Music EDX Capstone Project Report

MM

27/02/2022

Contents

INTRODUCTION

This is a report for the capstone project required to obtain the Data Science Professional Certification issued by Harvard University, via EDX platform.

Measurement of a song's success is a relative matter. It could be assessed based on volume of sales, how popular it might be or from a purely artistic point of view regarding its aesthetics and characteristics.

For this project, I have defined as a measure of song success: whether a song was listed in the Billboard Hot 100 chart, and therefore classified as a "hit".

I obtained and combined relevant data to predict whether a song can become a "hit" based on its audio features.

This analysis does not account for other possible confounding factors, such as marketing, performer abilities and trajectory, social or geographical environment, among others factors that could have a direct influence on a song becoming a hit.

This project was of particular interest to me. Among my hobbies, I like to compose, record music and play various instruments. This analysis helped me understanding better the characteristics of successful songs.

Libraries

The following libraries are required for proper execution of the related R script file:

```
library(tidyverse)
library(stringr)
library(spotifyr)
library(purrr)
library(skimr)
library(caret)
library(DataExplorer)
library(corrplot)
library(hrbrthemes)
library(factoextra)
library(Boruta)
library(rpart)
library(rpart.plot)
library(packcircles)
library(viridis)
library(caretEnsemble)
```

Data Wrangling

I have obtained the following data sets:

- **Hot 100 Billboard songs** see link: This is a compilation of all songs that made the top 100's in the billboard charts between 1957 and 2017. I used the file called "Hot 100 Audio Features.csv" to gather songs that were defined as a "hit" for the purpose of this project.
- **Dataset of songs in Spotify** see link : I used the file called "genres_v2.csv" to obtain audio feature data for songs that were not included in the hot 100 billboard songs dataset, (hence a "non-hit" for the purpose of this project).

I performed data wrangling operations to combine these datasets as follows:

- removing rows in genres_v2 missing song name
- removing rows in Hot_100_Audio_Features missing genre
- renaming columns as appropriate
- defining a class field for non_hits in genres_v2 (set as 0, and labeled as "No")
- defining a class field for hits in hot_100 (set as 1, and labeled as "Yes")
- dropping non-useful columns
- defining the order of columns selected for the final dataset
- sub-setting songs from genres_v2 that were not in Hot_100_features (object name: genres_v2_no_hits)
- genres_v2 file did not include performer data. I obtained a sample of performers' names of non-hit songs for context, by connecting to Spotify via its Web API (sample size was 210 due to Spotify download limitations)
- trimming the datasets to keep unique records for each track
- excluding rows with no features from hot_100 data
- Combining the genres_v2_no_hits and hot_100_features sets into a single dataset (object name: "Dataset"), while ensuring that classes remained balanced (same amount of songs from each dataset)
- created a new field in Dataset for a cleansed genre variable
- checking completeness of the new Dataset

The following code was used to perform the above mentioned data loading and wrangling operations:

```
#####DATA LOADING####
#loading the data
Hot_100_Audio_Features <- read.csv("~/Hot 100 Audio Features.csv")
genres_v2 <- read.csv("~/genres_v2.csv")

##view the columns available
names(Hot_100_Audio_Features)
names(genres_v2)

##checking which columns are not in both datasets

setdiff(colnames(Hot_100_Audio_Features), colnames(genres_v2))

setdiff( colnames(genres_v2),colnames(Hot_100_Audio_Features))

##checking unique records per dataset

length(unique(Hot_100_Audio_Features$spotify_track_id))
length(unique(genres_v2$id))

##checking column classes
a<- lapply(Hot_100_Audio_Features, class)
b<-lapply(genres_v2, class)
str(Hot_100_Audio_Features)

#exploring NA's
Hot_100_Audio_Features %>% filter(is.na(SongID)== TRUE) %>% count()
Hot_100_Audio_Features %>% filter(is.na(Song)== TRUE) %>% count()
Hot_100_Audio_Features %>% filter(is.na(spotify_track_id)== TRUE) %>% count()
Hot_100_Audio_Features[(Hot_100_Audio_Features$Song==""),] %>% count()
Hot_100_Audio_Features[(Hot_100_Audio_Features$spotify_genre==""),]  %>% count()
Hot_100_Audio_Features[(is.na(Hot_100_Audio_Features$spotify_genre)== TRUE),]  %>% count()
Hot_100_Audio_Features[(Hot_100_Audio_Features$spotify_genre == "[]"),]  %>% count()

genres_v2 %>% filter(is.na(id ==TRUE)) %>% count()
genres_v2 %>% filter(is.na(song_name ==TRUE)) %>% count()
genres_v2[(genres_v2$song_name==""),]  %>% count()
genres_v2[(genres_v2$genre== "[]"),]  %>% count()

######DATA WRANGLING########
#removing rows in genres_v2 missing song name
genres_v2 <- genres_v2[!(genres_v2$song_name==""),]

#removing rows in Hot_100_Audio_Features  missing genre
Hot_100_Audio_Features <- Hot_100_Audio_Features[!(Hot_100_Audio_Features$spotify_genre==""),]
Hot_100_Audio_Features <- Hot_100_Audio_Features[(Hot_100_Audio_Features$spotify_genre != "[]"),]

#renaming columns
genres_v2 <- genres_v2 %>%   rename(., track_id = id )
Hot_100_Audio_Features <- Hot_100_Audio_Features %>%
  rename(., track_id = spotify_track_id, song_name = Song, genre = spotify_genre,
         duration_ms = spotify_track_duration_ms)
```

```r
#defining class field
genres_v2$Is_hit <- 0 #means No hit
Hot_100_Audio_Features$Is_hit <- 1  # means Yes, it's a hit

#columns to drop
#Hot_100_Audio_Features:
exclude_Hot <- c("spotify_track_album", "SongID", "spotify_track_preview_url",
                 "spotify_track_explicit","spotify_track_popularity")

#genres_v2
exclude_genr <- c("type","uri","track_href", "analysis_url", "Unnamed..0" , "title")

#defining columns wanted in specific order

valid_fields <- c("track_id", "song_name", "Is_hit", "genre", "key", "mode", "tempo",
"danceability", "energy", "loudness", "speechiness", "acousticness",
"instrumentalness", "liveness", "valence","duration_ms",  "time_signature", "Performer")

#identifying the songs from genres_v2 that are not in Hot_100 (i.e. true no hits)

genres_v2_no_hits <- subset(genres_v2 ,!(track_id%in%Hot_100_Audio_Features$track_id))

####OBTAINING SAMPLE OF PERFORMERS NAMES OF NON-HIT SONGS FOR CONTEXT###

#creating authentication
Sys.setenv(SPOTIFY_CLIENT_ID = Client_Id)  # client id not shared here for security
Sys.setenv(SPOTIFY_CLIENT_SECRET = Client_secret) #client secret not shared here for security

#defining a function to obtain artists names
spotify_filter_artist_id_func <- function(x) {
  c<- get_track(x, market = NULL, authorization = get_spotify_access_token())
  d<- c$artists$id
  e<- get_artists(d, authorization = get_spotify_access_token())
  f<- e$name
  g<-data.frame(song_id= x, artist_id = d, artist_name = f)
  return(g)
}

#getting samples (limited to 50 songs per run)

spotify_filter_artist_id <- lapply( sample(genres_v2_no_hits$track_id,50), spotify_filter_artist_id_func

spotify_filter_artist_id <-  bind_rows(spotify_filter_artist_id, .id = "column_label")

spotify_filter_artist_id2 <- lapply( sample(genres_v2_no_hits$track_id,50), spotify_filter_artist_id_fu

spotify_filter_artist_id2 <-  bind_rows(spotify_filter_artist_id, .id = "column_label")


spotify_filter_artist_id3 <- lapply( sample(genres_v2_no_hits$track_id,50), spotify_filter_artist_id_fu

spotify_filter_artist_id3 <-  bind_rows(spotify_filter_artist_id, .id = "column_label")
```

```r
#merging samples
no_hits_performers_sample <- rbind(spotify_filter_artist_id, spotify_filter_artist_id2, spotify_filter_a

#####################################################################

#creating "Performers" column in genres_v2_no_hits

genres_v2_no_hits <- merge(genres_v2_no_hits, no_hits_performers_sample[ , c("song_id", "artist_name")]

genres_v2_no_hits <- genres_v2_no_hits %>% rename(., Performer = artist_name)

#keeping only unique tracks

genres_v2_no_hits<- genres_v2_no_hits %>% distinct(track_id, .keep_all = TRUE)

Hot_100_Audio_Features_uniques <- Hot_100_Audio_Features %>% distinct(track_id, .keep_all = TRUE)

#checking datasets completeness

genres_v2_no_hits[(genres_v2_no_hits$track_id==""),] %>% count()
genres_v2_no_hits %>% filter(is.na(genre)== TRUE) %>% count()
genres_v2_no_hits  %>% filter(genre == "") %>% count()

Hot_100_Audio_Features_uniques[(Hot_100_Audio_Features_uniques$track_id==""),] %>% count()

Hot_100_Audio_Features_uniques %>% filter(is.na(track_id)== TRUE) %>% count()

Hot_100_Audio_Features_uniques %>% filter(is.na(danceability)== TRUE) %>% count()

Hot_100_Audio_Features_uniques %>% filter(is.na(genre)== TRUE) %>% count()

Hot_100_Audio_Features_uniques  %>% filter(genre == "") %>% count()

Hot_100_Audio_Features_uniques  %>% filter(genre == "[]") %>% count()

#excluding rows with no features from hot_100 data

Hot_100_Audio_Features_uniques<- Hot_100_Audio_Features_uniques %>% filter(!is.na(danceability)== TRUE)

#COMBINING DATASETS and keeping balanced classes (at 16656 each)

#combining datasets
Dataset <- rbind(genres_v2_no_hits %>% select(all_of(valid_fields)),Hot_100_Audio_Features_uniques[1:16

#cleansing genre variable in Dataset to eliminate special characters and also only take the first genre

Dataset$genre_clean  <- Dataset$genre  %>%   ifelse(str_detect(., ",", negate = FALSE),  str_extract(.,
  str_replace_all(., "[:punct:]", " ") %>% trimws(.)


#dataset profiling
summary(Dataset)
```

```
Dataset_skim_summary <- Dataset  %>% skim()

#checking completeness of final dataset
Dataset[(Dataset$key==""),] %>% count()

Dataset %>% filter(is.na(track_id)== TRUE) %>% count()

Dataset %>% filter(genre == "[]") %>% count()

Dataset %>% filter(genre == "") %>% count()

#ensuring classes remain balanced
Dataset %>% group_by(Is_hit) %>% summarise(n = n())

#counting songs per genre
dataset_genre <- Dataset %>% select(genre,genre_clean) %>%   group_by(genre_clean, genre) %>% summarise

#deleting tables not longer needed to liberate memory:
rm(list=c("genres_v2", "genres_v2_no_hits", "Hot_100_Audio_Features","Hot_100_Audio_Features_uniques",
          "spotify_filter_artist_id", "spotify_filter_artist_id2", "spotify_filter_artist_id3",
          "a", "b", "valid_fields","exclude_genr", "exclude_Hot", "Dataset_skim_summary"))

######DATA SPLITTING FOR MODELLING#####

# Split-out validation dataset

set.seed(75, sample.kind="Rounding")

# create a list of 80% of the rows in the cleansed Dataset  for training
validationIndex <- createDataPartition(y = Dataset$Is_hit , times = 1, p=0.80, list=FALSE)
# select 20% of the data for validation
validation <- Dataset[-validationIndex,]
# use the remaining 80% of data to training the models
training <- Dataset[validationIndex,]
```

This new "Dataset" was then used to create a training partition (80%) and validation partition (20%).

I made a choice to maintain classes balanced to avoid any undue influence from this element. However, this might distant from the actual population of songs across the world (where there is a lower proportion of hit songs than non-hit songs). This has been considered in the conclusions of this report.

Further review and analysis was undertaken as explained in the following sections and I documented the project through the following files for submission to EDX:

- An Rmd file report detailing the methodology applied,
- A PDF report (issued from the Rmd file above mentioned), and
- An R script file that generates the analysis and predictions

About the source datasets

Hot 100 Billboard songs

The Billboard Hot 100 is the music industry standard record chart in the United States for songs, published weekly by Billboard magazine. Chart rankings are based on sales, radio play, and online streaming in the

United States.

Every week, Billboard releases "The Hot 100" chart of songs that were trending on sales and airplay for that week. This dataset is a collection of all "The Hot 100" charts released since its inception in 1958. The dataset includes audio features for the songs taken from the Spotify database. The dataset is hosted in data.world website.

Dataset of songs in Spotify

This is a data set containing several songs by genres and provides the related Spotify audio features. It is hosted in Kaggel. It was created in December 2020.

About the combined dataset

The following fields were included in the final Dataset before partitioning the data into training and validation sets:

```
sapply(Dataset, class) %>% as.data.frame(.)
```

```
##                            .
## track_id           character
## song_name          character
## Is_hit               numeric
## genre              character
## key                  integer
## mode                 integer
## tempo                numeric
## danceability         numeric
## energy               numeric
## loudness             numeric
## speechiness          numeric
## acousticness         numeric
## instrumentalness     numeric
## liveness             numeric
## valence              numeric
## duration_ms          integer
## time_signature       integer
## Performer          character
## genre_clean        character
```

- **track_id**: the Spotify ID for the track
- **song_name**: title of the song
- **Is_hit**: binary variable stating 0 if the song was not within the top 100 billboard chart or 1 if it was included (labels "No", "Yes")
- **genre**: identify the song style or musical tendency
- **key**: The key the track is in. Integers map to pitches using standard pitch class notation. E.g. 0 = C, 1 = C /D , 2 = D, and so on. If no key was detected, the value is -1.
- **mode**: Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.
- **tempo**: The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.

- **danceability**: Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.
- **energy**: Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.
- **loudness**: The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typically range between -60 and 0 db.
- **speechiness**: Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.
- **acousticness**: A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
- **instrumentalness**: Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.
- **liveness**: Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.
- **valence**: a measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).
- **duration_ms**: The duration of the track in milliseconds.
- **time_signature**: An estimated time signature. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure). The time signature have the following values: 1, 3, 4, 5 representing the beats on a bar for the time signature (1/4, 3/4, 4/4, 5/4). Items with value zero represents that data was missing. For example category "4/4" should be read as "4 by 4".
- **Performer**: name of the artist that perform the song. This data is missing for the majority of non-hit songs due to data limitations.
- **genre_clean**: this is the genre field without special characters and only the first genre for those songs that had more than one.

The consolidated Dataset contained 33,312 records and 19 variables. The following is a detail of top 10 performers by hit and non-hit songs for context:

```
#checking performers' names for hit & non-hit songs in consolidated Dataset (before partition)
#hit songs
Dataset %>% filter(Is_hit== "1") %>% group_by(Performer,Is_hit) %>%
  summarise(number_songs = n()) %>% arrange(desc(number_songs)) %>% head(10)
```

```
## `summarise()` has grouped output by 'Performer'. You can override using the `.groups` argument.
```

```
## # A tibble: 10 x 3
## # Groups:   Performer [10]
##    Performer          Is_hit number_songs
##    <chr>               <dbl>        <int>
```

```
##  1 Glee Cast               1          131
##  2 Taylor Swift            1           83
##  3 Drake                   1           70
##  4 The Beatles             1           46
##  5 Madonna                 1           43
##  6 The Rolling Stones      1           43
##  7 Elton John              1           41
##  8 Stevie Wonder           1           40
##  9 Justin Bieber           1           39
## 10 Aretha Franklin         1           38
```

```
#non-hit songs
Dataset %>% filter(Is_hit== "0" & !is.na(Performer)) %>% group_by(Performer,Is_hit) %>%
  summarise(number_songs = n()) %>% arrange(desc(number_songs)) %>% head(10)
```

```
## `summarise()` has grouped output by 'Performer'. You can override using the `.groups` argument.
```

```
## # A tibble: 10 x 3
## # Groups:   Performer [10]
##    Performer       Is_hit number_songs
##    <chr>            <dbl>        <int>
##  1 Aroc!                0            1
##  2 CHVRN                0            1
##  3 DaBaby               0            1
##  4 Elestu               0            1
##  5 Eric Bellinger       0            1
##  6 Freddie Dredd        0            1
##  7 Fukkit               0            1
##  8 Ghostface Playa      0            1
##  9 gizmo                0            1
## 10 Great Dane           0            1
```

```
#checking Michael Jackson songs
Dataset %>% filter(Performer == "Michael Jackson") %>% group_by(Performer,Is_hit) %>%
  summarise(number_songs = n())
```

```
## `summarise()` has grouped output by 'Performer'. You can override using the `.groups` argument.
```

```
## # A tibble: 1 x 3
## # Groups:   Performer [1]
##   Performer       Is_hit number_songs
##   <chr>            <dbl>        <int>
## 1 Michael Jackson      1           27
```

To illustrate the nature of the dataset, I also ran a check for Michael Jackson as he was one of the most popular performers of all time. We can see that for hit songs most performer names are familiar, while this is not the case for non-hit songs.

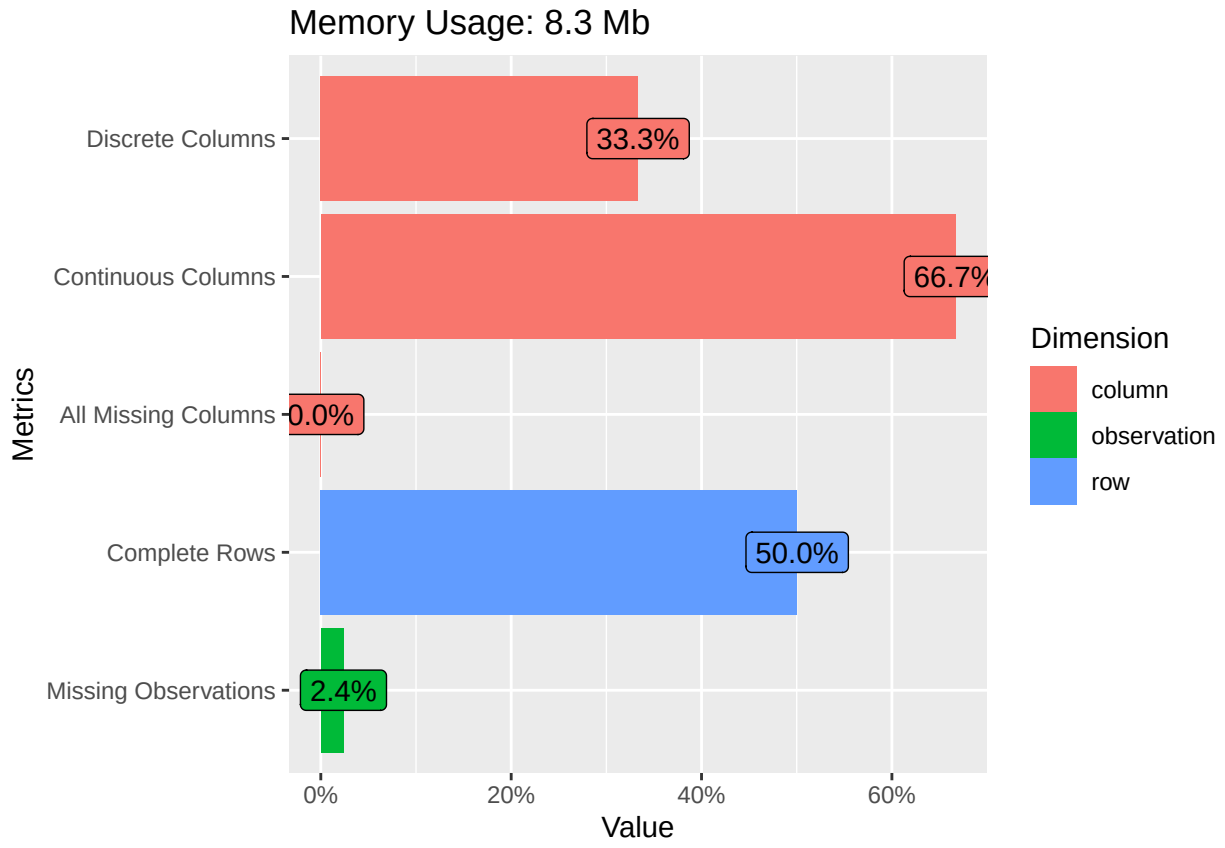
About the training dataset

The training dataset contained 26,650 observations and originally 19 columns, later two more columns were created. The following details were noted:

- There were 14 continue and 7 categorical variables in the training dataset (This is the same for the validation set).
- Only 50% of the rows were fully completed, which indicate the presence of NA values in the dataset. Most NA's related to field: "Performer".

```
# dimensions of dataset
dim(training)

#exploring training data
training %>% introduce()
```

```
#exploring training data
training %>% plot_intro()
```




```
#checking for missing values

training %>% profile_missing()  # data frame of the missing values % per variable
```

```
##               feature num_missing pct_missing
## 1            track_id           0   0.0000000
## 2           song_name           0   0.0000000
## 3              Is_hit           0   0.0000000
## 4               genre           0   0.0000000
## 5                 key           0   0.0000000
```

```
## 6                 mode           0   0.0000000
## 7                tempo           0   0.0000000
## 8         danceability           0   0.0000000
## 9               energy           0   0.0000000
## 10            loudness           0   0.0000000
## 11         speechiness           0   0.0000000
## 12        acousticness           0   0.0000000
## 13    instrumentalness           0   0.0000000
## 14            liveness           0   0.0000000
## 15             valence           0   0.0000000
## 16         duration_ms           0   0.0000000
## 17      time_signature           0   0.0000000
## 18           Performer       13312   0.4995122
## 19         genre_clean           0   0.0000000
## 20 genre_clean_factors           0   0.0000000
## 21           new_genre           0   0.0000000
```

The following indicates the variables classes and provides an overview of the training dataset first 3 rows:

```
## variables types
sapply(training, class) %>% as.data.frame(.)
```

```
##                            .
## track_id            character
## song_name           character
## Is_hit                 factor
## genre               character
## key                   numeric
## mode                  numeric
## tempo                 numeric
## danceability          numeric
## energy                numeric
## loudness              numeric
## speechiness           numeric
## acousticness          numeric
## instrumentalness      numeric
## liveness              numeric
## valence               numeric
## duration_ms           numeric
## time_signature        integer
## Performer           character
## genre_clean         character
## genre_clean_factors    factor
## new_genre             numeric
```

```
## reviewing the first 5 rows
head(training, n=3)
```

```
##                 track_id                                song_name Is_hit     genre
## 1 003VDDA7J3Xb2ZFlNx7nIZ                                  YELL OH     No       Rap
## 2 004Vn8Q2q19hVTAvcNrocD                                  Eclipse     No Dark Trap
## 3 006hfStQk9L7MVT9RcDeva A Part Of Me (feat. Laura Whiteside)     No       Emo
##   key mode   tempo danceability energy loudness speechiness acousticness
```

```
## 1   6    0  74.496        0.842  0.578   -6.050      0.1380      0.00419
## 2   7    1 128.007        0.516  0.853   -8.572      0.0476      0.00131
## 3   4    1 200.036        0.372  0.875   -0.946      0.0435      0.37600
##   instrumentalness liveness valence duration_ms time_signature Performer
## 1         0.00e+00    0.228  0.1900      236779              4      <NA>
## 2         8.79e-01    0.669  0.0512      195000              4      <NA>
## 3         5.16e-05    0.103  0.3130      189415              4      <NA>
##   genre_clean genre_clean_factors new_genre
## 1         Rap                Rap 0.8784861
## 2   Dark Trap          Dark Trap 0.4083665
## 3         Emo                Emo 0.5298805
```

I had a first glance of the distributions for most variables. Inspected the variable "key" and noted that the dataset includes songs in all keys:

```
# summary
summary(training)

#checking key field distribution
unique(training$key)
```

Finally, confirmed that the classes remained balanced after partition:

```
#ensuring classes remained balanced after partition

# class distribution
cbind(freq=table(training$Is_hit), percentage=prop.table(table(training$Is_hit))*100)
```

```
##      freq percentage
## No  13325         50
## Yes 13325         50
```

About the validation dataset

The validation dataset contained 6,662 observations and originally 19 columns. Later, two more columns were created.

We can see that the same columns are included in the validation dataset. The whole range of song keys are included and classes remained balanced after partition.

```
# summary
str(validation)
```

```
## 'data.frame':    6662 obs. of  21 variables:
##  $ track_id           : chr  "00oy0A1Y3IS8SgP8W8xS30" "00rOE00OKN8IlG1MQvgHma" "00TxSiyaZAqr0xUpGEe5
##  $ song_name          : chr  "Feeling Whitney" "We're Not the Same" "Tellin Ya (feat. Lil PJ)" "Hity
##  $ Is_hit             : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ genre              : chr  "Hiphop" "Trap Metal" "Underground Rap" "Hiphop" ...
##  $ key                : num  0 8 1 11 7 4 8 1 10 8 ...
##  $ mode               : num  1 0 0 0 1 1 1 0 0 0 ...
##  $ tempo              : num  162 181 154 181 149 ...
##  $ danceability       : num  0.433 0.291 0.765 0.824 0.867 0.773 0.687 0.559 0.798 0.933 ...
```

```
##  $ energy             : num  0.223 0.954 0.422 0.817 0.403 0.489 0.653 0.963 0.553 0.878 ...
##  $ loudness           : num  -12.62 -4.19 -8.76 -5.93 -13.74 ...
##  $ speechiness        : num  0.0375 0.465 0.485 0.36 0.484 0.141 0.0696 0.0694 0.348 0.0791 ...
##  $ acousticness       : num  0.81 0.000131 0.104 0.169 0.099 0.117 0.0185 0.0372 0.102 0.0884 ...
##  $ instrumentalness   : num  3.67e-06 1.90e-02 0.00 0.00 9.30e-06 0.00 5.34e-01 4.73e-
01 0.00 0.00 ...
##  $ liveness           : num  0.229 0.501 0.107 0.354 0.229 0.163 0.326 0.238 0.113 0.0833 ...
##  $ valence            : num  0.227 0.324 0.267 0.685 0.59 0.596 0.529 0.163 0.158 0.409 ...
##  $ duration_ms        : num  257267 170342 174545 193000 167706 ...
##  $ time_signature     : int  4 4 4 4 4 4 4 4 4 4 ...
##  $ Performer          : chr  NA NA NA NA ...
##  $ genre_clean        : chr  "Hiphop" "Trap Metal" "Underground Rap" "Hiphop" ...
##  $ genre_clean_factors: Factor w/ 312 levels "a cappella","acoustic pop",..: 199 305 310 199 310 199
##  $ new_genre          : num  0.637 0.977 0.994 0.637 0.994 ...
```

```
unique(validation$key)
```

```
##  [1]  0  8  1 11  7  4 10  9  6  2  5  3
```

```
#ensuring classes remained balanced after partition

# class distribution
cbind(freq=table(validation$Is_hit), percentage=prop.table(table(validation$Is_hit))*100)
```

```
##     freq percentage
## No  3331         50
## Yes 3331         50
```

Cleanising of certain variables

To aid with data visualisations and interpretation of the model, I relabeled the class variable "Is_hit" and converted the genres variable to factors using the following code:

```
#modifying labels for ease of understanding

#on training dataset
levels(training$Is_hit) <- c("No","Yes")
head(training$Is_hit)

#on validation dataset
levels(validation$Is_hit) <- c("No","Yes")
head(validation$Is_hit)

#converting genres clean column to factors

training$genre_clean_factors <- as.factor(training$genre_clean)

#replicating it in validation dataset

validation$genre_clean_factors <- as.factor(validation$genre_clean)
```

METHODOLOGY:

Exploratory data analysis

Initial assumptions and expectations

Before looking in detail at the data, I started my analysis by theorising the following ideas based on my musical knowledge and intuition.

