Statistical Learning Project

Marlon Helbing, Daniele Virzì, Nemanja Ilic

2024-05-13

1 Introduction

This project is a collaborative effort between three students, **Marlon Helbing**, **Daniele Virzì**, **Nemanja Ilic**. It is an academic project that will be graded based on the quality and depth of our analysis. The project aims to apply the concepts and techniques from the **Statistical Learning Course** to a real-world dataset.

1.1 Project goals

The main objectives of this project are:

- **Exploratory Data Analysis (EDA)**: We aim to uncover patterns and trends in the data that could provide insights into what makes a song popular on ***Spotify***.
- **Regression Model**: This model aims to predict a song's popularity, in terms of streams, based on various features present in the dataset.

1.2 Methodology

Our analysis will be divided into the following sections:

1. **Introduction**.
2. **Data Loading**.
3. **Data Cleaning**
4. **Data Preprocessing**.
5. **Exploratory Data Analysis (EDA)**.
6. **Model Building**.
7. **Model Evaluation**.
8. **Model Selection**.
9. **Conclusion**.

1.3 Dataset

- **Most Streamed Spotify Songs 2023**: This dataset, sourced from Kaggle, is a comprehensive collection of the most streamed songs of 2023 as listed on Spotify. It was chosen for our project due to our shared love for music and our domain knowledge in this area. It provides insights into each song's attributes, popularity, and presence on various music platforms. This makes it an excellent resource for our project, as it allows us to explore a wide range of factors that could potentially influence a song's popularity.

The dataset contains the following columns:

1. `track_name`: *Name of the song.*
2. `artist(s)_name`: *Name of the artist(s) of the song.*
3. `artist_count`: *Number of artists contributing to the song.*
4. `released_year`: *Year when the song was released.*
5. `released_month`: *Month when the song was released.*
6. `released_day`: *Day of the month when the song was released.*
7. `in_spotify_playlists`: *Number of Spotify playlists the song is included in.*
8. `in_spotify_charts`: *Presence and rank of the song on Spotify charts.*
9. `streams`: *Total number of streams on Spotify.*
10. `in_apple_playlists`: *Number of Apple Music playlists the song is included in.*
11. `in_apple_charts`: *Presence and rank of the song on Apple Music charts.*
12. `in_deezer_playlists`: *Number of Deezer playlists the song is included in.*
13. `in_deezer_charts`: *Presence and rank of the song on Deezer charts.*
14. `in_shazam_charts`: *Presence and rank of the song on Shazam charts.*
15. `bpm`: *Beats per minute, a measure of song tempo.*
16. `key`: *Key of the song.*
17. `mode`: *Mode of the song (major or minor).*
18. `danceability_%`: *Percentage indicating how suitable the song is for dancing.*
19. `valence_%`: *Positivity of the song's musical content.*
20. `energy_%`: *Perceived energy level of the song.*
21. `acousticness_%`: *Amount of acoustic sound in the song.*
22. `instrumentalness_%`: *Amount of instrumental content in the song.*
23. `liveness_%`: *Presence of live performance elements.*
24. `speechiness_%`: *Amount of spoken words in the song.*

2 Data Loading

As previously stated, the scope of this project is to assess the knowledge we have gained from the **Statistical Learning Course**. Because of this, we were only permitted to utilize the models and techniques covered in the course for our project work; we were not permitted to use any data analysis techniques that were not covered in class or any `Tidyverse` R-packages, like `ggplot` or `ggplot2`.

2.1 Packages required

To analyze this data, we will use the following R packages:

```
library(corrplot)
```

2.2 Loading the dataset

The dataset is imported to R as CSV files.

```
Spotify <- read.csv("~/Desktop/R_project/data/spotify-2023.csv")
```

3 Data Cleaning

In this section, we had performed some data cleaning tasks to ensure that the dataset is ready for analysis. This includes checking and handling appropriately missing values and duplicates.

