Final Report Data Taming- Spotify Genre Prediction

Ujjwal Bhardwaj (a1881450)

2023-08-15

Executive Summary

The task at hand for me as a data scientist at Spotify had the aim of improving the user experience by creating a model that predicts song genres based on important characteristics like "release_year", "speechiness", "danceability", and "tempo". 1000 songs per genre were used in the project, which was conducted using a cleaned version of the carefully curated data-set due to computational constraints.

I delved into various concerns raised by Spotify's founders throughout the analysis. These investigations covered topics like identifying discrepancies in song popularity across various genres, examining variations in speeches within each genre, and developing strategies to track changes in popularity over time. Three different models were developed and assessed as part of the strategy: random forest, linear discriminant analysis, and K-nearest neighbors. I used a variety of performance metrics, such as AUC, sensitivity, and specificity, to identify the best model. I then chose the model that showed the best performance based on these metrics and explained the reasoning behind this decision to the founders.

I carried out extensive testing after identifying the best model by predicting song genres using the chosen variables and then comparing my predictions to the actual song genres. The main goal of this project was to improve playlist recommendations by providing accurate genre predictions for Spotify's music streaming service.

In a nutshell, my work as a data scientist at Spotify involved building a model that predicts song genres using useful features. This project, which was motivated by the desire to improve user experiences, involved careful dataset curation, in-depth analysis of dynamics related to genres, careful model selection, and stringent testing procedures.

Methods used

1- Data-> The sample of the Spotify songs data-set from TidyTuesday's data was used for this analysis. The data contains details about 32833 songs from various playlists and song genres on Spotify, along with 23 additional musical variables that provide precise insights into a particular song.

2- Steps Involved-> The following steps need to be completed in order to fulfill the study objective. These steps vary from data preparation, cleaning, transformation and quality check.

a) Import data using read_csv() from the readr package.
b) Select relevant columns using dplyr package: track name, playlist genre, track popularity, tempo, danceability, release date, speechiness.
c) Remove rows with missing values using na.omit().
d) Convert release date to date format with mutate() and as.Date().
e) Subset data to 1000 songs per genre using group_by(), sample_n(), and ungroup().
f) Convert playlist genre to factor with mutate(), drop remaining missing values with drop_na(), and select relevant columns with select().
g) Verify data structure with glimpse() for further analysis.
h) Analyze and model the data.

i) Evaluate model performance and make predictions.

3- Software (libraries and packages)-> The following R packages: **readr, tidyr, tidymodels, tidyverse, janitor, skimr, and rsample** were used in the analysis.

Result Section

Exploratory Data Analysis The analysis investigated the relationship between Spotify song features and listenership. We started by looking at differences in song popularity between genres. For this, two visualization techniques were used. The first was a box plot that demonstrated each genre's median, interquartile range, and range of popularity. Notable findings include that Latin and pop genres have the highest levels of popularity, while R&B has the widest interquartile range. A swarm plot, the second method, indicated data density across genres. Both methods highlighted different genres' median popularity, but there was still a lot of overlap.

The differences in speechiness between genres were then looked into. Two visualization techniques were used: a box plot displaying each genre's median, interquartile range, and range of speechiness values and a violin plot displaying speechiness density per genre. These methods showed that, in line with expectations, the Rap genre exhibited higher median speechiness than rock music, which showed the lowest levels. Other genres, however, were characterized by noticeable overlap.

Finally, it was investigated how the popularity of each genre has evolved over time. A line plot tracing popularity trends and a heatmap demonstrating colored tiles representing average popularity were utilized in tandem to visualize the average song's popularity by year and genre. With pop and rock dominating the 1960s, giving way to EDM in the early 2000s, and further shifting to Latin, R&B, and Rap in 2020, these visualizations showed how genre popularity has changed. Both methods, over the years, reinforced variations in genre-specific popularity.

In conclusion, despite significant genre overlap, our exploratory data analysis showed apparent differences in song popularity and speechiness among genres. Since the data is very dispersed, it is clear that there is little correlation between popularity and the year that songs are released. Therefore, incorporating additional variables beyond the first four is crucial for improving the model. The other variables in the dataset have also been correlated. Hence, we can say that conclusions from additional analyses of tempo and danceability overlapped similarly.

Model Efficiency Description (Discussion Section)

We employed the pre-selected models that the company recommended, incorporating additional variables. Using Approach 1, the Linear Discriminant Analysis (LDA) model demonstrated a 0.473 accuracy, which indicates that correct genre prediction is made about 50% of the time. Sensitivity measured at 0.497 indicated that 50% of positive class songs (i.e., the correct genre) were correctly identified. In contrast, specificity measured at 0.891 indicated that 89.1% of negative class songs (i.e., the incorrect genre) were correctly identified. The accuracy of this model had a 95% confidence interval of 0.448 to 0.499, which pointed to a consistent accuracy within this range in subsequent studies.

The LDA model using Approach 2 produced comparable results without specific metrics or confidence intervals.

The K Nearest Neighbor (KNN) model had an accuracy of 0.515, corresponding to 51% accurate genre prediction. Positive class songs were correctly identified with a sensitivity of 0.501 and negative class songs with a specificity of 0.900, respectively. This model performed better than the LDA model.

The Random Forest (RF) model, which predicted genres correctly approximately 55% of the time, had the highest accuracy in general, with a score of 0.545. Both the sensitivity and specificity were 0.574 and 0.914. The 95% confidence interval ranging from 0.519 to 0.570, further supported the RF model's consistent accuracy within this domain.

The Random Forest (RF) model, which had the highest specificity and sensitivity, ultimately was the most accurate. As a result, the RF model is suggested for future modeling efforts.

Final Outcome

Our developed random forest model demonstrated a 54.2% accuracy on the test dataset. With a kappa score of roughly 0.50, it performs more effectively than random guessing, but there is scope for enhancement.

Grid search was applied for hyperparameter optimization to adjust mtry and min_n. Various mtry and min_n combinations were evaluated using 5-fold cross-validation, with the selection made based on the area under the ROC curve (AUC). mtry = 4 and min_n = 30 proved to be the ideal hyperparameters, and the resultant AUC on the validation set was 0.579.

Tempo, danceability, and valence were key predictors of the variable importance plot. These correspond to the suggested variables by the company, demonstrating their significance in genre prediction.

Predictive testing correctly identified a pop song's genre, demonstrating the model's utility but not its perfect accuracy.

In conclusion, our random forest model performed admirably at predicting genre, and variable importance analysis offered illuminating guidance. However, given the current accuracy's modest level, improvements are necessary.

Conclusion

This project aimed to develop a machine-learning model to predict Spotify playlist genres based on audio features. The foundation for modeling is established by data exploration and preprocessing. Cross-validation and grid search were utilized to train and fine-tune a random forest classifier, which resulted in a test set accuracy of 54.2%.

It's essential to understand the complexity of predicting genre solely from audio attributes, which frequently feature significant inter-genre audio commonalities, even though the accuracy may not be exceptional. Enhanced accuracy may be achievable with playlist name/ID integration, as shown by the random forest model's 100% accuracy rating, although it was used only for educational purposes. Nevertheless, both exploratory analysis and foundational analysis benefit from using our model.

We support additional data aggregation and diversity to boost the precision of genre classification while acknowledging the model's limitations. Given their potential to capture intricate audio-genre associations, advanced techniques like neural networks or ensemble models are also worth taking into consideration.

Appendix- Implementation and Working of our Model

```
# Lets us start with loading libraries and read data
pacman::p_load(readr,tidyr,tidymodels,tidyverse,janitor,skimr,rsample)
pacman::p_load(ggbeeswarm)
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
library(discrim)
```

```
##
## Attaching package: 'discrim'
```

```
## The following object is masked from 'package:dials':
##
```

```
##     smoothness
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following objects are masked from 'package:yardstick':
##
##     precision, recall, sensitivity, specificity
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
pacman::p_load(randomForest)
pacman::p_load(tidyverse,knitr)
library(randomForestExplainer)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
library(randomForest)
```

```
# Retrieving the data
spotify_data <- readr::read_csv('https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/da
```

```
## Rows: 32833 Columns: 23
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (10): track_id, track_name, track_artist, track_album_id, track_album_na...
## dbl (13): track_popularity, danceability, energy, key, loudness, mode, speec...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# Setting the seed
head(spotify_data)
```

```
## # A tibble: 6 x 23
##   track_id               track_name track_artist track_popularity track_album_id
##   <chr>                  <chr>      <chr>                   <dbl> <chr>
## 1 6f807x0ima9a1j3VPbc7VN I Don't C~ Ed Sheeran                  66 2oCs0DGTsRO98~
## 2 0r7CVbZTWZgbTCYdfa2P31 Memories ~ Maroon 5                    67 63rPSO264uRjW~
## 3 1z1Hg7Vb0AhHDiEmnDE79l All the T~ Zara Larsson                70 1HoSmj2eLcsrR~
## 4 75FpbthrwQmzHlBJLuGdC7 Call You ~ The Chainsm~                60 1nqYsOef1yKKu~
## 5 1e8PAfcKUYoKkxPhrHqw4x Someone Y~ Lewis Capal~                69 7m7vv9wlQ4i0L~
## 6 7fvUMiyapMsRRxr07cU8Ef Beautiful~ Ed Sheeran                  67 2yiy9cd2QktrN~
## # i 18 more variables: track_album_name <chr>, track_album_release_date <chr>,
## #   playlist_name <chr>, playlist_id <chr>, playlist_genre <chr>,
## #   playlist_subgenre <chr>, danceability <dbl>, energy <dbl>, key <dbl>,
## #   loudness <dbl>, mode <dbl>, speechiness <dbl>, acousticness <dbl>,
## #   instrumentalness <dbl>, liveness <dbl>, valence <dbl>, tempo <dbl>,
## #   duration_ms <dbl>
```

```r
set.seed(1881450)
```

```r
# Now we shall work on the loaded data.

# Selecting variables and cleaning the data
spotify_data_select <- spotify_data %>%
  dplyr::select(track_name, playlist_genre,track_popularity, key,instrumentalness,duration_ms,track_popu

  # Removing the missing values
  na.omit() %>%

  # Converting the track_album_release_date to a date format using mutate()
  mutate(track_album_release_date = as.Date(track_album_release_date, format = "%Y"))

# Now we will be sub-setting the data to 1,000 songs per genre as required

spotify_data_select <- spotify_data_select %>%
  group_by(playlist_genre) %>%
  sample_n(1000) %>%
  ungroup()

# Now converting playlist_genre to a factor and will drop any remaining missing values followed by the

spotify_data_select <- spotify_data_select %>%
  mutate(playlist_genre = factor(playlist_genre)) %>%
  dplyr::select(track_name, playlist_genre,track_popularity,key,instrumentalness,duration_ms,loudness,a
  drop_na()

# Now that the data is cleaned and subsets have been made, we will Check the structure
glimpse(spotify_data_select)
```

