
var_names <- names(S)[-1]  #this gives column titles of everything except  genre
var_names
```

```
evaluation <- test
###evaluation$track_genre_top = as.numeric(evaluation$track_genre_top)
###   return risk instead of classification
evaluation$prob <- predict(fits, newdata = evaluation)
head(evaluation)
summary(evaluation$prob)
#   Calculate baseline
countF <- nrow(subset(S, S$track_genre_top == "Folk") )
countF
baselineF <- countF / nrow(evaluation$prob == "Folk")
baselineF

##Decision TREE
##Decision Tree wants factor outcome - prep...
if (!require(C50)) {
        install.packages("C50")
   }
###   ROC curve\n
 if (!require(pROC)) {
        install.packages("pROC")
   }
   library(C50)
   library(pROC)
str(S)
S$track_genre_top = as.factor(Nor_S$track_genre_top)
#M$track_genre_top = droplevels(as.factor(M$track_genre_top))
train$track_genre_top = droplevels(as.factor(train$track_genre_top))
test$track_genre_top = droplevels(as.factor(test$track_genre_top))

install.packages("rpart")
install.packages("rpart.plot")
library(rpart)
library(rpart.plot)


# fit the model
dt <- C5.0(x = train[, var_names], y = train$track_genre_top)
# see the summary of model
summary(dt)
plot(dt)
tree= rpart(train$track_genre_top ~ . , data = train, cp=.01)
rpart.plot(tree, box.palette="RdBu", shadow.col = "gray", nn=TRUE)
```

```
tree= rpart(train$track_genre_top ~ . , data = train, cp=.007)
rpart.plot(tree, box.palette="RdBu", shadow.col = "gray", nn=TRUE)

###   now, validate test
##    predict() method returns a vector of result
dt_pred <- predict(dt, newdata = test)
###   merger dt_prediction value to test dataset
dt_evaluation <- cbind(test, dt_pred)
head(test)
head(dt_evaluation)
### compare dt_prediction result to actual value
dt_evaluation$correct <- ifelse(dt_evaluation$track_genre_top ==
dt_evaluation$dt_pred, 1, 0)
head(dt_evaluation)
###    accuracy rate
sum(dt_evaluation$correct) / nrow(dt_evaluation)
###   confusion matrix
table(dt_evaluation$track_genre_top, dt_evaluation$dt_pred)
###   True Positive Rate (Sensitivity)  TPR = TP / P
###   = count of true positive dt_prediction divided by total positive truth

cm = confusionMatrix( dt_evaluation$dt_pred , dt_evaluation$track_genre_top)
cm

TPR_F <- sum(dt_evaluation$dt_pred == 'Folk' & dt_evaluation$track_genre_top ==
'Folk') / sum(dt_evaluation$track_genre_top == 'Folk')
TPR_F
TPR_C <- sum(dt_evaluation$dt_pred == 'Classical' & dt_evaluation$track_genre_top ==
'Classical') / sum(dt_evaluation$track_genre_top == 'Classical')
TPR_C
TPR_H <- sum(dt_evaluation$dt_pred == 'Hip-Hop' & dt_evaluation$track_genre_top ==
'Hip-Hop') / sum(dt_evaluation$track_genre_top == 'Hip-Hop')
TPR_H
###   True Negative Rate (Specificity)  TNR = TN / N
###   = count of true negative dt_prediction divided by total negative truth
#TNR_F <- sum(dt_evaluation$dt_pred != 'Folk' & dt_evaluation$track_genre_top !=
'Folk') / sum(dt_evaluation$track_genre_top != 'Folk')
#TNR_F
#TNR_C <- sum(dt_evaluation$dt_pred != 'Classical' &
dt_evaluation$track_genre_top != 'Classical') / sum(dt_evaluation$track_genre_top !=
'Classical')
#TNR_C
```

```
#TNR_H <- sum(dt_evaluation$dt_pred != 'Hip-Hop' &
dt_evaluation$track_genre_top != 'Hip-Hop') / sum(dt_evaluation$track_genre_top !=
'Hip-Hop')
#TNR_H