3.1 Checking the structure of the data

```
# Check the first 5 rows of the dataset to understand the structure of the data
head(Spotify)
```

3.2 Handling missing values

```
# Check how many missing values there are in the dataset
missing_values <- sum(is.na(Spotify))
missing_values
```

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

3.3 Handling duplicates

```
# Check how many duplicates there are in the dataset
duplicates <- sum(duplicated(Spotify))
duplicates
```

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

4 Data Preprocessing

In order to perform the analysis, we need to preprocess the data. This includes changing the datatype of some columns, removing unnecessary columns, and handling categorical values.

```
# Check the structure of the dataset to understand the datatypes
str(Spotify)
```

```
## 'data.frame':    953 obs. of  24 variables:
##  $ track_name          : chr  "Seven (feat. Latto) (Explicit Ver.)" "LALA" "vampire" "Cruel Summer"
##  $ artist.s._name      : chr  "Latto, Jung Kook" "Myke Towers" "Olivia Rodrigo" "Taylor Swift" ...
##  $ artist_count        : int  2 1 1 1 1 2 2 1 1 2 ...
##  $ released_year       : int  2023 2023 2023 2019 2023 2023 2023 2023 2023 2023 ...
##  $ released_month      : int  7 3 6 8 5 6 3 7 5 3 ...
##  $ released_day        : int  14 23 30 23 18 1 16 7 15 17 ...
##  $ in_spotify_playlists: int  553 1474 1397 7858 3133 2186 3090 714 1096 2953 ...
##  $ in_spotify_charts   : int  147 48 113 100 50 91 50 43 83 44 ...
##  $ streams             : chr  "141381703" "133716286" "140003974" "800840817" ...
##  $ in_apple_playlists  : int  43 48 94 116 84 67 34 25 60 49 ...
##  $ in_apple_charts     : int  263 126 207 207 133 213 222 89 210 110 ...
##  $ in_deezer_playlists : chr  "45" "58" "91" "125" ...
##  $ in_deezer_charts    : int  10 14 14 12 15 17 13 13 11 13 ...
##  $ in_shazam_charts    : chr  "826" "382" "949" "548" ...
##  $ bpm                 : int  125 92 138 170 144 141 148 100 130 170 ...
##  $ key                 : chr  "B" "C#" "F" "A" ...
##  $ mode                : chr  "Major" "Major" "Major" "Major" ...
##  $ danceability_.      : int  80 71 51 55 65 92 67 67 85 81 ...
##  $ valence_.           : int  89 61 32 58 23 66 83 26 22 56 ...
##  $ energy_.            : int  83 74 53 72 80 58 76 71 62 48 ...
##  $ acousticness_.      : int  31 7 17 11 14 19 48 37 12 21 ...
##  $ instrumentalness_.  : int  0 0 0 0 63 0 0 0 0 0 ...
##  $ liveness_.          : int  8 10 31 11 11 8 8 11 28 8 ...
##  $ speechiness_.       : int  4 4 6 15 6 24 3 4 9 33 ...
```

4.1 Convert into date datatype

We had the date of release given in three different columns (day,month,year) as *characters* and we wanted to change this into one column and save it as datatype *date*. Typical format for date is '*2022-06-22*'. As a way to achieve this result, we used `paste()`with the parameter `sep = '-'` to aggregate the three columns into a new column , and then we converted the column datatype using `as.Date()` with parameter `format = "%Y-%m-%d"`. We dropped the original three columns since they were no longer relevant.

```
# First we create a new column and store the date in the wanted format
Spotify$released_date <- paste(Spotify$released_year,
                               Spotify$released_month,
                               Spotify$released_day,
                               sep = '-')

# We give it the wanted datatype of DATE
Spotify$released_date <- as.Date(Spotify$released_date,
                                 format = "%Y-%m-%d")
```

```r
# We can directly drop our other 3 columns storing the dates
Spotify <- subset(Spotify, select = -c(released_year,
                                       released_month,
                                       released_day))
```

4.2 Convert into factor datatype

We have noticed that tha columns **key** and **mode** were *characters* and we wanted to convert them into *factors* using `as.factor()`.

```r
# Change columns with categorical values into the right format (i.e. factor)
Spotify$key <- as.factor(Spotify$key)
Spotify$mode <- as.factor(Spotify$mode)
```

