Final Report

aditya venugopalan a1899824

2023-11-17

Executive Summary

The following report is an analysis based on the Dataset provided by the world famous audio streaming and media services provider SPOTIFY. As the largest music streaming service in the world with the estimate of 165 million paying subscribers , the founders of spotify are very much intrested in improving their customer services . In order to improve thier services, as a data scientist I was requested to give a prediction based on a prediction model , the predictons were mainly focusing on genre when comapred with other variables based on year of release, dancebability, tempo, speechiness . This prediction could help to provide accurate recommendations to the spotify team as well as the users for a better experience . When we beggan our analysis the dataset was holding information about 32,833 songs accompanied by 23 variables. Based on this three predictive models were set up in such a way that they gave away different metrics such as AUC, Accuracy , Sensitivity and Specificity. After the models were set up they were comapred among themselves. It was later found out that after fitting and evaluation Random Forest model was the best model for genre prediction.

Methods

The analysis was successfully analyzed on the programming language called R studio. The R studio is an integrated environment for R, which is done by using R language

The spotify dataset consisted of 32,833 observations on 23 variables or features.

But since the founders were only intrested in 4 varaibles:

a) track_album_release_date: this variable gives us information about the year when the song was released

b) speechiness - this variable tells how speechy the song is

c) danceability - this variable tells us how dancebale the song is

d) temp - this feature tells us about the tempo of the song

The reaqson for mentioning the other features is simply because currently it is not the concern of the founders , but if needed they can be viewed muliple times as they are called again and again before the process of data cleanining and dimensions reduction.

The goal of this analysis was to build a project was to build models thar are caare were low accuracy with Pop being the least accurate. ##The 4 metrics used to evaluate the performance of the model are: ## Sensitivity: measures how the model tells correctly identifies instances of a specific genre. ## Specificity: measures how well the model correctly tells instances that are not of a specific genre. ## Also the variables that the founders are interested in, the variables which are also used in predicting the genre of the song are energy, acous,valence, duration, and loudness of the song. ## Accuracy: measures the overall correctness of the model's predictions across all genres. #Precision: focuses on how well the model's positive predictions match the actual positive instances.

Conclusion

We can succesffully derive that the features the founders were intrested in has a huge influence on song popularity .The features which the founders asked us to anlayze on that is song release year,speechiness, danceability, and tempo showed influence in determining genre of the song.

The Random Forest model is was selected for genre prediction due to its best and accurate performance in comparison to Linear Discriminant Analysis and K-Nearest Neighbors models. We can get better performance by using other algorithms such as Logistic Regression, Naive Bayes Classifiers as these classification algorithms can improve comparing performance.

Last but not the least in future there is high possibility of music industry to grow therefore, continuos efforst should be made for improving and implementing different algorithms and strategies to achieve these goals.

The best model rf model had sesnitivity of 0.564, specificty of 0.912 , accuracy of 0.560 and precison of 0.560

Appendix

First we will begin our analysis with loading relevant libraries. With the help of these libraries it will be easy and possible for us to achieve success in our analysis

Importing the data

Now data will be imported in the format of a csv file .

```
spotify_songs <- readr::read_csv('https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/
```

Initial Data Analysis

Here we will have a brief look at the data , which is in it's purest raw form. In other technical words this data has possibility of having some erros which can give us some trouble or might give us not the best prediction or analysis with the highest accuracy.

```r
Initial_spotify_songs_data <- spotify_songs %>%
  skim_without_charts()
Initial_spotify_songs_data
```

Table 1: Data summary

Name	Piped data
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
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

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
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

Observations derived from the initial Data Analysis

(1) We can observe that there are total 32833 observations in our dataset .

(2) There are in total 23 variables which also include the independent ones .

(3) Out of those 23 variables we have 10 variables which belong to the character data type and 13 variables that belong to numeric data type .

(4) We can also observe that there are some missing values in the variables of "track_name" , "track_artist" and "track_album_name" respectively. Also each of the above mentioned variables have the same amount of missing data which is 5.

(5) Due to limitations in computing power, we will be doing the following steps in order to achieve our goals.

(a) The dataset will be reduced to 1000 observations per class of the independent variable , "playlist_genre"

(b) The missing will be removed as well, since they are a barrier for us to analyse the data.

Data Cleaning

```r
spotify_cleaned <- na.omit(spotify_songs)
inspect_na(spotify_cleaned)
```

```
## # A tibble: 23 x 3
##    col_name                   cnt  pcnt
##    <chr>                    <int> <dbl>
##  1 track_id                     0     0
##  2 track_name                   0     0
##  3 track_artist                 0     0
##  4 track_popularity             0     0
##  5 track_album_id               0     0
##  6 track_album_name             0     0
##  7 track_album_release_date     0     0
##  8 playlist_name                0     0
##  9 playlist_id                  0     0
## 10 playlist_genre               0     0
## # i 13 more rows
```

As expected the above data has no missing values for the operation and execution of eliminating the missing values(Na values).

Selecting relevant columns

```r
spotify_cleaned <- spotify_cleaned %>%
  subset(select = -c(track_id, track_name, track_artist,
track_album_id, track_album_name,
playlist_name, playlist_id,
playlist_subgenre))
```

In the above chunk, rather than selecting the subset of coulumns that we want (which I had initially intended), I decided mention the columns that will be filtered out.

From the above tables we understand that in the case of the variable playlist__genre have comparatively less unique values hence should a factor data type also the data type for the variable "track__album__release__date" is not correct since it should be a numerical data type. Also we are only inrested in the year when the song was released .

Data Type Conversion

```r
spotify_cleaned <- spotify_cleaned %>% mutate(
  playlist_genre = factor(playlist_genre),
year =str_match(track_album_release_date,'(\\d{4})')[,1] %>% paste0("-01-01") %>% year()
)


head(spotify_cleaned$year)
```

```
## [1] 2019 2019 2019 2019 2019 2019
```

Well, the above piece of code might look small, but a lot of things are happening here, so I will take my time and explain.

first we changed the data type of data variable playlist_genre to factor. Then during the analysis it was found out that the data variable year had two different formats 'yyyy-mm-dd' and 'yyyy'. However we are only intrested in the year of the track released therefore I decided to keep the 'yyyy' format as the optimal one. First the dates were convertred to 'yyyy' format by extracting the values which had 4 consecutive digits from the string date. Next the paste0() function appends "-01-01" to each year string, and the reason behind this was to create "yyyy-mm-dd' format. After that the year() parses these date strings and converts them to a Date object in lubridate package(already installed and extracted). and finnaly these date objects are stored in the variable called year

Dimension reduction

```
set.seed(1899824)
genre_cleaned <- spotify_cleaned %>%
  group_by(playlist_genre) %>%
  slice_sample(n=1000)
genre_cleaned %>%
  count(playlist_genre)
```

```
## # A tibble: 6 x 2
## # Groups:   playlist_genre [6]
##   playlist_genre     n
##   <fct>          <int>
## 1 edm             1000
## 2 latin           1000
## 3 pop             1000
## 4 r&b             1000
## 5 rap             1000
## 6 rock            1000
```

Now we can see that the size of the dataset is smaller , which means it will now require less computational power.

slice_sample() was used and the purpose for doing this was, it chooses randomly n number of observations from every group of genre.

