Spotify
Data Taming
Final Report

Farmaan

2023-08-05

Executive Summary

As a data scientist at Spotify, our objective is to improve the customer experience and one of the factors to do so is by creating a model that predicts which genre a song belongs to. The main idea behind developing the prediction model is to improve the recommendation and advertising systems of the service. In addition, this will also help in the compilation and updation of playlists. Our decision making and execution will affect around 365 million active users including the 165 million paying subscribers, hence we aim to do the best to maintain Spotify's competitive edge.

With the help of a dataset which contained information about quite a large number of songs and their playlists. The features we talk about are the song's popularity, speechiness, danceability, year of resease, tempo and more. With the help of these features we expect to predict the genre of a song and make suitable playlists with the same. Spotify even aims to better its song recommendations to users using the genre predictions features. The data was cleaned, understood and then reduced to 1000 entries from each genre due to computational complexities. Then we trained the models namely Linear Discriminant Analysis, K-Nearest Neighbors and Random Forest. The performance of these models was compared to get the best one out with the help of metrics like accuracy, area under the curve (AUC), sensitivity and specificity. Conclusively, Random Forest was deemed to be the best model and was even tested on a separate section of data, compared with actual genre and this model performed quite well. The in-depth analysis and reasoning is provided in the report, in addition to recommendations for the stakeholders of Spotify.

Methods

To perform this project we have used R programming language in the R version 4.3.0 (2023-04-21 ucrt) using the packages: tidyverse, skimr, e1071, knitr, dplyr, corrplot, rsample, recipes, parsnip, dials, tune, yardstick, discrim, proc, MASS and workflows.

Data Preprocessing

After loading the data provided from the online source using the read_csv function. From the data we see that there are various features about each song given and the numeric ones in these are popularity, danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo and duration. After removing all the unimportant data, we convert the genre into a factor datatype. Subgenre is also dropped because it is a subclass of genre and if genre is unknown, subgenre will also be unknown. Extracted the years from the release dates and divided them into decades. Converted the duration

from milliseconds to seconds and checked for any discrepancy in the values. There was some error with data collection since some famous songs from superstars had been assigned with 0 popularity, I ended up removing these. There also existed repetition of songs, sometimes even with different genre assigned to the same song. Removed the duplicates keeping only the track with the highest popularity.

Using graphical analysis, explored the relationship between some factors and the song's genre. We found out that:

1. From the figures 5 and 6 we realize that different distributions for each genre mean that genres gain or lose popularity with time. With edm having a recent uptrend. Also, from Figure 7 we deduce that if the song was released before 1990, it was most likely from rock genre. In 1990s, there is an approximately equal probability of getting a rap or r&b. In 2000s, it will tend towards r&b, rock, rap and latin. Finally, in 2010s edm, pop and rap are most probable.
2. Speechiness also varies across the genres meaning that some genres include more lyrics compared to others, from figures 8 and 9 we deduce that the median speechiness of the rap genre tends to be higher than the rest while as median speechiness for rock seems to be lowest.
3. Danceability also varies across the genres meaning the it's easier to dance on certain genres, from figures 10 and 11 we deduce that the median daceability of the rap and latin genre tends to be higher than the rest while as median daceability for rock seems to be lowest.
4. Tempo also varies across the genres meaning each genre might have a specific tempo range, from figures 12 and 13 we deduce that the median tempo close to 125 is most likely that of an edm.

These explored relations are important to understand the effect on the predicted genre. Further, we reduce the data for computational purposes and using stratified sampling, choose 1000 songs from each genre to proceed, seen in Table 6.

Exploratory Data Analysis

Further, we explore the relationship between the genre variable and the rest of the predictors mainly using visualizations. The explored track features specifically include popularity, speechiness, and release year. Also, we check the correlation between the fellow predictor variables. All this analysis is included in the Appendix, figures 14 to 19.

Further Data Preprocessing

Since we are done with the exploratory data analysis we drop the variables like year (we have the decade variable), song name and artist name since these are no longer required. With the finalized predictors, we split the dataset into training and testing datasets. Preparing a recipe for the preprocessing of the training and testing we follow the given steps:

1. Decade is a categorical variable hence we create a dummy variable.
2. The variables danceability and energy are left skewed (negatively skewed), we adjust them first.
3. Then we apply the best applicable transformations to the right skewed variables speechiness, danceability, energy, liveness, tempo, duration, instrumentalness, acousticness.
4. Furthermore, normalize all the predictors bringing them to same scale.
5. Remove one of the highly correlated predictors as they don't add much information to the model.
6. Convert the predictors into principal components of the of the data.

Now we have the resultant genre variable and the predictors in the principal component form. From these principal components we find the most important ones that explain at least 90% of variation in the data. From the screeplot in Figure 20, we see that 13 Principal Components do the job, hence we update the recipe to just use the best 13 Principal components. Applying this recipe on the training dataset using juice function and on the testing dataset using the bake function.

Model Preperation

As the expert statistician's suggestions for the genre prediction, going ahead into the model building stage. We use three different models namely Linear Discriminant Analysis (LDA), K-Nearest Neighbours (KNN) model with a range of 1 to 100 and 20 levels, and Random Forest (RF) with 100 trees and 5 levels. Using the bootstrapping method, we tune the KNN and RF models to extract the best hyperparameters comparing and getting the hyperparameters that produce the best AUC. The tuning performance metrics of KNN model is shown in figure 21 while that of RF model are shown in figure 22.

Model Comparison

Using the K-fold Cross Validation, we fit the three models prepared and compare their collected performance metrics AUC, even the Accuracy can be compared. Then getting the prediction of the training data itself gives us the confusion matrix, sensitivity and specificity of each of these models. From these metrics we figure out the best working model for the current application and the dataset, This best model's performance is then tested on the testing dataset and cross verified the performance metrics like Accuracy, AUC, Sensitivity and Specificity from the confusion matrix.

Results

The provided dataset contains information about the songs on Spotify and the aim is to predict the genre based on various features of a song. The analysis showed that the genre is affected by the variables like popularity, speechiness, danceability, tempo and the year or decade the song is from.

The exploratory data analysis of the with the help of figures (5 to 19) helped identify the relation of the variables. We realised that rock was the most common genre before 1990. Also, currently pop music is the most famous. Rap music is far ahead in terms of speechines and is close to latin in terms of danceability.

To predict the genre we have used three different models namely Linear Discriminant Analysis (LDA), K-Nearest Neighbors (KNN) and Random Forest (RF). After following the suggestions of an expert statistician, we created three of these models for achieving our goal of predicting genre from song information. After comparing these models on the basis of metrics like Accuracy, Area Under the Curve (AUC), sensitivity and specificity, we found out that the predictive ability of the Random Forest model is the best. Detailed information given in the Appendix.

Discussion

During the analysis, it is clear that there are different levels of popularity for each of the genre and the frequency of genre also depends on the decade. In addition, speechines varies across genres, tempo and danceability are also effected by the genre. These all variables with a few other will affect and help us in improving the song recommendation system of Spotify.

To predict the song's genre we trained three different classification models out of which the Random Forest was deemed to be the best. From the Table 13 we can see that the LDA model has an accuracy of 48.62% and the AUC of 0.8015. The 20 levelled KNN model has an accuracy of 49.75% and the AUC of 0.81155. Finally,

for the RF model, the accuracy is 52.09% and the AUC is 0.8178. The RF model outperforms other not only based on accuracy and auc but also sensitivity and specificity. This is the best indication the we should go ahead and deploy the RF model out of these three to predict the genre of songs for recommendation.

Finally, testing the RF model on the test dataset shows a pretty consistent behaviour to the one we saw when predicting the training dataset. We get an accuracy of 51.73% and the AUC of 0.8193.

Conclusion

In conclusion, the genre of songs vary with a number of features, each genre having its special identity and some overlapping identities. We see that the popularity of songs differs based on genre and also the genre song frequency depends on the decade. Such properties will be used by the model to predict the genre and provide users with better service.

Further, the analysis revealed that the Random Forest model outperformed the Linear Discriminant Analysis model and the K-Nearest Neighbors model. This was potrayed by the use of metrics such as accuracy, AUC, sensitivity and specificity. The reasoning have also been explained for the decisions made.

In this report I have tried to leave out any personal biases that might affect the model performance. We can try out other classification techniques upcoming with new research on this data to achieve the same goal. Keeping up with research in this field is very important as there might be better models out there for this domain specific data.

Appendix

Data Preprocessing

```
# importing required libraries
library(tidyverse)
library(skimr)
library(e1071)
library(knitr)
library(dplyr)
library(corrplot)
library(rsample)
library(recipes)
library(parsnip)
library(dials)
library(tune)
library(yardstick)
library(discrim)
library(pROC)
library(MASS)
library(workflows)
```

```
#Reading the data
spotify_songs <-
  readr::read_csv('https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2020/2020-0

#Understanding the Data
spotify_songs %>% head()
```

```
## # 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>
```

```
spotify_songs %>% skim_without_charts(.data_name = "Spotify Songs")
```

Table 1: Data summary

Name	Spotify Songs
Number of rows	32833
Number of columns	23
Column type frequency:	
character	10
numeric	13
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
track_id	0	1	22	22	0	28356	0
track_name	5	1	1	144	0	23449	0
track_artist	5	1	2	69	0	10692	0
track_album_id	0	1	22	22	0	22545	0
track_album_name	5	1	1	151	0	19743	0
track_album_release_date	0	1	4	10	0	4530	0
playlist_name	0	1	6	120	0	449	0
playlist_id	0	1	22	22	0	471	0
playlist_genre	0	1	3	5	0	6	0
playlist_subgenre	0	1	4	25	0	24	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
track_popularity	0	1	42.48	24.98	0.00	24.00	45.00	62.00	100.00
danceability	0	1	0.65	0.15	0.00	0.56	0.67	0.76	0.98
energy	0	1	0.70	0.18	0.00	0.58	0.72	0.84	1.00

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
key	0	1	5.37	3.61	0.00	2.00	6.00	9.00	11.00
loudness	0	1	-6.72	2.99	-46.45	-8.17	-6.17	-4.64	1.27
mode	0	1	0.57	0.50	0.00	0.00	1.00	1.00	1.00
speechiness	0	1	0.11	0.10	0.00	0.04	0.06	0.13	0.92
acousticness	0	1	0.18	0.22	0.00	0.02	0.08	0.26	0.99
instrumentalness	0	1	0.08	0.22	0.00	0.00	0.00	0.00	0.99
liveness	0	1	0.19	0.15	0.00	0.09	0.13	0.25	1.00
valence	0	1	0.51	0.23	0.00	0.33	0.51	0.69	0.99
tempo	0	1	120.88	26.90	0.00	99.96	121.98	133.92	239.44
duration_ms	0	1	225799.81	59834.01	4000.00	187819.00	216000.00	253585.00	517810.00

```
#Dropping the few null values seen when data was skimmed
spotify_songs <- spotify_songs %>% drop_na()

#Dropping columns which act like index but don't contribute towards finding genre
spotify_songs <- spotify_songs %>%
  dplyr::select(-c(track_id,track_album_id,track_album_name,
            playlist_name,playlist_id))

#genre is categorical hence we convert to factors
spotify_songs$playlist_genre <- factor(spotify_songs$playlist_genre)

#We don't know genre, so the subgenre will also be unknown.
#Not needed for prediction
spotify_songs <- spotify_songs %>%
  dplyr::select(-playlist_subgenre)
```

```
#Getting meaning from Release Date Column
#Extract just the year
spotify_songs$track_album_release_date <-
  as.Date(spotify_songs$track_album_release_date, format = "%Y")
spotify_songs$year <- year(spotify_songs$track_album_release_date)

#Understanding the yearly distribution of songs
summary(spotify_songs$year)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1957    2008    2016    2011    2019    2020
```

```
spotify_songs %>% ggplot(aes(x=year))+geom_histogram() +
  ggtitle("Distribution of release year of songs") +
  labs(caption = "Figure 1. The year wise distribution of songs also shows that most of the songs are fr
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

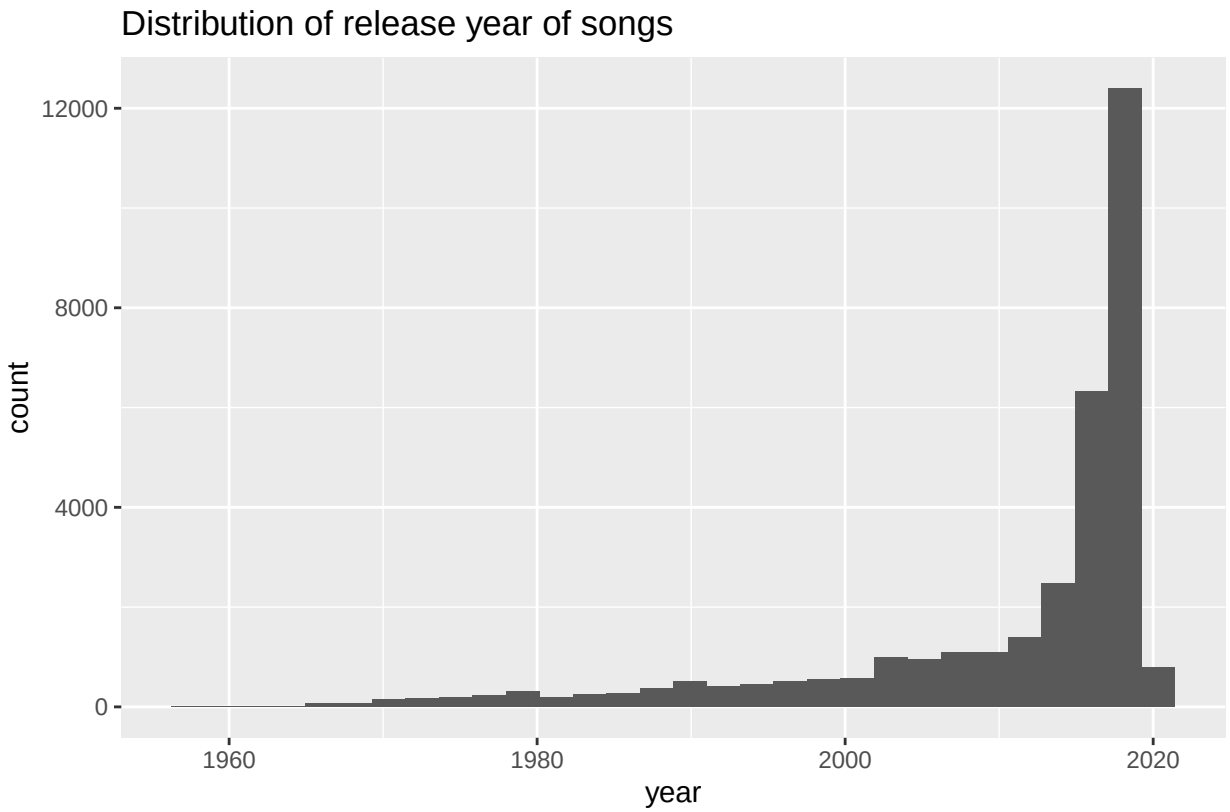


Figure 1. The year wise distribution of songs also shows that most of the songs are from recent years.