4.3 Convert into integer datatype

Moreover, the columns **streams**, **in_deezer_playlists**, and **in_shazam_charts** , unlike the other playlists and charts features, were *characters* so we have converted them into *integers* using `as.integer()`.

```r
# Change columns with categorical values into the right format (i.e. integer)
Spotify$streams <- as.integer(Spotify$streams)
Spotify$in_deezer_playlists <- as.integer(Spotify$in_deezer_playlists)
Spotify$in_shazam_charts <- as.integer(Spotify$in_shazam_charts)
```

4.4 Separating main artist and features

We also wanted to split the column **artist(s)_name** into more columns. One containing the main artist and the other containing the features of the song. Since in the column **artist(s)_name** the names are separated by commas and the main artist stands at the first position, we used the `strsplit()` function with `parameter split = ","` to split the column at each comma. We then have calculated the maximum number of artists in a single cell and padded the vectors with 0 to make them the same length. Then we have converted the list to a dataframe and we have noticed that there were a space as first charachter of each features cell and the name were wrong. So , before merging them to the main dataset, we have cleaned and renamed the features columns. Finally we have merged it into our original dataframe.

```r
# Split the artist name column at each comma
split_col <- strsplit(Spotify$artist.s._name,
                      split = ",")

# Get the maximum number of artists in a single cell
max_artists <- max(sapply(split_col,
                          length))

# Make all the vectors the same length by padding with 0
split_col <- lapply(split_col, function(x) { c(x,
                                               rep(0,
                                                   max_artists - length(x))) })

# Convert the list to a dataframe
split_df <- do.call(rbind,
```

```
                    split_col)

# Check the result
head(split_df)

# Removing the first space of the column from 2 to 8 if there is one
for (i in 2:8) {
  split_df[,i] <- gsub("^\\s+",
                       "",
                       split_df[,i])
}


# Rename the first column to main_artist and the other columns to feature_1 to feature_7
colnames(split_df) <- c('main_artist',
                        paste0('feature_',
                               1:7))

# Merge into our original dataframe
spotify_with_splitted_artists <- cbind(Spotify,
                                       split_df)

# Make a csv file with the new dataset
write.csv(spotify_with_splitted_artists,
          "spotify_with_splitted_artists.csv")

# Check the first 5 rows of the new dataset
head(spotify_with_splitted_artists)
```