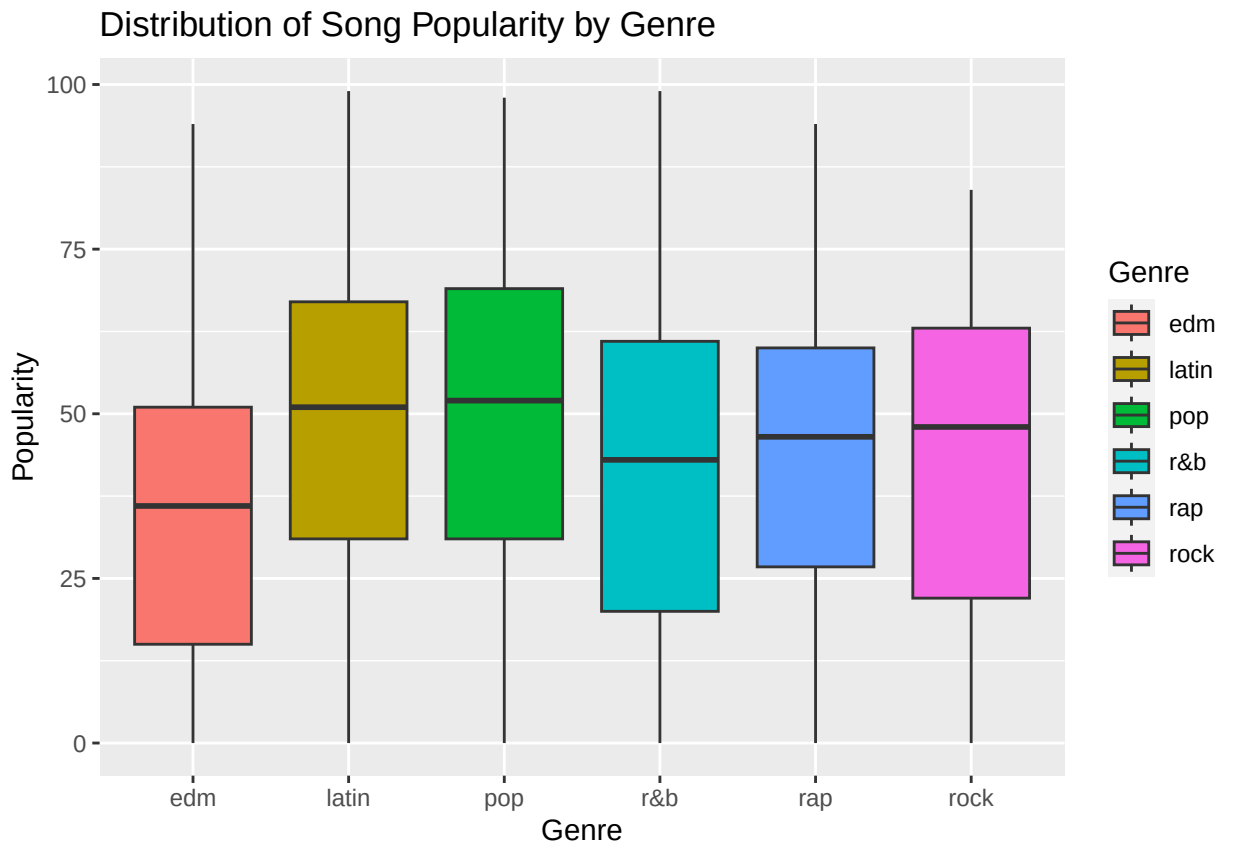
```
## Rows: 6,000
## Columns: 15
## $ track_name               <chr> "No Eyes - Radio Edit", "Gold Skies - Tiësto ~
## $ playlist_genre           <fct> edm, edm, edm, edm, edm, edm, edm, edm, edm, ~
## $ track_popularity         <dbl> 65, 37, 73, 29, 47, 27, 44, 65, 49, 0, 20, 57~
## $ key                      <dbl> 3, 2, 11, 6, 2, 0, 7, 4, 6, 1, 7, 1, 1, 7, 5,~
## $ instrumentalness         <dbl> 2.44e-03, 8.01e-01, 0.00e+00, 1.72e-06, 1.36e~
## $ duration_ms              <dbl> 204992, 306133, 210627, 207160, 182800, 20596~
## $ loudness                 <dbl> -9.925, -2.387, -2.457, -2.307, -2.436, -10.3~
## $ acousticness             <dbl> 0.028900, 0.000169, 0.135000, 0.298000, 0.003~
## $ valence                  <dbl> 0.5700, 0.4310, 0.7510, 0.7780, 0.6600, 0.191~
## $ energy                   <dbl> 0.483, 0.972, 0.716, 0.748, 0.969, 0.347, 0.3~
## $ liveness                 <dbl> 0.1030, 0.1490, 0.2510, 0.0540, 0.2530, 0.167~
## $ tempo                    <dbl> 120.022, 128.020, 125.008, 140.084, 150.010, ~
## $ danceability             <dbl> 0.930, 0.536, 0.747, 0.889, 0.592, 0.797, 0.8~
## $ track_album_release_date <date> 2013-08-16, 2014-08-16, 2012-08-16, 2018-08-~
## $ speechiness              <dbl> 0.0411, 0.0737, 0.0750, 0.0769, 0.0487, 0.066~
```

Question 1: Does the popularity of songs differ between genres?

```r
# Creating a plot using ggplot
ggplot(spotify_data_select, aes(x = playlist_genre, y = track_popularity, fill = playlist_genre)) +
```

```
  geom_boxplot() +
  labs(x = "Genre", y = "Popularity", title = "Distribution of Song Popularity by Genre") +
  scale_fill_hue(name = "Genre")
```

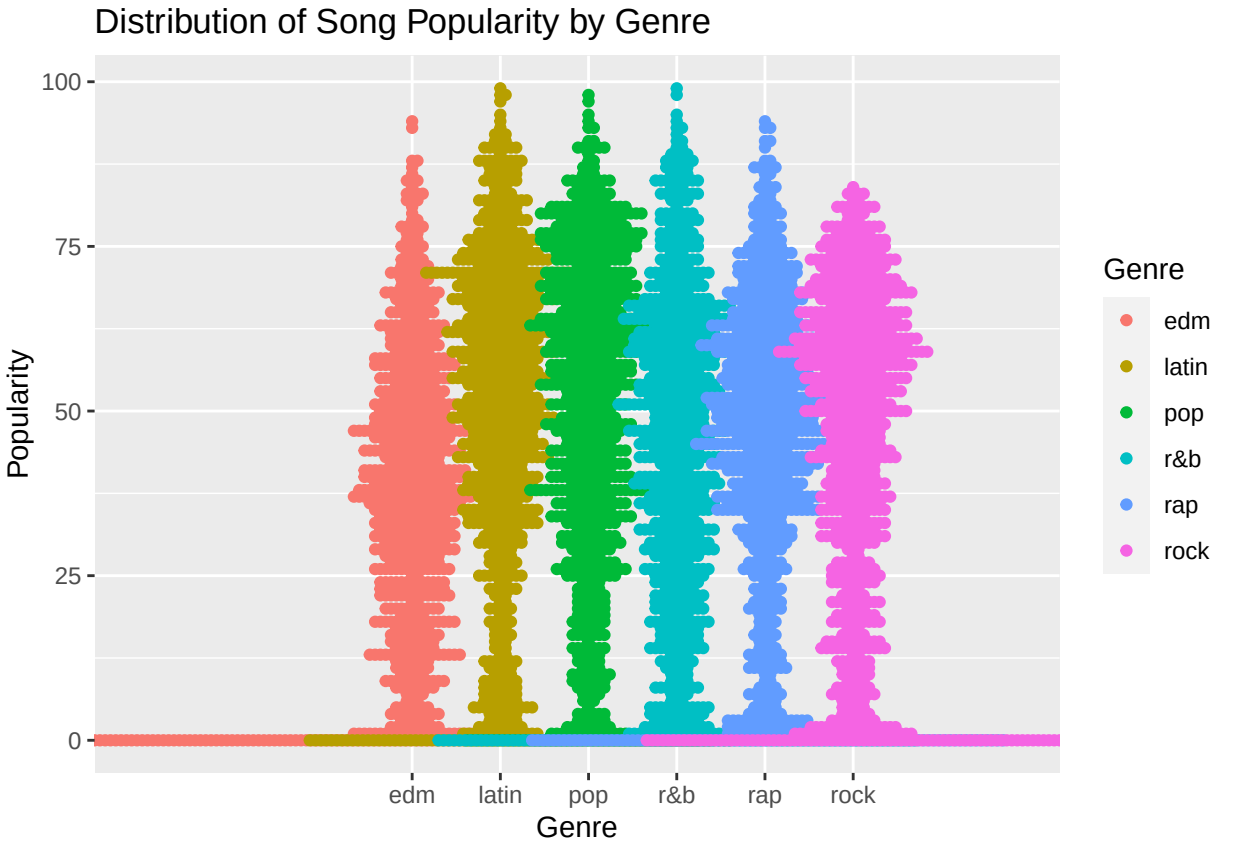
Below is the first figure to answer our question:



Explanation -> A visualization of the distribution of song popularity by genre, using a subset of 1,000 songs from each genre sourced from the Spotify music streaming service, is shown in the image. The y-axis measures song popularity on a scale of 0 to 100, while the x-axis delineates various musical genres. Different fill colors serve as genre category indicators. Notably, the visualization reveals differences in song popularity across genres, with some genres (notably *Pop* and *Latin* genres) showing noticeably elevated median popularity values in contrast to others.

```
# Creating a second plot
ggplot(spotify_data_select, aes(x = playlist_genre, y = track_popularity, color = playlist_genre)) +
  geom_beeswarm() +
  labs(x = "Genre", y = "Popularity", title = "Distribution of Song Popularity by Genre") +
  scale_color_hue(name = "Genre")
```

Here is another figure to support our answer:

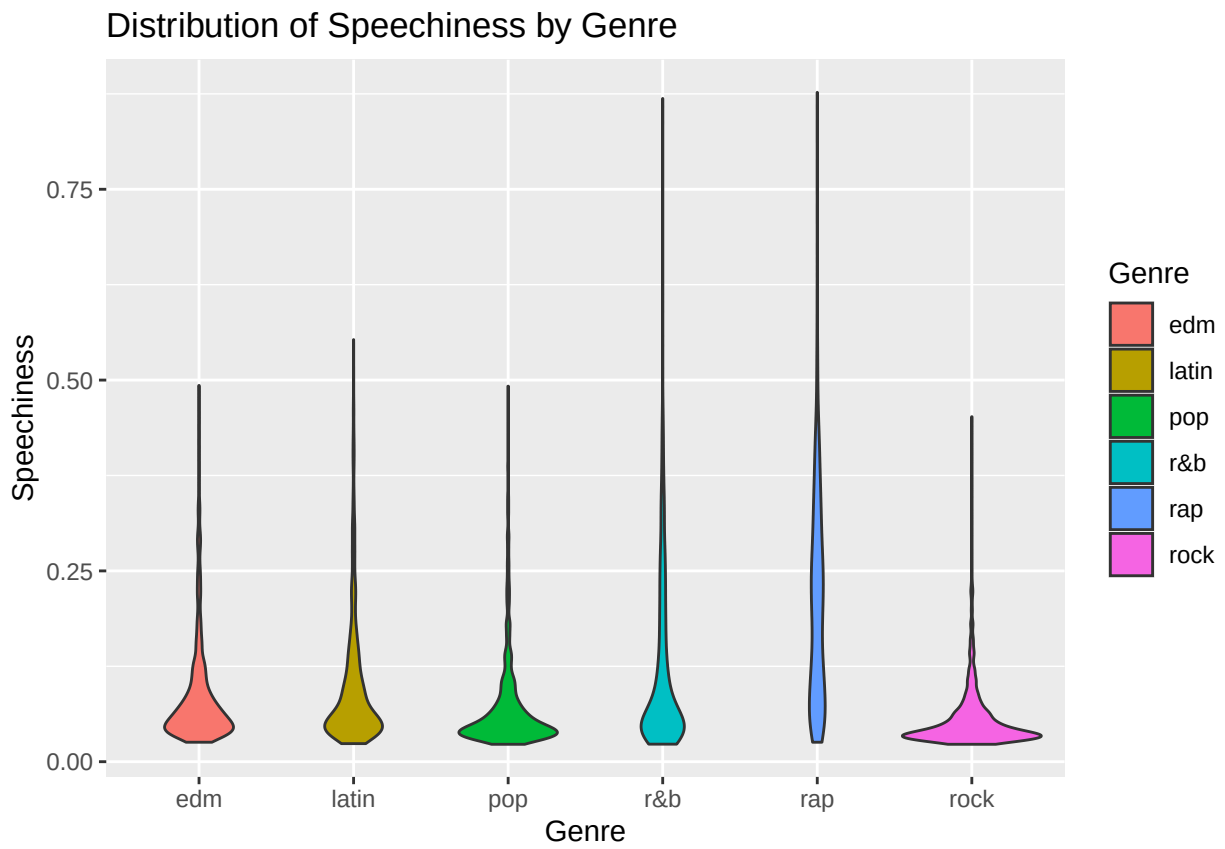


The above code creates a *beeswarm plot* that efficiently depicts the popularity of songs across different genres. The "spotify_data_select" label refers to the data-set used for this analysis. The y-axis measures the popularity of the tracks, while the x-axis depicts the various genres. The fill color's use of various tones makes it easier to distinguish between various genres. To further improve its interpret ability, the plot is thoughtfully enhanced with a title and appropriately labeled axes.

Question 2: Is there a difference in speechiness for each genre?

```
# Developing a graph of genre v/s speechiness
ggplot(spotify_data_select, aes(x = playlist_genre, y = speechiness, fill = playlist_genre)) +
  geom_violin() +
  labs(x = "Genre", y = "Speechiness", title = "Distribution of Speechiness by Genre") +
  scale_fill_hue(name = "Genre")
```

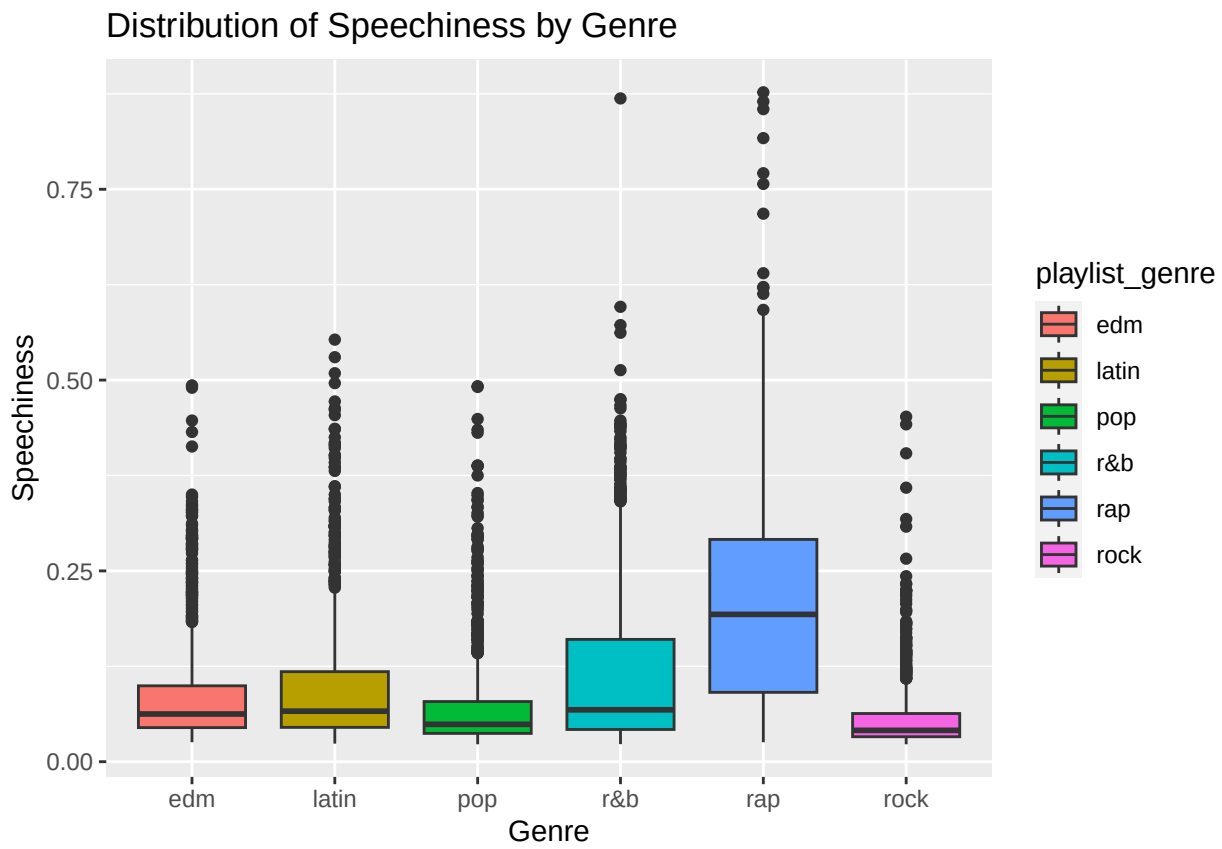
Below is the first figure to answer our question:



The plot that is being presented uses a subset of 1,000 songs from each genre on the music streaming service Spotify to demonstrate the differences in speechiness between genres. Different music genres are listed on the x-axis, and speechiness scores, which are normalized between 0 and 1, are quantified on the y-axis. Different fill colors are used to identify different genre categories. The plot's observed divergence in speechiness values across genres is noteworthy, with some genres showing noticeably higher median speechiness than others.

```
# Generating another figure to support our answer.
ggplot(spotify_data_select, aes(x = playlist_genre, y = speechiness, fill = playlist_genre)) +
  geom_boxplot() +
  labs(x = "Genre", y = "Speechiness", title = "Distribution of Speechiness by Genre") +
  scale_color_hue(name = "Genre")
```

Here is another figure to support our answer:

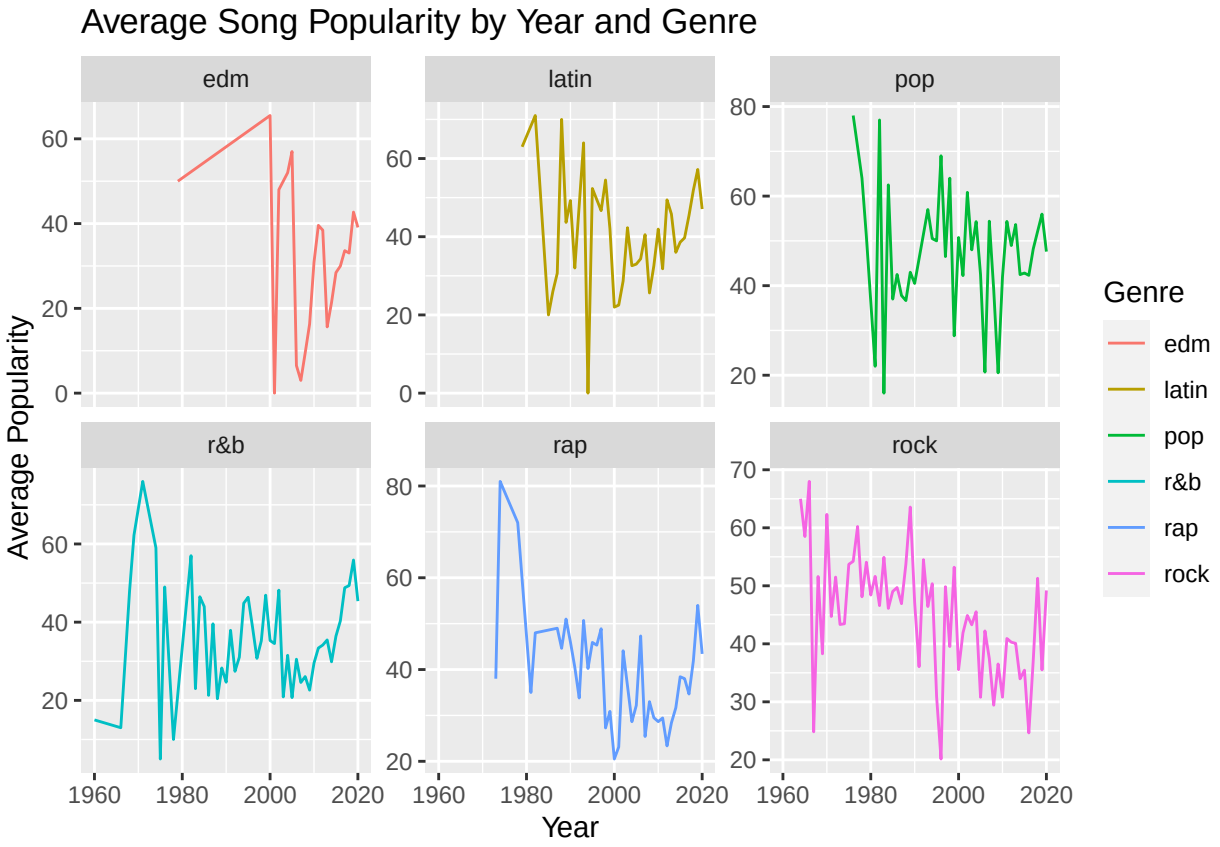


The provided boxplot, which uses information from the "spotify_data_select" dataset, graphically illustrates the distribution of speechiness across genres. Different music genres are categorized on the x-axis, and speechiness scores, normalized within the range of 0 to 1, are quantified on the y-axis. A palette of distinctive colors is used to distinguish between various genres. This plot's interpretability is enhanced by a pertinent title and appropriately labeled axes. It is noteworthy that the boxplot suggests differences in speechiness between genres, with some genres showing higher median speechiness values than others.

