# Spotify Genre Classification- Statistical Learning Project

Felice Francario

## Contents

## 1 Introduction

The objective of this project is to analyze the diverse landscape of music genres and develop a classification model capable of predicting the genre of a song based on its audio features. Music, as a universal form of expression, encompasses a vast array of genres, each characterized by distinct musical elements, themes, and emotions. Understanding and classifying these genres is not only valuable for music enthusiasts and artists but also holds significant implications for music recommendation systems, content curation, and genre-specific analysis.

The proliferation of digital music platforms and streaming services has led to an exponential growth in the availability and accessibility of music across genres. However, manually categorizing and organizing this vast musical repertoire is a daunting task. Automated classification models offer a promising solution by leveraging machine learning algorithms to discern patterns and relationships among audio features and genre labels.

In this project, we embark on a comprehensive exploration of music genres, drawing insights from a dataset of songs spanning diverse genres and cultural influences. By analyzing key audio features such as tempo, energy, danceability, and instrumentalness, we aim to uncover the underlying characteristics that distinguish one genre from another.

Our primary goal is to develop a robust classification model capable of accurately predicting the genre of a song based solely on its audio features.

## 1.1 Dataset

The dataset was downloaded from Kaggle in a CSV format.This dataset provides comprehensive information about Spotify tracks encompassing a diverse collection of 125 genres.

```
data<-read.csv("train.csv")
dim(data)
```

```
## [1] 114000     21
```

The dataset has already been cleaned and contains no missing values

```
sum(is.na(data))
```

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

The dataset comprises multiple columns, each representing distinctive audio features associated with individual tracks.

```
names<-colnames(data)
names
```

```
##  [1] "Unnamed..0"       "track_id"         "artists"          "album_name"
##  [5] "track_name"       "popularity"       "duration_ms"      "explicit"
##  [9] "danceability"     "energy"           "key"              "loudness"
## [13] "mode"             "speechiness"      "acousticness"     "instrumentalness"
## [17] "liveness"         "valence"          "tempo"            "time_signature"
## [21] "track_genre"
```

The first 5 descriptive columns were removed as they are irrelevant for the objective of the project

```
#removing unnamed..0,track_id,artists,album_name,track_name columns
df<-data[,-c(1,2,3,4,5)]
colnames(df)
```

```
##  [1] "popularity"       "duration_ms"      "explicit"         "danceability"
##  [5] "energy"           "key"              "loudness"         "mode"
##  [9] "speechiness"      "acousticness"     "instrumentalness" "liveness"
## [13] "valence"          "tempo"            "time_signature"   "track_genre"
```

### 1.1.1 Descrpition of the features

1. **popularity**: A score indicating how popular a track is on Spotify (ranging from 0 to 100).
2. **duration__ms**: The duration of a track in milliseconds.
3. **explicit:** Indicates whether a track contains explicit content (True or False). Explore Audio Features: This dataset includes various audio features associated with each track. Here are some notable ones:
4. **Danceability**: Danceability measures how suitable a track is for dancing, ranging from 0 to 1. Tracks with high danceability scores are more energetic and rhythmic, making them ideal for dancing.

5. **Energy**: Energy represents intensity and activity within a song on a scale from 0 to 1. Tracks with high energy tend to be more fast-paced and intense.
6. **Loudness**: Loudness indicates how loud or quiet an entire song is in decibels (dB). Positive values represent louder songs while negative values suggest quieter ones.
7. **Key**: Key refers to different musical keys assigned integers ranging from 0-11, with each number representing a different key. Knowing the key can provide insights into the mood and tone of a song.
8. **mode**: The tonal mode of the track, represented by an integer value (0 for minor, 1 for major).
9. **speechiness**: A score ranging from 0 to 1 that represents the presence of spoken words in a track.
10. **acousticness**: A score ranging from 0 to 1 that represents the extent to which a track possesses an acoustic quality.
11. **instrumentalness**: A score ranging from 0 to 1 that represents the likelihood of a track being instrumental.
12. **liveness** A score ranging from 0 to 1 that represents the presence of an audience during the recording or performance of a track.
13. **Valence**: Valence measures the musical positiveness conveyed by a track, ranging from 0 to 1. High valence values indicate more positive or happy tracks, while lower values suggest more negative or sad ones.
14. **Tempo**: Tempo is the speed or pace of a song in beats per minute (BPM). It gives an idea about how fast or slow a track is.
15. **time_signature**: The number of beats within each bar of the track.

### 1.1.2 Response variable

16. **track_genre**: The genre of the track. This will be the **response variable**.

```r
unique(df$track_genre)
```

```
##   [1] "acoustic"          "afrobeat"          "alt-rock"
##   [4] "alternative"       "ambient"           "anime"
##   [7] "black-metal"       "bluegrass"         "blues"
##  [10] "brazil"            "breakbeat"         "british"
##  [13] "cantopop"          "chicago-house"     "children"
##  [16] "chill"             "classical"         "club"
##  [19] "comedy"            "country"           "dance"
##  [22] "dancehall"         "death-metal"       "deep-house"
##  [25] "detroit-techno"    "disco"             "disney"
##  [28] "drum-and-bass"     "dub"               "dubstep"
##  [31] "edm"               "electro"           "electronic"
##  [34] "emo"               "folk"              "forro"
##  [37] "french"            "funk"              "garage"
##  [40] "german"            "gospel"            "goth"
##  [43] "grindcore"         "groove"            "grunge"
##  [46] "guitar"            "happy"             "hard-rock"
##  [49] "hardcore"          "hardstyle"         "heavy-metal"
##  [52] "hip-hop"           "honky-tonk"        "house"
##  [55] "idm"               "indian"            "indie-pop"
##  [58] "indie"             "industrial"        "iranian"
##  [61] "j-dance"           "j-idol"            "j-pop"
##  [64] "j-rock"            "jazz"              "k-pop"
##  [67] "kids"              "latin"             "latino"
##  [70] "malay"             "mandopop"          "metal"
##  [73] "metalcore"         "minimal-techno"    "mpb"
##  [76] "new-age"           "opera"             "pagode"
##  [79] "party"             "piano"             "pop-film"
##  [82] "pop"               "power-pop"         "progressive-house"
##  [85] "psych-rock"        "punk-rock"         "punk"
##  [88] "r-n-b"             "reggae"            "reggaeton"
##  [91] "rock-n-roll"       "rock"              "rockabilly"
##  [94] "romance"           "sad"               "salsa"
##  [97] "samba"             "sertanejo"         "show-tunes"
## [100] "singer-songwriter" "ska"               "sleep"
## [103] "songwriter"        "soul"              "spanish"
## [106] "study"             "swedish"           "synth-pop"
## [109] "tango"             "techno"            "trance"
## [112] "trip-hop"          "turkish"           "world-music"
```

Since there are too many different genres, only the 9 most important genres were selected for this analysis.

```
genres<-c('classical','country','electronic','hip-hop','jazz','rock','pop','blues','reggae')

#Filter the dataframe with selected genres
df_filtered<-df[df$track_genre %in% genres,]

y<-as.factor(df_filtered$track_genre)

df_filtered$track_genre <- as.factor(df_filtered$track_genre)
unique(y)
```

```
## [1] blues      classical  country    electronic hip-hop    jazz       pop
## [8] reggae     rock
## Levels: blues classical country electronic hip-hop jazz pop reggae rock
```

The dataset contains exactly 1000 tracks for each genre.

```
table(y)
```

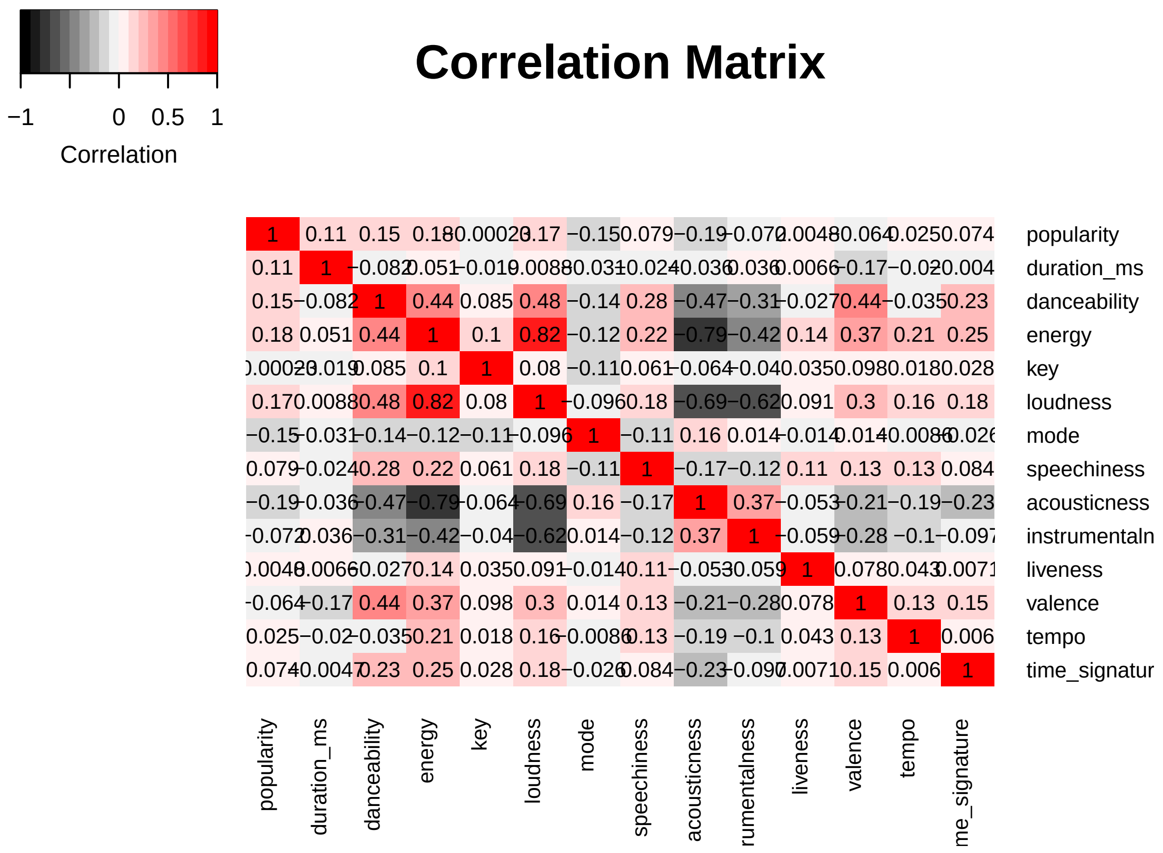
```
## y
##      blues  classical    country electronic    hip-hop       jazz        pop
##       1000       1000       1000       1000       1000       1000       1000
##     reggae       rock
##       1000       1000
```

## 2 Exploratory Data Analysis

In this section, we conduct exploratory data analysis (EDA) to gain insights into the distributional characteristics of various features in our dataset. EDA plays a crucial role in understanding the underlying patterns and relationships among different variables before proceeding with further analysis or modeling. We will view the relationship between the various features and the distribution of the various features based on the track genre, our response variable.

## 2.1 Correlation between Features

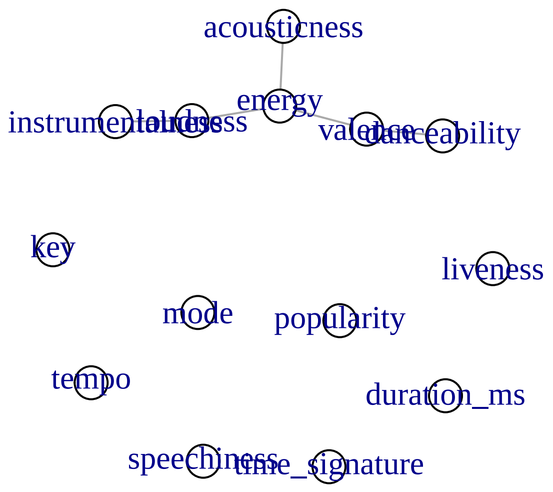
```r
#considering only numeric features
numeric_df <- df_filtered[, sapply(df_filtered, is.numeric)]
cor_matrix<-cor(numeric_df)
variable_names <- colnames(cor_matrix)
heatmap.2(cor_matrix,
          trace = "none", # Turn off row/column dendrogram
          col = colorRampPalette(c("black", "white", "red"))(20),
          main = "Correlation Matrix",
          dendrogram="none",
          #xlab = "Variables", ylab = "Variables",
          key = TRUE, # Add color key for the gradient
          key.title = NA, # Remove the default key title
          key.xlab = "Correlation", # Add x-axis label for the key
          key.ylab = NULL, # Remove the default y-axis label for the key
          Rowv = variable_names,  # Set row order
          Colv = variable_names,  # Set column order
          density.info = "none", # Turn off density plot
          symkey = FALSE, # Do not plot a symmetric key
          add.expr = {
            # Add text annotations for correlation values
            for (i in 1:nrow(cor_matrix)) {
              for (j in 1:ncol(cor_matrix)) {
                text(i ,15- j , format(cor_matrix[i, j], digits = 2),
                     col = "black", cex = 0.8)
              }
            }
          },
          #width = 6, # Adjust width of the plot
          #height = 6, # Adjust height of the plot
          cexRow = 0.8, # Adjust text size for rows
          cexCol = 0.8 # Adjust text size for columns

)
```



From the correlation matrix , we can notice that danceability, energy and loudness are significantly correlated negatively with acousticness, instrumentalness while they are positively correlated with each other. In particular there is a very high correlation of 0.82 between energy and loudness.

From the following you can see the how above mentioned features that are partially correlated with each other while the rest of the features are basically conditionaly independent with repect to each other.

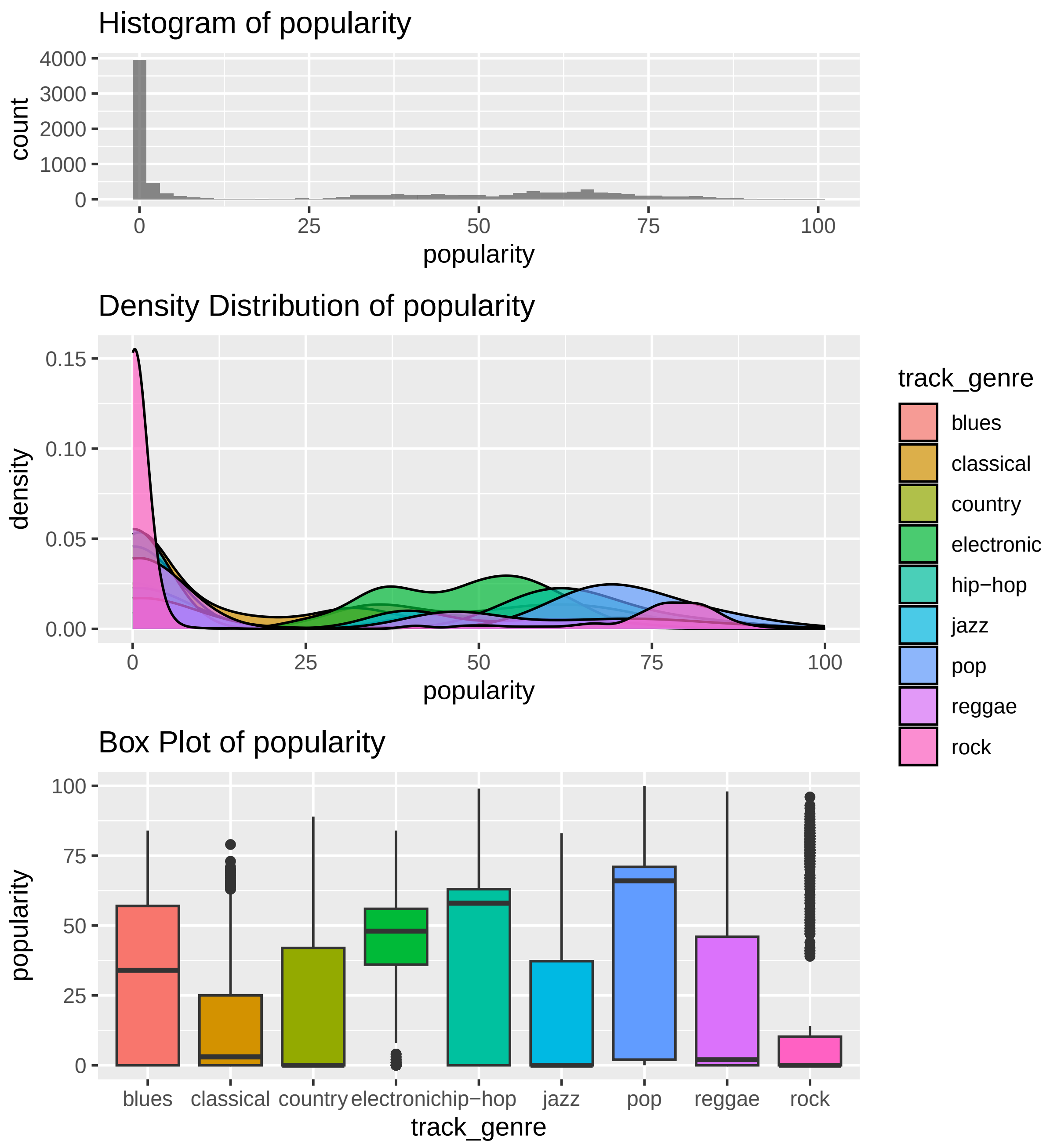
```r
S <- var(numeric_df)
R <- -cov2cor(solve(S))

thr <- 0.3

G <- abs(R)>thr
diag(G) <- 0

#Gi <- as(G, "igraph")
#plot(Gi)
Gi <- graph_from_adjacency_matrix(as.matrix(G), mode = "undirected", weighted = NULL, diag = FALSE)
plot(Gi,vertex.color="white")
```