Skimming the data

Again we will skim the data to make sure each data variable is in its proper data type , by doing so we will get an idea where modifications are required in order to proceed

```
spotify_cleaned
```

```
## # A tibble: 32,828 x 16
##    track_popularity track_album_release_date playlist_genre danceability energy
##               <dbl> <chr>                    <fct>                 <dbl>  <dbl>
##  1               66 2019-06-14               pop                   0.748  0.916
##  2               67 2019-12-13               pop                   0.726  0.815
##  3               70 2019-07-05               pop                   0.675  0.931
##  4               60 2019-07-19               pop                   0.718  0.93
##  5               69 2019-03-05               pop                   0.65   0.833
##  6               67 2019-07-11               pop                   0.675  0.919
##  7               62 2019-07-26               pop                   0.449  0.856
##  8               69 2019-08-29               pop                   0.542  0.903
##  9               68 2019-06-14               pop                   0.594  0.935
## 10               67 2019-06-20               pop                   0.642  0.818
## # i 32,818 more rows
## # i 11 more variables: key <dbl>, loudness <dbl>, mode <dbl>,
## #   speechiness <dbl>, acousticness <dbl>, instrumentalness <dbl>,
## #   liveness <dbl>, valence <dbl>, tempo <dbl>, duration_ms <dbl>, year <dbl>
```

```
skim_without_charts(spotify_cleaned)
```

Table 4: Data summary

Name	spotify_cleaned
Number of rows	32828
Number of columns	16
Column type frequency:	
character	1
factor	1
numeric	14
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
track_album_release_date	0	1	4	10	0	4529	0

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
playlist_genre	0	1	FALSE	6	edm: 6043, rap: 5743, pop: 5507, r&b: 5431

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
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	225796.83	59836.49	4000.00	187804.50	216000.00	253581.25	517810.00
year	0	1	2011.14	11.42	1957.00	2008.00	2016.00	2019.00	2020.00

Now we can see our data is ready to be processed further as we see fit. We made necessary changes in order for the analysis

Exploratory Analysis

we will begin our actual analysis by understanding how and what factors can help to predict the following

a)the year the song was released;

b) how "speechy" the song is;

c) how danceable the song is; and

d) the tempo of the song

1. Genre vs the year the song was released

First we will extract all values of genre possible. The reason behind this to make sure all values are included and hence we can also observe whether the data produce by plot is accurate or not.

Unfortunately i won't be extracting all possible values of variable "year" since there are lot of values in variable "year"

```r
genre <-unique(spotify_cleaned$playlist_genre)
print(genre)
```

```
## [1] pop   rap   rock  latin r&b   edm
## Levels: edm latin pop r&b rap rock
```

```r
spotify_cleaned %>%
  ggplot(aes(x = fct_reorder(playlist_genre,year), y = year, fill = playlist_genre)) +
  geom_boxplot() +
  labs(title = "Boxplot of the released year of the song for each genre", x = "Genre", y = "Year Releas
  theme_minimal() + labs(caption = " Figure 1 : track album release year for each genre " )
```

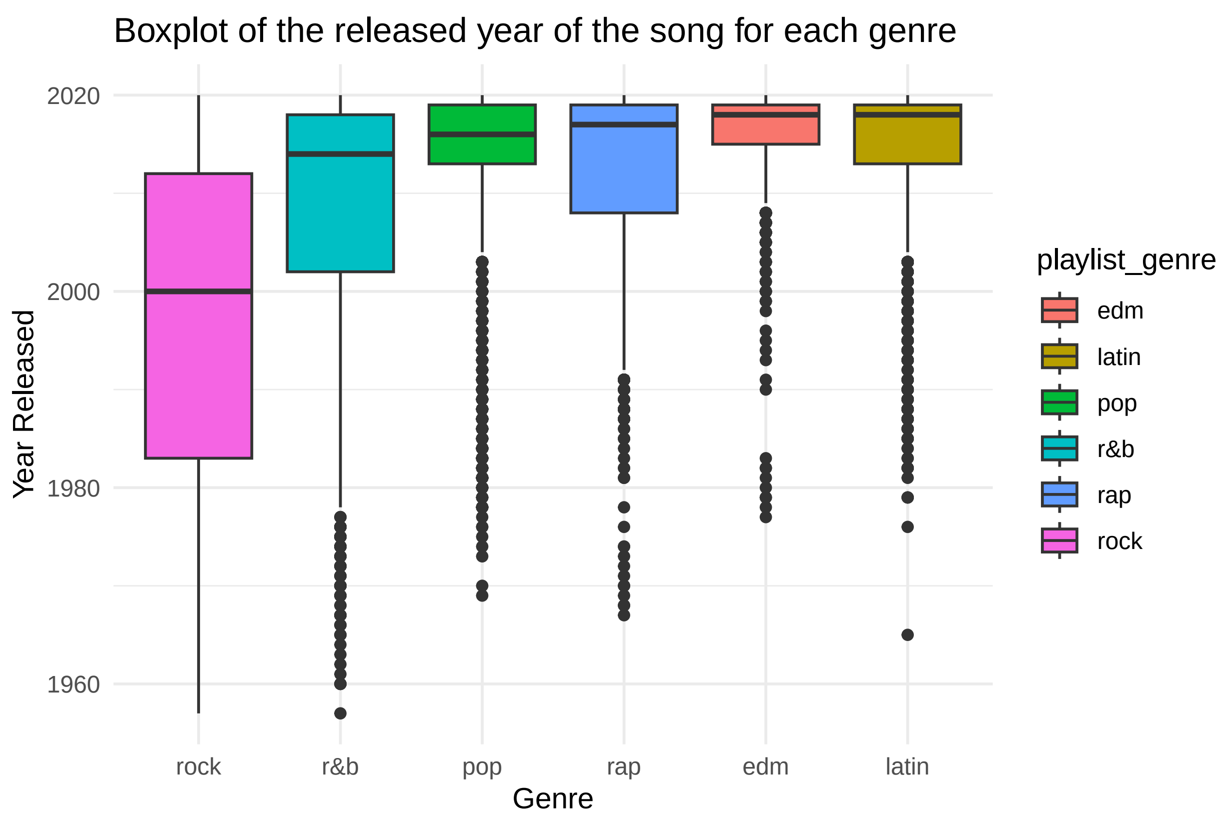


Figure 1 : track album release year for each genre

2. Genre vs how "speechy" the song is

Again let's begin with knowing all possible entires or values of the variable "speechiness".

```
speechiness <-unique(spotify_cleaned$speechiness)
head(speechiness)
```

```
## [1] 0.0583 0.0373 0.0742 0.1020 0.0359 0.1270
```

```
spotify_cleaned %>%
  ggplot(aes(x= fct_reorder(playlist_genre, speechiness), y= speechiness, fill = playlist_genre)) +
  geom_boxplot() +
  labs(title = "Boxplot of 'speechy'  of the song for each genre", x = "Genre", y = "Speechiness" ) +
  theme_minimal()  +labs(caption = " Figure 2 : speechiness for each genre " )
```