```
spotify_songs %>% ggplot(aes(x=year))+geom_boxplot() +
  ggtitle("Distribution of release year of songs") +
  labs(caption = "Figure 2. Another display of year wise distribution of songs. Old songs are rarely pr
       The oldest song belongs to year 1957")
```

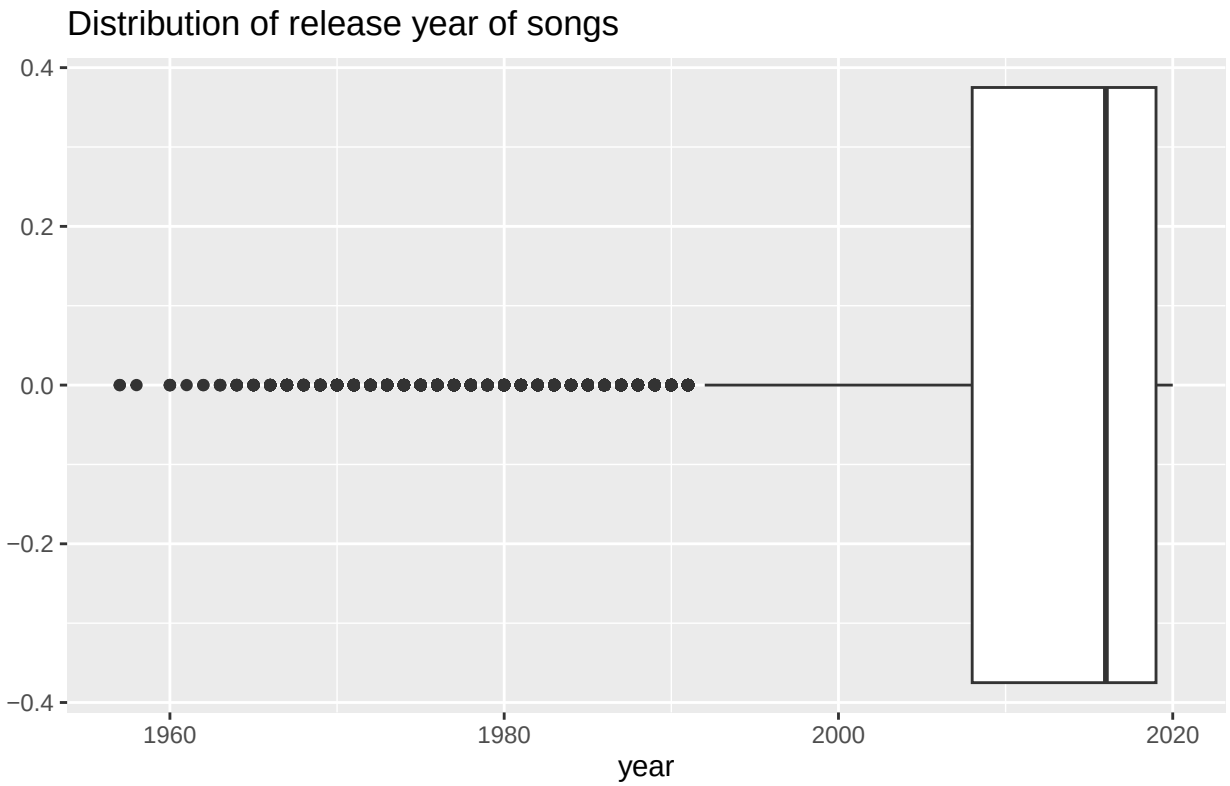


Figure 2. Another display of year wise distribution of songs. Old songs are rarely present, hence are outliers. The oldest song belongs to year 1957

```
#Assign what decade the songs belongs to
#Music genre popularity doesn't go away in a year thus decade will be a better predictor.
spotify_songs <- spotify_songs %>%
  mutate(decade = case_when(year <= 1979 ~ 'Pre-1980s',
                            year <= 1989 ~  '1980s',
                            year <= 1999 ~ '1990s',
                            year <= 2009 ~ '2000s',
                            year <= 2020 ~ '2010s',
                            TRUE ~ 'NA'))

#Changing decade to an ordinal categorical (factor) datatype
spotify_songs$decade <-
  factor(spotify_songs$decade,
         levels = c('Pre-1980s', '1980s', '1990s', '2000s', '2010s'),
         ordered = TRUE)

#Count of songs in each decade
spotify_songs %>% count(decade) %>%
  knitr::kable(caption = "Least amount of songs come from Pre-1980s and a lot of them from 2010s")
```

Table 4: Least amount of songs come from Pre-1980s and a lot of them from 2010s

decade	n
Pre-1980s	1141
1980s	1306
1990s	2310
2000s	4077
2010s	23994

```
spotify_songs %>% ggplot(aes(x=decade))+geom_bar() +
  ggtitle("Count of songs from each decade")+
  labs(caption = "Figure 3. Spotify dataset has more number songs from the recent decades. Huge rise in
```

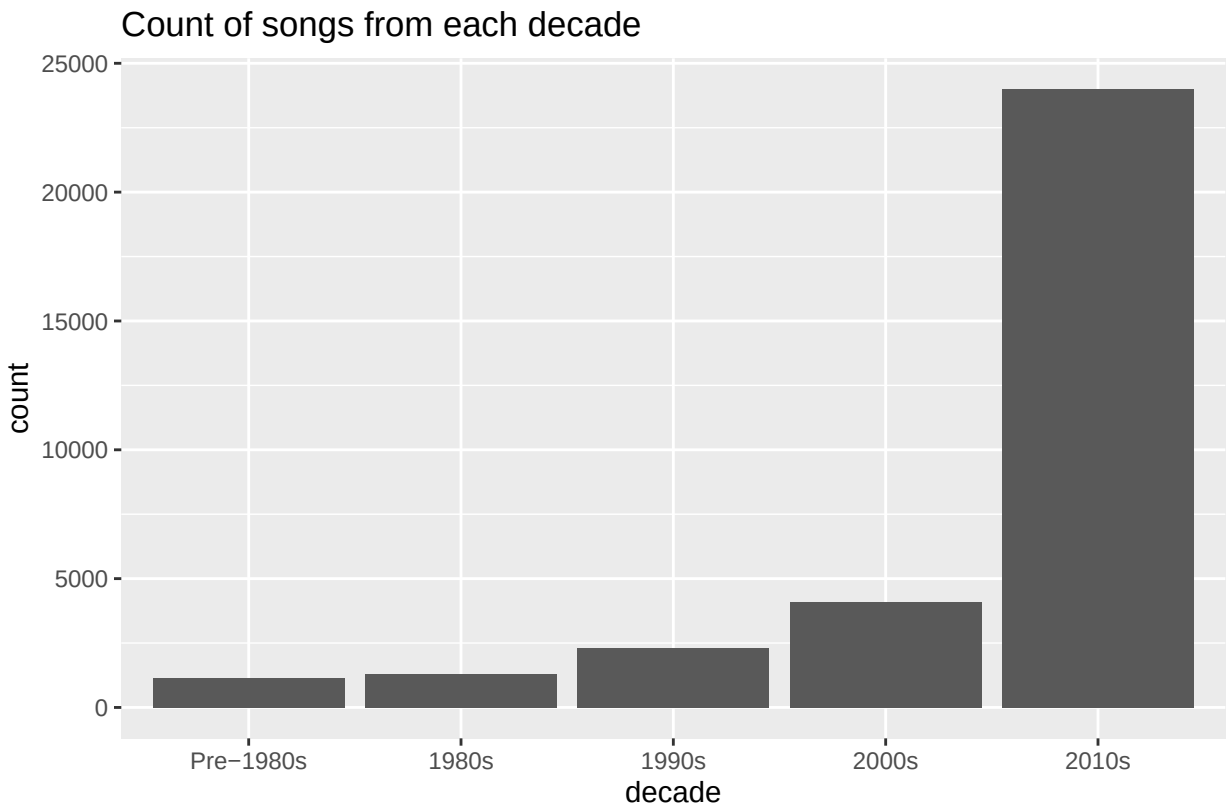


Figure 3. Spotify dataset has more number songs from the recent decades. Huge rise in songs in the 2010s

```
#No need for the date variable
#We need the year variable for some more analysis
spotify_songs <- spotify_songs %>% dplyr::select(-c(track_album_release_date))
```

```
#Duration data cleaning
#Millisecond to Seconds
spotify_songs <- spotify_songs %>%
  mutate(duration_sec = duration_ms*0.001)

#Dropping duration_ms
```

```
spotify_songs <- spotify_songs %>%
  dplyr::select(-c(duration_ms))

summary(spotify_songs$duration_sec)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     4.0   187.8   216.0   225.8   253.6   517.8
```

```
spotify_songs %>% ggplot(aes(x=duration_sec))+geom_boxplot()+
  ggtitle("Range of durations of songs")+
  labs(caption = "Figure 4. The length of the playtime of all the songs.
       The shortest song duration is 4 seconds which is an extreme outlier.")
```

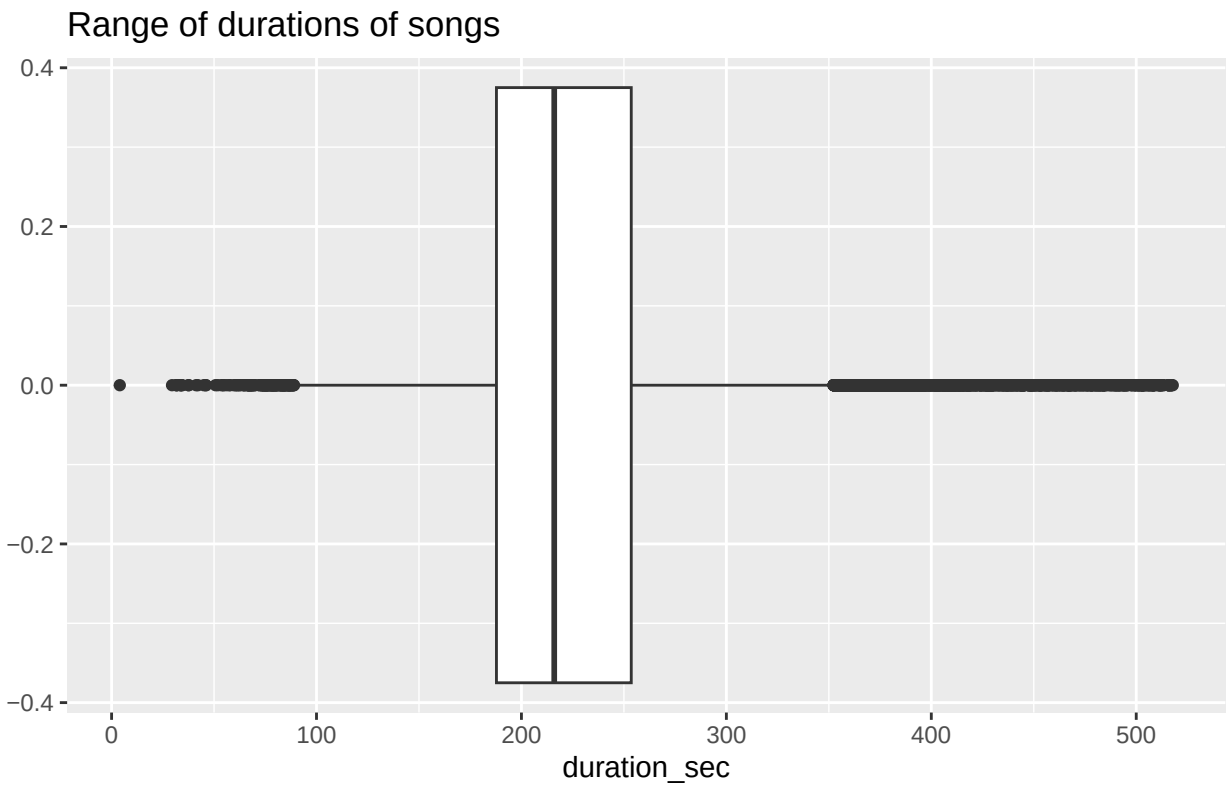


Figure 4. The length of the playtime of all the songs.
The shortest song duration is 4 seconds which is an extreme outlier.

```
#The song being just 4 second long is unrealistic and extreme outlier
# but the max duration of around 518 seconds (8.6 minutes) is realistic.
#Remove only the song that is 4 sec long
spotify_songs <- spotify_songs %>% filter(duration_sec>4)

#Now we have a realistic range 29.49 sec to 517.81 sec
summary(spotify_songs$duration_sec)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   29.49  187.81  216.00  225.80  253.58  517.81
```

We remove the tracks with 0 popularity, since this is surely an error made while collecting data. There are songs from superstars like Eminem, DJSnake, 50 Cent on this list. These songs can't have 0 popularity.

```
spotify_songs %>%
  filter(track_popularity==0)
```

```
## # A tibble: 2,697 x 18
##    track_name   track_artist track_popularity playlist_genre danceability energy
##    <chr>        <chr>                  <dbl> <fct>                 <dbl>  <dbl>
##  1 Siren        SUNMI                      0 pop                   0.605  0.894
##  2 Lollipop (C~ Aqua                       0 pop                   0.716  0.981
##  3 Around The ~ Aqua                       0 pop                   0.705  0.975
##  4 I'm Yours    Influencers~               0 pop                   0.559  0.844
##  5 Talk About ~ Bancali                    0 pop                   0.498  0.723
##  6 You Don't K~ Armand Van ~               0 pop                   0.684  0.85
##  7 TURN         TaeyoungBoy                0 pop                   0.704  0.482
##  8 Heaven       The Neighbo~               0 pop                   0.555  0.712
##  9 Kings of th~ Lonely The ~               0 pop                   0.294  0.929
## 10 Gossip       You Me At S~               0 pop                   0.43   0.944
## # i 2,687 more rows
## # i 12 more variables: key <dbl>, loudness <dbl>, mode <dbl>,
## #   speechiness <dbl>, acousticness <dbl>, instrumentalness <dbl>,
## #   liveness <dbl>, valence <dbl>, tempo <dbl>, year <dbl>, decade <ord>,
## #   duration_sec <dbl>
```

```
#Removing songs where popularity = 0
spotify_songs <- spotify_songs %>%
  filter(track_popularity>0)
```

There are a number of songs which get repeated, some even with different genre for the same song. Thus we remove the duplicates and only keep the entry which has the highest popularity. We order these duplicates in descending order of their popularity and extract the most popular.

```
#Dropping Duplicates
spotify_songs <- spotify_songs %>%
  arrange(desc(track_popularity)) %>%
  group_by(track_name, track_artist) %>%
  dplyr::slice(1) %>%
  ungroup()

#No Duplicates left
```

```
#Changing column names to easier form
names(spotify_songs) <- c("name", "artist", "popularity", "genre", "danceability", "energy", "key",
                          "loudness", "mode", "speechiness", "acousticness", "instrumantalness",
                          "liveness", "valence", "tempo", "year", "decade", "duration")

#Songs left from each genre in the cleaned Dataset
spotify_songs %>%
  count(genre) %>%
  knitr::kable(caption = "Total songs in the dataset from each genre.")
```

Table 5: Total songs in the dataset from each genre.

genre	n
edm	4034
latin	3571
pop	4416
r&b	3736
rap	4750
rock	3458

There are a number of questions to be answered before Exploratory Analysis. To find out how the following factors can predict a song's genre:

1. The year the song was released.

```
#year song was released
spotify_songs %>% ggplot(aes(y=year, x = genre, fill=genre)) + geom_boxplot() +
  ggtitle("Genre-wise release year of songs") +
  labs(caption = "Figure 5. We see that rock music has been there from the initial years, edm is a very
       r&b might have started early but picked up around 1980, most rap songs started in late 1990s,
       Latin and pop have a similar distribution.")
```

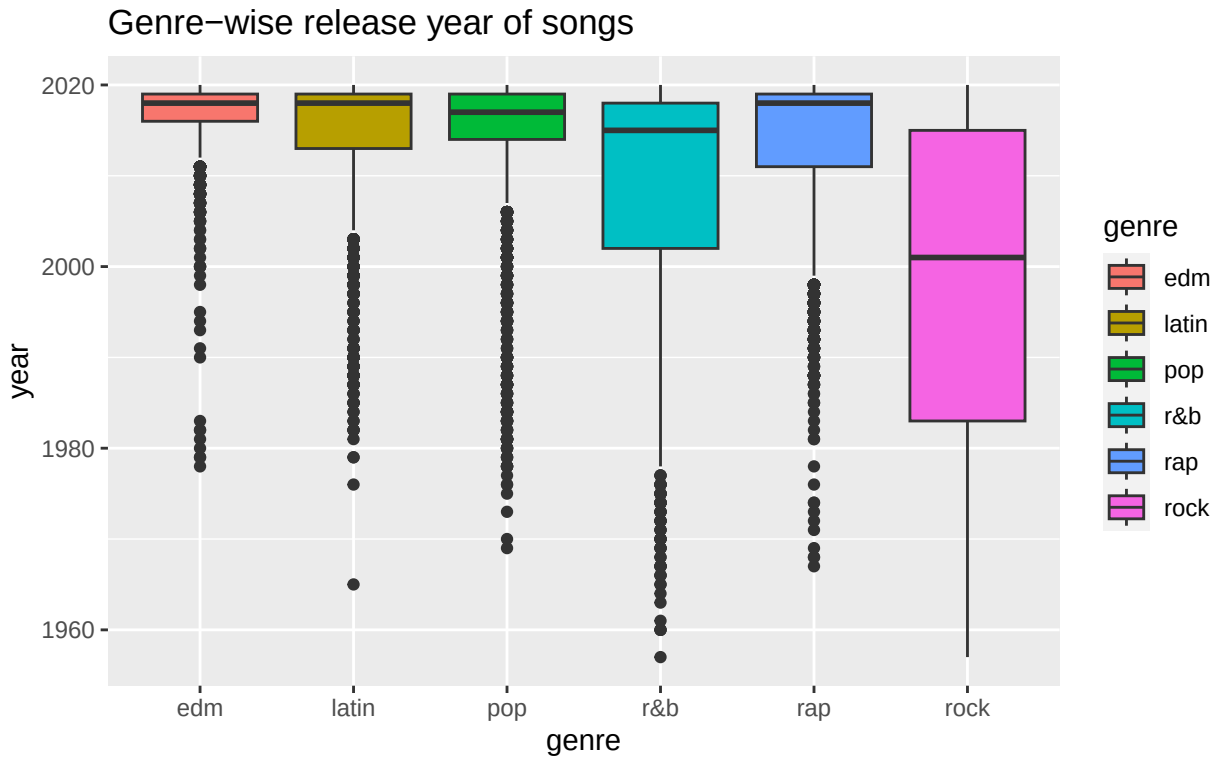


Figure 5. We see that rock music has been there from the initial years, edm is a very new genre, r&b might have started early but picked up around 1980, most rap songs started in late 1990s, Latin and pop have a similar distribution.