## 2.2 Feature Distributions

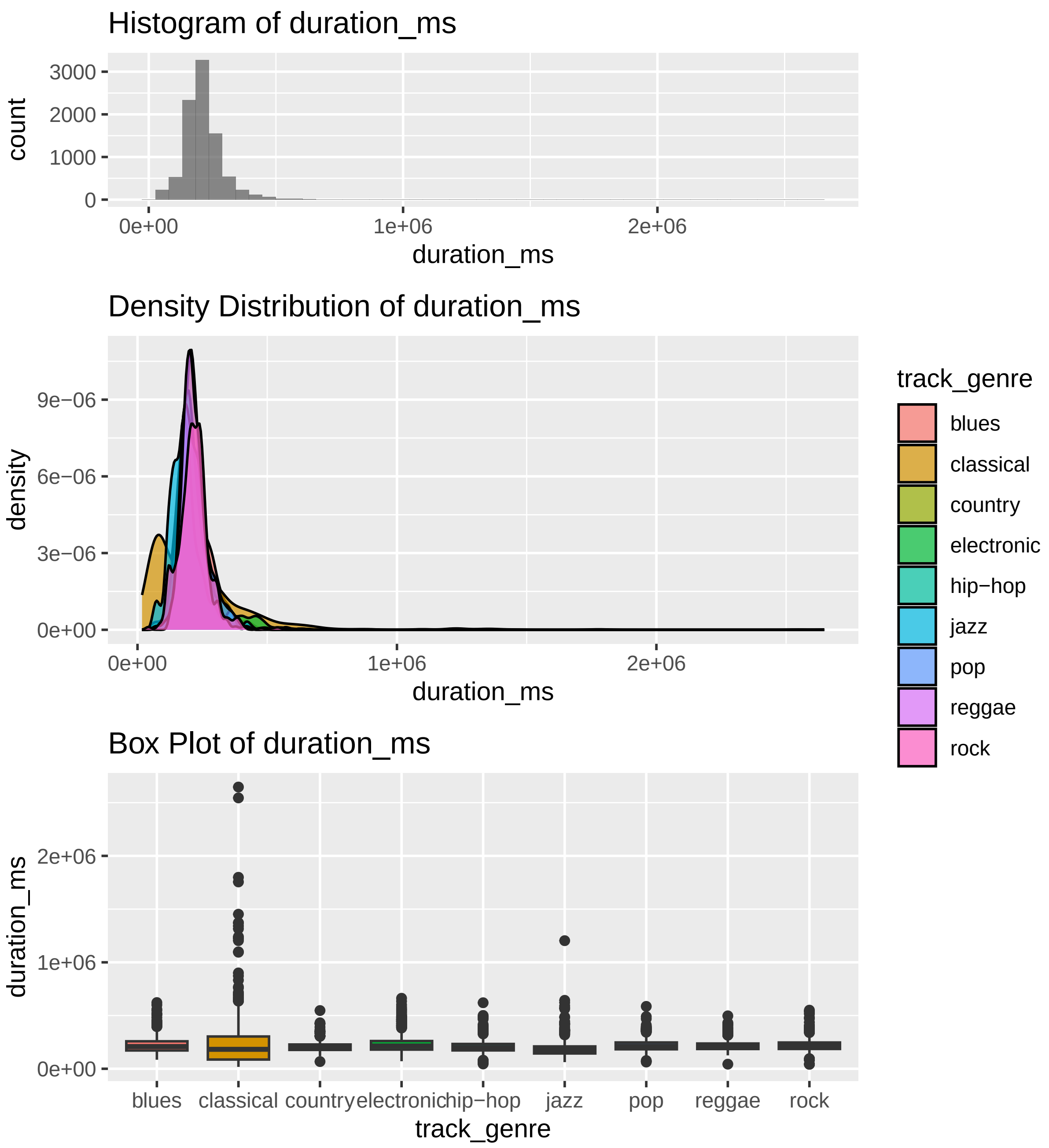
In this subsection we examine the distribution of individual features and their relationship with the target variable. For each feature, we generate histograms, density plots, and box plots to visualize the distribution across different classes.

1. **Popularity**



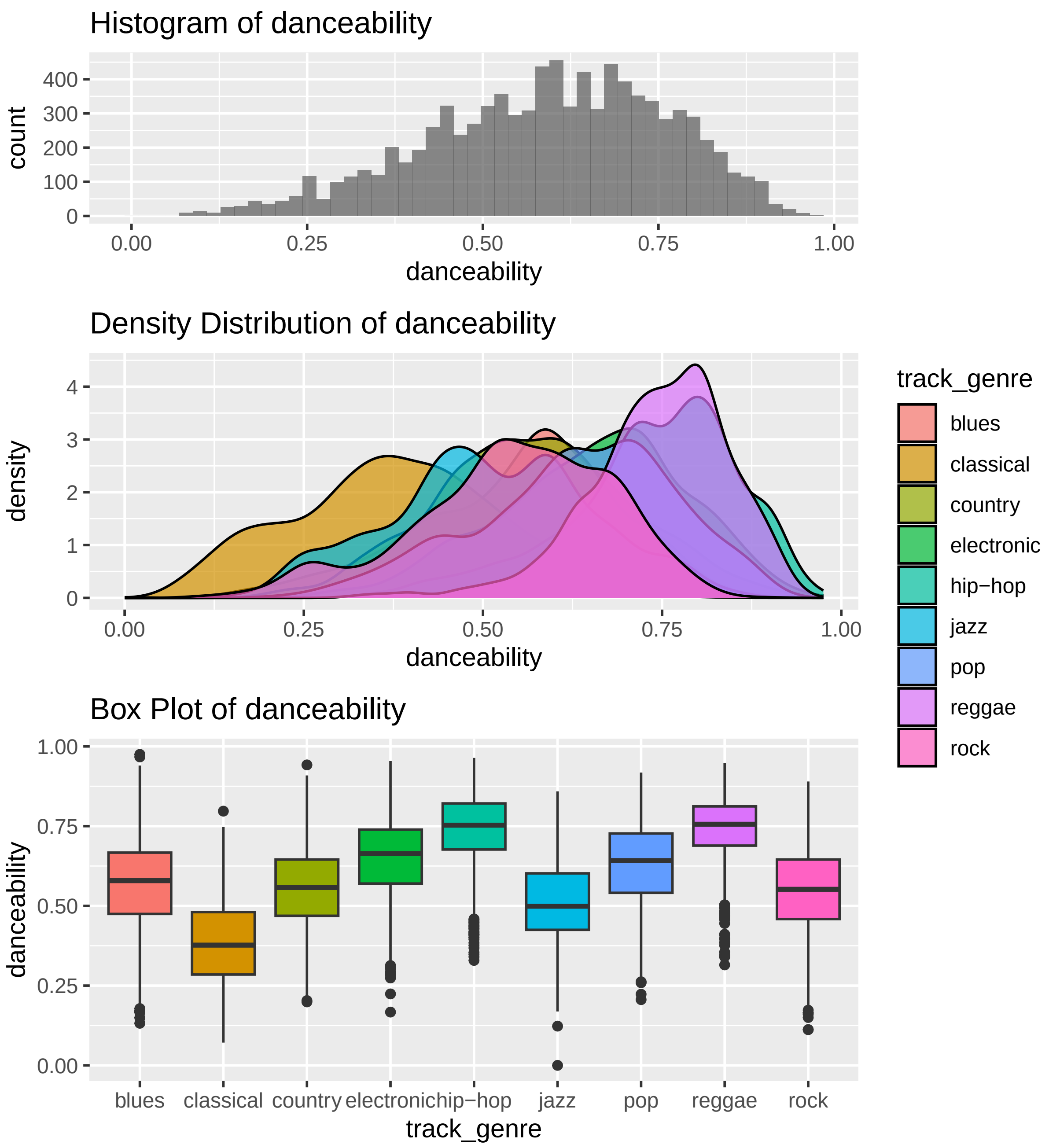
Classical,country, jazz, reggae and rock seem to generally be the least popular genres, with the majority of the rock tracks with a particularly low popularity score , but with a lot of outliers with a very high score. The most popular genres are pop,hip-hop and electronic. Hip-hop and pop have the highest spread in popularity values,with a a very high median value but left skewed distributions.

2. **Duration-ms**



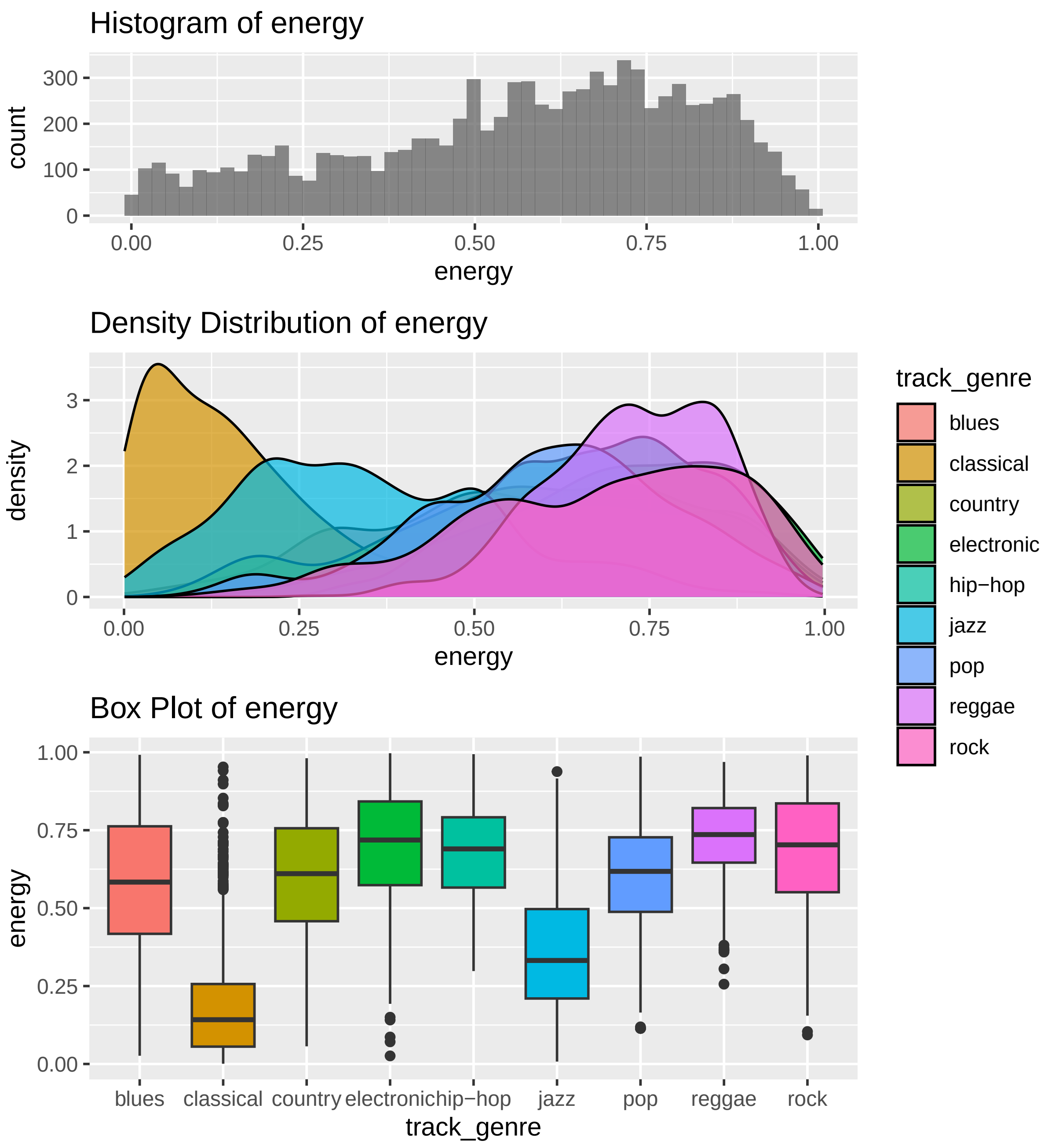
All the genres have a relatively similar distribution except the classical genre which has the highest variance and lowest median value. The classical genre has the highest range of track duration with a relatively high concentration of short tracks but with a huge amount of outliers with extremely high track duration.

3. **Danceability**



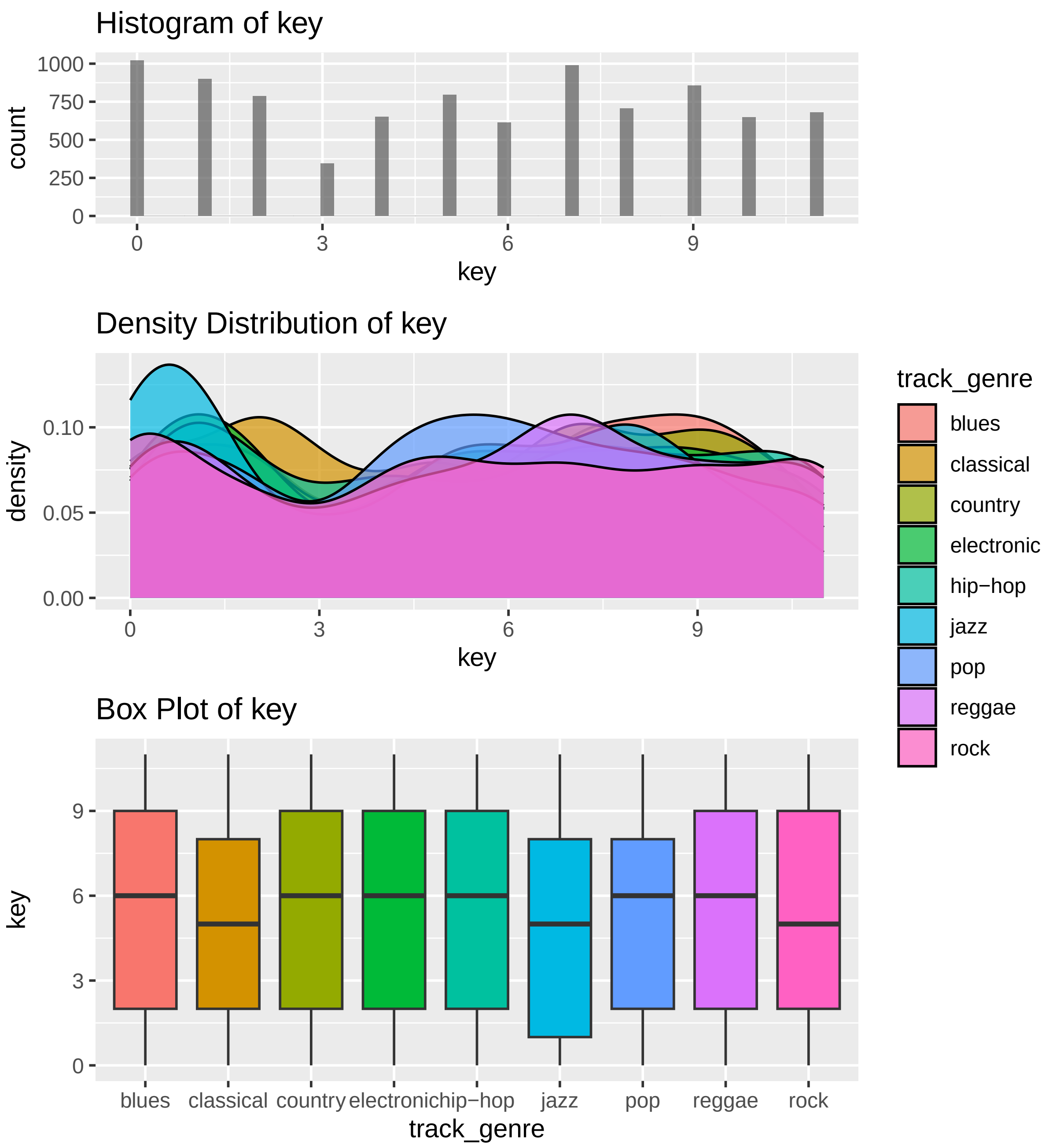
Hip-Hop and Reggae have the heighest dancebility scores, with Hip-Hop in general having a slightly higher concentration of tracks near the very maximum value of danceability. Classical and Jazz obviously have the lowest danceability score.

4. **Energy**



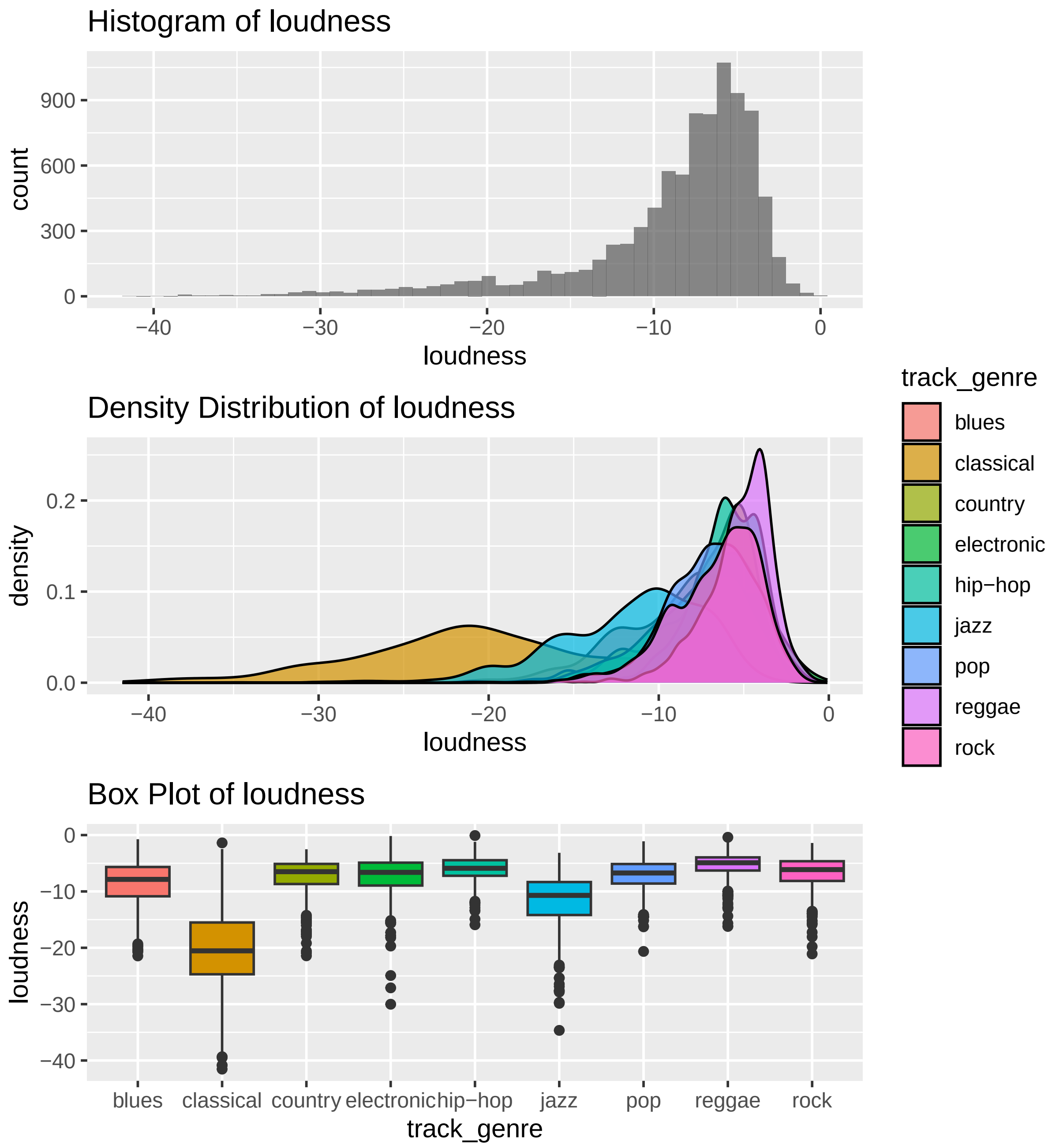
The distributions of the energy score are generally similar to danceability but with even lower scores for classical and jazz tracks. The main differences are that electronic tracks have a higher energy score than hip-hop and rock has a generally much higher energy score compared to danceability.