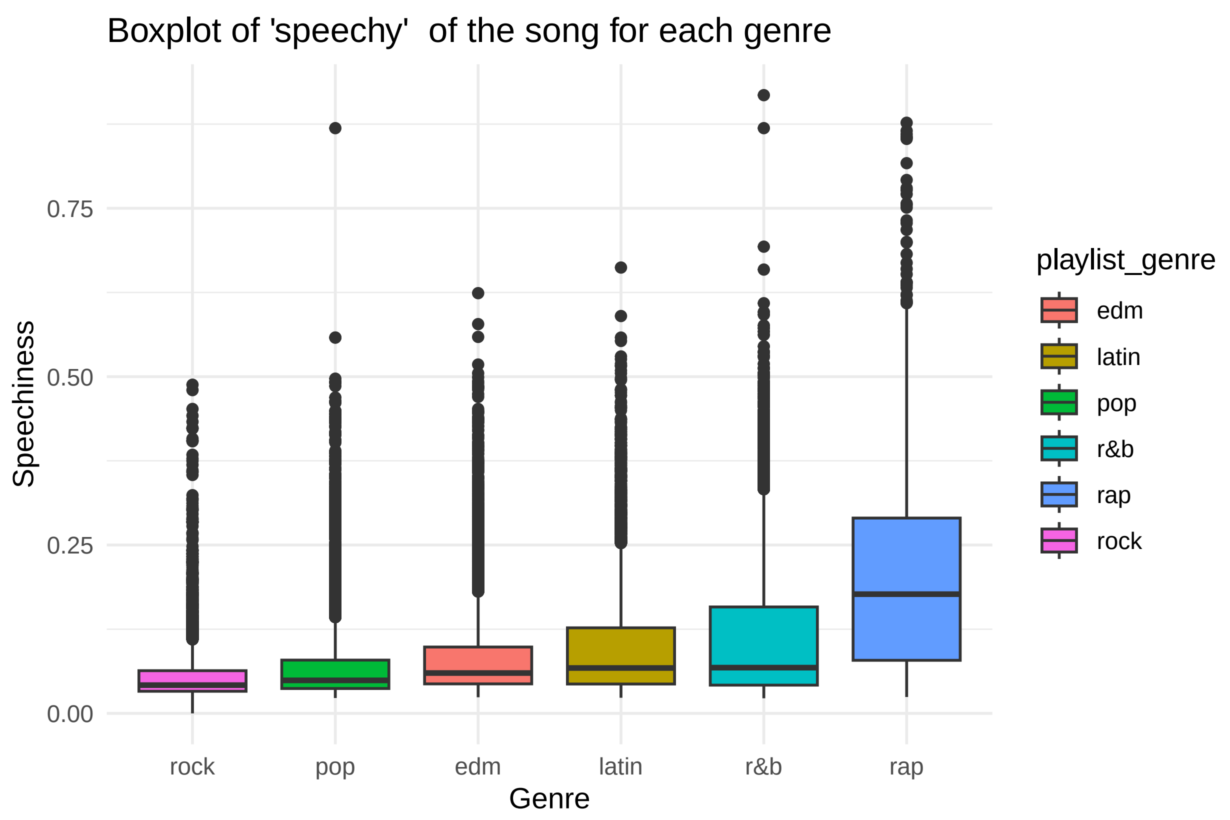


Figure 2 : speechiness for each genre

# 3. Genre vs how danceable the song is

We will begin with getting all possible values of dancebility . One thing I would like to point out that the values are extracted just to make sure all of them are included and not being exculded, there is no intention of making the report complicated by displaying such huge range of values. However the function head will be used just to make sure we get a rough idea about the possible values , but all possible values can be seen if requested .

```r
spotify_cleaned %>%
  ggplot(aes(x = fct_reorder(playlist_genre, danceability), y = danceability, fill = playlist_genre)) +
  geom_boxplot() +
  labs(title=  "Boxplot of how 'danceablility'  is   for each genre", x = "Genre", y = "Danceability" )
  theme_minimal() +labs(caption = " Figure 3 : danceability for each genre " )
```

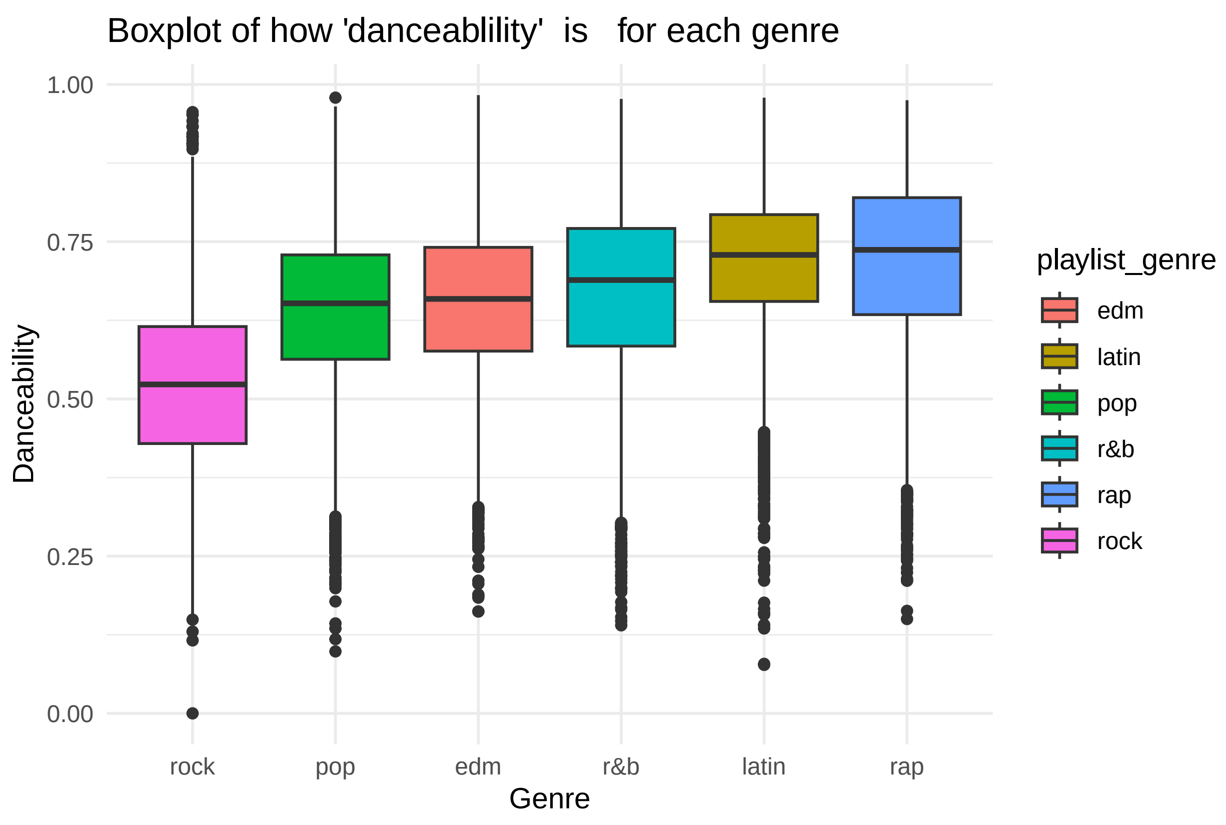


Figure 3 : danceability for each genre

# 4. Genre vs the tempo of the song

We will here also repeat the same process by extracting all possible values of the variable "tempo"

```
tempo <-unique(spotify_cleaned$speechiness)
head(tempo)
```

```
## [1] 0.0583 0.0373 0.0742 0.1020 0.0359 0.1270
```

```
spotify_cleaned %>%
  ggplot(aes(x= fct_reorder(playlist_genre, tempo), y= tempo, fill= playlist_genre)) +
  geom_boxplot() +
    labs(title=  "Boxplot of how 'tempo'  is   for each genre", x = "Genre", y = "Tempo" ) +
  theme_minimal() +labs(caption = " Figure 4 : tempo for each genre " )
```