I have separated these in assumptions subsequently supported and non-supported by data.

Initial assumptions subsequently supported by data:

- **genre**: should have an impact as most people have preferences for particular genres over others
- **key**: might not have an impact, but certain keys are more common within certain genres than other, therefore this could be a minor contributing predictor
- **mode**: songs in a major scale tend to be more vibrant and happy. I believe most people would prefer songs in major scale as oppose to minor scales, but this will depend greatly of the population
- **danceability**: danceable songs are not necessary more popular among the general population, but can be a factor
- **speechiness**: Most songs do not have speeches within the track (although some genres would have more). Songs with no speeches would have values between 0.33 & 0.66. Some popular songs will have some elements of speechiness but I believe are not the majority
- **acousticness**: it is possible that acoustic music is less popular. It probably has some relation with genre as well
- **instrumentalness**: it is probable that songs with lyrics are more popular than instrumental songs. Therefore we should see more popular songs with low instrumentalness
- **liveness**: I believe most popular songs are studio recording rather than live versions
- **valence**: I believe that people would have a preference for songs that have a sound resembling happiness and positivism

Initial assumptions subsequently non-supported by data:

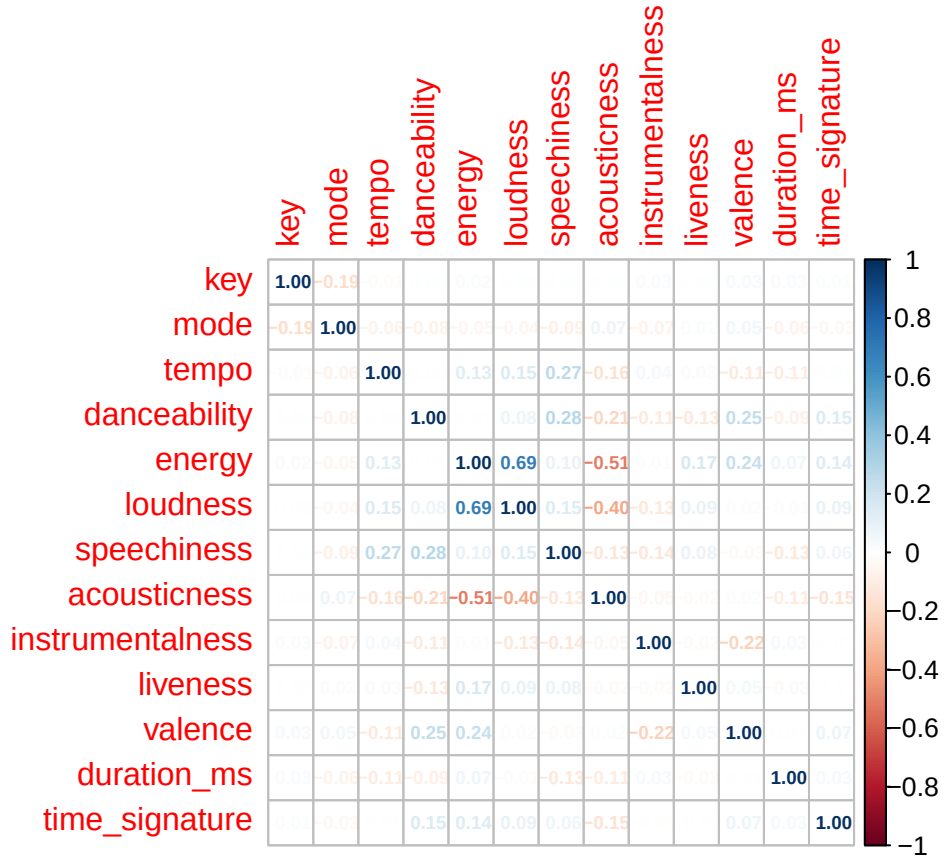
- **energy**: It is possible that people would prefer more energetic songs than slow songs. Although some ballads are quite popular.
- **loudness**: this item should be associated with energy and may be also higher in songs that include more instruments or are recorded at higher volumes. My guess is that people would prefer songs that have a medium to high level of loudness. This probably needs to be considered based on population preference (partially supported by data)
- **duration_ms**: I believe songs that are too long or short will not be very popular (partially supported by data)
- **time signature**: most conventional time signature is 4/4. Therefore, most popular songs should be in this time signature (partially supported by data)
- **Other factors**: many other factors outside audio features have an impact, such as marketing, artist, etc. (No data available).

Data Visualisations

I used the above to guide my exploratory data analysis and formulate research questions to identify features that could be good predictors:

Correlations

```
# Data visualizations

# Creating index of predictors
features_index <- c(5:17,20)
features_index_num <- c(5:17)

#checking correlations
corr_training <-cor(training[features_index_num])

corrplot(corr_training, method="number",type= "full",insig = "blank", number.cex = 0.6)
```



Correlations insights:

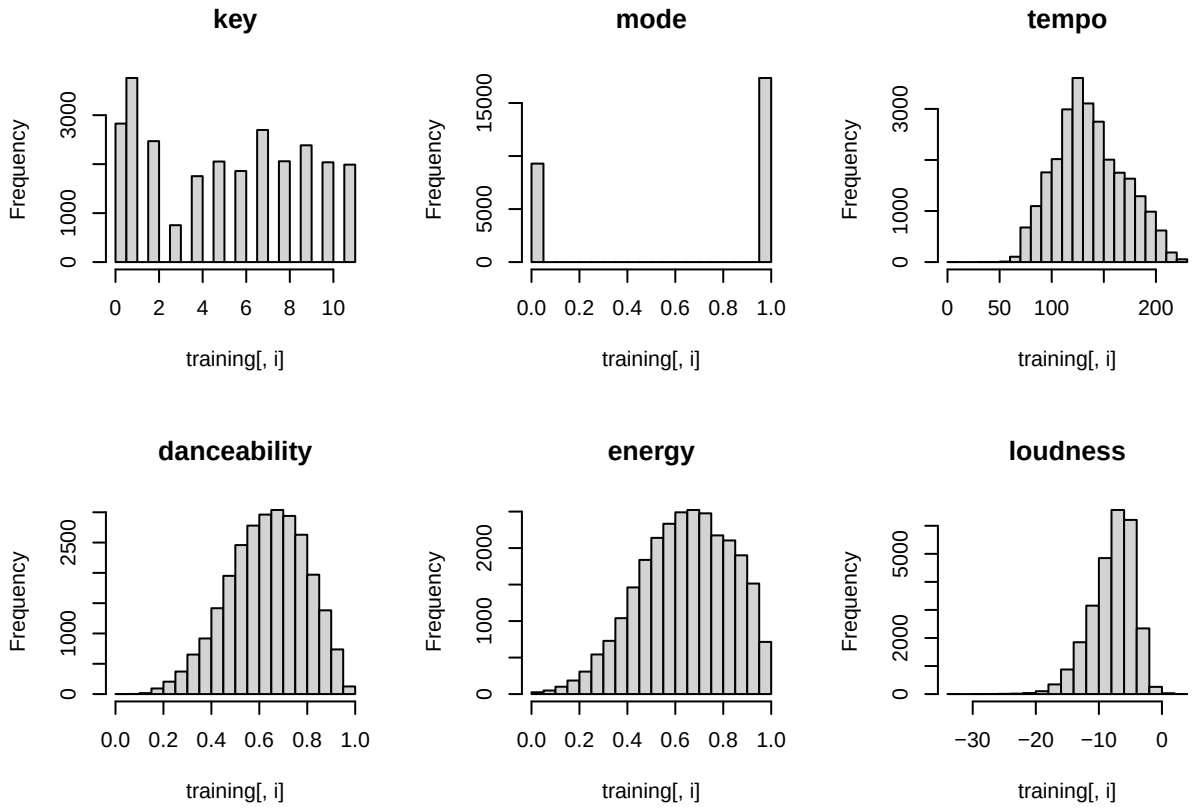
- There is a positive correlation between energy & loudness (0.69). Therefore, energetic songs have a tendency to be louder. We could probably exclude one of these features.
- There is a negative correlation between energy/loudness and acousticness. Songs that are more energetic might have less acoustic instruments (probably the instrumentation includes electric/electronic music as well as computer generated effects)

- There are some other minor positive correlations, such as: ** valence slightly correlates to danceability, ** speechiness slightly correlates to danceability and tempo
- There are some minor negative correlations: ** danceability and acousticness (i.e there is a small tendency to use electric instruments in songs that are made for dancing), ** danceability and instrumentalness, ** valence and instrumentalness

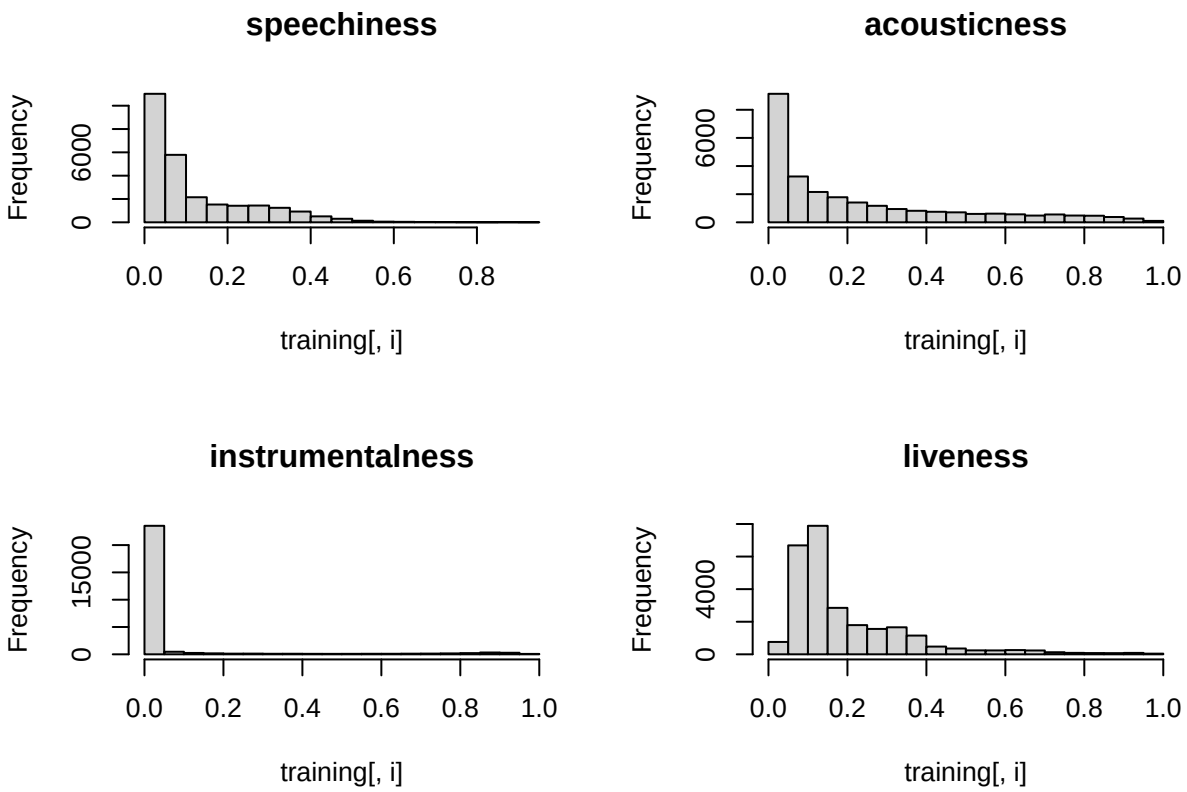
Distributions: histograms for each potential predictor

The following code allows to have a quick glance at the distribution of the variables:

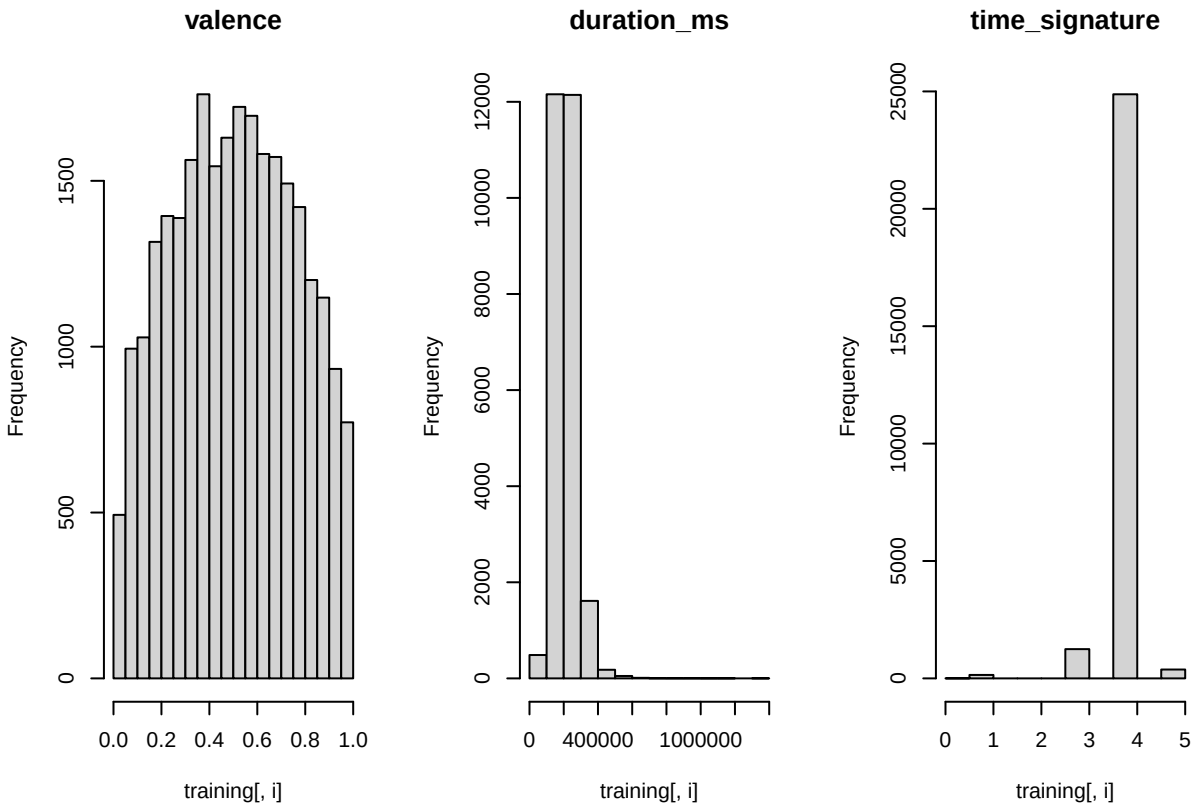
```
# histograms each numeric predictor (separated in groups to ease visualization)
par(mfrow=c(2,3))
for(i in 5:10) {
  hist(training[,i], main=names(training)[i])
}
```




```
par(mfrow=c(2,2))
for(i in 11:14) {
  hist(training[,i], main=names(training)[i])
}
```



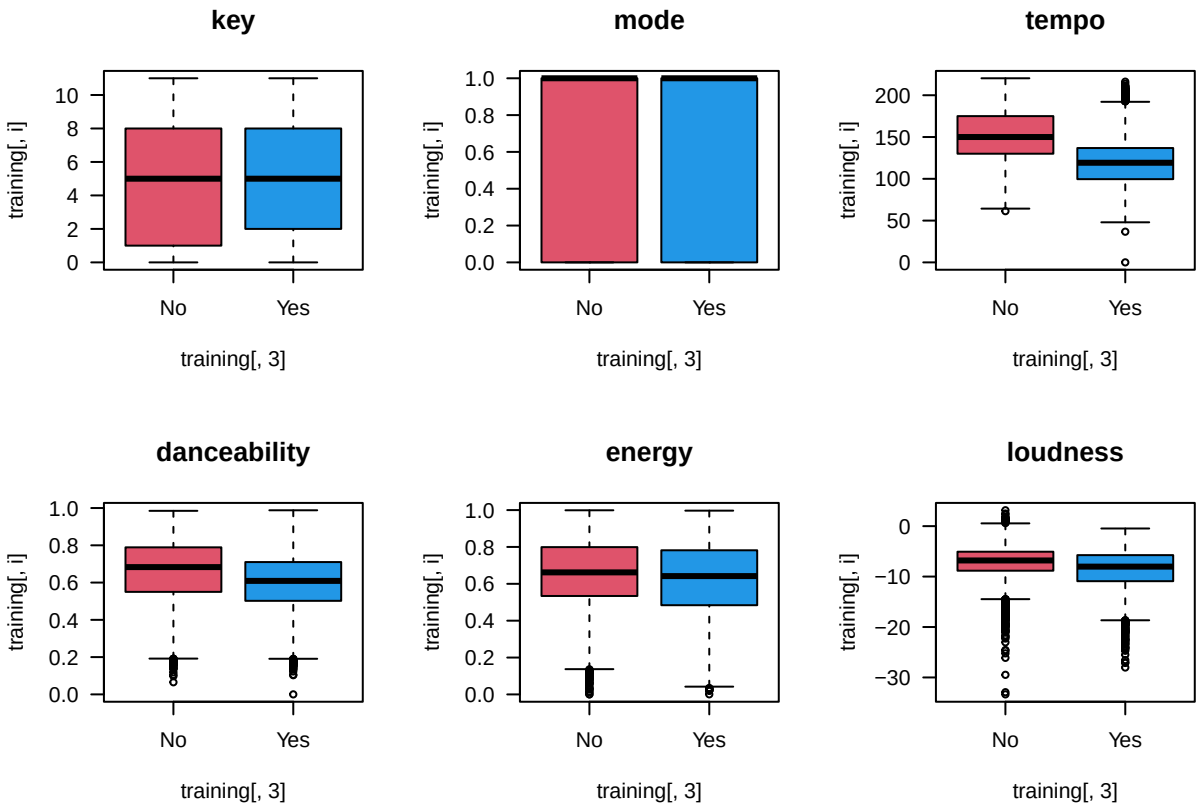

```
par(mfrow=c(1,3))
for(i in 15:17) {
  hist(training[,i], main=names(training)[i])
}
```



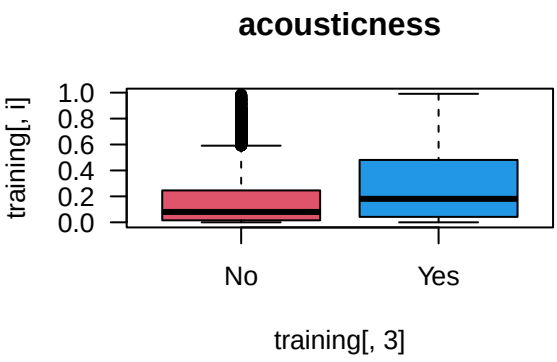
Distributions: box plots for each potential predictor

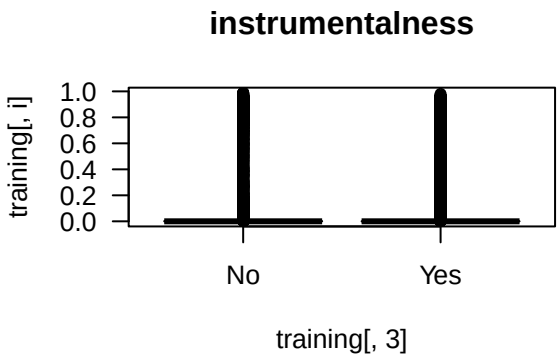
I explored the distributions per classes for all potential predictors by using box plots:

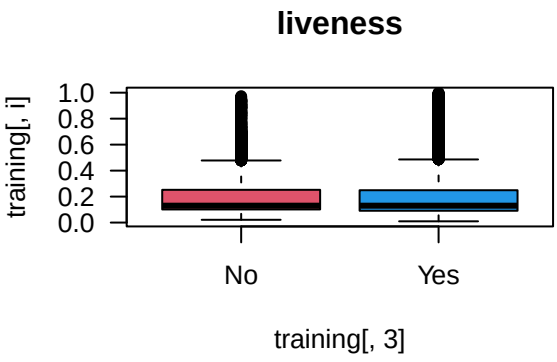
```
par(mfrow=c(2,3))
for(i in 5:10) {
  boxplot(training[,i] ~ training[,3] , main=names(training)[i], las =1, col= c(2,4))

}
```



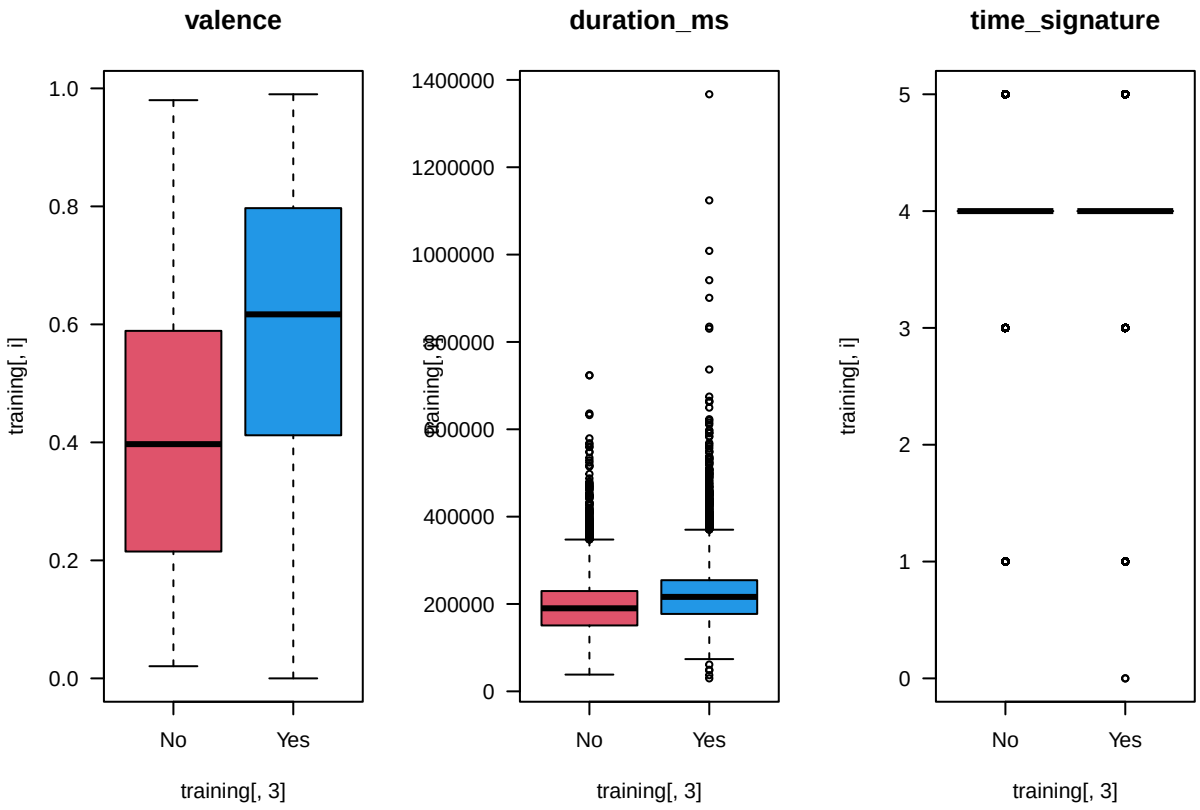

```
par(mfrow=c(2,2))
for(i in 11:14) {
  boxplot(training[,i] ~ training[,3] , main=names(training)[i], las =1, col= c(2,4))

}
```



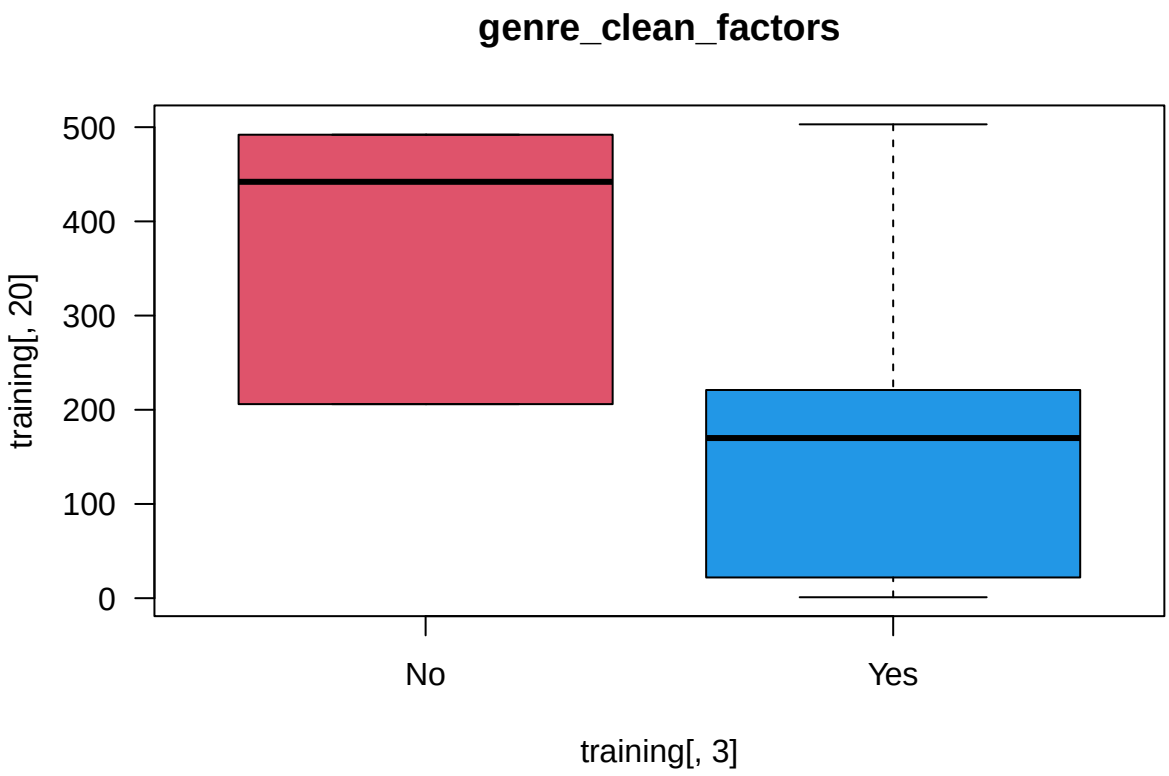





```
par(mfrow=c(1,3))
for(i in 15:17) {
  boxplot(training[,i] ~ training[,3] , main=names(training)[i], las =1, col= c(2,4))

  }
```




```
par(mfrow=c(1,1))
boxplot(training[,20] ~ training[,3] , main=names(training)[20], las =1, col= c(2,4))
```



Distributions insights:

At first glance we can see a variety of distributions, with some variables presenting a normal distribution (tempo, danceability, energy, loudness and valence), while others are right skewed (speechiness, acousticness, instrumentalness and liveness). There are also a number of categorical variables.

When splitting the distribution by class, we can see that certain variables such as genres, valence and tempo have a clear distinction in the distribution between classes and might be good candidates for predictors.

Research questions:

To deepen my understanding of the data I conducted further analysis to address research questions derived from the insights previously obtained.