###   False Positive Rate  (1 - Specificity)  FPR = FP / N
###   = count of false positive dt_prediction divided by total negative truth
###   = sum(dt_evaluation$dt_pred == 'yes'& dt_evaluation$if_affair == 'no') /
sum(dt_evaluation$if_affair_50K == 'no')
FPR <- 1 - TNR
FPR
###   False Negative Rate FNR (1 - Sensitivity)  FNR = FN / P
###   = count of false negative dt_prediction divided by total positive truth
###   = sum(dt_evaluation$dt_pred == 'no'& dt_evaluation$if_affair == 'yes') /
sum(dt_evaluation$if_affair == 'yes')
FNR <- 1 - TPR
FNR
###   dt_precision equals
###   = number of true positive dt_prediction  / total positive dt_prediction


dt_precision_F <- sum(dt_evaluation$track_genre_top == 'Folk' &
dt_evaluation$dt_pred == 'Folk') / sum(dt_evaluation$dt_pred == 'Folk')
dt_precision_F
dt_precision_C <- sum(dt_evaluation$track_genre_top == 'Classical' &
dt_evaluation$dt_pred == 'Classical') / sum(dt_evaluation$dt_pred == 'Classical')
dt_precision_C
dt_precision_H <- sum(dt_evaluation$track_genre_top == 'Hip-Hop' &
dt_evaluation$dt_pred == 'Hip-Hop') / sum(dt_evaluation$dt_pred == 'Hip-Hop')
dt_precision_H
average = (dt_precision_F + dt_precision_C + dt_precision_H) / 3
average
###   dt_recall equals = TPR
###   = true positive dt_prediction / total true positive
dt_recall_F <- sum(dt_evaluation$track_genre_top == 'Folk' & dt_evaluation$dt_pred
== 'Folk') / sum(dt_evaluation$track_genre_top == 'Folk')
dt_recall_F
dt_recall_C <- sum(dt_evaluation$track_genre_top == 'Classical' &
dt_evaluation$dt_pred == 'Classical') / sum(dt_evaluation$track_genre_top ==
'Classical')
dt_recall_C
dt_recall_H <- sum(dt_evaluation$track_genre_top == 'Hip-Hop' &
dt_evaluation$dt_pred == 'Hip-Hop') / sum(dt_evaluation$track_genre_top == 'Hip-
Hop')
```

```
dt_recall_H

 ###  F score
F <- 2 * dt_precision * dt_recall / (dt_precision + dt_recall)
F

 F1_C = (2*(dt_precision_C * dt_recall_C)) / (dt_precision_C + dt_recall_C)
 F1_C
 F1_F = (2*(dt_precision_F * dt_recall_F)) / (dt_precision_F + dt_recall_F)
 F1_F
F1_H = (2*(dt_precision_H * dt_recall_H)) / (dt_precision_H + dt_recall_H)
 F1_H
F1_Average = (F1_C + F1_H + F1_F) / 3
F1_Average

#install.packages("e1071")
library(e1071)

Balance.model = naiveBayes(track_genre_top ~ . , data = train)
Balance.model
plot(Balance.model)

Balance.predict = predict(Balance.model, test) # type = 'class')
Balance.predict
results = data.frame(actual = test[ , 'track_genre_top'], predicted = Balance.predict)
table(results)

plot(results)
head(results$predicted)
head(results$actual)
nrow(results[results$predicted == results$actual,])/nrow(results)
plot(results)

results$correct <- ifelse(results$actual == results$predicted, 1, 0)
head(results)
###    accuracy rate
sum(results$correct) / nrow(results)
install.packages("caret")
###   confusion matrix
library(caret)
cm = confusionMatrix(results$predicted, results$actual)
cm

table(dt_evaluation$track_genre_top, dt_evaluation$dt_pred)
```

```
###   True Positive Rate (Sensitivity)  TPR = TP / P
###   = count of true positive dt_prediction divided by total positive truth

NB_precision_F <- sum(results$actual == 'Folk' & results$predicted == 'Folk') /
sum(results$predicted == 'Folk')
NB_precision_F
NB_precision_C <- sum(results$actual == 'Classical' & results$predicted == 'Classical') /
sum(results$predicted == 'Classical')
NB_precision_C
NB_precision_H <- sum(results$actual == 'Hip-Hop' & results$predicted == 'Hip-Hop') /
sum(results$predicted == 'Hip-Hop')
NB_precision_H
averageNB = (NB_precision_F + NB_precision_C + NB_precision_H) / 3
averageNB
###   dt_recall equals = TPR
###   = true positive dt_prediction / total true positive
NB_recall_F <- sum(results$actual == 'Folk' & results$predicted == 'Folk') /
sum(results$actual == 'Folk')
NB_recall_F
NB_recall_C <- sum(results$actual == 'Classical' & results$predicted == 'Classical') /
sum(results$actual == 'Classical')
NB_recall_C
NB_recall_H <- sum(results$actual == 'Hip-Hop' & results$predicted == 'Hip-Hop') /
sum(results$actual == 'Hip-Hop')
NB_recall_H
 averageR_NB =  (NB_recall_F + NB_recall_C  + NB_recall_H) / 3
 averageR_NB
 ###  F score
#F <- 2 * dt_precision * dt_recall / (dt_precision + dt_recall)
F

