Compulsory Exercise 2: Spotify tracks dataset

Cosimo Faeti Johan Lagardère

20 avril, 2023

Abstract

The purpose of this project is to find a model that can predict the popularity of a new song based on its features efficiently. To achieve this, we analyzed an open source dataset of Spotify songs, with a focus on the popularity feature. We used various models, including regularization linear models such as multiple linear, Lasso, and Ridge regression, as well as tree-based and ensemble methods like boosting, bagging, and random forest. We compared the performance of these models using indicators such as MSE or R-squared. Our key findings show that ensemble methods performed better regularization models in predicting song popularity. The MSE are going from 511 for multiple linear regression to 243 for bagging and random Forest. Also in a lot of application boosting outperform other method but here boosting has a MSE of 354 just slightly better than a pruned tree 376. Also the two predictors which contributes the most to the popularity of a song are duration_ms and acousticness. Our results have important implications for the music industry as they have the potential to help music industry professionals better understand what makes a song popular and potentially guide them in creating more successful songs. Additionally, these findings can contribute to the development of more accurate recommendation systems for music streaming services. One potential limitation of our study is that we focused solely on the Spotify dataset of highly rated songs. It's possible that the results would be different if we had included lower-rated songs or songs from other platforms. Another limitation is that our analysis is based solely on the features of the songs, and we did not take into account external factors such as marketing and promotion efforts. Future studies could explore the influence of these external factors on song popularity. Overall, our study provides valuable insights into what makes a song popular, and can help guide music industry professionals in creating more successful songs. By using a wide variety of modeling techniques, we have demonstrated that it is possible to accurately predict song popularity based on its features, which has important implications for both the music industry and music consumers.

Introduction: Scope and purpose of your project

For our project, we choose to study songs and what contributes to make them popular. We found on Kaggle an open source dataset of Spotify songs rated 10/10 in usability. It contains a lot of features but one that we will focus on for our analysis is the popularity between 0 and 100. The scope of this project is to perform prediction of the popularity of a new songs based on its feature(regression task). We will for that use different model: regularization linear model: linear, Lasso and Ridge regression and then tree based /ensemble methods: boosting, bagging and random forest. We will compare the performance of those model on indicators such as MSE which is common for all those models.

Descriptive data analysis/statistics

```
# Loading Dataset
df <- read.csv(file = "dataset.csv")

# Overview of the dataset
str(df)
```

```
## 'data.frame':    114000 obs. of  21 variables:
##  $ X               : int  0 1 2 3 4 5 6 7 8 9 ...
##  $ track_id        : chr  "5SuOikwiRyPMVoIQDJUgSV" "4qPNDBW1i3p13qLCt0Ki3A" "1iJBSr7s7jYXzM8EGcbK5b"
```

```
##  $ artists         : chr  "Gen Hoshino" "Ben Woodward" "Ingrid Michaelson;ZAYN" "Kina Grannis" ...
##  $ album_name      : chr  "Comedy" "Ghost (Acoustic)" "To Begin Again" "Crazy Rich Asians (Original
##  $ track_name      : chr  "Comedy" "Ghost - Acoustic" "To Begin Again" "Can't Help Falling In Love"
##  $ popularity      : int  73 55 57 71 82 58 74 80 74 56 ...
##  $ duration_ms     : int  230666 149610 210826 201933 198853 214240 229400 242946 189613 205594 ...
##  $ explicit        : chr  "False" "False" "False" "False" ...
##  $ danceability    : num  0.676 0.42 0.438 0.266 0.618 0.688 0.407 0.703 0.625 0.442 ...
##  $ energy          : num  0.461 0.166 0.359 0.0596 0.443 0.481 0.147 0.444 0.414 0.632 ...
##  $ key             : int  1 1 0 0 2 6 2 11 0 1 ...
##  $ loudness        : num  -6.75 -17.23 -9.73 -18.52 -9.68 ...
##  $ mode            : int  0 1 1 1 1 1 1 1 1 1 ...
##  $ speechiness     : num  0.143 0.0763 0.0557 0.0363 0.0526 0.105 0.0355 0.0417 0.0369 0.0295 ...
##  $ acousticness    : num  0.0322 0.924 0.21 0.905 0.469 0.289 0.857 0.559 0.294 0.426 ...
##  $ instrumentalness: num  1.01e-06 5.56e-06 0.00 7.07e-05 0.00 0.00 2.89e-06 0.00 0.00 4.19e-03 ...
##  $ liveness        : num  0.358 0.101 0.117 0.132 0.0829 0.189 0.0913 0.0973 0.151 0.0735 ...
##  $ valence         : num  0.715 0.267 0.12 0.143 0.167 0.666 0.0765 0.712 0.669 0.196 ...
##  $ tempo           : num  87.9 77.5 76.3 181.7 119.9 ...
##  $ time_signature  : int  4 4 4 3 4 4 3 4 4 4 ...
##  $ track_genre     : chr  "acoustic" "acoustic" "acoustic" "acoustic" ...
```

```
summary(df)
```

```
##        X            track_id           artists           album_name
##  Min.   :     0   Length:114000      Length:114000      Length:114000
##  1st Qu.: 28500   Class :character   Class :character   Class :character
##  Median : 57000   Mode  :character   Mode  :character   Mode  :character
##  Mean   : 57000
##  3rd Qu.: 85499
##  Max.   :113999
##   track_name          popularity      duration_ms        explicit
##  Length:114000      Min.   :  0.00   Min.   :      0   Length:114000
##  Class :character   1st Qu.: 17.00   1st Qu.: 174066   Class :character
##  Mode  :character   Median : 35.00   Median : 212906   Mode  :character
##                     Mean   : 33.24   Mean   : 228029
##                     3rd Qu.: 50.00   3rd Qu.: 261506
##                     Max.   :100.00   Max.   :5237295
##   danceability        energy            key            loudness
##  Min.   :0.0000   Min.   :0.0000   Min.   : 0.000   Min.   :-49.531
##  1st Qu.:0.4560   1st Qu.:0.4720   1st Qu.: 2.000   1st Qu.:-10.013
##  Median :0.5800   Median :0.6850   Median : 5.000   Median : -7.004
##  Mean   :0.5668   Mean   :0.6414   Mean   : 5.309   Mean   : -8.259
##  3rd Qu.:0.6950   3rd Qu.:0.8540   3rd Qu.: 8.000   3rd Qu.: -5.003
##  Max.   :0.9850   Max.   :1.0000   Max.   :11.000   Max.   :  4.532
##       mode         speechiness       acousticness    instrumentalness
##  Min.   :0.0000   Min.   :0.00000   Min.   :0.0000   Min.   :0.00e+00
##  1st Qu.:0.0000   1st Qu.:0.03590   1st Qu.:0.0169   1st Qu.:0.00e+00
##  Median :1.0000   Median :0.04890   Median :0.1690   Median :4.16e-05
##  Mean   :0.6376   Mean   :0.08465   Mean   :0.3149   Mean   :1.56e-01
##  3rd Qu.:1.0000   3rd Qu.:0.08450   3rd Qu.:0.5980   3rd Qu.:4.90e-02
##  Max.   :1.0000   Max.   :0.96500   Max.   :0.9960   Max.   :1.00e+00
##     liveness         valence           tempo        time_signature
##  Min.   :0.0000   Min.   :0.0000   Min.   :  0.00   Min.   :0.000
##  1st Qu.:0.0980   1st Qu.:0.2600   1st Qu.: 99.22   1st Qu.:4.000
##  Median :0.1320   Median :0.4640   Median :122.02   Median :4.000
##  Mean   :0.2136   Mean   :0.4741   Mean   :122.15   Mean   :3.904
```

```
##  3rd Qu.:0.2730   3rd Qu.:0.6830   3rd Qu.:140.07   3rd Qu.:4.000
##  Max.   :1.0000   Max.   :0.9950   Max.   :243.37   Max.   :5.000
##  track_genre
##  Length:114000
##  Class :character
##  Mode  :character
##
##
##
```

The dimensions are 114000 observations (rows) of 21 variables (columns). The variables are composed by: * Qualitative variables: track_id, artists, album_name, track_name, explicit, key, mode, time_signature, track_genre * Quantitative variables: popularity, duration_ms, danceability, energy, loudness, speechiness, acousticness, instrumentalness, liveness, valence, tempo

```
# Checking duplicates (no duplicates) and missing values (no missing values)
anyDuplicated(df)
```

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

```
any(is.na(df))
```

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

There are no duplicates or missing data in the dataset.