6. **Key**



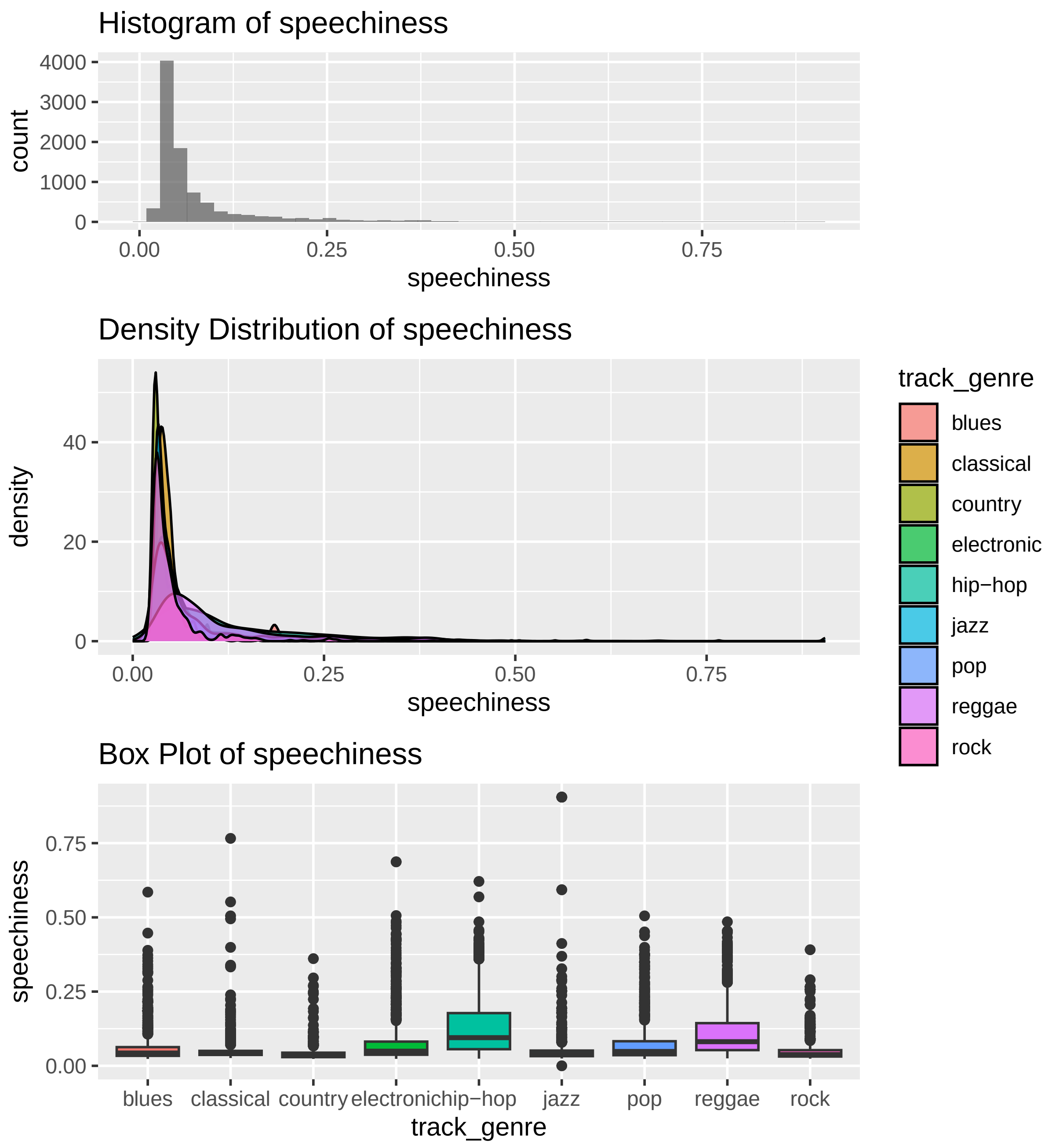
Visually the feature "Key" doesn't have a big effect on the differences between the genres.

7. **Loudness**



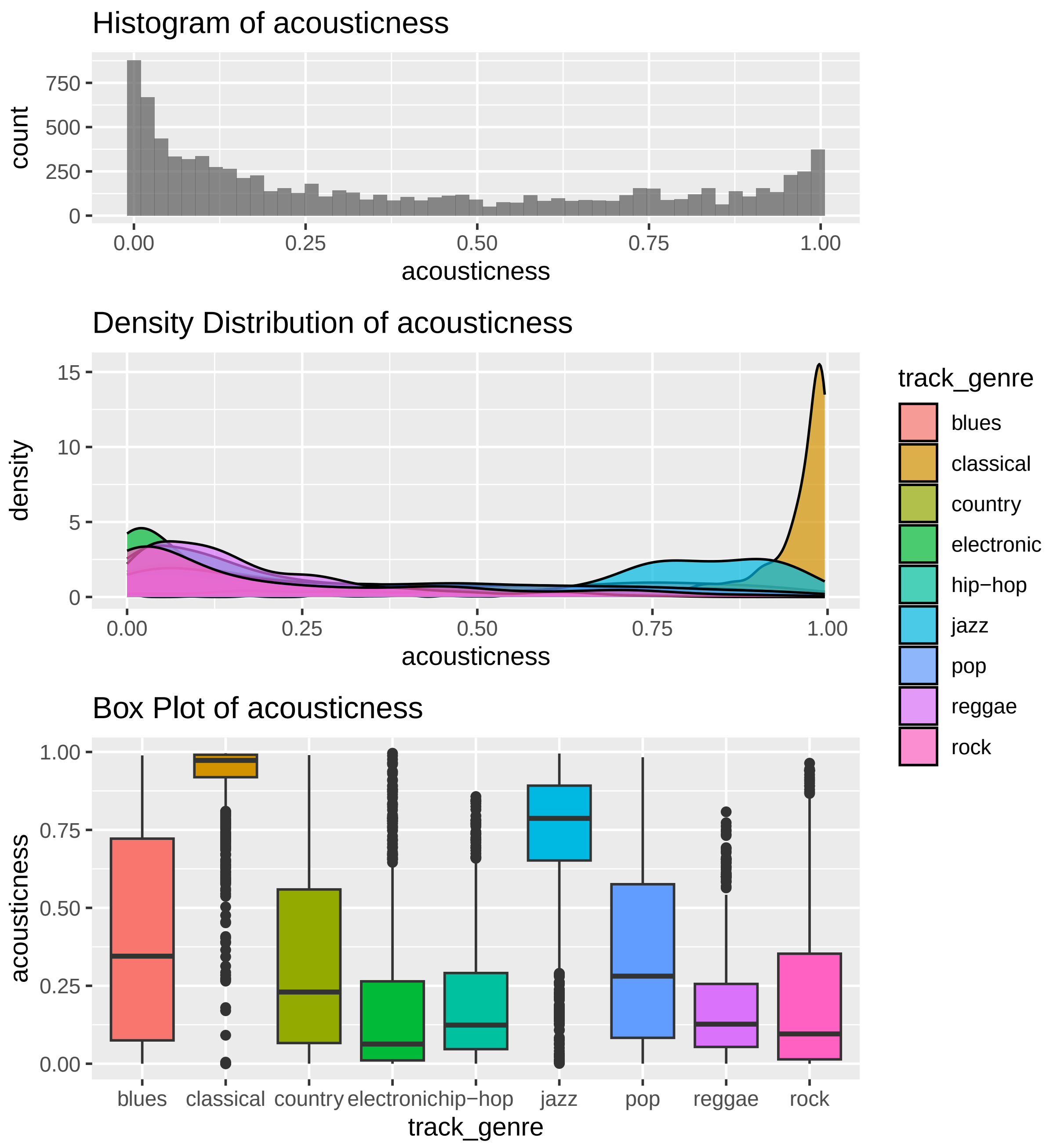
Reggae and Hip-hop are generally the loudest, while jazz and classical are generally quieter.This feature has a 0.82 correlation with energy , so the similarity in distributions is expected.

8. **Speechiness**



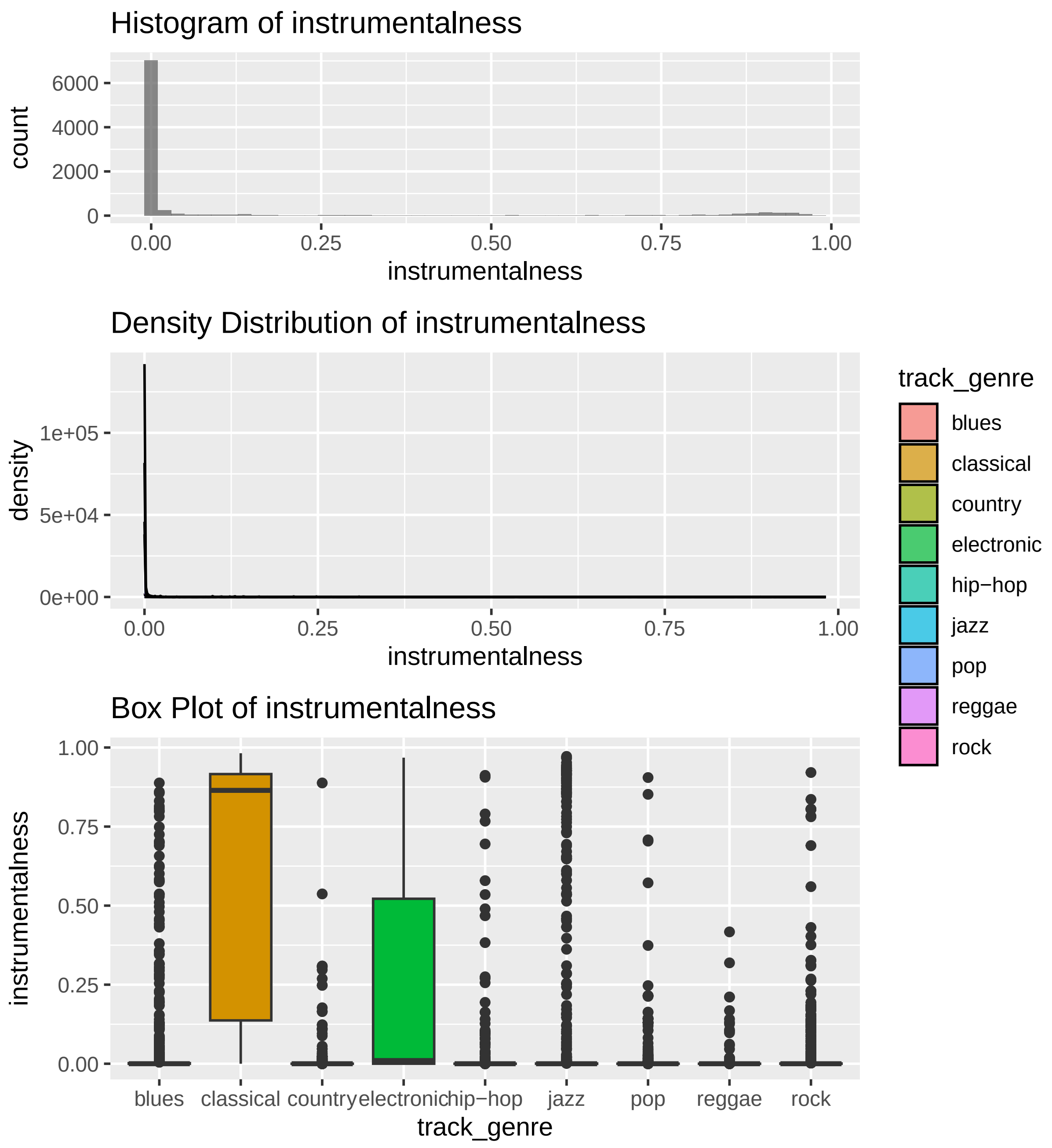
All the genres have a majority of their tracks concentrated at a very low score, with country in particular having the lowest variance while Hip-hop has the highest median value and variance.

9. **Acousticness**



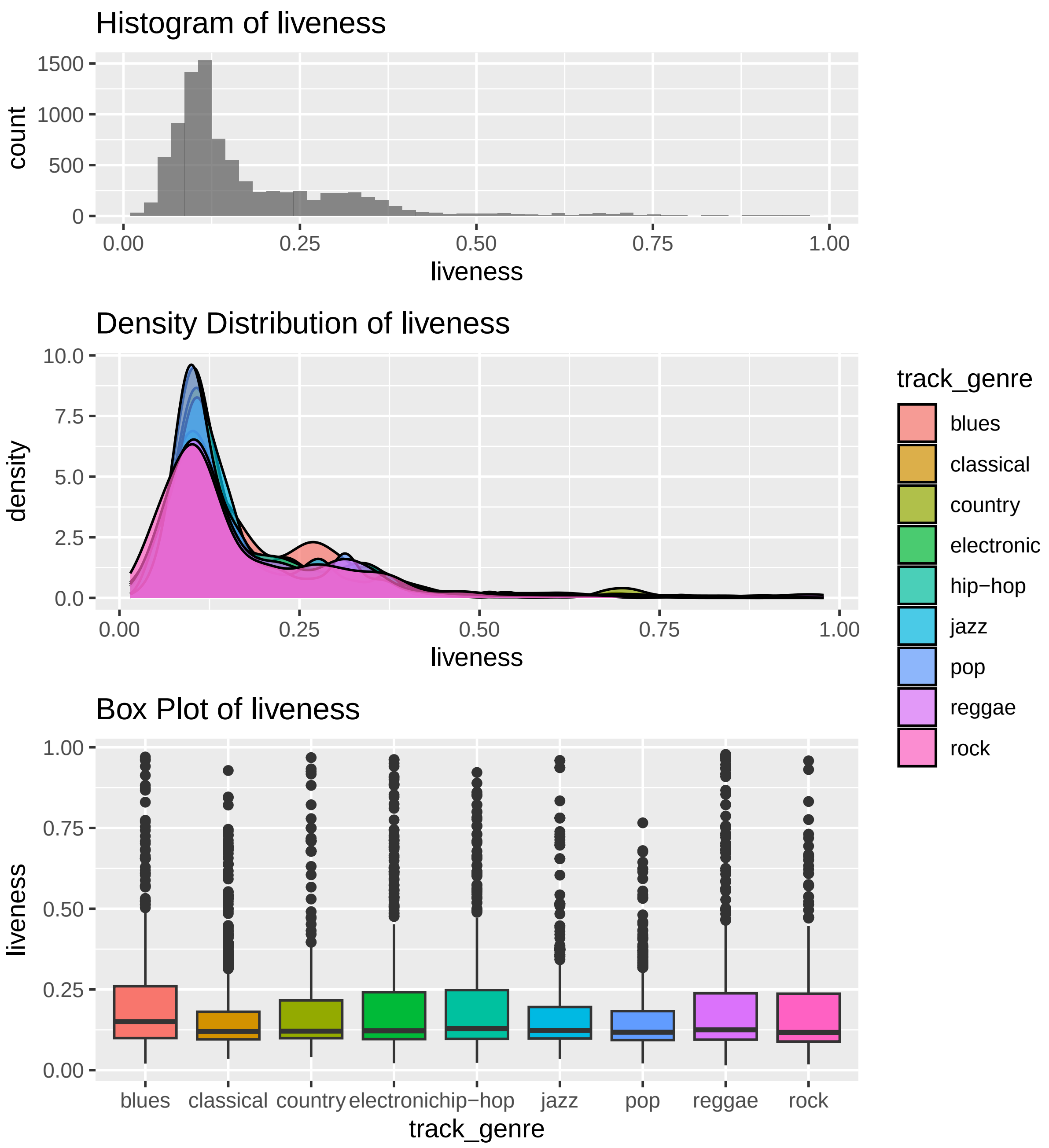
This feature has a high variance for all the genres with the exception of the classical genre which is highly concentrated close to the maximum value of 1. Elcetronic, hip-hop and reggae have in general the lowest acousticness score.