 F1_RC = (2*(NB_precision_C * NB_recall_C)) / (NB_precision_C + NB_recall_C)
 F1_RC
 F1_RF = (2*(NB_precision_F * NB_recall_F)) / (NB_precision_F + NB_recall_F)
 F1_RF
F1_RH = (2*(NB_precision_H * NB_recall_H)) / (NB_precision_H + NB_recall_H)
 F1_RH
F1_RAverage = (F1_RC + F1_RH + F1_RF) / 3
F1_RAverage


#Process Echonest.csv
read.csv
```

```
#############################################################################
########################################
###Below is data processing from first times###
###Do first time. Then start with Music3###
Tracks = read.csv("tracks.csv", na.strings= c("", " ", "  ", "   "), header=TRUE,
blank.lines.skip=TRUE)
summary(Tracks$track.7)
nrow(Tracks)
names(Tracks)
Tracks1 = Tracks[complete.cases(Tracks$track.7), ]
summary(Tracks1$track.7)
col_index = c(1,9,12,27,32,33,34,41,42,43,45,48,53)
summary(Tracks1$track.7)
nrow(Tracks1)
names(Tracks1)
Tracks2 = Tracks1[ ,col_index]
names(Tracks2)
nrow(Tracks2)
colnames(Tracks2) = c("track_ID", "album_listens", "album_title", "artist_name",
"set_split", "set_subset", "track_bitrate", "track_genre_top", "track_genres",
"track_genres_all", "track_interest", "track_listens", "track_title")
nrow(Tracks2)
Tracks2 = Tracks2[complete.cases(Tracks2$track_ID), ]
summary(Tracks2$track_genre_top)
nrow(Tracks2[complete.cases(Tracks2$track_genre_top), ])
nrow(Tracks2[complete.cases(Tracks2), ])

write.csv(Tracks2, "Tracks2.csv")
Tracks2 = read.csv("Tracks2.csv")

###Read in Track2 from now on from here##
Tracks2 = read.csv("Tracks2.csv", na.strings= c("", " ", "  ", "   "), header=TRUE,
stringsAsFactors = FALSE, blank.lines.skip=TRUE)
nrow(Tracks2)
names(Tracks2)
summary(Tracks2$track_genre_top)
Tracks2$track_genre_top = as.factor(Tracks2$track_genre_top)
```

```
Features = read.csv("features.csv", na.strings= c("", " ", "  ", "   "), header=TRUE,
stringsAsFactors = FALSE, blank.lines.skip=TRUE)
colnames(Features)[1] = "track_ID"
names(Features)
col_index = c('track_ID', 'chroma_cens', "chroma_cqt", 'chroma_stft', 'mfcc', 'rmse',
'spectral_bandwidth', 'spectral_centroid', 'spectral_contrast', 'spectral_rolloff',
'tonnetz', 'zcr')
Feat = Features[ , col_index]
names(Feat)

nrow(Feat)
Feat = Feat[complete.cases(Feat$track_ID), ]
nrow(Feat)

Music = merge(x=Music3, y=Feat, by = "track_ID", all.x = FALSE)
nrow(Music)
summary(Music)
str(Music)

write.csv(Music, "Music.csv") #file with 3 genres and 2k songs each genre

##Trying with 3 different Genres...
Class = Tracks2[Tracks2$track_genre_top == "Classical", ]
nrow(Class)
summary(Class)
Folk = Tracks2[Tracks2$track_genre_top == "Folk", ]
nrow(Folk)
summary(Folk)
Hip = Tracks2[Tracks2$track_genre_top == "Hip-Hop", ]
nrow(Hip)
Songs = rbind(Class, Folk, Hip)
nrow(Songs)
summary(Songs)
Songs$track_genre_top = droplevels(as.factor(Songs$track_genre_top))

write.csv(Songs, "Songs.csv") #file with 3 genres Folk, Classical and Hip-Hop
Features = read.csv("features.csv", na.strings= c("", " ", "  ", "   "), header=TRUE,
stringsAsFactors = FALSE, blank.lines.skip=TRUE)
colnames(Features)[1] = "track_ID"
names(Features)
col_index = c('track_ID', 'chroma_cens', "chroma_cqt", 'chroma_stft', 'mfcc', 'rmse',
'spectral_bandwidth', 'spectral_centroid', 'spectral_contrast', 'spectral_rolloff',
'tonnetz', 'zcr')
Feat = Features[ , col_index]
```

```
names(Feat)

nrow(Feat)
Feat = Feat[complete.cases(Feat$track_ID), ]
nrow(Feat)
write.csv(Feat, "Sm_Feat.csv")  # smaller version of Features with WAY less columns for
less data