The aim is to predict the popularity of songs by predicting the popularity score, which is a numerical value that ranges from 0 to 100. Thus, some qualitative variables such as track_id, artists, album_name, track_name and the fist unnamed column will be discarded from the data set. Moreover, track_genre and explicit will be converted into numerical variables. The former ranges from 1 to 114, while the latter from 1 (= False) to 2 (= True).

```
Spotify = subset(df, select = -c(X, track_id, artists, album_name, track_name))
str(Spotify)
```

```
## 'data.frame':    114000 obs. of  16 variables:
##  $ popularity      : int  73 55 57 71 82 58 74 80 74 56 ...
##  $ duration_ms     : int  230666 149610 210826 201933 198853 214240 229400 242946 189613 205594 ...
##  $ explicit        : chr  "False" "False" "False" "False" ...
##  $ danceability    : num  0.676 0.42 0.438 0.266 0.618 0.688 0.407 0.703 0.625 0.442 ...
##  $ energy          : num  0.461 0.166 0.359 0.0596 0.443 0.481 0.147 0.444 0.414 0.632 ...
##  $ key             : int  1 1 0 0 2 6 2 11 0 1 ...
##  $ loudness        : num  -6.75 -17.23 -9.73 -18.52 -9.68 ...
##  $ mode            : int  0 1 1 1 1 1 1 1 1 1 ...
##  $ speechiness     : num  0.143 0.0763 0.0557 0.0363 0.0526 0.105 0.0355 0.0417 0.0369 0.0295 ...
##  $ acousticness    : num  0.0322 0.924 0.21 0.905 0.469 0.289 0.857 0.559 0.294 0.426 ...
##  $ instrumentalness: num  1.01e-06 5.56e-06 0.00 7.07e-05 0.00 0.00 2.89e-06 0.00 0.00 4.19e-03 ...
##  $ liveness        : num  0.358 0.101 0.117 0.132 0.0829 0.189 0.0913 0.0973 0.151 0.0735 ...
##  $ valence         : num  0.715 0.267 0.12 0.143 0.167 0.666 0.0765 0.712 0.669 0.196 ...
##  $ tempo           : num  87.9 77.5 76.3 181.7 119.9 ...
##  $ time_signature  : int  4 4 4 3 4 4 3 4 4 4 ...
##  $ track_genre     : chr  "acoustic" "acoustic" "acoustic" "acoustic" ...
```

```
# Convert track_genre and explicit in numerical variable
Spotify$track_genre <- as.numeric(factor(Spotify$track_genre))
Spotify$explicit <- as.numeric(factor(Spotify$explicit))
str(Spotify)
```

```
## 'data.frame':    114000 obs. of  16 variables:
##  $ popularity      : int  73 55 57 71 82 58 74 80 74 56 ...
##  $ duration_ms     : int  230666 149610 210826 201933 198853 214240 229400 242946 189613 205594 ...
##  $ explicit        : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ danceability    : num  0.676 0.42 0.438 0.266 0.618 0.688 0.407 0.703 0.625 0.442 ...
##  $ energy          : num  0.461 0.166 0.359 0.0596 0.443 0.481 0.147 0.444 0.414 0.632 ...
##  $ key             : int  1 1 0 0 2 6 2 11 0 1 ...
##  $ loudness        : num  -6.75 -17.23 -9.73 -18.52 -9.68 ...
##  $ mode            : int  0 1 1 1 1 1 1 1 1 1 ...
##  $ speechiness     : num  0.143 0.0763 0.0557 0.0363 0.0526 0.105 0.0355 0.0417 0.0369 0.0295 ...
##  $ acousticness    : num  0.0322 0.924 0.21 0.905 0.469 0.289 0.857 0.559 0.294 0.426 ...
##  $ instrumentalness: num  1.01e-06 5.56e-06 0.00 7.07e-05 0.00 0.00 2.89e-06 0.00 0.00 4.19e-03 ...
##  $ liveness        : num  0.358 0.101 0.117 0.132 0.0829 0.189 0.0913 0.0973 0.151 0.0735 ...
##  $ valence         : num  0.715 0.267 0.12 0.143 0.167 0.666 0.0765 0.712 0.669 0.196 ...
##  $ tempo           : num  87.9 77.5 76.3 181.7 119.9 ...
##  $ time_signature  : int  4 4 4 3 4 4 3 4 4 4 ...
##  $ track_genre     : num  1 1 1 1 1 1 1 1 1 1 ...
```

Range of the variables are shown below:

```
# quant = which(names(Spotify)%in%c('popularity',
# 'duration_ms','explicit','danceability', 'energy', 'key', 'loudness','mode',
# 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence',
# 'tempo', 'time_signature', 'track_genre'))
sapply(Spotify[, ], range)  # sapply(Spotify[, quant], range)
```

```
##      popularity duration_ms explicit danceability energy key loudness mode
## [1,]          0           0        1        0.000      0   0  -49.531    0
## [2,]        100     5237295        2        0.985      1  11    4.532    1
##      speechiness acousticness instrumentalness liveness valence   tempo
## [1,]       0.000        0.000                0        0   0.000   0.000
## [2,]       0.965        0.996                1        1   0.995 243.372
##      time_signature track_genre
## [1,]              0           1
## [2,]              5         114
```

Mean, median and standard deviation of the variables:

```
# Mean, Median & Standard Deviation of quantitative predictors
sapply(Spotify[, ], mean)  #sapply(Spotify[, quant], mean)
```

```
##       popularity      duration_ms         explicit     danceability
##     3.323854e+01     2.280292e+05     1.085500e+00     5.668001e-01
##           energy              key         loudness             mode
##     6.413828e-01     5.309140e+00    -8.258960e+00     6.375526e-01
##      speechiness     acousticness instrumentalness         liveness
##     8.465211e-02     3.149101e-01     1.560496e-01     2.135528e-01
##          valence            tempo   time_signature      track_genre
##     4.740682e-01     1.221478e+02     3.904035e+00     5.750000e+01
```

```
sapply(Spotify[, ], sd)  #sapply(Spotify[, quant], sd)
```

```
##       popularity      duration_ms         explicit     danceability
##     2.230508e+01     1.072977e+05     2.796255e-01     1.735422e-01
##           energy              key         loudness             mode
##     2.515291e-01     3.559987e+00     5.029337e+00     4.807092e-01
##      speechiness     acousticness instrumentalness         liveness
```

```
##     1.057324e-01     3.325227e-01     3.095548e-01     1.903777e-01
##          valence            tempo   time_signature      track_genre
##     2.592611e-01     2.997820e+01     4.326208e-01     3.290784e+01
```

```
sapply(Spotify[, ], median)  #sapply(Spotify[, quant], median)
```

```
##       popularity      duration_ms         explicit     danceability
##      3.50000e+01      2.12906e+05      1.00000e+00      5.80000e-01
##           energy              key         loudness             mode
##      6.85000e-01      5.00000e+00     -7.00400e+00      1.00000e+00
##      speechiness     acousticness instrumentalness         liveness
##      4.89000e-02      1.69000e-01      4.16000e-05      1.32000e-01
##          valence            tempo   time_signature      track_genre
##      4.64000e-01      1.22017e+02      4.00000e+00      5.75000e+01
```

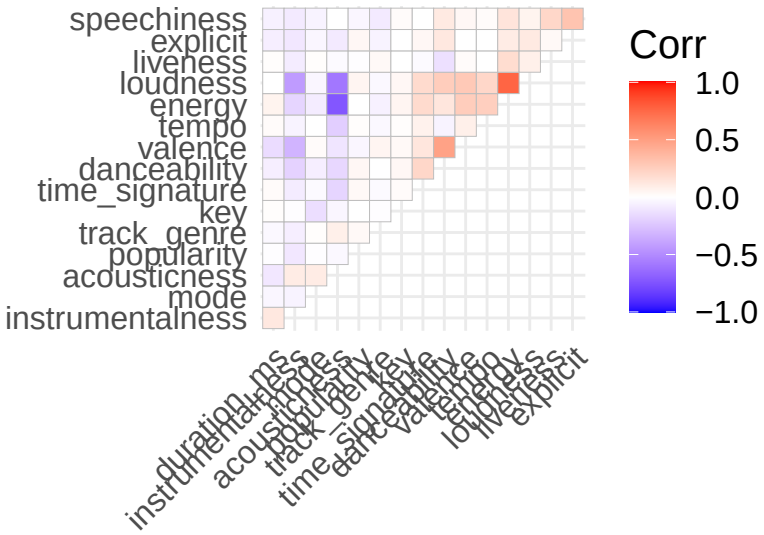
Correlation and Heatmap are shown below:

```
# ggpairs(df[, quant]) + theme_minimal()
corr <- round(cor(Spotify[, ]), 2)
corr
```

```
##                  popularity duration_ms explicit danceability energy   key
## popularity             1.00       -0.01     0.04         0.04   0.00  0.00
## duration_ms           -0.01        1.00    -0.07        -0.07   0.06  0.01
## explicit               0.04       -0.07     1.00         0.12   0.10  0.00
## danceability           0.04       -0.07     0.12         1.00   0.13  0.04
## energy                 0.00        0.06     0.10         0.13   1.00  0.05
## key                    0.00        0.01     0.00         0.04   0.05  1.00
## loudness               0.05        0.00     0.11         0.26   0.76  0.04
## mode                  -0.01       -0.04    -0.04        -0.07  -0.08 -0.14
## speechiness           -0.04       -0.06     0.31         0.11   0.14  0.02
## acousticness          -0.03       -0.10    -0.09        -0.17  -0.73 -0.04
## instrumentalness      -0.10        0.12    -0.10        -0.19  -0.18 -0.01
## liveness              -0.01        0.01     0.03        -0.13   0.18  0.00
## valence               -0.04       -0.15     0.00         0.48   0.26  0.03
## tempo                  0.01        0.02     0.00        -0.05   0.25  0.01
## time_signature         0.03        0.02     0.04         0.21   0.19  0.02
## track_genre            0.03       -0.03    -0.05         0.00  -0.06 -0.01
##                  loudness  mode speechiness acousticness instrumentalness
## popularity           0.05 -0.01       -0.04        -0.03            -0.10
## duration_ms          0.00 -0.04       -0.06        -0.10             0.12
## explicit             0.11 -0.04        0.31        -0.09            -0.10
## danceability         0.26 -0.07        0.11        -0.17            -0.19
## energy               0.76 -0.08        0.14        -0.73            -0.18
## key                  0.04 -0.14        0.02        -0.04            -0.01
## loudness             1.00 -0.04        0.06        -0.59            -0.43
## mode                -0.04  1.00       -0.05         0.10            -0.05
## speechiness          0.06 -0.05        1.00         0.00            -0.09
## acousticness        -0.59  0.10        0.00         1.00             0.10
## instrumentalness    -0.43 -0.05       -0.09         0.10             1.00
## liveness             0.08  0.01        0.21        -0.02            -0.08
## valence              0.28  0.02        0.04        -0.11            -0.32
## tempo                0.21  0.00        0.02        -0.21            -0.05
## time_signature       0.19 -0.02        0.00        -0.18            -0.08
## track_genre         -0.03  0.01       -0.09         0.08            -0.07
##                  liveness valence tempo time_signature track_genre
```

