ST344 Technical Appendix

Group 2 - 1729346, 1711656, 1703980, 1601492, 1941456

26/11/2019

Exploratory Data Analysis

Libraries

The libraries used to complete this task were:

```
library(readxl)
library(dplyr)
library(ggplot2)
library(corrplot)
library(tidyverse)
library(lubridate)
```

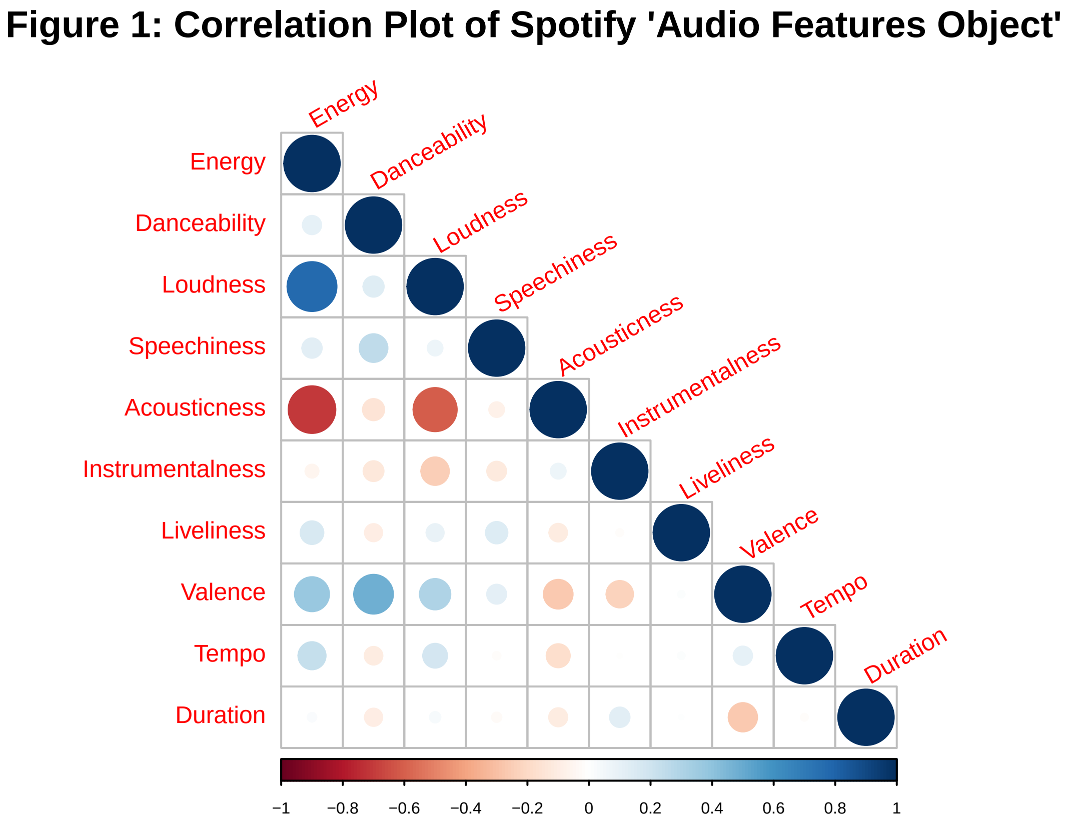
Data Manipulations

```
# Extracting Data
Spotify <- read_excel("edited_spotify.xlsx")

MyData <- {Spotify %>%
    group_by(Artist, AlbumName, AlbumReleaseDate) %>%
    summarize(AlbumLoudness = mean(TrackLoudness),
              AlbumTempo = mean(TrackTempo),
              AlbumAcousticness = mean(TrackAcousticness))}

# Date Parsing
MyData$AlbumReleaseDate <- parse_date_time(MyData$AlbumReleaseDate,
                                           orders = c("y", "ym", "ymd"))
```

Correlation Plot

Following is the code for the correlation plot produced.

```
Spotify1 <- Spotify[sample(nrow(Spotify),20),]

# Subsetting Data
NewData <- Spotify %>%
  select(TrackEnergy,TrackDanceability,TrackLoudness,
         TrackSpeechiness,TrackAcousticness,TrackInstrumentalness,
         TrackLiveness,TrackValence,TrackTempo,TrackDuration)

colnames(NewData) <- c("Energy","Danceability","Loudness",
                       "Speechiness","Acousticness","Instrumentalness",
                       "Liveliness","Valence","Tempo","Duration")

cor <- cor(NewData)
corrplot(cor,mar=c(0,0,2,0),type="lower",
         tl.srt = 30, tl.cex = 0.75, cl.cex = 0.5,
         title = "Figure 1: Correlation Plot of Spotify 'Audio Features Object'")
```