Question 3: How does track popularity change over time?

```
spotify_data_select %>%
  group_by(playlist_genre, year = year(track_album_release_date)) %>%
  summarize(avg_popularity = mean(track_popularity), .groups = 'drop') %>%
  ggplot(aes(x = year, y = avg_popularity, color = playlist_genre)) +
  geom_line() +
  labs(x = "Year", y = "Average Popularity", title = "Average Song Popularity by Year and Genre") +
  scale_color_hue(name = "Genre") +
  facet_wrap(~ playlist_genre, scales = "free_y")
```

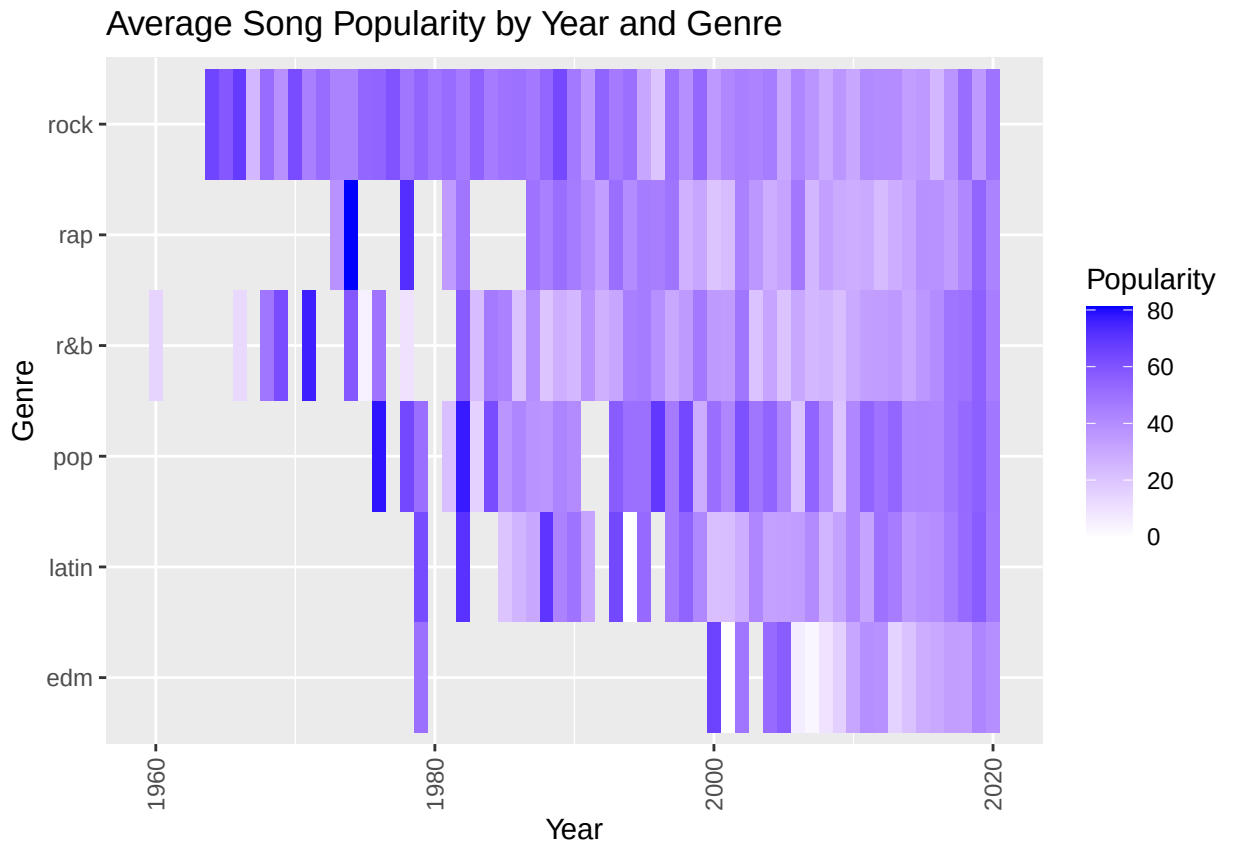
Below is the first figure to answer our question:



The graphic depiction shows how the average song's popularity has changed over time within each genre. Data was aggregated by year and genre for this analysis, and the mean popularity for each grouping was then calculated. The resulting story is divided into distinct panels, each of which is devoted to a different genre. These panels display line plots showing the average popularity's trajectory over time. Years are indicated on the x-axis, and average popularity is quantified on the y-axis. The differentiation of genres is made easier by the use of distinctive colors. The plot's implications illustrate the dynamic nature of typical song popularity by highlighting temporal variations and disparities across various genres.

```
spotify_data_select %>%
  group_by(playlist_genre, year = year(track_album_release_date)) %>%
  summarize(avg_popularity = mean(track_popularity), .groups = 'drop') %>%
  ggplot(aes(x = year, y = playlist_genre, fill = avg_popularity)) +
  geom_tile() +
  labs(x = "Year", y = "Genre", title = "Average Song Popularity by Year and Genre") +
  scale_fill_gradient(name = "Popularity", low = "white", high = "blue") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5))
```

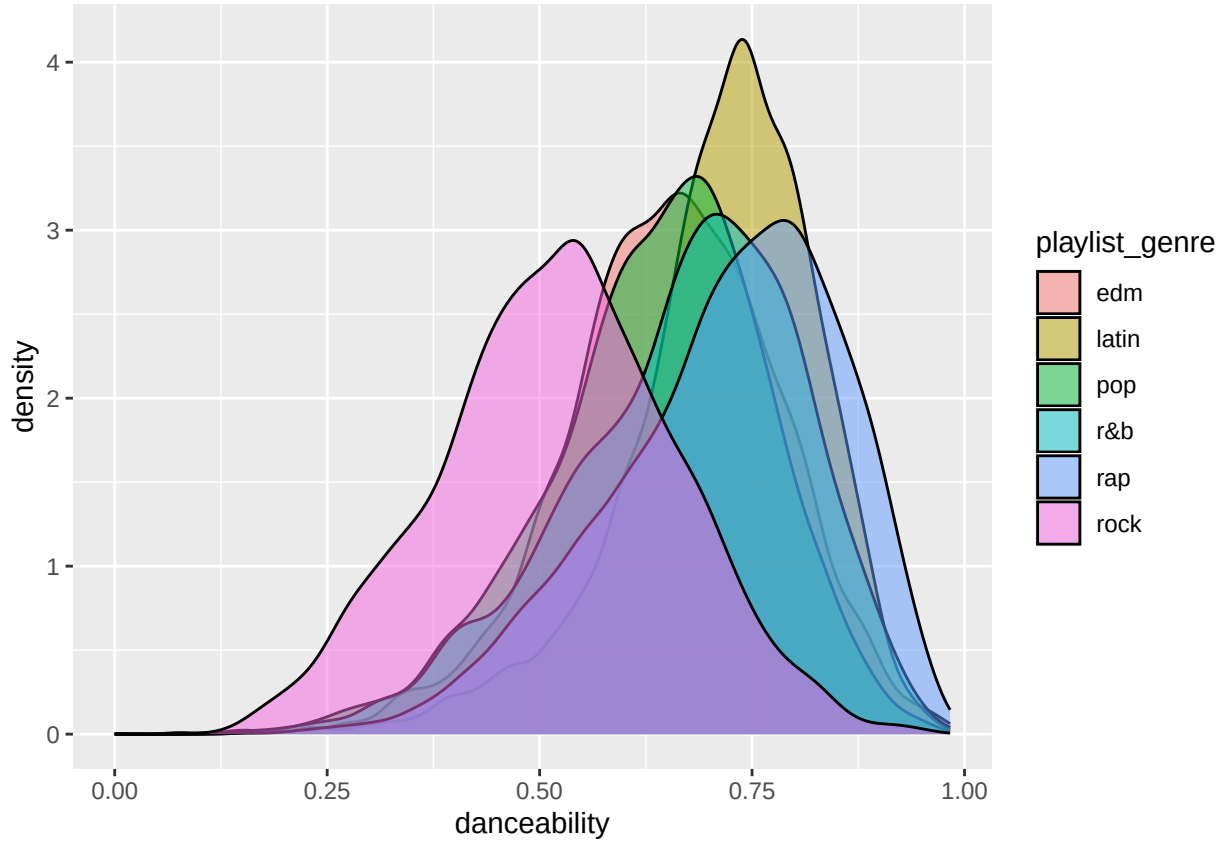
Here is another figure to support our answer:



By creating a tiled heatmap using the provided code, the average song popularity in relation to year and genre can be seen. The dataset had been organized into groups based on genre and year prior to this visualization. The average track popularity for each grouping is calculated. The plot that results uses the year as the x-axis, the genre as the y-axis, and the fill color to show average popularity. Utilizing the scale_fill_gradient function makes it easier to alter the color scheme of the heatmap. Additionally, the theme function is used to rotate the x-axis labels by 90 degrees in order to improve legibility and avoid overlap. This combination of customization and visualization options enables a thorough comprehension of the relationship between year, genre, and average song popularity.

```
# The below code helps us to visualize the relationship between the attributes- 'playlist_genre' and 'd
ggplot(spotify_data, aes(x = danceability, fill = playlist_genre)) +
  geom_density(alpha = 0.5)
```

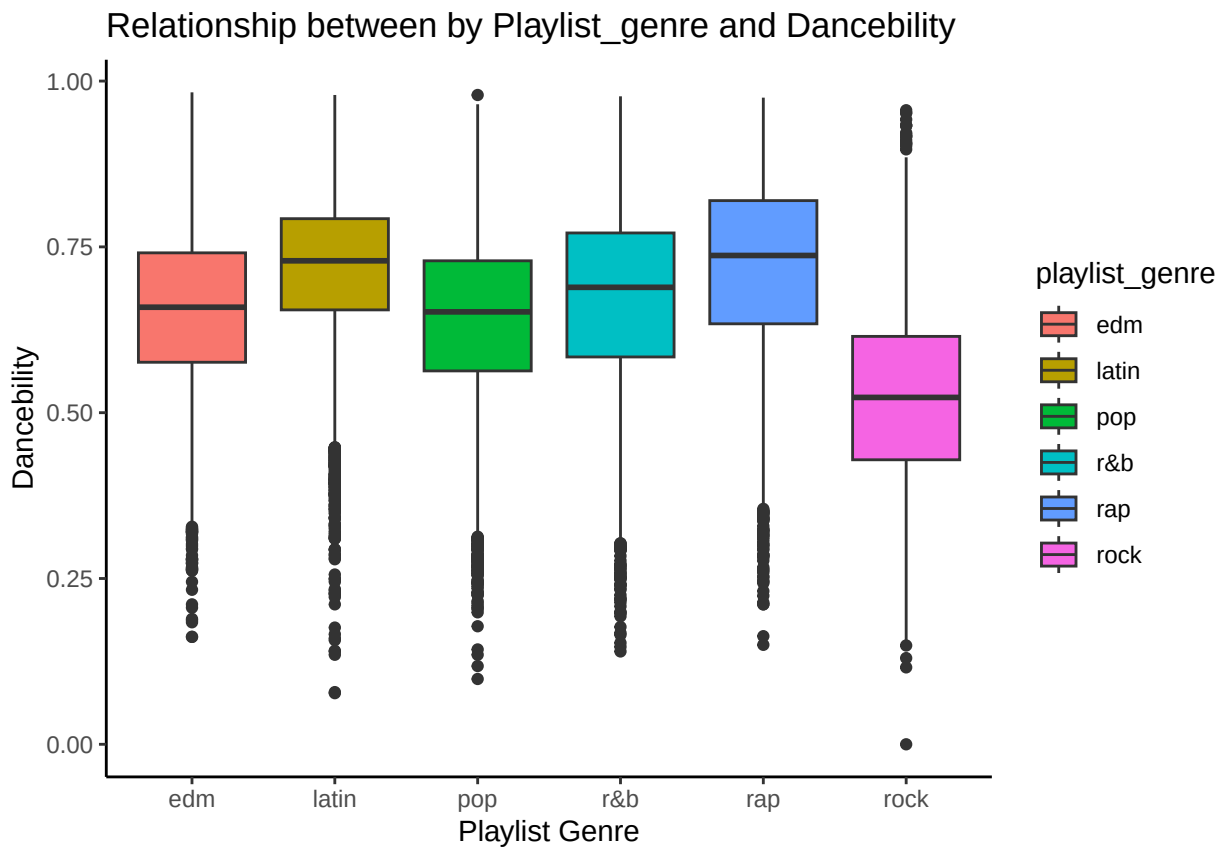
For additional information, we will perform a deeper analysis on "danceability" and "tempo".