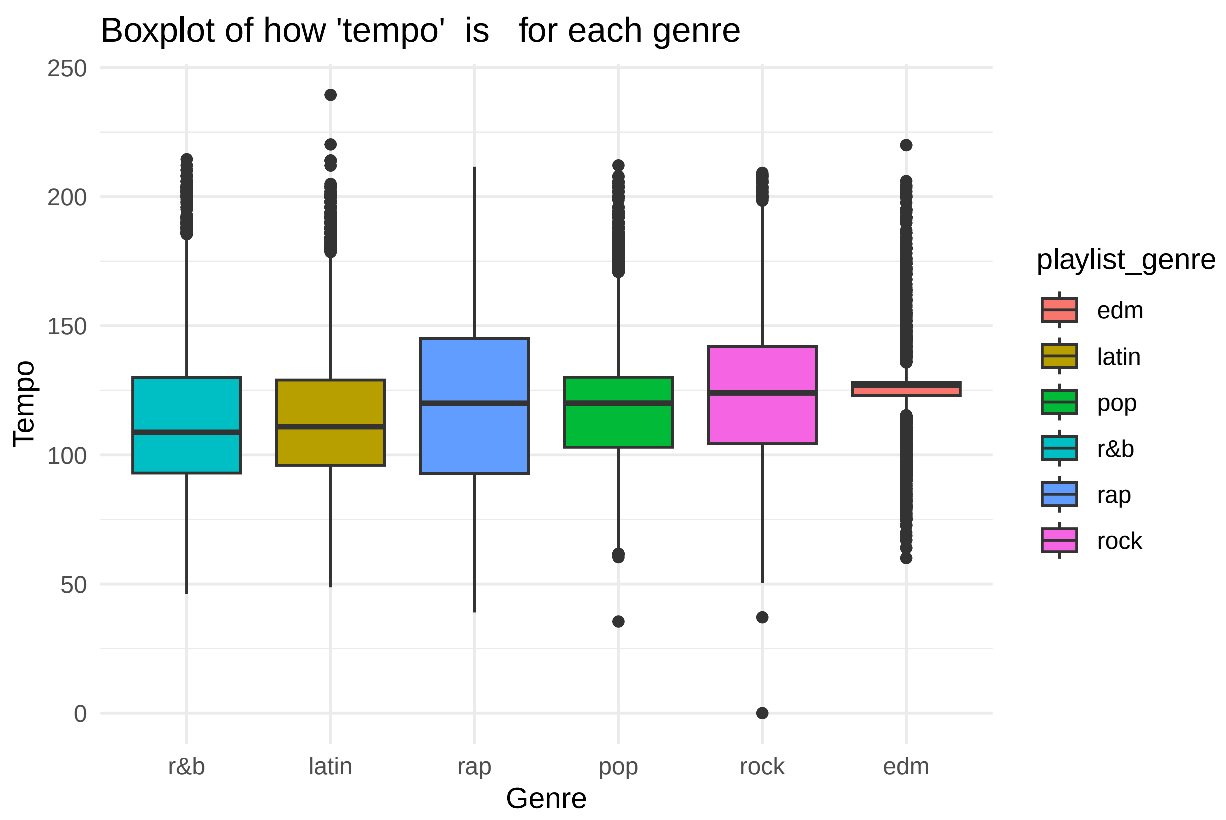


Figure 4 : tempo for each genre

# Exploring further

now we will dig a bit deeper and make follwoing comparisons

a) Does the popularity of songs differ between genres?

b)Is there a difference in speechiness for each genre?

c) How does track popularity change over time?

1. Genre vs popularity

```
spotify_cleaned %>%
  ggplot(aes(x= fct_reorder(playlist_genre, track_popularity), y= track_popularity, fill= playlist_genr
  geom_boxplot() +
  labs( title = "Boxplot of  'popularity'  for  each song genre", x = "Genre", y = "Popularity of song"
  theme_minimal()+ labs(caption = " Figure 5 : popularity for each genre " )
```