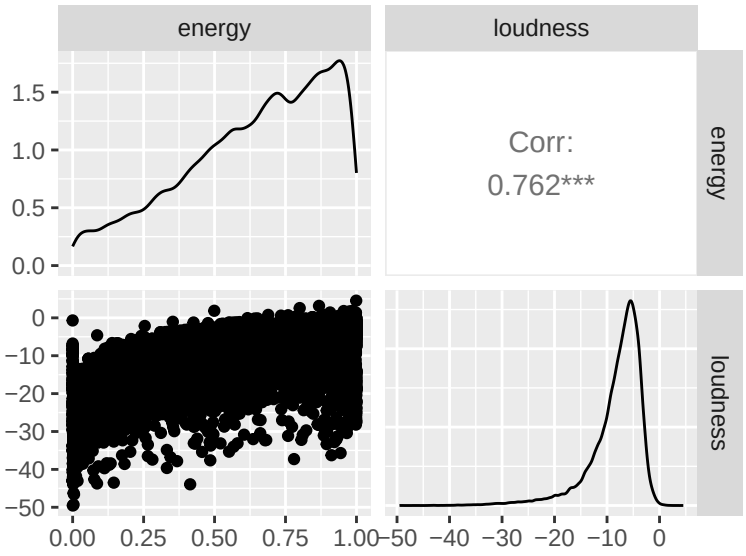
```
## popularity          -0.01   -0.04  0.01          0.03        0.03
## duration_ms          0.01   -0.15  0.02          0.02       -0.03
## explicit             0.03    0.00  0.00          0.04       -0.05
## danceability        -0.13    0.48 -0.05          0.21        0.00
## energy               0.18    0.26  0.25          0.19       -0.06
## key                  0.00    0.03  0.01          0.02       -0.01
## loudness             0.08    0.28  0.21          0.19       -0.03
## mode                 0.01    0.02  0.00         -0.02        0.01
## speechiness          0.21    0.04  0.02          0.00       -0.09
## acousticness        -0.02   -0.11 -0.21         -0.18        0.08
## instrumentalness    -0.08   -0.32 -0.05         -0.08       -0.07
## liveness             1.00    0.02  0.00         -0.02        0.03
## valence              0.02    1.00  0.08          0.13        0.05
## tempo                0.00    0.08  1.00          0.07       -0.03
## time_signature      -0.02    0.13  0.07          1.00       -0.02
## track_genre          0.03    0.05 -0.03         -0.02        1.00
```

```
# cov <- round(cov(Spotify[, ]), 2) cov
heatmap <- ggcorrplot(corr, hc.order = TRUE, type = "upper", lab = FALSE) + theme(text = element_text(s
    plot.margin = unit(c(0.001, 0.001, 0.001, 0.001), "cm"))
heatmap
```

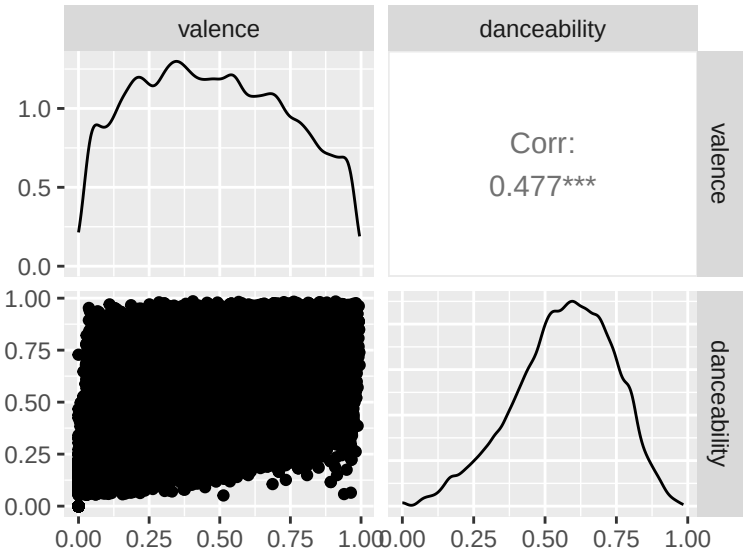


As we can observe from the heatmap, popularity has no significant correlation with any variables. In addition, there exists a strong positive relationship between energy-loudness, valence-danceability, while a strong negative relationship between energy-acousticness, loudness-acousticness and loudness-instrumentalness. A more precise visualization of the relationships among these variables is shown in the scatterplots below:

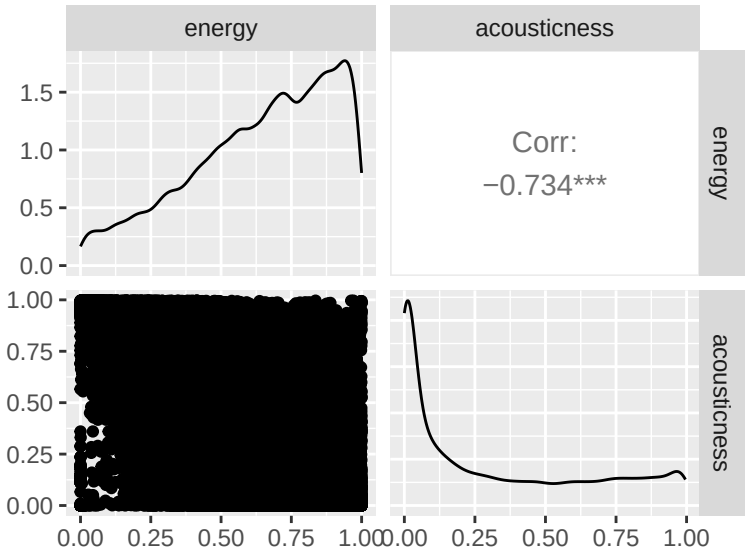
```
# Scatterplot energy - loudness (Corr = 0.76)
EnergyLoudness <- ggpairs(Spotify[, c(5, 7)])
EnergyLoudness
```



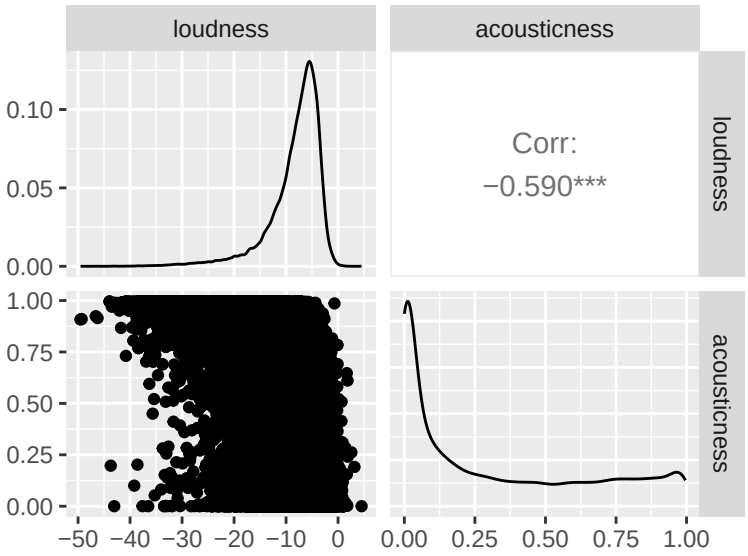

```
# Scatterplot valence - danceability (Corr = 0.48)
ValenceDanceability <- ggpairs(Spotify[, c(13, 4)])
ValenceDanceability
```