```
spotify_songs %>% ggplot(aes(x=year)) + geom_histogram() + facet_wrap(~genre) +
    theme( axis.text.x = element_text( angle = 90 ) )+
  ggtitle("Distribution of reelease years of songs")+
  labs(caption = "Figure 6. From distributions we see that rock followed by r&b have been there for lon
       There have been a lot of new rap songs but it started around 1990, pop and latin are similar in
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

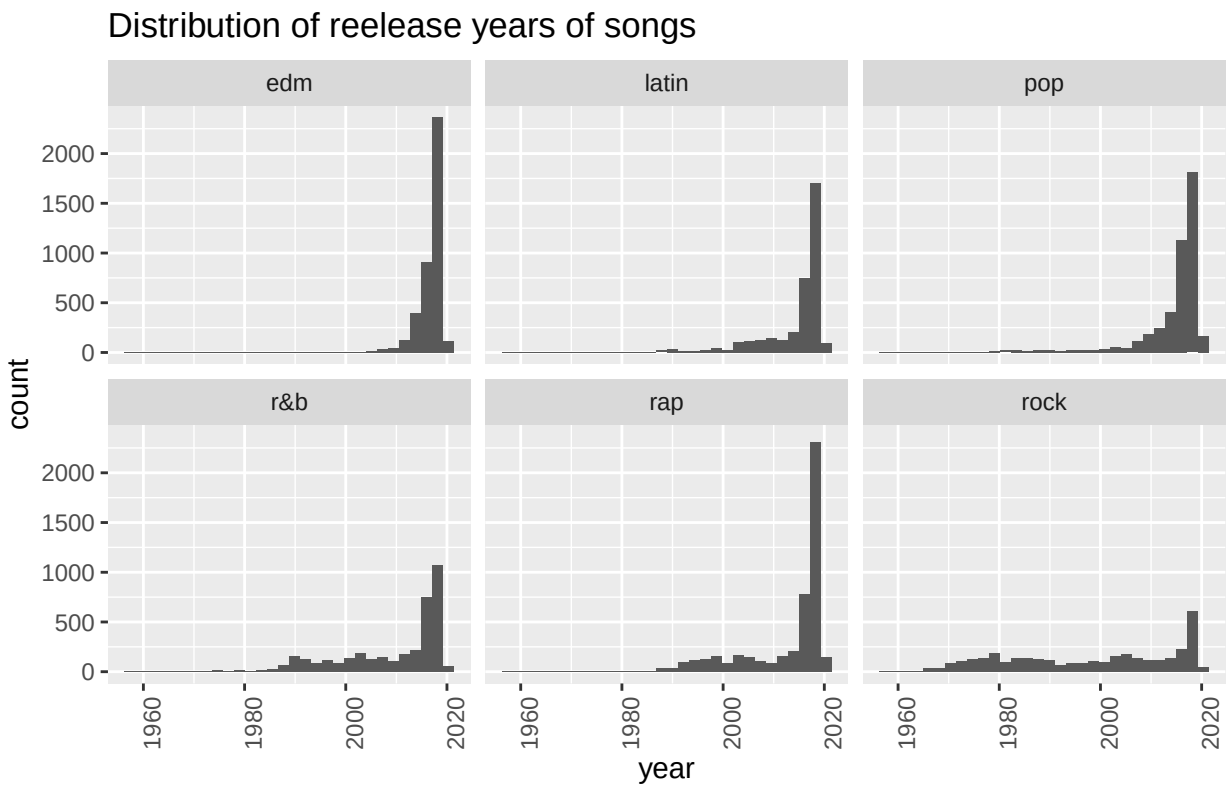


Figure 6. From distributions we see that rock followed by r&b have been there for long time, edm is the latest genre. There have been a lot of new rap songs but it started around 1990, pop and latin are similar in distribution.

```
#Based on decade
#Creating a proportional bar chart
spotify_songs %>% ggplot(aes(x=decade, fill = genre)) +
  geom_bar(position = "fill", col = "black")+
  ggtitle("Decade-wise ratio of song genre")+
  labs(caption = "Figure 7. We can see that in the Pre-1980s and in 1980s, the song is most likely to b
       followed by r&b. In the 1990s, rap and r&b are equiprobable followed by rock.
       In 2000s, r&b, rap and rock are equiprobable with latin and pop not far behind.
       In 2010s, edm, pop and rap are the most probable, rock the least.")
```

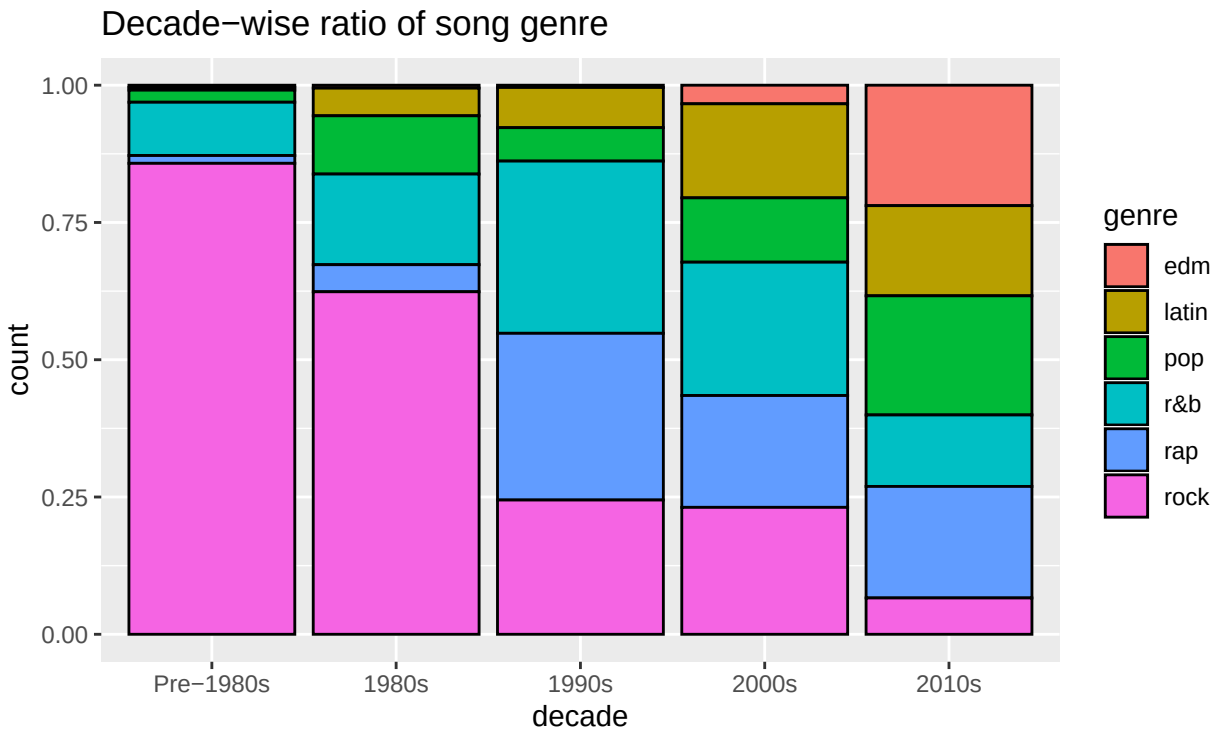


Figure 7. We can see that in the Pre−1980s and in 1980s, the song is most likely to be of rock genre, followed by r&b. In the 1990s, rap and r&b are equiprobable followed by rock. In 2000s, r&b, rap and rock are equiprobable with latin and pop not far behind. In 2010s, edm, pop and rap are the most probable, rock the least.

From the above 3 figures (Figure 5,6,7), we can deduce that if the song was released before 1990, it was most likely from rock genre. In 1990s, it might be rap or r&b. In 2000s, It will tend towards r&b, rock, rap and latin. Finally, in 2010s edm, pop and rap are most probable.

2. How "speechy" the song is.

```
#Speechy
spotify_songs %>% ggplot(aes(y=speechiness, x = genre, fill=genre)) + geom_boxplot()+
  ggtitle("Genre-wise speechiness of songs")+
  labs(caption = "Figure 8. We can see that rap is the most speechy followed by r&b and latin.
       The least speechy genre is rock, with pop and edm having a bit more speechiness")
```

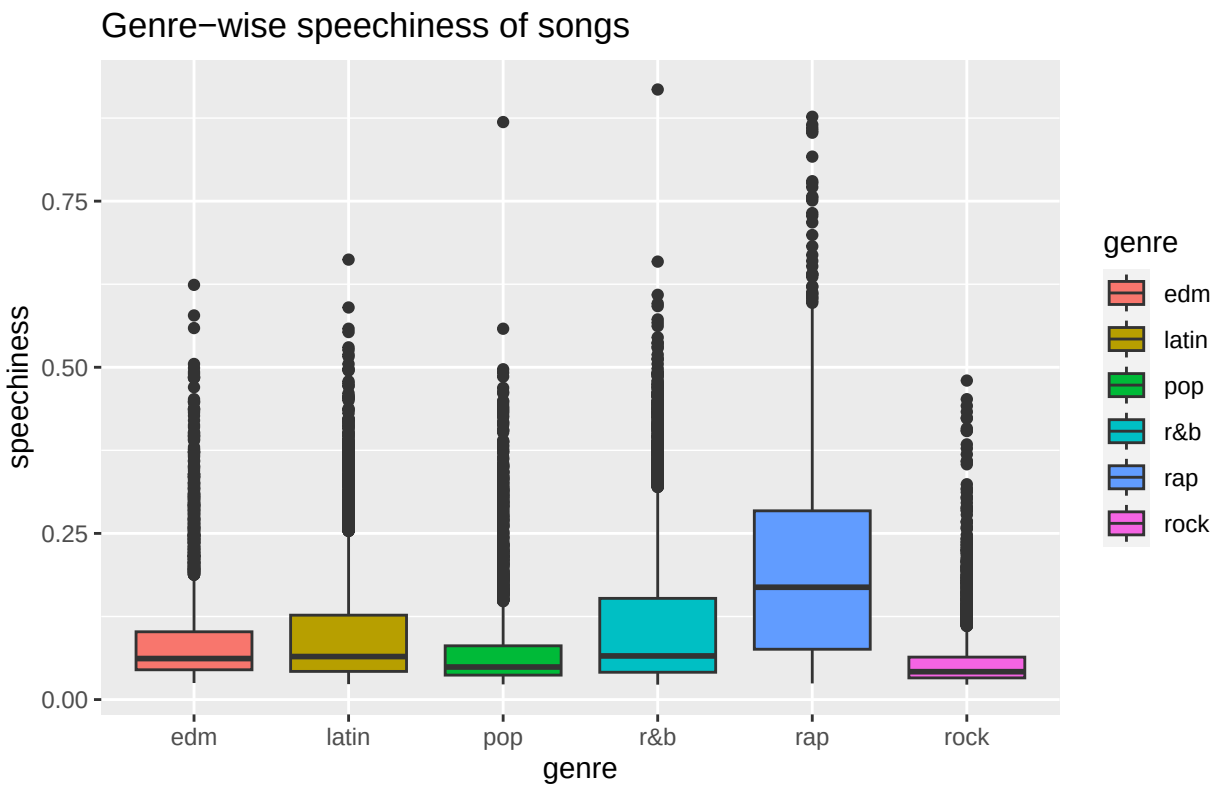


Figure 8. We can see that rap is the most speechy followed by r&b and latin.
The least speechy genre is rock, with pop and edm having a bit more speechiness

```
spotify_songs %>% ggplot(aes(x=speechiness)) + geom_histogram() + facet_wrap(~genre) +
  ggtitle("Distribution of speechiness of songs")+
  labs(caption = "Figure 9. From distributions it seems that rap is the most speechy why rock and pop a
       Latin and r&b have lesser speechiness than rap.")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

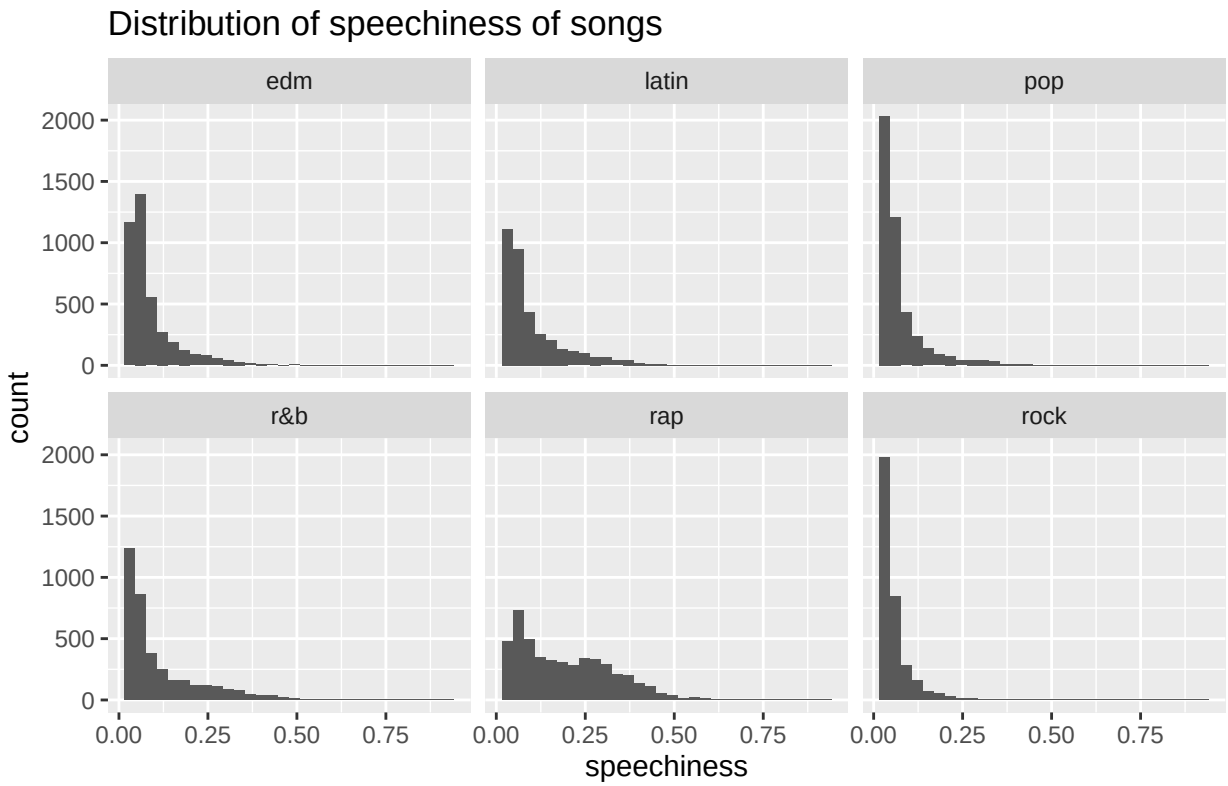


Figure 9. From distributions it seems that rap is the most speechy why rock and pop are the least. Latin and r&b have lesser speechiness than rap.

So more the speechiness, the genre tends to be rap, and lesser the speechines, the genre tends to be rock.

3. How danceable the song is.