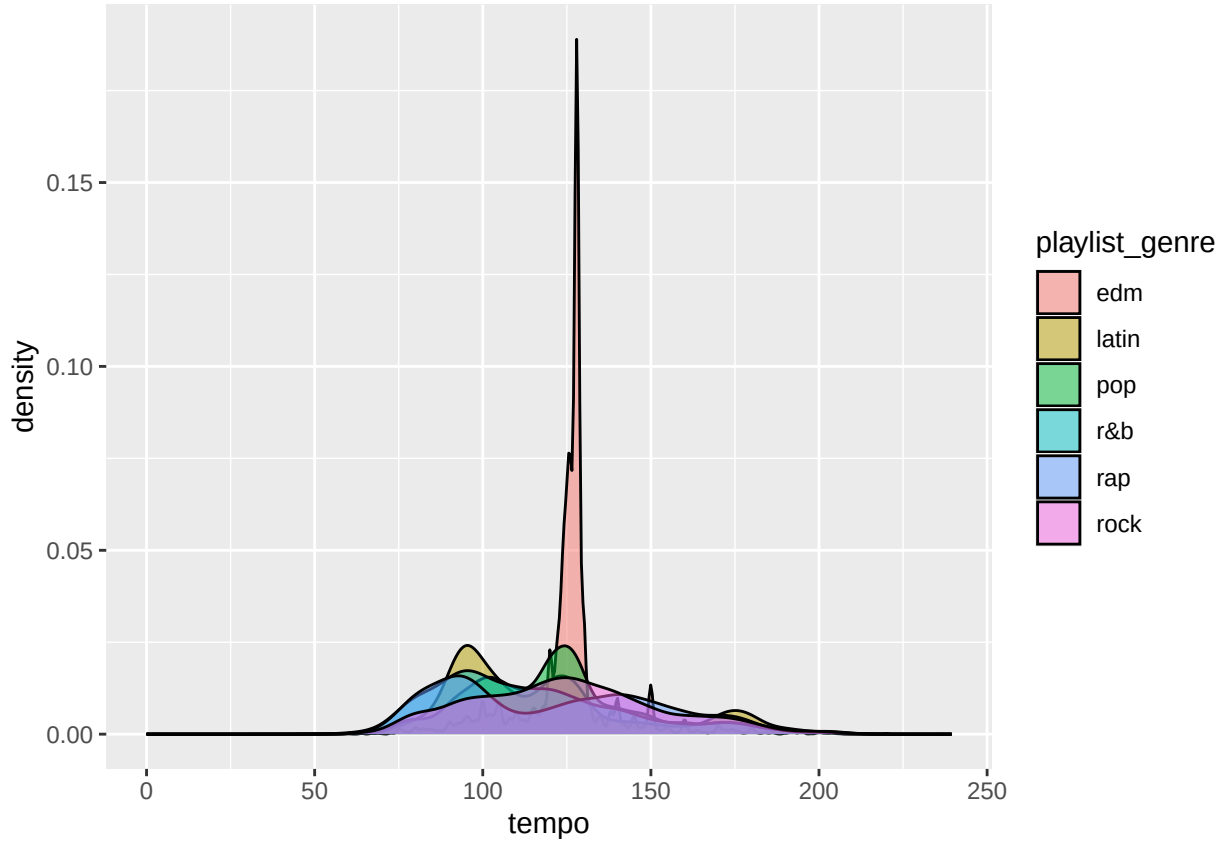
The plot effectively illustrates that while other genres appear to maintain relatively consistent danceability levels, the "Latin" genre exhibits the highest level of danceability. Only one particular variable's distribution is shown in the density plots.

Similar findings can be drawn from the accompanying boxplot, which is displayed below.

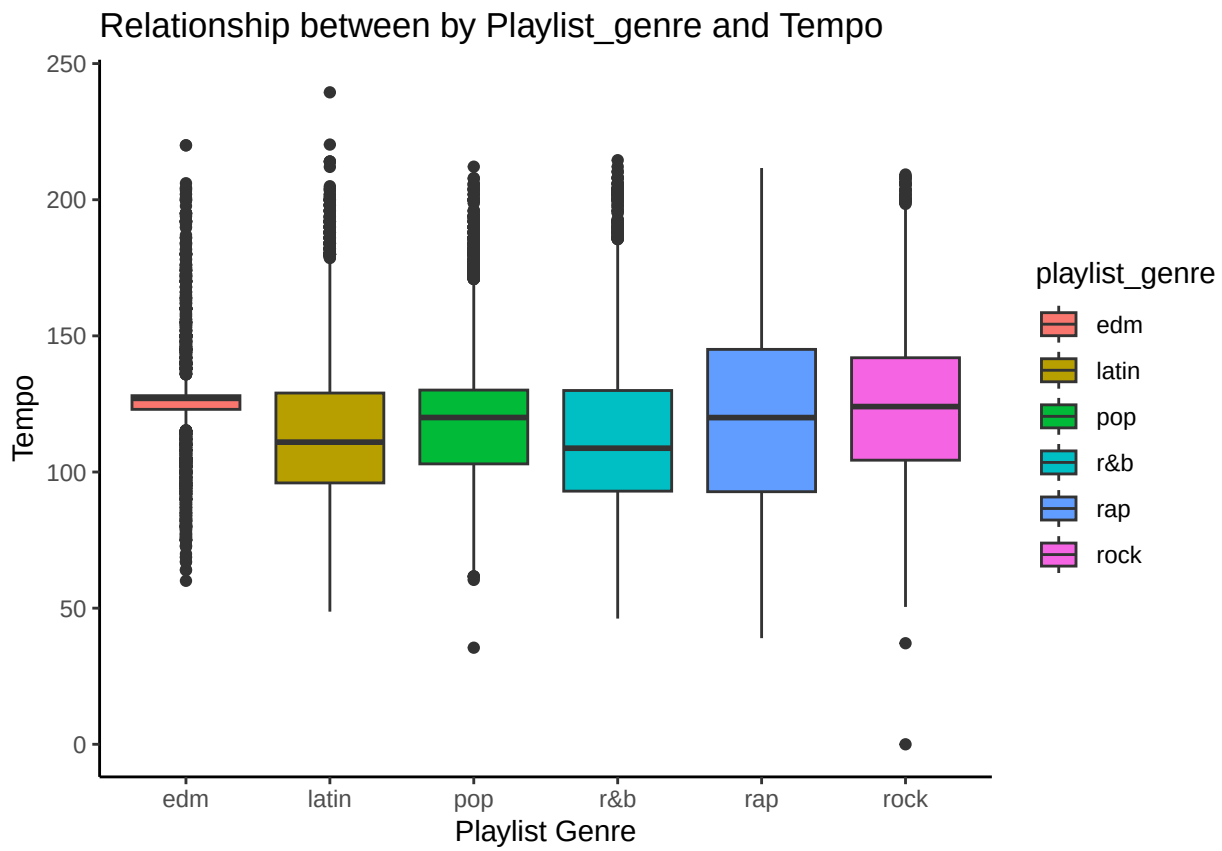
```
# The below code helps us to visualize the relationship between the attributes- 'playlist_genre' and 'd

ggplot(spotify_data, aes(x = playlist_genre, y = danceability, fill = playlist_genre)) +
  geom_boxplot()+
  labs(x = "Playlist Genre", y = "Dancebility") +
  ggtitle("Relationship between by Playlist_genre and Dancebility") +
  theme_classic()
```




```
# The below code helps us to visualize the relationship between the attributes- 'playlist_genre' and 't
ggplot(spotify_data, aes(x = tempo, fill = playlist_genre)) +
  geom_density(alpha = 0.5)
```




```
# The below code helps us to visualize the relationship between the attributes- 'playlist_genre' and 't
ggplot(spotify_data, aes(x = playlist_genre, y = tempo, fill = playlist_genre)) +
  geom_boxplot() +
  labs(x = "Playlist Genre", y = "Tempo") +
  ggtitle("Relationship between by Playlist_genre and Tempo") +
  theme_classic()
```



Rap exhibits the most *overt dominance*, while the *EDM tempo* appears to have the *highest interquartile range (IQR)*.

In order to decide which variables to include in our predictive model, we should now look at the correlations between the remaining variables. This is essential because not all predictions may be entirely accurate given the current variables.

```
nums <- unlist(lapply(spotify_data, is.numeric), use.names = FALSE)
num_col<-names(spotify_data[ , nums])
df_cor<-cor(spotify_data[num_col],method="pearson")
head(df_cor)
```

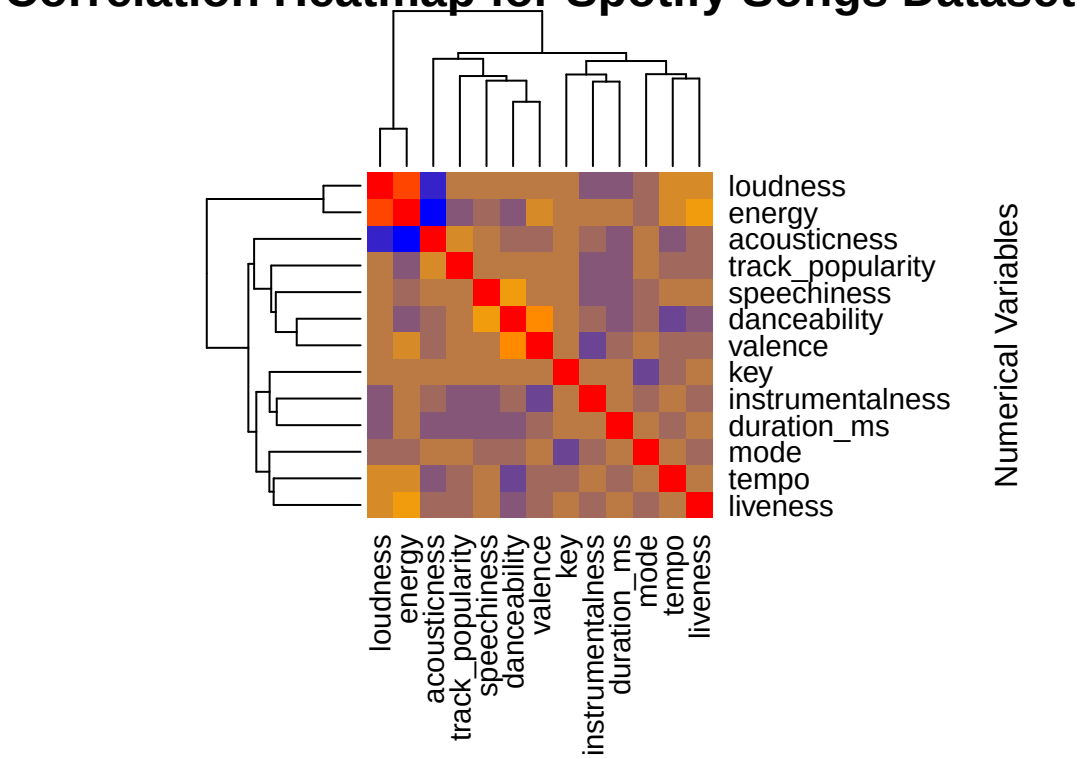
```
##                  track_popularity danceability       energy           key
## track_popularity     1.0000000000   0.06474767 -0.109111533 -0.0006503533
## danceability         0.0647476713   1.00000000 -0.086073156  0.0117364748
## energy              -0.1091115325  -0.08607316  1.000000000  0.0100516957
## key                 -0.0006503533   0.01173647  0.010051696  1.0000000000
## loudness             0.0576870774   0.02533509  0.676624523  0.0009586305
## mode                 0.0106365762  -0.05864740 -0.004799733 -0.1740929567
##                       loudness         mode  speechiness acousticness
## track_popularity  0.0576870774  0.010636576  0.006819442  0.085159337
## danceability      0.0253350882 -0.058647400  0.181721334 -0.024519058
## energy            0.6766245234 -0.004799733 -0.032149611 -0.539744630
## key               0.0009586305 -0.174092957  0.022606990  0.004305858
## loudness          1.0000000000 -0.019289482  0.010338981 -0.361638165
## mode             -0.0192894815  1.000000000 -0.063512355  0.009415361
```

```
##                  instrumentalness     liveness    valence        tempo
## track_popularity     -0.149872413 -0.054584440 0.03323133 -0.005378063
## danceability         -0.008655078 -0.123859417 0.33052326 -0.184084351
## energy                0.033246579  0.161223049 0.15110330  0.149951107
## key                   0.005967818  0.002887181 0.01991391 -0.013370199
## loudness             -0.147824018  0.077612601 0.05338356  0.093767360
## mode                 -0.006740665 -0.005548974 0.00261447  0.014329047
##                  duration_ms
## track_popularity -0.14368235
## danceability     -0.09687879
## energy            0.01261144
## key               0.01513931
## loudness         -0.11505750
## mode              0.01563373
```