10. **Instrumentalness**



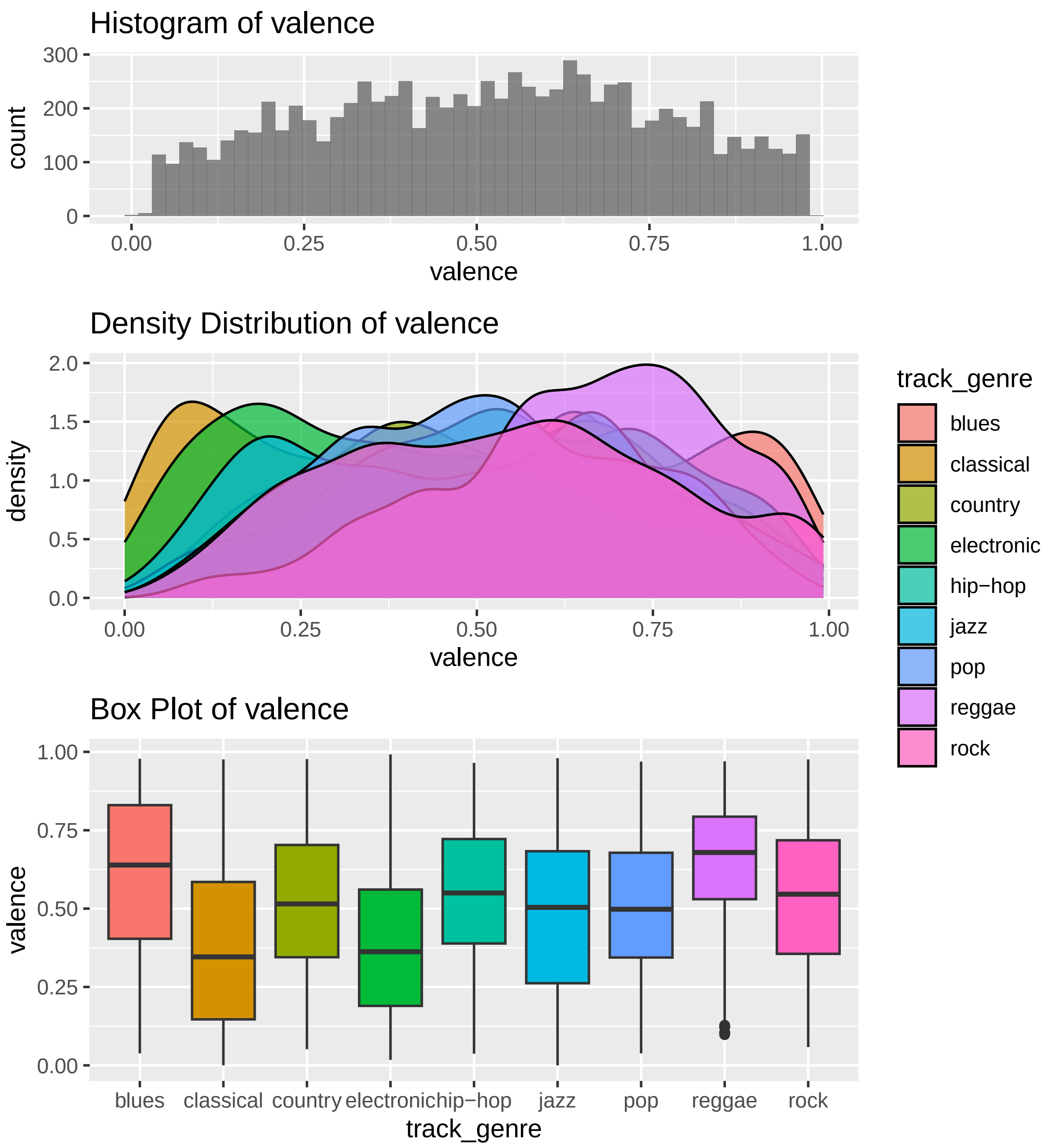
Aside from classical and electronic, all the genres have an instrumentalness score close to 0. Classical is heavily left skewed distribution with a high variance while electronic has a very right skewed distribution with its median value close to 0 but a very high variance.

11. **Liveness**



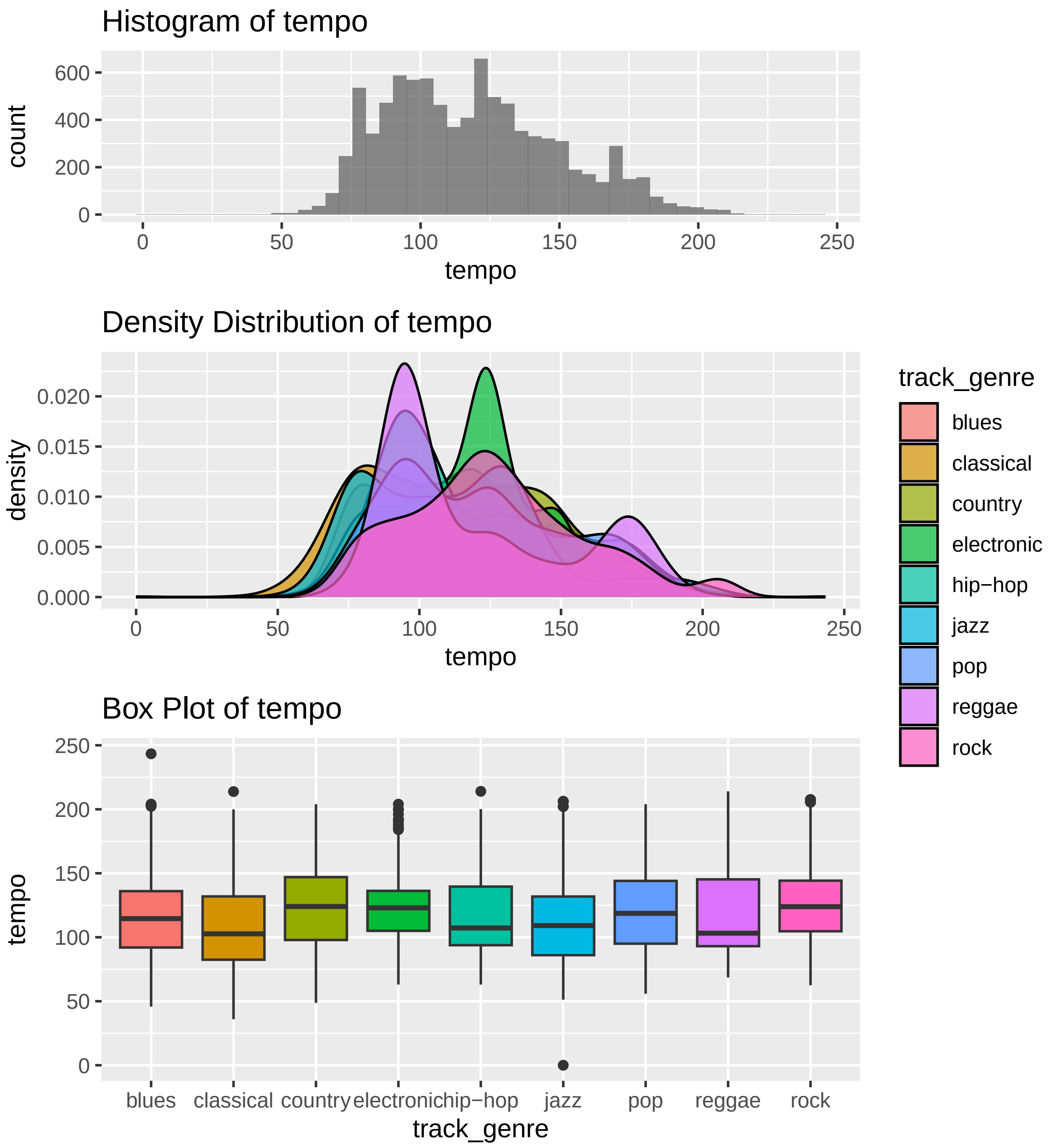
Visually there doesn't seem to be much difference in the liveness distribution of all the genres.They all have a right skewed distribution with median around 0.12 with the exception of the blues genres which has a slightly higher median value than all the other genres. There a lot of outliers for each genre.

12. **Valence**



Reggae and blues have the highest valence scores, while classical and electroncic have the lowest.This indicated that reggae and blues generally have a more positive vibe while classical and electronic are generally more negative.

13. **Tempo**



The genre distribution are multimodal and have peaks around distinct values.In particular Reggae and electronic have the most identifiable distributions.

## 3 Models

In this section, we will apply various tuned models trained on a training set for a multi class classification task evaluated on a separate test set. To create the training and test set, the dataset was divided with 80/20 ratio.Here we create a model matriX X and scale it to a normal distribution N(0,1) to reduce the effect of the scale of a feature on predictions.

```r
set.seed(1)
X<-model.matrix(track_genre~.,data=df_filtered)[,-1]

numeric_vars <- sapply(df_filtered[,-16], is.numeric)
categorical_vars <- !numeric_vars


# Scale numerical variables in the model matrix
X_scaled<-X
X_scaled[, numeric_vars] <- scale(X[, numeric_vars])


train <- sample(1:nrow(X), 4*nrow(X)/5)
test <- (-train)
#X.train<-X[train,]
#X.test<-X[test,]
y.train<-y[train]
y.test <- y[test]

X_scaled.train<-X_scaled[train,]
X_scaled.test<-X_scaled[test,]
```

## 3.1 Multinomial Logistic regression

In this subsection we will apply different variants of a Multinomial regression with feature selection or regularization. Multinomial logistic regression is a powerful technique for modeling relationships between multiple categorical outcome variables and predictor variables. The multinomial logistic regression model extends the binary logistic regression model to handle multiple classes. The conditional probability of observing class $k$ given predictor variables $\mathbf{x}$ is given by:

$$P(Y = k|\mathbf{x}) = \frac{e^{\beta_{0k} + \beta_k^T \mathbf{x}}}{1 + \sum_{l=1}^{K-1} e^{\beta_{0l} + \beta_l^T \mathbf{x}}}$$

where $K$ is the number of classes, $\beta_{0k}$ are the intercepts for each class, $\beta_k$ are the coefficient vectors for each class, and $\mathbf{x}$ is the vector of predictor variables.

Next, we discuss the implementation details of multinomial logistic regression in R, leveraging popular libraries such as `glmnet` or `nnet`. In particular, we explore the `multinom()` function provided by the `nnet` package in R. The `multinom()` function fits multinomial logistic regression models using the maximum likelihood estimation method and can handle both nominal and ordinal response variables.

### 3.1.1 Base Model

```r
multinom_model <- multinom(track_genre ~ ., data = df_filtered[train,])
#summary(multinom_model)
```

```r
predictions <- predict(multinom_model, newdata = df_filtered[test,], type = "class")
confusion_matrix <- table(predictions, y.test)
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
print(paste("Accuracy:", accuracy))
```

```
## [1] "Accuracy: 0.495555555555556"
```

We get and accuracy of around 50% relatively good improvement compared to the to the value of the 11.1% accuracy of a random guess.

```r
# Display results
print("Confusion Matrix:")
```

```
## [1] "Confusion Matrix:"
```

```r
print(confusion_matrix)
```

```
##              y.test
## predictions  blues classical country electronic hip-hop jazz pop reggae rock
##   blues         53         9      13          7       6    9  13     12   15
##   classical      5       177       2          3       0   17   0      0    0
##   country       11         0      89         11       4   20  13      7   46
##   electronic    12         0       4         95      14   12  10      5   10
##   hip-hop        3         0       3         32      81    1  34     46    2
##   jazz          38        16      23          3       0  115  12      0   12
##   pop           26         3      11         19      39    8  76     20   19
##   reggae        14         0       7          9      54    5  26    131   12
##   rock          27         2      45         15       4   10  14      4   75
```

Looking at the Confusion Matrix we and the overall precision values in the table below , we can notice that the classical genre is classified with a relatively high precision of 86.7% while the rest are all below 50% with the exception of electronic and jazz with 58.6% and 52.8% respectively. We can notice that The Blues genre is most confused by jazz, hip-hop is most confused by Reggae and rock is most confused by country.

```r
# Print or use the metrics_df data frame
print(metrics_df)
```

```
##                    Precision    Recall  F1_Score Specificity
## Class: blues       0.3868613 0.2804233 0.3251534   0.7195767
## Class: classical   0.8676471 0.8550725 0.8613139   0.1449275
## Class: country     0.4427861 0.4517766 0.4472362   0.5482234
## Class: electronic  0.5864198 0.4896907 0.5337079   0.5103093
## Class: hip-hop     0.4009901 0.4009901 0.4009901   0.5990099
## Class: jazz        0.5251142 0.5837563 0.5528846   0.4162437
## Class: pop         0.3438914 0.3838384 0.3627685   0.6161616
## Class: reggae      0.5077519 0.5822222 0.5424431   0.4177778
## Class: rock        0.3826531 0.3926702 0.3875969   0.6073298
```

#### 3.1.2 Subset Selection

In this subsection, we delve into subset selection methods for model comparison, specifically focusing on backward subset selection using AIC, forward subset selection using AIC, backward selection using BIC, and forward selection using BIC. Subset selection methods are a class of techniques used to identify the most informative subset of predictor variables for a given statistical model. These methods aim to strike a balance between model complexity and predictive performance, ultimately leading to models that are more interpretable and generalize better to unseen data. The Akaike Information Criterion (AIC) and Bayesian

Information Criterion (BIC) are widely used model selection criteria that balance goodness of fit with model complexity. They are defined as follows:

$$\mathrm{AIC} = -2\log(L) + 2k$$

$$\mathrm{BIC} = -2\log(L) + k\log(n)$$

where $L$ is the likelihood of the model, $k$ is the number of parameters in the model, and $n$ is the number of observations. The AIC and BIC penalize models with higher complexity, encouraging the selection of simpler models that adequately explain the data.

- **Backward selection with AIC**

Backward subset selection is a stepwise approach that starts with the full model and iteratively removes predictors that contribute the least to the model's fit. In backward subset selection with AIC, predictors are sequentially removed until the AIC is minimized, indicating the best subset of predictors.

```
model <- multinom(track_genre ~ ., data = df_filtered[train,])

# Perform stepwise selection using step AIC function
step_model <- step(model, direction = "backward")
```

```
step_model$anova
```

```
##    Step Df Deviance Resid. Df Resid. Dev      AIC
## 1       NA       NA      7072   20199.25 20455.25
## 2 - key  8 13.37702      7080   20212.63 20452.63
```

Only the key feature was removed. Testing the new model trained on the new model trained on a subset removing the key feature, the overall accuracy on a test set actually slightly decreased by around 1%.

```
predictions <- predict(step_model, newdata = df_filtered[test,], type = "class")
confusion_matrix <- table(predictions, y.test)
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
print(paste("Accuracy:", accuracy))
```

```
## [1] "Accuracy: 0.488888888888889"
```

From the confusion Matrix and the table of metrics, we can see that the main genre negatively affected by removing the key feature is the Blues genre with a reduction in precision of nearly 4% and its increase in confusion with jazz.

```
print("Confusion Matrix:")
```

```
## [1] "Confusion Matrix:"
```

```
print(confusion_matrix)
```

```
##              y.test
## predictions  blues classical country electronic hip-hop jazz pop reggae rock
##   blues         45         9      13          6       6    9  13     12   15
##   classical      5       177       2          3       0   17   0      0    0
##   country       10         0      86         11       4   19  13      7   45
##   electronic    12         0       4         97      14   12  10      5   10
##   hip-hop        3         0       3         31      81    1  33     47    2
##   jazz          45        16      25          3       0  115  13      0   14
##   pop           26         3      11         19      38    9  76     21   18
##   reggae        14         0       6          8      55    5  26    129   13
##   rock          29         2      47         16       4   10  14      4   74
```

```r
# Print or use the metrics_df data frame
print(metrics_df)
```

```
##                   Precision    Recall  F1_Score Specificity
## Class: blues      0.3515625 0.2380952 0.2839117   0.7619048
## Class: classical  0.8676471 0.8550725 0.8613139   0.1449275
## Class: country    0.4410256 0.4365482 0.4387755   0.5634518
## Class: electronic 0.5914634 0.5000000 0.5418994   0.5000000
## Class: hip-hop    0.4029851 0.4009901 0.4019851   0.5990099
## Class: jazz       0.4978355 0.5837563 0.5373832   0.4162437
## Class: pop        0.3438914 0.3838384 0.3627685   0.6161616
## Class: reggae     0.5039062 0.5733333 0.5363825   0.4266667
## Class: rock       0.3700000 0.3874346 0.3785166   0.6125654
```

- **Forward selection with AIC**

Forward subset selection is another stepwise approach that begins with an empty model and iteratively adds predictors that most improve the model fit. Similar to backward selection, forward selection utilizes the AIC criterion to determine the optimal subset of predictors by adding variables until the AIC is minimized.

```r
output <- capture.output({
#Forward feature selection(AIC)
selected_features <- c()
min_AIC <- Inf

while (TRUE) {
  remaining_features <- setdiff(colnames(df_filtered), c("track_genre", selected_features))

  if (length(remaining_features) == 0) {
    break
  }

  best_AIC <- Inf
  best_feature <- NULL

  for (feature in remaining_features) {
    current_model <- multinom(track_genre ~ ., data = df_filtered[train, c(selected_features, feature,
    current_AIC <- AIC(current_model)

    if (current_AIC < best_AIC) {
      best_AIC <- current_AIC
      best_feature <- feature
    }
  }

  if (best_AIC < min_AIC) {
    min_AIC <- best_AIC
    selected_features <- c(selected_features, best_feature)
  } else {
    break  # No improvement in AIC, stop the loop
  }
}


})
```

The same subset was obtained with forward selection, only key was removed.

```
print(selected_features)
```

```
##  [1] "acousticness"     "danceability"     "instrumentalness" "popularity"
##  [5] "speechiness"      "valence"          "loudness"         "mode"
##  [9] "energy"           "duration_ms"      "explicit"         "tempo"
## [13] "time_signature"   "liveness"
```

- **Backward selection with BIC**

Backward subset selection with the Bayesian Information Criterion (BIC) follows a similar approach to AIC-based selection but utilizes the BIC criterion instead. BIC penalizes model complexity more severely than AIC, often resulting in more parsimonious models. In backward subset selection with BIC, predictors are eliminated iteratively until the BIC is minimized, leading to a model with optimal predictive performance.

```
model <- multinom(track_genre ~ ., data = df_filtered[train,])

# Perform stepwise selection using step function
step_model <- step(model, direction = "backward",k=log(length(train)),trace=0)
```

```
step_model$anova
```

```
##    Step Df Deviance Resid. Df Resid. Dev      AIC
## 1        NA       NA      7072   20199.25 21336.13
## 2 - key  8 13.37702      7080   20212.63 21278.45
```

```
step_model$coefnames
```

```
##  [1] "(Intercept)"      "popularity"       "duration_ms"      "explicitTrue"
##  [5] "danceability"     "energy"           "loudness"         "mode"
##  [9] "speechiness"      "acousticness"     "instrumentalness" "liveness"
## [13] "valence"          "tempo"            "time_signature"
```

The same subset was selected, with only the “key” feature removed.