Principal Component Analysis

Libraries

The libraries used to complete this task were:

```
library(readxl)
library(dplyr)
library(ggplot2)
library(corrplot)
library(tidyverse)
```

Data Manipulations

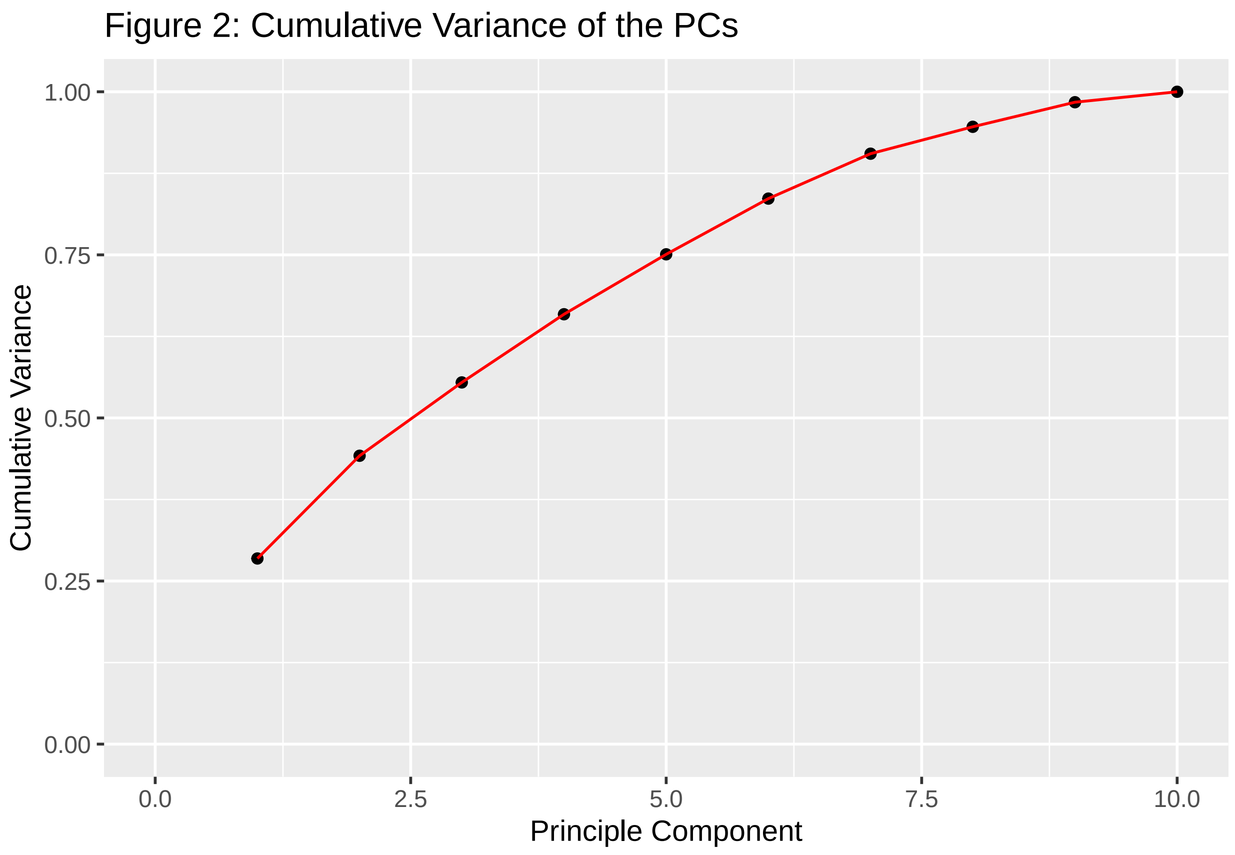
Importing the data and selecting the 10 variables.

```
Spotify <- read_excel("edited_spotify.xlsx")

NewData <- Spotify %>%
  select(TrackEnergy,TrackDanceability,TrackLoudness,
         TrackSpeechiness,TrackAcousticness,TrackInstrumentalness,
         TrackLiveness,TrackValence,TrackTempo,TrackDuration)
```