5 Exploratory Data Analysis (EDA)

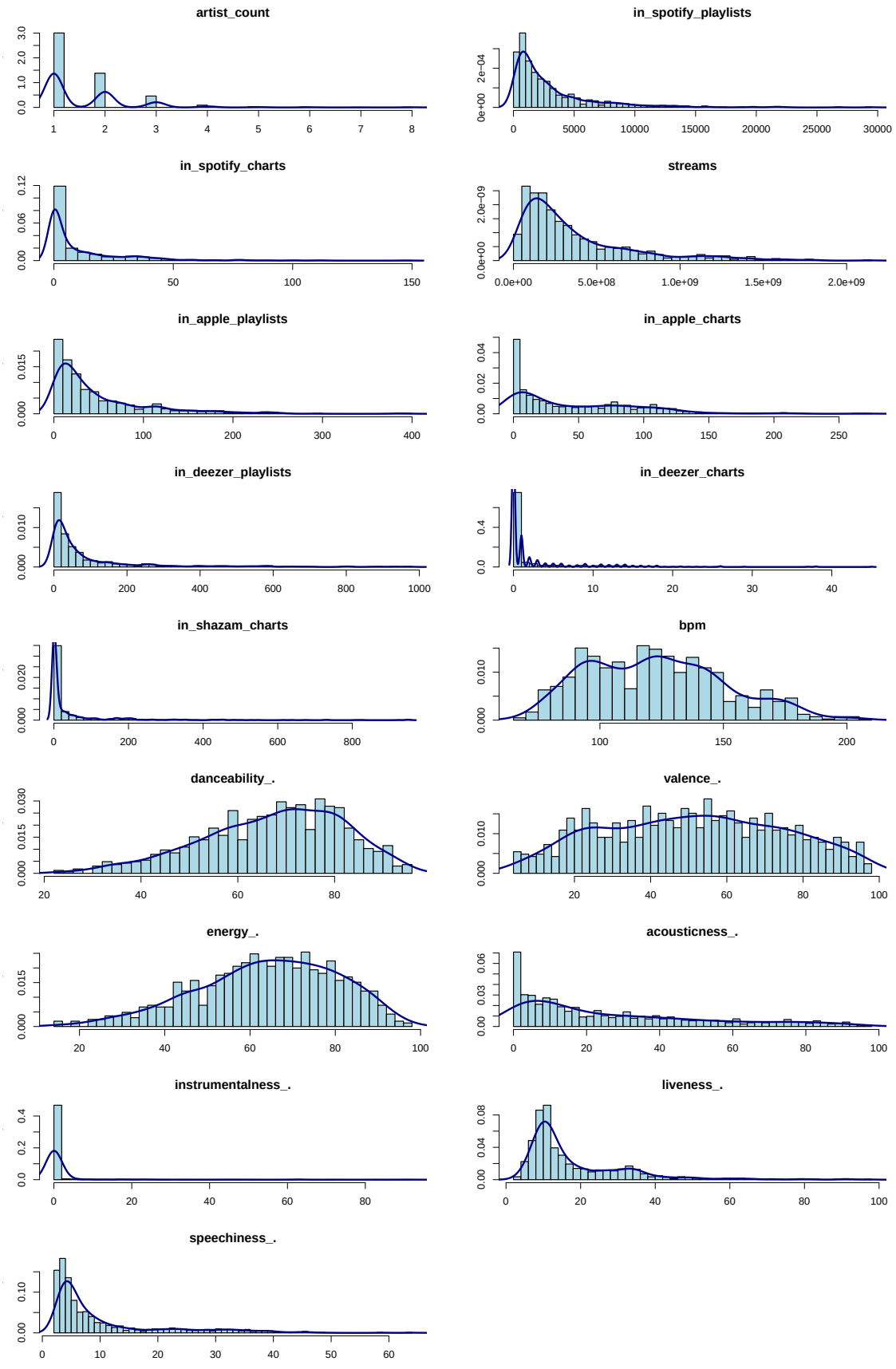
5.1 Data Summary

```
# Check the summary of the dataset to understand the distribution of the data
summary(Spotify)
```