I have divided this analysis in two groups:

- features describing songs types
- features describing elements of songs speed

Analysis part I: features describing songs types

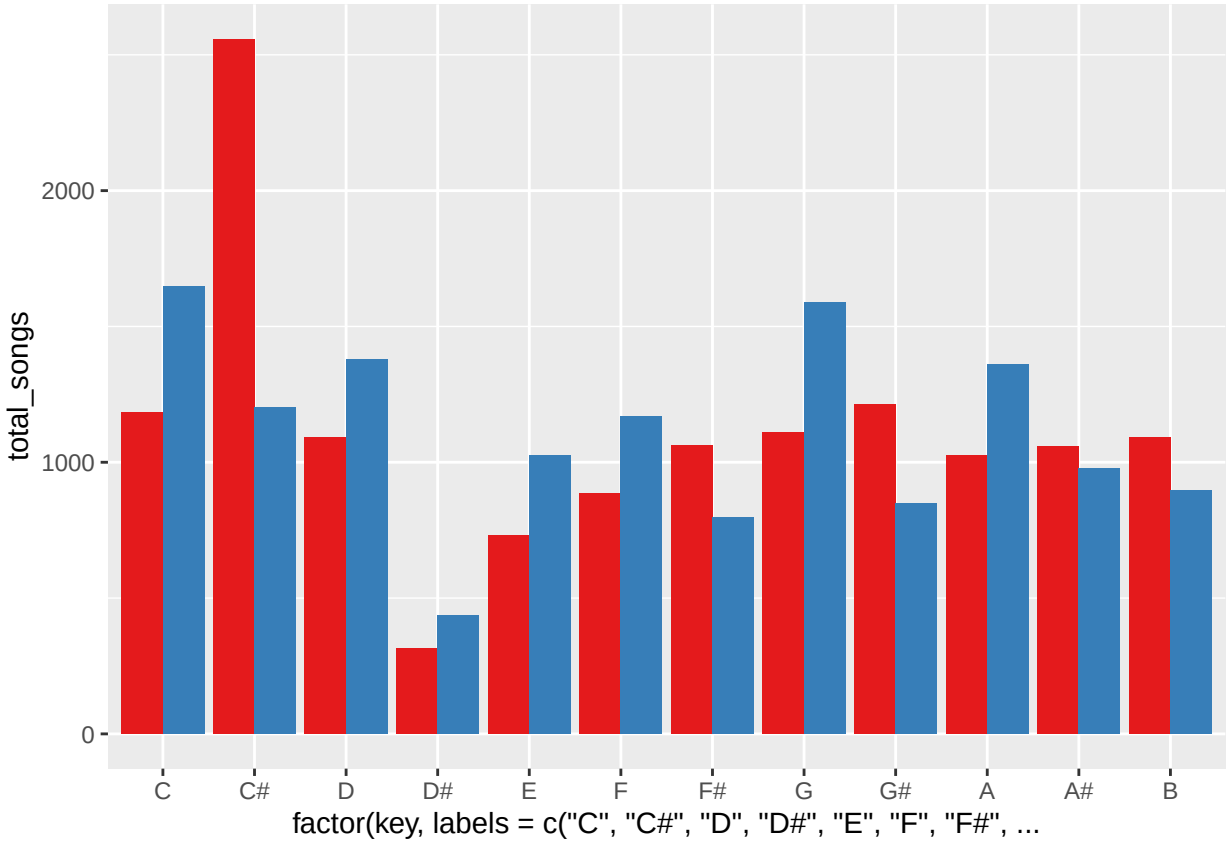
What are the most common keys by class?

To answer this question i used the following code to determine the number of songs by key and by class:

```
by_key_count <-training %>% select(key, Is_hit) %>% group_by( key, Is_hit) %>% summarise(total_songs = n
```

and plotted the result as follows:

```
by_key_count %>% ggplot(., aes(x=factor(key, labels =c("C","C#","D","D#","E","F","F#","G","G#","A","A#
                               y = total_songs, fill = Is_hit )) +
   geom_bar(stat = "Identity" , position = "dodge") +
   scale_fill_brewer(palette = "Set1")  +
   theme(legend.position="none")
```



insights:

- There are hit and non-hit songs across all tonalities (keys).
- Key analysis indicates that most common hit songs are in C (0) , G (7), D(2) and A(9) in that order. C# tonality is the key with most non-hit songs. The distribution of classes for each key might serve to calculate the probability of a song being a hit. Therefore, it might be a good candidate for predictor in logistic regression models.
- We can see that songs composed in D# scales are less frequent, and there is a tendency in this tonality to favor hit songs.

what are the most common genres by class?

The following code helped me answering the question:

```
#counting the songs by genre and class
by_genre_count <-training %>% select(genre_clean_factors, Is_hit) %>% group_by( genre_clean_factors, Is

summary(by_genre_count$total_songs)
```

```
#I also considered the proportion of of hit non-hit by genre and the distribution
prop_bygenre <- prop.table(table(training[,c("genre_clean_factors", "Is_hit")]), margin = 2 ) *100

# distribution of  songs per genre
by_genre_count %>% group_by(total_songs) %>%
  summarise(freq= n())%>% ggplot(., aes(x=as.factor(total_songs), y = freq )) +
  geom_bar(stat = "Identity" ) +
  theme(legend.position="none") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))

#top ten genres for hit songs (by count of songs)
by_genre_count %>% filter(Is_hit == "1") %>%
    arrange(desc(total_songs)) %>%
    head(10)

#top ten genres for non-hit songs (by count of songs)
by_genre_count %>% filter(Is_hit == "0") %>%
    arrange(desc(total_songs)) %>%
    head(10)
```

Insights:

- Top ten genres for hit songs (by count of songs) are: 1 adult standards 2 album rock 3 dance pop 4 contemporary country 5 classic soul 6 brill building pop 7 alt hip hop 8 disco 9 alternative metal 10 pop
- There are only eight genres in the data for non-hit songs (by count of songs):

  1 Underground Rap
  2 Dark Trap 3 Hip hop
  4 RnB
  5 Emo
  6 Trap Metal
  7 Rap
  8 Pop

Are genres, key and mode potential good predictors?

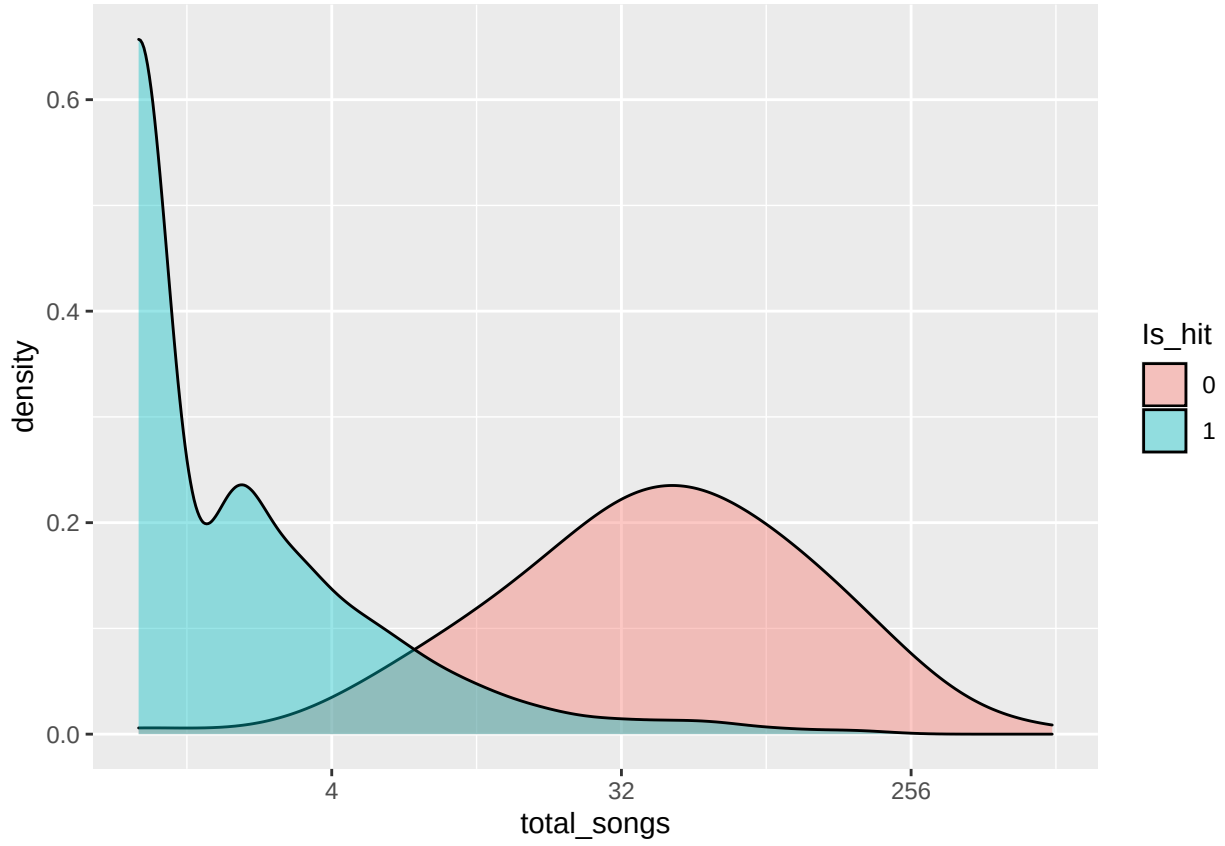
To answer this question i grouped data for these three variables:

```
#considering genres, key and mode as predictors

by_genre_key_mode <-training %>% select(genre_clean_factors, key, mode, Is_hit) %>% group_by( genre_clea
```

I studied the following visualisations:

- songs by genre and class:

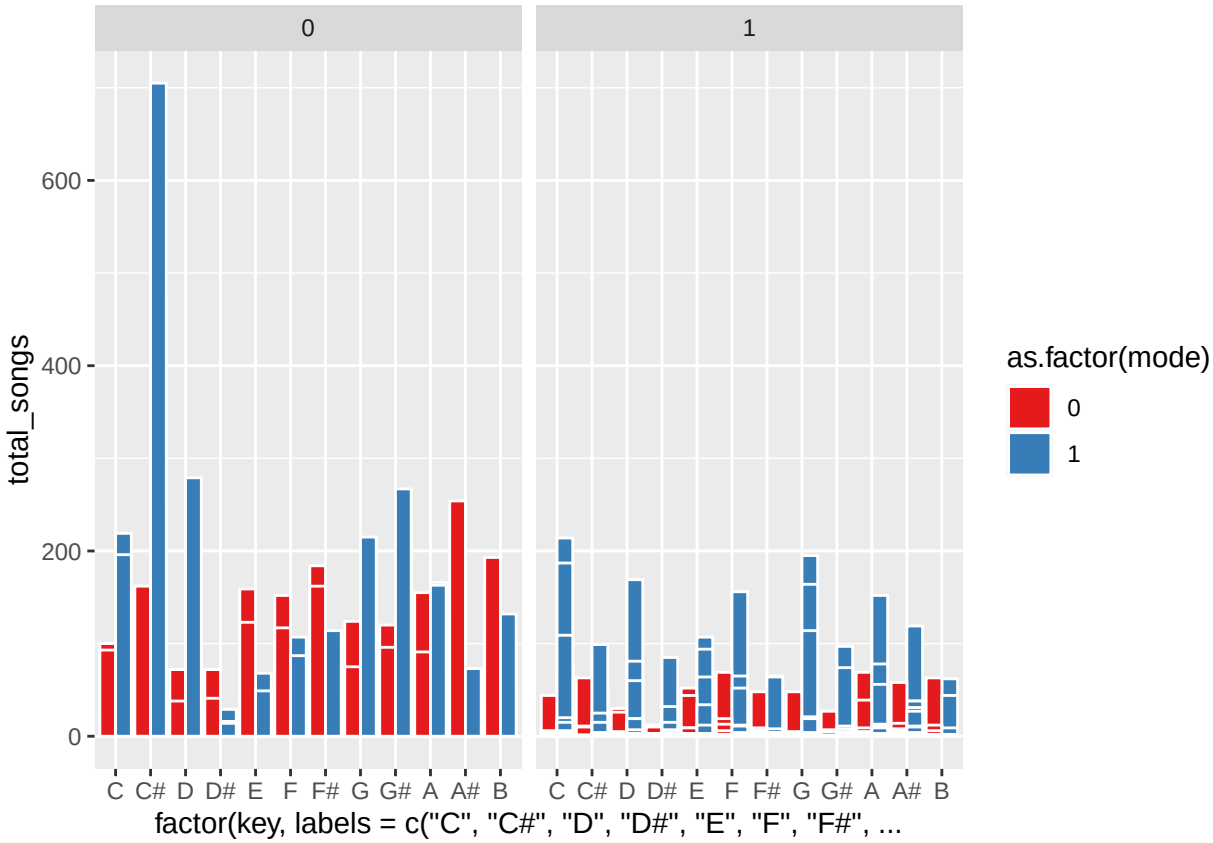
```
#checking distribution of songs by genre and class
by_genre_key_mode %>%
ggplot(., aes(x=total_songs, group=Is_hit, fill=Is_hit)) +
  geom_density(adjust=1.5, alpha=.4) +
  scale_x_continuous(trans='log2')
```



- relationship between mode and key:

```
#considering key and mode relationship
prop.table(table(training[,c("key","mode")]) ) *100
```

```
#considering key and mode relationship
by_genre_key_mode %>%
ggplot(., aes(x = factor(key, labels =c("C","C#","D","D#","E","F","F#","G","G#","A","A#","B")), y = tota
  geom_bar(
    aes(fill = as.factor(mode) ), stat = "identity", color = "white",
    position = position_dodge(0.9)
  )+
  facet_wrap(~ Is_hit) +
  scale_fill_brewer(palette = "Set1")
```



- considering mode by itself as potential predictor:

Understanding whether there is dependency between the class variable ("Is_hit") and Mode:

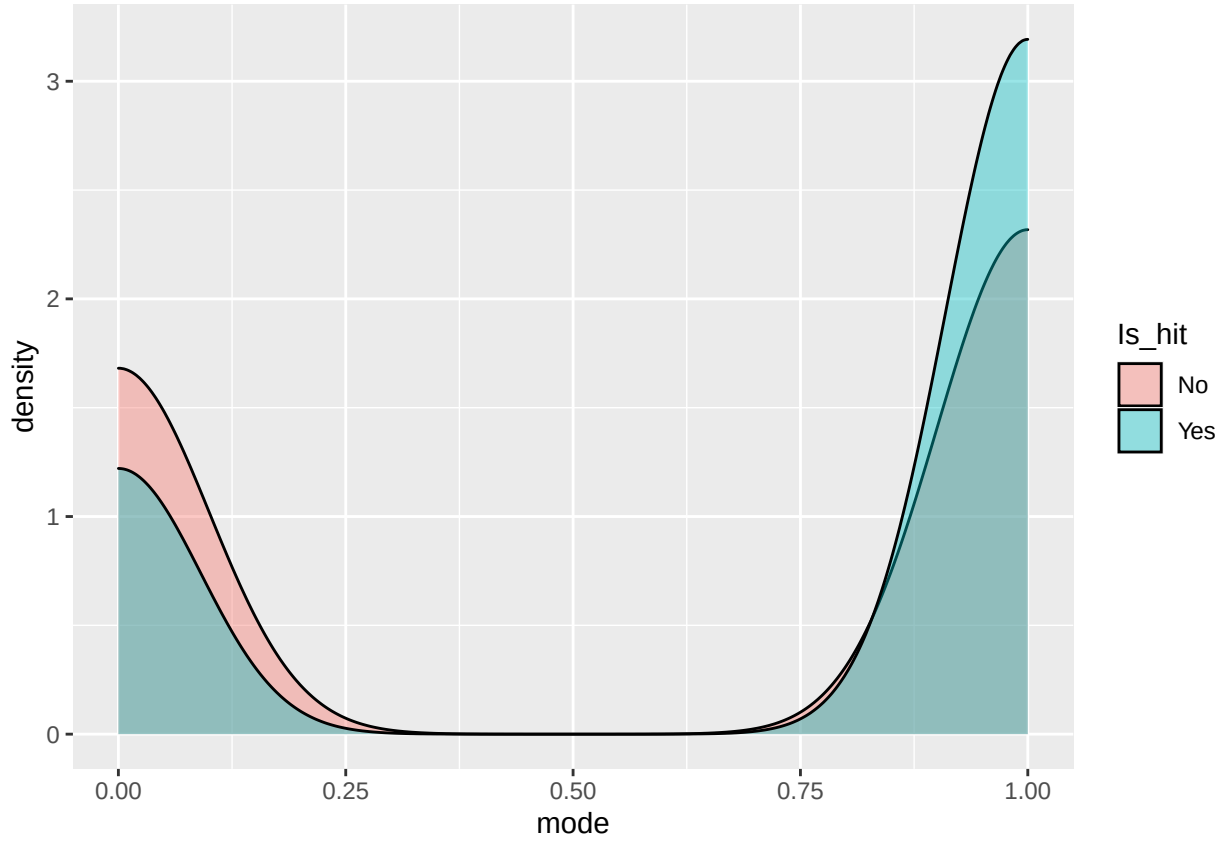
```
prop.table(table(training[,c("Is_hit","mode")]), margin = 2 ) *100
```

```
##       mode
## Is_hit        0        1
##    No  60.31438 44.48220
##    Yes 39.68562 55.51780
```

```
chisq.test(training$Is_hit, training$mode)  # independent variables
```

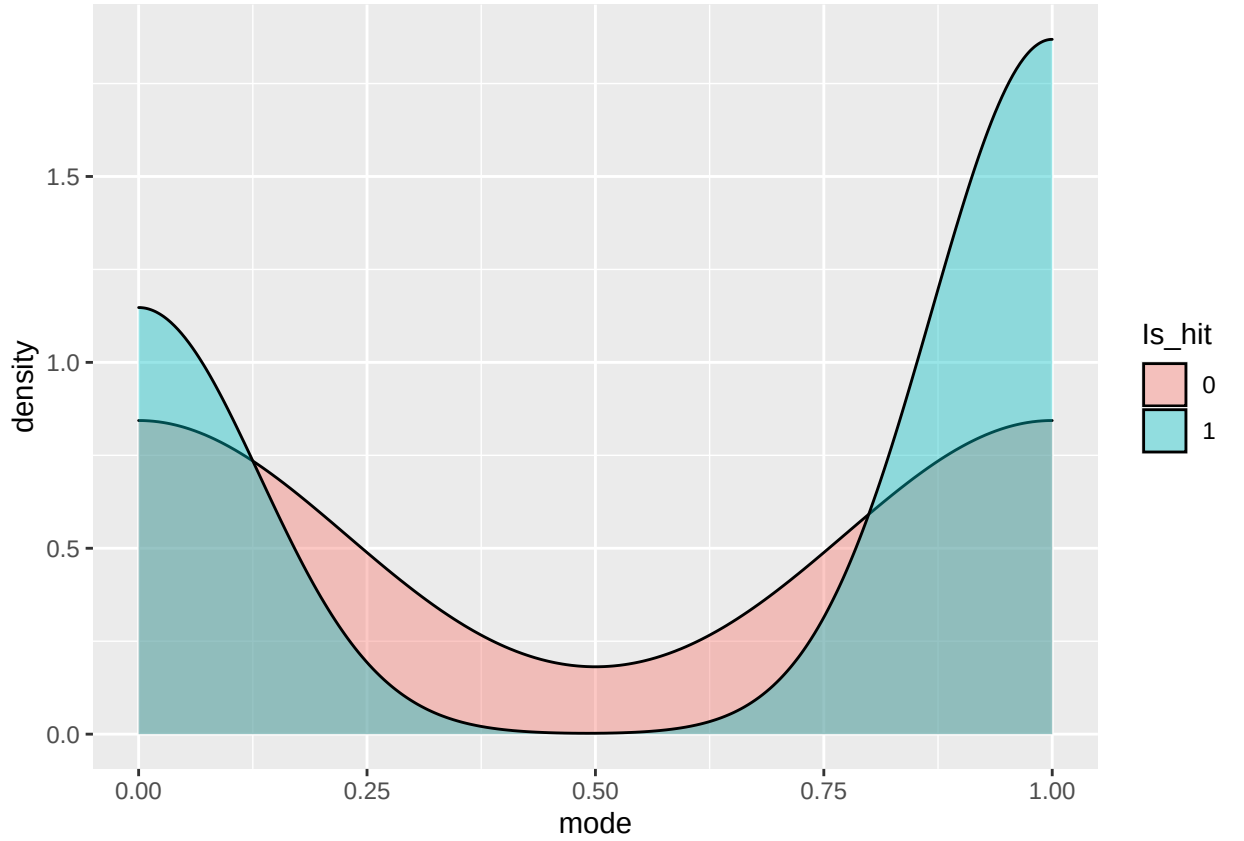
```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  training$Is_hit and training$mode
## X-squared = 606.06, df = 1, p-value < 2.2e-16
```

```
#mode density by class
training  %>%
ggplot(., aes(x=mode, group=Is_hit, fill=Is_hit)) +
   geom_density(adjust=1.5, alpha=.4)
```



Relationship between mode, key and genre by class

```
#considering mode in relation to genre and key
by_genre_key_mode  %>% group_by(mode) %>%
  ggplot(., aes(x=mode, group=Is_hit, fill=Is_hit)) +
  geom_density(adjust=1.5, alpha=.4)
```



insights:

- The density distribution of songs per genres stratified by class (hit/non-hit) shows a clear distinction between the classes. This together with the analysis from genres boxplot stratified by class indicates that genres could be a good predictor and should be included in the model.
- When combining mode with key is noted that most hit songs are in Major scales, except for B tonality where there is a balance. This is different for non-hit songs where there is a mix of songs composed in major and minor scales that have prevalence in the various tonalities.
- Usually non-hit songs are a majority for sharp/flat tone . While, hit songs composed in natural tones are more frequent than non-hit songs.
- There is overlap between the density distributions for classes by mode. However, when this is seen in the light of genre and key the overlapping reduced, allowing to predict some areas for hit songs. Mode might be a minor contributor to prediction and should therefore be included in the model.
- mode and IS_hit are independent as stated by the low p-value of the Chi.Square test conducted.

Are instrumentalness, acousticness and speechiness potential good predictors?

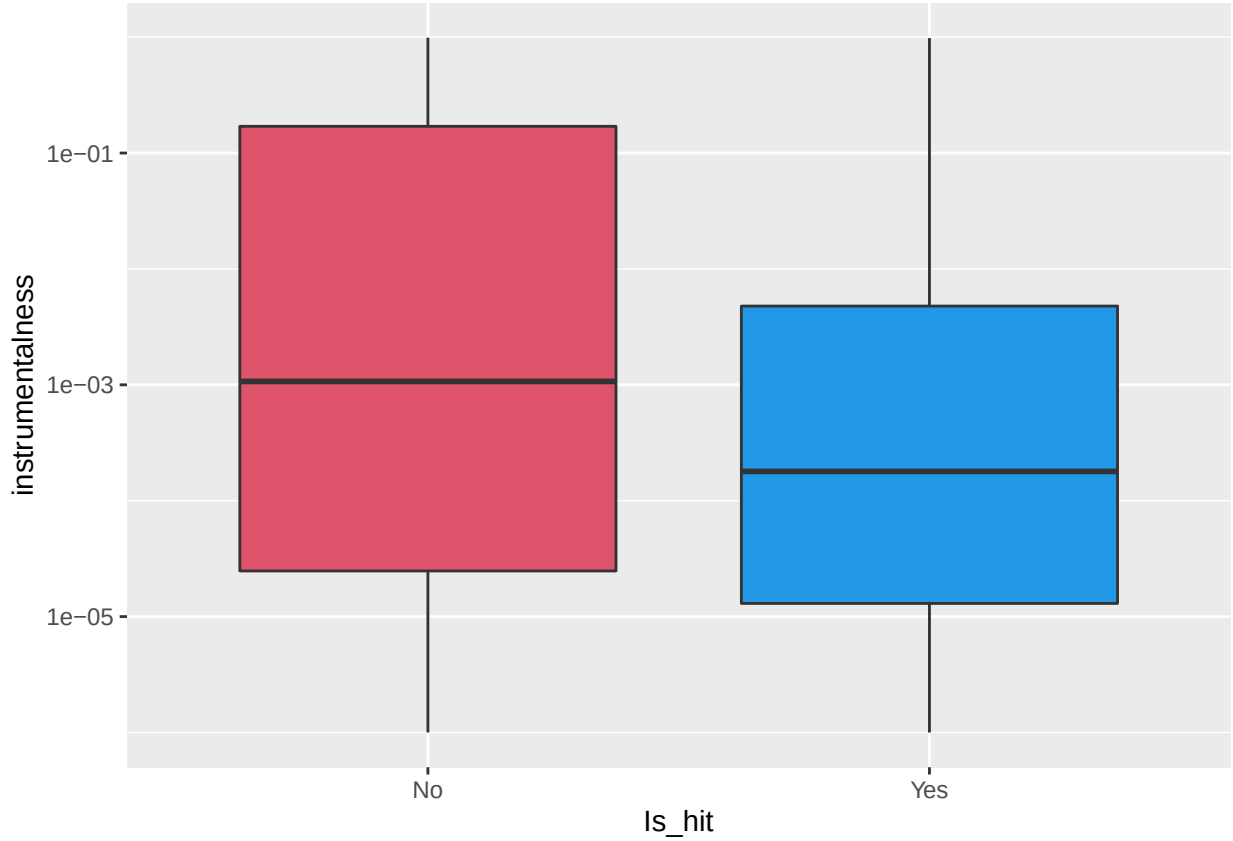
I studied the following visualisations:

- songs by instrumentalness and class:

```
#considering distribution of instrumentalness
#boxplot by class
training %>%
  ggplot(., aes(x=Is_hit, y=instrumentalness)) +
  geom_boxplot( fill= c(2,4)) +
  xlab("Is_hit")+
  scale_y_log10()
```

```
## Warning: Transformation introduced infinite values in continuous y-axis
```