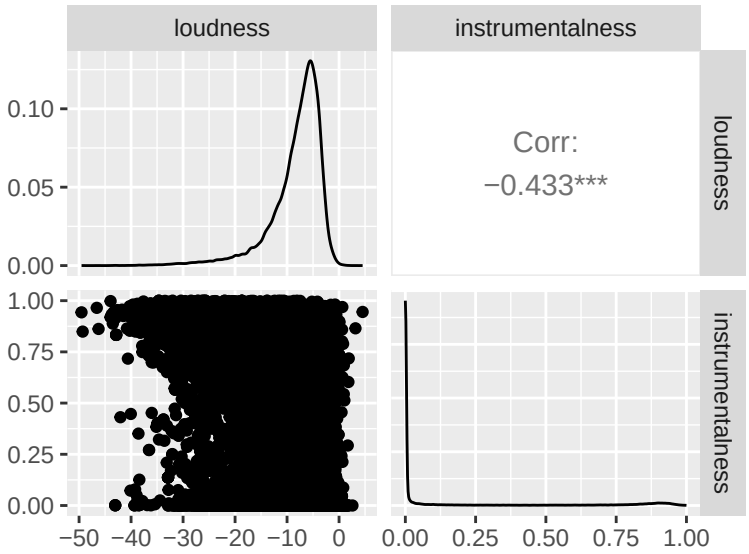

```
# Scatterplot energy - acousticness (Corr = - 0.73)
EnergyAcousticness <- ggpairs(Spotify[, c(5, 10)])
EnergyAcousticness
```




```
# Scatterplot loudness - acousticness (Corr = - 0.59)
LoudnessAcousticness <- ggpairs(Spotify[, c(7, 10)])
LoudnessAcousticness
```

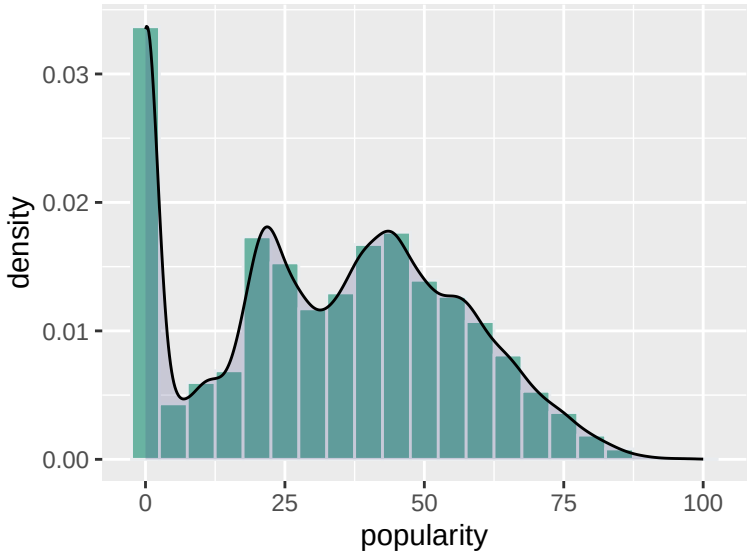



```
# Scatterplot loudness - instrumentalness (Corr = - 0.43)
LoudnessInstrumentalness <- ggpairs(Spotify[, c(7, 11)])
LoudnessInstrumentalness
```



The density plot of popularity shows that the vast majority of songs have a low popularity score, i.e. a score below the 50 threshold. It is also evident that most of the tracks are defined as non-popular with a score of zero.

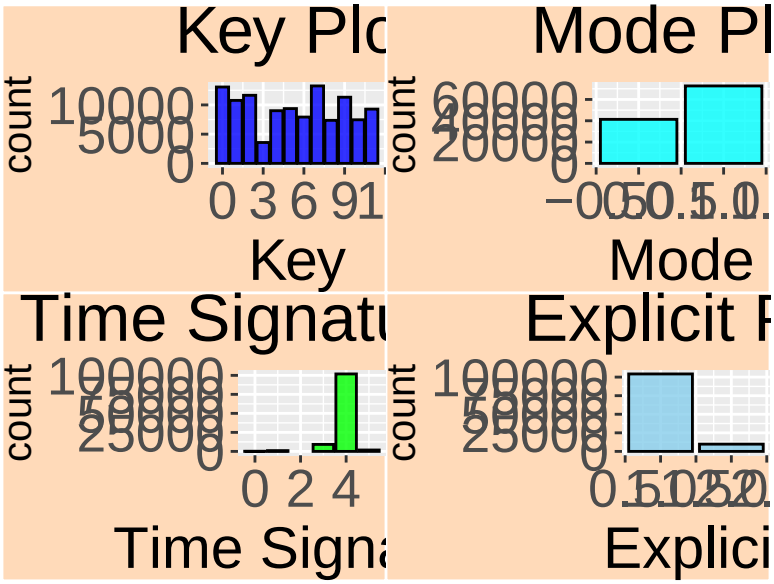
```
PopPlot <- ggplot(Spotify, aes(x = popularity)) + geom_histogram(aes(y = ..density..),
    binwidth = 5, fill = "#69b3a2", color = "#e9ecef") + geom_density(alpha = 0.2,
    fill = "#404080")
PopPlot
```



The barplots show the distribution of qualitative variables. For instance, we can derive that the majority of the track are non-explicit or the most common number of beats in each bar of the track is 4.

```
# Barplots of qualitative variables
tema = theme(plot.background = element_rect(fill = "#FFDAB9"), plot.title = element_text(size = 25,
    hjust = 0.5), axis.title.x = element_text(size = 22, color = "black"), axis.text.x = element_text(s
    axis.text.y = element_text(size = 20))
keyPlot <- ggplot(data = Spotify, mapping = aes(x = key)) + geom_bar(fill = "blue",
    color = "black", linewidth = 0.5, alpha = 0.8) + theme(text = element_text(size = 15),
```

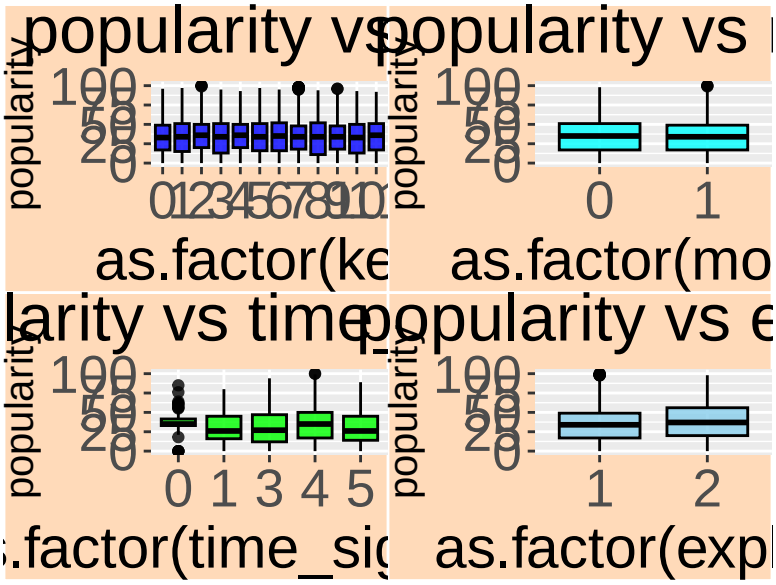
```
    plot.margin = unit(c(0.001, 0.001, 0.001, 0.001), "cm")) + xlab("Key") + ggtitle("Key Plot") +
    tema
modePlot <- ggplot(data = Spotify, mapping = aes(x = mode)) + geom_bar(fill = "cyan",
    color = "black", linewidth = 0.5, alpha = 0.8) + theme(text = element_text(size = 15),
    plot.margin = unit(c(0.001, 0.001, 0.001, 0.001), "cm")) + xlab("Mode") + ggtitle("Mode Plot") +
    tema
time_signaturePlot <- ggplot(data = Spotify, mapping = aes(x = time_signature)) +
    geom_bar(fill = "green", color = "black", linewidth = 0.5, alpha = 0.8) + theme(text = element_text
    plot.margin = unit(c(0.001, 0.001, 0.001, 0.001), "cm")) + xlab("Time Signature") +
    ggtitle("Time Signature Plot") + tema
explicitPlot <- ggplot(data = Spotify, mapping = aes(x = explicit)) + geom_bar(fill = "skyblue",
    color = "black", linewidth = 0.5, alpha = 0.8) + theme(text = element_text(size = 15),
    plot.margin = unit(c(0.001, 0.001, 0.001, 0.001), "cm")) + xlab("Explicit") +
    ggtitle("Explicit Plot") + tema
plot_grid(keyPlot, modePlot, time_signaturePlot, explicitPlot, nrow = 2, ncol = 2)
```