- **Forward selection with BIC**

Forward subset selection with BIC starts with an empty model and adds predictors based on the BIC criterion. Similar to the forward selection with AIC, this method iteratively adds predictors until the BIC is minimized, resulting in a subset of predictors that optimally balances model fit and complexity.

```r
output<-capture.output({
selected_features <- c()
min_BIC <- Inf

while (TRUE) {
  remaining_features <- setdiff(colnames(df_filtered), c("track_genre", selected_features))

  if (length(remaining_features) == 0) {
    break
  }

  best_BIC <- Inf
  best_feature <- NULL

  for (feature in remaining_features) {
    current_model <- multinom(track_genre ~ ., data = df_filtered[train, c(selected_features, feature,
    current_BIC <- BIC(current_model)

    if (current_BIC < best_BIC) {
      best_BIC <- current_BIC
      best_feature <- feature
    }
  }

  if (best_BIC < min_BIC) {
    min_BIC <- best_BIC
    selected_features <- c(selected_features, best_feature)
  } else {
    break  # No improvement in BIC, stop the loop
  }
}
})
```

```r
length(selected_features)
```

```
## [1] 12
```

```r
print(selected_features)
```

```
##  [1] "acousticness"     "danceability"     "instrumentalness" "popularity"
##  [5] "speechiness"      "valence"          "loudness"         "mode"
##  [9] "energy"           "duration_ms"      "explicit"         "tempo"
```

With forward subset selction with BIC , 3 features were removed: key,liveness and mode. We can get a preliminary idea of the relative unimportance of these 3 features in the multiclass classification

```
multinom_model_bic <- multinom(track_genre ~ ., data = df_filtered[train,c(selected_features,"track_gen

# Make Predictions
predictions <- predict(multinom_model_bic, newdata = df_filtered[test,c(selected_features,"track_genre")

# Evaluate the Model
confusion_matrix <- table(predictions, y.test)
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
```

```
print(paste("Accuracy:", accuracy))
```

```
## [1] "Accuracy: 0.493888888888889"
```

The accuracy on the model applied on the test set is slightly better than the previous subset selection of only removing key,but is still slighlty less by an insignificant 0.5% compared to the full model.

```
print("Confusion Matrix:")
```

```
## [1] "Confusion Matrix:"
```

```
print(confusion_matrix)
```

```
##              y.test
## predictions  blues classical country electronic hip-hop jazz pop reggae rock
##   blues         53        10      14          6       8    9  16     13   12
##   classical      5       178       2          3       0   17   0      0    0
##   country       12         0      96         12       4   22  13      5   46
##   electronic    11         0       4         93      12   12  10      5   11
##   hip-hop        3         0       3         31      80    1  32     47    2
##   jazz          41        15      26          2       0  113  12      0   14
##   pop           23         2      12         22      37    9  71     19   18
##   reggae        15         0       7          7      58    5  27    130   13
##   rock          26         2      33         18       3    9  17      6   75
```

```
# Print or use the metrics_df data frame
print(metrics_df)
```

```
##                   Precision    Recall  F1_Score Specificity
## Class: blues      0.3758865 0.2804233 0.3212121   0.7195767
## Class: classical  0.8682927 0.8599034 0.8640777   0.1400966
## Class: country    0.4571429 0.4873096 0.4717445   0.5126904
## Class: electronic 0.5886076 0.4793814 0.5284091   0.5206186
## Class: hip-hop    0.4020101 0.3960396 0.3990025   0.6039604
## Class: jazz       0.5067265 0.5736041 0.5380952   0.4263959
## Class: pop        0.3333333 0.3585859 0.3454988   0.6414141
## Class: reggae     0.4961832 0.5777778 0.5338809   0.4222222
## Class: rock       0.3968254 0.3926702 0.3947368   0.6073298
```

Overall by rounding the accuracies of all 3 models seen till now, no difference was found in accuracy,while it looks like mode,key and liveness features have an insignificant effect on the multiclass classification.

### 3.1.3 Ridge Regularization

Ridge regression is a type of linear regression that introduces regularization to mitigate multicollinearity and reduce the variance of the parameter estimates. It adds a penalty term to the least squares objective function, constraining the magnitude of the coefficient estimates. In this subsection, we apply ridge regularization using the glmnet package in R, setting the tuning parameter $\lambda$ to control the strength of regularization.

Ridge regression minimizes the following objective function:

$$\text{minimize}\left(||\mathbf{y}-\mathbf{X}\beta||_2^2+\lambda||\beta||_2^2\right)$$

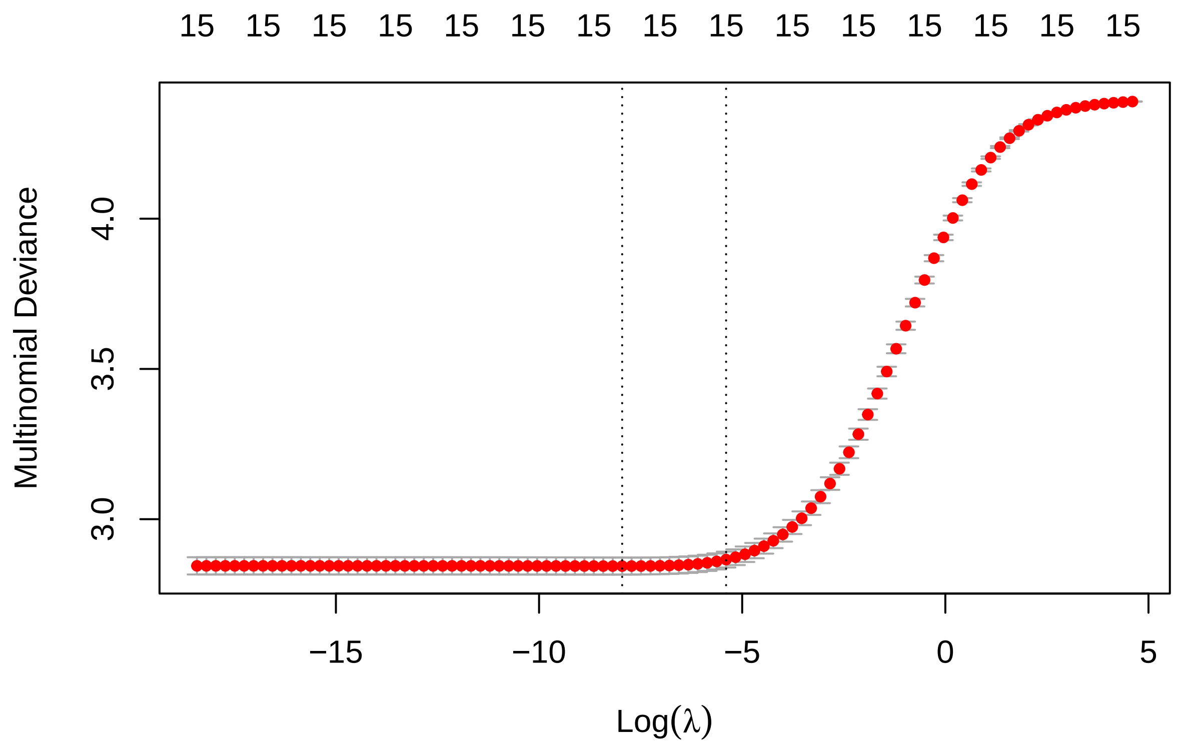
where $||\mathbf{y}-\mathbf{X}\beta||_2^2$ is the residual sum of squares, $||\beta||_2^2$ is the L2-norm penalty term, and $\lambda$ is the regularization parameter. The L2-norm penalty term shrinks the coefficient estimates towards zero, reducing their variance and mitigating overfitting.

The glmnet package provides efficient algorithms for fitting generalized linear models with elastic-net regularization, which combines ridge (L2) and lasso (L1) penalties. However, by setting the parameter alpha to 0, we exclusively apply ridge regularization in our model.

Through ridge regularization, we aim to stabilize the parameter estimates and improve the generalization performance of our model. We evaluate the effectiveness of ridge regularization by tuning the regularization parameter $\lambda$ by cross-validation. By incorporating ridge regularization into our modeling pipeline, we strive to strike a balance between model complexity and predictive accuracy.

```r
fit_ridge<-glmnet(X_scaled[train,],y[train],family="multinomial",alpha=0)
#plot(fit_ridge)
```

```r
lambda_interval<-10^seq(2,-8,length=100)
cvfit_ridge<-cv.glmnet(X_scaled[train,],y[train],family="multinomial",alpha=0,lambda=lambda_interval)
plot(cvfit_ridge)
```



```
#No Regularization
pred <- predict(fit_ridge, s = 0, newx = X_scaled[test, ],
                exact = TRUE, type="class",x = X_scaled[train, ], y = y[train])
pred_accuracy<-mean(pred==y.test)
pred_accuracy
```

```
## [1] 0.4938889
```

```
bestlam <- cvfit_ridge$lambda.min
bestlam
```

```
## [1] 0.0003511192
```

```
ridge_pred=predict(cvfit_ridge,newx=X_scaled[test,],type="class", s="lambda.min")
ridge_pred_accuracy<-mean(ridge_pred==y.test)
ridge_pred_accuracy
```

```
## [1] 0.4944444
```

```
conf_matrix <- table(Actual = y.test, Predicted = ridge_pred)
print(conf_matrix)
```

```
##            Predicted
## Actual       blues classical country electronic hip-hop jazz pop reggae rock
##   blues         51         5      10         12       4   38  26     14   29
##   classical      9       177       0          0       0   16   3      0    2
##   country       13         1      88          4       3   25  11      7   45
##   electronic     7         3      11         95      32    3  19      9   15
##   hip-hop        6         0       4         15      82    0  36     55    4
##   jazz           7        17      21         12       1  116   8      5   10
##   pop           12         0      13         10      33   12  77     27   14
##   reggae        13         0       7          6      47    0  19    129    4
##   rock          14         0      44         10       2   14  19     13   75
```

```r
# Print or use the metrics_df data frame
print(metrics_df)
```

```
##                   Precision    Recall  F1_Score Specificity
## Class: blues      0.2698413 0.3863636 0.3177570   0.6136364
## Class: classical  0.8550725 0.8719212 0.8634146   0.1280788
## Class: country    0.4467005 0.4444444 0.4455696   0.5555556
## Class: electronic 0.4896907 0.5792683 0.5307263   0.4207317
## Class: hip-hop    0.4059406 0.4019608 0.4039409   0.5980392
## Class: jazz       0.5888325 0.5178571 0.5510689   0.4821429
## Class: pop        0.3888889 0.3532110 0.3701923   0.6467890
## Class: reggae     0.5733333 0.4980695 0.5330579   0.5019305
## Class: rock       0.3926702 0.3787879 0.3856041   0.6212121
```

### 3.1.4 Lasso Regularization

Lasso (Least Absolute Shrinkage and Selection Operator) regression is another form of linear regression that introduces regularization to select a subset of relevant predictors and shrink the coefficients of less important predictors to zero. This technique helps in building parsimonious models and handling multicollinearity.

Lasso regression minimizes the following objective function:

$$\text{minimize}\left(||\mathbf{y}-\mathbf{X}\beta||_2^2+\lambda||\beta||_1\right)$$

where $||\mathbf{y}-\mathbf{X}\beta||_2^2$ is the residual sum of squares, $||\beta||_1$ is the L1-norm penalty term, and $\lambda$ is the regularization parameter.

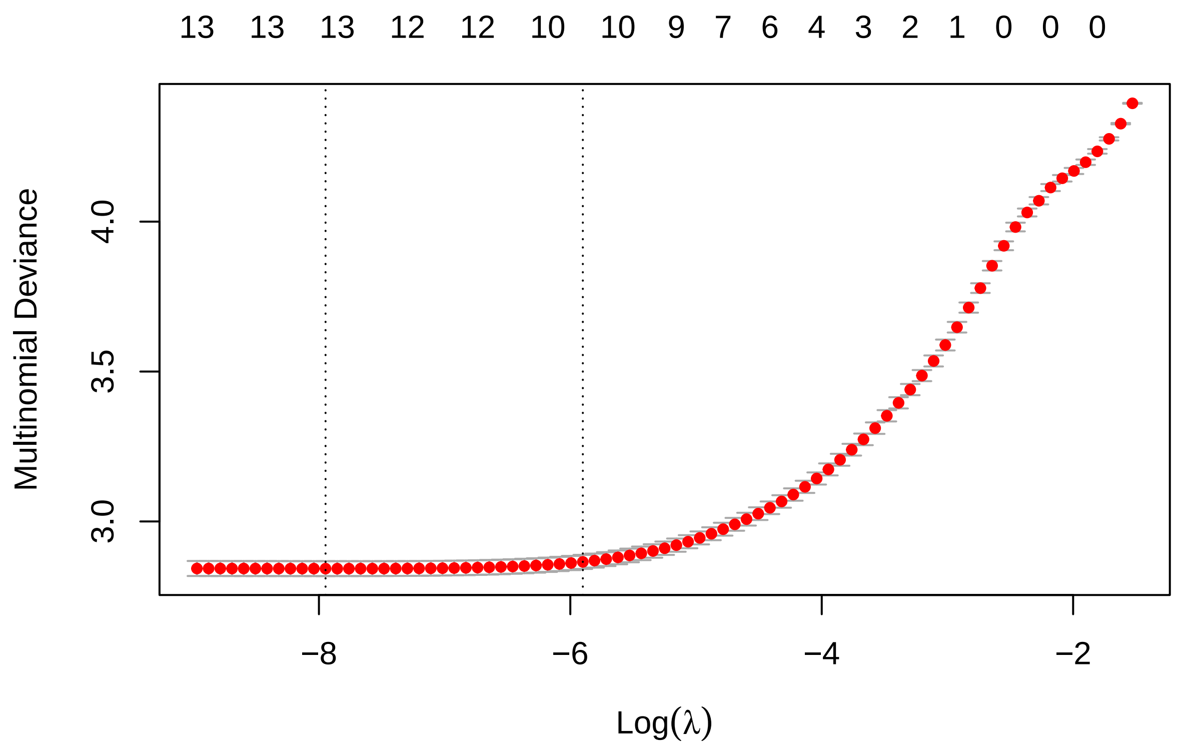
In this subsection, we apply Lasso regularization using the glmnet package in R. By setting the parameter alpha to 1, we exclusively apply Lasso regularization in our model. The L1-norm penalty encourages sparsity in the coefficient estimates, effectively performing variable selection by driving some coefficients to zero.

The tuning parameter $\lambda$ controls the strength of regularization and the degree of shrinkage applied to the coefficient estimates. Through cross-validation we select the optimal value of $\lambda$ that balances model complexity and predictive performance.

We evaluate the effectiveness of Lasso regularization by examining the resulting coefficient estimates, identifying significant predictors, and assessing the model's performance on validation data. Through this approach, we strive to build robust and parsimonious models that capture the underlying patterns in the data while minimizing overfitting.

```r
fit_lasso<-glmnet(X_scaled[train,],y[train],family="multinomial",alpha=1)
custom_lambda_values <- 10^seq(10, -4, length = 20)  # Customize the range and number of lambda values
```

```
cvfit_lasso <- cv.glmnet(X_scaled[train,], y[train], family = "multinomial", alpha = 1)
plot(cvfit_lasso)
```



```
bestlam.1se <- cvfit_lasso$lambda.1se
bestlam.1se
```

```
## [1] 0.002737711
```

```
lasso_coefficients<-coef(fit_lasso,s=bestlam.1se)
non_zero_counts <- sapply(lasso_coefficients, function(mat) sum(nnzero(mat)))

# Print the non-zero counts
print(non_zero_counts)
```

```
##      blues  classical    country electronic    hip-hop       jazz        pop 
##         13          8          9         12         11         14         13 
##     reggae       rock 
##         11          9
```

```
lasso_1se_pred=predict(cvfit_lasso,newx=X_scaled[test,],type="class", s="lambda.1se")
lasso_1se_pred_accuracy<-mean(lasso_1se_pred==y.test)
lasso_1se_pred_accuracy
```

```
## [1] 0.4938889
```

```
conf_matrix <- table(Actual = y.test, Predicted = lasso_1se_pred)
print(conf_matrix)
```

```
##             Predicted
## Actual        blues classical country electronic hip-hop jazz pop reggae rock
##   blues          49         4      10         10       4   41  26     15   30
##   classical      10       175       0          0       0   17   3      0    2
##   country        12         1      87          4       3   28  13      7   42
##   electronic      7         4      10         91      34    2  20     12   14
##   hip-hop         7         0       5         15      83    0  34     54    4
##   jazz            6        17      19         12       1  122   7      6    7
##   pop            11         0      12         13      31   12  74     27   18
##   reggae         11         0       3          4      46    0  22    134    5
##   rock           13         0      43         13       1   17  19     11   74
```

```
# Print or use the metrics_df data frame
print(metrics_df)
```

```
##                                   Class Precision    Recall  F1_Score Specificity
## Class: blues           Class: blues 0.2592593 0.3888889 0.3111111   0.6111111
## Class: classical   Class: classical 0.8454106 0.8706468 0.8578431   0.1293532
## Class: country       Class: country 0.4416244 0.4603175 0.4507772   0.5396825
## Class: electronic Class: electronic 0.4690722 0.5617284 0.5112360   0.4382716
## Class: hip-hop       Class: hip-hop 0.4108911 0.4088670 0.4098765   0.5911330
## Class: jazz             Class: jazz 0.6192893 0.5104603 0.5596330   0.4895397
## Class: pop               Class: pop 0.3737374 0.3394495 0.3557692   0.6605505
## Class: reggae         Class: reggae 0.5955556 0.5037594 0.5458248   0.4962406
## Class: rock             Class: rock 0.3874346 0.3775510 0.3824289   0.6224490
```

#### 3.1.4.1 Lasso Coefficients

In this subsection, we employ a bootstrap approach to assess the stability and variability of Lasso coefficients across different samples. Bootstrap resampling involves randomly sampling the dataset with replacement to generate multiple bootstrap samples. We repeat this process for a specified number of iterations, 1000 bootstraps in this case, to obtain a distribution of parameter estimates.