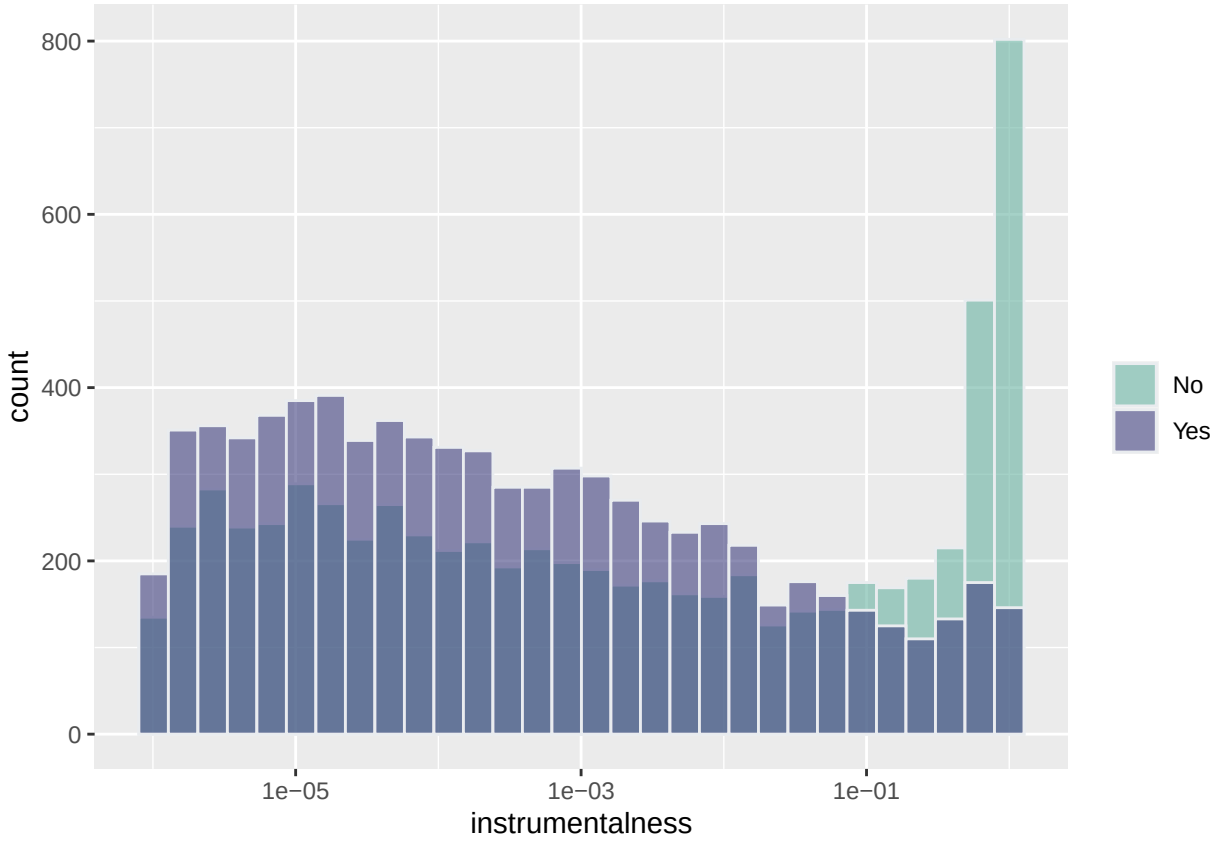
```
## Warning: Removed 11916 rows containing non-finite values (stat_boxplot).
```




```
# histogram by class
training %>%
  ggplot( aes(x=instrumentalness, fill=Is_hit)) +
  geom_histogram( color="#e9ecef", alpha=0.6, position = 'identity', bins = 30) +
  scale_fill_manual(values=c("#69b3a2", "#404080")) +
  labs(fill="")+
  scale_x_log10()
```

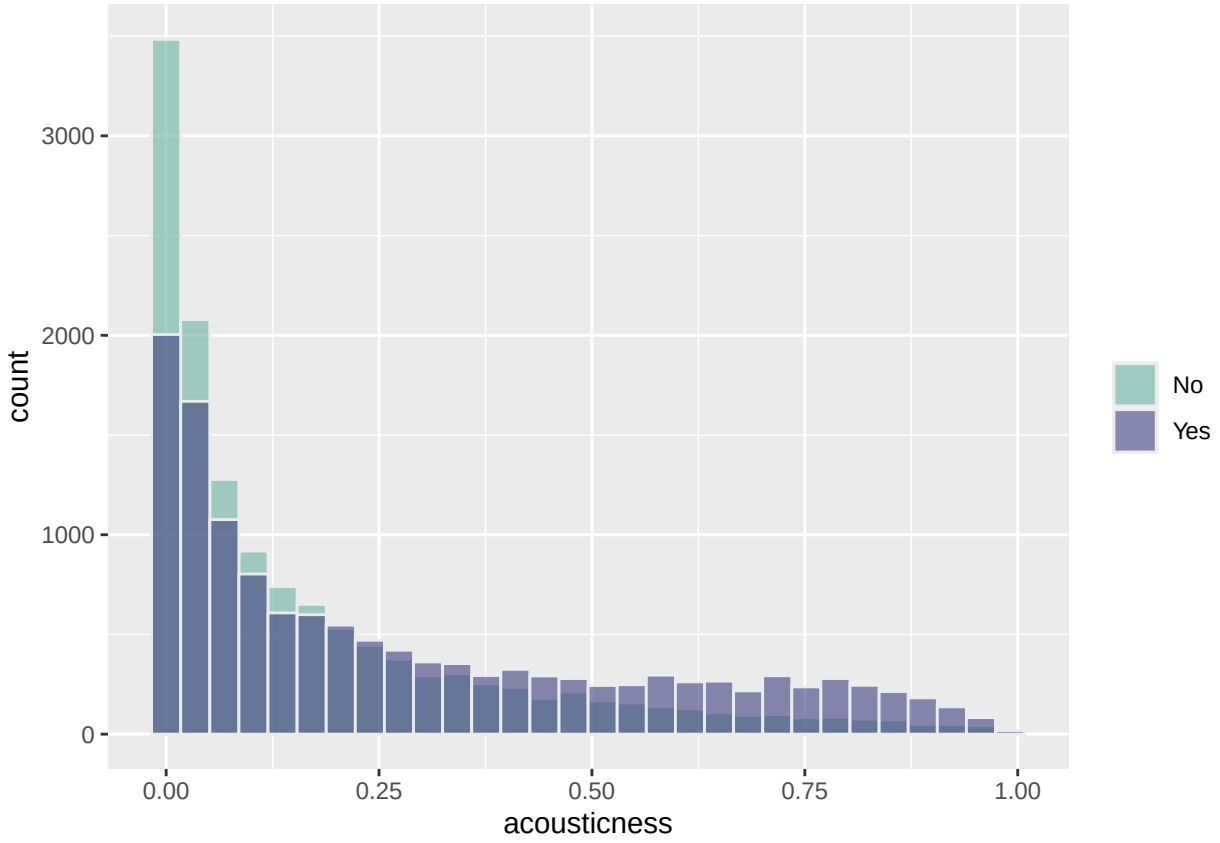
```
## Warning: Transformation introduced infinite values in continuous x-axis
```

```
## Warning: Removed 11916 rows containing non-finite values (stat_bin).
```

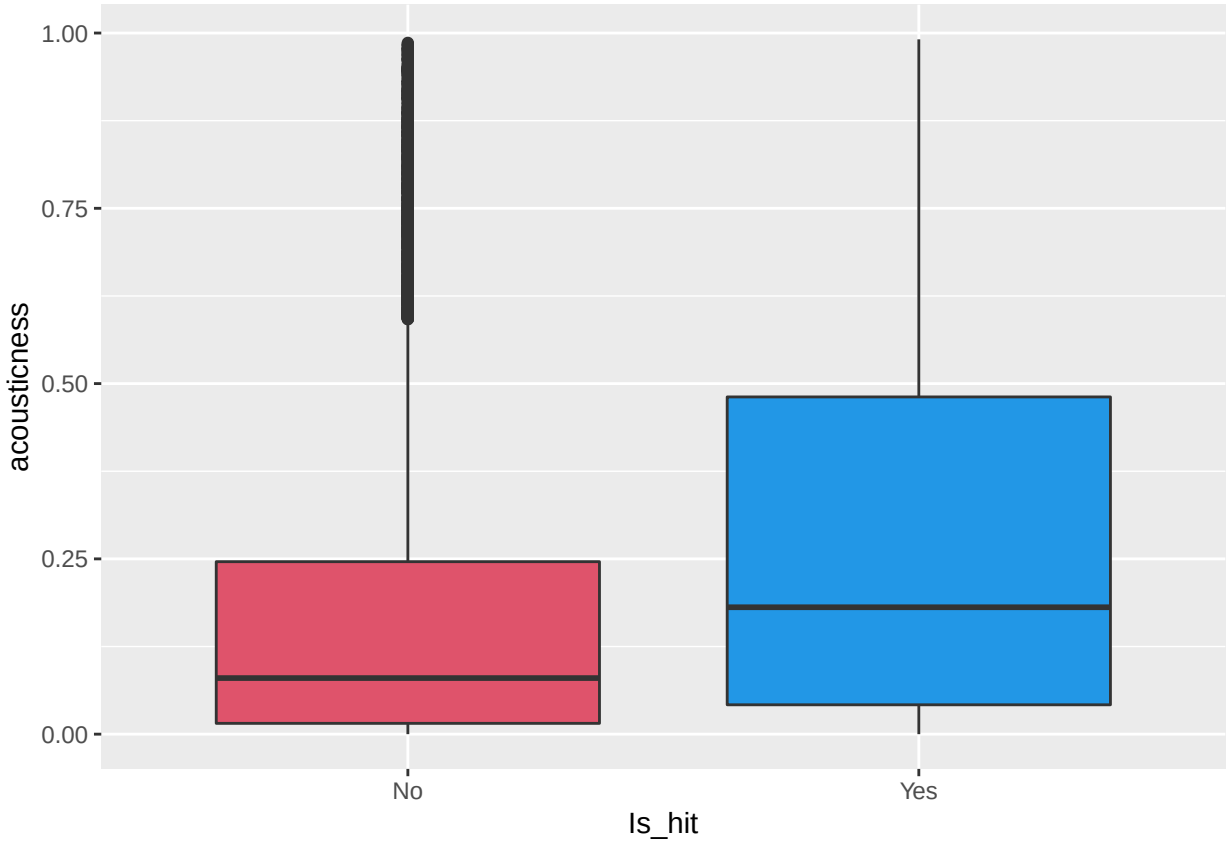


- songs by acousticness and class:

```
#considering the distribution of acousticness
#histogram by class
training %>%
  ggplot( aes(x=acousticness, fill=Is_hit)) +
  geom_histogram( color="#e9ecef", alpha=0.6, position = 'identity', bins = 30) +
  scale_fill_manual(values=c("#69b3a2", "#404080")) +
   labs(fill="")
```




```
#boxplot by class
training %>%
  ggplot(., aes(x=Is_hit, y=acousticness)) +
  geom_boxplot( fill= c(2,4)) +
  xlab("Is_hit")
```

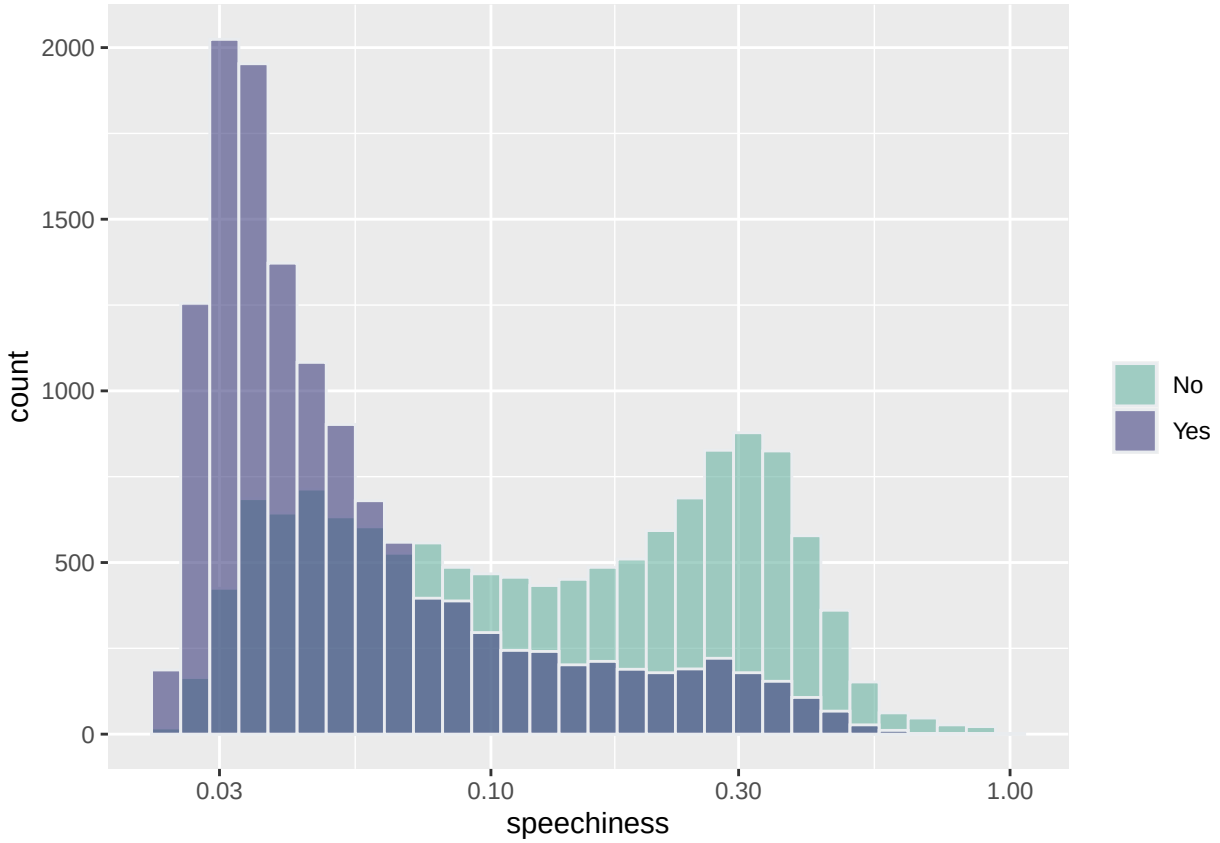


- songs by speechiness and class:

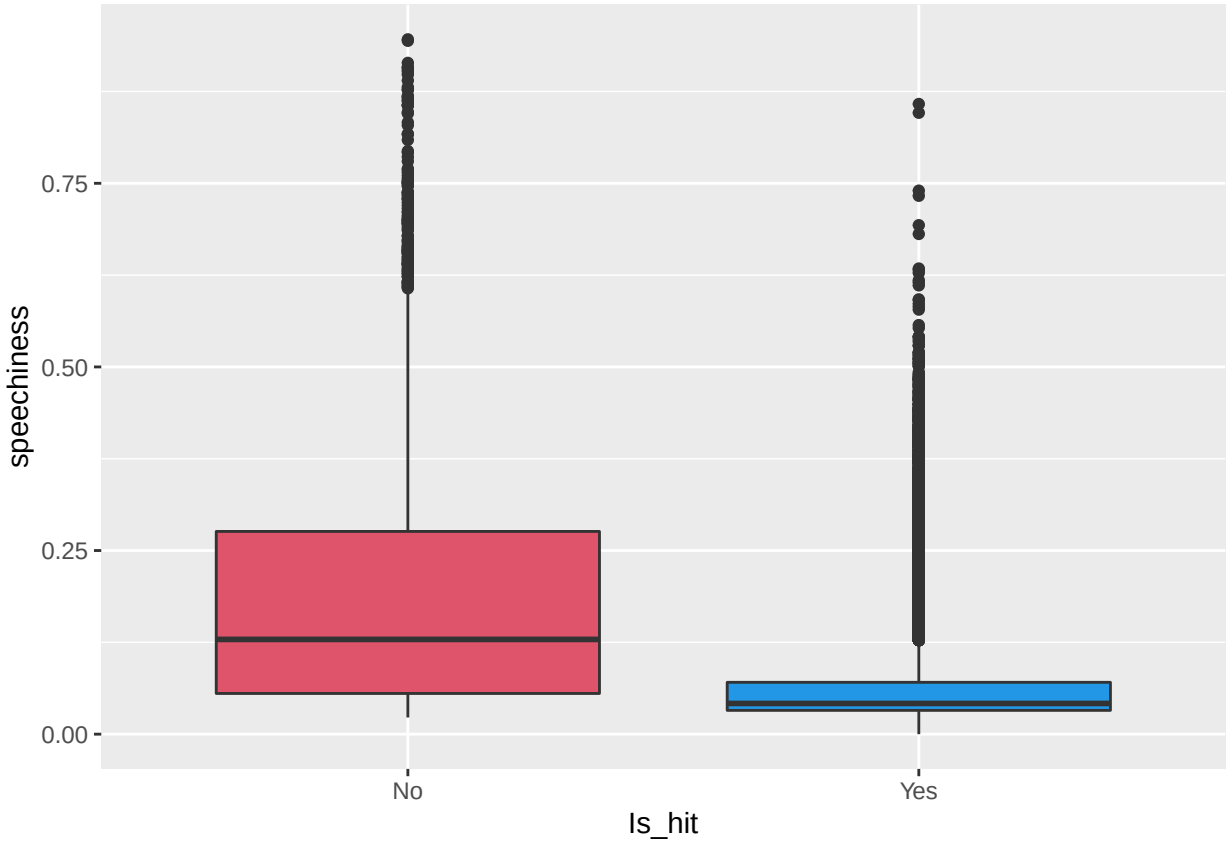
```
#considering the distribution of speechiness
#histogram by class
training %>%
  ggplot( aes(x=speechiness, fill=Is_hit)) +
  geom_histogram( color="#e9ecef", alpha=0.6, position = 'identity', bins = 30) +
  scale_fill_manual(values=c("#69b3a2", "#404080")) +
  labs(fill="") +
  scale_x_log10()
```

```
## Warning: Transformation introduced infinite values in continuous x-axis
```

```
## Warning: Removed 1 rows containing non-finite values (stat_bin).
```

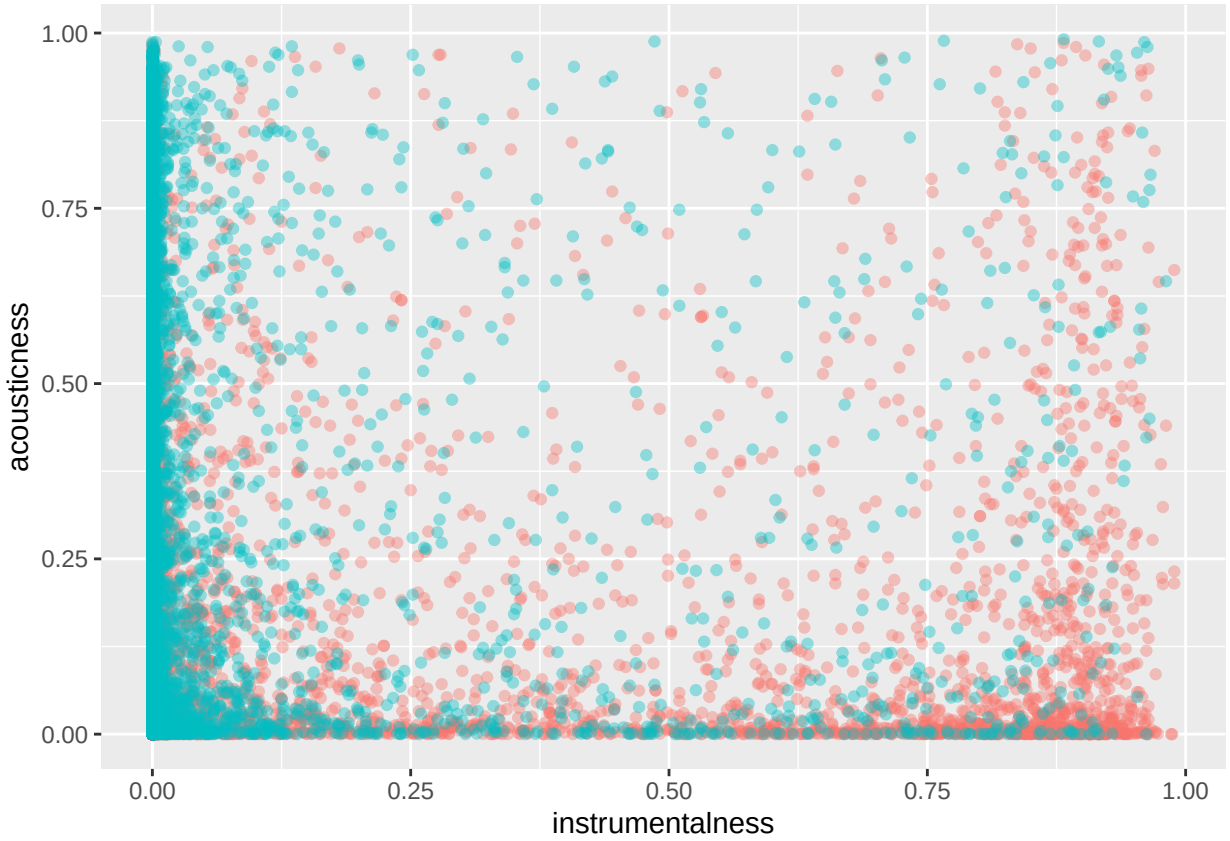



```
#boxplot by class
training %>%
  ggplot(., aes(x=Is_hit, y=speechiness)) +
  geom_boxplot( fill= c(2,4)) +
  xlab("Is_hit")
```



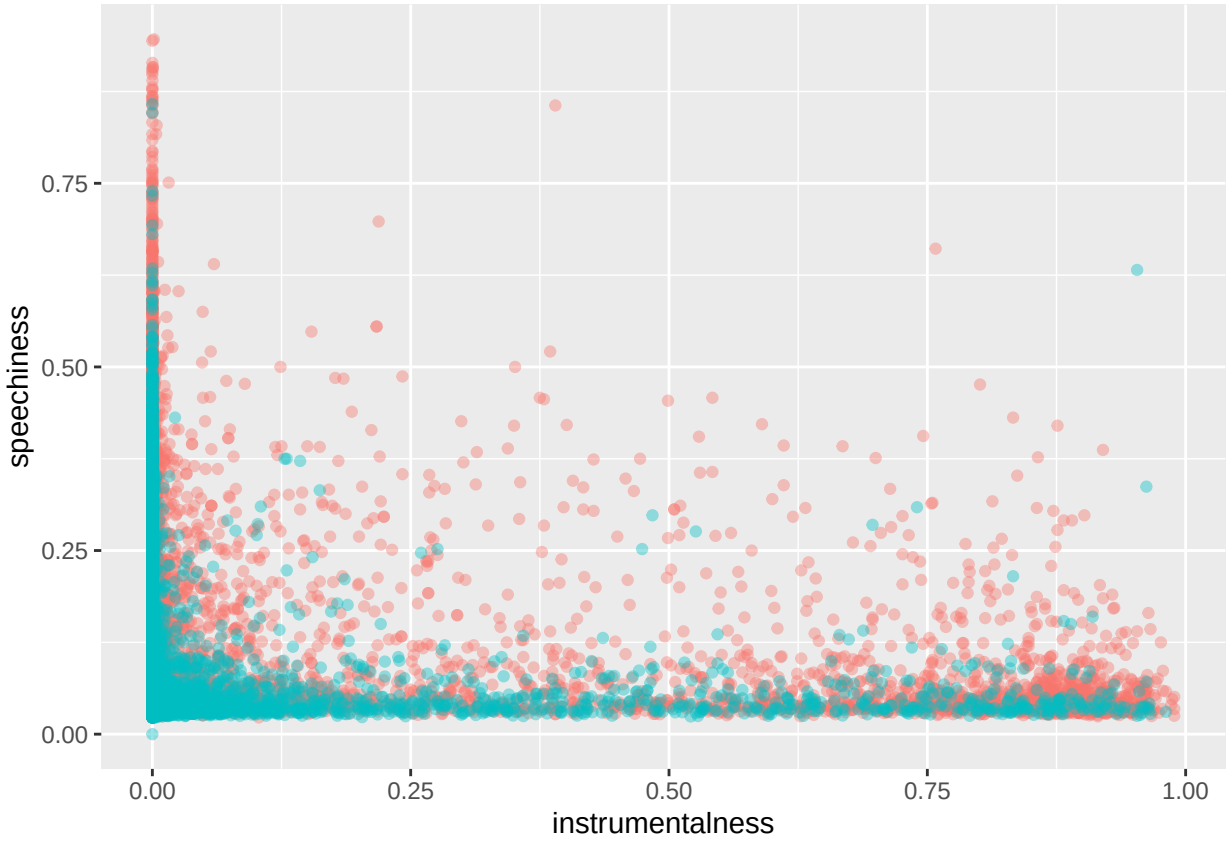
- relationship between instrumentalness and acousticness:

```
#considering the relationship between instrumentalness and acousticness
training %>%
  ggplot(., aes(x=instrumentalness, y=acousticness, color=Is_hit)) +
  geom_point(alpha = 0.4) +
  theme(legend.position="none")
```



- relationship between instrumentalness and speechiness:

```
#considering the relationship between instrumentalness and speechiness
training %>%
  ggplot(., aes(x=instrumentalness, y=speechiness, color=Is_hit)) +
  geom_point(alpha = 0.4) +
  theme(legend.position="none")
```



insights:

- Instrumentalness analysis indicates that instrumental songs (i.e no vocals) have a tendency to be non-hit songs. However, there is significant overlapping of the classes in the distribution. This feature by itself would not be a good predictor.
- Acousticness analysis indicate that most songs are not acoustic. However, there is more presence of acoustic songs within the hit group than the non-hit group. Some outliers noted in the non-hit group, with value above 0.6, indicated that very acoustic songs tends to be non-hits. This feature might make a small contribution to predictions
- Speechiness suggest that most songs contain both music and words. The non-hit songs have a bi-modal distribution with most songs showing high levels of speechiness. This feature might be a good contributor to predictions.
- the relationship between instrumentalness and acousticness shows a large portion of hits songs that are either low in instrumentalness or acousticness. As instrumentalness increases, the probability of being a non-hit song slightly increases. However, we can see a number of songs that are high in both features. The combination of these two variables might be good candidates for a classification tree model.
- For instrumentalness and speechiness, we can see that hit songs tend to be low in speechiness or instrumentaless. This could be related to certain genres such as: rap and hip hop, where the signing techniques resemble speeches. There are few outliers for the hit group that are high in both features. An increase in both features seems to yield a larger number of non-hit songs.

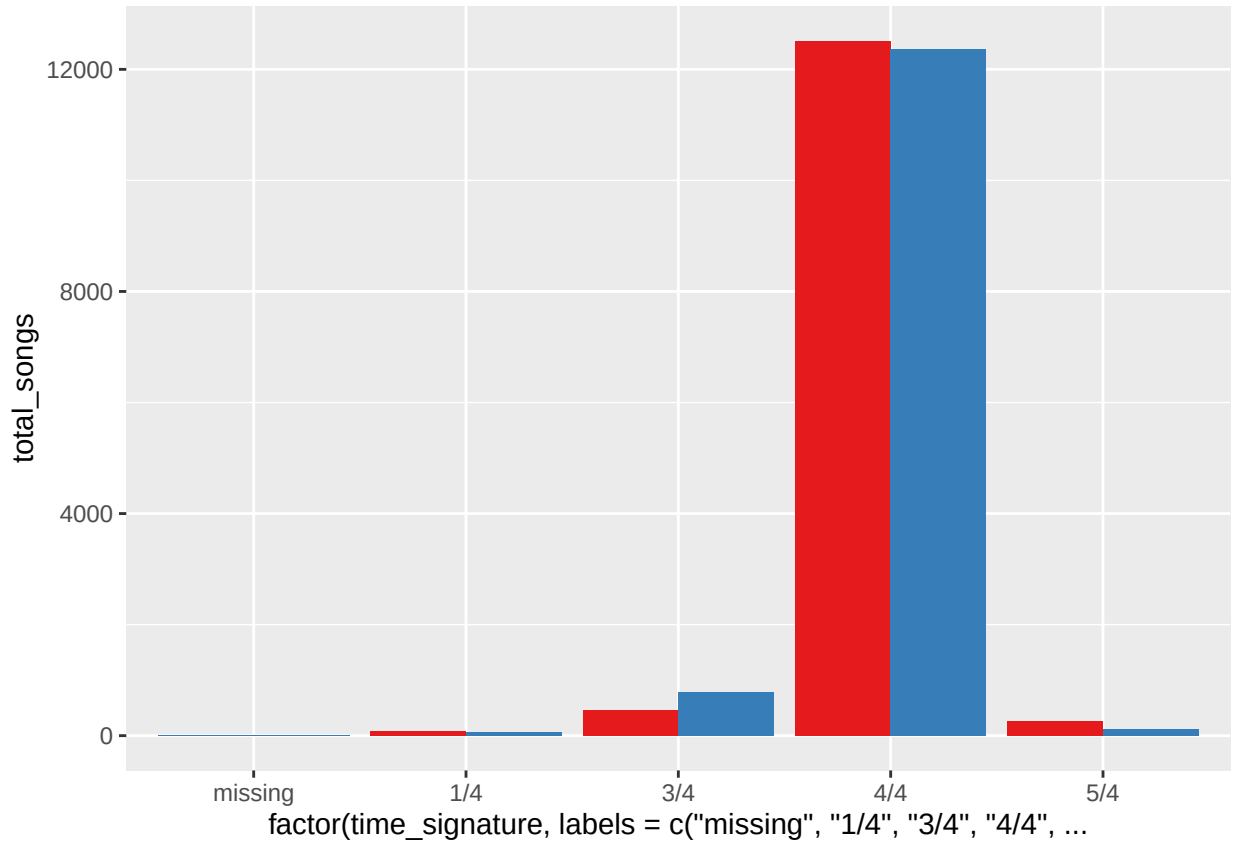
Analysis part II: features describing elements of songs speed

Are time_signature and liveness potential good predictors?