Boxplots of key, mode and explicit (against popularity) shows the same distribution and same median among the data, while different behaviors appears for different values of time_signature:

```
# Boxplots of qualitative variables/Popularity
boxplotPopKey <- ggplot(Spotify, aes(as.factor(key), popularity)) + geom_boxplot(fill = "blue",
    color = "black", linewidth = 0.5, alpha = 0.8) + theme(text = element_text(size = 15),
    plot.margin = unit(c(0.001, 0.001, 0.001, 0.001), "cm")) + labs(title = "popularity vs key") +
    tema
boxplotPopMode <- ggplot(Spotify, aes(as.factor(mode), popularity)) + geom_boxplot(fill = "cyan",
    color = "black", linewidth = 0.5, alpha = 0.8) + theme(text = element_text(size = 15),
    plot.margin = unit(c(0.001, 0.001, 0.001, 0.001), "cm")) + labs(title = "popularity vs mode") +
    tema
boxplotPopTimeSignature <- ggplot(Spotify, aes(as.factor(time_signature), popularity)) +
    geom_boxplot(fill = "green", color = "black", linewidth = 0.5, alpha = 0.8) +
    theme(text = element_text(size = 15), plot.margin = unit(c(0.001, 0.001, 0.001,
        0.001), "cm")) + labs(title = "popularity vs time_signature") + tema
boxplotPopExpl <- ggplot(Spotify, aes(as.factor(explicit), popularity)) + geom_boxplot(fill = "skyblue"
    color = "black", linewidth = 0.5, alpha = 0.8) + theme(text = element_text(size = 15),
    plot.margin = unit(c(0.001, 0.001, 0.001, 0.001), "cm")) + labs(title = "popularity vs explicit") +
```

```
    tema
plot_grid(boxplotPopKey, boxplotPopMode, boxplotPopTimeSignature, boxplotPopExpl,
    nrow = 2, ncol = 2)
```



Methods

The aim is to predict the popularity of songs which is a numerical value ranges from 0 to 100, so the task is a regression problem. The main methods used to solve it are linear and tree-based models. In details, we will explore linear regression and some regularization approaches such as lasso, ridge and polynomial, while from the family of tree-based models we will see regression tree and ensemble methods such as bagging, random forest and boosting.

The first method used is a multiple linear regression which is basically trying to find a linear relationship between our predictors $x_i = (x_{i1}, x_{i2}, ..., x_{ip})$ and our response y_i. There are four major assumptions in our model: 1) the distribution of model residuals should be approximately normal 2)independence of the observations 3) the variance of error in our prediction should be homogeneous 4)the line of best fit through the data points should be linear. If all these assumptions are respected the performance of the multiple linear regression in predicting popularity can be really good. Then there is the assessment of the goodness of fit of the model with firstly the multiple R-squared who gives us the R-squared (by squaring it) which indicates the variance in popularity that can be explained by the predictors in the model.

The Lasso and Ridge regression are based on the same principle: minimize the sum between the RSS(residual sum of squares) and a shrinkage penalty. The expression for the RSS is $RSS = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$. For Lasso the shrinkage is $\lambda \sum_{j=1}^{p} |\beta_j|$ and for Ridge $\lambda \sum_{j=1}^{p} \beta_j^2$ where n is the number of observations, p the number of predictors and these shrinkage are respectively the L1 and L2 norm of the coefficients. We will choose a value of λ that produces the lowest possible test MSE using k-fold cross validation and then using this value we can determine the R-squared and the MSE of our model.

The main idea behind tree-based methods is to derive a set of decision rules for segmenting the predictor space into a number of finer and finer regions. All points in the same region will be given the same predictive value, in our regression case, as the mean of all values in that square.

Regression tree. In our case, we want to build a tree that predict the popularity of songs by setting splitting rules to segment the predictor space and summarized them in a tree. In general, we assume a dataset composed of n pairs (x_i, y_i) ,$i = 1, ..., n$, and each predictor is $x_i = (x_{i1}, x_{i2}, ..., x_{ip})$, the goal is to predict y_i. The process of building a regression tree can be divided into two steps: * Divide the predictor space into J distinct and non-overlapping regions, $R_1, R_2, ..., R_J$ * For every observation that falls into region R_j we

make the same prediction, which is the mean of the responses for the training observations that fall into R_j The aim is to divide the predictor space into non-overlapping regions $R_1, R_2, ..., R_J$ such that minimize the residual sum of squares (RSS) on the training set, i.e., $RSS = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$. In our case, in order to reduce the variance (at the cost of an increased bias) given by a large tree, we pruned it back in order to obtain a subtree. In general, for this task is used a cost complexity pruning $C_\alpha(T) = Q(T) + \alpha|T|$. A key component is the parameter α (in our case, using `rpart` package is represented by `cp`) which penalize the number of terminal nodes by ensuring that the tree does not get too many branches. To get the optimal α for the pruning step, we explored the complexity parameter with optimal (smallest) estimated prediction error.

The main advantage of trees is the simplicity in which we can interpret the results. On the other hand, as our case will show, large trees are not easy to interpret. In addition, we have a poor prediction performance (high variance) and trees are not very robust to changes in the data. Given the above drawbacks, but also due to the nature of the problem, we ended up addressing them with ensemble methods (bagging, random forests and boosting).

Bagging grows many trees from a bootstrapped data and average to get rid of the non-robustness and high variance. The idea is mainly based on bootstrapping which is the process of drawing with replacement n observations from our sample. After bootstrapped the first sample, we repeat the process B times and get B bootstrap samples. For each bootstrap sample $b = 1, ..., B$ we construct a tree $\hat{f}^{*b}(x)$. In our regression problem, we take the average of all of the predictions and use this as the final result, i.e., $\hat{f}_{avg}(x) = \frac{1}{B} \sum_{b=1}^{B} Var\hat{f}^{*b}$. A drawback of bagging (but also for random forests) is that it becomes difficult to interpret the results. In fact, instead of having just one tree, the resulting model consists of many trees. To overcome this problem, we use the variable importance plot that show the relative importance of each predictors. The predictors are sorted according to their importance.

Random forests inject randomness (and less variance) by allowing a random selection of predictors to be used for the splits at each node. As in bagging, we build a number of trees on bootstrapped training samples, but each time a split in a tree is considered, a random selection of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors. In regression, the optimal m is given by $m = p/3$.

Boosting make one tree, then another based on the residuals from the previous, and repeat it. The final predictor is a weighted sum of these trees. Different from the previous two ensemble methods, boosting requires hyperparameter tuning for number of trees (B), shrinkage parameter (λ) that control the rate at which boosting learns, and interaction depth (d), i.e., the number of splits in each tree. For this purpose we used 5-folds cross-validation.

One of the technique that we used is cross-validation: splitting of the data into folds training the model on a subset of the data and then testing it on the remaining subset. By training and testing the model on different subsets of the data, cross-validation can provide a more reliable estimate of the model's performance than simply using a single train-test split. The most common form of cross-validation is k-fold cross-validation, where the data is divided into k equally sized folds. The model is then trained on k-1 folds and tested on the remaining fold. This process is repeated k times, with each fold used as the test set once. The results of each iteration are averaged to give an overall estimate of the model's performance.

MSE measures the average of the squared differences between the predicted and actual values. In other words, it calculates the average of the errors squared. The reason we square the errors is to penalize larger errors more heavily, as they have a greater impact on the overall performance of the model. MSE is used to compare the quality of different machine learning models because it provides a quantitative measure of how well the model is able to predict the target variable and it can be used on very different kinds of models.