```
#Danceable
spotify_songs %>% ggplot(aes(y=danceability, x = genre, fill= genre)) +
  geom_boxplot()+
  ggtitle("Genre-wise danceability of songs")+
  labs(caption = "Figure 10. We can see that rock is the least danceable genre while rap and latin being
       most danceable. Pop, edm and r&b have fairly similar danceability.")
```

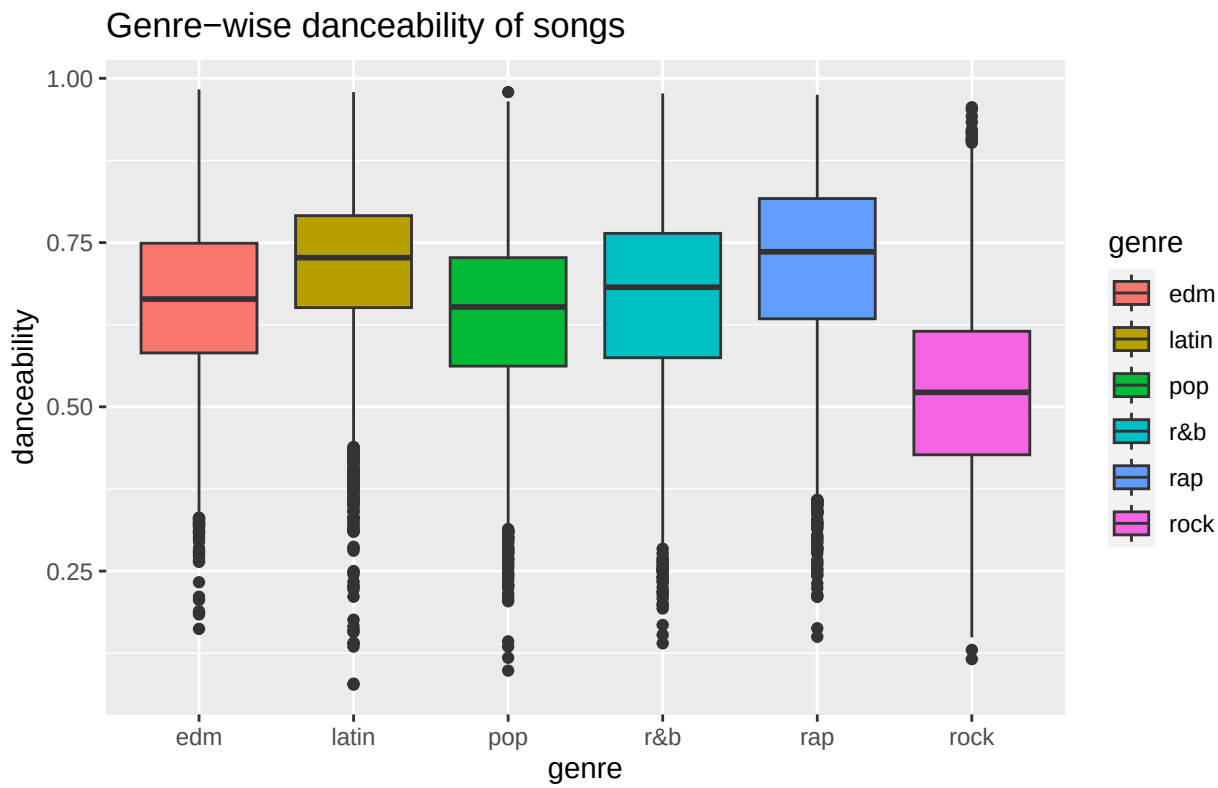


Figure 10. We can see that rock is the least danceable genre while rap and latin being the most danceable. Pop, edm and r&b have fairly similar danceability.

```
spotify_songs %>% ggplot(aes(x=danceability)) + geom_histogram() + facet_wrap(~genre)+
    theme( axis.text.x = element_text( angle = 90 ) )+
  ggtitle("Distribution of danceability of songs")+
  labs(caption = "Figure 11. From distributions it seems that rap and latin are left skewed hence most
  Pop, edm and r&b are symmetrical hence moderately danceable while rock is right skewed hence least da
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

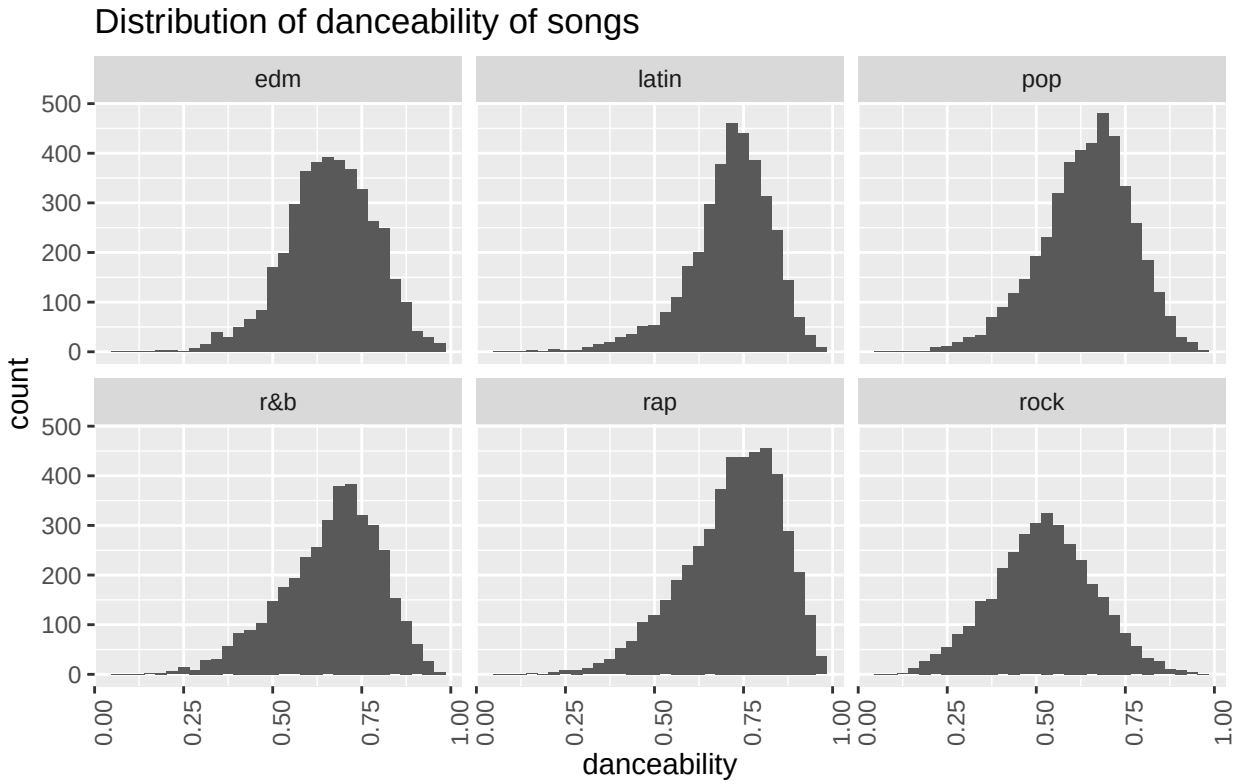


Figure 11. From distributions it seems that rap and latin are left skewed hence most danceable. Pop, edm and r&b are symmetrical hence moderately danceable while rock is right skewed hence least danceable.

With low danceability we can assume that the genre might be rock while with high danceability it can be rap or latin.

4. The tempo of the song.

```
#Tempo
spotify_songs %>% ggplot(aes(y=tempo, x = genre, fill=genre)) + geom_boxplot()+
  ggtitle("Genre-wise tempo of songs")+
  labs(caption = "Figure 12. We can see that edm tempo is very close to 125 for most of the songs.
       Rap, rock and pop also have a tempo close to 125 but spread around.
       Latin and r&b with lower tempo and spread around")
```

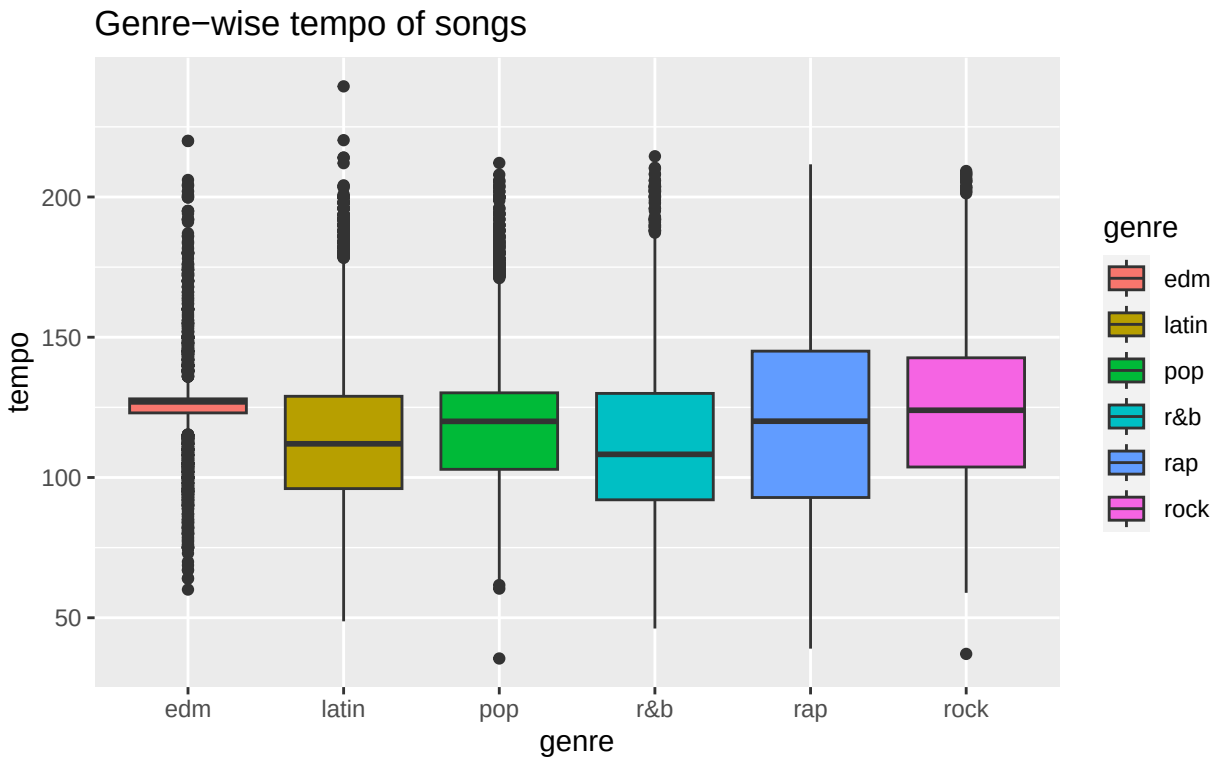


Figure 12. We can see that edm tempo is very close to 125 for most of the songs. Rap, rock and pop also have a tempo close to 125 but spread around. Latin and r&b with lower tempo and spread around

```
spotify_songs %>% ggplot(aes(x=tempo)) + geom_histogram() + facet_wrap(~genre)+
  ggtitle("Distribution of tempo of songs")+
  labs(caption = "Figure 13. From distributions we see bulk of edm's have tempo around 125,
       latin and r&b being right skewed and lower tempo, while others being fairly symmetrical.")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

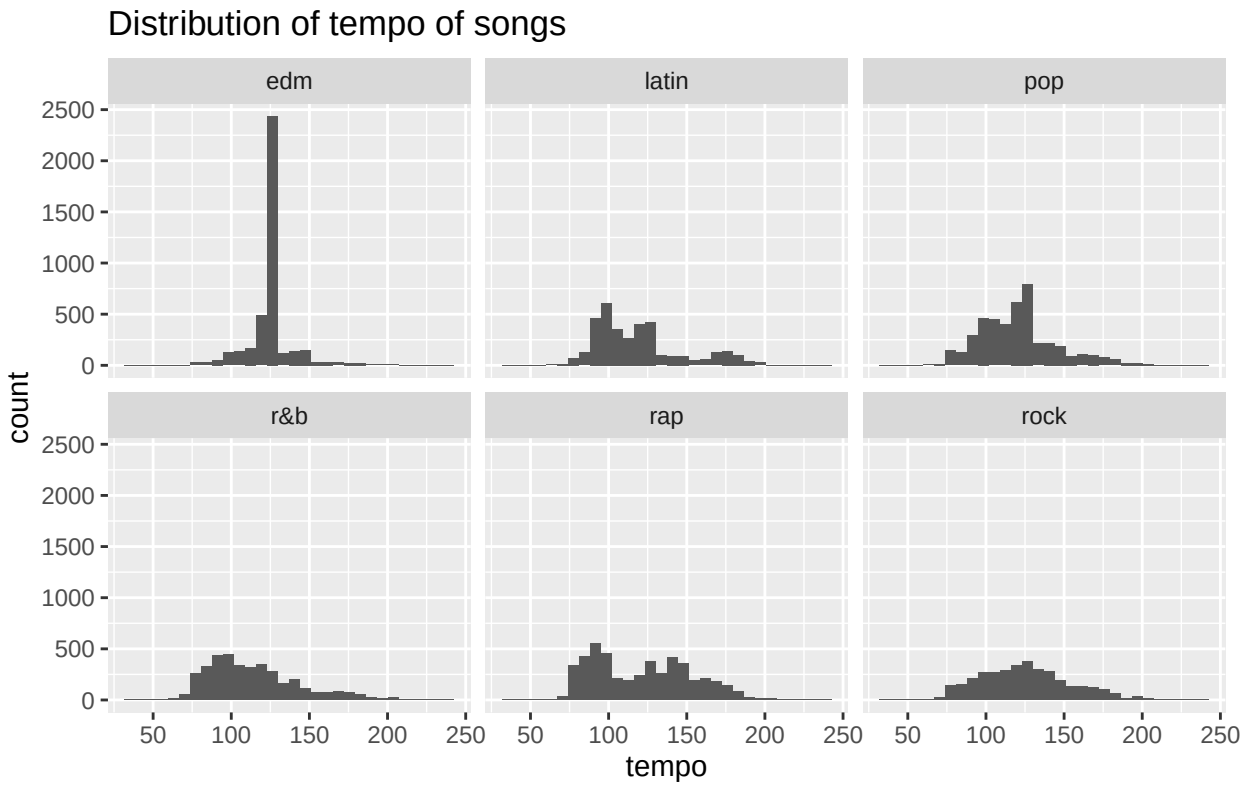


Figure 13. From distributions we see bulk of edm's have tempo around 125, latin and r&b being right skewed and lower tempo, while others being fairly symmetrical.

Tempo does not give much of a trend to predict the genre but if the value is close to 125, it is most likely an edm song.

For computation complexity, we are reducing the dataset by stratifying the data based on the genre of the songs. We take 1000 songs from each genre

```
# Selecting 1000 random samples from each genre
set.seed(1875837)
spotify_songs <- spotify_songs %>%
  group_by(genre) %>%
  sample_n(size=1000) %>% ungroup()

#Check that we get only 1000 from each genre
summary_data <- spotify_songs %>%
  group_by(genre) %>%
  summarise(Count = n())
summary_data %>% kable(caption = "Reduced and updated (Stratified) dataset with 1000 songs from each ge
```

Table 6: Reduced and updated (Stratified) dataset with 1000 songs from each genre.

genre	Count
edm	1000
latin	1000
pop	1000
r&b	1000

genre	Count
rap	1000
rock	1000

Exploratory Data Analysis

From the following figures, we can draw various insights regarding our dataset.

1. Does the popularity of songs differ between genres?

```
#Popularity vs genre
spotify_songs %>%
  ggplot(aes(y = popularity, x = genre, fill = genre)) +
  geom_boxplot() +
  ggtitle("Popularity vs genre") +
  labs(caption = "Figure 14. We can clearly see that edm and r&b is the least popular while the other g
       fairly close.")
```

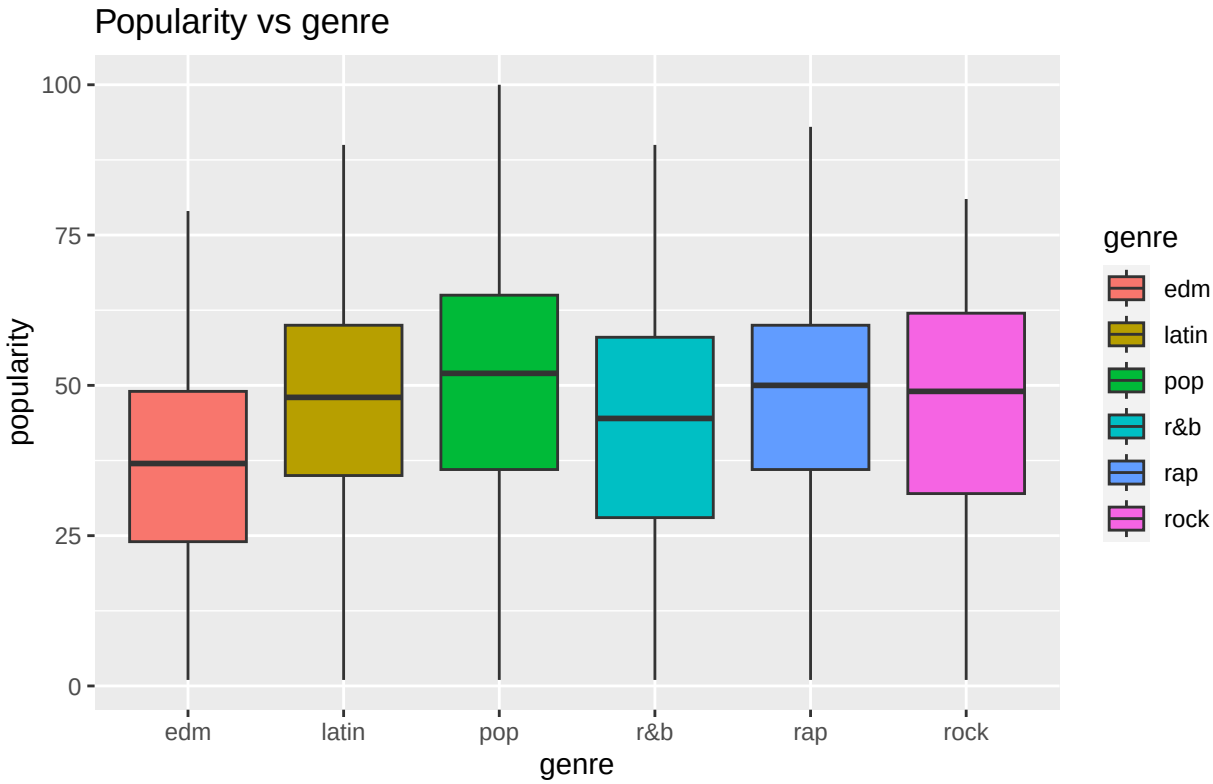


Figure 14. We can clearly see that edm and r&b is the least popular while the other genres are fairly close.