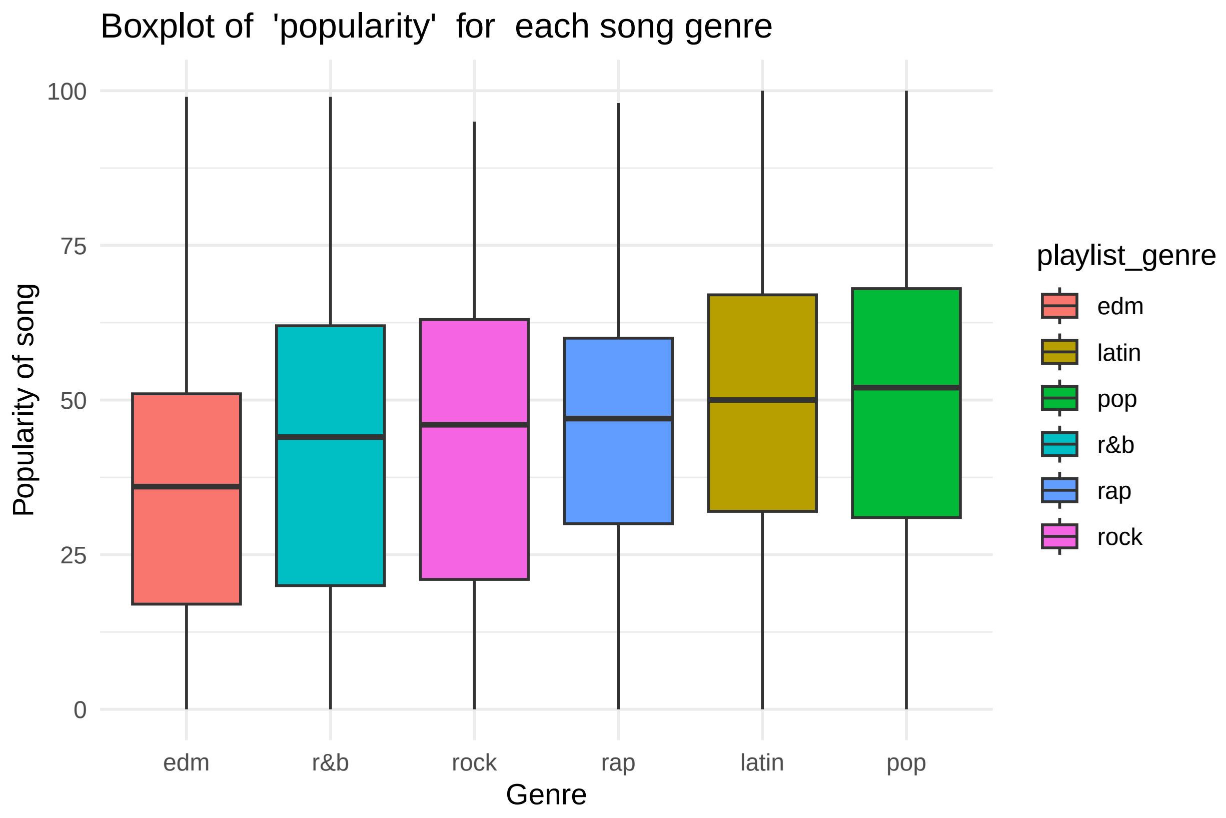
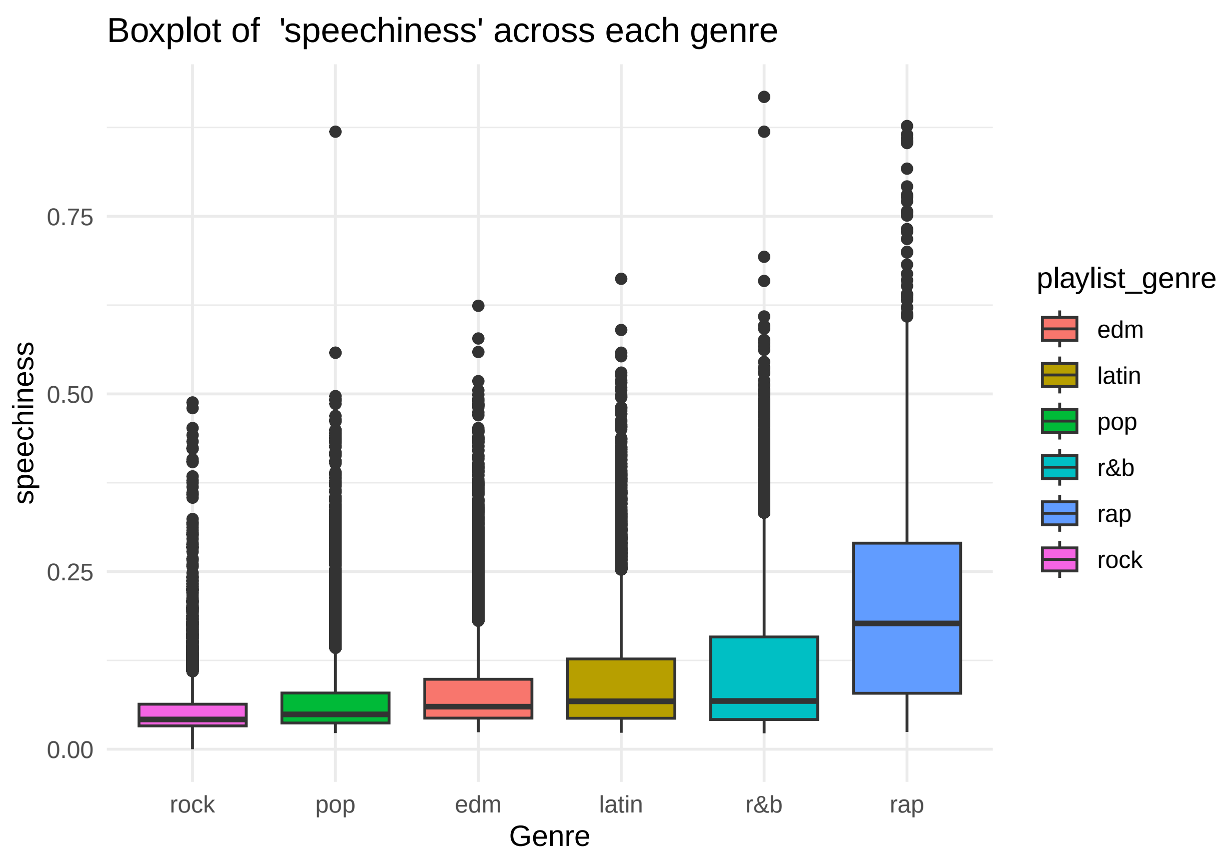


Figure 5 : popularity for each genre

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

```r
spotify_cleaned %>%
  ggplot(aes(x= fct_reorder(playlist_genre, speechiness), y= speechiness, fill= playlist_genre)) +
  geom_boxplot() +
  labs( title = "Boxplot of  'speechiness' across each genre", x = "Genre", y = "speechiness" ) +
  theme_minimal()
```



3. Track Popularity vs Time when the song was released (year)

.

```r
track_across_years <- spotify_cleaned %>%
  group_by(playlist_genre, year) %>%
  summarize(average_pop = mean(track_popularity),  count = n())
```

```
## `summarise()` has grouped output by 'playlist_genre'. You can override using
## the `.groups` argument.
```

```r
 ggplot(data = track_across_years, aes(x = year, y = average_pop, color = playlist_genre)) +
  geom_point() +
  labs(
    title = "Scatterplot of Track Popularity over Time",
    x = "Year of Release",
    y = "Average Popularity"
  ) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) + labs(caption = " Figure 6 : popularity for
```

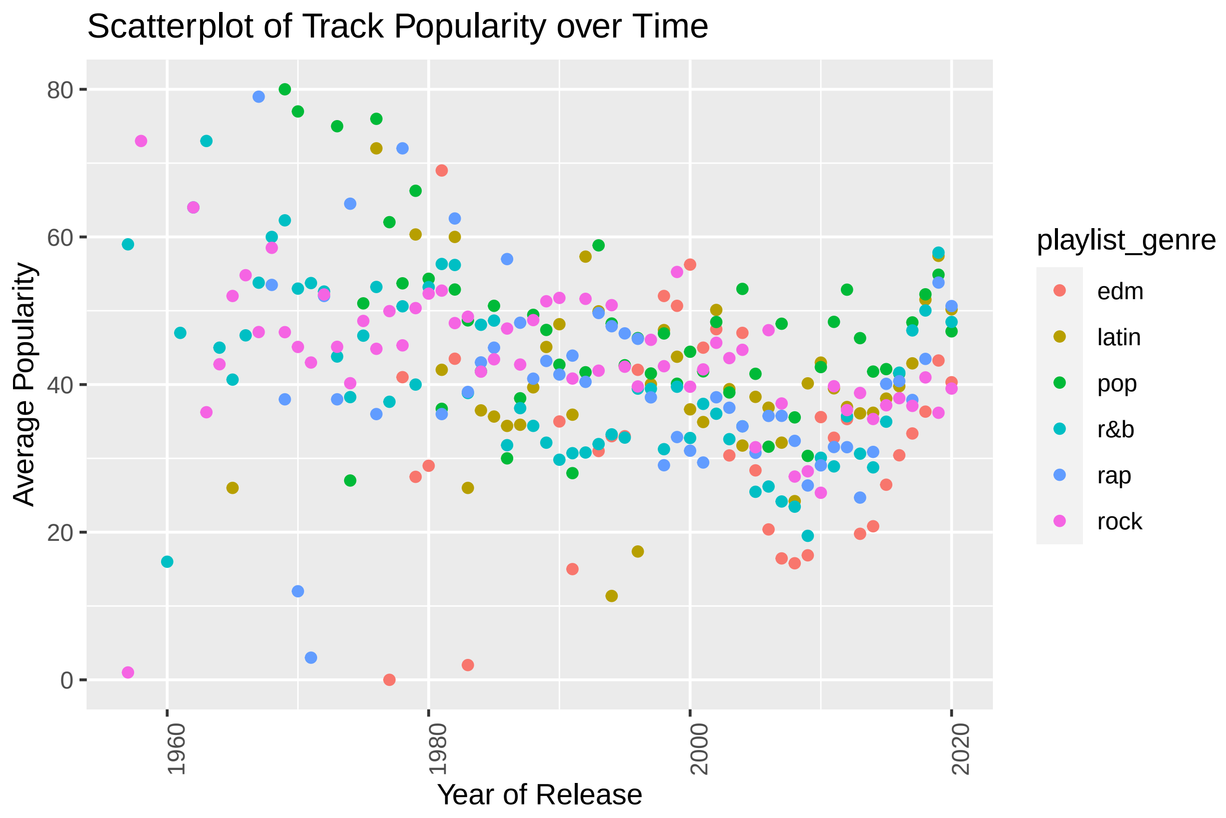


Figure 6 : popularity for each genre

Splitting and Preprocessing the data

Now we will split the dataset into training set and testing set , to understand how the models will work.

```r
set.seed(1899824)
spotify_split <- initial_split(genre_cleaned)
spotify_split
```

```
## <Training/Testing/Total>
## <4500/1500/6000>
```

```r
spotify_trained <- training(spotify_split)
spotify_tested <- testing(spotify_split)
```