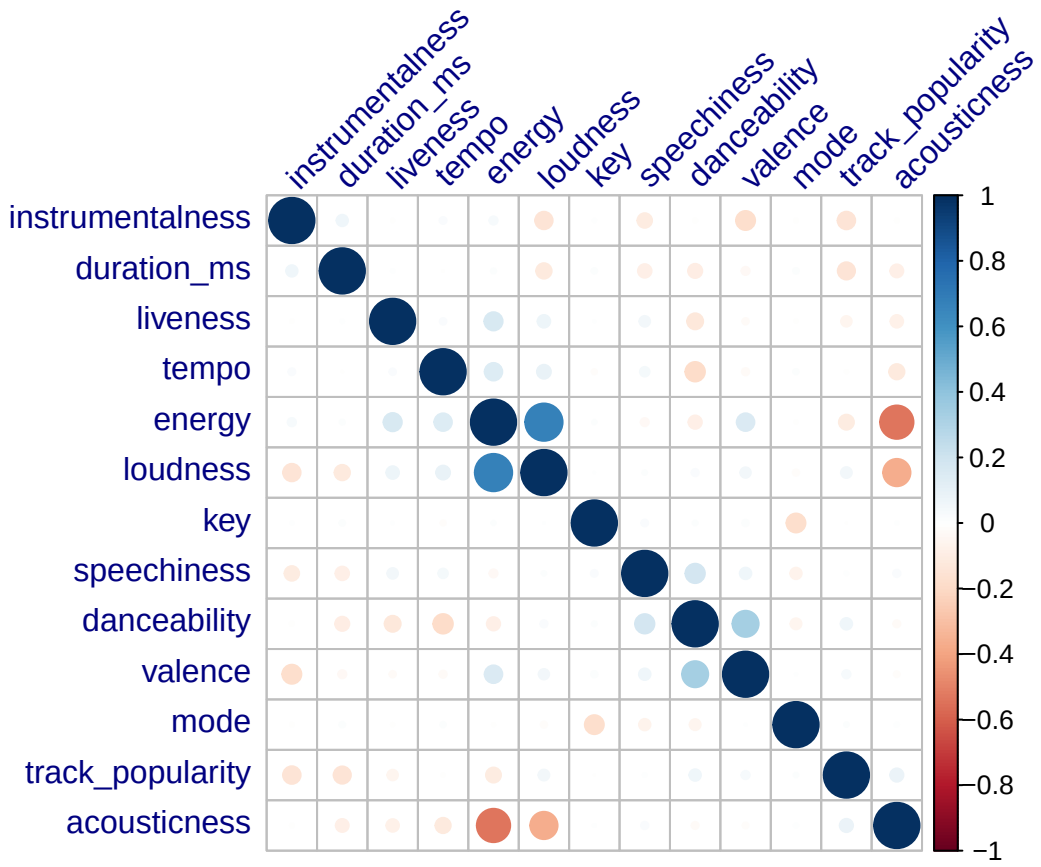
```
str(spotify_data[num_col])
```

```
## tibble [32,833 x 13] (S3: tbl_df/tbl/data.frame)
##  $ track_popularity: num [1:32833] 66 67 70 60 69 67 62 69 68 67 ...
##  $ danceability    : num [1:32833] 0.748 0.726 0.675 0.718 0.65 0.675 0.449 0.542 0.594 0.642 ...
##  $ energy          : num [1:32833] 0.916 0.815 0.931 0.93 0.833 0.919 0.856 0.903 0.935 0.818 ...
##  $ key             : num [1:32833] 6 11 1 7 1 8 5 4 8 2 ...
##  $ loudness        : num [1:32833] -2.63 -4.97 -3.43 -3.78 -4.67 ...
##  $ mode            : num [1:32833] 1 1 0 1 1 1 0 0 1 1 ...
##  $ speechiness     : num [1:32833] 0.0583 0.0373 0.0742 0.102 0.0359 0.127 0.0623 0.0434 0.0565 0.03
##  $ acousticness    : num [1:32833] 0.102 0.0724 0.0794 0.0287 0.0803 0.0799 0.187 0.0335 0.0249 0.05
##  $ instrumentalness: num [1:32833] 0.00 4.21e-03 2.33e-05 9.43e-06 0.00 0.00 0.00 4.83e-06 3.97e-06 0
##  $ liveness        : num [1:32833] 0.0653 0.357 0.11 0.204 0.0833 0.143 0.176 0.111 0.637 0.0919 ...
##  $ valence         : num [1:32833] 0.518 0.693 0.613 0.277 0.725 0.585 0.152 0.367 0.366 0.59 ...
##  $ tempo           : num [1:32833] 122 100 124 122 124 ...
##  $ duration_ms     : num [1:32833] 194754 162600 176616 169093 189052 ...
```

```
col<- colorRampPalette(c("blue", "orange", "red"))(20)
heatmap(x =df_cor, col = col, symm = TRUE,margins = c(10,10),
        main = "Correlation Heatmap for Spotify Songs Dataset",
        xlab = "Numerical Variables",
        ylab = "Numerical Variables")
```




```
# Corrplot for the correlation matrix
corrplot(df_cor, order = "hclust",
         tl.col = "navy", tl.srt = 45)
```



Notably, there is a strong collinearity between energy and acousticness, indicating limited utility in our analysis. It's crucial to realize that the plots we've shown provide a visual way to understand how the numerical variables in the "spotify_data" dataset interact with one another. It's important to stress once more that correlation does not imply causation. While these visualizations offer insightful information, a more thorough analysis may be necessary to fully comprehend the complex relationships between these variables.

```
# converting the response variable to a factor and eliminating extra variables before we begin modeling
spotify_data_select <- spotify_data_select %>%
  mutate(playlist_genre = factor(playlist_genre)) %>%
  dplyr::select( playlist_genre,track_popularity,key,instrumentalness,duration_ms,loudness,acousticness
  drop_na()

# Now that the data is cleaned again and subsets have been made, we will Check the structure
glimpse(spotify_data_select)
```

```
## Rows: 6,000
## Columns: 14
## $ playlist_genre          <fct> edm, edm, edm, edm, edm, edm, edm, edm, edm, ~
## $ track_popularity        <dbl> 65, 37, 73, 29, 47, 27, 44, 65, 49, 0, 20, 57~
## $ key                     <dbl> 3, 2, 11, 6, 2, 0, 7, 4, 6, 1, 7, 1, 1, 7, 5,~
## $ instrumentalness        <dbl> 2.44e-03, 8.01e-01, 0.00e+00, 1.72e-06, 1.36e~
## $ duration_ms             <dbl> 204992, 306133, 210627, 207160, 182800, 20596~
## $ loudness                <dbl> -9.925, -2.387, -2.457, -2.307, -2.436, -10.3~
## $ acousticness            <dbl> 0.028900, 0.000169, 0.135000, 0.298000, 0.003~
## $ valence                 <dbl> 0.5700, 0.4310, 0.7510, 0.7780, 0.6600, 0.191~
## $ energy                  <dbl> 0.483, 0.972, 0.716, 0.748, 0.969, 0.347, 0.3~
```

```
## $ liveness                 <dbl> 0.1030, 0.1490, 0.2510, 0.0540, 0.2530, 0.167~
## $ tempo                    <dbl> 120.022, 128.020, 125.008, 140.084, 150.010, ~
## $ danceability             <dbl> 0.930, 0.536, 0.747, 0.889, 0.592, 0.797, 0.8~
## $ track_album_release_date <date> 2013-08-16, 2014-08-16, 2012-08-16, 2018-08-~
## $ speechiness              <dbl> 0.0411, 0.0737, 0.0750, 0.0769, 0.0487, 0.066~
```

Model Building

Model 1 -> Linear Discriminant Analysis (LDA)

```r
# Loading the necessary Libraries
pacman::p_load(dplyr,tidymodels,discrim,MASS,caret)

# Load and preprocess the data
# Split the data into training and testing sets
set.seed(1881450)
spotify_split <- initial_split(spotify_data_select)
spotify_train <- training(spotify_split)
spotify_test <- testing(spotify_split)

# Fit the LDA model using the MASS engine
spotify_lda <- lda(playlist_genre ~ ., data = spotify_train)

# Make predictions on the test set
spotify_preds <- predict(spotify_lda, newdata = spotify_test)

# Evaluate model performance
confusionMatrix(spotify_preds$class, spotify_test$playlist_genre)
```

Implementation 1

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction edm latin pop r&b rap rock
##      edm   139    27  36   9  27   16
##      latin  35   110  60  38  58    9
##      pop    51    37  94  47  24   32
##      r&b    14    21  17  79  28   20
##      rap    14    32  18  45 143    3
##      rock    1    10  18  32  11  145
##
## Overall Statistics
##
##                Accuracy : 0.4733
##                  95% CI : (0.4478, 0.499)
##     No Information Rate : 0.194
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.368
##
##  Mcnemar's Test P-Value : 4.257e-09
##
## Statistics by Class:
```

```
##
##                      Class: edm Class: latin Class: pop Class: r&b Class: rap
## Sensitivity             0.54724      0.46414    0.38683    0.31600    0.49141
## Specificity             0.90770      0.84165    0.84805    0.92000    0.90736
## Pos Pred Value          0.54724      0.35484    0.32982    0.44134    0.56078
## Neg Pred Value          0.90770      0.89328    0.87737    0.87055    0.88112
## Prevalence              0.16933      0.15800    0.16200    0.16667    0.19400
## Detection Rate          0.09267      0.07333    0.06267    0.05267    0.09533
## Detection Prevalence    0.16933      0.20667    0.19000    0.11933    0.17000
## Balanced Accuracy       0.72747      0.65289    0.61744    0.61800    0.69939
##                      Class: rock
## Sensitivity              0.64444
## Specificity              0.94353
## Pos Pred Value           0.66820
## Neg Pred Value           0.93765
## Prevalence               0.15000
## Detection Rate           0.09667
## Detection Prevalence     0.14467
## Balanced Accuracy        0.79399
```

Using the knowledge gathered from above, *0.473* is the accuracy, and *0.448* to *0.499* is the 95% confidence interval.