```
mean.popularity <- spotify_songs %>%
  group_by(genre) %>%
  summarize(mean.pop = mean(popularity))

spotify_songs %>% ggplot() +
```

```
  geom_histogram(aes(popularity, col=popularity),
                 color = 'blue', alpha = .75) +
  geom_vline(data = mean.popularity, col = 'red',
             mapping = aes(xintercept = mean.pop))+
  geom_label(data = mean.popularity, col = 'red',
             aes(x = mean.pop, y = -5,
                 label = paste('Mean:', round(mean.pop,1)))) +
  facet_wrap(~genre) +
  ggtitle('Genre based distribution of popularity') +
  labs(caption = "Figure 15. EDM is the least popular while Pop is the most popular followed by rap.
       Rock and latin have similar popularity.")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

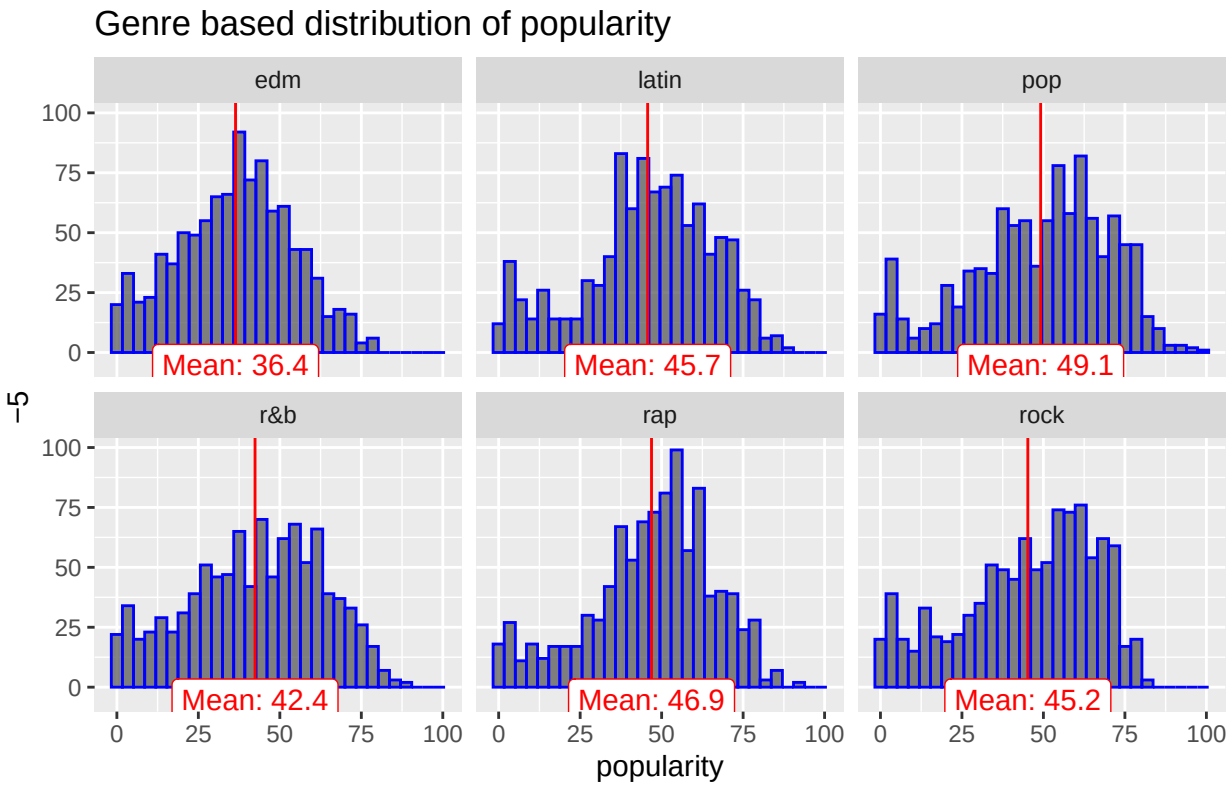


Figure 15. EDM is the least popular while Pop is the most popular followed by rap. Rock and latin have similar popularity.

Yes popularity of songs differs between genre. Pop tends to be the most popular, edm the least, r&b the 2nd least while the other three (rap, rock and latin) pretty close

2. Is there a difference in speechiness for each genre?

```
#Speechiness vs genre
skewness(spotify_songs$speechiness)
```

```
## [1] 2.047556
```

```r
median.speechiness <- spotify_songs %>%
  group_by(genre) %>%
  summarize(median.speech = median(speechiness))
median.speechiness
```

```
## # A tibble: 6 x 2
##   genre median.speech
##   <fct>         <dbl>
## 1 edm          0.0625
## 2 latin        0.0643
## 3 pop          0.0479
## 4 r&b          0.064
## 5 rap          0.17
## 6 rock         0.0424
```

```r
median.speechiness %>% ggplot(aes(x = genre, y = median.speech, fill  = genre)) + geom_bar(stat = "iden
  ggtitle("Median Speechiness across genres")+
  labs(caption = "Figure 16: Median speechiness by genre, rap leads the way while rock comes last.")
```

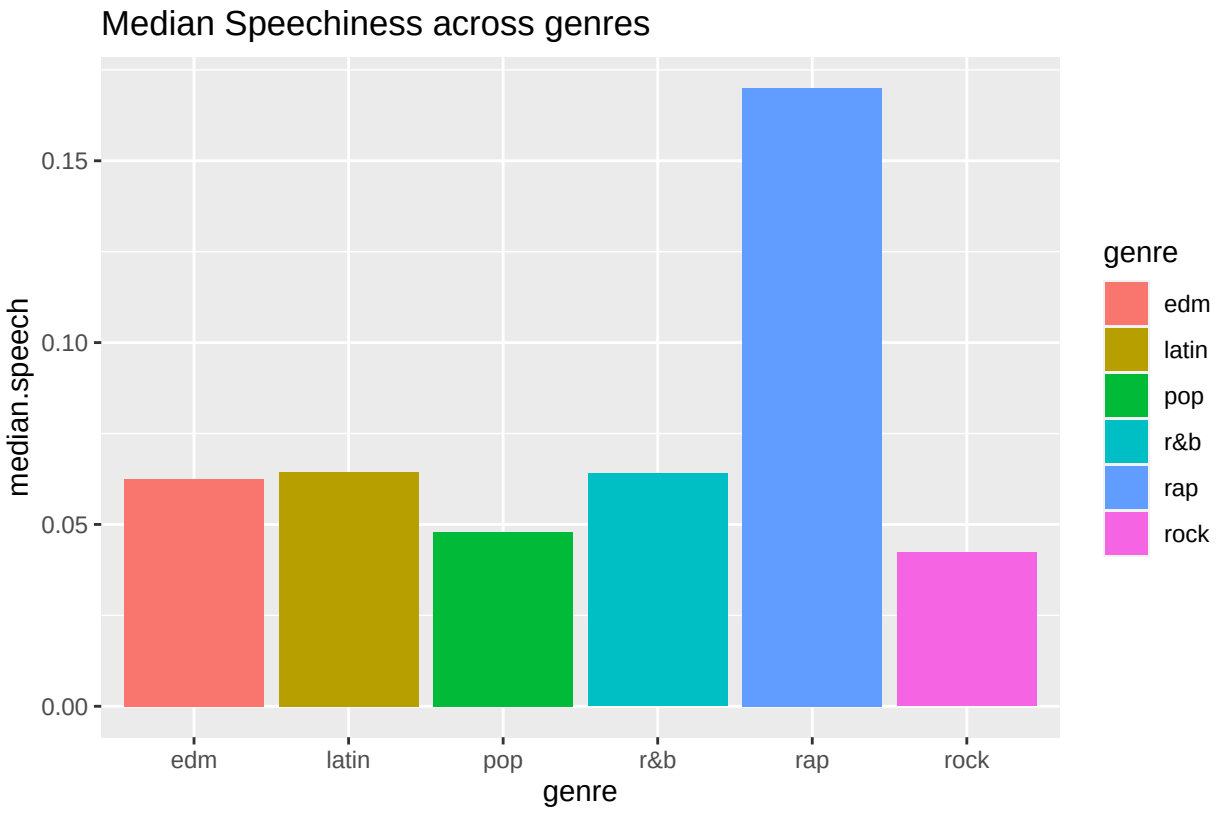


Figure 16: Median speechiness by genre, rap leads the way while rock comes last.

Since, Highly right skewed, we have checked the median. Yes, there is a difference in speechiness among different genre. Rap tends to be clearly the most speechy while rock and pop is the least speechy.

3. How does track popularity change over time?

```
#Track popularity based on genre vs year
spotify_songs %>% group_by(year) %>%
  summarise(Average_Popularity = mean(popularity)) %>%
  ggplot(aes(x=year, y= Average_Popularity)) + geom_point() +
  geom_smooth(se = F)+
  ggtitle("Average popularity each year")+
  labs(caption = "Figure 17: Yearly mean popularity of songs remains pretty constant.
       Early years due to less data, it seems to be less.")
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

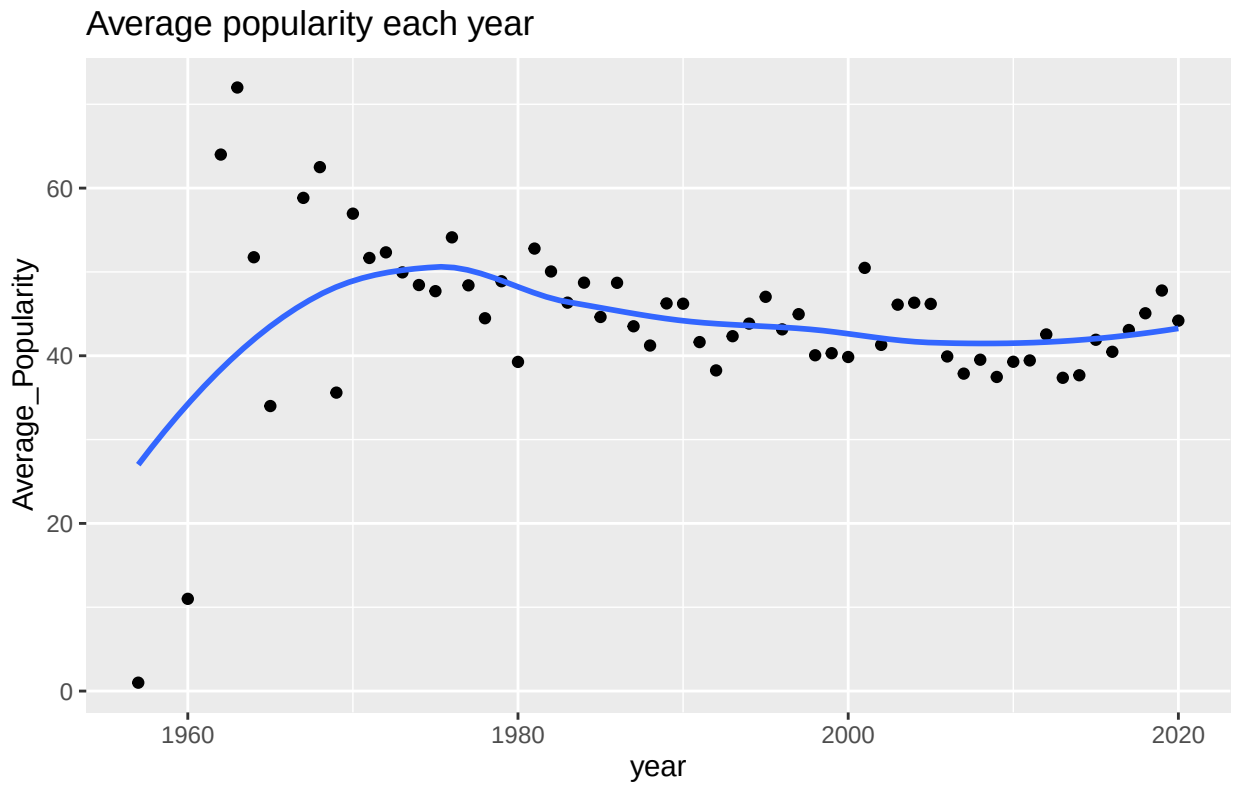


Figure 17: Yearly mean popularity of songs remains pretty constant. Early years due to less data, it seems to be less.

```
#Decade vs popularity (of each genre)
spotify_songs %>% group_by(decade) %>%
  filter(n()>2) %>% ggplot() +
  geom_boxplot(aes(x=decade, y = popularity, fill = decade))+
  facet_wrap(~genre)+
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))+
  ggtitle('Genre based popularity each decade') +
  labs(caption = "Figure 18. We can see that edm was pretty much nonexistent pre 2000,
       rap very less pre 1980s")
```

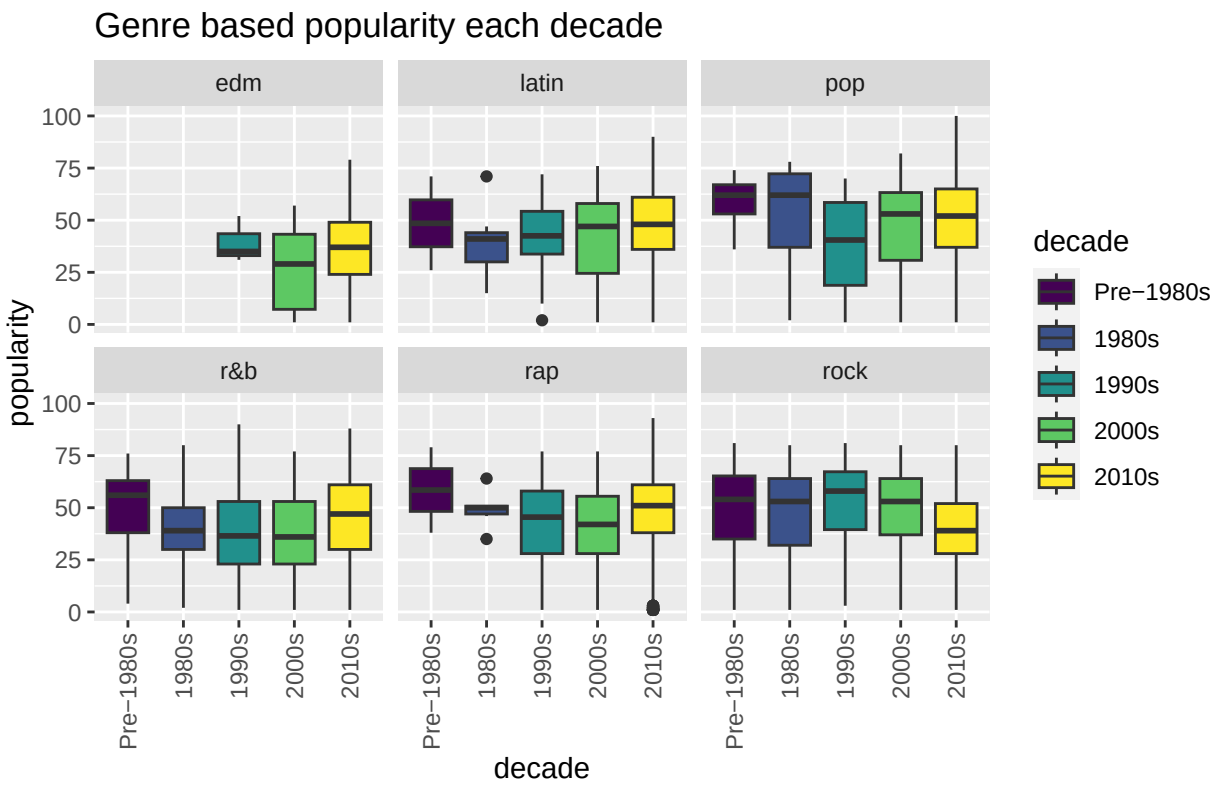


Figure 18. We can see that edm was pretty much nonexistent pre 2000, rap very less pre 1980s

In figure 17, We take the yearly average popularity of songs, and notice that initial years has less songs, hence it is low. But overall when good amount of data is present, yearly average popularity is pretty constant.

Figure 18 tells us how with time/decade the popularity of each genre changes. EDM songs can be seen in late 1990s, and has always been one of the least popular. Rap gained popularity in 1990s, but had very few tracks before 1990. Rock used to be very popular before but is less popular now.

We have extracted all the useful information and explored the relations of year variable. Thus removing this, in addition to song name and artist variable which do not add much significance.