Preprocessing

```r
spotify_recipe <- recipe(playlist_genre ~., data = spotify_trained) %>%
  step_rm(contains("date")) %>%
step_zv( all_predictors() ) %>%
step_normalize( all_predictors()) %>%
step_corr(all_predictors() ) %>%
prep()
spotify_recipe
```

```
##

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

##

## -- Inputs

## Number of variables by role

## outcome:    1
## predictor: 15

##

## -- Training information

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

##

## -- Operations

## * Variables removed: track_album_release_date | Trained

## * Zero variance filter removed: <none> | Trained

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

## * Correlation filter on: <none> | Trained
```

```r
spotify_trained_pre <- juice(spotify_recipe)
spotify_tested_pre <- bake(spotify_recipe, new_data = spotify_tested)
```

```r
head(spotify_trained_pre)
```

```
## # A tibble: 6 x 15
##   track_popularity danceability energy    key loudness   mode speechiness
##              <dbl>        <dbl>  <dbl>  <dbl>    <dbl>  <dbl>       <dbl>
## 1            0.434       -0.535  1.44  -0.382   1.39    0.887     -0.294
## 2           -1.23         1.51  -0.150 -0.942  -0.0336  0.887      0.0315
## 3           -0.944       -1.16   1.46   0.458   1.45   -1.13      -0.268
## 4           -0.457        1.23   0.816  1.58    0.348   0.887     -0.612
## 5           -1.71        -0.279  1.07   1.30    0.392   0.887     -0.734
## 6            0.272       -0.659  1.27  -0.942   1.05    0.887     -0.416
## # i 8 more variables: acousticness <dbl>, instrumentalness <dbl>,
## #   liveness <dbl>, valence <dbl>, tempo <dbl>, duration_ms <dbl>, year <dbl>,
## #   playlist_genre <fct>
```

```
head(spotify_tested_pre)
```

```
## # A tibble: 6 x 15
##   track_popularity danceability energy    key loudness   mode speechiness
##              <dbl>        <dbl>  <dbl>  <dbl>    <dbl>  <dbl>       <dbl>
## 1           -0.336       0.185   1.36   1.02     0.754  0.887      -0.584
## 2           -1.07        0.0531  0.716  1.58    -0.609 -1.13       -0.651
## 3           -0.984       1.17    0.772  1.58     1.09  -1.13       -0.406
## 4            0.799      -1.84    1.50   0.178    1.50  -1.13        0.447
## 5           -0.822      -0.735   1.09   0.738    0.928 -1.13       -0.596
## 6           -0.133       0.689   0.189 -0.942    0.734  0.887      -0.753
## # i 8 more variables: acousticness <dbl>, instrumentalness <dbl>,
## #   liveness <dbl>, valence <dbl>, tempo <dbl>, duration_ms <dbl>, year <dbl>,
## #   playlist_genre <fct>
```

Bootstrapping

In this section, before we begin our model tuning and our process for K- nearest neighbours we will create Bootstrap with 10 bootstrap samples and the resampling will be done for the "playlist_genre".

```
set.seed( 1899824 )
spotify_boot <- bootstraps( spotify_trained_pre, times = 10, strata = playlist_genre )
```

Model Building

One of the most important part of my report. Here I will be building 3 models that are : a) LDA model b) Knn model c) Random Forest Model. After building all the three models above mentioned , they will be analysed in such a way that the best model is selected . The selection will be based on follwing conditions. 1) AUC 2) Sensitivity 3) Specificity

The reasons for selecting a particular model since it was proven to be best and accurate will also be explained in both techinal and non techinal languages(terms).

K-Nearest Neighbour Model

We will begin by craeating a model specification for a k-NN model , here we will be tuning the neighbour parameter

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

Now we will be using the function called grid_regular which is used to make a grid of 20 k-values that ranges from 1 to 100 , this is done in order to tune our model .

```
spotify_k_grid <- grid_regular(neighbors( range = c( 1, 100 )),
levels = 20 )
spotify_k_grid
```

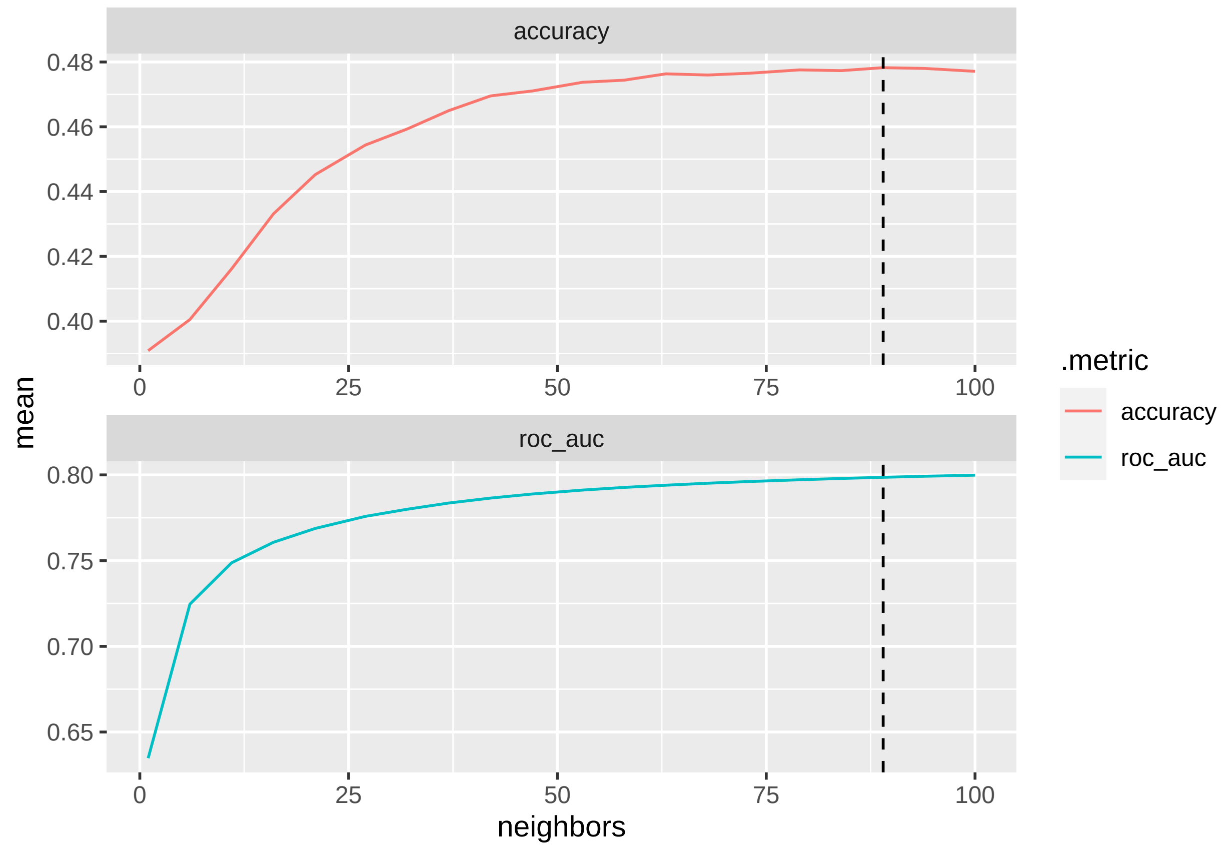
```
## # A tibble: 20 x 1
##    neighbors
##        <int>
##  1         1
##  2         6
##  3        11
##  4        16
##  5        21
##  6        27
##  7        32
##  8        37
##  9        42
## 10        47
## 11        53
## 12        58
## 13        63
## 14        68
## 15        73
## 16        79
## 17        84
## 18        89
## 19        94
## 20       100
```