```
set.seed(1881450)
# Perform cross-validation on the spotify_data_select dataset with 10 stratified folds
spotify_data_select_cv <- vfold_cv(spotify_data_select, v = 10, strata = playlist_genre)

spotify_spec <- discrim_linear(mode = "classification") %>% set_engine("MASS")

set.seed(1881450)

# Fit the model using cross-validation and collect results
spotify_spec_resamples <- fit_resamples(object = spotify_spec,
                                        preprocessor = recipe(playlist_genre ~ ., data = spotify_data_s
                                        resamples = spotify_data_select_cv)
spotify_spec_resamples %>% collect_metrics()
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy multiclass 0.487    10 0.00614 Preprocessor1_Model1
## 2 roc_auc  hand_till  0.804    10 0.00488 Preprocessor1_Model1
```

After using a different method to implement LDA, we observe a similar pattern of outcomes, with accuracy of *0.487* and roc_auc of *0.804*.

Model 2 -> K-Nearest Neighbors Model

```
set.seed(1881450)
# Load the required packages
pacman::p_load(recipes,themis,yardstick,tune)

# Preprocessimg the data
spotify_recipe_data <- recipe( playlist_genre ~ ., data = spotify_train ) %>%
  themis::step_downsample( playlist_genre ) %>%
```

```
  step_date(track_album_release_date) %>%
  step_rm() %>%
  prep()
spotify_recipe_data
```

Before we start, we will pre-process the data:

```
##
## -- Recipe ----------------------------------------------------------------------
##
## -- Inputs
## Number of variables by role
## outcome:    1
## predictor: 13
##
## -- Training information
## Training data contained 4500 data points and no incomplete rows.
##
## -- Operations
## * Down-sampling based on: playlist_genre | Trained
## * Date features from: track_album_release_date | Trained
## * Variables removed: <none> | Trained
```

1- *step_date(track_album_release_date)*: This step is used to transform the track_album_release_date variable into a date format, allowing for proper handling of date-related operations and comparisons during pre-processing and analysis.

2- *step_rm()*: This step removes any variables that were not specified in the formula of the recipe, ensuring that only the relevant predictor variables are retained in the pre-processed data-set, simplifying subsequent analysis and modeling.

```
set.seed(1881450)
spotify_train_preproc <- juice( spotify_recipe_data )
spotify_test_preproc <- bake( spotify_recipe_data, spotify_test )
head(spotify_test_preproc)
```

```
## # A tibble: 6 x 17
##   track_popularity   key instrumentalness duration_ms loudness acousticness
##              <dbl> <dbl>            <dbl>       <dbl>    <dbl>        <dbl>
## 1               65     3       0.00244         204992    -9.93       0.0289
## 2               73    11       0               210627    -2.46       0.135
## 3               29     6       0.00000172      207160    -2.31       0.298
## 4                0     7       0.748           190693    -6.69       0.0102
## 5               18     3       0.902           206462    -5.96       0.041
## 6                8     9       0.00306         216765    -5.82       0.0891
## # i 11 more variables: valence <dbl>, energy <dbl>, liveness <dbl>,
## #   tempo <dbl>, danceability <dbl>, track_album_release_date <date>,
## #   speechiness <dbl>, playlist_genre <fct>,
## #   track_album_release_date_dow <fct>, track_album_release_date_month <fct>,
```

```
## #   track_album_release_date_year <int>
```

```
spotify_train_preproc %>%
  skim_without_charts()
```

Table 1: Data summary

Name	Piped data
Number of rows	4254
Number of columns	17
Column type frequency:	
Date	1
factor	3
numeric	13
Group variables	None

Variable type: Date

skim_variable	n_missing	complete_rate	min	max	median	n_unique
track_album_release_date	0	1	1960-08-16	2020-08-16	2016-08-16	58

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
playlist_genre	0	1	FALSE	6	edm: 709, lat: 709, pop: 709, r&b: 709
track_album_release_date_dow	0	1	FALSE	7	Fri: 1375, Thu: 708, Tue: 509, Wed: 508
track_album_release_date_month	0	1	FALSE	1	Aug: 4254, Jan: 0, Feb: 0, Mar: 0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
track_popularity	0	1	42.52	25.26	0.00	23.00	46.00	62.00	99.00
key	0	1	5.38	3.60	0.00	2.00	6.00	9.00	11.00
instrumentalness	0	1	0.08	0.22	0.00	0.00	0.00	0.00	0.98
duration_ms	0	1	225063.65	59493.10	29493.00	186813.00	216024.00	253623.50	513440.00
loudness	0	1	-6.70	3.06	-35.96	-8.14	-6.02	-4.67	1.14
acousticness	0	1	0.18	0.22	0.00	0.01	0.08	0.26	0.98
valence	0	1	0.52	0.23	0.01	0.34	0.52	0.70	0.98
energy	0	1	0.70	0.18	0.00	0.58	0.72	0.84	1.00
liveness	0	1	0.19	0.15	0.02	0.09	0.13	0.25	0.99
tempo	0	1	121.10	27.44	50.45	99.48	121.98	135.24	219.99
danceability	0	1	0.65	0.15	0.08	0.56	0.67	0.76	0.98
speechiness	0	1	0.11	0.10	0.02	0.04	0.06	0.13	0.88
track_album_release_date_year	0	1	2011.00	11.44	1960.00	2008.00	2016.00	2019.00	2020.00

```
# Define the KNN specification
set.seed(1881450)
knn_spec <- nearest_neighbor( mode = "classification", neighbors = tune() ) %>%
  set_engine( "kknn" )
```

```
# Perform 5-fold cross validation
set.seed(1881450)
spotify_cv <- vfold_cv( spotify_train_preproc, v = 5 ,strata = playlist_genre)
```

```
# Define the tuning grid
k_grid <- grid_regular( neighbors( range = c( 1, 100 ) ),
                        levels = 20 )
```

```
# Tune the KNN model using grid search
knn_tune <- tune_grid(object = knn_spec,
                      preprocessor = recipe(playlist_genre ~ .,
                                            data = spotify_train_preproc),
                      resamples = spotify_cv,
                      grid = k_grid )
```

Now we will be tuning and then fitting a model

```
## Warning: package 'kknn' was built under R version 4.3.1
```

```
set.seed(1881450)
# Select the model with the best accuracy
best_acc <- select_best( knn_tune, "accuracy")
best_acc
```

```
## # A tibble: 1 x 2
##   neighbors .config
##       <int> <chr>
## 1        47 Preprocessor1_Model10
```

We can clearly see that the value of k, gives us the best accuracy possible, which is 89.

```
# Finalize the model using the selected hyperparameters
knn_spec_final <- finalize_model( knn_spec, best_acc )
knn_spec_final
```

```
## K-Nearest Neighbor Model Specification (classification)
##
## Main Arguments:
##   neighbors = 47
##
## Computational engine: kknn
```

```
# Fit the final KNN model to the training data
spotify_knn <- knn_spec_final %>%
  fit( playlist_genre ~ . , data = spotify_train_preproc )
spotify_knn
```

```
## parsnip model object
##
```

```
##
## Call:
## kknn::train.kknn(formula = playlist_genre ~ ., data = data, ks = min_rows(47L,     data, 5))
##
## Type of response variable: nominal
## Minimal misclassification: 0.4757875
## Best kernel: optimal
## Best k: 47
```

```
# Make predictions on the test data
knn_preds <- predict( spotify_knn,
                      spotify_test_preproc,
                      type = "class" )
knn_preds %>%
  head()
```

```
## # A tibble: 6 x 1
##   .pred_class
##   <fct>
## 1 rap
## 2 pop
## 3 latin
## 4 edm
## 5 edm
## 6 latin
```

```
# Bind the predicted values to the test data
knn_preds <- knn_preds %>%
  bind_cols( dplyr::select( spotify_test_preproc, playlist_genre) )
knn_preds %>%
  head()
```

```
## # A tibble: 6 x 2
##   .pred_class playlist_genre
##   <fct>       <fct>
## 1 rap         edm
## 2 pop         edm
## 3 latin       edm
## 4 edm         edm
## 5 edm         edm
## 6 latin       edm
```

```
# Calculate the confusion matrix and other metrics
knn_preds %>%
  conf_mat( truth = playlist_genre, estimate = .pred_class )
```

```
##           Truth
## Prediction edm latin pop r&b rap rock
##      edm   168    28  50  17  37   19
##      latin  21   101  45  36  34    3
##      pop    40    46  86  39  26   28
##      r&b     9    26  27  97  33   22
##      rap    10    29  21  41 148    3
##      rock    6     7  14  20  13  150
```

```
knn_preds %>%
  sens( truth = playlist_genre, estimate = .pred_class ) %>%
```

```
  bind_rows( knn_preds %>%
               spec( truth = playlist_genre, estimate = .pred_class ) )
```

```
## # A tibble: 2 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 sens    macro          0.501
## 2 spec    macro          0.900
```

The above model has a sensitivity and specificity of 0.501 and 0.900, respectively.

```
# Collect the tuning metrics
head(tail(knn_tune %>%
            collect_metrics(),5))
```

```
## # A tibble: 5 x 7
##   neighbors .metric  .estimator  mean     n std_err .config
##       <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1        89 roc_auc  hand_till  0.829     5 0.00427 Preprocessor1_Model18
## 2        94 accuracy multiclass 0.515     5 0.00664 Preprocessor1_Model19
## 3        94 roc_auc  hand_till  0.829     5 0.00428 Preprocessor1_Model19
## 4       100 accuracy multiclass 0.515     5 0.00686 Preprocessor1_Model20
## 5       100 roc_auc  hand_till  0.829     5 0.00428 Preprocessor1_Model20
```

This model's highest accuracy and ROC_Auc are 0.515 and 0.829, respectively.

Model 3 -> K-Nearest Neighbors Model

```
# Set the seed for reproducibility
set.seed(1881450)
```

```
# Fit a random forest model on the preprocessed training set
spotify_rf <- randomForest(playlist_genre ~ ., data = spotify_train_preproc,
                           ntree = 100, maxdepth = 5)
```

```
# Use the fitted model to make predictions on the preprocessed test set
spotify_preds <- predict(spotify_rf, newdata = spotify_test_preproc)
```

```
# Evaluate the performance of the random forest model using the confusion matrix
confusionMatrix(spotify_preds, spotify_test_preproc$playlist_genre)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction edm latin pop r&b rap rock
##      edm   179    17  42   9  25    6
##      latin  25   114  45  33  27    6
##      pop    30    41  84  32  14   23
##      r&b     4    28  27 102  44   20
##      rap    12    30  23  52 173    5
##      rock    4     7  22  22   8  165
##
## Overall Statistics
##
##                Accuracy : 0.5447
##                  95% CI : (0.5191, 0.5701)
```