```
#Removing final set of unimportant columns like song name, artist and year.
spotify_songs <-
  spotify_songs %>%
  dplyr::select(-c(year, name, artist))
spotify_songs %>% skim_without_charts()
```

Table 7: Data summary

Name	Piped data
Number of rows	6000
Number of columns	15
Column type frequency:	
factor	2
numeric	13

Table 7: Data summary

Group variables	None

Variable type: factor

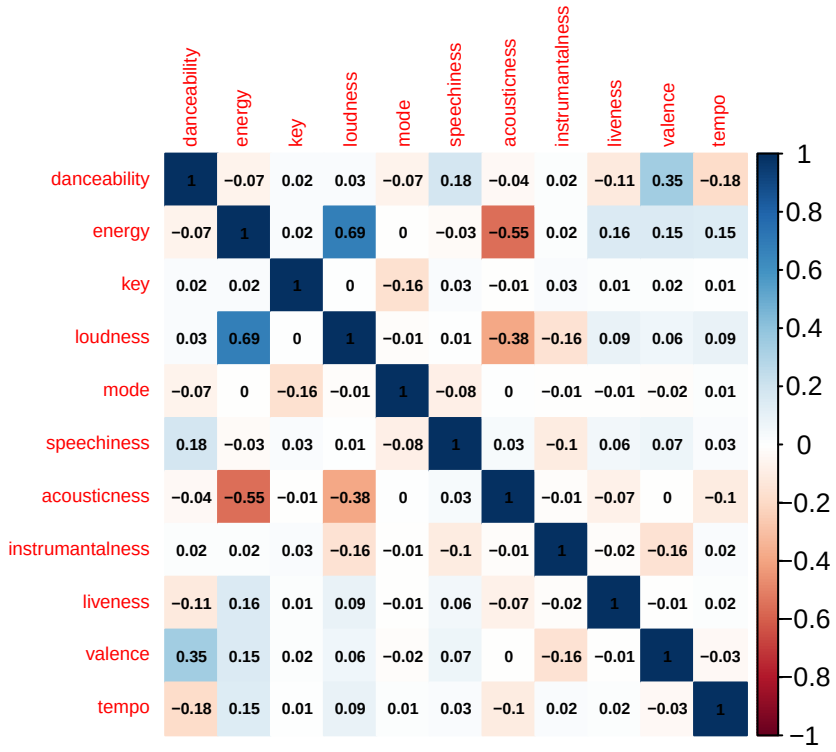
skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
genre	0	1	FALSE	6	edm: 1000, lat: 1000, pop: 1000, r&b: 1000
decade	0	1	TRUE	5	201: 4384, 200: 687, 199: 437, 198: 256

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
popularity	0	1	44.30	20.29	1.00	31.00	46.00	59.00	100.00
danceability	0	1	0.65	0.15	0.08	0.56	0.67	0.76	0.98
energy	0	1	0.70	0.19	0.01	0.58	0.72	0.85	1.00
key	0	1	5.41	3.57	0.00	2.00	6.00	8.00	11.00
loudness	0	1	-6.82	3.00	-35.43	-8.32	-6.29	-4.73	0.64
mode	0	1	0.57	0.50	0.00	0.00	1.00	1.00	1.00
speechiness	0	1	0.10	0.10	0.02	0.04	0.06	0.13	0.92
acousticness	0	1	0.19	0.23	0.00	0.02	0.09	0.28	0.99
instrumantalness	0	1	0.09	0.23	0.00	0.00	0.00	0.01	0.98
liveness	0	1	0.19	0.15	0.02	0.09	0.13	0.24	0.99
valence	0	1	0.52	0.23	0.00	0.33	0.52	0.70	0.99
tempo	0	1	121.30	26.96	48.98	100.00	122.01	135.01	220.25
duration	0	1	223.86	60.41	31.43	186.00	214.22	251.74	517.81

Finding the correlation between the song features given. There are a few feature with high correlation. These will be dropped when creating the recipe.

```
spotify_songs %>%
  dplyr::select(3:13) %>% #these are the song attributes
  scale() %>%
  cor() %>%
  corrplot(method = 'color',
           type = 'full',
           addCoef.col = "black",
           number.cex = 0.5,
           tl.cex = .6,
           title = 'Correlation Plot of song features',
           mar = c(1,0,2,0),
           sub = "Figure 19. Carrelation between all the song features")
```



Further Data Preprocessing

```
#Splitting
set.seed(1875837)
songs_split <- initial_split(spotify_songs, strata = "genre")
spotify_train <- training( songs_split )
spotify_test <- testing( songs_split )

#Stratified
spotify_train %>% group_by(genre) %>%
  summarize(Count = n()) %>% kable(caption = "Training Dataset Size")
```

Table 10: Training Dataset Size

genre	Count
edm	750
latin	750
pop	750
r&b	750
rap	750
rock	750

```
spotify_test %>% group_by(genre) %>%
  summarize(Count = n()) %>% kable(caption = "Testing Dataset Size")
```

Table 11: Testing Dataset Size

genre	Count
edm	250
latin	250
pop	250
r&b	250
rap	250
rock	250

```
#Creating the recipe for preprocessing the data
spotify_recipe <- recipe(genre~., data = spotify_train) %>%
  step_dummy(decade) %>% #for categorical predictors
  step_mutate(danceability = max(danceability+1)-danceability,
              energy = max(energy+1)-energy) %>% #left skew to right skew
  step_log(speechiness, danceability, energy,
           liveness, tempo, duration)%>% #removing skewness
  step_log(instrumantalness,offset = 1) %>% #removing skewness
  step_sqrt(acousticness) %>% #removing skewness
  step_normalize(all_predictors()) %>% #normalize
  step_corr(all_predictors(), threshold = .6) %>% #remove highly correlated predictors
  step_pca(all_predictors()) %>% #convert to principal components
  prep()

spotify_recipe
```

```
##

## -- Recipe ----------------------------------------------------------------

##

## -- Inputs

## Number of variables by role

## outcome:    1
## predictor: 14

##

## -- Training information

## Training data contained 4500 data points and no incomplete rows.

##
```

```
## -- Operations

## * Dummy variables from: decade | Trained

## * Variable mutation for: ~max(danceability + 1) - danceability, ... | Trained

## * Log transformation on: speechiness, danceability, energy, ... | Trained

## * Log transformation on: instrumantalness | Trained

## * Square root transformation on: acousticness | Trained

## * Centering and scaling for: popularity, danceability, energy, ... | Trained

## * Correlation filter on: loudness | Trained

## * PCA extraction with: popularity, danceability, energy, key, ... | Trained
```

```
#Finding optimum number of PCs
sdev <- spotify_recipe$steps[[8]]$res$sdev
ve <- sdev^2 / sum(sdev^2)

PC.pve <- tibble(
  pc = fct_inorder( str_c("PC", 1:16) ),
  pve = cumsum( ve ) # Takes the cumulative sum to get our PVE
)

PC.pve %>%
  ggplot( aes( x = pc,
               y = pve,
               fill = pve >= 0.9 ) ) + # Let's find the PC that explains
  # 90% of our variance
  geom_col() +
  theme( axis.text.x = element_text( angle = 90 ) ) + #rotate the x-axis labels
  ggtitle("Screeplot of the Principal Components")+
  labs(caption = "Figure 20. 13 Principal Components get us information about more than 90% of variatio
```

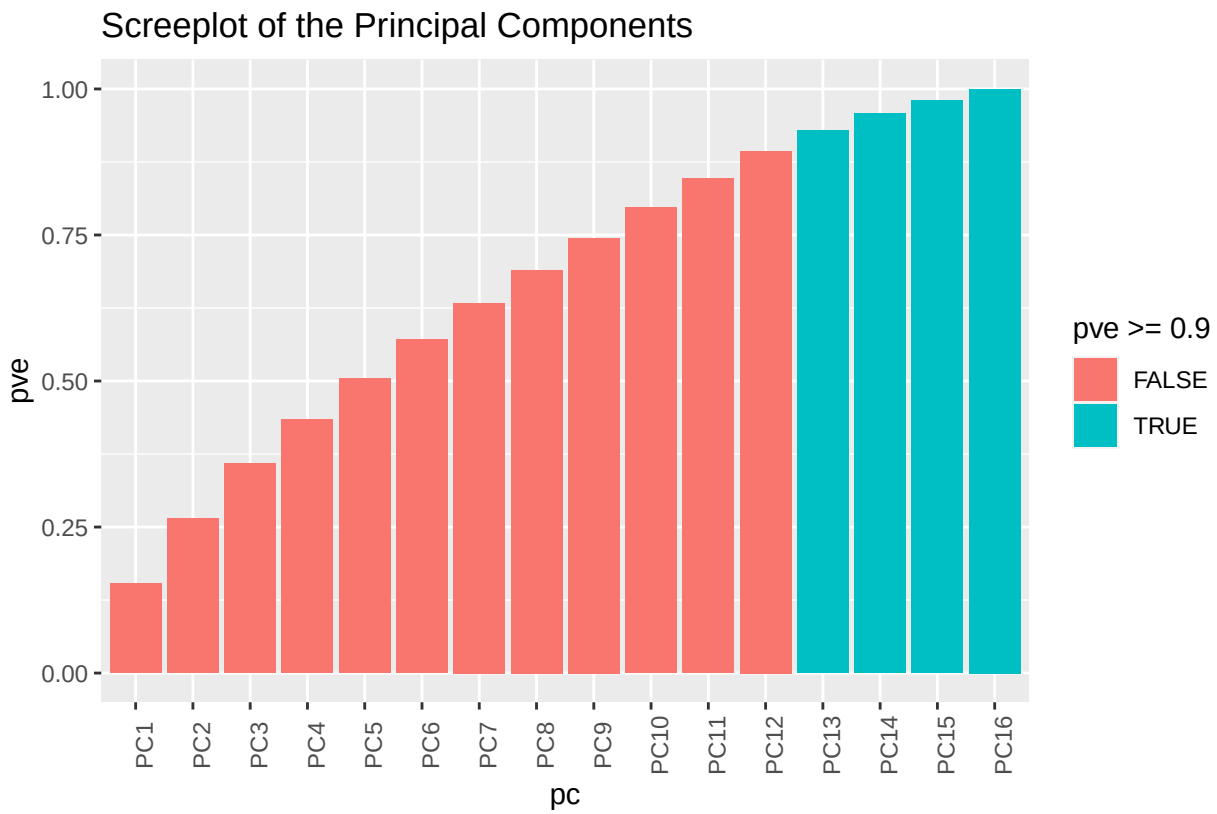


Figure 20. 13 Principal Components get us information about more than 90% of variation.

```
PC.pve %>%
  filter( pve >= 0.9) %>% # Let's look at those explaining 90% or more
  kable(caption = "13 Principal components explain more than 90% of the variation.")
```

Table 12: 13 Principal components explain more than 90% of the variation.

pc	pve
PC13	0.9290546
PC14	0.9583633
PC15	0.9800271
PC16	1.0000000

```
#Deciding we need 13 PCs, apply on recipe
spotify_recipe <- recipe(genre~., data = spotify_train) %>%
  step_dummy(decade) %>%
  step_mutate(danceability = max(danceability+1)-danceability,
              energy = max(energy+1)-energy) %>%
  step_log(speechiness, danceability, energy,
           liveness, tempo, duration)%>%
  step_log(instrumantalness,offset = 1) %>%
  step_sqrt(acousticness) %>%
  step_normalize(all_predictors()) %>%
  step_corr(all_predictors(), threshold = .6) %>%
```

```
  step_pca(all_predictors(), num_comp = 13) %>%
  prep()
spotify_recipe
```

```
##
## -- Recipe ----------------------------------------------------------------
##
## -- Inputs
## Number of variables by role
## outcome:    1
## predictor: 14
##
## -- Training information
## Training data contained 4500 data points and no incomplete rows.
##
## -- Operations
## * Dummy variables from: decade | Trained
## * Variable mutation for: ~max(danceability + 1) - danceability, ... | Trained
## * Log transformation on: speechiness, danceability, energy, ... | Trained
## * Log transformation on: instrumantalness | Trained
## * Square root transformation on: acousticness | Trained
## * Centering and scaling for: popularity, danceability, energy, ... | Trained
## * Correlation filter on: loudness | Trained
## * PCA extraction with: popularity, danceability, energy, key, ... | Trained
```

```
#Apply recipe on training data
spotify_train_preproc <- juice(spotify_recipe)
spotify_train_preproc %>% head()
```

```
## # A tibble: 6 x 14
##   genre   PC01   PC02   PC03   PC04   PC05   PC06    PC07   PC08    PC09    PC10
##   <fct>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>   <dbl>  <dbl>   <dbl>   <dbl>
## 1 edm    0.111 -0.276 -0.822  2.27   2.27  -0.268  0.0668  0.777  0.216   0.294
## 2 edm    0.882 -0.117 -2.54   1.87   1.50  -0.764 -2.13    0.267  1.04   -0.325
## 3 edm    0.713  1.60   0.990 -1.15  -1.44   0.412  1.59   -0.505 -0.964  -0.483
## 4 edm    0.692  3.06   0.905  3.36   1.12  -0.509 -0.591   0.510  0.857   2.05
## 5 edm    1.47   2.43   0.674 -1.20   1.30   1.24  -0.742   0.375  0.0764  0.827
## 6 edm   -2.98  -1.50  -1.35  -0.745  0.593  2.30  -0.0962  0.516 -0.0121 -0.0196
## # i 3 more variables: PC11 <dbl>, PC12 <dbl>, PC13 <dbl>
```

```
#Test dataset preprocessing
spotify_test_preproc <- bake(spotify_recipe, spotify_test)
spotify_test_preproc %>% head()
```

```
## # A tibble: 6 x 14
##   genre  PC01   PC02    PC03   PC04   PC05    PC06   PC07   PC08    PC09
##   <fct> <dbl>  <dbl>   <dbl>  <dbl>  <dbl>   <dbl>  <dbl>  <dbl>   <dbl>
## 1 edm   0.984 -0.277 -1.96   -0.211  0.998 -0.0538 -1.21  -0.231  0.856
## 2 edm   0.738  0.930 -1.79    0.288  1.09  -0.231  -1.11  -1.53   0.0882
## 3 edm   1.20   1.15  -0.725   0.360 -0.505 -0.746  -0.978 -0.644  1.35
## 4 edm   0.778  1.01   0.0557  0.225 -0.516 -0.0718 -0.657 -0.786 -1.26
## 5 edm   1.64   2.38   0.198   1.20  -0.132  0.821  -0.258 -0.309  0.204
## 6 edm   0.914  1.33  -0.430   0.763  0.983 -0.479  -1.37   1.49   1.91
## # i 4 more variables: PC10 <dbl>, PC11 <dbl>, PC12 <dbl>, PC13 <dbl>
```

Model Preperation

```
#Model Specifications

# LDA
lda_spec <-
  discrim_linear(mode = "classification") %>%
  set_engine( "MASS" )
lda_spec
```

```
## Linear Discriminant Model Specification (classification)
##
## Computational engine: MASS
```

```
#KNN
knn_spec <-
  nearest_neighbor( mode = "classification",
                    neighbors = tune()) %>%
  set_engine("kknn")
knn_spec
```

```
## K-Nearest Neighbor Model Specification (classification)
##
## Main Arguments:
```

```
##   neighbors = tune()
##
## Computational engine: kknn
```

```
#RF
rf_spec <- rand_forest(
  mode = "classification",
  mtry = tune(),
  trees = 100,
  min_n = tune()) %>%
  set_engine( "ranger")
rf_spec
```

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

```
#Bootstrapping
set.seed(1875837)
spotify_boots <-
  bootstraps(spotify_train_preproc, times = 10,  strata = genre)

#Cross Validation
set.seed(1875837)
spotify_cv <- vfold_cv(spotify_train_preproc, v = 10, strata = genre )
```

```
# KNN Tuning
# Creating KNN Grid
knn_spec_grid <- grid_regular(neighbors(range = c(1,100)),
                              levels = 20)
knn_spec_grid
```

```
## # A tibble: 20 x 1
##    neighbors
##        <int>
##  1         1
##  2         6
##  3        11
##  4        16
##  5        21
##  6        27
##  7        32
##  8        37
##  9        42
## 10        47
## 11        53
## 12        58
```

```
## 13        63
## 14        68
## 15        73
## 16        79
## 17        84
## 18        89
## 19        94
## 20       100
```

```
#Tuning KNN hyperparameter k using bootstraps
set.seed(1875837)
knn_grid <- tune_grid( object = knn_spec,
                       preprocessor = recipe(genre ~ . , data = spotify_train_preproc),
                       resamples = spotify_boots,
                       grid = knn_spec_grid )
```

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