We apply the Lasso regression using the `glmnet` package in R to each bootstrap sample and extract the coefficients for each predictor variable. By aggregating the coefficient estimates across all bootstrap samples, we obtain insights into the robustness and significance of predictor variables in the Lasso model.

To visualize the distribution of Lasso coefficients for each class, we create box plots where the y-axis represents the coefficient values and the x-axis represents the predictor variables. Each box plot corresponds to a class, providing insights into the magnitude and variability of coefficients across different classes.

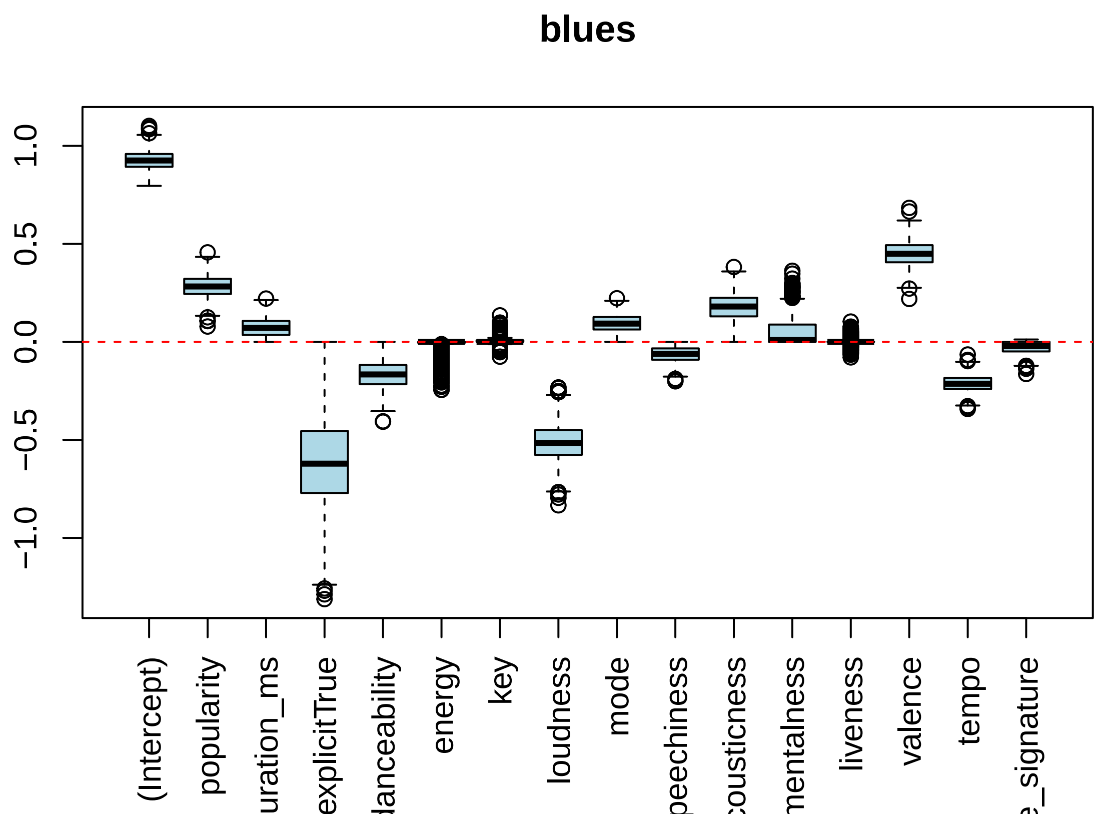
Through this analysis, we aim to identify important predictors that consistently contribute to the classification of each class while accounting for the variability introduced by the bootstrap resampling process. Box plots serve as effective visualizations to compare coefficient distributions and detect potential outliers or influential predictors.

```
library(doParallel)
library(foreach)
num_cores <- detectCores()
cl <- makeCluster(num_cores)
registerDoParallel(cl)
```

```
# Define the number of bootstraps
num_bootstraps <- 1000

# Define the function to get coefficients from bootstrapped samples
get_coefs <- function( indices) {
  #coef(model, s = cv.glmnet(X_scaled.train[indices, ], y.train[indices], family = "multinomial", alpha
  library(glmnet)
  model <- glmnet(
    x = X_scaled.train[indices, ],
    y = y.train[indices],
    family = "multinomial",
    alpha=1,
    parallel = TRUE
  )

  # Extract and return the coefficients as a matrix
  #return(as.matrix(coef(model,s=bestlam.1se)))
  coefs_list <- coef(model, s = bestlam.1se)

  # Convert the list of coefficient vectors to a list of matrices
  coefs_matrices <- lapply(coefs_list, as.matrix)

  return(coefs_matrices)
  }

# Bootstrap procedure in parallel
boot_coefs <- foreach(i = 1:num_bootstraps, .combine = "rbind") %dopar% {
  set.seed(i)  # Set a seed for reproducibility
  indices <- sample(1:nrow(X_scaled.train), nrow(X_scaled.train), replace = TRUE)
  get_coefs( indices)
}

# Stop the parallel backend
stopCluster(cl)
```

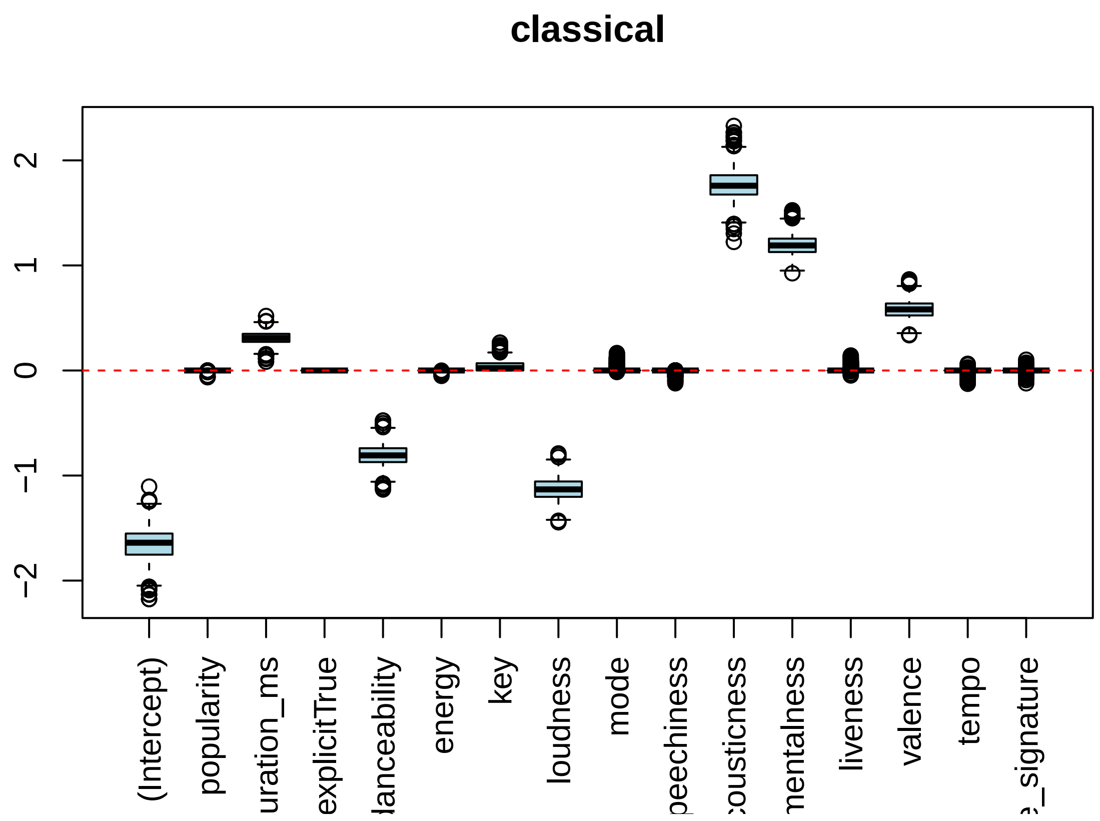
1. Blues

```
#par(mfrow = c(3,3))
i<-1
#for (i in (1:ncol(boot_coefs))) {
  class_coefs <- boot_coefs[,i]
  class_coefs_c<-do.call(cbind,class_coefs)
  boxplot(t(class_coefs_c), main = colnames(boot_coefs)[i],
          col = "lightblue",
          names = colnames(class_coefs),xaxt="n")
  abline(h=0,col="red",lty=2)
  axis(1, at = seq_along(rownames(class_coefs_c)), labels = rownames(class_coefs_c), las = 2)
```

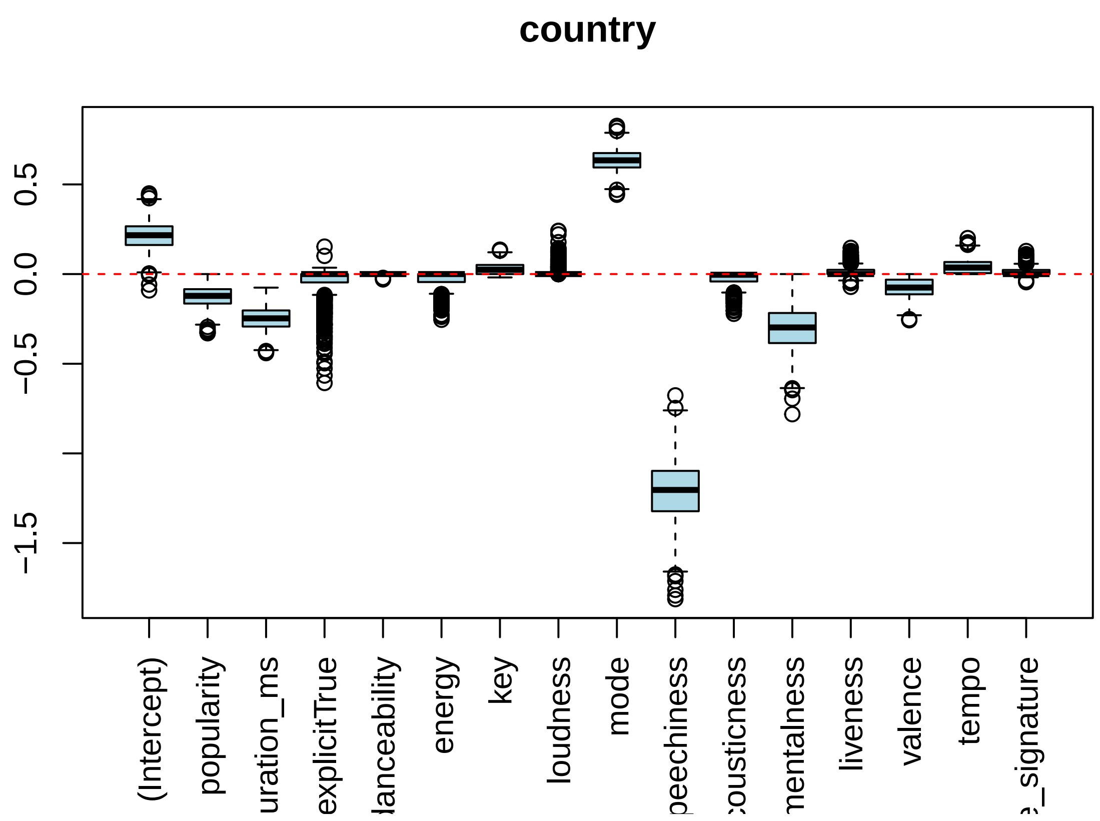


```
#}
#par(mfrow=c(1,1))
```

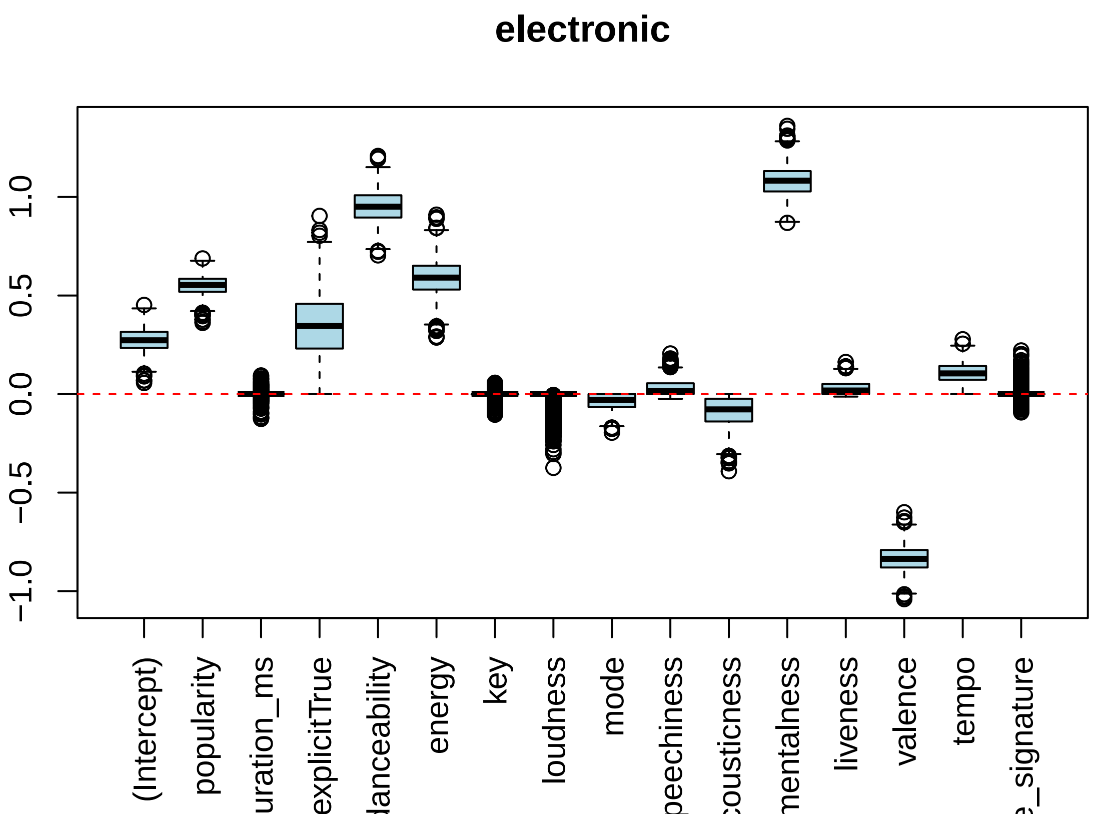
2. Classical


classical
2
1
0
−1
−2
(Intercept)
popularity
uration_ms
explicitTrue
lanceability
energy
key
loudness
mode
peechiness
cousticness
nentalness
liveness
valence
tempo
e_signature

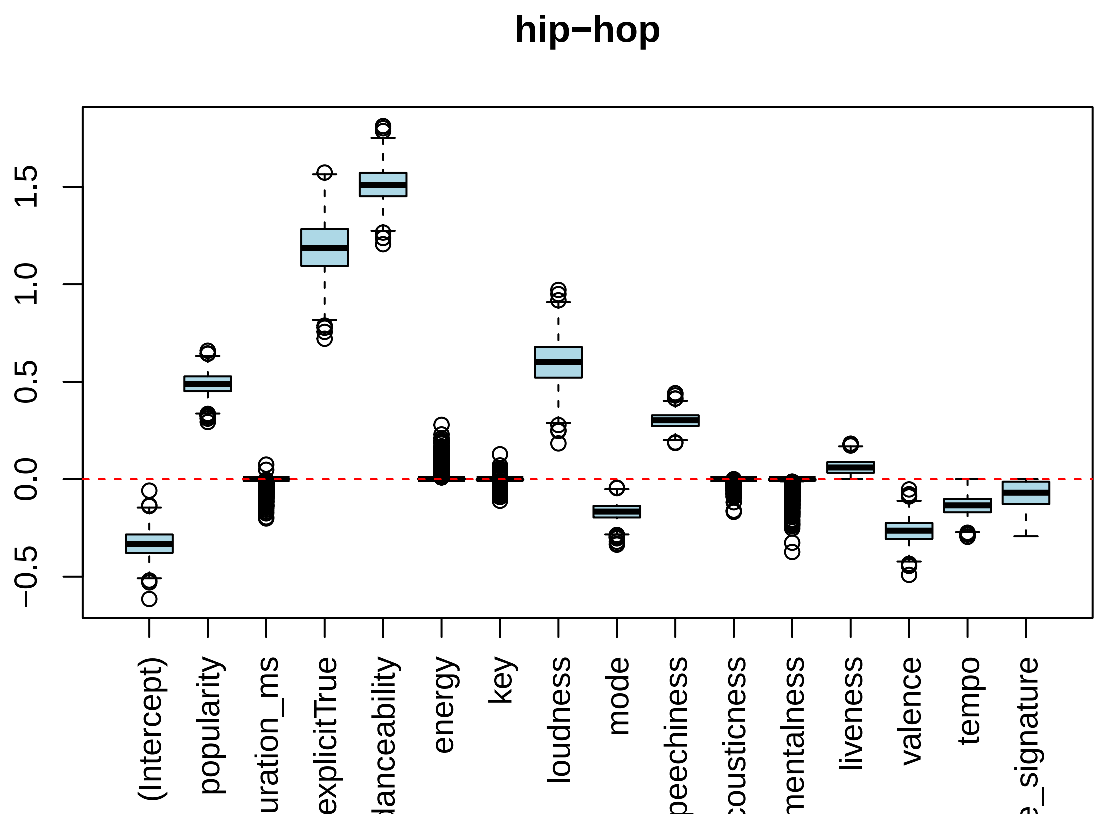
3. Country



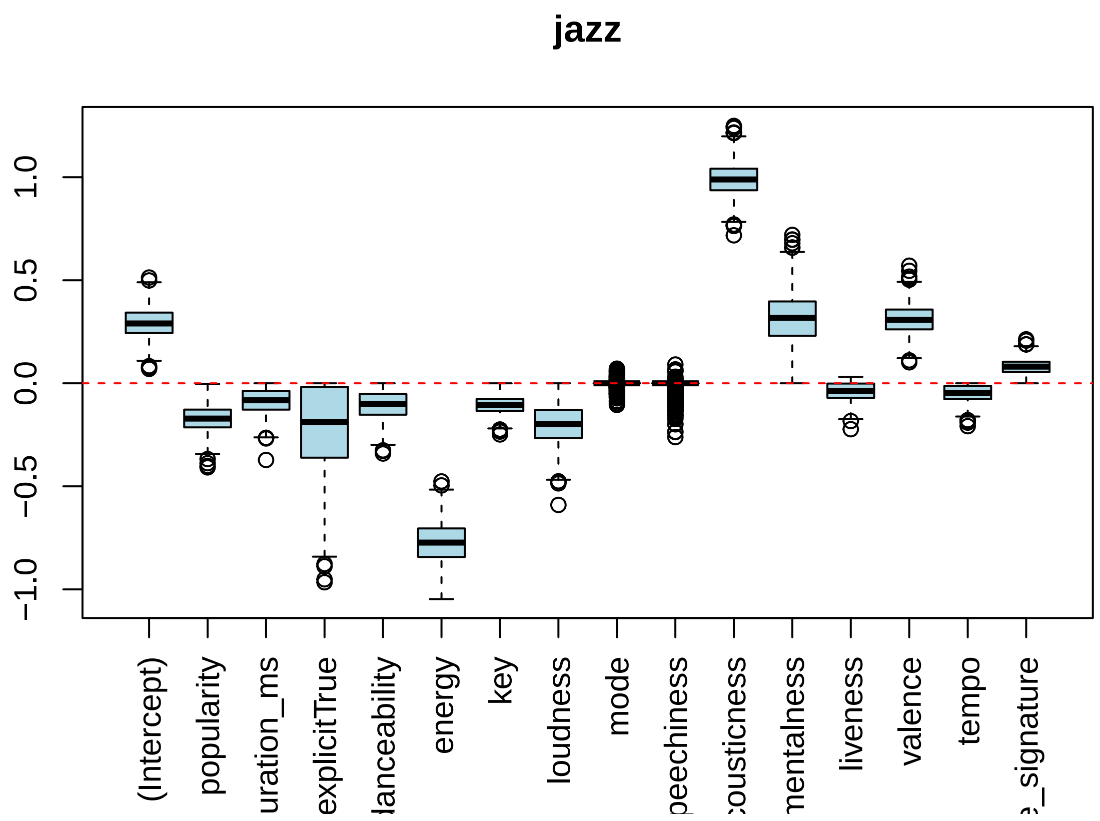
4. Electronic


electronic
1.0
0.5
0.0
−0.5
−1.0
(Intercept)
popularity
uration_ms
explicitTrue
lanceability
energy
key
loudness
mode
peechiness
cousticness
mentalness
liveness
valence
tempo
e_signature