5.1.1 Histograms and Kernel Density

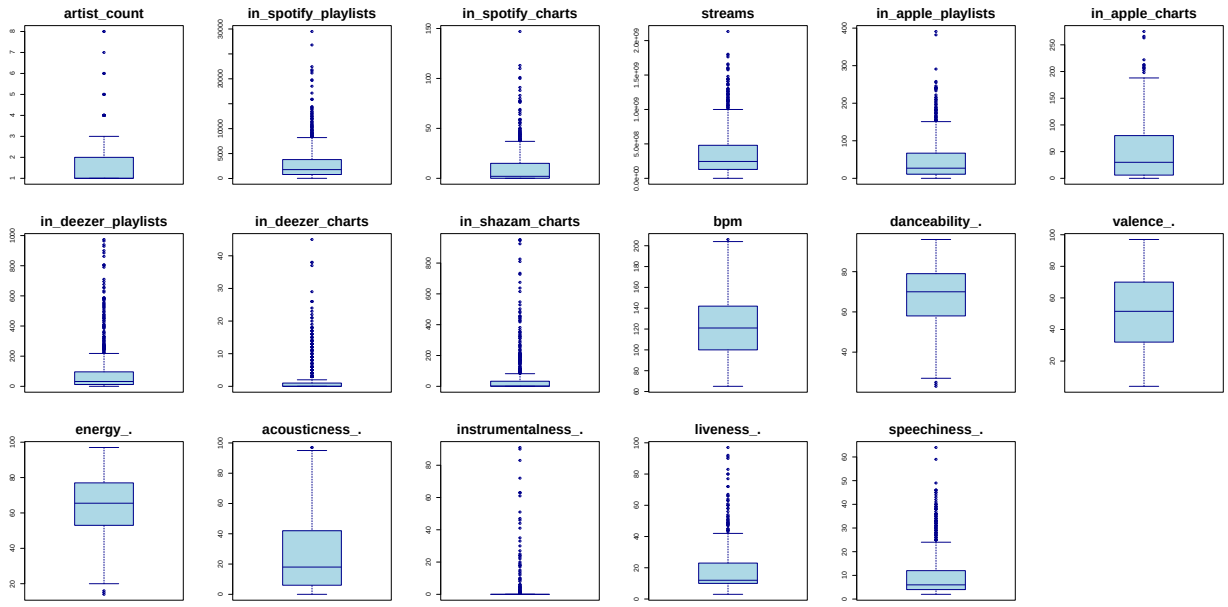
```
# Access ONLY numerical data
numerical_Spotify <- Spotify[sapply(Spotify,
                                    is.numeric)]

# Drop NA if needed
numerical_Spotify <- na.omit(numerical_Spotify)

# Get number of columns
num_of_columns <- length(names(numerical_Spotify))
```

```r
# Set up the graphics window to have a suitable number of rows and columns
par(mfrow = c(9, 2),
    mar = c(3, 3, 3, 3))

# Now, create a histogram and a density plot for each column
for(col in names(numerical_Spotify)) {
  # Create a histogram with probability densities
  hist(numerical_Spotify[[col]],
       breaks = 50,
       prob = TRUE,
       main = paste(col),
       xlab = col,
       ylab = "Density",
       lwd = 0.5,
       col = "lightblue")

  # Compute density data
  density_data <- density(numerical_Spotify[[col]],
                          na.rm = TRUE)

  # Add a kernel density plot (smoothed version of the histogram)
  lines(density_data,
        col = "darkblue",
        lwd = 2)
}
```


artist_count
in_spotify_playlists
in_spotify_charts
streams
in_apple_playlists
in_apple_charts
in_deezer_playlists
in_deezer_charts
in_shazam_charts
bpm
danceability_.
valence_.
energy_.
acousticness_.
instrumentalness_.
liveness_.
speechiness_.