Song = merge(x=Songs, y=Feat, by = "track_ID", all.x = FALSE)
nrow(Song)
summary(Song)
str(Song)

write.csv(Song, "Song.csv") #file with 3 genres, less columns and 7595 records

################################################
#analyze Echonest.csv
Echo = read.csv("echonest.csv", na.strings= c("", " ", "  ", "   "), header=TRUE,
blank.lines.skip=TRUE, stringsAsFactors = FALSE)
head(Echo)
summary(Echo)
names(Echo)[1] = "track_ID"
Echo = Echo[ , c(1:9,27:250)]
nrow(Echo)
EchoM = Song = merge(x=Tracks2, y=Echo, by = "track_ID", all.x = FALSE)
summary(EchoM)


######ASSOCIATED RULE MINING####

if(!require(arules)){
install.packages("arules")
}
library(arules)

if(!require(arulesViz)){
install.packages("arulesViz")
}
library(arulesViz)

if(!require(igraph)){
install.packages("igraph")
}
library(igraph)
```

```
if(!require(visNetwork)){
install.packages("visNetwork")
}
library(visNetwork)


if(!require(plyr)){
install.packages("plyr")
}
library(plyr)
str(S)
Sub = S[3:13]
head(Sub)
str(Sub)
Cat = discretizeDF(Sub)

head(Cat)
New = cbind(S$track_genre_top, Cat)
head(New)
names(New)[1] = "track_genre_top"


# generate association rules
rules <- apriori(New, parameter = list(sup = 0.35, conf = 0.8, target = "rules"),
appearance = list(default = 'lhs', rhs = c('track_genre_top=Folk')))  #,
'track_genre_top=Classical', 'track_genre_top=Hip-Hop')))
rules <- apriori(New, parameter = list(sup = 0.15, conf = 0.6, target = "rules"),
appearance = list(default = 'lhs', rhs = c('track_genre_top=Folk',
'track_genre_top=Classical', 'track_genre_top=Hip-Hop')))
rules

rules <- sort(rules, decreasing = TRUE, by = "support")

inspect(rules[1:7])

rulesC <- sort(rules, decreasing = TRUE, by = "confidence")

inspect(rulesC[1:7])

rulesL <- sort(rules, decreasing = TRUE, by = "lift")

inspect(rulesL[1:7])
```

```
top7_rules <- sort(rules, decreasing = TRUE, by = "support")[1:7]

top7_Lift_rules <- sort(rulesL, decreasing = TRUE, by = "lift")[1:7]

#   overview of rules
plot(top7_rules, shading = "lift", control=list(main = "Two-key plot of Song Genre
Prediction"))

#   Targeting Music Genre
        rule_F <- apriori(New, parameter = list(sup = 0.1, conf = 0.6, target
                = "rules"), appearance = list(default = 'lhs', rhs = 'track_genre_top=Folk'))

        rule_F <- sort(rule_F, decreasing = TRUE, by = "confidence")
        inspect(rule_F[1:5])

        top5_Folk_rules <- sort(rule_F, decreasing = TRUE, by = "confidence")[1:5]

rule_C <- apriori(New, parameter = list(sup = 0.055, conf = 0.5, target
                = "rules"), appearance = list(default = 'lhs', rhs =
'track_genre_top=Classical'))


        rule_C <- sort(rule_C, decreasing = TRUE, by = "confidence")
inspect(rule_C[1:5])

top5_Class_rules <- sort(rule_C, decreasing = TRUE, by = "confidence")[1:5]


rule_H <- apriori(New, parameter = list(sup = 0.15, conf = 0.6, target
                = "rules"), appearance = list(default = 'lhs', rhs = 'track_genre_top=Hip-
Hop'))

rule_H <- sort(rule_H, decreasing = TRUE, by = "confidence")
inspect(rule_H[1:2])



#   parallel coordinates plot
plot(top7_rules, method = "paracoord", shading = "support", title = "Graph for Overall
Top 7 Rules ")

#   create a basic graph structure
ig <- plot(top7_rules, method = "graph",  title = "Graph for Overall Top 7 Rules ")
igF <- plot(top5_Folk_rules, method = "graph")
```

```
igF <- plot(top5_Class_rules, method = "graph")


#   use igraph
ig_df <- get.data.frame(ig, what = "both")
#   generate nodes

nodes <- data.frame(id = ig_df$vertices$name,
                    #   the size of nodes: could change to lift or confidence
                    value = ig_df$vertices$support,
                    title = ifelse(ig_df$vertices$label == "", ig_df$vertices$name,
                    ig_df$vertices$label), ig_df$vertices)

#   generate edges
edges <- ig_df$edges

#   directed network    manipulate network
network <- visNetwork(nodes, edges) %>%
                          visOptions(manipulation = TRUE) %>%   #    manipulate network
                          visEdges(arrows = 'to', scaling = list(min = 2, max = 2)) %>%

                                                                  #   directed network
                          visInteraction(navigationButtons = TRUE)   # navigation buttons
network
```