Results and interpretation:

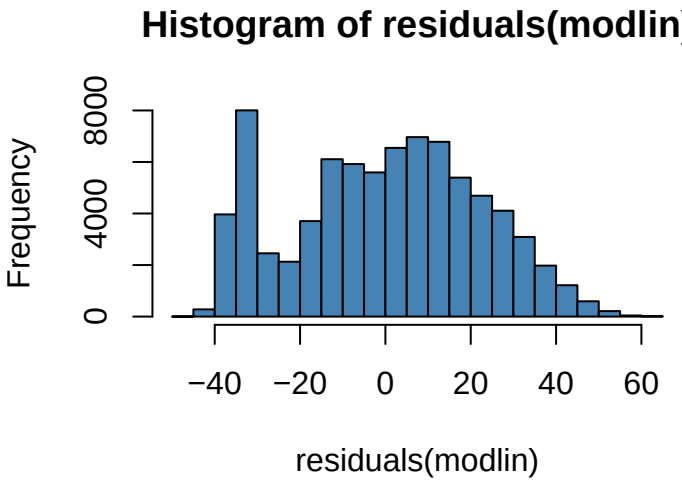
As a first step, we split the dataset 70% in training set and 30% in test set.

```
# Splitting data into train (70%) and test (30%) set
set.seed(1)
```

```
n = nrow(Spotify)
train = sample(1:n, 0.7 * nrow(Spotify), replace = FALSE)
test = (1:n)[-train]
Spotify_train = Spotify[train, ]
Spotify_test = Spotify[test, ]
```

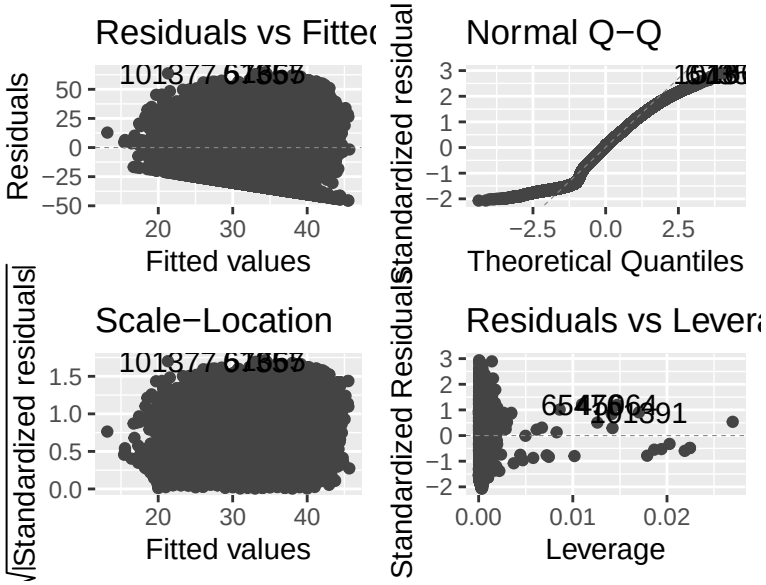
#multiple linear regression We first define our model using the training data and all the predictors in Spotify_train.

```
modlin = lm(popularity ~ ., data = Spotify_train)
hist(residuals(modlin), col = "steelblue")
```



The distribution of the residuals seems to be containing two mode like a sum of two gaussian but we can't reject the hypothesis of linearity of the residuals with just that.

```
library(ggfortify)
autoplot(modlin, smooth.colour = NA)
```



It's hard to see if the variance of the residuals is consistent over all the estimation because we can't see if the residuals are equally scattered at every fitted value. But with the QQ plot we can clearly see that the residuals are non linear the hypothesis of linearity can be rejected. We can't really conclude something from the scale location plot. Finally there is almost no leverage in the data set.

```
summary(modlin)
```

```
##
## Call:
## lm(formula = popularity ~ ., data = Spotify_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -45.527 -15.607   1.458  16.226  64.664
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)       2.872e+01  1.115e+00  25.763  < 2e-16 ***
## duration_ms      -2.060e-06  7.289e-07  -2.826  0.00472 **
## explicit          3.603e+00  2.959e-01  12.175  < 2e-16 ***
## danceability      7.858e+00  5.599e-01  14.035  < 2e-16 ***
## energy           -3.074e+00  6.397e-01  -4.806 1.55e-06 ***
## key              -2.315e-02  2.210e-02  -1.047  0.29488
## loudness          1.289e-01  2.807e-02   4.593 4.37e-06 ***
## mode             -6.761e-01  1.655e-01  -4.085 4.42e-05 ***
## speechiness      -1.402e+01  8.269e-01 -16.954  < 2e-16 ***
## acousticness     -1.004e+00  3.650e-01  -2.751  0.00595 **
## instrumentalness -7.535e+00  3.057e-01 -24.654  < 2e-16 ***
## liveness          1.217e+00  4.373e-01   2.784  0.00537 **
## valence          -9.557e+00  3.785e-01 -25.253  < 2e-16 ***
## tempo             1.448e-02  2.721e-03   5.321 1.03e-07 ***
## time_signature    1.104e+00  1.876e-01   5.883 4.05e-09 ***
## track_genre       1.859e-02  2.401e-03   7.741 9.96e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 22.02 on 79784 degrees of freedom
## Multiple R-squared:  0.02539,    Adjusted R-squared:  0.02521
## F-statistic: 138.6 on 15 and 79784 DF,  p-value: < 2.2e-16
```

```
pred.modlin = predict(modlin, data = Spotify_train)
mse.lin = mean((pred.modlin - Spotify_test$popularity)^2)
mse.lin
```

```
## [1] 511.7771
```

From the summary, we directly see that we have a R-squared of 0.025 which shows that the proportion of variance explained by the model is really small. This can be due to the non-linearity of the data set. Finally we predict with our model using the testing data set values of popularity that we compare to the popularity on our test data set.

#Lasso and Ridge regression

```
# define response variable
y_train <- Spotify_train$popularity
y_test <- Spotify_test$popularity
# define matrix of predictor variables excluding popularity
```

```
quant2 = which(names(Spotify) %in% c("duration_ms", "explicit", "danceability", "energy",
    "key", "loudness", "mode", "speechiness", "acousticness", "instrumentalness",
    "liveness", "valence", "tempo", "time_signature", "track_genre"))
x_train <- data.matrix(Spotify_train[, quant2])
x_test <- data.matrix(Spotify_test[, quant2])
```

We perform the study on Lasso and Ridge at the same time as the only difference is alpha=1 for lasso and alpha=0 for ridge

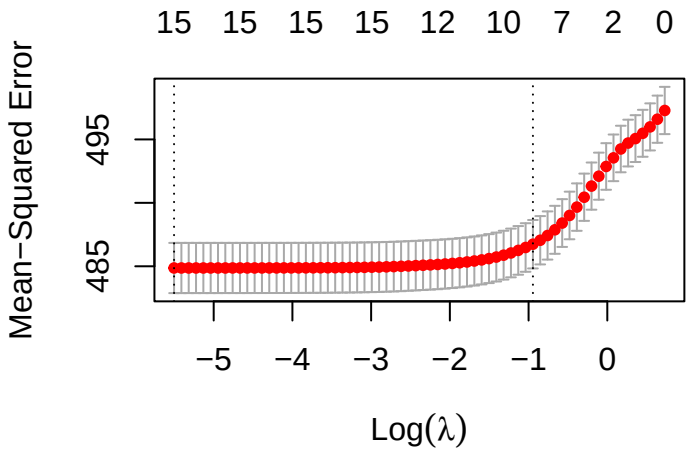
```
# perform k-fold cross-validation to find optimal lambda value
cv_modellasso <- cv.glmnet(x_train, y_train, alpha = 1)
cv_modelridge <- cv.glmnet(x_train, y_train, alpha = 0)
# find optimal lambda value that minimizes test MSE
best_lambda_lasso <- cv_modellasso$lambda.min
log(best_lambda_lasso)
```

```
## [1] -5.503478
```

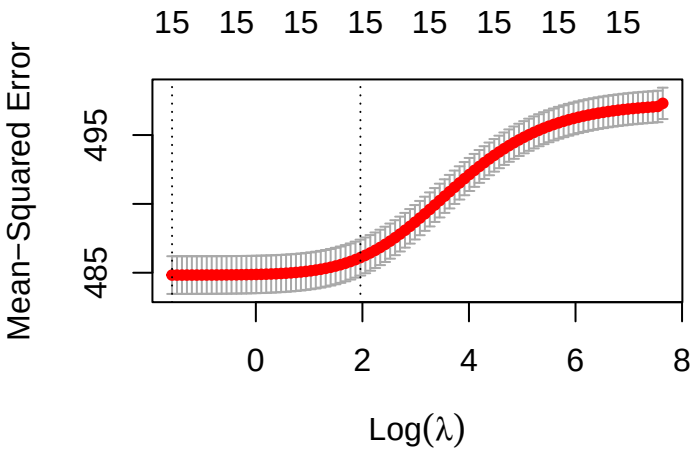
```
best_lambda_ridge <- cv_modelridge$lambda.min
log(best_lambda_ridge)
```

```
## [1] -1.572803
```

```
# produce plot of test MSE by lambda value
plot(cv_modellasso)
```




```
plot(cv_modelridge)
```



With these plot we can determine that the best value of lambda for both method is when lamdbda is minimal and we get their value: $\lambda_{lasso}^{opt} = -5.503478$ and $\lambda_{ridge}^{opt} = -1.572803$

```
pred.lasso = predict(cv_modellasso, newx = x_test, type = "response")
mse_lasso <- mean((pred.lasso - y_test)^2)
mse_lasso
```

```
## [1] 486.9217
```

```
pred.ridge = predict(cv_modelridge, newx = x_test, type = "response")
mse_ridge <- mean((pred.ridge - y_test)^2)
mse_ridge
```

```
## [1] 486.4636
```