5.1.2 Boxplots

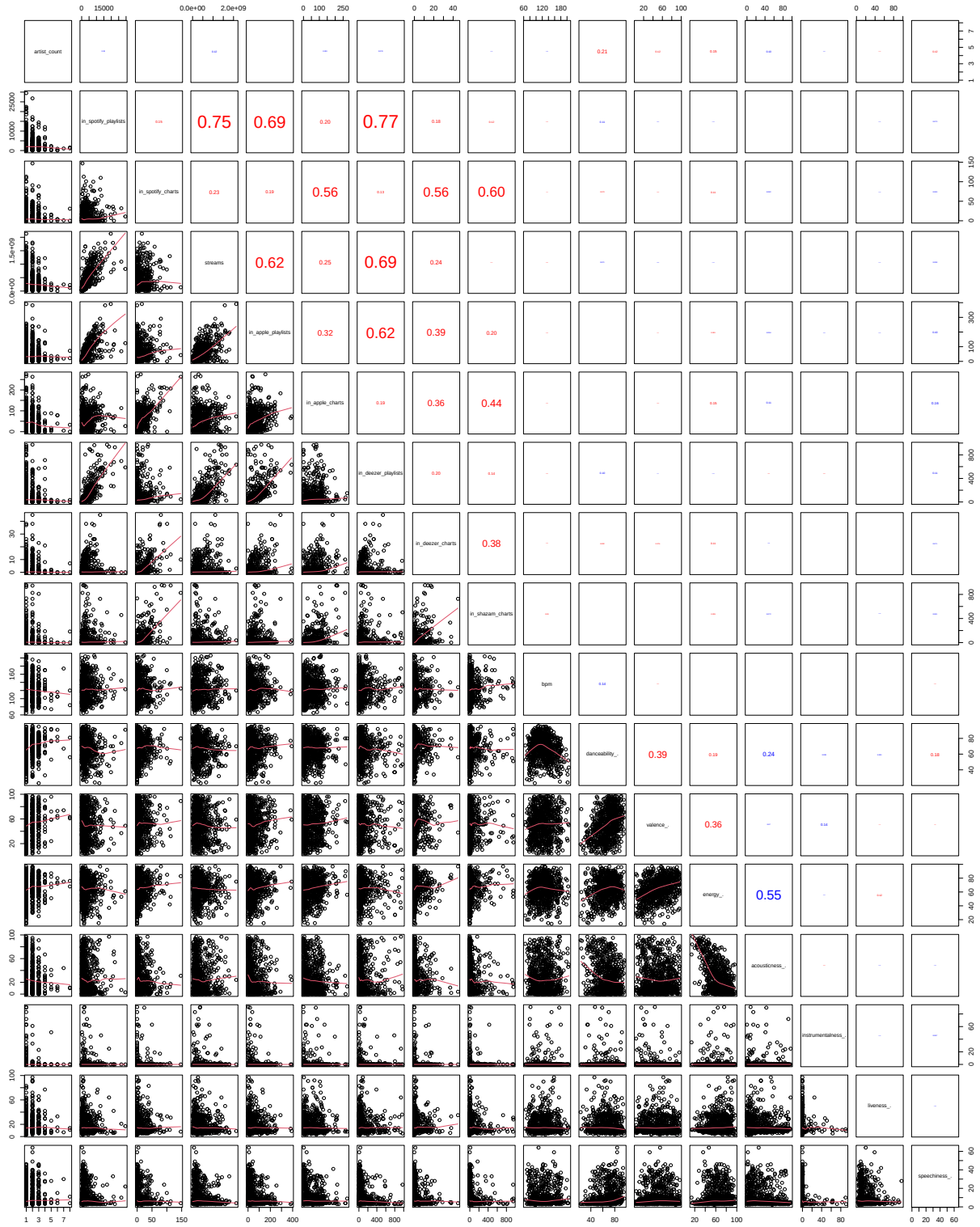
```
# Set up the graphics window to have a suitable number of rows and columns
par(mfrow = c(3, 6),
    mar = c(3, 3, 3, 3))

# Loop through each numerical column
for(col in names(numerical_Spotify)) {
  # Create a boxplot for the column
  boxplot(numerical_Spotify[[col]],
          main = paste(col),
          xlab = col,
          ylab = "Value",
          lwd = 0.5,
          col = "lightblue",
          border = "darkblue",
          cex.main = 2)
}
```



5.1.3 Pairplot

```
# Set up the graphics window
par(mar=c(1,1,1,1))

# Define a panel function that will be used to plot the correlation values
panel.cor <- function(x, y, digits = 2, prefix = "", cex.cor, ...)
{
  usr <- par("usr"); on.exit(par(usr))
  par(usr = c(0, 1, 0, 1))
  r <- cor(x, y)
```

```
  txt <- format(c(abs(r), 0.123456789), digits = digits)[1]
  txt <- paste0(prefix, txt)

  # Set the size of the text
  if(missing(cex.cor)) cex.cor <- 1.0/strwidth(txt)

  # Use color to indicate the sign of the correlation
  col <- ifelse(r < 0, "blue", "red")
  text(0.5, 0.5, txt, cex = cex.cor * abs(r), col = col)
}

# Create a pairplot of the numerical columns
pairs(numerical_Spotify, upper.panel=panel.cor, lower.panel=panel.smooth)
```


artist_count
in_spotify_playlists
in_spotify_charts
streams
in_apple_playlists
in_apple_charts
in_deezer_playlists
in_deezer_charts
in_shazam_charts
bpm
danceability_.
valence_.
energy_.
acousticness_.
instrumentalness_.
liveness_.
speechiness_.
0.75
0.69
0.20
0.77
0.23
0.19
0.56
0.13
0.56
0.60
0.62
0.25
0.69
0.24
0.32
0.62
0.39
0.20
0.19
0.36
0.44
0.20
0.38
0.21
0.39
0.19
0.24
0.36
0.55