```
#Check graph for best values
knn_grid %>%
  collect_metrics() %>%
  ggplot( aes( x = neighbors, y = mean)) +
  geom_point( size = 2 ) +
  geom_line( alpha = 0.75 ) +
  ggtitle("KNN performance metrics for different hyperparameters")+
  labs(caption = "Figure 21. The accuracy and roc_auc of the KNN model while tuning for hyperparameter
  facet_wrap( ~ .metric, scales = "free", nrow = 2)
```

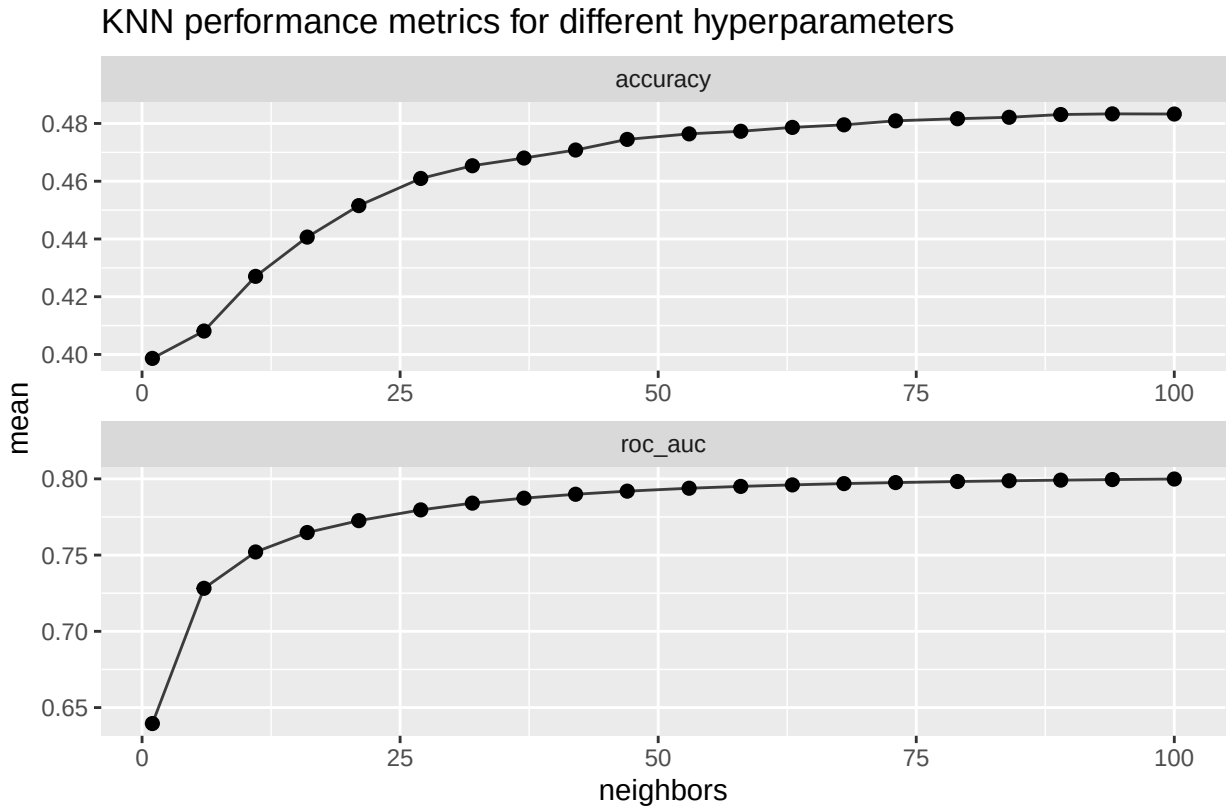


Figure 21. The accuracy and roc_auc of the KNN model while tuning for hyperparameter − neighbors.

```
#Selecting best KNN based on AUC
best_knn_auc <- select_best( knn_grid, "roc_auc")

final_knn <- finalize_model(knn_spec, best_knn_auc)
final_knn
```

```
## K-Nearest Neighbor Model Specification (classification)
##
## Main Arguments:
##   neighbors = 100
##
## Computational engine: kknn
```

```
# RF Tuning
# Creating RF Grid
rand_spec_grid <- grid_regular(
  finalize( mtry(),
            spotify_train_preproc %>%
              dplyr::select(-genre) ),
  min_n(),
  levels = 5 )
rand_spec_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
```

```
#Tuning RF hyperparameters mtry and min_n
set.seed(1875837)
rf_grid <- tune_grid( object = rf_spec,
                      preprocessor = recipe(genre ~ . , data = spotify_train_preproc),
                      resamples = spotify_boots,
                      grid = rand_spec_grid )
```

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

```
#Check graph for best Hyperparameter values
rf_grid %>%
  collect_metrics() %>%
  mutate( min_n = as.factor( min_n ) ) %>%
  ggplot( aes( x = mtry, y = mean, colour = min_n ) ) +
  geom_point( size = 2 ) +
  geom_line( alpha = 0.75 ) +
  ggtitle("RF performance metrics for different hyperparameters")+
  labs(caption = "Figure 22. The accuracy and roc_auc of the RF model while tuning for hyperparameters
       mtry and min_n") +
  facet_wrap( ~ .metric, scales = "free", nrow = 2)
```

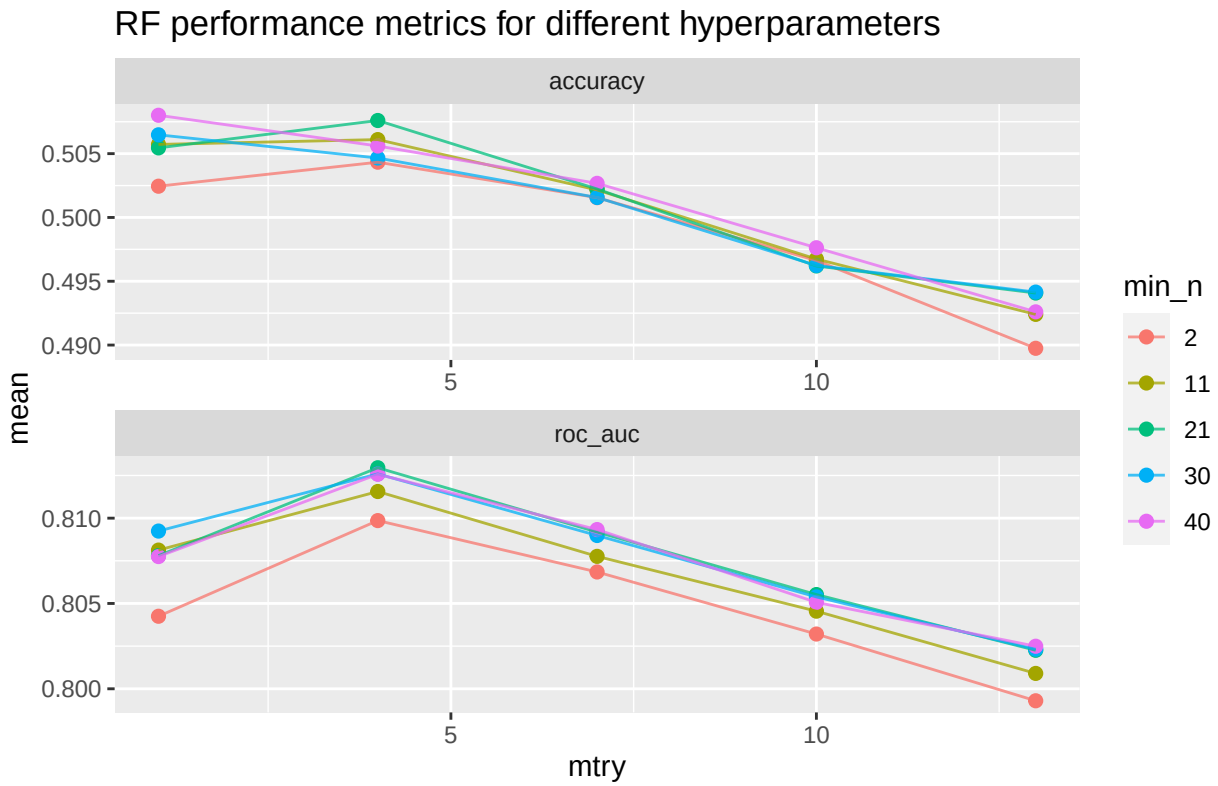


Figure 22. The accuracy and roc_auc of the RF model while tuning for hyperparameters – mtry and min_n

```
#Selecting best RF model based on AUC
best_rf_auc <- select_best( rf_grid, "roc_auc" )

final_rf <- finalize_model( rf_spec, best_rf_auc)
final_rf
```

```
## Random Forest Model Specification (classification)
##
## Main Arguments:
##   mtry = 4
##   trees = 100
##   min_n = 21
##
## Computational engine: ranger
```

Model Comparison

```
#Cross Validations of each model
#LDA CV
spotify_lda_workflow <-
  workflow() %>%
  add_model(lda_spec) %>%
  add_formula(genre~.)
```

```
spotify_lda_resamples <-
  fit_resamples(
    spotify_lda_workflow,
    resamples = spotify_cv)

#LDA Accuracy and AUC after the K-fold CV
lda_accuracy <- spotify_lda_resamples %>%
  collect_metrics() %>%
  filter(row_number() == 1) %>%
  dplyr::select(mean) %>%
  pull
lda_auc <- spotify_lda_resamples %>%
  collect_metrics() %>%
  filter(row_number() == 2) %>%
  dplyr::select(mean) %>%
  pull

#KNN CV
set.seed(1875837)
knn_cv <- fit_resamples( object = final_knn,
                         preprocessor = recipe(genre ~ . , data = spotify_train_preproc),
                         resamples = spotify_cv )

#KNN Accuracy and AUC after the K-fold CV
knn_accuracy <- knn_cv %>%
  collect_metrics() %>%
  filter(row_number() == 1) %>%
  dplyr::select(mean) %>%
  pull
knn_auc <- knn_cv %>%
  collect_metrics() %>%
  filter(row_number() == 2) %>%
  dplyr::select(mean) %>%
  pull

#RF CV
set.seed(1875837)
rf_cv <- fit_resamples( object = final_rf,
                         preprocessor = recipe(genre ~ . , data = spotify_train_preproc),
                         resamples = spotify_cv )

#RF Accuracy and AUC after the K-fold CV
rf_accuracy <- rf_cv %>%
  collect_metrics() %>%
  filter(row_number() == 1) %>%
  dplyr::select(mean) %>%
  pull
rf_auc <- rf_cv %>%
  collect_metrics() %>%
  filter(row_number() == 2) %>%
  dplyr::select(mean) %>%
  pull
```

Apply the three models again on train dataset to find confusion matrix, sensitivity and specificity.

```
#LDA fit on Training Data
set.seed(1875837)
spotify_lda <- lda_spec %>%
  fit( genre ~ . , data = spotify_train_preproc )

#LDA Predictions of Training Data
train_lda_preds <- predict( spotify_lda, # The predictions
                            new_data = spotify_train_preproc,
                            type = "class") %>%
  bind_cols(spotify_train_preproc %>%  # Add the truth
              dplyr::select(genre)) %>%
  rename(lda_pred = .pred_class)

#LDA Confusion Matrix
lda_conf_mat <- train_lda_preds %>%
  conf_mat(truth = genre, estimate = lda_pred)

lda_conf_mat %>%
  autoplot(type = "heatmap") +
  ggtitle("Confusion matrix of LDA model predicting training data")+
  labs(caption = "Figure 23. Confusion matrix of the trained LDA model when applied back on training da
  theme_minimal()
```

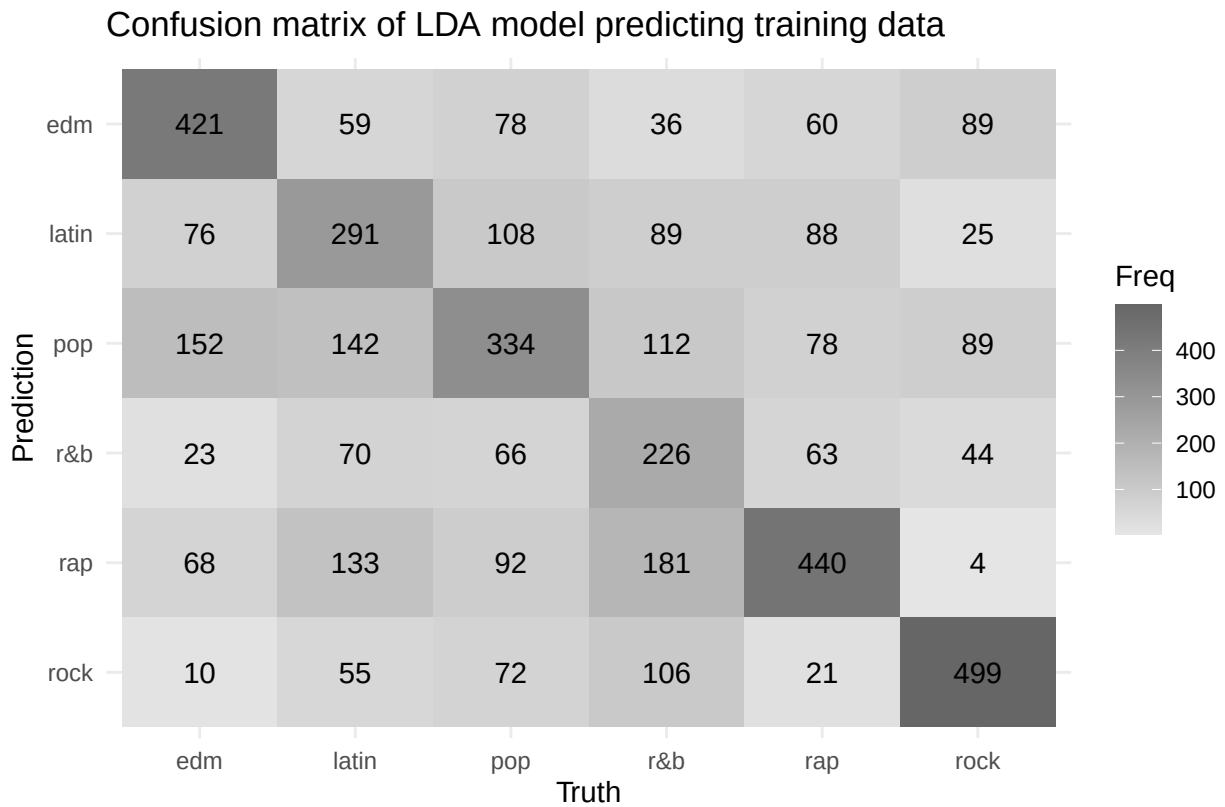


Figure 23. Confusion matrix of the trained LDA model when applied back on training data

```r
#Sensitivity
lda_sensitivity <- train_lda_preds %>%
  sens( truth = genre, estimate = lda_pred ) %>%
  dplyr::select(.estimate) %>% pull()

#Specificity
lda_specificity <- train_lda_preds %>%
  spec( truth = genre, estimate = lda_pred ) %>%
  dplyr::select(.estimate) %>% pull()
```

```r
#KNN fit on Training Data
set.seed(1875837)
spotify_knn <- final_knn %>%
  fit( genre ~ . , data = spotify_train_preproc )

#KNN Predictions of Training Data
train_knn_preds <- predict( spotify_knn, # The predictions
                            new_data = spotify_train_preproc,
                            type = "class") %>%
  bind_cols(spotify_train_preproc %>%  # Add the truth
              dplyr::select(genre)) %>%
  rename(knn_pred = .pred_class)