Conducting PCA

Performing PCA on the data to find the components, standard deviation, variance and the cumulative proportion of variance that is accounted for by each principal component. The plot demonstrates the cumulative variance.

```r
spotify.pca <- prcomp(NewData, center = TRUE,scale. = TRUE)

std_dev <- spotify.pca$sdev

pr_var <- std_dev^2

prop_varex <- pr_var/sum(pr_var)

cumsum <- data.frame(c(1:10),cumsum(prop_varex))

ggplot(cumsum, aes(x = c(1:10), y = cumsum(prop_varex))) +
  geom_point() +
  coord_cartesian(ylim = c(0, 1),xlim=c(0,10)) +
  ggtitle("Figure 2: Cumulative Variance of the PCs") +
  labs(x="Principle Component", y="Cumulative Variance") +
  geom_line(color="red")
```



Task 1

Libraries

The libraries used to complete this task were:

```r
library(ggplot2)
library(dplyr)
library(tidyr)
library(rio)
library(knitr)
library(readxl)
library(lubridate)
library(caret)
library(stringr)
```

Data Manipulations

Standard function used to normalise an input data set:

```r
normalise <- function(dat){
  for(i in 1:ncol(dat)){
    dat[i] <- (dat[i] - min(dat[i])) / (max(dat[i]) - min(dat[i]))
  }
  return(dat)
}
```

The following chunk contains all functions used for data manipulation/generation.

```r
# Function to import the main Spotify data set
albumdatgen <- function(){
  maindat <- read_excel("edited_spotify.xlsx")

  # NA Removal
  maindat <- na.omit({
    maindat %>%
      group_by(Artist, AlbumName, AlbumReleaseDate, ArtistPopularity)
  })

  # Date Parsing
  maindat$AlbumReleaseDate = as.Date(parse_date_time(
    maindat$AlbumReleaseDate, orders = c("y", "ym", "ymd")))

  # Grouping data by albums because there is no data to
      # calculate an individual track's popularity.
  albumdat = {maindat %>%
      group_by(Artist,ArtistPopularity,AlbumName,
               AlbumReleaseDate,AlbumPopularity,ArtistNumFollowers,
               AlbumWeeksOnChart,AlbumWeeksNumberOne,
               AlbumBestChartPosition) %>%
      summarise(AlbumDuration = mean(TrackDuration),
                AlbumDanceability = mean(TrackDanceability),
                AlbumEnergy = mean(TrackEnergy),
                AlbumLoudness = mean(TrackLoudness),
                AlbumSpeechiness = mean(TrackSpeechiness),
                AlbumAcousticness = mean(TrackAcousticness),
                AlbumInstrumentalness = mean(TrackInstrumentalness),
                AlbumLiveness = mean(TrackLiveness),
                AlbumValence = mean(TrackValence),
                AlbumTempo = mean(TrackTempo)
                 )}

  # Calculating the AlbumPopularityRating and ReleaseYear for each album.
  albumdat <- {albumdat %>%
    group_by(Artist, AlbumName, AlbumReleaseDate) %>%
      mutate(AlbumPopularityRating =
                ((ArtistPopularity + AlbumPopularity)/2) +
                floor((ArtistNumFollowers/10^6)) +
                (AlbumWeeksOnChart/100) +
                (AlbumWeeksNumberOne/10) +
                (1 - (AlbumBestChartPosition/100)),
             ReleaseYear = year(AlbumReleaseDate))
  }
  albumdat[10:19] <- normalise(albumdat[10:19])
  return(albumdat)
}

# Training data set generator. This removes test_data from
    # the main data set and returns the data over an appropriate
    # period of release dates.
trainsetgen <- function(test_data, data, year){
  train_data <- anti_join(data, test_data, by = c("Artist", "AlbumName"))

  if(year <= 1964){
    return(filter(train_data, (ReleaseYear <= 1964)))
  }
  else if(year > 1964 & year <= 1975){
    return(filter(train_data, (ReleaseYear > 1964), (ReleaseYear <= 1975)))
  }
  else if(year > 1975 & year <= 1992){
      return(filter(train_data, (ReleaseYear > 1975), (ReleaseYear <= 1992)))
  }
  else if(year > 1992 & year <= 2002){
      return(filter(train_data, (ReleaseYear > 1992), (ReleaseYear <= 2002)))
  }
  else if(year > 2002){
    return(filter(train_data, (ReleaseYear > 2002)))
  }
}

# Random test data set generator. This was initially used to test the
    # predictor using randomly generated test sets.
testsetgen <- function(){
  intrain <- createDataPartition(y = albumdat$AlbumPopularityRating,
                                 p= 0.05, list = FALSE)
  return(albumdat[intrain,])
}
```