We then determine the best coefficient for our two model: β_j^{opt}:

```
best_model_lasso <- glmnet(x_train, y_train, alpha = 1, lambda = best_lambda_lasso)
coef(best_model_lasso)
```

```
## 16 x 1 sparse Matrix of class "dgCMatrix"
##                              s0
## (Intercept)       2.865675e+01
## duration_ms      -2.018672e-06
## explicit          3.589287e+00
## danceability      7.837970e+00
## energy           -2.928312e+00
## key              -2.188565e-02
## loudness          1.254824e-01
## mode             -6.661347e-01
## speechiness      -1.398397e+01
## acousticness     -9.487131e-01
## instrumentalness -7.536153e+00
## liveness          1.162739e+00
## valence          -9.538731e+00
## tempo             1.427353e-02
## time_signature    1.094454e+00
## track_genre       1.846894e-02
```

```
best_model_ridge <- glmnet(x_train, y_train, alpha = 0, lambda = best_lambda_ridge)
coef(best_model_ridge)
```

```
## 16 x 1 sparse Matrix of class "dgCMatrix"
##                              s0
## (Intercept)       2.880209e+01
## duration_ms      -2.022058e-06
## explicit          3.573293e+00
## danceability      7.664192e+00
## energy           -3.069839e+00
## key              -2.311189e-02
## loudness          1.294290e-01
## mode             -6.701328e-01
## speechiness      -1.380511e+01
## acousticness     -1.007182e+00
## instrumentalness -7.432949e+00
## liveness          1.166759e+00
## valence          -9.369029e+00
## tempo             1.417093e-02
## time_signature    1.100236e+00
## track_genre       1.846413e-02
```

Finally we predict and we can determine the MSE and the R-squared of the two methods.

```
# use fitted best model to make predictions
y_predicted_lasso <- predict(best_model_lasso, s = best_lambda_lasso, newx = x_test)
y_predicted_ridge <- predict(best_model_ridge, s = best_lambda_ridge, newx = x_test)
# find SST and SSE
sst <- sum((y_test - mean(y_test))^2)
sse_lasso <- sum((y_predicted_lasso - y_test)^2)
sse_ridge <- sum((y_predicted_ridge - y_test)^2)
# find R-Squared
rsq_lasso <- 1 - sse_lasso/sst
rsq_lasso
```

```
## [1] 0.02616177
```

```
rsq_ridge <- 1 - sse_ridge/sst
rsq_ridge
```

```
## [1] 0.0261379
```

```
mse_lasso <- mean((y_predicted_lasso - y_test)^2)
mse_lasso
```

```
## [1] 484.9738
```

```
mse_ridge <- mean((y_predicted_ridge - y_test)^2)
mse_ridge
```

```
## [1] 484.9857
```

For regression tree, we first force the model to build a very large tree via the arguments of the function `rpart.control`. Note that by using a simple greedy approach, the tree would result in a single terminal node (more details are shown below). At the same time, to obtain a good picture of the evolution of the error, we set the smallest complexity parameter to be considered by the cross-validation experiment to a very low value (1e-8).

```
## Regression tree
set.seed(3445)
mycontrol <- rpart.control(minsplit = 2, cp = 1e-05, xval = 10)
fit.regrtree <- rpart(popularity ~ ., data = Spotify_train, method = "anova", control = mycontrol)
```

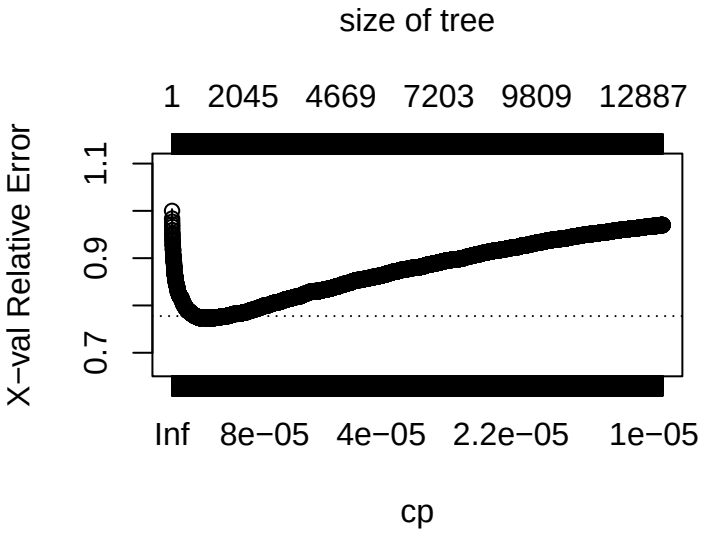
```
# Plotting the tree rpart.plot(fit.regrtree, main = 'Regression Tree',
# compress=TRUE)

# Predict on the test set
pred.regrtree = predict(fit.regrtree, newdata = Spotify_test)
mse.regrtree = mean((pred.regrtree - Spotify_test$popularity)^2)
mse.regrtree  #mse = 464.1089
```

```
## [1] 464.1089
```

As expected, using the previous approach lead us to a large tree and the MSE on test set is 464.1089 which is a better result than linear model. In order to lower the variance, with an higher cost of more bias, we prune the tree back to obtain a subtree:

```
# Pruning step: Plotting values of cp
plotcp(fit.regrtree)
```




```
# Find optimal cp
(b <- fit.regrtree$cptable[which.min(fit.regrtree$cptable[, "xerror"]), "CP"])  # optimal cp = 0.000144
```

```
## [1] 0.0001445644
```

By looking at the plot for the complexity parameter `cp` can be found out that the optimal value is about 0.001 - thesis confirmed by the code.

```
set.seed(3446)
fit.regrtreeprune <- prune(fit.regrtree, cp = b)
```

```
# Predict on test set
pred.regrtreeprune = predict(fit.regrtreeprune, newdata = Spotify_test)
mse.regrtreeprune = mean((pred.regrtreeprune - Spotify_test$popularity)^2)
mse.regrtreeprune  # mse = 376.672
```

```
## [1] 376.672
```

As a result of pruning step, we get smaller tree with respect to the previous one but still composed by too many nodes. Important to notice that the MSE test is dropped to 376.672.

For the sake of completeness, we report a regression tree developed using `tree()` package, instead of `rpart()`. In this specific case, we configured the tree using `tree.control` by setting default parameters: the minimum number of observations to include in either child node `mincut=1` and the smallest allowed node size `minsize=2`. As a result, we get a single terminal node and a MSE test of 498.0025.

```
# First trial with tree() packages -> one terminal node
set.seed(3447)
fit.regrtreeOneNode = tree(popularity ~ ., data = Spotify_train, control = tree.control(nobs = nrow(Spo
    mincut = 1, minsize = 2))
summary(fit.regrtreeOneNode)
```

```
##
## Regression tree:
## tree(formula = popularity ~ ., data = Spotify_train, control = tree.control(nobs = nrow(Spotify_trai
##     mincut = 1, minsize = 2))
## Variables actually used in tree construction:
## character(0)
## Number of terminal nodes:  1
## Residual mean deviance:  497.3 = 39680000 / 79800
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -33.240 -16.240   1.763   0.000  16.760  66.760
```

```
# plot(fit.regrtreeOneNode) text(fit.regrtreeOneNode, pretty=0)
pred.regrtreeOneNode = predict(fit.regrtreeOneNode, newdata = Spotify_test)
mse.regrtreeOneNode = mean((pred.regrtreeOneNode - Spotify_test$popularity)^2)
mse.regrtreeOneNode  # 498.0025
```

```
## [1] 498.0025
```

In the light of the results achieved, the regression tree is not able to capture variability in the data. A solution is using ensemble methods which can be an effective improvement in the accuracy of the regression models because they tend to reduce the risk of overfitting and can capture complex non-linear relationships between variables.