#KNN Confusion Matrix
knn_conf_mat <- train_knn_preds %>%
  conf_mat(truth = genre, estimate = knn_pred)

knn_conf_mat %>%
  autoplot(type = "heatmap") +
  ggtitle("Confusion matrix of KNN model predicting training data")+
  labs(caption = "Figure 24. Confusion matrix of the trained KNN model when applied back on training da
  theme_minimal()
```

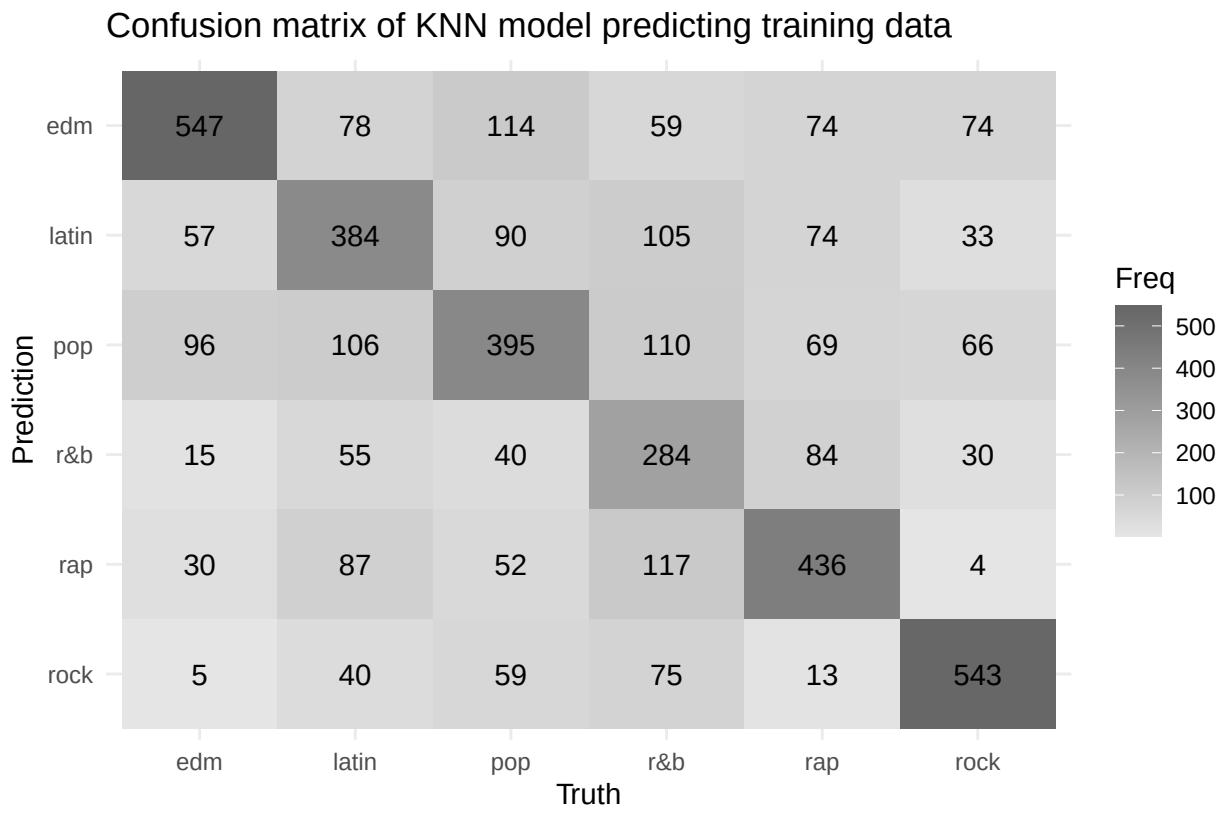


Figure 24. Confusion matrix of the trained KNN model when applied back on training data

```
#Sensitivity
knn_sensitivity <- train_knn_preds %>%
  sens( truth = genre, estimate = knn_pred ) %>%
  dplyr::select(.estimate) %>% pull()

#Specificity
knn_specificity <- train_knn_preds %>%
  spec( truth = genre, estimate = knn_pred ) %>%
  dplyr::select(.estimate) %>% pull()
```

```
#RF fit on Training Data
set.seed(1875837)
spotify_rf <- final_rf %>%
  fit( genre ~ . , data = spotify_train_preproc )


#RF Predictions of Training Data
train_rf_preds <- predict( spotify_rf, # The predictions
                           new_data = spotify_train_preproc,
                           type = "class") %>%
  bind_cols( spotify_train_preproc %>%  # Add the truth
               dplyr::select(genre)) %>%
  rename(rf_pred = .pred_class)

#RF Confusion Matrix
rf_conf_mat <- train_rf_preds %>% conf_mat(truth = genre, estimate = rf_pred)
```

```
rf_conf_mat %>%
  autoplot(type = "heatmap") +
  ggtitle("Confusion matrix of RF model predicting on Training Data")+
  labs(caption = "Figure 25. Confusion matrix of the trained RF model when applied back on training data
  theme_minimal()
```

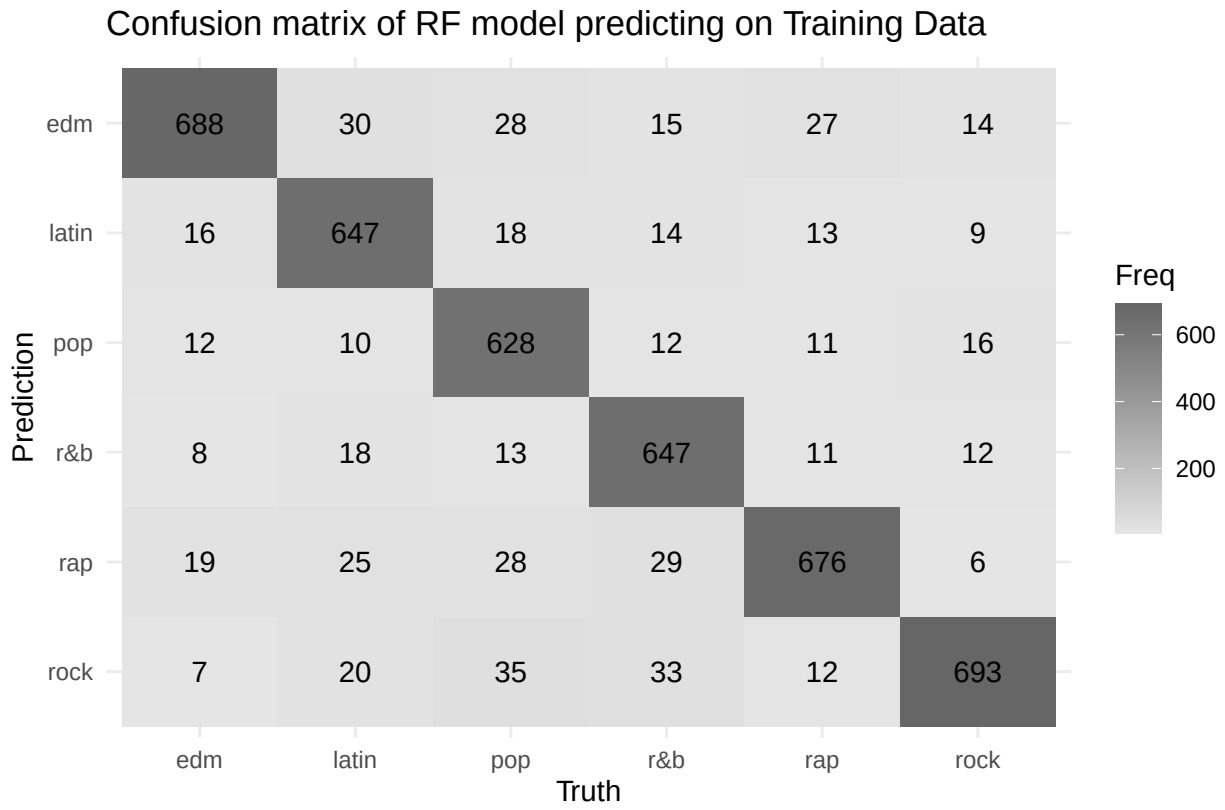


Figure 25. Confusion matrix of the trained RF model when applied back on training data

```
#Sensitivity
rf_sensitivity <- train_rf_preds %>%
  sens( truth = genre, estimate = rf_pred ) %>%
  dplyr::select(.estimate) %>% pull()

#Specificity
rf_specificity <- train_rf_preds %>%
  spec( truth = genre, estimate = rf_pred ) %>%
  dplyr::select(.estimate) %>% pull()
```

Comparison Table

```
tribble(
  ~Model, ~Accuracy, ~AUC, ~Sensitivity, ~Specificity,
  "LDA", lda_accuracy, lda_auc, lda_sensitivity, lda_specificity,
  "KNN", knn_accuracy, knn_auc, knn_sensitivity, knn_specificity,
  "RF", rf_accuracy, rf_auc, rf_sensitivity, rf_specificity
) %>% kable(caption = "Comparison of the three classification models (LDA, KNN and RF. Random Forest mo
```

Table 13: Comparison of the three classification models (LDA, KNN and RF. Random Forest model clearly outperforms the rest.

Model	Accuracy	AUC	Sensitivity	Specificity
LDA	0.4862222	0.8015230	0.4913333	0.8982667
KNN	0.4975556	0.8115550	0.5753333	0.9150667
RF	0.5208889	0.8177896	0.8842222	0.9768444

From the above comparison table, Table 13, it is clear that the Random Forest model is easily the best among the three models implemented. With better accuracy, AUC, sensitivity and specificity, it checks all the boxes. Now we will test the Randome Forest model on the test dataset to get its test performance metrics.

Best Model Testing

RF is the best model

```
#RF Fit on Testing Dataset
test_rf_preds <- predict( spotify_rf, # The predictions
                             new_data = spotify_test_preproc,
                          type = "class") %>%
  bind_cols( spotify_test_preproc %>%  # Add the truth
               dplyr::select(genre)) %>%
  rename(rf_pred = .pred_class)

#Confusion matrix of best model on test data
rf_conf_mat <- test_rf_preds %>% conf_mat(truth = genre, estimate = rf_pred)

rf_conf_mat %>%
  autoplot(type = "heatmap") +
  ggtitle("Confusion matrix of RF model applied on Test Data")+
  labs(caption = "Figure 26. The Confusion Matrix of RF model (proven best model),
       when applied on the preprocessed test data.") +
  theme_minimal()
```

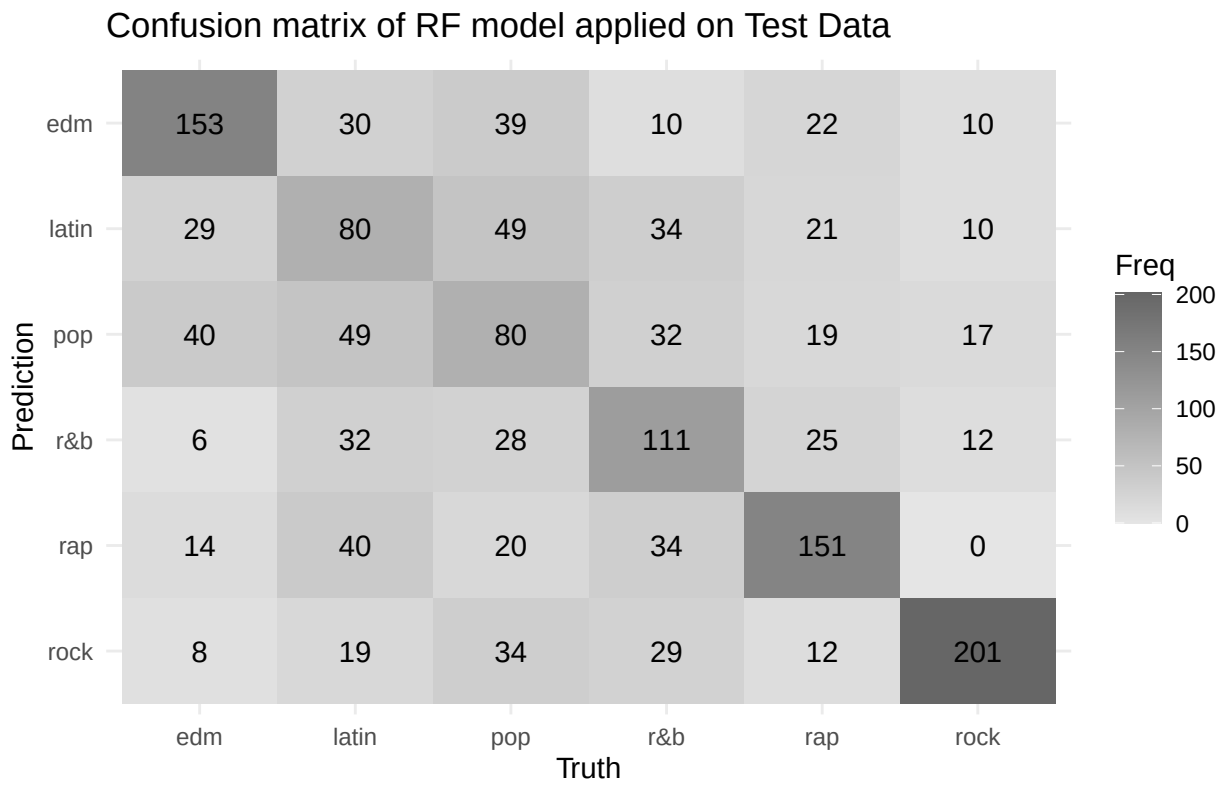


Figure 26. The Confusion Matrix of RF model (proven best model), when applied on the preprocessed test data.

```
test_rf_preds <- test_rf_preds %>%
  bind_cols(predict( spotify_rf, # The predictions
                     new_data = spotify_test_preproc,
                     type = "prob"))

test_rf_preds %>% roc_curve(truth = genre, .pred_edm:.pred_rock) %>%
  autoplot() + labs(title = "ROC Curves facetted by genre",
                    caption = "Figure 27. The ROC curves of each genre for the prepared RF model.
                    Rock, edm and rap seem to be clearly distinguishable for the model")
```

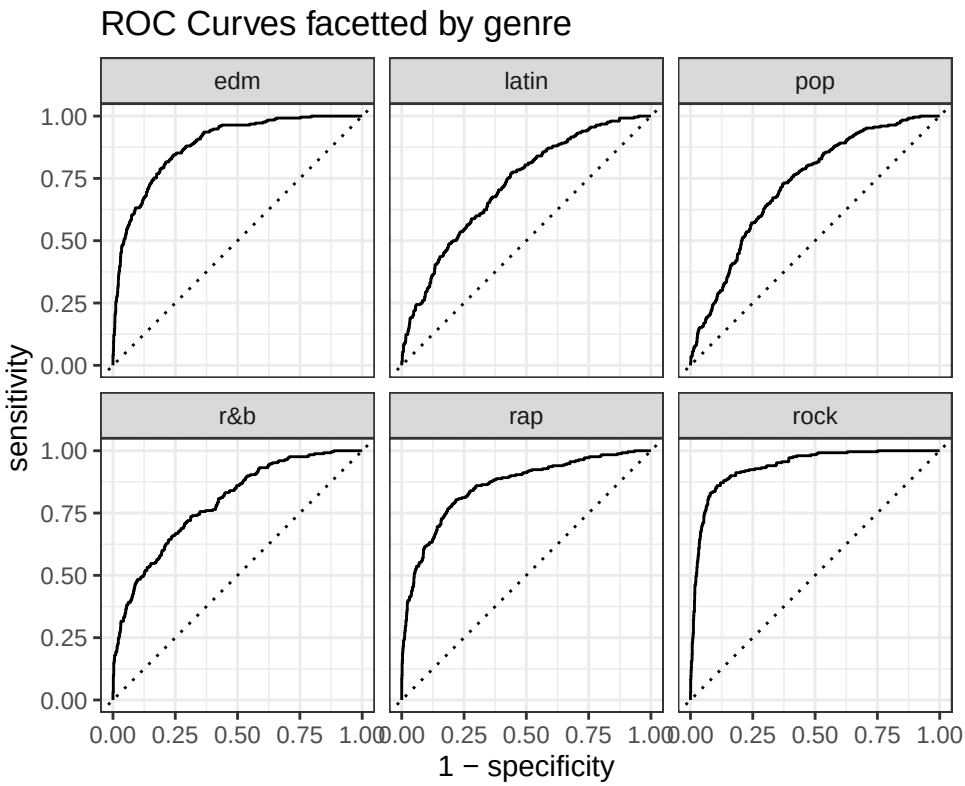


Figure 27. The ROC curves of each genre for the prepared RF model. Rock, edm and rap seem to be clearly distinguishable for the model

```
#Accuracy
test_acc <- test_rf_preds %>% metrics( truth = genre, estimate = rf_pred) %>%
  filter(row_number()==1) %>%
  dplyr::select(.estimate) %>% pull

#AUC
test_auc <- test_rf_preds %>% roc_auc(truth = genre, .pred_edm:.pred_rock) %>%
  dplyr::select(.estimate) %>% pull

#Sensitivity
test_sens <- test_rf_preds %>%
  sens(truth = genre, estimate=rf_pred) %>%
  dplyr::select(.estimate) %>% pull

#Specificity
test_spec <- test_rf_preds %>% spec(truth = genre, estimate=rf_pred) %>%
  dplyr::select(.estimate) %>% pull
```

From the ROC curve (Figure 27) we see that rock, edm and rap cover the more area (or are farther from the y=x line, or random guessing line), this tells us that the RF model works best for these three genre compared to others (latin, spop and r&b). This, information gained by us can also be checked from the confusion matrix (Figure 26), we see that the model has predicted correctly 201/250 (rock), 153/250 (edm), 151/250 (rap), 111/250 (r&b), 80/250 (latin) and 80/250 (pop).

The performance metrics of the best model (RF Model) on preprocessed test dataset is:

```
tribble(
  ~Metric, ~Value,
  "Accuracy", test_acc,
  "AUC", test_auc,
  "Sensitivity", test_sens,
  "Specificity", test_spec
) %>% kable(caption = "The various performance metrics of the Random Forest model when applied to the te
```

Table 14: The various performance metrics of the Random Forest model when applied to the test dataset. With Accuracy of almost 51.7%.

Metric	Value
Accuracy	0.5173333
AUC	0.8192635
Sensitivity	0.5173333
Specificity	0.9034667