The above tibble is nothing but a grid of 20 k-values.

```
spotify_knn_tune <- tune_grid(object = spotify_knn,
                      preprocessor = recipe(playlist_genre ~ . , data = spotify_trained_pre),
                      resamples = spotify_boot,
                      grid = spotify_k_grid )
```

Now we will see the variation of accuracy and auc for different neighbour values of K-NN

```
spotify_knn_tune %>%
  collect_metrics() %>%
  ggplot( aes( x = neighbors, y = mean, color = .metric)) +
  geom_line() +
  facet_wrap( ~.metric, scales = "free", nrow = 3) +
  geom_vline(xintercept = 89, linetype = "dashed", color = "black")
```



```
spotify_knn_best <- select_best( spotify_knn_tune, "accuracy")
spotify_knn_best
```

```
## # A tibble: 1 x 2
##   neighbors .config
##       <int> <chr>
## 1        89 Preprocessor1_Model18
```

Last step we will finalize our code

```
spotify_knn_final <- finalize_model( spotify_knn, spotify_knn_best )
spotify_knn_final
```

```
## K-Nearest Neighbor Model Specification (classification)
##
## Main Arguments:
##   neighbors = 89
##
## Computational engine: kknn
```

Random forest Model

Now moving onto the second model which is Random forest model. In this model we will be be tuning and selecting the best random model

First step involves making the model specification to tune mtry and min_n parameters

```r
spotify_rf <- rand_forest(
mode = "classification",
mtry = tune(),
trees = 100,
min_n = tune()
) %>%
set_engine( "ranger", importance = "permutation" )
```

Now the next step involves making grid to tune Rf model

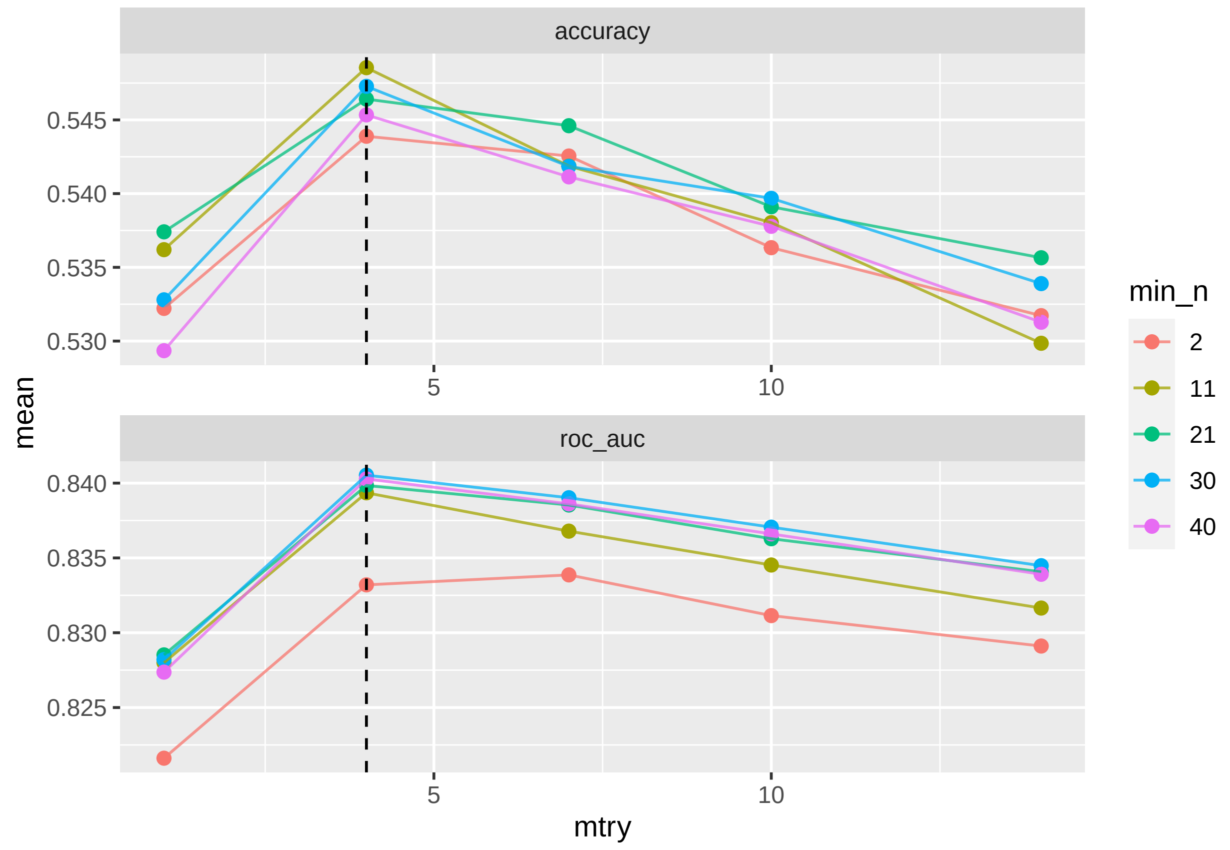
```r
spotify_rf_grid <- grid_regular(
finalize( mtry(),spotify_trained_pre %>%
            dplyr::select( -playlist_genre ) ),
min_n(),
levels = 5 )
```

Now we will tune our random forest model

```r
set.seed( 1899824 )
spotify_rf_grid <- tune_grid( object = spotify_rf, preprocessor =  recipe(playlist_genre ~ . , data = sp
resamples = spotify_boot,
grid = spotify_rf_grid )
```

Now its tme to observe the changes of accuracy and AUC for min_n and mtry

```r
spotify_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 ) +
facet_wrap( ~ .metric, scales = "free", nrow = 3 ) +
  geom_vline(xintercept = 4, linetype = "dashed", color = "black")
```



Now we will select the best RD model

```
spotify_rf_best <- select_best(spotify_rf_grid, "accuracy")
spotify_rf_best
```

```
## # A tibble: 1 x 3
##    mtry min_n .config
##   <int> <int> <chr>
## 1     4    11 Preprocessor1_Model07
```

Now the final steps involves finalizing the model

```
spotify_rf_final <- finalize_model(spotify_rf, spotify_rf_best)
spotify_rf_final
```

```
## Random Forest Model Specification (classification)
##
## Main Arguments:
##   mtry = 4
##   trees = 100
##   min_n = 11
```

```
##
## Engine-Specific Arguments:
##   importance = permutation
##
## Computational engine: ranger
```

Model Selection

Since we are working with a limited dataset along with less computational power , 5 is optimal choice for the number of folds

```r
set.seed(1899824)
spotify_cv <- vfold_cv(spotify_trained_pre, v=5)
```

Linear Discriminant Analysis Model

Now we will make a model specific for LDA model.

```r
spotify_lda <- discrim_linear("classification") %>%
  set_engine("MASS")
spotify_lda
```

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

Model Fitting

Now we will be using the function called fit_resamples() to fit our respective models which will tell us which model is the best model ,this will be achieved by looking at the roc, accuracy, sesnsitivity and specificity.