Random forests is the first ensemble methods we look at. We set the number of variables samples as candidate at each split `mtry` equals to 5, i.e., $m = p/3 = 16/3 = 5$, and number of trees `ntree` equals to 500.

```
## Random Forest (m=p/3=16/3=5) (almost 2 hours)
set.seed(3449)
fit.randforest = randomForest(popularity ~ ., data = Spotify_train, mtry = (ncol(Spotify_train) -
    1)/3, ntree = 500, importance = TRUE)
pred.randforest = predict(fit.randforest, newdata = Spotify_test)
mse.randforest = mean((pred.randforest - Spotify_test$popularity)^2)
mse.randforest  # 245.6213
```

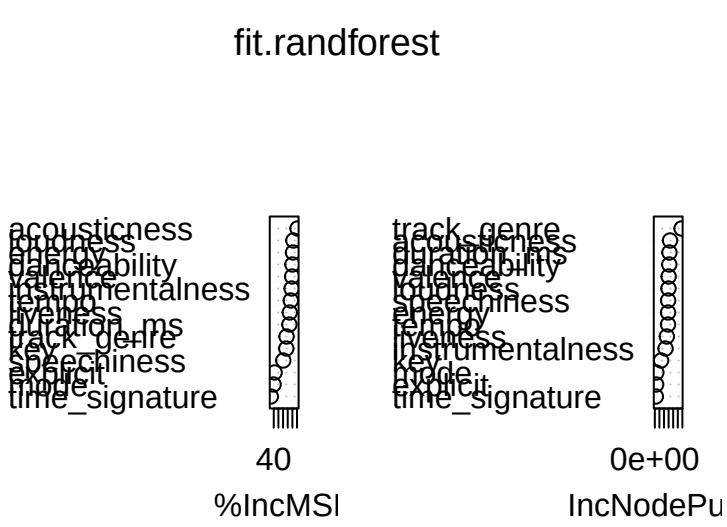
```
## [1] 245.7426
```

```
importance(fit.randforest)  # track_genre, then duration_ms-danceability-acousticness-valence
```

```
##                   %IncMSE IncNodePurity
## duration_ms      107.23923     3182421.6
## explicit          43.72073      285870.0
## danceability     120.13319     3142610.6
```

```
## energy           122.60964     2888665.4
## key               93.98235     1366187.3
## loudness         124.26891     2972826.6
## mode              39.38694      331916.9
## speechiness       80.96713     2895592.1
## acousticness     142.13629     3313332.6
## instrumentalness 116.76147     2333622.3
## liveness         108.79955     2612814.2
## valence          118.99129     3011071.6
## tempo            113.31616     2878539.1
## time_signature    27.40523      223321.1
## track_genre       95.01508     5820104.6
```

```
varImpPlot(fit.randforest)
```



As a result, we get a MSE test equals to 245.6213 which is a significant improvement with respect to the single regression tree. According to the variable importance plot, the variables that contribute most to predicting the popularity of songs are `track_genre`, `duration_ms`, `danceability`, `acousticness` and `valence`.

The second ensemble method used is bagging. We set the `mtry` equals to the number of predictor variables, i.e., 15, since bagging uses all predictors to grow every tree, and number of trees `ntree` equals to 500.

```
## Bagging (m=p=16) (3 hours)
set.seed(3450)
fit.bagging = randomForest(popularity ~ ., data = Spotify_train, mtry = ncol(Spotify_train) -
    1, ntree = 500, importance = TRUE)
pred.bagging = predict(fit.bagging, newdata = Spotify_test)
mse.bagging = mean((pred.bagging - Spotify_test$popularity)^2)
mse.bagging  # 243.7598
```

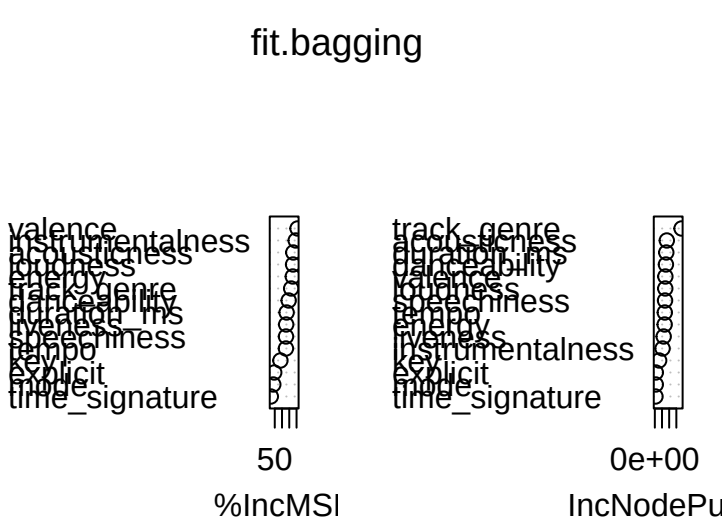
```
## [1] 243.4029
```

```
importance(fit.bagging)  # duration_ms-acousticness
```

```
##                   %IncMSE IncNodePurity
## duration_ms      131.70035     3199410.0
## explicit          46.42132      277052.1
## danceability     145.88934     3091472.8
```

```
## energy           174.07753     2743598.2
## key               89.02695     1208979.6
## loudness         176.10007     2905934.6
## mode              39.99748      268631.6
## speechiness      127.40738     2844154.0
## acousticness     177.73631     3334435.5
## instrumentalness 194.12843     2156989.0
## liveness         128.60250     2466176.9
## valence          206.82321     2935053.8
## tempo            126.90747     2777227.4
## time_signature    22.83464      182226.9
## track_genre      162.88779     7444185.1
```

```
varImpPlot(fit.bagging)
```



We get a slightly improvement in MSE test (i.e., 243.7598) with respect to random forests. Again, according to the variable importance plot, the variables that contribute most to predicting the popularity of songs are `duration_ms` and `acousticness`.

The last ensemble method used is boosting. It requires an hyperparameter tuning for number of trees `n.trees`, interaction depth `interaction.depth` and shrinkage parameter `shrinkage`, that is performed by using 5-fold cross-validation. As a first step, we divided the dataset into training set (50%), validation set (20%) and test set (30%):

```
## Boosting
set.seed(4)
ntrain = nrow(Spotify_train)
train2val = sample(1:ntrain, 0.14 * nrow(Spotify_train), replace = FALSE)
Spotify_val = Spotify_train[train2val, ]
Spotify_trainCV = Spotify_train[-train2val, ]
```

Fit a basic boosting model on training set:

```
basic <- gbm(popularity ~ ., data = Spotify_trainCV, distribution = "gaussian", n.trees = 100,
    interaction.depth = 4, shrinkage = 0.1)
```

Then, we start implementing the 5-folds cross validation. We create the grid of hyperparameters to search over:

```
hyper_grid <- expand.grid(n.trees = seq(100, 500, by = 100), interaction.depth = seq(1,
    5, by = 1), shrinkage = c(0.1, 0.01, 0.001), n.minobsinnode = 10)
```

Define the cross-validation method and the evaluation metric: 5-fold cross-validation to evaluate the performance of each set of hyperparameters and using RMSE (root mean squared error) as the evaluation metric.

```
ctrl <- trainControl(method = "cv", number = 5, verboseIter = TRUE, returnResamp = "all",
    savePredictions = "all")
metric <- "RMSE"
```

Train the model (on the validation set) which will perform the hyperparameter tuning using the `gbm` algorithm:

Extract the best hyperparameters and retrain the model on the new training set. The best hyperparameters are `n.trees = 500`, `interaction.depth = 5` and `shrinkage = 0.1`.

```
best_n_trees <- tune$bestTune$n.trees  #500
best_n_trees
```

```
## [1] 500
```

```
best_interaction_depth <- tune$bestTune$interaction.depth  # 5
best_interaction_depth
```

```
## [1] 5
```

```
best_shrinkage <- tune$bestTune$shrinkage  # 0.1
best_shrinkage
```

```
## [1] 0.1
```

```
set.seed(9)
fit.boosting <- gbm(popularity ~ ., data = Spotify_trainCV, distribution = "gaussian",
    n.trees = best_n_trees, interaction.depth = best_interaction_depth, shrinkage = best_shrinkage)
```

Evaluate the performance of the final model on the test set:

```
pred.boosting = predict(fit.boosting, newdata = Spotify_test)
mse.boosting = mean((pred.boosting - Spotify_test$popularity)^2)
mse.boosting  # 354.2969
```

```
## [1] 354.2969
```

Boosting has worst performance with respect to random forests and bagging. In fact, the MSE test is 354.2969.

Barplots for MSE test of all the methods:

```
library(knitr)
Models <- c("Linear Regression", "RegrTree (one node)", "Ridge Regression", "Lasso Regression",
    "RegrTree (not pruned)", "RegrTree (pruned)", "Boosting", "Random Forests", "Bagging")
MSEtest <- c(mse.lin, mse.regrtreeOneNode, mse_ridge, mse_lasso, mse.regrtree, mse.regrtreeprune,
    mse.boosting, mse.randforest, mse.bagging)
MSEtestdf = data.frame(Models, MSEtest)
MSE_table <- kable(MSEtestdf)
MSE_table
```

Models	MSEtest
Linear Regression	511.7771
RegrTree (one node)	498.0025

Models	MSEtest
Ridge Regression	484.9857
Lasso Regression	484.9738
RegrTree (not pruned)	464.1089
RegrTree (pruned)	376.6720
Boosting	354.2969
Random Forests	245.7426
Bagging	243.4029

So with this table we have the MSE value from the worst to the best methods which are as we said before bagging and random forest.

Summary

As we can observe from the results of previous section, the best results are given by the tree/ensemble models. The reason why the tree/ensemble models gave us better results than linear models is to find in the nature of the problem itself. In fact, there is a highly non-linear and complex relationship between the features and the response, then models such as bagging, random forest, boosting outperform classical approaches like linear regression. In details, bagging and random forests outperform with respect to other models in predicting the popularity of songs - more precisely variables such as `duration_ms` and `acousticness` are relevant in this type of analysis.