The *fetch_test()* function takes a string 'filename' (e.g. "test_albums_2.txt") as input and parses the txt file into a tibble which is compatible with the rest of the operations of this task.

```r
fetch_test <- function(filename){
  source <- paste("test_albums",filename, sep = "/")
  TData <- read.table(source)

  test_data <- matrix(ncol = 2)

  #Word selects all phrases between the commas
  for (i in (1:dim(TData)[1])) {
    test_data <-
      rbind(test_data, c(word(TData[i, 1], 1:2, sep = fixed(': '))))
  }

  test_data <- test_data[-1, ]
  test_data <- as_tibble(test_data)
  colnames(test_data) <- c("Artist", "AlbumName")

  test_data <- filter(albumdat, (Artist %in% as.list(test_data$Artist)) &
                      (AlbumName %in% as.list(test_data$AlbumName)))

  return(test_data)
}
```

Models and Predictors

Both models for this task using the caret package. This package enables training k-nearest neighbours models and provides other built-in useful analytical functions such as principal component analysis.

The *run_knn()* function is used to train the k-nearest neighbours predictors for the model without using principal component analysis. It trains 3 different predictors and uses the average of their predictions as a final prediction for the model. To control the nuances of our model training function, we set the training control *trctrl*, such that the resampling method is 'repeated cross-validation' with 5 resampling iterations and 2 complete sets of folds to compute for our repeated cross-validation.

```r
run_knn <- function(test_data, artistname, album){
  entry <- filter(albumdat, (Artist == artistname)&(AlbumName == album))
  year <- entry$ReleaseYear
  training_set <- trainsetgen(test_data, albumdat, year)

  trctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 2)

  pred1 <- train(AlbumPopularityRating ~ AlbumDanceability +
                   AlbumEnergy + AlbumLiveness +
                   AlbumValence + AlbumLoudness + AlbumTempo, data = training_set,
                 method = "knn", trControl=trctrl,
                 preProcess = c("center", "scale"), tuneLength = 10)

  pred2 <- train(AlbumPopularityRating ~  AlbumAcousticness +
                   AlbumInstrumentalness, data = training_set,
                 method = "knn", trControl=trctrl,
                 preProcess = c("center", "scale"), tuneLength = 10)

  pred3 <- train(AlbumPopularityRating ~ AlbumSpeechiness +
                 AlbumDuration, data = training_set,
                 method = "knn", trControl=trctrl,
                        preProcess = c("center", "scale"), tuneLength = 10)

  poppred1 <-  predict(pred1, newdata = entry)
  poppred2 <-  predict(pred2, newdata = entry)
  poppred3 <-  predict(pred3, newdata = entry)

  poppred <- mean(poppred1, poppred2, poppred3)
  entry <- {entry %>% group_by(Artist, AlbumName) %>%
      mutate(PopularityPrediction = poppred)}

  return(entry$PopularityPrediction)
}
```

The *run_pca.knn()* function is used to train the k-nearest neighbours predictors for the model using principal component analysis. It trains the model using *preProcOptions* which sets the *preProcessing* options and uses the first 6 principal components of the data. This model uses the same training control options as the previous.

```r
run_pca.knn <- function(test_data, artistname, album){
  entry <- filter(albumdat, (Artist == artistname)&(AlbumName == album))
  year <- entry$ReleaseYear
  training_set <- trainsetgen(test_data, albumdat, year)

  trctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 2,
                         preProcOptions = c(pcaComp = 6))

  pred <- train(AlbumPopularityRating ~ AlbumDuration + AlbumDanceability +
                          AlbumEnergy + AlbumLiveness +
                          AlbumValence + AlbumAcousticness +
                          AlbumInstrumentalness + AlbumSpeechiness +
                          AlbumLoudness + AlbumTempo, data = training_set,
                        method = "knn", trControl=trctrl, tuneLength = 10)

  poppred <-  predict(pred, newdata = entry)
  entry <- {entry %>% group_by(Artist, AlbumName) %>%
      mutate(PopularityPrediction = poppred)}

  return(entry$PopularityPrediction)
}
```