- songs by time_signature and class:

```
#considering time_signature distribution

#time_signature distribution by classes

summary(training$time_signature)

by_time_sign_count <-training %>% select(time_signature, Is_hit) %>% group_by( time_signature, Is_hit) %
```

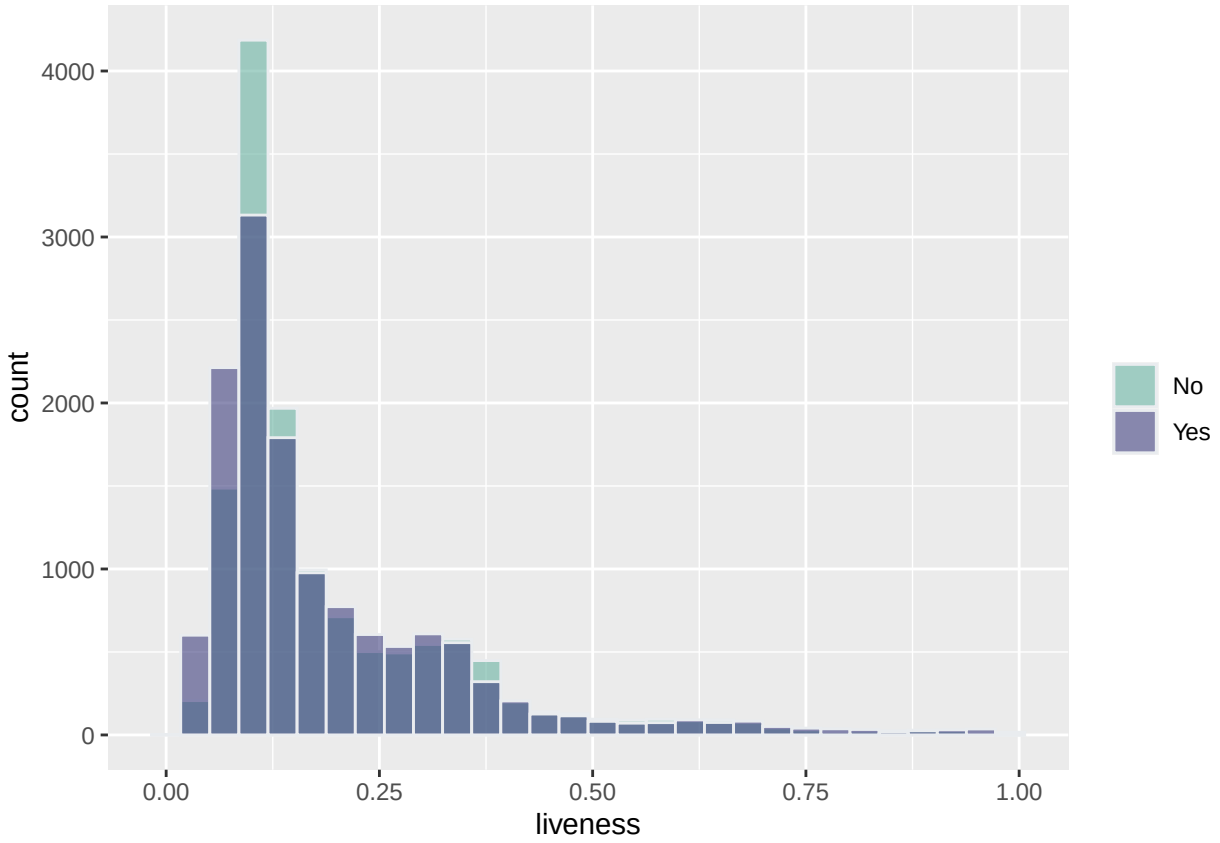
```
#class by time_signature categories
by_time_sign_count %>%
  ggplot(., aes(x=factor(time_signature, labels =c("missing", "1/4","3/4","4/4","5/4")),
                y = total_songs, fill = Is_hit )) +
  geom_bar(stat = "Identity" , position = "dodge") +
  scale_fill_brewer(palette = "Set1")  +
  theme(legend.position="none")
```



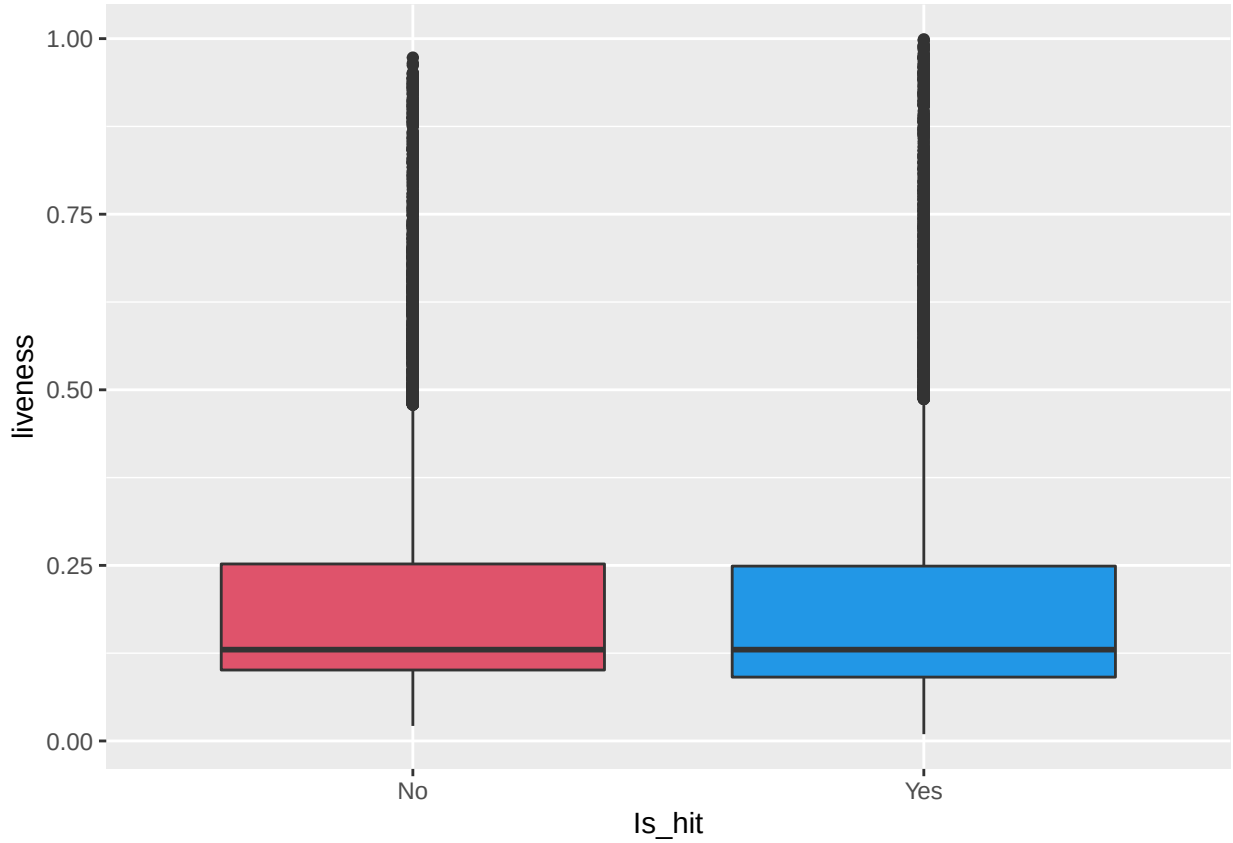
- songs by liveness and class:

```
#considering the distribution of liveness
#histogram by class
training %>%
  ggplot( aes(x=liveness, fill=Is_hit)) +
```

```
  geom_histogram( color="#e9ecef", alpha=0.6, position = 'identity', bins= 30) +
  scale_fill_manual(values=c("#69b3a2", "#404080")) +
  labs(fill="")
```




```
#boxplot by class
training %>%
  ggplot(., aes(x=Is_hit, y=liveness)) +
  geom_boxplot( fill= c(2,4)) +
  xlab("Is_hit")
```



insights:

- Time_signature indicates that most songs are written in "4/4". Hit and non-hit songs distribute similarly across the categories. Therefore this variable by itself is not useful for prediction. Two songs had value zero (i.e. equivalent to n/a). The "3/4" category had more hit than non-hit songs, while in the "5/4" category there is prevalence of non-hit songs.
- Liveness seems to have a similar distribution across the classes and might not be an important contributor to predictions. The median (around 0.125) and distribution per classes is similar. This indicates that majority of songs (both hit and non-hit) are not recorded in front of a live audience. Some outliers were noted at the upper end of the distribution .

Are tempo, valence, duration, danceability, and energy potential good predictors?

The distributions visualisations previously performed indicate that energy and danceability do not has distinct distribution among the classes, therefore no further review of these distributions was considered. However, i studied the relationship of these variables with other variables of interest. Previously, I have also noted that energy and loudness are correlated.

- songs by tempo and class:

```
# considering tempo interquartile by classes

training %>% filter(Is_hit== "No") %>% select(tempo) %>% summary(.)
```

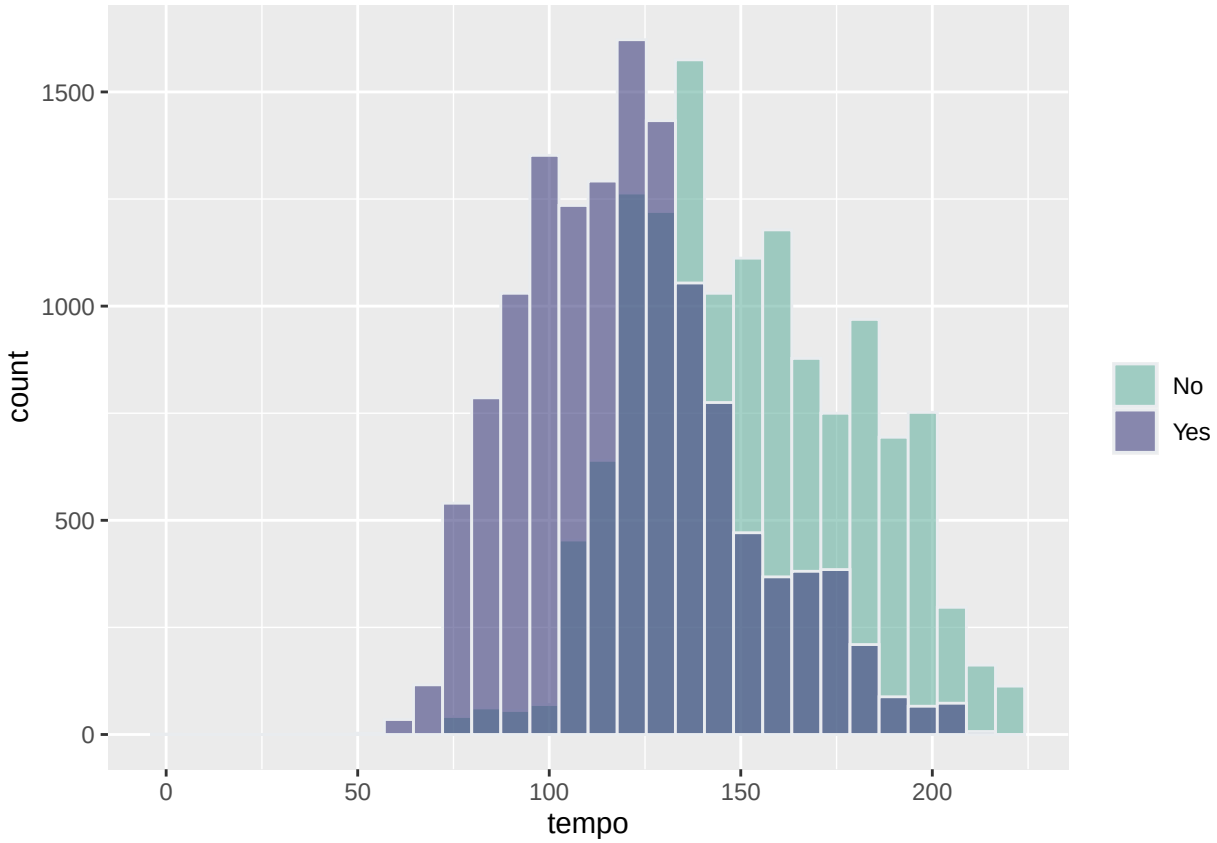
```
##      tempo
```

```
##  Min.   : 61.31
##  1st Qu.:130.01
##  Median :149.98
##  Mean   :152.46
##  3rd Qu.:174.99
##  Max.   :220.29
```

```
training %>% filter(Is_hit== "Yes") %>% select(tempo) %>% summary(.)
```

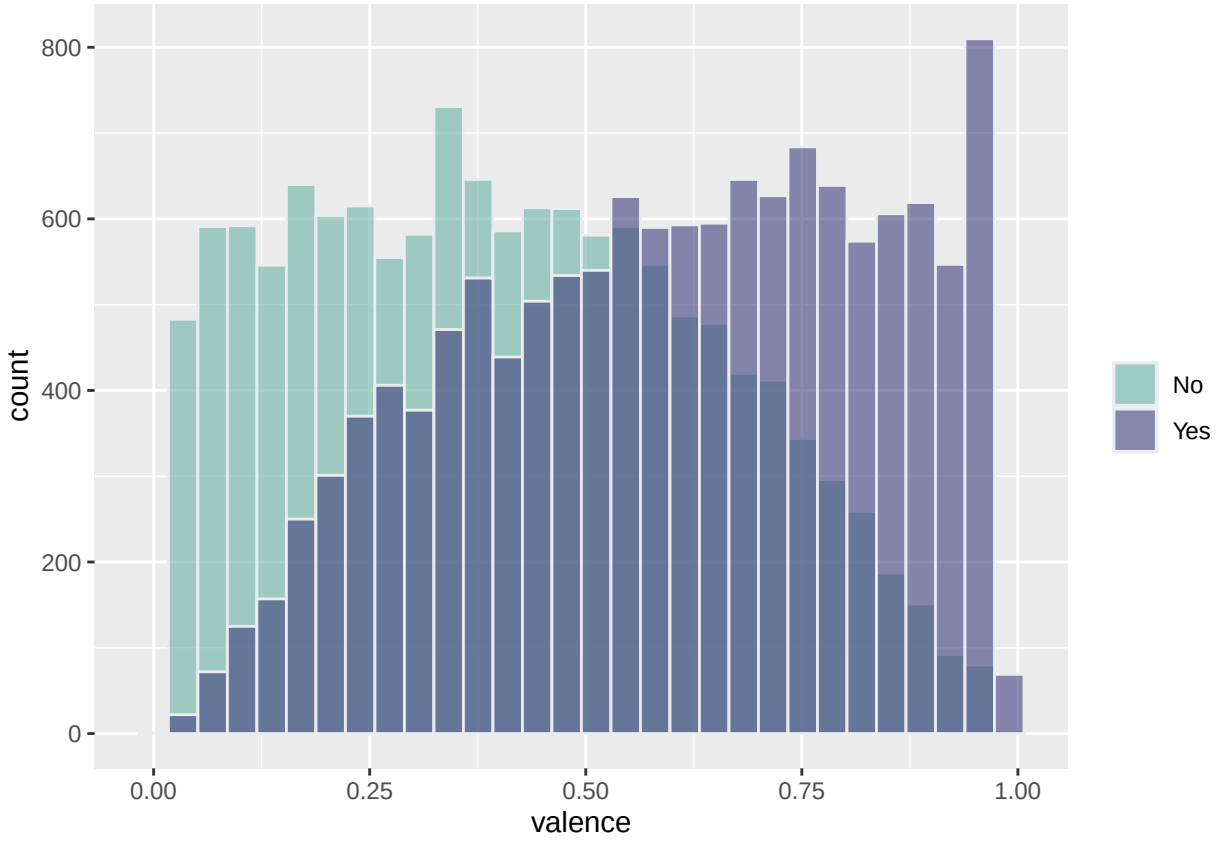
```
##      tempo
##  Min.   :  0.00
##  1st Qu.: 99.58
##  Median :119.20
##  Mean   :120.59
##  3rd Qu.:136.80
##  Max.   :216.20
```

```
#tempo distribution by classes
training %>%
  ggplot( aes(x=tempo, fill=Is_hit)) +
  geom_histogram( color="#e9ecef", alpha=0.6, position = 'identity', bins = 30) +
  scale_fill_manual(values=c("#69b3a2", "#404080")) +
  labs(fill="")
```



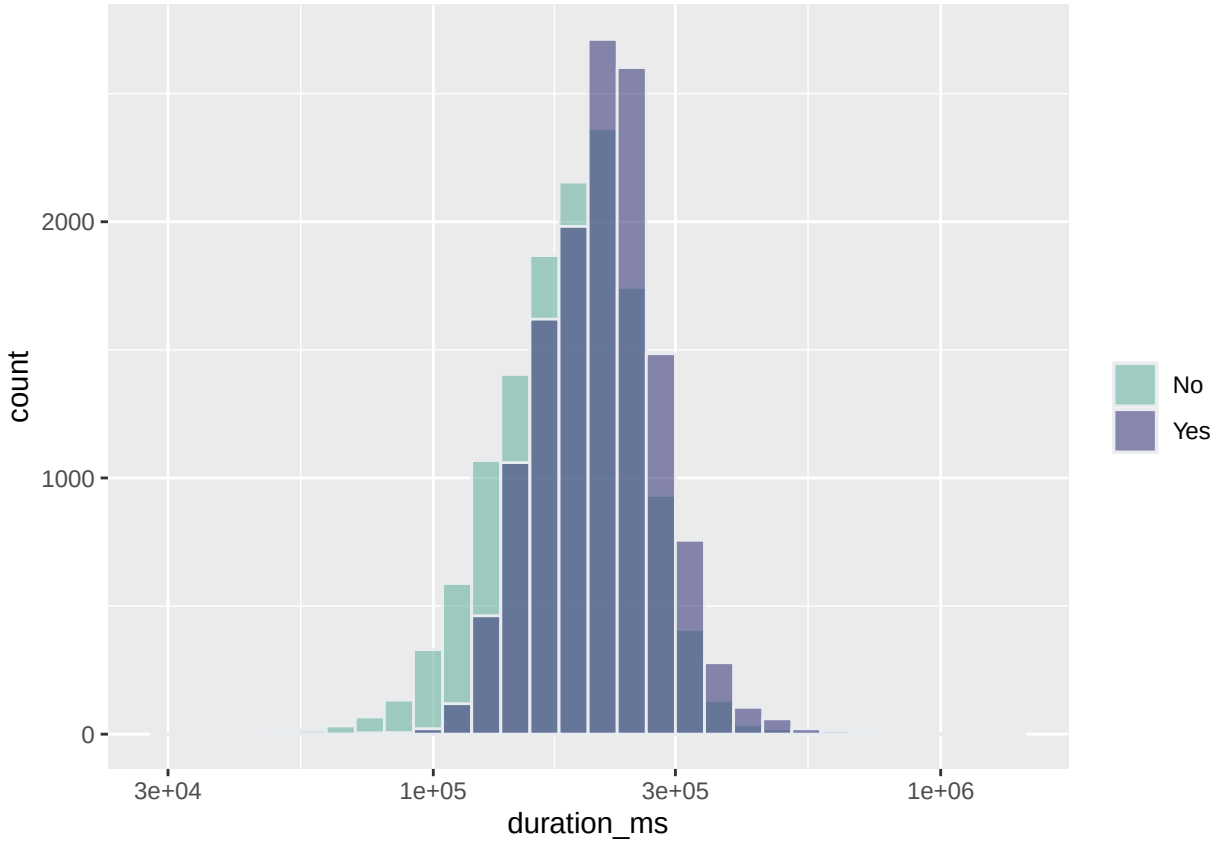
- songs by valence and class:

```
#valence distribution by classes
training %>%
  ggplot( aes(x=valence, fill=Is_hit)) +
  geom_histogram( color="#e9ecef", alpha=0.6, position = 'identity', bins = 30) +
  scale_fill_manual(values=c("#69b3a2", "#404080")) +
    labs(fill="")
```

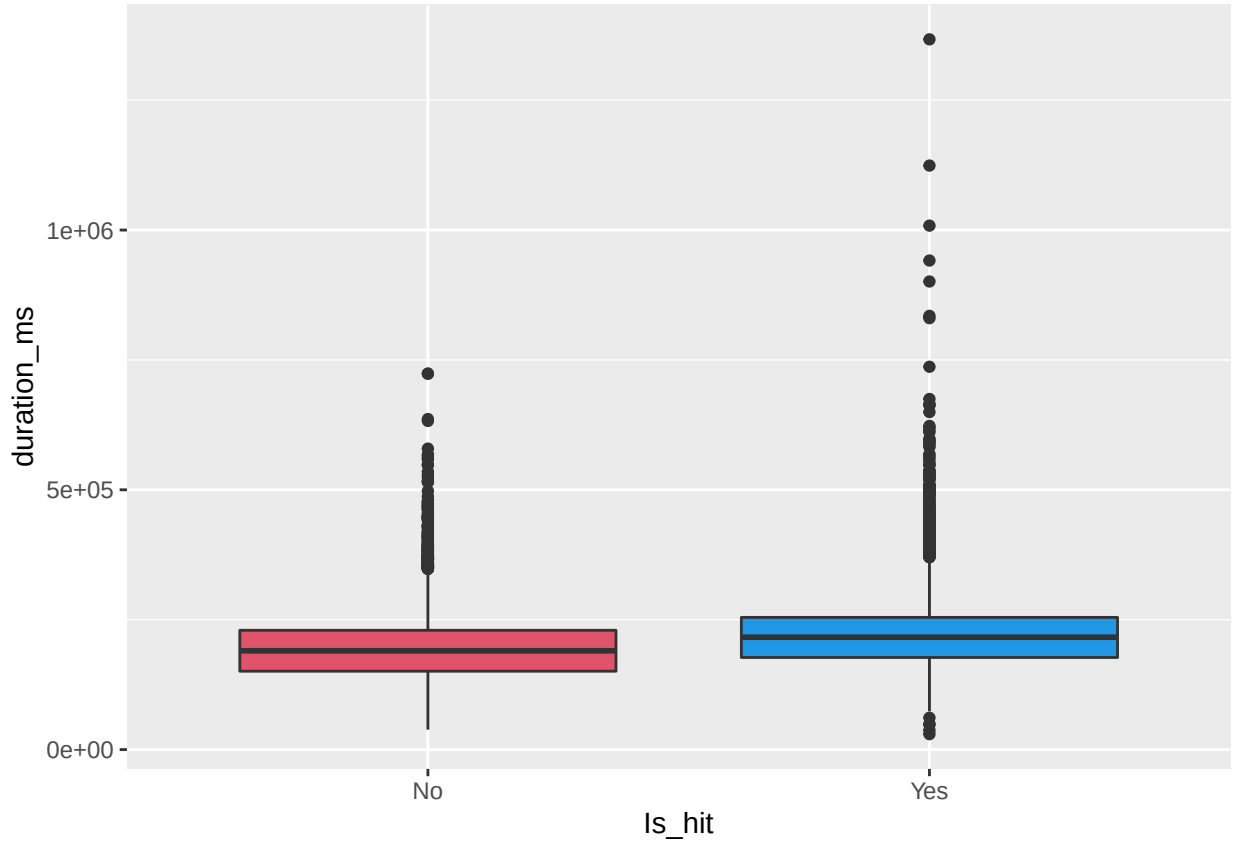


- songs by duration and class:

```
#considering the distribution of duration
#histogram by class
training %>%
  ggplot( aes(x=duration_ms, fill=Is_hit)) +
  geom_histogram( color="#e9ecef", alpha=0.6, position = 'identity', bins = 30) +
  scale_fill_manual(values=c("#69b3a2", "#404080")) +
  labs(fill="") +
  scale_x_log10()
```

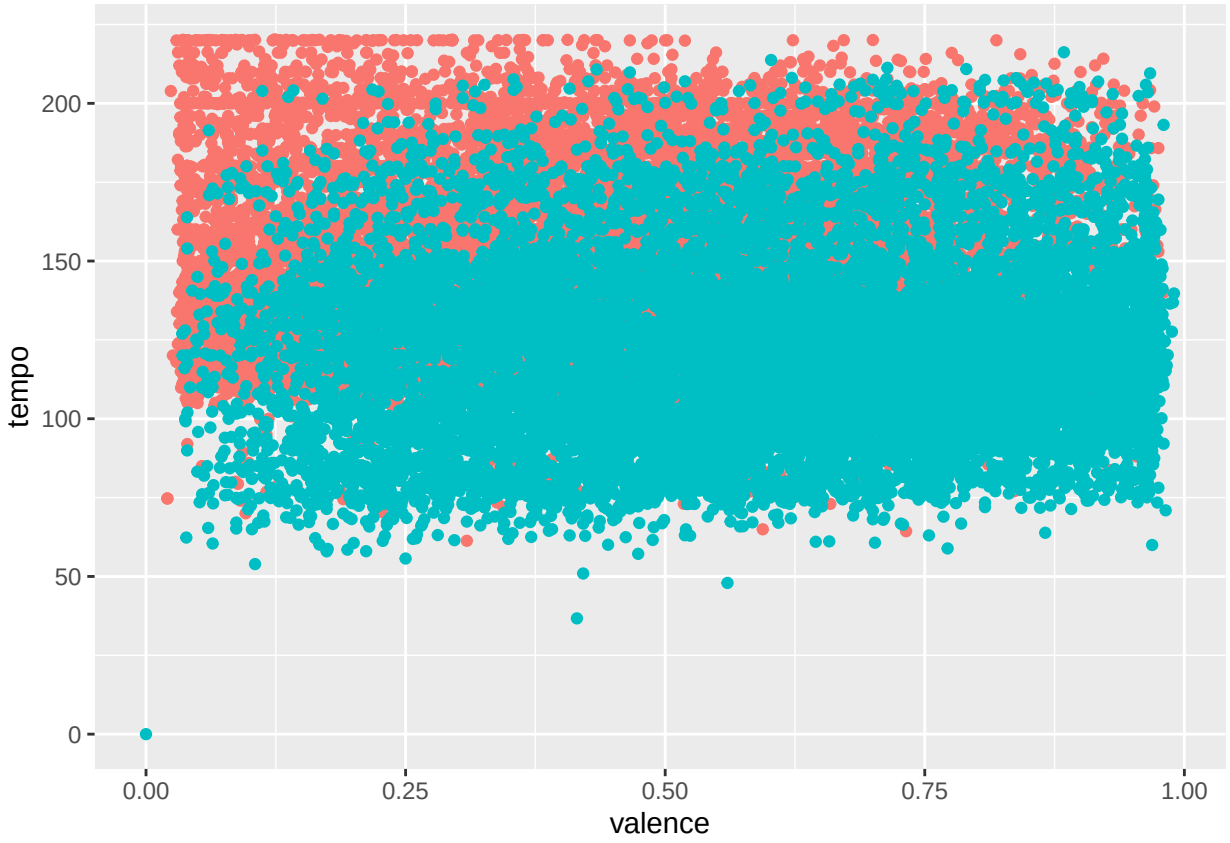



```
#boxplot by class
training %>%
  ggplot(., aes(x=Is_hit, y=duration_ms)) +
  geom_boxplot( fill= c(2,4)) +
  xlab("Is_hit")
```