5. Hip-Hop

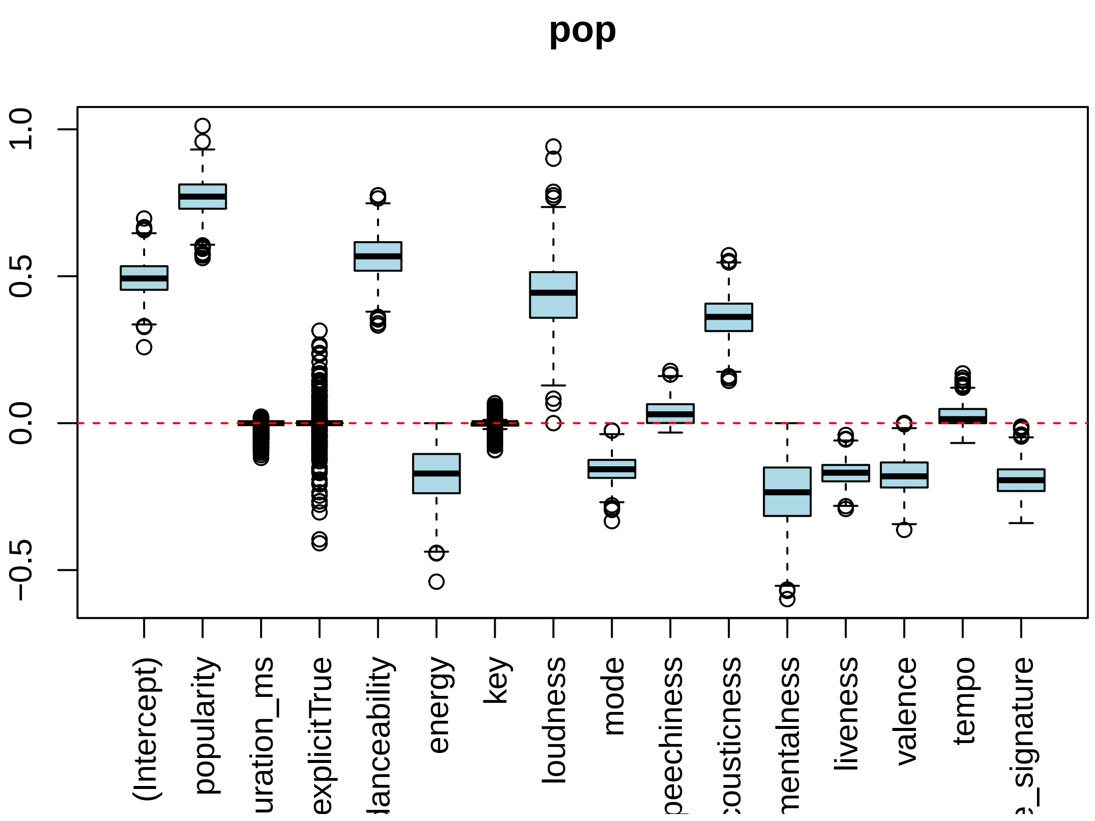

hip−hop
−0.5
0.0
0.5
1.0
1.5
(Intercept)
popularity
uration_ms
explicitTrue
lanceability
energy
key
loudness
mode
peechiness
cousticness
mentalness
liveness
valence
tempo
e_signature

6. Jazz

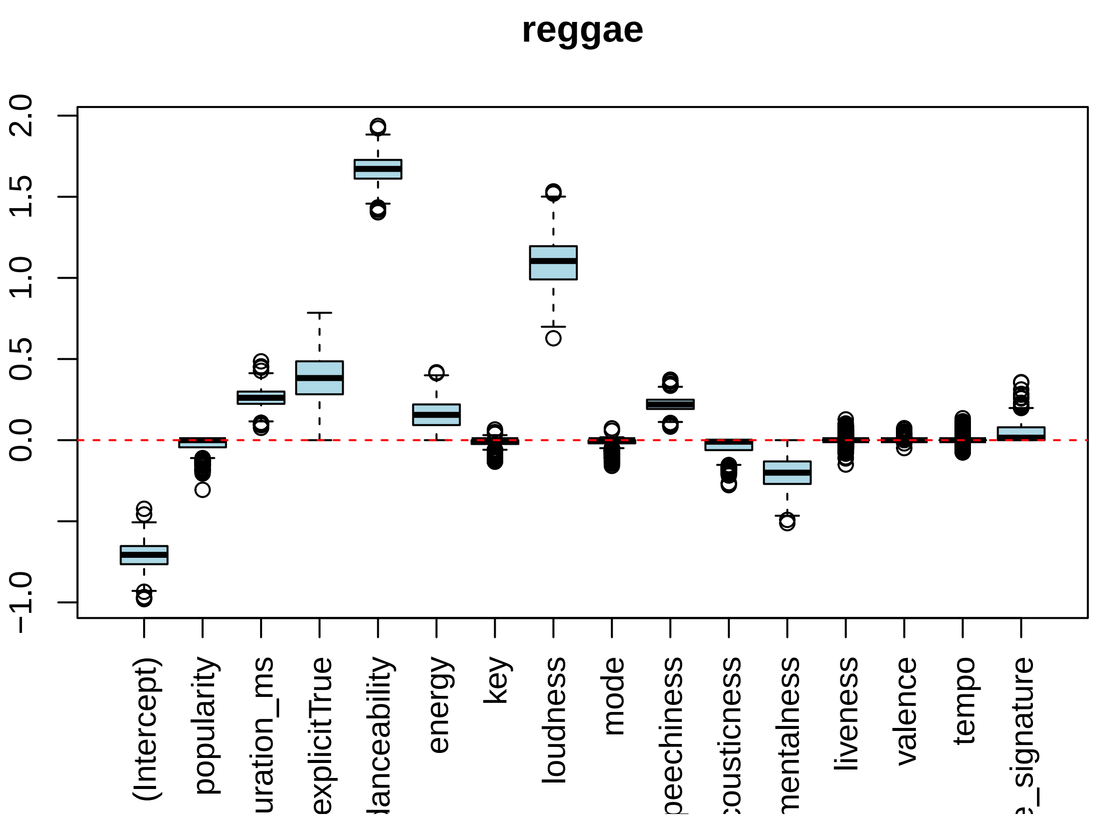


jazz
1.0
0.5
0.0
−0.5
−1.0
(Intercept)
popularity
uration_ms
explicitTrue
lanceability
energy
key
loudness
mode
peechiness
cousticness
mentalness
liveness
valence
tempo
e_signature

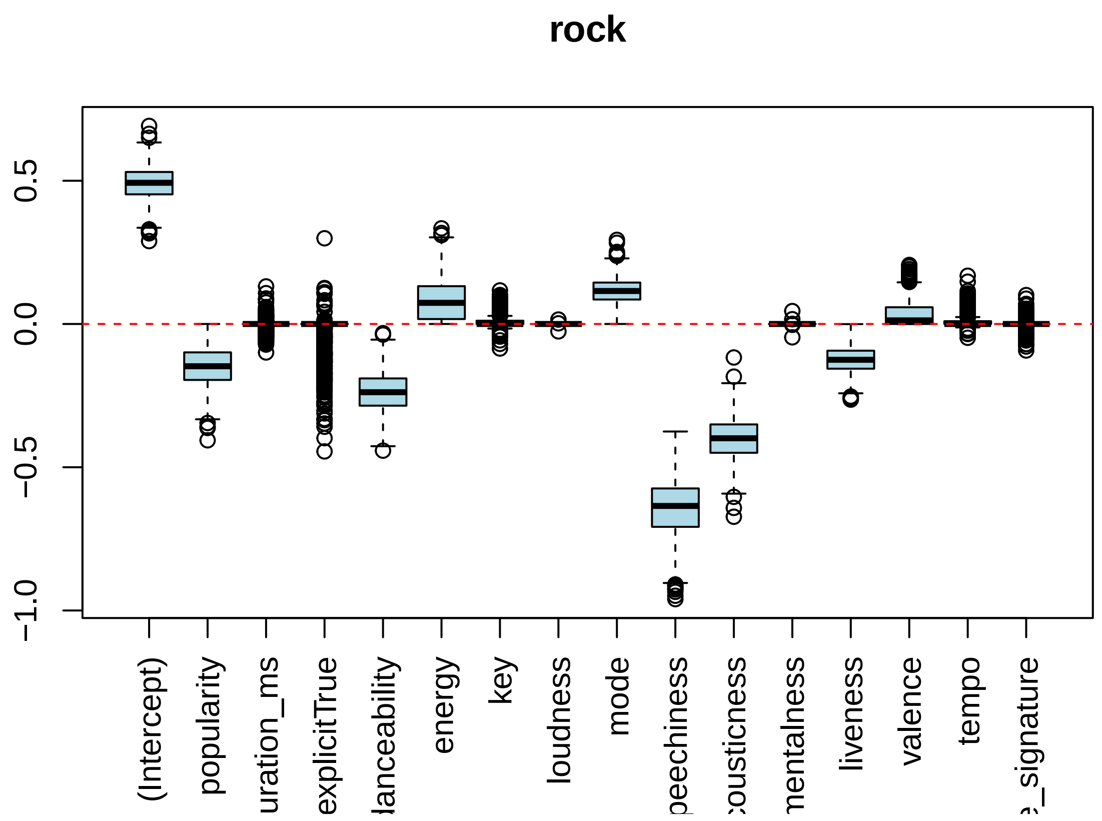
7. Pop


pop
1.0
0.5
0.0
−0.5
(Intercept)
popularity
uration_ms
explicitTrue
lanceability
energy
key
loudness
mode
peechiness
cousticness
mentalness
liveness
valence
tempo
e_signature

8. Reggae


reggae
2.0
1.5
1.0
0.5
0.0
−1.0
(Intercept)
popularity
uration_ms
explicitTrue
lanceability
energy
key
loudness
mode
peechiness
cousticness
mentalness
liveness
valence
tempo
e_signature

9. Rock



### 3.1.5 Elastic Net

Elastic Net regularization is a hybrid of ridge and Lasso regularization techniques, combining the strengths of both methods to handle multicollinearity, perform variable selection, and shrink coefficient estimates. Elastic Net introduces two tuning parameters, $\alpha$ and $\lambda$, controlling the balance between the L1-norm and L2-norm penalties.

The objective function for Elastic Net regularization is defined as:

$$\text{minimize}\left(||\mathbf{y}-\mathbf{X}\beta||_2^2+\lambda\left(\alpha||\beta||_1+(1-\alpha)||\beta||_2^2\right)\right)$$

where $||\mathbf{y}-\mathbf{X}\beta||_2^2$ is the residual sum of squares, $||\beta||_1$ is the L1-norm penalty term, $||\beta||_2^2$ is the L2-norm penalty term, $\lambda$ is the regularization parameter, and $\alpha$ controls the mixing ratio between L1 and L2 penalties.

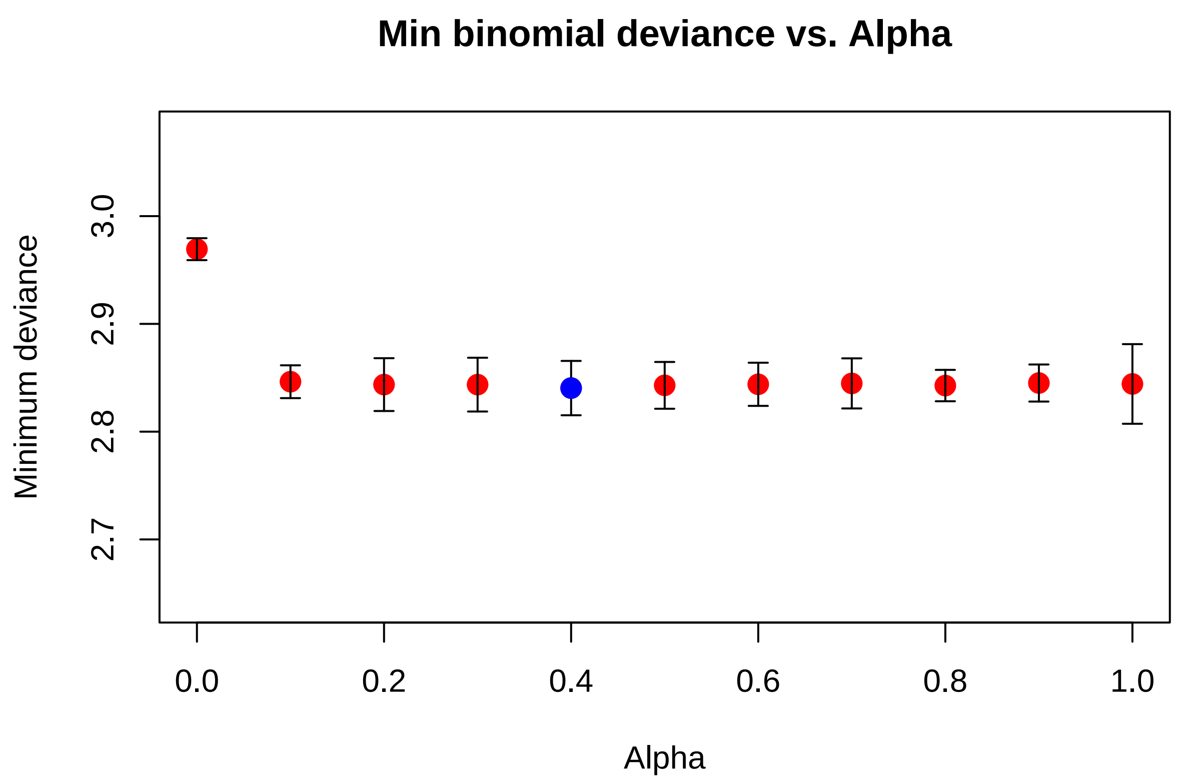
In this subsection, we apply Elastic Net regularization using the glmnet package in R. By setting $\alpha$ to a value between 0 and 1, we adjust the balance between Lasso and ridge penalties. A value of $\alpha = 1$ corresponds to pure Lasso regularization, while $\alpha = 0$ corresponds to pure ridge regularization.