```
##     No Information Rate : 0.194
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4528
##
##  Mcnemar's Test P-Value : 0.4276
##
## Statistics by Class:
##
##                      Class: edm Class: latin Class: pop Class: r&b Class: rap
## Sensitivity              0.7047       0.4810     0.3457     0.4080     0.5945
## Specificity              0.9205       0.8923     0.8886     0.9016     0.8991
## Pos Pred Value           0.6439       0.4560     0.3750     0.4533     0.5864
## Neg Pred Value           0.9386       0.9016     0.8754     0.8839     0.9021
## Prevalence               0.1693       0.1580     0.1620     0.1667     0.1940
## Detection Rate           0.1193       0.0760     0.0560     0.0680     0.1153
## Detection Prevalence     0.1853       0.1667     0.1493     0.1500     0.1967
## Balanced Accuracy        0.8126       0.6867     0.6172     0.6548     0.7468
##                      Class: rock
## Sensitivity               0.7333
## Specificity               0.9506
## Pos Pred Value            0.7237
## Neg Pred Value            0.9528
## Prevalence                0.1500
## Detection Rate            0.1100
## Detection Prevalence      0.1520
## Balanced Accuracy         0.8420
```

The Accuracy of Random Forest Model is 0.545 with 95% Confidence of (0.519, 0.570)

```
pacman::p_load(tidyverse,knitr)

Final_Model_Collec <- tibble(
  `Model Name` = c(
    "Linear Discriminant Anaylsis (LDA) - Method/Approach 1",
    "Linear Discriminant Anaylsis (LDA) - Method/Approach 2",
    "K Nearest Neighbour (KNN)",
    "Random Forest (RF)"
  ),
  `Accuracy` = c(0.473, 0.487, 0.515, 0.545),
  `Lower CI` = c(0.448, NA, NA, 0.519),
  `Upper CI` = c(0.499, NA, NA, 0.570),
  `Sensitivity` = c(0.497, NA, 0.501, 0.574),
  `Specificity` = c(0.891, NA, 0.900, 0.914)
)

Final_Model_Collec %>%
  mutate(`95% CI` = glue::glue("{round(`Lower CI`, 4)}, {round(`Upper CI`, 4)}")) %>%
  dplyr::select(`Model Name`, `Accuracy`, `95% CI`, `Sensitivity`, `Specificity`) %>%
  knitr::kable(
    digits = 3,
    format.args = list(big.mark = ","),
    caption = "Accuracy, 95% Confidence Interval, Sensitivity, and Specificity for Four Different Models
```

```
  )
```

Now that all of our three models have been implemented, we will be finding out which is the best from the three.

Table 5: Accuracy, 95% Confidence Interval, Sensitivity, and Specificity for Four Different Models

Model Name	Accuracy	95% CI	Sensitivity	Specificity
Linear Discriminant Anaylsis (LDA) - Method/Approach 1	0.473	0.448, 0.499	0.497	0.891
Linear Discriminant Anaylsis (LDA) - Method/Approach 2	0.487	NA, NA	NA	NA
K Nearest Neighbour (KNN)	0.515	NA, NA	0.501	0.900
Random Forest (RF)	0.545	0.519, 0.57	0.574	0.914

Fitting The Best Model : Final Random Forest

```
# sets a specific seed value for the random number generator to ensure reproducibility
set.seed(1881450)
 # creates a random forest specification with hyperparameters "trees", "mtry", and "min_n"; sets the en
rf_spec_tune <- rand_forest( mode = "classification",
                             trees = 100,
                             mtry = tune(),
                             min_n = tune() ) %>%
  set_engine( "ranger", importance = "permutation" )
rf_spec_tune
```

```
## Random Forest Model Specification (classification)
##
## Main Arguments:
##   mtry = tune()
##   trees = 100
##   min_n = tune()
##
## Engine-Specific Arguments:
##   importance = permutation
##
## Computational engine: ranger
```

```
set.seed(1881450)
# generates a grid of hyperparameter values to search over using the "finalize" function to set the "mt
params_grid <- grid_regular( finalize( mtry(), spotify_train %>% dplyr::select( -playlist_genre ) ),
                             min_n(),
                             levels = 5)


params_grid
```

```
## # A tibble: 25 x 2
##     mtry min_n
##    <int> <int>
##  1     1     2
##  2     4     2
##  3     7     2
```

```
##  4    10     2
##  5    13     2
##  6     1    11
##  7     4    11
##  8     7    11
##  9    10    11
## 10    13    11
## # i 15 more rows
```

```
set.seed(1881450)
# registers the parallel backend to enable parallel processing
doParallel::registerDoParallel()
# tunes the random forest specification with the recipe preprocessor using the generated grid of hyperp
rf_tuned <- tune_grid( object = rf_spec_tune,
                       preprocessor = recipe(playlist_genre ~ . , data = spotify_train_preproc),
                       resamples = spotify_cv,
                       grid = params_grid )
# displays the results of the tuning process
rf_tuned
```

```
## # Tuning results
## # 5-fold cross-validation using stratification
## # A tibble: 5 x 4
##   splits             id    .metrics          .notes
##   <list>             <chr> <list>            <list>
## 1 <split [3402/852]> Fold1 <tibble [50 x 6]> <tibble [0 x 3]>
## 2 <split [3402/852]> Fold2 <tibble [50 x 6]> <tibble [0 x 3]>
## 3 <split [3402/852]> Fold3 <tibble [50 x 6]> <tibble [0 x 3]>
## 4 <split [3402/852]> Fold4 <tibble [50 x 6]> <tibble [0 x 3]>
## 5 <split [3408/846]> Fold5 <tibble [50 x 6]> <tibble [0 x 3]>
```

```
# selects the best model based on the ROC AUC metric
best_auc <- select_best( rf_tuned, "roc_auc" )
# finalizes the random forest model with the best hyperparameters
final_rf <- finalize_model( rf_spec_tune, best_auc )
# displays the final random forest model
final_rf
```

```
## Random Forest Model Specification (classification)
##
## Main Arguments:
##   mtry = 7
##   trees = 100
##   min_n = 21
##
## Engine-Specific Arguments:
##   importance = permutation
##
## Computational engine: ranger
```

```
# fits the final random forest model using the preprocessed training data
spotify_rf <- final_rf %>% fit( playlist_genre ~ . , data = spotify_train_preproc )
# displays the fitted random forest model
spotify_rf
```

```
## parsnip model object
```

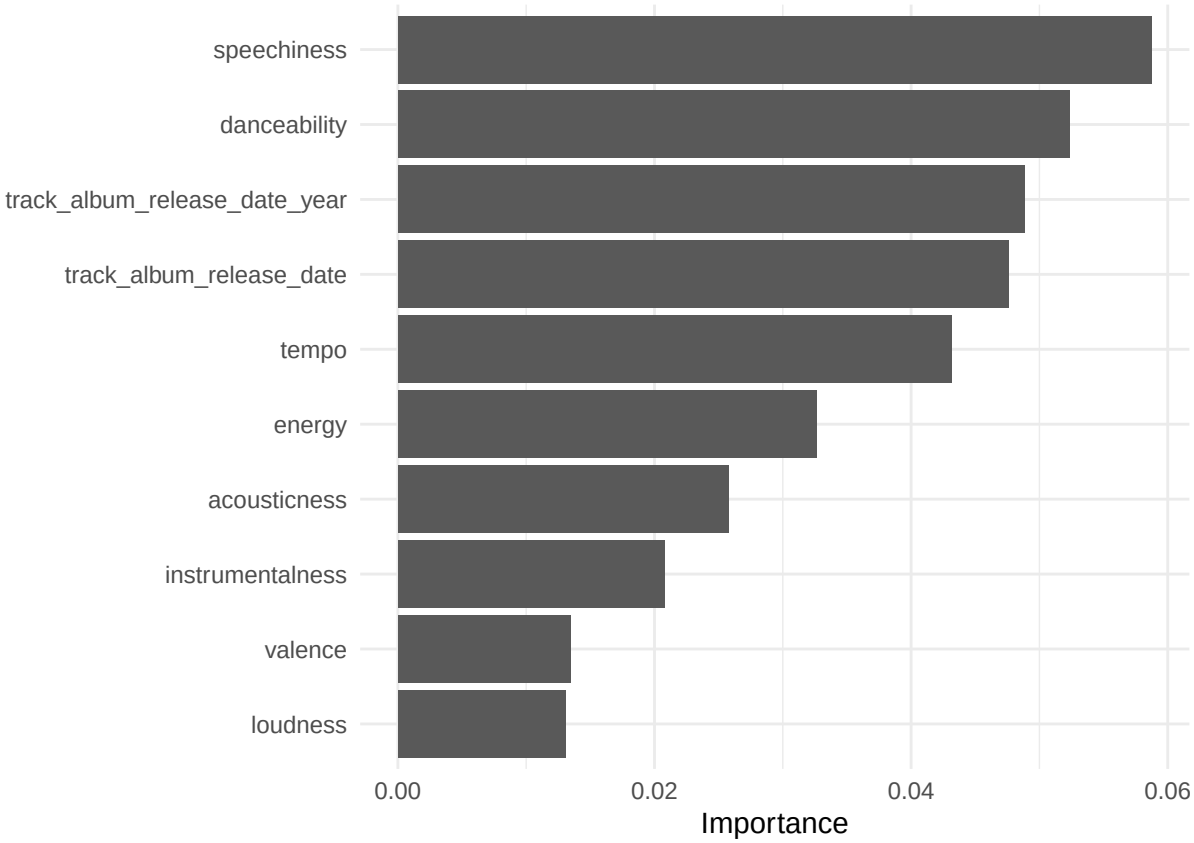
```
##
## Ranger result
##
## Call:
##  ranger::ranger(x = maybe_data_frame(x), y = y, mtry = min_cols(~7L,      x), num.trees = ~100, min.
##
## Type:                             Probability estimation
## Number of trees:                  100
## Sample size:                      4254
## Number of independent variables:  16
## Mtry:                             7
## Target node size:                 21
## Variable importance mode:         permutation
## Splitrule:                        gini
## OOB prediction error (Brier s.):  0.4221944
```

```
# generates a variable importance plot
library(vip)
```

```
##
## Attaching package: 'vip'
```

```
## The following object is masked from 'package:utils':
##
##     vi
```

```
spotify_rf %>%
  vip() +
  theme_minimal()
```




```
# generates predictions for the test set using the fitted random forest model
rf_preds <- predict( spotify_rf, # Get class prediction
                     new_data = spotify_test_preproc,
                     type = "class" ) %>%
  bind_cols( spotify_test_preproc %>% #add on the true value
               dplyr::select( playlist_genre ) )
head(rf_preds,10)
```

```
## # A tibble: 10 x 2
##    .pred_class playlist_genre
##    <fct>       <fct>
##  1 pop         edm
##  2 latin       edm
##  3 latin       edm
##  4 rock        edm
##  5 edm         edm
##  6 pop         edm
##  7 edm         edm
##  8 pop         edm
##  9 edm         edm
## 10 edm         edm
```

Due to the poor accuracy of our model, this prediction is inconsistent.

```
# calculates and displays the performance metrics (accuracy, sensitivity, specificity, etc.) for the ra
rf_preds %>%
  metrics( truth = playlist_genre, estimate = .pred_class )
```

```
## # A tibble: 2 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.542
## 2 kap      multiclass     0.449
```

The Accuracy of the Random Forest Model is *0.542*