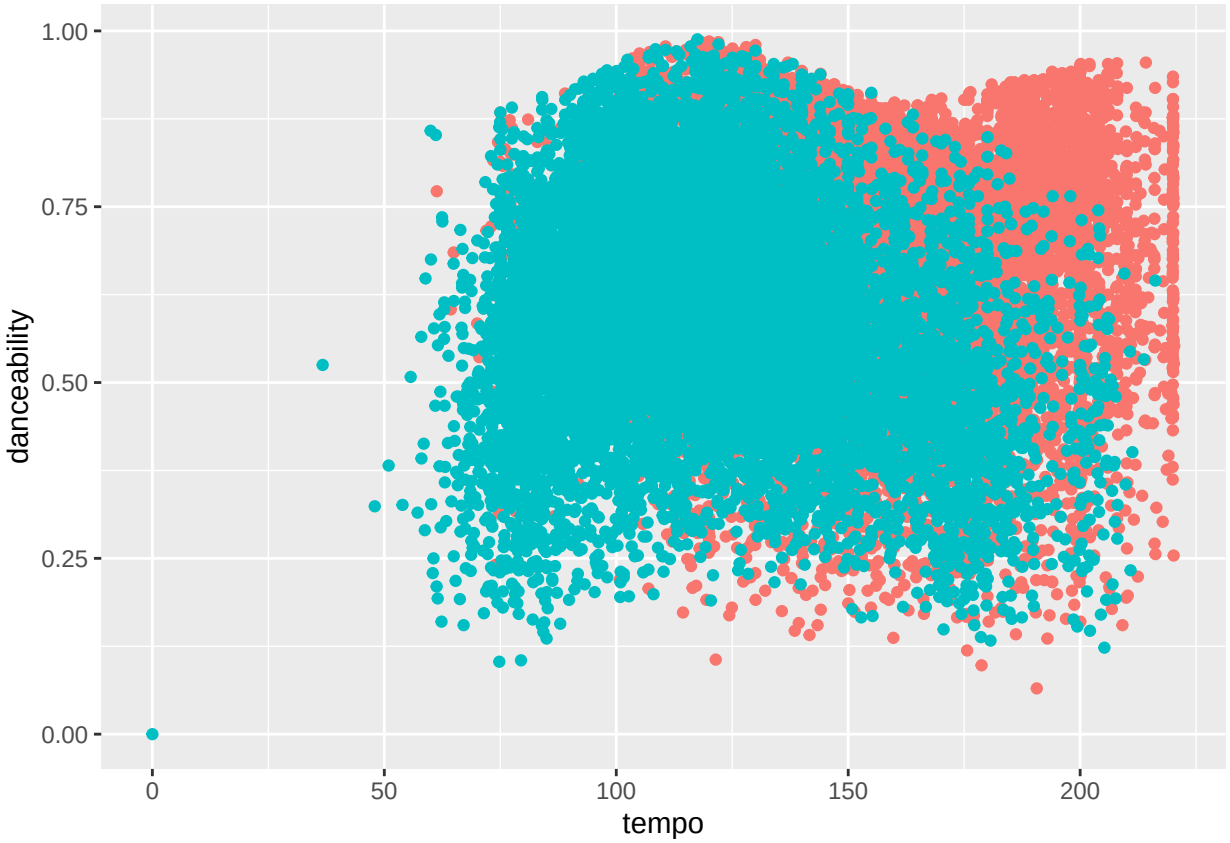
- relationship of tempo and valence :

```
#considering tempo and valence relationship
training %>%
  ggplot(., aes(x=valence, y=tempo, color=Is_hit)) +
  geom_point() +
  theme(legend.position="none")
```



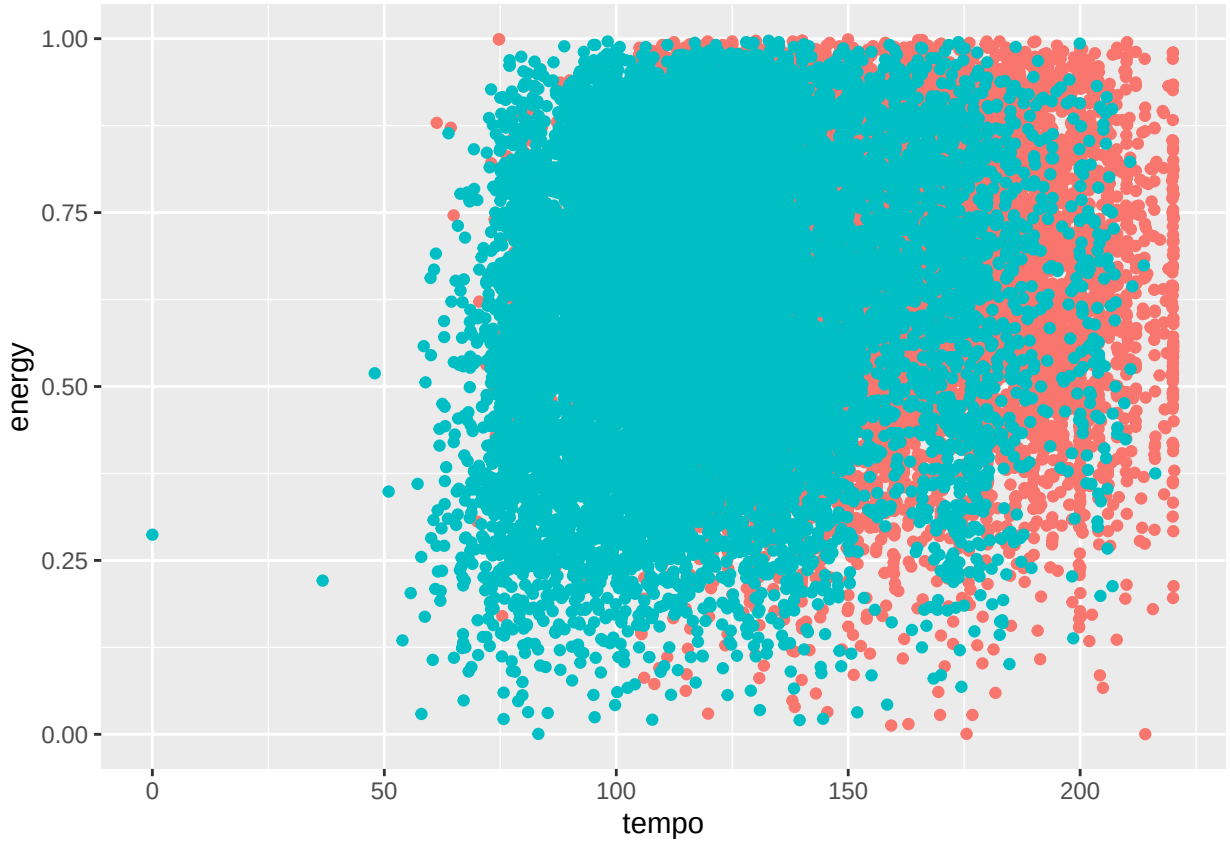
- relationship of tempo and danceability :

```
#considering tempo and danceability relationship
training %>%
  ggplot(., aes(x=tempo, y=danceability, color=Is_hit)) +
  geom_point() +
  theme(legend.position="none")
```



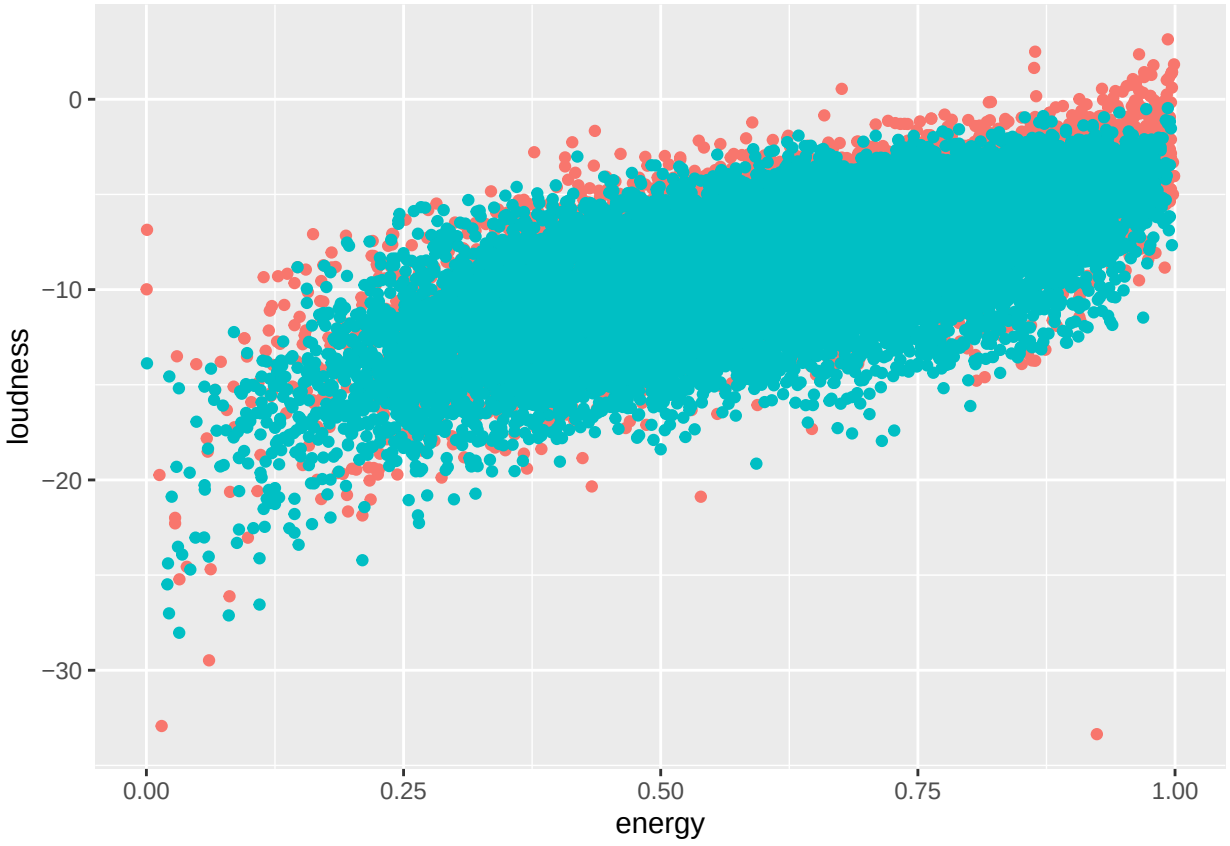
- relationship of tempo and energy :

```
#considering tempo and energy relationship
training %>%
  ggplot(., aes(x=tempo, y=energy, color=Is_hit)) +
  geom_point() +
  theme(legend.position="none")
```



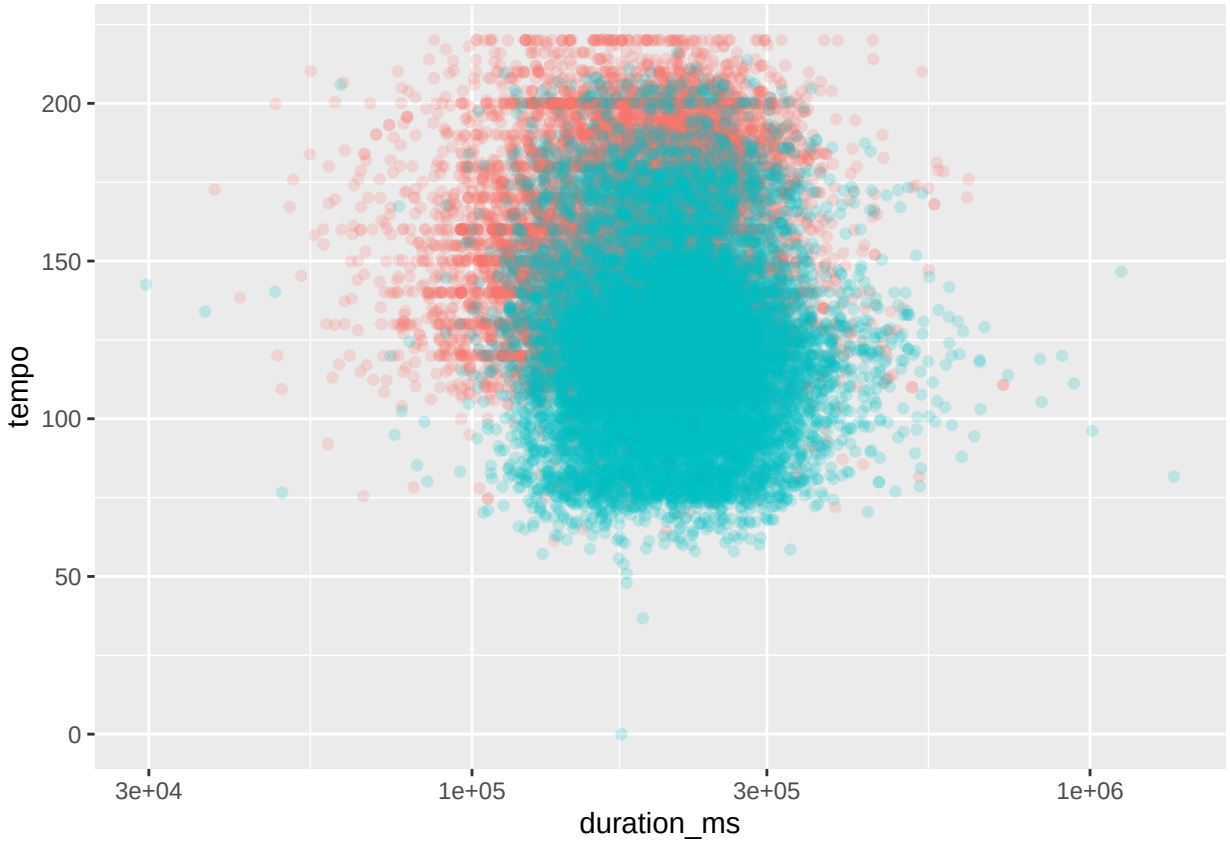
- relationship of energy and loudness :

```
#considering the relationship between energy and loudness
training %>%
  ggplot(., aes(x=energy, y=loudness, color=Is_hit)) +
  geom_point() +
  theme(legend.position="none")
```



- relationship of duration and tempo :

```
#considering the relationship between tempo and duration
training %>%
  ggplot(., aes(x=duration_ms, y=tempo, color=Is_hit)) +
  geom_point(alpha = 0.2) +
  theme(legend.position="none")+
  scale_x_log10()
```



insights:

- Tempo has a different distribution per class with a median lower for hit songs than the second quartile for non-hit songs. There is some level of overlapping on the distribution by classes, but some areas are distinct. Tempo might be a good contributor to predictions.
- The average tempo for hit songs is around 120 bpm, this is the top level of "moderato" tempo, with an interquartile between 99 to 136 bpm (andante moderato to allegro tempo). Non-hit songs have a mean of 156 bpm with an interquartile between 130 to 174 bpm (allegro to vivacissimo). This means songs that have very fast pace will be less popular than songs that have a moderate tempo, with the average song keeping a slightly fast tempo.
- Danceability has a normal distribution with a median of around 0.7. Hit songs has a lower median than non-hit songs, but distributions overlap. There are few outliers at the bottom quartile.
- Energy also has a distribution that resemble a normal distribution but slightly left skewed. Loundness also appears to have a normal distribution. It is positive correlated to energy and might be convenient to exclude one of these variables from the model.
- For danceability and energy relation the classes overlap. These two parameters might not be good predictors on their own.
- Valence seems to have a distinct distribution between the classes with hit songs having a median above the interquartile for non-hit songs. Hit songs have a mean valence around 0.6. There is some level of overlapping between the classes on valence distribution. Valence can be a contributor to predictions.
- Duration_ms appears to have a normal distribution with classes overlapping. Hit songs seems to have slight longer duration than no-hit songs in average. Some outliers were noted for both classes, with

higher outliers for the hit songs (i.e. longer songs). This feature might be a minor contributor to predictions.

- When considered the relationship between duration and tempo, it seems that faster songs that are longer tend to be in the hit group, while shorter songs in general tend to be non-hit, with a predominance of moderate to fast songs.

Features selection

The previous analysis provided two sets of possible features that can be used in the models as follows:

- by song type
- by song speed

A third alternative taken was looking for automated feature selection using the package Boruta.

Boruta uses Random Forest. The method performs a top-down search for relevant features by comparing original attributes' importance with importance achievable at random, estimated using their permuted copies, and progressively eliminating irrelevant features to stabilise that test. It uses statistically significance to decide the order of the features.

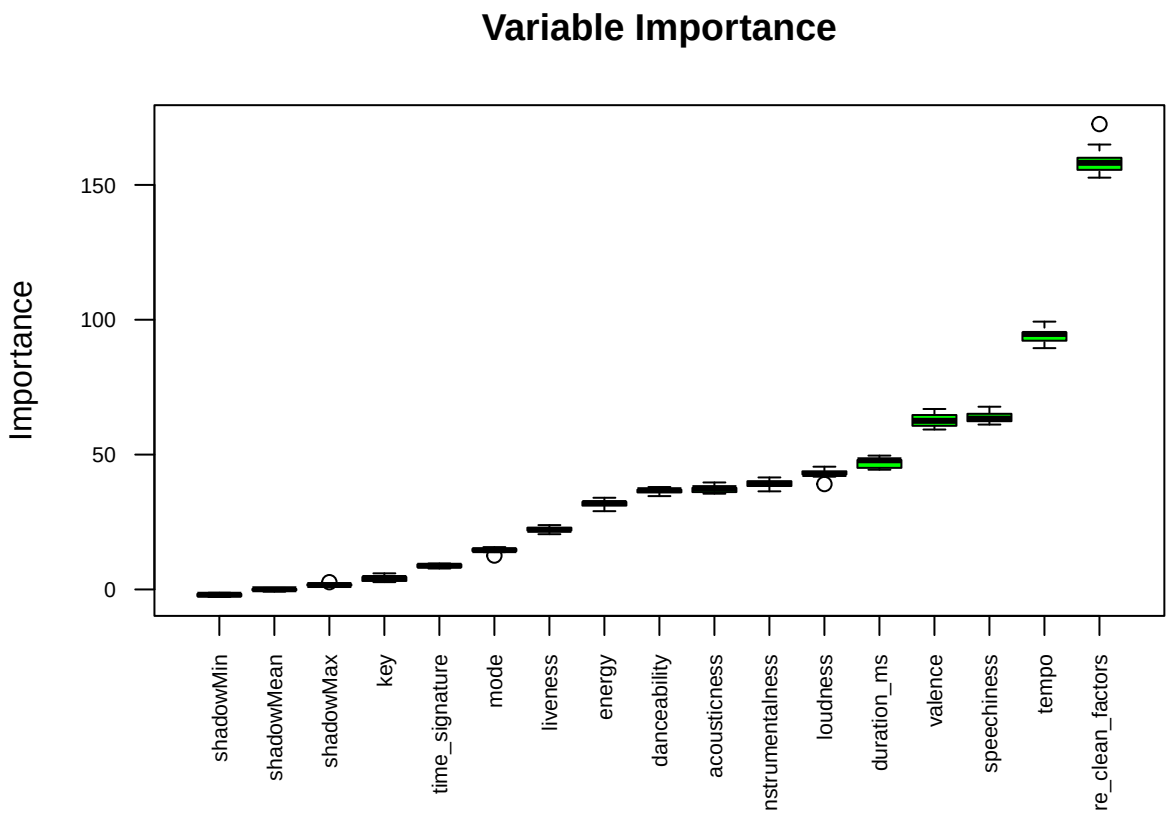
This provided a third set of features to be then subsequently tested in various potential models. The following code was used:

```
#considering the relationship between tempo and duration
# a) Feature Selection

#creating indexes to evaluate features
colnames(training)
features_index_plusclass <- c(3, 5:17,20)

#define test data to check run time before running the full feature selection model
set.seed(75)
test_data <- sample_n(training[,features_index_plusclass],500)

#considering automated feature selection
# test train the model for first model group
set.seed(75)
startTime <- Sys.time()
boruta_music <- Boruta(Is_hit ~ ., data=na.omit(test_data), doTrace=0)
endTime <- Sys.time()
print(endTime - startTime)
print((as.numeric(endTime - startTime)/500)*26650)

#if execution time from test is acceptable run the  model over full data
# obtaining significant variables

set.seed(75)
startTime <- Sys.time()
boruta_music <- Boruta(Is_hit ~ ., data=na.omit(training[features_index_plusclass]), doTrace=0)
endTime <- Sys.time()
print(endTime - startTime)

# Getting significant variables including tentative
```

```
boruta_signif_vars <- getSelectedAttributes(boruta_music, withTentative = TRUE)
roughFixMod_music <- TentativeRoughFix(boruta_music)
boruta_signif_vars <- getSelectedAttributes(roughFixMod_music)
print(boruta_signif_vars)

colnames(training[features_index_plusclass])

# Variable Importance Scores
imps <- attStats(roughFixMod_music)
imps2 = imps[imps$decision != 'Rejected', c('meanImp', 'decision')]
head(imps2[order(-imps2$meanImp), ])  # descending sort
```

The following result was obtained:

```
# Plot variable importance
plot(boruta_music, cex.axis=.7, las=2, xlab="", main="Variable Importance")
```



Data pre-processing

After exploring and visualising the data noted that the following transformations were required:

```
#Pre-processing: encoding genre factors into a numeric with range 0 to 1

#converting gender range from 0 to 1  the distribution
```

```r
preProcess_range_model <- preProcess(data.frame(new_negre =as.numeric(training[,c("genre_clean_factors"
new_genre_encoded <- predict(preProcess_range_model, newdata = data.frame(new_negre= as.numeric(trainin

#appending, checking class and visualising distributions to ensure it remains unaltered

training$new_genre <- new_genre_encoded$new_negre
class(training$new_genre)

par(mfrow=c(1,2))
for(i in 20:21) {
  boxplot(training[,i] ~ training[,3] , main=names(training)[i], las =1, col= c(2,4))

}

##performing same transformation on validation dataset

#converting gender range from 0 to 1  the distribution

preProcess_range_model <- preProcess(data.frame(new_negre =as.numeric(validation[,c("genre_clean_factor
new_genre_encoded <- predict(preProcess_range_model, newdata = data.frame(new_negre= as.numeric(validat

#appending, checking class and visualising distributions

validation$new_genre <- new_genre_encoded$new_negre
class(validation$new_genre)

par(mfrow=c(1,2))
for(i in 20:21) {
  boxplot(validation[,i] ~ validation[,3] , main=names(validation)[i], las =1, col= c(2,4))

}

#removing objects no longer needed
rm(preProcess_range_model, new_genre_encoded )

##Transforming integers to numeric to avoid issues with caret processing
#checking classes
## variables types
sapply(training, class) %>% as.data.frame(.)

#ensuring that transformation does not affect output
data.frame( trans = unique(as.numeric(training$key)), norm = unique(training$key))
data.frame( trans = unique(as.numeric(training$mode)), norm = unique(training$mode))
data.frame( trans = head(as.numeric(training$duration_ms)), norm = head(training$duration_ms))

#transforming integer variables to numeric in training dataset

training$key <- as.numeric(training$key)
training$mode <- as.numeric(training$mode)
training$duration_ms <-as.numeric(training$duration_ms)

#transforming integer variables to numeric in validation dataset
```

```
validation$key <- as.numeric(validation$key)
validation$mode <- as.numeric(validation$mode)
validation$duration_ms <-as.numeric(validation$duration_ms)
```

I selected three sets of potential predictors and created indexes to slice the data, as follows:

- features per song type:

```
#features of songs characteristics (song types)
type_group_index<- c(3, 21,5,6,11,12,13)
```

```
#features of songs characteristics (song types)
sapply(training[,type_group_index], class) %>% as.data.frame(.)
```

```
##                         .
## Is_hit            factor
## new_genre        numeric
## key              numeric
## mode             numeric
## speechiness      numeric
## acousticness     numeric
## instrumentalness numeric
```

- features per song speed:

```
#features of songs speed
speed_group_index <- c(3,7,8, 9,15,16)
```

```
#features of songs speed
sapply(training[,speed_group_index], class) %>% as.data.frame(.)
```

```
##                       .
## Is_hit         factor
## tempo         numeric
## danceability numeric
## energy        numeric
## valence       numeric
## duration_ms   numeric
```

- features from automated selection (top 6 features):

```
#from Automated features selection
AFS_group_index <- c(3,21,7, 11, 15,16,10)
```

```
#from Automated features selection
sapply(training[,AFS_group_index], class) %>% as.data.frame(.)
```

```
##                       .
## Is_hit        factor
```

```
## new_genre   numeric
## tempo       numeric
## speechiness numeric
## valence     numeric
## duration_ms numeric
## loudness    numeric
```

To start with the first round of spot checking of algorithms i defined data subsets of 3000 random records to check processing time and potential performance:

```
#define test data to check run time before running the full feature selection model
set.seed(75)
n_test <- 3000
test_data2 <- sample_n(training[,type_group_index],n_test)

test_data3 <- sample_n(training[,speed_group_index],n_test)

test_data4 <- sample_n(training[,AFS_group_index],n_test)
```

Spot-Check of Algorithms

Given that this a binary classification problem, I have selected an initial group of algorithms that can be used in this kind of tasks, I consulted various sources to get the following definitions:

- **Logistic Regression:** is a classification algorithm used to assign observations to a discrete set of classes. Logistic regression transforms its output using the logistic sigmoid function to return a probability value
- **k-nearest-neighbor (KNN):** is a data classification method for estimating the likelihood that a data point will become a member of one group or another based on how close the data point is to a nearby group
- **Classification And Regression Trees (CART):** is an algorithm that builds a decision tree based on Gini's impurity index as splitting criterion. It builds a binary tree by splitting a node into two child nodes repeatedly
- **Naive Bayes:** is a probabilistic classifier based on Bayes' theorem, which assumes that each feature makes an independent and equal contribution to the probability of a sample belonging to a specific class. It assumes that features do not interact with each other
- **Support Vector Machine (SVM):** is a type of deep learning algorithm that performs supervised learning for classification or regression of data groups. It builds a learning model that assigns new examples to one group or another
- **Random Forest:** is a classification algorithm consisting of many decisions trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree

Using the following code I spot-checked these algorithms with each groups of features previously selected, using only a portion of the data (3000 records):

```
# 10-fold cross validation with 3 repeats
trainControl <- trainControl(method="repeatedcv", number=10, repeats=3, savePredictions = T, classProbs=
metric <- "Accuracy"

# Test options and evaluation metric over small subset of data

# considering algorithms for models based on songs characteristics
# LG
set.seed(75)
startTime <- Sys.time()
typegroup.fit.glm <- train(Is_hit ~ ., data = test_data2,  method="glm", metric=metric, trControl=train
endTime <- Sys.time()
print(endTime - startTime)
print((as.numeric(endTime - startTime)/n_test)*26650)

# KNN
set.seed(75)
startTime <- Sys.time()
typegroup.fit.knn <- train(Is_hit ~ ., data=test_data2, method="knn", metric=metric, trControl=trainCon
endTime <- Sys.time()
print(endTime - startTime)
print((as.numeric(endTime - startTime)/n_test)*26650)

# CART
set.seed(75)
startTime <- Sys.time()
typegroup.fit.cart <- train(Is_hit ~ ., data=test_data2, method="rpart", metric=metric,
                  trControl=trainControl, na.action=na.exclude)
endTime <- Sys.time()
print(endTime - startTime)
print((as.numeric(endTime - startTime)/n_test)*26650)

# Naive Bayes
set.seed(75)
startTime <- Sys.time()
typegroup.fit.nb <- train(Is_hit ~ ., data=test_data2, method="nb", metric=metric, trControl=trainContr
endTime <- Sys.time()
print(endTime - startTime)
print((as.numeric(endTime - startTime)/n_test)*26650)

# SVM
set.seed(75)
startTime <- Sys.time()
typegroup.fit.svm <- train(Is_hit ~ ., data=test_data2, method="svmRadial", metric=metric,
                 trControl=trainControl, na.action=na.exclude)
endTime <- Sys.time()
print(endTime - startTime)
print((as.numeric(endTime - startTime)/n_test)*26650)

# RF
set.seed(75)
startTime <- Sys.time()
typegroup.fit.RF <- train(Is_hit ~ ., data=test_data2, method="rf", ntree=5, metric=metric,
```

```
                trControl=trainControl, na.action=na.exclude)
endTime <- Sys.time()
print(endTime - startTime)
print((as.numeric(endTime - startTime)/n_test)*26650)

# considering algorithms for models based on songs speed
# LG
set.seed(75)
startTime <- Sys.time()
speedgroup.fit.glm <- train(Is_hit ~ ., data = test_data3,  method="glm", metric=metric, trControl=trai
endTime <- Sys.time()
print(endTime - startTime)
print((as.numeric(endTime - startTime)/n_test)*26650)

# KNN
set.seed(75)
startTime <- Sys.time()
speedgroup.fit.knn <- train(Is_hit ~ ., data=test_data3, method="knn", metric=metric, trControl=trainCo
endTime <- Sys.time()
print(endTime - startTime)
print((as.numeric(endTime - startTime)/n_test)*26650)

# CART
set.seed(75)
startTime <- Sys.time()
speedgroup.fit.cart <- train(Is_hit ~ ., data=test_data3, method="rpart", metric=metric,
                             trControl=trainControl, na.action=na.exclude)
endTime <- Sys.time()
print(endTime - startTime)
print((as.numeric(endTime - startTime)/n_test)*26650)

# Naive Bayes
set.seed(75)
startTime <- Sys.time()
speedgroup.fit.nb <- train(Is_hit ~ ., data=test_data3, method="nb", metric=metric, trControl=trainCont
endTime <- Sys.time()
print(endTime - startTime)
print((as.numeric(endTime - startTime)/n_test)*26650)

# SVM
set.seed(75)
startTime <- Sys.time()
speedgroup.fit.svm <- train(Is_hit ~ ., data=test_data3, method="svmRadial", metric=metric,
                            trControl=trainControl, na.action=na.exclude)
endTime <- Sys.time()
print(endTime - startTime)
print((as.numeric(endTime - startTime)/n_test)*26650)

# RF
set.seed(75)
startTime <- Sys.time()
speedgroup.fit.RF <- train(Is_hit ~ ., data=test_data3, method="rf", ntree=5, metric=metric,
                           trControl=trainControl, na.action=na.exclude)
```

```
endTime <- Sys.time()
print(endTime - startTime)
print((as.numeric(endTime - startTime)/n_test)*26650)

# considering algorithms for models based on automated features selection

# LG  (with preprocess set to center to normalise the predictors as its beneficial for Logistic regress
set.seed(75)
startTime <- Sys.time()
AFSgroup.fit.glm <- train(Is_hit ~ ., data = test_data4,  method="glm", metric=metric, trControl=trainC
endTime <- Sys.time()
print(endTime - startTime)
print((as.numeric(endTime - startTime)/n_test)*26650)

# KNN
set.seed(75)
startTime <- Sys.time()
AFSgroup.fit.knn <- train(Is_hit ~ ., data=test_data4, method="knn", metric=metric, trControl=trainCont
endTime <- Sys.time()
print(endTime - startTime)
print((as.numeric(endTime - startTime)/n_test)*26650)

# CART
set.seed(75)
startTime <- Sys.time()
AFSgroup.fit.cart <- train(Is_hit ~ ., data=test_data4, method="rpart", metric=metric,
                           trControl=trainControl, na.action=na.exclude)
endTime <- Sys.time()
print(endTime - startTime)
print((as.numeric(endTime - startTime)/n_test)*26650)

# Naive Bayes
set.seed(75)
startTime <- Sys.time()
AFSgroup.fit.nb <- train(Is_hit ~ ., data=test_data4, method="nb", metric=metric, trControl=trainControl
endTime <- Sys.time()
print(endTime - startTime)
print((as.numeric(endTime - startTime)/n_test)*26650)

# SVM
set.seed(75)
startTime <- Sys.time()
AFSgroup.fit.svm <- train(Is_hit ~ ., data=test_data4, method="svmRadial", metric=metric,
                          trControl=trainControl, na.action=na.exclude)
endTime <- Sys.time()
print(endTime - startTime)
print((as.numeric(endTime - startTime)/n_test)*26650)

# RF
set.seed(75)
startTime <- Sys.time()
AFSgroup.fit.RF <- train(Is_hit ~ ., data=test_data4, method="rf", ntree=5, metric=metric,
                         trControl=trainControl, na.action=na.exclude)
```

```
endTime <- Sys.time()
print(endTime - startTime)
print((as.numeric(endTime - startTime)/n_test)*26650)

# Compare algorithms
results <- resamples(list(LG_type=typegroup.fit.glm, LG_speed=speedgroup.fit.glm, LG_AFS=AFSgroup.fit.gl
                          KNN_type=typegroup.fit.knn, KNN_speed=speedgroup.fit.knn, KNN_AFS=AFSgroup.fi
                          CART_type=typegroup.fit.cart, CART_speed=speedgroup.fit.cart, CART_AFS=AFSgrou
                          NB_type=typegroup.fit.nb, NB_speed=speedgroup.fit.nb, NB_AFS=AFSgroup.fit.nb,
                          SVM_type=typegroup.fit.svm, SVM_speed=speedgroup.fit.svm, SVM_AFS=AFSgroup.fi
                          RF_type=typegroup.fit.RF, RF_speed=speedgroup.fit.RF, RF_AFS=AFSgroup.fit.RF))
```

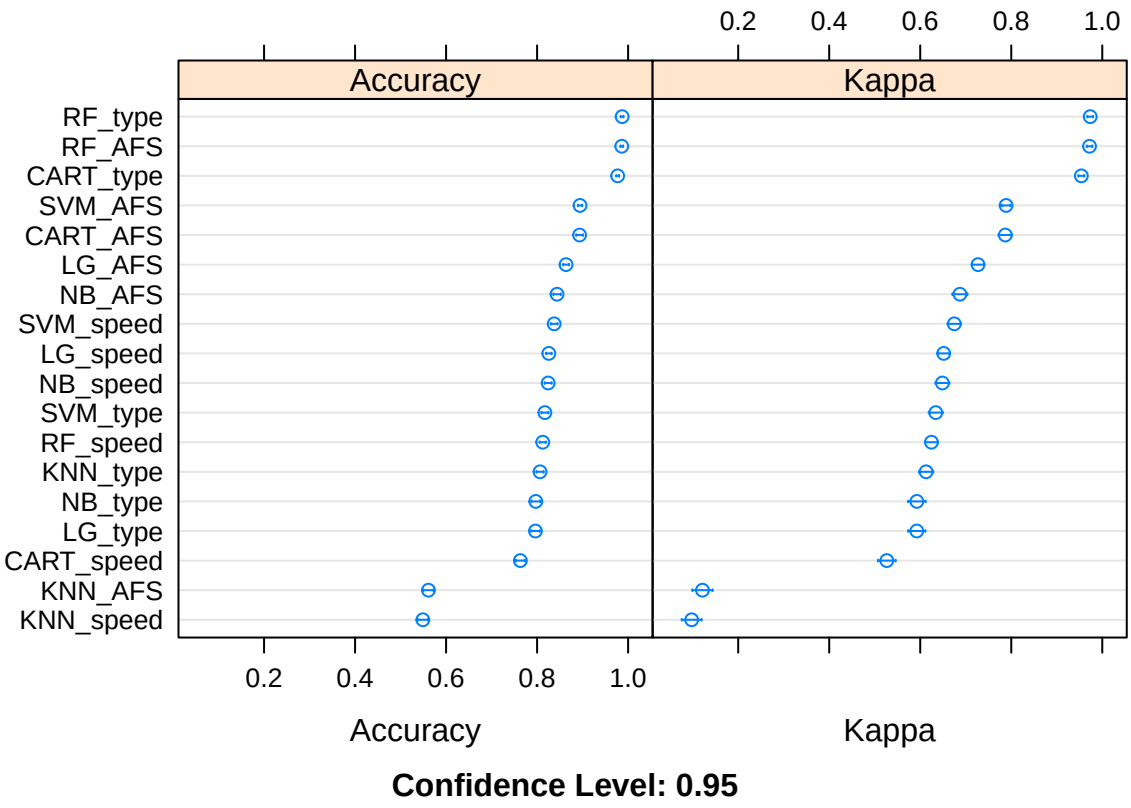
The following results were noted:

```
#spot check of models
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: LG_type, LG_speed, LG_AFS, KNN_type, KNN_speed, KNN_AFS, CART_type, CART_speed, CART_AFS, NB
## Number of resamples: 30
##
## Accuracy
##                  Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## LG_type     0.7433333 0.7750000 0.8033333 0.7963333 0.8100000 0.8433333    0
## LG_speed    0.7826087 0.8166667 0.8277592 0.8257682 0.8366667 0.8566667    0
## LG_AFS      0.8300000 0.8487154 0.8666667 0.8635695 0.8704319 0.9000000    0
## KNN_type    0.7700000 0.7900000 0.8116667 0.8064444 0.8200000 0.8400000    0
## KNN_speed   0.4866667 0.5270598 0.5492475 0.5488954 0.5694048 0.6066667    0
## KNN_AFS     0.5016722 0.5453045 0.5592642 0.5612122 0.5828488 0.6066667    0
## CART_type   0.9566667 0.9733333 0.9766667 0.9770000 0.9800000 0.9933333    0
## CART_speed  0.7100000 0.7500000 0.7700000 0.7632017 0.7826827 0.8073090    0
## CART_AFS    0.8361204 0.8782358 0.8935105 0.8933197 0.9066667 0.9297659    0
## NB_type     0.7500000 0.7808333 0.7983333 0.7971111 0.8191667 0.8366667    0
## NB_speed    0.7600000 0.8133333 0.8269546 0.8243230 0.8370736 0.8566667    0
## NB_AFS      0.7940199 0.8269231 0.8452547 0.8437816 0.8590468 0.8933333    0
## SVM_type    0.7800000 0.8025000 0.8166667 0.8171111 0.8291667 0.8600000    0
## SVM_speed   0.7792642 0.8300000 0.8369380 0.8374279 0.8525000 0.8666667    0
## SVM_AFS     0.8662207 0.8829431 0.8966667 0.8943279 0.9035742 0.9130435    0
## RF_type     0.9533333 0.9833333 0.9866667 0.9867778 0.9900000 1.0000000    0
## RF_speed    0.7692308 0.8008333 0.8133313 0.8123278 0.8250000 0.8466667    0
## RF_AFS      0.9667774 0.9808692 0.9866444 0.9860073 0.9900000 1.0000000    0
##
## Kappa
##                    Min.    1st Qu.     Median       Mean   3rd Qu.      Max.
## LG_type     0.485981308 0.55017422 0.60650893 0.59239506 0.6198733 0.6869171
## LG_speed    0.565163889 0.63323552 0.65549561 0.65154959 0.6733551 0.7133333
## LG_AFS      0.659788313 0.69727774 0.73329747 0.72703271 0.7407636 0.7998221
## KNN_type    0.540122634 0.57988761 0.62321034 0.61289776 0.6401200 0.6798719
## KNN_speed  -0.027077498 0.05409617 0.09841922 0.09770447 0.1387703 0.2132984
## KNN_AFS     0.001613518 0.09061395 0.11681164 0.12147830 0.1655194 0.2127735
```

```
## CART_type   0.913448735 0.94665955 0.95332089 0.95400864 0.9600160 0.9866649
## CART_speed  0.420000000 0.50015536 0.54000000 0.52650514 0.5653015 0.6146307
## CART_AFS    0.673936750 0.75685468 0.78752761 0.78696138 0.8136791 0.8594768
## NB_type     0.497184232 0.55980484 0.59496700 0.59259059 0.6372809 0.6718457
## NB_speed    0.520000000 0.62655462 0.65390728 0.64867624 0.6742080 0.7133333
## NB_AFS      0.587014252 0.65337214 0.69025649 0.68715733 0.7178532 0.7864389
## SVM_type    0.559706471 0.60506532 0.63308872 0.63412856 0.6586871 0.7199627
## SVM_speed   0.558444464 0.66000000 0.67397202 0.67485924 0.7049315 0.7333333
## SVM_AFS     0.732725485 0.76601715 0.79341596 0.78870065 0.8073139 0.8261161
## RF_type     0.906703985 0.96666667 0.97333689 0.97355579 0.9800000 1.0000000
## RF_speed    0.538301443 0.60159952 0.62665804 0.62463626 0.6500000 0.6934287
## RF_AFS      0.933580476 0.96174014 0.97328565 0.97201087 0.9799947 1.0000000
##            NA's
## LG_type       0
## LG_speed      0
## LG_AFS        0
## KNN_type      0
## KNN_speed     0
## KNN_AFS       0
## CART_type     0
## CART_speed    0
## CART_AFS      0
## NB_type       0
## NB_speed      0
## NB_AFS        0
## SVM_type      0
## SVM_speed     0
## SVM_AFS       0
## RF_type       0
## RF_speed      0
## RF_AFS        0
```

```
dotplot(results)
```



Confidence Level: 0.95

Evaluating Selected Algorithms

Based on the above cross-validated results over the subset data, and considering processing time, I have selected the following models for consideration over the full dataset:

```
# Checking performance with full training dataset to select best models

# RF type
set.seed(75)
startTime <- Sys.time()
typegroup.fit.RF2 <- train(Is_hit ~ ., data=training[,type_group_index], method="rf", ntree=5, metric=me
                           trControl=trainControl, na.action=na.exclude)
endTime <- Sys.time()
print(endTime - startTime)


# RF AFS
set.seed(75)
startTime <- Sys.time()
AFSgroup.fit.RF2 <- train(Is_hit ~ ., data=training[,AFS_group_index], method="rf", ntree=5, metric=met
                          trControl=trainControl, na.action=na.exclude, preProcess= "center")
endTime <- Sys.time()
print(endTime - startTime)

# CART Type
```

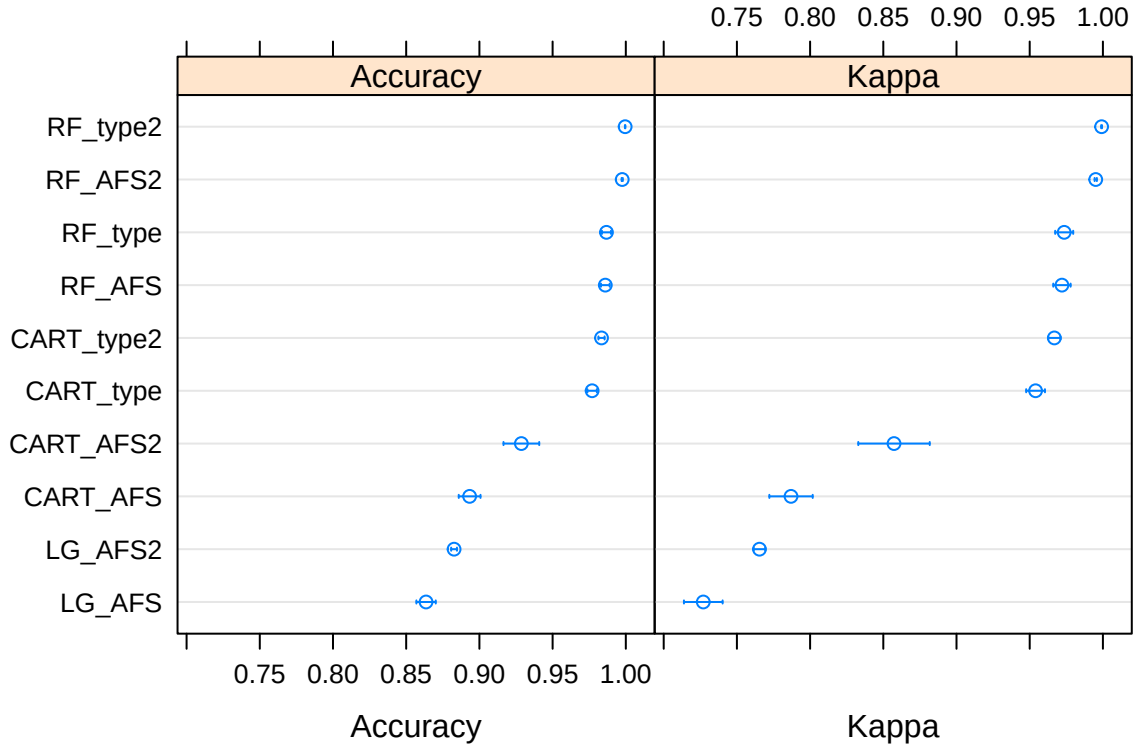
```
set.seed(75)
startTime <- Sys.time()
typegroup.fit.cart2 <- train(Is_hit ~ ., data=training[,type_group_index], method="rpart", metric=metri
                             trControl=trainControl, na.action=na.exclude, preProcess= "center")
endTime <- Sys.time()
print(endTime - startTime)

#SVM_AFS excluded due to long processing time

# CART AFS
set.seed(75)
startTime <- Sys.time()
AFSgroup.fit.cart2 <- train(Is_hit ~ ., data=training[,AFS_group_index], method="rpart", metric=metric,
                             trControl=trainControl, na.action=na.exclude, preProcess= "center")
endTime <- Sys.time()
print(endTime - startTime)

# LG  AFS
set.seed(75)
startTime <- Sys.time()
AFSgroup.fit.glm2 <- train(Is_hit ~ ., data = training[,AFS_group_index],  method="glm", metric=metric,
endTime <- Sys.time()
print(endTime - startTime)

# Compare first selection of algorithms
results_fisrt_select <- resamples(list(RF_type=typegroup.fit.RF,RF_AFS=AFSgroup.fit.RF,
                                       CART_type=typegroup.fit.cart, CART_AFS=AFSgroup.fit.cart,
                                       LG_AFS=AFSgroup.fit.glm, RF_type2=typegroup.fit.RF2,RF_AFS2=AFSg
                                       CART_type2=typegroup.fit.cart2, CART_AFS2=AFSgroup.fit.cart2,
                                       LG_AFS2=AFSgroup.fit.glm2))
```

```
#spot check of models
summary(results_fisrt_select)
```

```
##
## Call:
## summary.resamples(object = results_fisrt_select)
##
## Models: RF_type, RF_AFS, CART_type, CART_AFS, LG_AFS, RF_type2, RF_AFS2, CART_type2, CART_AFS2, LG_A
## Number of resamples: 30
##
## Accuracy
##                 Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## RF_type    0.9533333 0.9833333 0.9866667 0.9867778 0.9900000 1.0000000    0
## RF_AFS     0.9667774 0.9808692 0.9866444 0.9860073 0.9900000 1.0000000    0
## CART_type  0.9566667 0.9733333 0.9766667 0.9770000 0.9800000 0.9933333    0
## CART_AFS   0.8361204 0.8782358 0.8935105 0.8933197 0.9066667 0.9297659    0
## LG_AFS     0.8300000 0.8487154 0.8666667 0.8635695 0.8704319 0.9000000    0
## RF_type2   0.9988743 0.9992496 0.9996248 0.9995497 0.9999062 1.0000000    0
## RF_AFS2    0.9943715 0.9973724 0.9977482 0.9975485 0.9981244 0.9992495    0
## CART_type2 0.9741088 0.9787115 0.9834897 0.9834147 0.9886502 0.9909944    0
## CART_AFS2  0.8912228 0.8983766 0.9078970 0.9286556 0.9624765 0.9677298    0
## LG_AFS2    0.8716698 0.8803940 0.8835334 0.8827018 0.8864809 0.8927232    0
```

```
##
## Kappa
##                  Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## RF_type    0.9067040 0.9666667 0.9733369 0.9735558 0.9800000 1.0000000    0
## RF_AFS     0.9335805 0.9617401 0.9732856 0.9720109 0.9799947 1.0000000    0
## CART_type  0.9134487 0.9466596 0.9533209 0.9540086 0.9600160 0.9866649    0
## CART_AFS   0.6739368 0.7568547 0.7875276 0.7869614 0.8136791 0.8594768    0
## LG_AFS     0.6597883 0.6972777 0.7332975 0.7270327 0.7407636 0.7998221    0
## RF_type2   0.9977486 0.9984992 0.9992495 0.9990995 0.9998125 1.0000000    0
## RF_AFS2    0.9887430 0.9947447 0.9954963 0.9950969 0.9962487 0.9984991    0
## CART_type2 0.9482175 0.9574229 0.9669794 0.9668294 0.9773003 0.9819886    0
## CART_AFS2  0.7824456 0.7967562 0.8157885 0.8573129 0.9249527 0.9354593    0
## LG_AFS2    0.7433423 0.7607897 0.7670668 0.7654039 0.7729623 0.7854464    0
```

```
dotplot(results_fisrt_select)
```



I evaluated the performance of these models over the validation dataset to check if there was any tendency to overfit. I defined a function to expedite the calculation of the confusion matrix and obtain a summary of the results:

```
###Checking for possible overfitting of the models

#defining a function to check performance with validation dataset
validation_function <- function(x,y) {
  predict_test <-predict(x, newdata=y)
  a <- confusionMatrix(predict_test,y$Is_hit)
```

```
  b<- data.frame(row.names = "results", acc = a$overall['Accuracy'],
                 pval= a$overall['AccuracyPValue'],
                 sen= a$byClass['Sensitivity'],
                 spe = a$byClass['Specificity'],
                 f1= a$byClass['F1'])
  return(b)
}

#summarising results

selected_models <- list(RF_type2=typegroup.fit.RF2,RF_AFS2=AFSgroup.fit.RF2,
                        CART_type2=typegroup.fit.cart2, CART_AFS2=AFSgroup.fit.cart2,
                        LG_AFS2=AFSgroup.fit.glm2)
```

The following results were noted:

```
#checking for overfitting:
sapply(selected_models, validation_function, y = validation)
```

```
##      RF_type2     RF_AFS2      CART_type2 CART_AFS2 LG_AFS2
## acc  0.5954668    0.6154308    0.7997598  0.8980787 0.8818673
## pval 2.676074e-55 4.222582e-80 0          0         0
## sen  0.2080456    0.2482738    0.6346443  0.981387  0.8916241
## spe  0.982888     0.9825878    0.9648754  0.8147703 0.8721105
## f1   0.3396226    0.392315     0.7601582  0.9059166 0.8830088
```

I noted that the Random Forest models were significantly over-fitted and not performing well for sensitivity. Similarly the Cart for the type-group features was showing signals of overfitting and lower sensitivity than specificity. Therefore I discarded these models.

I selected the following models as they had higher accuracy and performed well in both sensitivity and specificity:

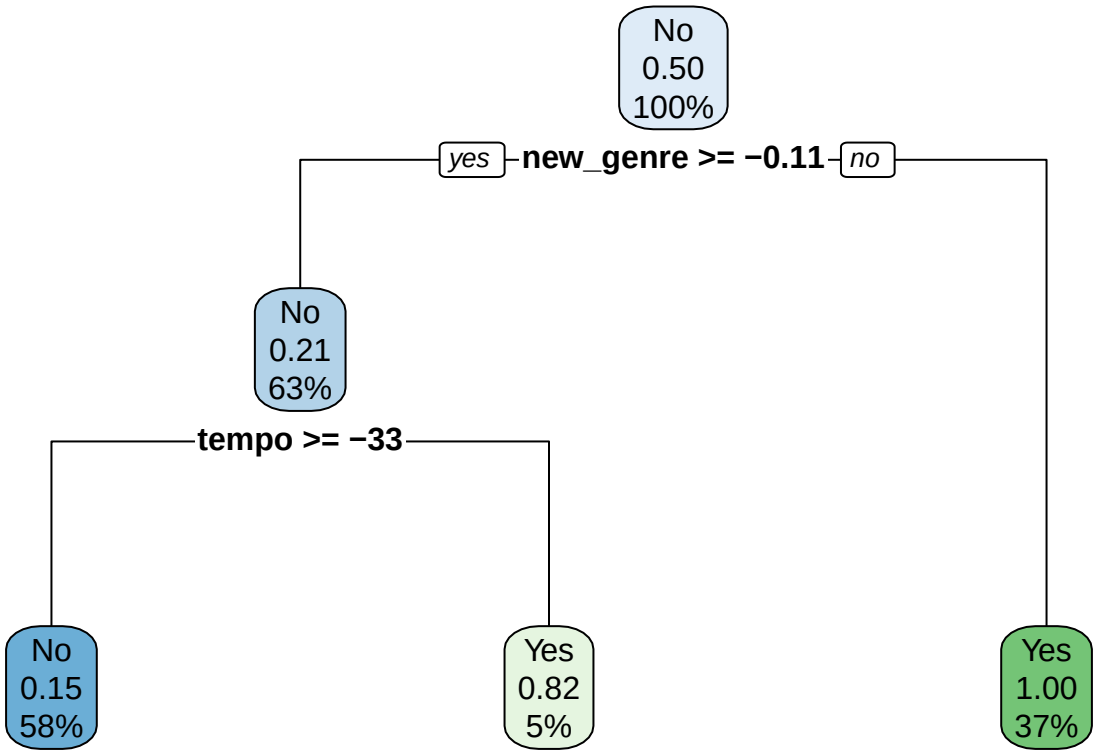
- CART over automated selected features group.
- Logistic Regression over automated selected features group.

Interpretation of individual final models selected

CART

The CART model created a binary tree by using two features: genre and tempo, as noted below:

```
#plotting the tree produced by the Cart model:
#AFS.CART
rpart.plot(AFSgroup.fit.cart2$finalModel)
```



Interpretation has been a bit obscured due to data transformation. We applied "Preprocess = center" to the data in the processing of the models. This subtracts the mean of the predictor's data from each data point. We have also transformed the categorical variable "genre" to a numerical vector for ease of use in the model.

The tree starts indicating that classes are balanced ("No" represents 50% of the classes)

The model performs a cutoff at certain genres and divides the population between hit (about 37% of the songs are classified as hits based on genre grouping) and the remaining 63% is further evaluated using the tempo feature.

For tempo we know that the mean is 136.5:

```
#Tempo mean
mean(training$tempo)
```

```
## [1] 136.5239
```

Looking at the tree we can see that for this training dataset any songs that was not classified as a hit due to genre, and had a tempo above 103.5 (this is: -0.33 + 136.5), was classified as a hit, and represented 5% of the songs. The remaining group was classified as non-hit (58%).

LOGISTIC REGRESSION

Logistic regression models are based on the logistic function:

$$\frac{1}{(1+e^{-value})}$$

Where:

- e represents the base of the natural logarithms
- value is the numerical value that we want to transform.

This can also be represented this way:

$$g(p) = log(\frac{p}{1-p})$$

Our Logistic regression model has six predictor and can be represented as follows:

$$g\langle p(x1, x2...x6)\rangle = g\langle Pr(Y = 1|X1 = x1, X2 = x2, \cdots, X6 = x6)\rangle = \beta 0 + \beta 1x1 + \beta 2x2 + \cdots + \beta 6x6$$

Another way to look at it is:

$$y = \frac{e^{(\beta 0+\beta 1*X1+\beta 2*x2+\cdots+\beta 6*x6)}}{(1+e^{(\beta 0+\beta 1*X+\beta 2*x2+\cdots+\beta 6*x6)})}$$

Where:

- x1 to x6 are the six predictors placed in our model.

The Logistic Regression model summary provides a detail of the coefficients calculated for each variable:

```
#AFS.GLM
summary(AFSgroup.fit.glm2)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.3342  -0.3626  -0.0015   0.2905   3.4152
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  7.508e-02  2.187e-02   3.434 0.000596 ***
## new_genre   -6.327e+00  1.009e-01 -62.686  < 2e-16 ***
## tempo       -3.779e-02  7.841e-04 -48.189  < 2e-16 ***
## speechiness -4.509e+00  2.032e-01 -22.187  < 2e-16 ***
## valence      5.145e+00  9.865e-02  52.158  < 2e-16 ***
## duration_ms  9.322e-06  3.727e-07  25.014  < 2e-16 ***
## loudness    -9.295e-02  6.702e-03 -13.868  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 36945  on 26649  degrees of freedom
## Residual deviance: 14637  on 26643  degrees of freedom
## AIC: 14651
##
## Number of Fisher Scoring iterations: 6
```

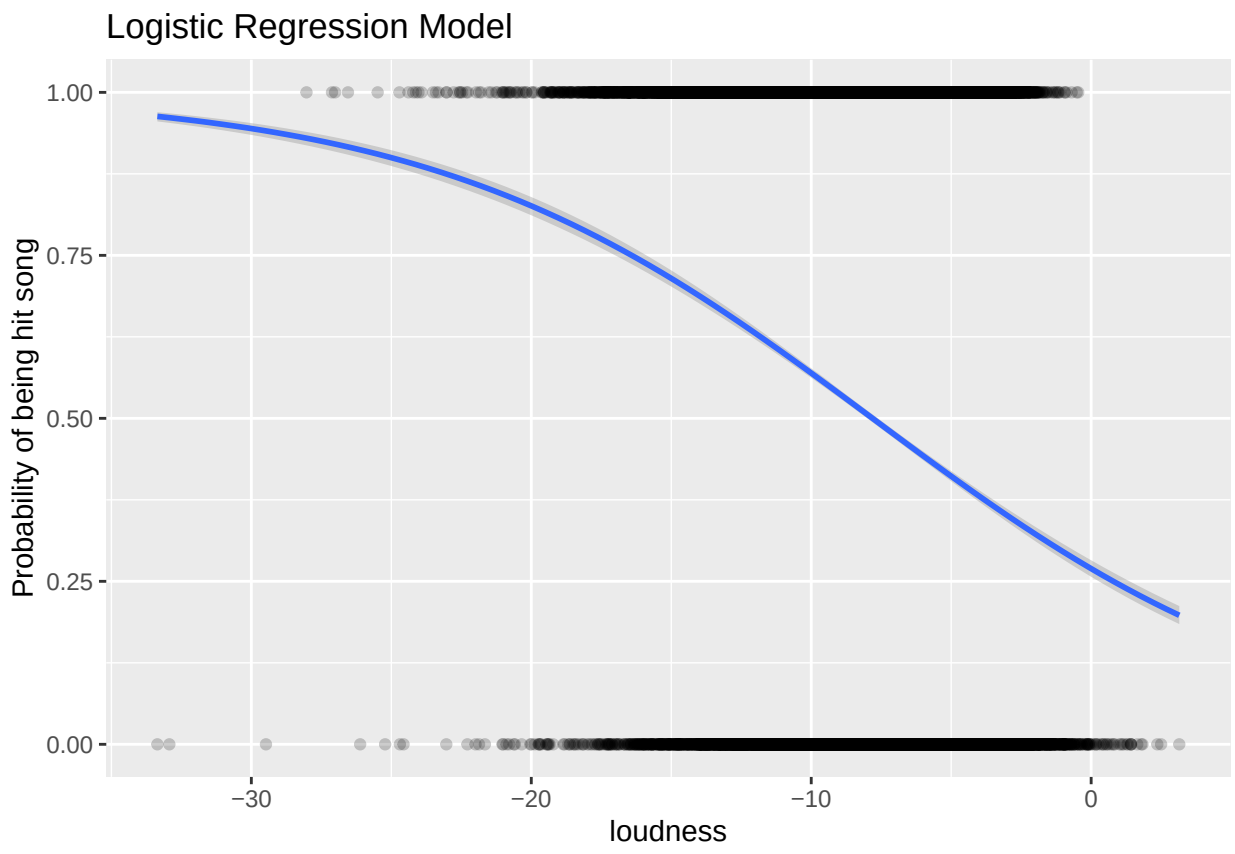
Please note that p-values are very low, indicating that there is relationship between the target variable ("Is_hit") and the predictors (i.e. rejecting the null hypothesis that changes in the predictors have no effect in the target variable).

In general, negative coefficients will indicate a decrease in the probability of the song being classified as a hit, when there is an increase in the predictor's value.

For example, an increase in loudness will lower the probability of the song being a hit more than the other predictors with negative coefficients:

```
#AFS.GLM
#probability curve for loudness example
training[,AFS_group_index]  %>%
  mutate(prob = ifelse(Is_hit == "Yes", 1, 0)) %>%
  ggplot(aes(loudness, prob)) +
  geom_point(alpha = 0.2) +
  geom_smooth(method = "glm", method.args = list(family = "binomial")) +
  labs(
    title = "Logistic Regression Model",
    x = "loudness",
    y = "Probability of being hit song"
  )
```

```
## `geom_smooth()` using formula 'y ~ x'
```

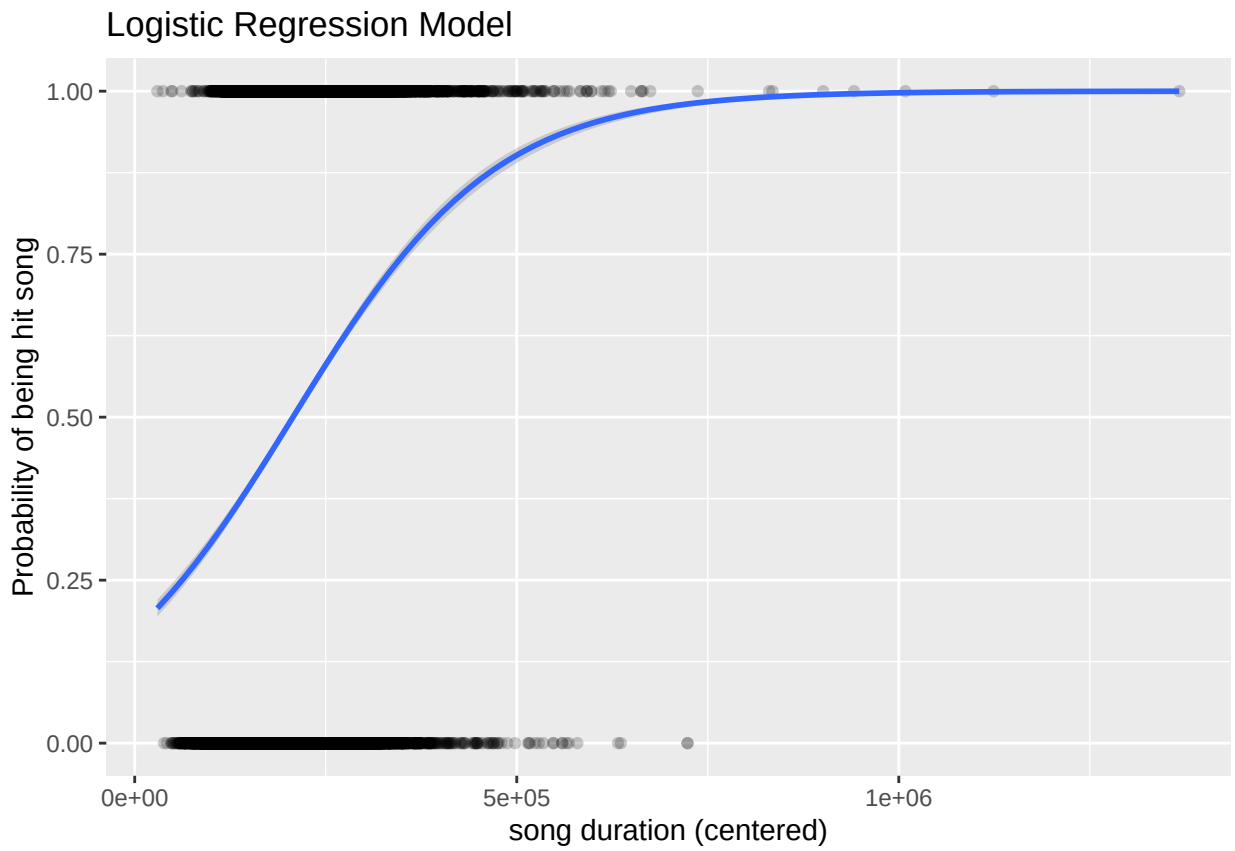


Positive coefficients indicate an increase in the probability of the song being classified as a hit, when there is an increase in the predictor's value.

An example is the duration feature. An increase on duration tends to increase the probability of the song being a hit, this need to be read carefully with the standard deviation as a sign of variability:

```
#AFS.GLM
#probability curve for duration example
training[,AFS_group_index]  %>%
  mutate(prob = ifelse(Is_hit == "Yes", 1, 0)) %>%
  ggplot(aes(duration_ms, prob)) +
  geom_point(alpha = 0.2) +
  geom_smooth(method = "glm", method.args = list(family = "binomial")) +
  labs(
    title = "Logistic Regression Model",
    x = "song duration (centered)",
    y = "Probability of being hit song"
  )
```

```
## `geom_smooth()` using formula 'y ~ x'
```



In our exploratory analysis we noted that hit songs had a slightly higher duration than non-hit songs, and the classes seemed to have a normal distribution. Therefore, songs that are too long or too shorts are unlikely to be classified as hit under this model.

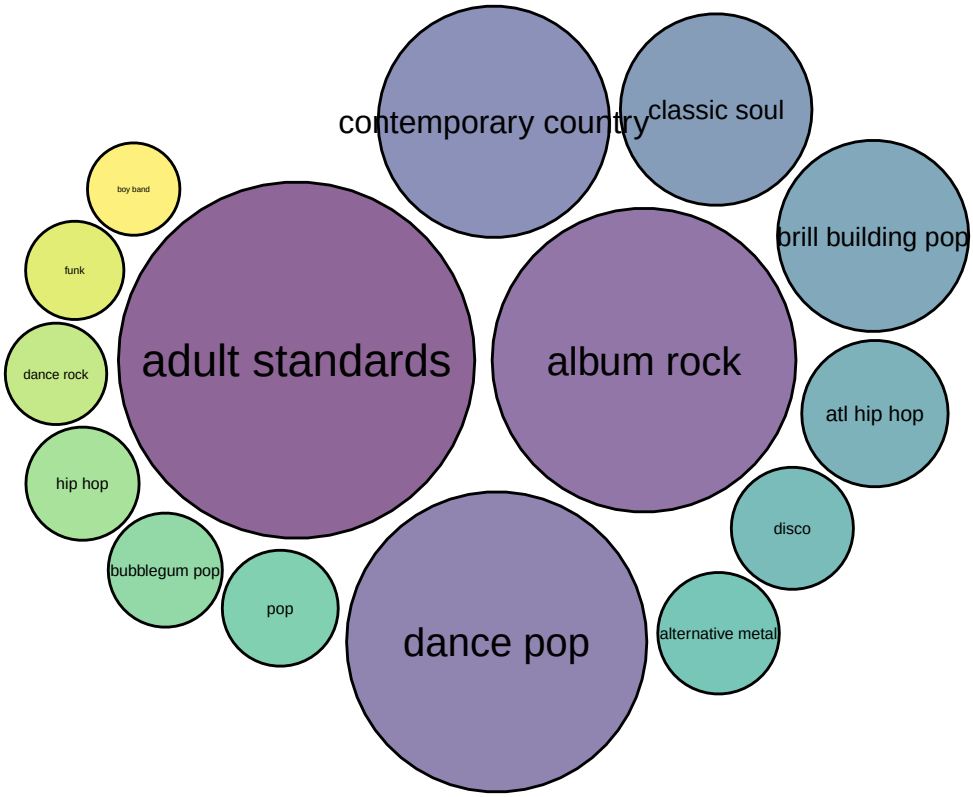
To aid on the understanding of genres, due to the data transformation, I decided to illustrate the probabilities of certain genres being associated with hit songs, by visualising the top 10 genres by class (in terms of number of songs), as follows:

- grouping data and creating the layouts

```
###expanding on genres interpretation

### visualization of Top15 genres by class
Top_15_genre_hit <- by_genre_count %>% filter(Is_hit == 1) %>% arrange(desc(total_songs)) %>% head(15)

Top_15_genre_non_hit <- by_genre_count %>% filter(Is_hit == 0) %>% arrange(desc(total_songs)) %>% head(


# Generate the layout for Top_15_genre_hit
packing <- circleProgressiveLayout(Top_15_genre_hit$total_songs, sizetype='area')
packing$radius <- 0.95*packing$radius
Top_15_genre_hit_pack <- cbind(Top_15_genre_hit, packing)
Top_15_genre_hit_pack.gg <- circleLayoutVertices(packing, npoints=50)

# Generate the layout for Top_15_genre_non_hit
packing_nh <- circleProgressiveLayout(Top_15_genre_non_hit$total_songs, sizetype='area')
packing_nh$radius <- 0.95*packing$radius
Top_15_genre_non_hit_pack <- cbind(Top_15_genre_non_hit, packing_nh)
Top_15_genre_non_hit_pack.gg <- circleLayoutVertices(packing_nh, npoints=50)
```

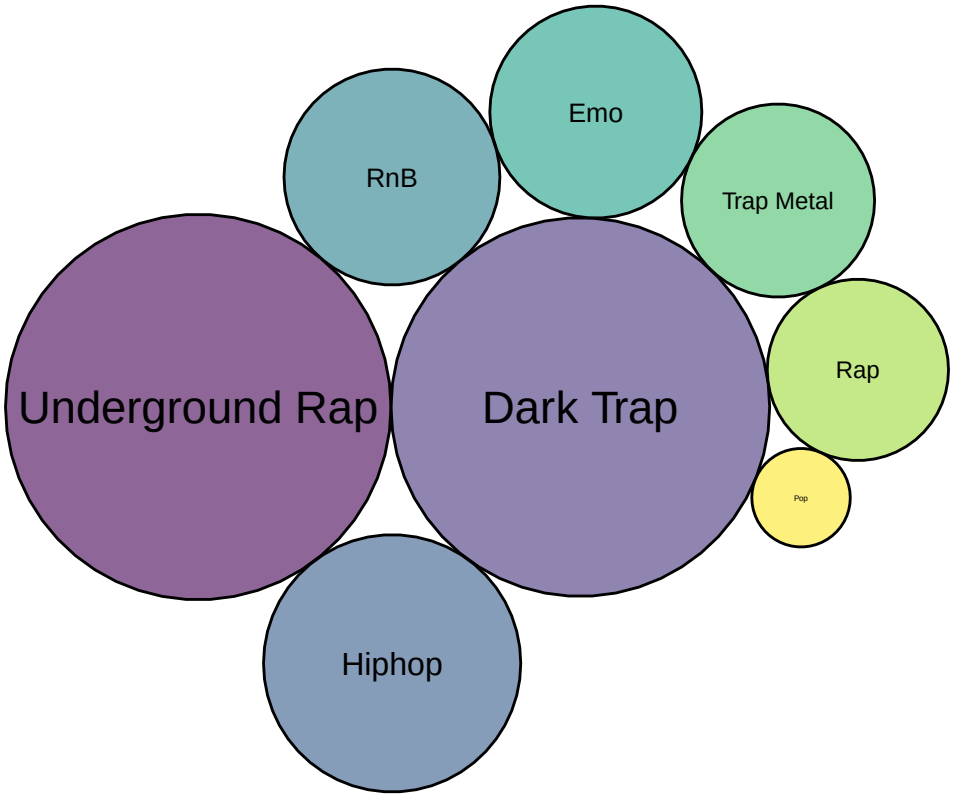
*displaying the Top-10 genres by hit songs:

```
# Plot Top_15_genre_hit
ggplot() +
  geom_polygon(data = Top_15_genre_hit_pack.gg, aes(x, y, group = id, fill=id), colour = "black", alpha
  scale_fill_viridis() +
  geom_text(data = Top_15_genre_hit_pack, aes(x, y, size=total_songs, label = genre_clean_factors), col
  theme_void() +
  theme(legend.position="none")+
  coord_equal()
```



*displaying the Top-10 genres by non-hit songs:

```
# Plot Top_15_genre_hit
ggplot() +
  geom_polygon(data = Top_15_genre_non_hit_pack.gg, aes(x, y, group = id, fill=id), colour = "black", al
  scale_fill_viridis() +
  geom_text(data = Top_15_genre_non_hit_pack, aes(x, y, size=total_songs, label = genre_clean_factors),
  theme_void() +
  theme(legend.position="none")+
  coord_equal()
```



Final model ensemble and evaluation

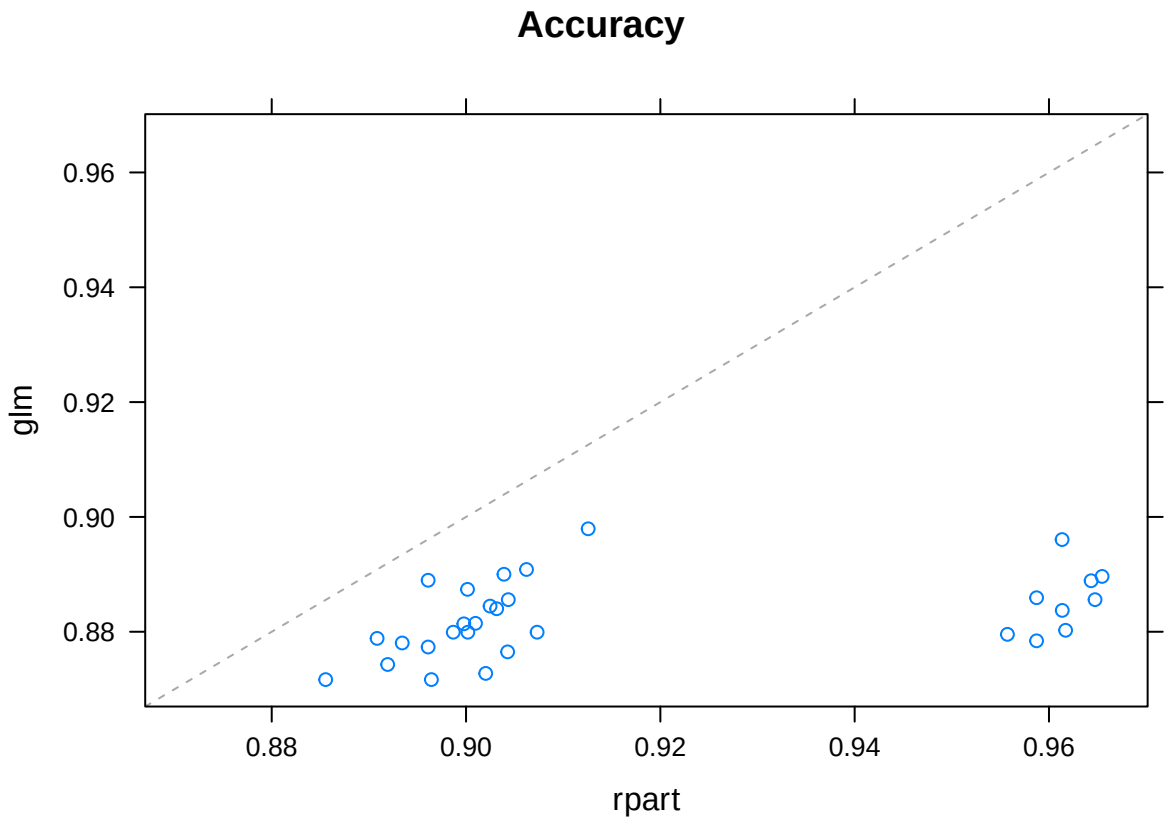
Given that both models performed similarly, I explored an ensemble of these models to obtain an improved accuracy, while maintaining reasonable performance in sensitivity and specificity.

I listed the two models to ensemble with the following code:

```
#ensemble of two models selected

model_list <- caretList(
  Is_hit~., data=training[,AFS_group_index],
  trControl=trainControl,
  methodList=c("glm", "rpart")
)
```

and checked whether models were correlated. If the models are correlated it means they are predicting mistakes in the same way, therefore it won't be useful to ensemble them. However, the following code shows that the models have low correlation:

```
#checking  models correlation
xyplot(resamples(model_list))
```




```
modelCor(resamples(model_list))
```

```
##             glm     rpart
## glm   1.0000000 0.3705314
## rpart 0.3705314 1.0000000
```

The following code allowed me to ensemble the models:

```
#ensemble
ensemble_ctrl <- trainControl(method = "repeatedcv", number = 10, repeats=3, returnResamp = "final",
                              savePredictions = "final",  classProbs = TRUE)

final_model <- caretEnsemble(
  model_list,
  metric="Accuracy",
  trControl= ensemble_ctrl)
```

The new model produce a higher level of accuracy than the individual models:

```
#final model
summary(final_model)
```

```
## The following models were ensembled: glm, rpart
## They were weighted:
```

```
## 5.4888 -4.2845 -5.264
## The resulting Accuracy is: 0.9304
## The fit for each individual model on the Accuracy is:
##  method  Accuracy AccuracySD
##     glm 0.8827016 0.00666575
##   rpart 0.9182874 0.02918040
```

The following code demonstrates how the model assigns importance levels to each of the predictors:

```
#importance of predictors
final_model_imp<- varImp(final_model)

final_model_imp[order(final_model_imp$overall,decreasing=TRUE),]
```

```
##               overall       glm     rpart
## new_genre   42.561698 34.649207 49.001771
## tempo       22.634385 24.359242 21.230503
## valence     17.182572 27.176409  9.048465
## speechiness 11.656772  5.904258 16.338814
## duration_ms  5.964573  7.910884  4.380447
## loudness     0.000000  0.000000  0.000000
```

Finally, I evaluated the model performance using the validation dataset and compared it with the performance of the individual models. We can see that the accuracy level has increased, while maintaining an acceptable level of performance with sensitivity and accuracy.

```
#evaluating models with validation dataset

#final model
validation_function(final_model ,validation)
```

```
##               acc pval       sen       spe       f1
## results 0.9187932    0 0.9510657 0.8865206 0.921332
```

```
#individual models
validation_function(AFSgroup.fit.cart2 ,validation)
```

```
##               acc pval      sen       spe        f1
## results 0.8980787    0 0.981387 0.8147703 0.9059166
```

```
validation_function(AFSgroup.fit.glm2 ,validation)
```

```
##               acc pval       sen       spe        f1
## results 0.8818673    0 0.8916241 0.8721105 0.8830088
```

CONCLUSION AND LIMITATIONS

The project aim was to predict a hit song based on its audio feature. This was a binary classification problem, with a base probability of 50% (affected by prevalence of the classes).

The analysis performed indicates that popular songs, which we call “hit songs” under the term of this project, share certain characteristics that allow for distinctions with non-hit songs.

The model indicates that the most relevant audio features to predict hit songs are:

- genre
- tempo

- valence
- speechiness
- duration_ms
- loudness

I was able to construct an model that provides an overall accuracy of approx. 91% while maintaining sensitivity and specificity over 88%.

To perform the analysis I have maintained the classes balanced. The intention was to eliminate any influence that unbalanced classes could have on the modelling. However, the reality might differ significantly. By intuition we can say that there is a lower portion of successful songs in the total universe of songs composed.

I am unable to assess that proportion as part of this study. However, potentially this could be assessed by looking at sample proportions of other studies performed on hit/non-hot songs or taking several samples across populations of songs and averaging the proportion of successful songs.

To test the efficacy of my final model, I sliced the validation dataset and created an arbitrary unbalanced test set (10% hits vs 90% non-hit songs) with the following code:

```
#testing final model on unbalanced dataset
(set.seed = 75)
validation_unbalanced_no <- validation %>% filter(Is_hit == "No") %>% sample_n(., 3000)
validation_unbalanced_yes <- validation %>% filter(Is_hit == "Yes") %>% sample_n(., 300)

validation_unbalanced <- rbind(validation_unbalanced_no, validation_unbalanced_yes) %>% arrange(.$track
```

When testing the final model to the unbalanced dataset we can see that the level of accuracy increases and maintains an good levels of sensibility and specificity:

```
#testing algorithm in unbalanced population
validation_function(final_model,validation_unbalanced)
```

```
##               acc         pval       sen       spe        f1
## results 0.9454545 4.393963e-15 0.9506667 0.8933333 0.9694086
```

Other limitations and possible further work

We might be able to predict potential hit songs using these characteristics, but it does not imply causality (i.e. these audio features will not necessarily make the song a hit). There could be other confounding factors that are driving the success of certain songs, which were not subject of my analysis.

To establish whether or not a causal relationship exists between the success of the song and its characteristics we would need to include data about those potential confounders (such as marketing, artist trajectory/ability, record company reputation, etc.) to control and account for its effect.

The analysis can be expanded by looking at the characteristics of songs within specific genres to identify the audio features of hit songs for particular genres of interest.

Other features could be considered, for example genre is related to aspects such as specific instrumentation (selection of instruments included in a song), this could be a variable to add. Other variables could be about the quality of the sound: masterization and mix, etc. This would require a measurement of these variables, which at the moment are not available in Spotify data.

Analysis can also be expanded to include geographical, socioeconomic and or audience profile data to stratify further the preference of the public. Also, we could study an specific period as it is expected that taste and preference would fluctuate over time.