Through cross-validation or other validation techniques, we select the optimal values of $\alpha$ and $\lambda$ that balance model complexity and predictive performance. Elastic Net regularization is particularly useful when dealing with high-dimensional data and highly correlated predictor variables. Through this approach, we strive to harness the advantages of both ridge and Lasso regularization techniques.

```r
set.seed(1)
# Initialize empty lists
alpha_list <- numeric()
```

```r
error_min_list <- numeric()
error_1se_list <- numeric()
lambda_min_list <- numeric()
lambda_1se_list <- numeric()

#custom_lambda_sequence <- 10^seq(1, -6, length = 100)
for (a in seq(0, 1, by = 0.1)) {

  cvfit_elastic_net <- cv.glmnet(X_scaled[train,],y[train],family="multinomial", alpha = a)#,lambda=cus
  #plot(cvfit_elastic_net)
  #title(main = bquote(alpha == .(a)))
  # Extract alpha, lambda.min, lambda.1se, and cross-validated errors
  alpha_value <- cvfit_elastic_net$glmnet.fit$alpha
  lambda_min <- cvfit_elastic_net$lambda.min
  lambda_1se <- cvfit_elastic_net$lambda.1se
  error_min <- cvfit_elastic_net$cvm[cvfit_elastic_net$lambda == lambda_min]
  error_1se <- cvfit_elastic_net$cvm[cvfit_elastic_net$lambda == lambda_1se]

  # Append values to lists
  alpha_list <- c(alpha_list, a)
  error_min_list <- c(error_min_list, error_min)
  error_1se_list <- c(error_1se_list, error_1se)
  lambda_min_list <- c(lambda_min_list, lambda_min)
  lambda_1se_list <- c(lambda_1se_list, lambda_1se)
}

results_df <- data.frame(
  alpha = seq(0, 1, by = 0.1),
  error_min = error_min_list,
  error_1se = error_1se_list,
  lambda_min = lambda_min_list,
  lambda_1se = lambda_1se_list
)

print(results_df)
```

```
##    alpha error_min error_1se   lambda_min  lambda_1se
## 1    0.0  2.969325  2.979513 0.0216958893 0.023811212
## 2    0.1  2.846341  2.861553 0.0004566777 0.003535889
## 3    0.2  2.843672  2.868176 0.0003990286 0.004482384
## 4    0.3  2.843657  2.868612 0.0004648759 0.004335448
## 5    0.4  2.840427  2.865648 0.0003486569 0.003916547
## 6    0.5  2.842974  2.864686 0.0004441283 0.003438724
## 7    0.6  2.843949  2.863980 0.0005369618 0.003144996
## 8    0.7  2.844777  2.868012 0.0005051271 0.003246994
## 9    0.8  2.842763  2.857330 0.0004419862 0.002358747
## 10   0.9  2.845128  2.862313 0.0004311816 0.002525439
## 11   1.0  2.844283  2.881225 0.0005129970 0.003619092
```



```r
alpha_chosen<-results_df$alpha[which.min(results_df$error_min)]

best_lam_elastic_net<-results_df$lambda_1se[which.min(results_df$error_min)]

best_elastic_net<-glmnet(X_scaled[train,],y[train],family="multinomial",alpha=alpha_chosen)
```

```r
elastic_net_pred<-predict(best_elastic_net,newx=X_scaled.test,type="class",s=best_lam_elastic_net)
elastic_net_pred_accuracy<-mean(elastic_net_pred==y.test)
elastic_net_pred_accuracy
```

```
## [1] 0.4955556
```

```r
conf_matrix <- table(Actual = y.test, Predicted = elastic_net_pred)
print(conf_matrix)
```

```
##             Predicted
## Actual        blues classical country electronic hip-hop jazz pop reggae rock
##   blues          51         4      10         12       4   39  25     14   30
##   classical       9       175       0          0       0   18   3      0    2
##   country        12         1      85          4       3   28  13      7   44
##   electronic      9         4       9         90      35    1  20     11   15
##   hip-hop         7         0       4         15      83    0  33     56    4
##   jazz            7        17      20         12       1  123   6      5    6
##   pop            11         0      11         13      31   12  78     26   16
##   reggae        11          0       4           5       47     0  22     132    4
##   rock          14          0      42          11        2    16  18      13   75
```

```
# Print or use the metrics_df data frame
print(metrics_df)
```

```
##                     Precision    Recall  F1_Score Specificity
## Class: blues      0.2698413 0.3893130 0.3187500   0.6106870
## Class: classical  0.8454106 0.8706468 0.8578431   0.1293532
## Class: country    0.4314721 0.4594595 0.4450262   0.5405405
## Class: electronic 0.4639175 0.5555556 0.5056180   0.4444444
## Class: hip-hop    0.4108911 0.4029126 0.4068627   0.5970874
## Class: jazz       0.6243655 0.5189873 0.5668203   0.4810127
## Class: pop        0.3939394 0.3577982 0.3750000   0.6422018
## Class: reggae     0.5866667 0.5000000 0.5398773   0.5000000
## Class: rock       0.3926702 0.3826531 0.3875969   0.6173469
```

## 3.2 K-Nearest Neighbours

K-Nearest Neighbors (KNN) is a non-parametric classification algorithm used for both classification and regression tasks. In KNN, the class of an observation is determined by the majority class among its K nearest neighbors in the feature space. KNN is particularly useful when dealing with non-linear relationships and high-dimensional data, making it a versatile choice for various classification tasks.

In this section, we apply KNN using the `knn()` function from the "class" package in R. We use the optimal value of K selected through cross-validation to build the final KNN model. The `knn()` function assigns the class label of the majority of the K nearest neighbors to each observation in the test dataset.

To choose the optimal value of K in KNN, we employ cross-validation using the `trainControl()` function from the caret package.

```
data.train <- data.frame(X_scaled.train, y.train)

# Define the training control with 10-fold cross-validation
ctrl <- trainControl(method = "cv", number = 10)

# Specify the tuning grid for k (e.g., values from 1 to 10)
grid <- expand.grid(k = 1:10)

# Train the KNN model with cross-validation
set.seed(123)  # Set a seed for reproducibility
knn_model <- train(y.train ~ ., data = data.train, method = "knn", trControl = ctrl, tuneGrid = grid)
```

The optimal K found from cross-validation is K=1

```
optimal_k <- knn_model$bestTune$k
knn_model <- knn(train = X_scaled[train,], test = X_scaled[test,], cl = df_filtered[train,16], k = opti

# Evaluate the model
conf_matrix <- table(Actual = y.test, Predicted = knn_model)
print(conf_matrix)
```

```
##             Predicted
## Actual       blues classical country electronic hip-hop jazz pop reggae rock
##   blues         70         2      19         19       7   34   9      7   22
##   classical      8       181       1          2       1   11   2      0    1
##   country       22         2     115          4       0    5   9     10   30
##   electronic    20         4      12        101      23    2  14     11    7
##   hip-hop        6         0       3         18      82    2  50     36    5
##   jazz          43        17       4          7       3  108   4      4    7
##   pop           11         2      13          7      50    2  85     16   12
##   reggae         9         0       4          8      35    2  34    120   13
##   rock           9         1      26          5       5    5  18      5  117
```

```r
Accuracy<-mean(knn_model==y.test)
Accuracy
```

```
## [1] 0.5438889
```

```r
# Print or use the metrics_df data frame
print(metrics_df)
```

```
##                   Precision    Recall  F1_Score Specificity
## Class: blues      0.3703704 0.3535354 0.3617571   0.6464646
## Class: classical  0.8743961 0.8660287 0.8701923   0.1339713
## Class: country    0.5837563 0.5837563 0.5837563   0.4162437
## Class: electronic 0.5206186 0.5906433 0.5534247   0.4093567
## Class: hip-hop    0.4059406 0.3980583 0.4019608   0.6019417
## Class: jazz       0.5482234 0.6315789 0.5869565   0.3684211
## Class: pop        0.4292929 0.3777778 0.4018913   0.6222222
## Class: reggae     0.5333333 0.5741627 0.5529954   0.4258373
## Class: rock       0.6125654 0.5467290 0.5777778   0.4532710
```

### 3.3 Random Forest

Random Forest is a powerful ensemble learning method that constructs a multitude of decision trees during training and outputs the mode of the classes (classification) or mean prediction (regression) of the individual trees. It's known for its robustness, scalability, and ability to handle high-dimensional data with complex interactions.

To optimize the performance of the Random Forest model and prevent overfitting, we utilize cross-validation techniques. We split the dataset into training and validation sets using the `trainControl()` function from the caret package. By varying parameters such as the number of trees and the number of variables randomly sampled as candidates at each split, we tune the model for optimal performance.

In this section, we apply Random Forest using the `randomForest()` function from the "randomForest" package in R. We specify the number of trees in the forest and other hyperparameters based on cross-validation results. The `randomForest()` function constructs the ensemble of decision trees and aggregates their predictions to make final classifications or predictions.

We evaluate the performance of the Random Forest model using appropriate evaluation metrics such as accuracy, precision, recall, and F1-score.

```r
#library(randomForest)

#library(caret)

# Create a training control with cross-validation
ctrl <- trainControl(method = "cv", number = 10, classProbs = TRUE)
levels(df_filtered$track_genre) <- make.names(levels(df_filtered$track_genre))
# Train a Random Forest model using cross-validation
rf_model <- train(track_genre ~ ., data = df_filtered[train,],
                  method = "rf",
                  trControl = ctrl)
```

From cross-validation we set mtry=8. “mtry” refers to the number of variables randomly sampled as candidates at each split when building each tree in the forest.

```r
predictions <- predict(rf_model, newdata = df_filtered[test,])

# Evaluate the model
confusion_matrix <- table(predictions, y.test)
confusion_matrix
```

```
##             y.test
## predictions  blues classical country electronic hip-hop jazz pop reggae rock
##   blues         88         8      15         17       4   38   5      0    3
##   classical      4       192       1          4       0   18   0      0    0
##   country       11         0     128          2       1    4   3      6   23
##   electronic    23         4       5        146       7   16   1      6    5
##   hip.hop        6         0       2         10     129    2  31     30    2
##   jazz          28         3       7          0       1  112   6      0    4
##   pop           10         0       9          6      34    3 129     24   11
##   reggae         5         0       5          5      24    2  16    157    4
##   rock          14         0      25          4       2    2   7      2  139
```

```r
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
print(paste("Accuracy:", accuracy))
```

```
## [1] "Accuracy: 0.677777777777778"
```

```r
# Print or use the metrics_df data frame
print(metrics_df)
```

```
##                   Precision    Recall  F1_Score Specificity
## Class: blues      0.4943820 0.4656085 0.4795640  0.53439153
## Class: classical  0.8767123 0.9275362 0.9014085  0.07246377
## Class: country    0.7191011 0.6497462 0.6826667  0.35025381
## Class: electronic 0.6854460 0.7525773 0.7174447  0.24742268
## Class: hip-hop    0.6084906 0.6386139 0.6231884  0.36138614
## Class: jazz       0.6956522 0.5685279 0.6256983  0.43147208
## Class: pop        0.5707965 0.6515152 0.6084906  0.34848485
## Class: reggae     0.7201835 0.6977778 0.7088036  0.30222222
## Class: rock       0.7128205 0.7277487 0.7202073  0.27225131
```

## 3.4 Gradient Boosting

Gradient Boosting is a powerful machine learning technique that builds a predictive model in the form of an ensemble of weak prediction models of decision trees. It sequentially trains new models to correct errors made by existing models, with each new model focusing on the residuals of the previous model.Gradient Boosting is well-suited for regression and classification tasks, especially when dealing with complex relationships and high-dimensional datasets

To optimize the performance of the Gradient Boosting model and prevent overfitting, we employ cross-validation techniques. We use the `trainControl()` function from the “caret” package to split the dataset into training and validation sets. By varying parameters such as the learning rate, tree depth, and the number of boosting iterations, we tune the model for optimal performance.

In this section, we apply Gradient Boosting using the `gbm()` function from the “gbm” package in R. The `gbm()` function iteratively fits a sequence of regression trees to the residuals of the previous model. We evaluate the performance of the Gradient Boosting model using accuracy. Additionally, we analyze the importance of predictor variables using a relative influence measure.

```
library(caret)
library(gbm)
```

```
## Warning: il pacchetto 'gbm' è stato creato con R versione 4.3.2
```

```
## Loaded gbm 2.1.9
```

```
## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.c
```

```
# Set up parameter grid for tuning (without subsample and colsample_bytree)
param_grid <- expand.grid(
  n.trees = c(50, 100, 500,1000),  # Adjust the values as needed
  interaction.depth = c(1, 2, 4, 6,8,10),
  shrinkage = c(0.01, 0.1, 0.3),
  n.minobsinnode = 10
)
param_grid <- expand.grid(
  n.trees = c(1000,1500,2000,5000),  # Adjust the values as needed
  interaction.depth = c(1, 5,10,15,20),
  shrinkage = c(0.001,0.01, 0.1),
  n.minobsinnode = 10
)
```

```
results <- list()

# Perform grid search over hyperparameter space
for (i in 1:nrow(param_grid)) {
  # Define parameters for this iteration
  params <- list(
    n.trees = param_grid$n.trees[i],
    interaction.depth = param_grid$interaction.depth[i],
    shrinkage = param_grid$shrinkage[i],
    n.minobsinnode = param_grid$n.minobsinnode[i],
    distribution = "multinomial"
  )
  # Perform cross-validation with caret and gbm
  ctrl <- trainControl(method = "cv", number = 5)
  cv_result <- train(
    x = df_filtered[train, -16],  # Exclude the response variable column
    y = as.factor(y.train),
    method = "gbm",
    trControl = ctrl,
    tuneGrid = data.frame(n.trees = params$n.trees, interaction.depth = params$interaction.depth, shrinl
  )

  # Store the results
  results[[paste0("n.trees_", params$n.trees, "_interaction_depth_", params$interaction.depth, "_shrinka
}

# Get the best model
#best_index <- which.min(sapply(results, function(x) x$RMSE))
best_model <- results[[which.max(lapply(results, function(x) x$results$Accuracy))]]
print(best_model)
```

The paramter values that we will use for the model are the following:

```
model_parameters <- data.frame(
  n.trees = best_model$bestTune$n.trees,
  interaction.depth = best_model$bestTune$interaction.depth,
  shrinkage = best_model$bestTune$shrinkage,
  n.minobsinnode = best_model$bestTune$n.minobsinnode
)

# Print the dataframe
print(model_parameters)
```

```
##   n.trees interaction.depth shrinkage n.minobsinnode
## 1    2000                10      0.01             10
```

```
  final_model <- train(
  x = df_filtered[, -16],  # Exclude the response variable column
  y = as.factor(df_filtered$track_genre),
  method = "gbm",
  trControl = trainControl(method = "none"),  # No resampling for final training
  tuneGrid = data.frame(
    n.trees = best_model$bestTune$n.trees,
    interaction.depth = best_model$bestTune$interaction.depth,
    shrinkage = best_model$bestTune$shrinkage,
    n.minobsinnode = best_model$bestTune$n.minobsinnode
  )
)
```

```
test_predictions <- predict(final_model, newdata = df_filtered[test,-16])
confusion_matrix <- table(test_predictions, y.test)
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
print(paste("Accuracy:", accuracy))
```

```
## [1] "Accuracy: 0.9"
```

```r
print(confusion_matrix)
```

```
##                  y.test
## test_predictions blues classical country electronic hip-hop jazz pop reggae
##       blues        162         0       6          3       0   12   0      0
##       classical      0       206       0          0       0    1   0      0
##       country        1         0     179          0       1    2   2      2
##       electronic     2         0       0        186       1    5   0      1
##       hip.hop        1         0       1          1     160    0  14      8
##       jazz          15         1       2          0       1  174   0      0
##       pop            1         0       2          2      17    2 164      6
##       reggae         1         0       2          1      20    0  14    208
##       rock           6         0       5          1       2    1   4      0
##                  y.test
## test_predictions rock
##       blues         1
##       classical     0
##       country       6
##       electronic    0
##       hip.hop       1
##       jazz          0
##       pop           1
##       reggae        1
##       rock        181
```

```r
# Print or use the metrics_df data frame
print(metrics_df)
```

```
##                     Precision    Recall  F1_Score Specificity
## Class: blues        0.8804348 0.8571429 0.8686327 0.142857143
## Class: classical    0.9951691 0.9951691 0.9951691 0.004830918
## Class: country      0.9274611 0.9086294 0.9179487 0.091370558
## Class: electronic   0.9538462 0.9587629 0.9562982 0.041237113
## Class: hip-hop      0.8602151 0.7920792 0.8247423 0.207920792
## Class: jazz         0.9015544 0.8832487 0.8923077 0.116751269
## Class: pop          0.8410256 0.8282828 0.8346056 0.171717172
## Class: reggae       0.8421053 0.9244444 0.8813559 0.075555556
## Class: rock         0.9050000 0.9476440 0.9258312 0.052356021
```

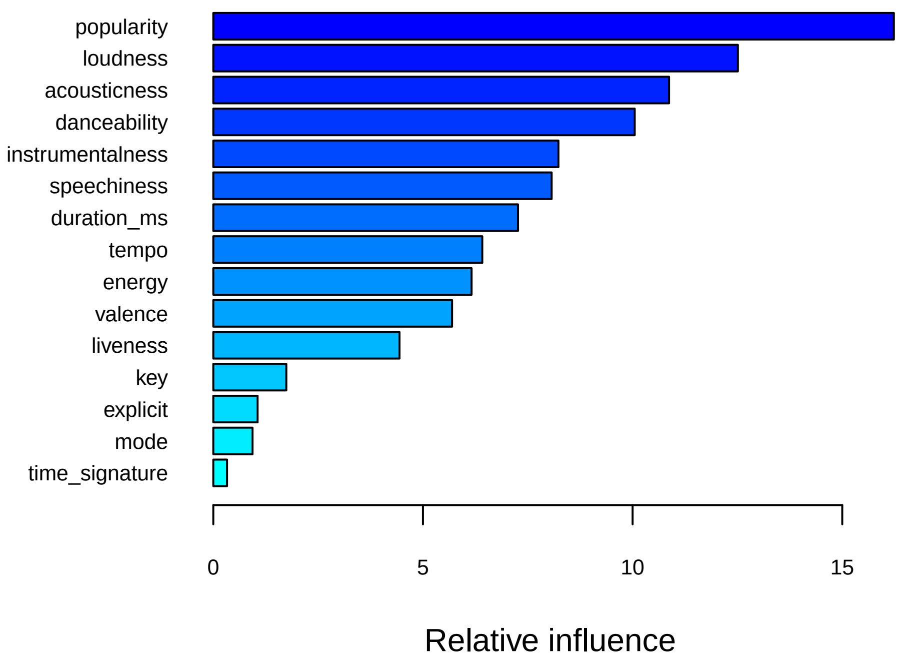
```r
mai.old<-par()$mai
mai.old
```

```
## [1] 1.02 0.82 0.82 0.42
```

```r
#new vector
mai.new<-mai.old
#new space on the left
mai.new[2] <- 2.5
mai.new
```

```
## [1] 1.02 2.50 0.82 0.42
```

```
#modify graphical parameters
par(mai=mai.new,cex.axis=0.7)
#summary(final_model, las=1)
#las=1 horizontal names on y
summary(final_model, las=1, cBar=15)
```



```
##                                 var    rel.inf
## popularity               popularity 16.2308476
## loudness                   loudness 12.5111344
## acousticness           acousticness 10.8698713
## danceability           danceability 10.0507139
## instrumentalness instrumentalness  8.2308774
## speechiness             speechiness  8.0696686
## duration_ms             duration_ms  7.2665409
## tempo                         tempo  6.4153764
## energy                       energy  6.1597559
## valence                     valence  5.6947981
## liveness                   liveness  4.4403517
## key                             key  1.7416439
## explicit                   explicit  1.0555830
## mode                           mode  0.9348006
## time_signature       time_signature  0.3280362
```

```
#cBar defines how many variables
#back to orginal window
par(mai=mai.old)
```