We will begin with using fit_resamples() to fit the lda model

```r
spotify_lda_rs <-fit_resamples(object = spotify_lda,

preprocessor = recipe(playlist_genre ~ . , data = spotify_trained_pre),
resamples = spotify_cv)
```

```
## Warning: package 'MASS' was built under R version 4.3.2
```

```r
spotify_lda_rs %>%
  collect_metrics()
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy multiclass 0.476     5 0.00655 Preprocessor1_Model1
## 2 roc_auc  hand_till  0.798     5 0.00367 Preprocessor1_Model1
```

KNN Model

we will use the same process in knn model as well

```r
spotify_knn_rs <- fit_resamples(object = spotify_knn_final,
preprocessor =recipe(playlist_genre ~ . , data = spotify_trained_pre),
resamples = spotify_cv)

spotify_knn_rs %>%
  collect_metrics()
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy multiclass 0.495     5 0.00247 Preprocessor1_Model1
## 2 roc_auc  hand_till  0.812     5 0.00247 Preprocessor1_Model1
```

Random forest model

```r
spotify_rf_rs <- fit_resamples(object = spotify_rf_final,preprocessor =recipe(playlist_genre ~ . , data
resamples = spotify_cv)

spotify_rf_rs %>%
  collect_metrics()
```

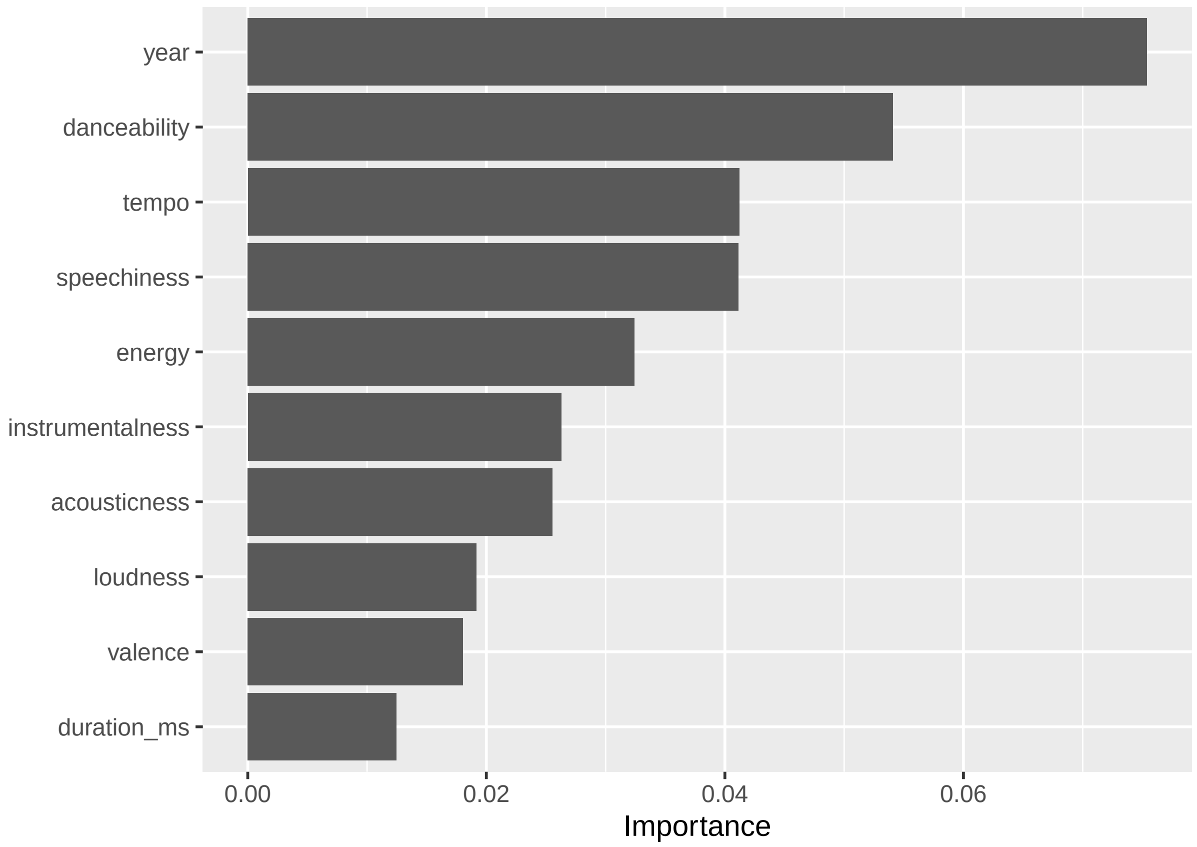
```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy multiclass 0.555     5 0.00247 Preprocessor1_Model1
## 2 roc_auc  hand_till  0.847     5 0.00343 Preprocessor1_Model1
```

From the above analysis it is very clear and evident that Random Forest model is the best model when we compare the "accuracy" and "roc_accuracy" of all three models

```r
set.seed(1899824)
spotify_rf_fit <- spotify_rf_final %>%
  fit(playlist_genre ~ . , data = spotify_trained_pre)
```

We will now observe the variable importance plot (VIP)

```
spotify_rf_fit %>%
vip( )
```



## Predictions will be obtained from the test dataset

```
spotify_rf_preds <- predict(spotify_rf_fit,new_data = spotify_tested_pre) %>%

bind_cols(playlist_genre =

spotify_tested_pre$playlist_genre)

spotify_rf_preds
```

```
## # A tibble: 1,500 x 2
##    .pred_class playlist_genre
##    <fct>       <fct>
##  1 edm         edm
##  2 edm         edm
##  3 pop         edm
##  4 rock        edm
##  5 edm         edm
##  6 latin       edm
##  7 edm         edm
##  8 edm          edm
##  9 edm          edm
## 10 edm          edm
## # i 1,490 more rows
```

now we can easily obtain the confusion matrix

```
spotify_rf_preds %>%
conf_mat(truth = playlist_genre, estimate = .pred_class)
```

```
##            Truth
## Prediction edm latin pop r&b rap rock
##      edm   171    21  37   2  13    6
##      latin  21   121  24  23  31    5
##      pop    40    46 105  35  17   17
##      r&b    12    29  34 108  30   17
##      rap     9    48  22  50 154    1
##      rock    6    14  18  25   6  182
```

Now we will get all the metrics

```
categorical_metrics <- metric_set(sens, spec,
yardstick::accuracy,
yardstick::precision)

spotify_rf_preds %>%
categorical_metrics(
truth = playlist_genre,
estimate = .pred_class
)
```

```
## # A tibble: 4 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 sens      macro          0.565
## 2 spec      macro          0.912
## 3 accuracy  multiclass     0.561
## 4 precision macro          0.560
```