predict_popularity() function is a call for training the models with/without pca and finding predictions for a *.txt* test file. The inputs are the filename and a boolean *pca* indicating whether pca is used in the model.

```r
predict_popularity <- function(filename, pca){
  test_data <- fetch_test(filename)

  if(pca){
    test_data <- test_data %>% group_by(Artist, AlbumName) %>%
      mutate(PredictedPopularity = run_pca.knn(test_data, Artist, AlbumName)) %>%
      mutate(AbsolutePercentageError =
               (abs(PredictedPopularity - AlbumPopularityRating)/
                  AlbumPopularityRating)*100)
  }
  else {
    test_data <- test_data %>% group_by(Artist, AlbumName) %>%
      mutate(PredictedPopularity = run_knn(test_data, Artist, AlbumName)) %>%
      mutate(AbsolutePercentageError =
               (abs(PredictedPopularity - AlbumPopularityRating)/
                  AlbumPopularityRating)*100)
  }
  return(sum(test_data$AbsolutePercentageError)/length(test_data))
}
```

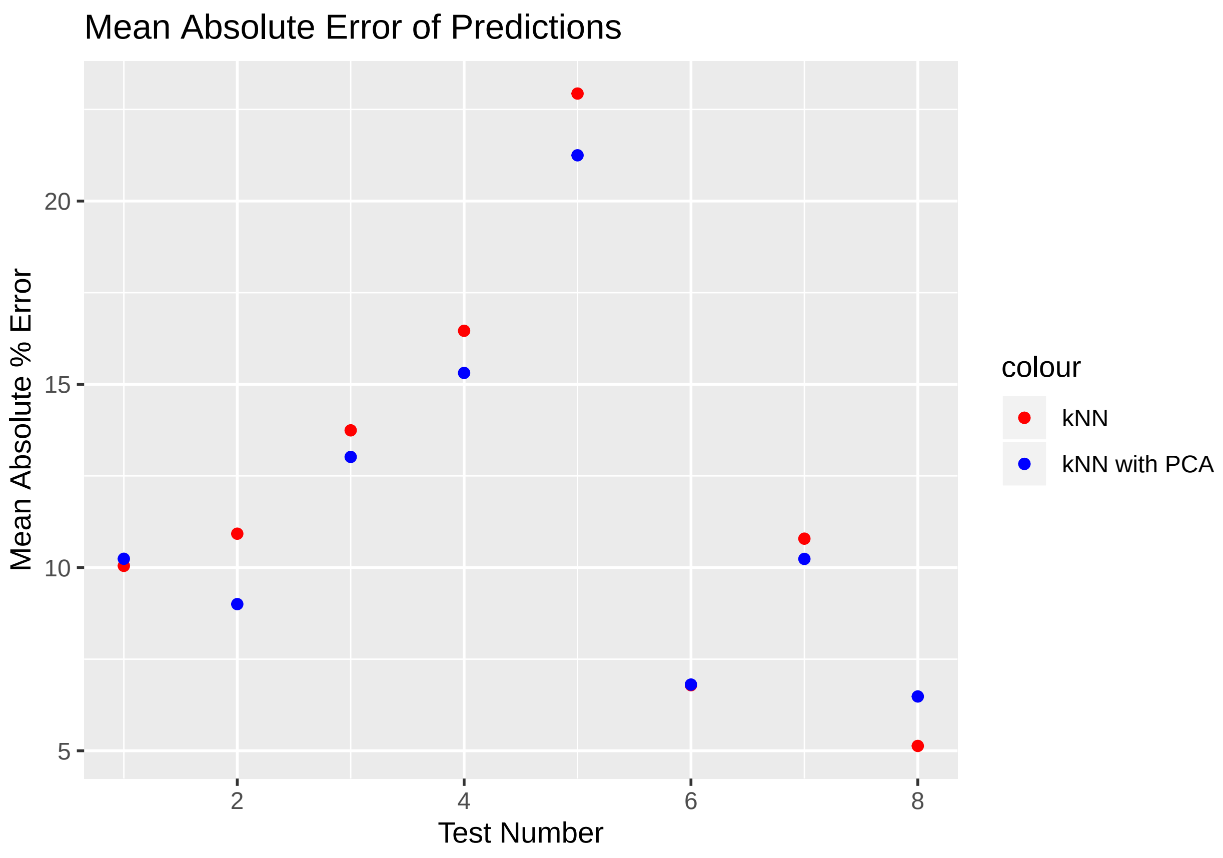
Test and Error Plot

Finding the predictions for all test_sets using the two models and storing their errors.

```r
albumdat <- albumdatgen()
errors.1 <- rep(-1,8)
errors.2 <- rep(-1,8)
for(i in 1:8){
  test_set <- paste("test_albums_",i,".txt", sep = "")
  errors.1[i] <- predict_popularity(test_set, pca=FALSE)
  errors.2[i] <- predict_popularity(test_set, pca=TRUE)
}
```

Plotting the errors for both models.

```r
A = data.frame(x = (errors.1), y=c(1:8))
B = data.frame(x = errors.2, y=c(1:8))
ggplot() +
  geom_point(aes(x = c(1:8),y = errors.1, color = "kNN"),data=A) +
  geom_point(aes(x = c(1:8),y = errors.2, color = "kNN with PCA"),data=B) +
    ggtitle("Mean Absolute Error of Predictions") +
    labs(x="Test Number", y="Mean Absolute % Error") +
    theme(legend.position="right") +
    scale_color_manual(values = c("red", "blue"))
```


Mean Absolute Error of Predictions
Mean Absolute % Error
Test Number
5
10
15
20
2
4
6
8
colour
kNN
kNN with PCA

Task 2

Libraries

The libraries used to complete this task were:

```r
library(readxl)
library(dplyr)
library(lubridate)
library(rio)
library(tidyr)
library(stringr)
library(stringi)
```

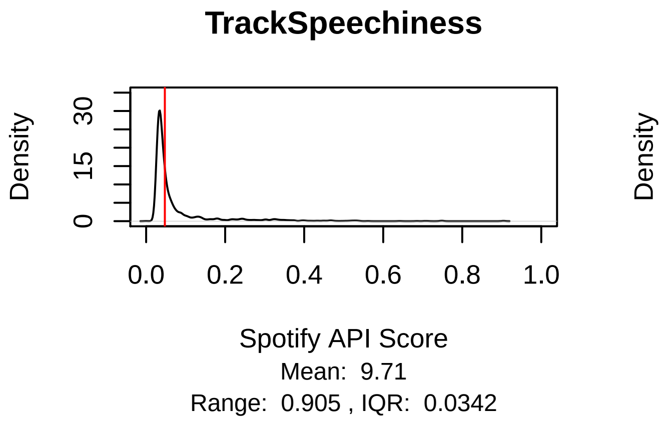
Data Manipulations

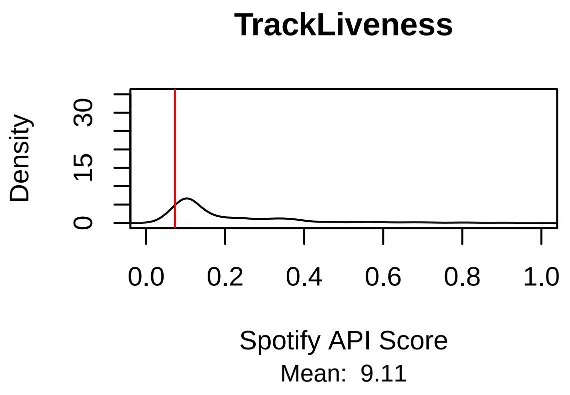
The following chunk contains all functions used for data manipulation/retreival.

```r
# Data Import and Date Parsing
Spotify_Data <- import("edited_spotify.xlsx", setclass = "tibble")
Spotify<- na.omit({
  Spotify_Data %>%
    group_by(Artist, AlbumName, AlbumReleaseDate, ArtistPopularity)
})
Spotify$AlbumReleaseDate = as.Date(parse_date_time(Spotify$AlbumReleaseDate, orders = c("y", "ym", "ymd

test <- read_excel("test_track1.xlsx")

df <- data.frame(Spotify)
cols <- c(1,3,6,13,17:27)
df <- df[,cols]
```

The *assignscore()* function is used to generate the 'scores'.

```r
assignscore <- function(varname, test, df){
  valuelist = df[varname]
  min1 = min(valuelist)
  max1 = max(valuelist)
  testinput = unlist(lapply(test[varname], "[[", 1))
  len = length(valuelist)
  for (i in (1:len)){
    valuelist[i] <- 10 - floor(abs(valuelist[i]-testinput)/((max1-min1)/10))
  }
  return(valuelist)
  #df[varname] <- valuelist
}
```





The following chunk contains the code for density plots of variables: