Data Taming Final Report

Thi Kim Oanh Nguyen - a1879781

2023-11-24

Executive Summary

Spotify, being the leading music streaming service globally, aims to predict song genres to improve user experiences and refine playlist recommendations.

During the exploratory data analysis, it became evident that analyzing factors such as release year, speechiness, danceability, and tempo contributes to predicting the playlist genre. Moreover, we observed that song popularity varies among genres and found distinct differences in speechiness across each genre. Interestingly, we noticed a declining trend in song popularity from 1970, followed by a resurgence starting around 2010.

In our modeling approach, we tested three models: Linear Discriminant Analysis (LDA), K-nearest Neighbors (KNN), and Random Forest. Our prediction relied on variables like song popularity, danceability, energy, key, mode, loudness, speechiness, acousticness, instrumentalness, liveness, valence, tempo, song duration, and release year. We omitted all categorical variables and retained all numerical variables.

Upon fine-tuning the hyperparameters for KNN within a range of 1 to 100 and 20 levels, and Random Forest models with 100 trees and 5 levels, we determined the best KNN model with a neighbor value of 100, and the optimal Random Forest model with mtry set at 4 and min_n at 40. Through cross-validation testing, the Random Forest model exhibited better performance compared to LDA and KNN, with the highest accuracy and ROC_AUC scores. However, during testing, the Random Forest model only achieved an accuracy of 56.933%.

In summary, the Random Forest model fails to accurately predict song genres, contradicting the founders' objectives. Therefore, further investigation is considered.

Methods

The dataset comprises songs from diverse Spotify playlists, encompassing 32,833 observations and 23 variables. Among these variables, there are 9 categorical and 13 numeric ones. These attributes encapsulate a wide spectrum of song features, including unique identifiers and song-specific details, album-related information such as release dates, playlist specifics encompassing genre and subgenre classifications, and musical attributes like danceability, energy, pitch, loudness, and mode, indicators for speechiness, acoustic nature, instrumental presence, along with parameters indicating liveness, positivity, tempo, and song duration. The dataset includes an outcome variable representing 6 distinct genres.

The steps taken are explained as follows:

- Data cleaning: We extracted the year of each song's release from the track album release date and converted it into a single numerical feature for predicting song genres. Unnecessary categorical features such as track IDs, album IDs, playlist IDs, track names, and album names were removed because they are unique identifiers or text fields that do not carry meaningful information about the song's musical characteristics or genre classification. We also handled missing values, ensuring data integrity.

- Data sampling: Due to computing limitations, the dataset was reduced to 6000 observations, with 1000 observations per genre.
- Data splitting: A seed value of 1879781 was set for reproducibility, and the sample dataset was divided into training and testing sets.
- Data preprocessing: Zero-variance predictors were removed, and all predictors were standardized to possess a mean of 0 and a standard deviation of 1. Highly correlated predictors were identified and eliminated. This preprocessing recipe was applied to the training set and testing set.
- Model specification: Specifications for three models (Linear Discriminant Analysis (LDA), K-nearest Neighbors (KNN), and Random Forest) were defined. The mode was set to classification, and engine configurations were made. For the KNN model, the number of neighbors was defined as a tuneable hyperparameter. The Random Forest model set the number of trees to 100 and tuned the number of variables randomly sampled at each split (mtry) and minimum node size (min_n).
- Model tuning (KNN and Random Forest): 5 bootstrapped samples were generated from the preprocessed training data. For KNN, we employ a grid of 20 levels ranging from 1 to 100 for the neighbors' parameter. For Random Forest, a grid of 5 levels was created for tuning hyperparameters (mtry and min_n). Model selection utilized the ROC_AUC metric. The best KNN model had a neighbor value of 100, achieving an ROC_AUC score of 80.12% (Table 7). Meanwhile, the optimal Random Forest model had mtry set at 4 and min_n at 40, achieving an ROC_AUC score of 84.13% (Table 8).

In this research, R version 4.3.2 and Rmarkdown in RStudio were used. The primary packages included dplyr, tidyr, lubridate, skimr, inspectdf, rsample, stringr, knitr, tidymodels, janitor, pROC, discrim, yardstick, vip, caret. Additionally, parallel processing capabilities were leveraged to potentially speed up computations.

Results

- Exploratory data analysis:

After excluding certain categorical variables, we still have song artists, playlist names and subgenre of playlists. We chose to exclude song artists and playlist names. The primary objective is to predict the playlist genre based on song attributes. While the artist's name or specific playlist names may contribute to the uniqueness of a song or playlist, they might not directly influence the prediction of the genre itself. Additionally, artist names and individual playlist names might introduce too much granularity or noise into the analysis. We also excluded the subgenre of the playlist. If we were to include the subgenre variable in the feature list, we could directly predict the playlist genre. Therefore, utilizing other variables for prediction did not seem meaningful in this scenario.

Figures 1 to 14 display the relationship between each numerical variable and the outcome variable (playlist genre). Consequently, all numerical variables were included in the feature list.

- Founders' questions:

Relationship of speechiness and the playlist genre (Figure 14): Shape: all are right-skewed and unimodal. Location: rap has the highest speechiness median, while rock has the lowest speechines median. Spread: rap has the highest IQR, while rock has the lowest. Outliers: there are potential outliers in all genres. Therefore, there is a difference in speechiness for each genre.

Relationship of danceability and the playlist genre (Figure 11): Shape: all are slightly left-skewed and unimodal. Location: rap has the highest danceability median, while rock is the lowest danceability median. Spread: rap has the highest IQR, while rock has the lowest. Outliers: there are potential outliers in all genres.

Relationship of tempo and the playlist genre (Figure 12): Shape: all are slightly right-skewed and unimodal. Location: rock and EDM almost have the highest median, while R&B has the lowest median. Spread: rap has the highest IQR, while edm has the lowest. Outliers: there are potential outliers in almost genres exception rap.

The popularity of songs differ between genres: Figure 13 demonstrates genre-specific variations in song popularity, with Pop being the most prevalent, while EDM shows the least popularity.

Relationship of the release year and the playlist genre and the change of track popularity over time: Figure 9 shows that there is a relationship between the release year of songs and the outcome variable. Figure 15 indicates a decline in overall popularity from 1970, followed by a resurgence after 2010. Early on, Rock dominated, but later, Pop and EDM gained traction, while Rap consistently maintained popularity.

- Model selection:

The model exhibiting the best cross-validation results should ideally perform well on our test set. During cross-validation, LDA achieved an accuracy of 47.778% and an ROC_AUC of 80.202%, while KNN displayed an accuracy of 49.644% and ROC_AUC of 81.424%, and Random Forest showcased an accuracy of 55.222% and an ROC_AUC of 84.695%. Therefore, Random Forest is chosen as the best performer due to its highest accuracy and ROC_AUC scores.

- Model evaluation:

We employed the chosen model from Model selection and generated predictions using the preprocessed test dataset. For performance metrics, we computed the accuracy score and the sensitivity and specificity for each genre. Additionally, we employed ROC curves to assess the model's performance.

It appears that "Year" stands out as the most influential variable in predicting playlist genres, closely followed by danceability. Speechiness, tempo, and energy hold relatively similar importance, followed by other factors.

Table 10 shows that the model a low score of accuracy with 56.933%. However, from Table 9 and Figure 16, the model exhibits strong specificity for all genres, indicating its accuracy in predicting songs outside specific genres. Yet, their sensitivity seem relatively lower, posing challenges in precisely identifying a song that belongs to a genre. Both EDM and rock display commendable performance in sensitivity (both 76.4%) and specificity (93.68% and 95.2%). However, despite their high specificity, Pop, R&B, Latin, and Rap show notably lower sensitivity.

Discussion

We sampled 600 observations randomly, with 100 observations per genre from the original data. These observations were used for prediction after preprocessing.

Overall, the model demonstrates high specificity scores, signifying its robustness in predicting songs that don't belong to particular genres. However, from Table 11 and Figure 17, accuracy (61.167%) and sensitivity appears relatively low, making accurate predictions of a song that belongs to a genre quite challenging. Rock and EDM show good performance in both sensitivity (79% and 82%) and specificity (95% and 95.2%). Conversely, despite achieving high specificity scores, Pop, R&B, Latin, and Rap exhibit significantly lower sensitivity scores.

Thus, this model struggles to accurately predict genres, especially for pop, R&B, Latin, and rap songs.

Conclusion

Spotify, a leading global music streaming service, aims to enhance user experiences and refine playlist recommendations by predicting song genres.

During our exploratory data analysis, we discovered that factors like release year, speechiness, danceability, and tempo contribute significantly to predicting playlist genres. We also noticed variations in song popularity across different genres, particularly observing distinct differences in speechiness among them. Interestingly, there's a noticeable decline in song popularity from the 1970s, followed by a resurgence around 2010.

In our modeling approach, we tested three models: Linear Discriminant Analysis (LDA), K-nearest Neighbors (KNN), and Random Forest. We used various variables such as song popularity, danceability, energy, key, loudness, and more. Categorical variables were excluded, retaining only numerical ones.

After fine-tuning hyperparameters for KNN and Random Forest models, the Random Forest model outperformed LDA and KNN in cross-validation tests, displaying the highest accuracy and ROC_AUC scores. However, during actual testing, the Random Forest model achieved only 56.933% accuracy. It performed well in predicting songs that do not belong to specific genres but struggled with identifying songs that do belong to a genre.

In summary, the Random Forest model's failure to accurately predict song genres contradicts the initial objectives set by Spotify. In our future work, we intend to experiment with larger samples, introduce additional features, and employ different models to achieve improved performance results.

Appendix

```
# libraries
pacman::p_load(dplyr, tidyr, lubridate, skimr, inspectdf, rsample, stringr, knitr,
               tidymodels, janitor, pROC, discrim, yardstick, vip, caret)
```

Data Preparation

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

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

```
spotify_songs
```

```
## # A tibble: 32,833 x 23
##    track_id               track_name track_artist track_popularity track_album_id
##    <chr>                  <chr>      <chr>                   <dbl> <chr>
##  1 6f807x0ima9a1j3VPbc7~ I Don't C~ Ed Sheeran                 66 2oCs0DGTsRO98~
```

```
##  2 0r7CVbZTWZgbTCYdfa2P~ Memories ~ Maroon 5                    67 63rPSO264uRjW~
##  3 1z1Hg7Vb0AhHDiEmnDE7~ All the T~ Zara Larsson                70 1HoSmj2eLcsrR~
##  4 75FpbthrwQmzHlBJLuGd~ Call You ~ The Chainsm~                60 1nqYsOef1yKKu~
##  5 1e8PAfcKUYoKkxPhrHqw~ Someone Y~ Lewis Capal~                69 7m7vv9wlQ4i0L~
##  6 7fvUMiyapMsRRxr07cU8~ Beautiful~ Ed Sheeran                  67 2yiy9cd2QktrN~
##  7 2OAylPUDDfwRGfe0lYql~ Never Rea~ Katy Perry                  62 7INHYSeusaFly~
##  8 6b1RNvAcJjQH73eZO4BL~ Post Malo~ Sam Feldt                   69 6703SRPsLkS4b~
##  9 7bF6tCO3gFb8INrEDcjN~ Tough Lov~ Avicii                      68 7CvAfGvq4RlIw~
## 10 1IXGILkPm0tOCNeq00kC~ If I Can'~ Shawn Mendes                67 4QxzbfSsVryEQ~
## # i 32,823 more rows
## # i 18 more variables: track_album_name <chr>, track_album_release_date <chr>,
## #   playlist_name <chr>, playlist_id <chr>, playlist_genre <chr>,
## #   playlist_subgenre <chr>, danceability <dbl>, energy <dbl>, key <dbl>,
## #   loudness <dbl>, mode <dbl>, speechiness <dbl>, acousticness <dbl>,
## #   instrumentalness <dbl>, liveness <dbl>, valence <dbl>, tempo <dbl>,
## #   duration_ms <dbl>
```

```r
skim(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	hist
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	

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
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	225799.81	59834.01	4000.00	187819.00	216000.00	253585.00	517810.00	

```r
# extract year from the 'date' column
spotify_songs <- spotify_songs %>%
  mutate(year = as.numeric(year(ymd(track_album_release_date)),
                          ordered = FALSE))
```

```
## Warning: There was 1 warning in `mutate()`.
## i In argument: `year = as.numeric(year(ymd(track_album_release_date)), ordered
##   = FALSE)`.
## Caused by warning:
## !  1886 failed to parse.
```

```r
# extract features
colnames(spotify_songs)
```

```
##  [1] "track_id"                 "track_name"
##  [3] "track_artist"             "track_popularity"
##  [5] "track_album_id"           "track_album_name"
##  [7] "track_album_release_date" "playlist_name"
##  [9] "playlist_id"              "playlist_genre"
## [11] "playlist_subgenre"        "danceability"
## [13] "energy"                   "key"
## [15] "loudness"                 "mode"
## [17] "speechiness"              "acousticness"
## [19] "instrumentalness"         "liveness"
## [21] "valence"                  "tempo"
## [23] "duration_ms"              "year"
```

```r
# remove unnecessary features
spotify_songs <- spotify_songs %>%
  dplyr::select(-track_id, -track_album_id, -track_name, -track_album_name,
                -playlist_id, -track_album_release_date)
colnames(spotify_songs)
```

```
##  [1] "track_artist"      "track_popularity"  "playlist_name"
##  [4] "playlist_genre"    "playlist_subgenre" "danceability"
##  [7] "energy"            "key"               "loudness"
## [10] "mode"              "speechiness"       "acousticness"
```

```
## [13] "instrumentalness"  "liveness"          "valence"
## [16] "tempo"             "duration_ms"       "year"
```

```
# handle missing values
spotify_songs <- spotify_songs %>% drop_na()
inspect_na(spotify_songs)
```

```
## # A tibble: 18 x 3
##    col_name             cnt  pcnt
##    <chr>              <int> <dbl>
##  1 track_artist           0     0
##  2 track_popularity       0     0
##  3 playlist_name          0     0
##  4 playlist_genre         0     0
##  5 playlist_subgenre      0     0
##  6 danceability           0     0
##  7 energy                 0     0
##  8 key                    0     0
##  9 loudness               0     0
## 10 mode                   0     0
## 11 speechiness            0     0
## 12 acousticness           0     0
## 13 instrumentalness       0     0
## 14 liveness               0     0
## 15 valence                0     0
## 16 tempo                  0     0
## 17 duration_ms            0     0
## 18 year                   0     0
```

```
# extract outcome variable
unique(spotify_songs$playlist_genre)
```

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

```
skim(spotify_songs)
```

Table 4: Data summary

Name	spotify_songs
Number of rows	30942
Number of columns	18
Column type frequency:	
character	4
numeric	14
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
track_artist	0	1	2	69	0	10316	0
playlist_name	0	1	6	120	0	449	0
playlist_genre	0	1	3	5	0	6	0
playlist_subgenre	0	1	4	25	0	24	0

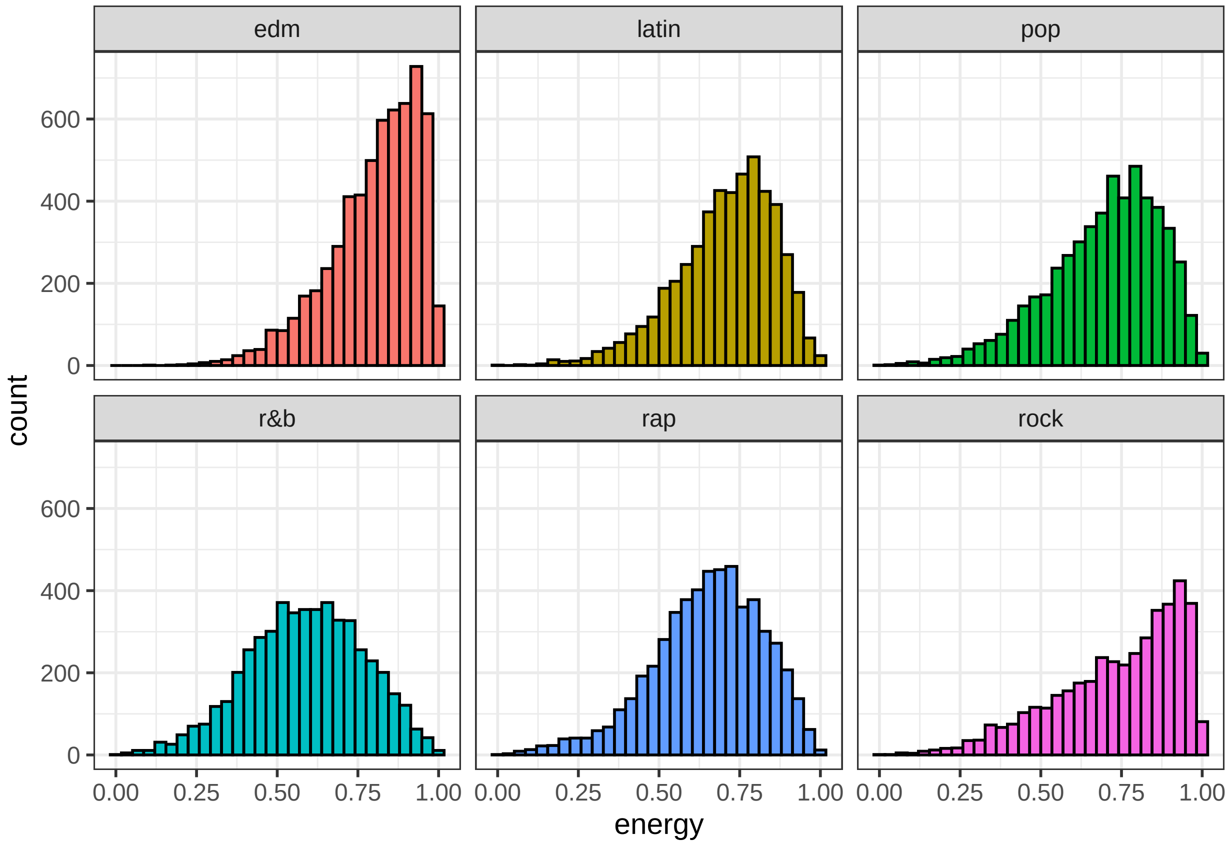
Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
track_popularity	0	1	42.76	24.95	0.00	25.00	45.00	62.00	100.00	
danceability	0	1	0.66	0.14	0.00	0.57	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	-6.64	2.95	-46.45	-8.07	-6.09	-4.61	1.27	
mode	0	1	0.56	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	0.09	0.23	0.00	0.00	0.00	0.01	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	120.94	26.85	0.00	99.97	122.00	133.52	239.44	
duration_ms	0	1	223946.64	59116.34	4000.00	186750.00	214400.00	251099.75	517810.00	
year	0	1	2012.20	10.40	1957.00	2010.00	2017.00	2019.00	2020.00	

Exploratory Data Analysis

```r
# energy
spotify_songs %>%
  ggplot(aes(x = energy, fill=playlist_genre)) +
  geom_histogram(colour="black", show.legend = FALSE) +
  facet_wrap(.~playlist_genre) +
  theme_bw()
```

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




```
spotify_songs %>%
  ggplot(aes(playlist_genre, energy, fill = playlist_genre)) +
           geom_boxplot() +
  scale_fill_brewer(palette = "Set3") +
  theme_bw() +
  theme(plot.caption = element_text(hjust = 0.5)) +
  labs(caption = "Fig.1: Boxplots of energy against playlist genre",
       x = "Playlist Genre",
       y = "Energy",
       fill = "Playlist Genre")
```

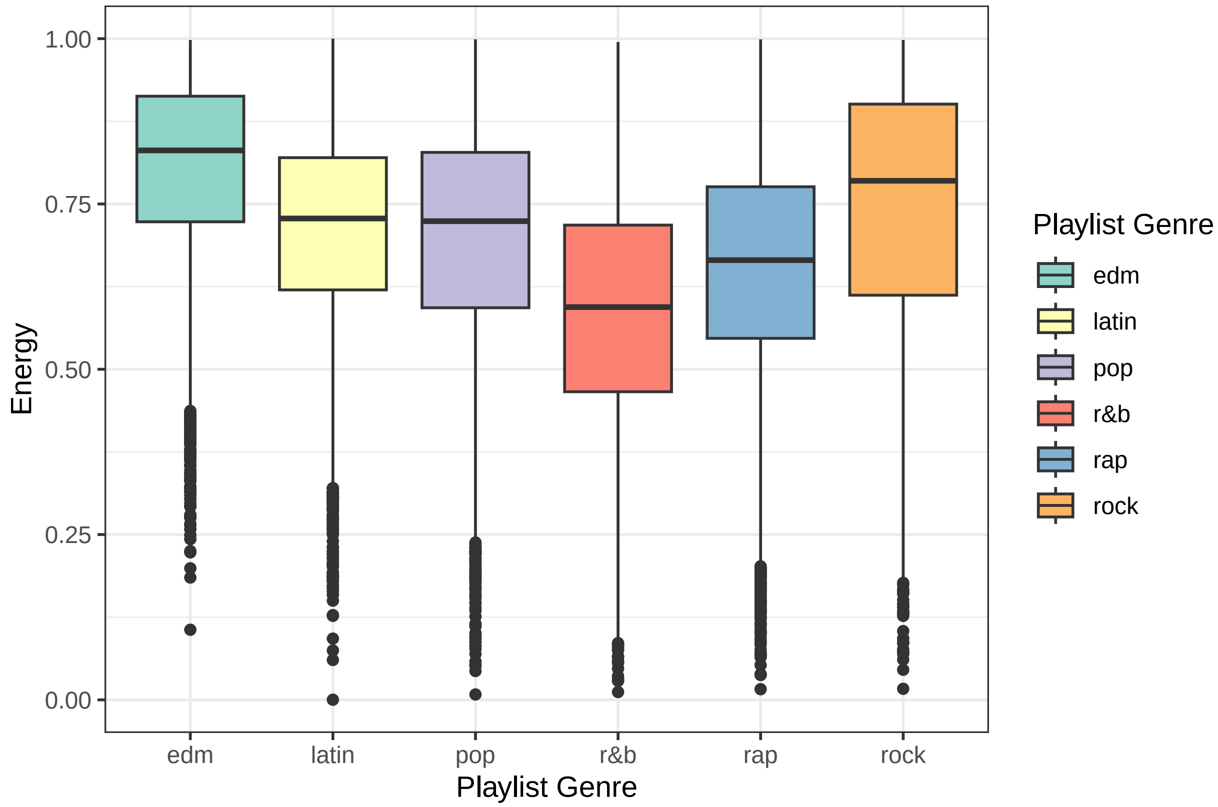
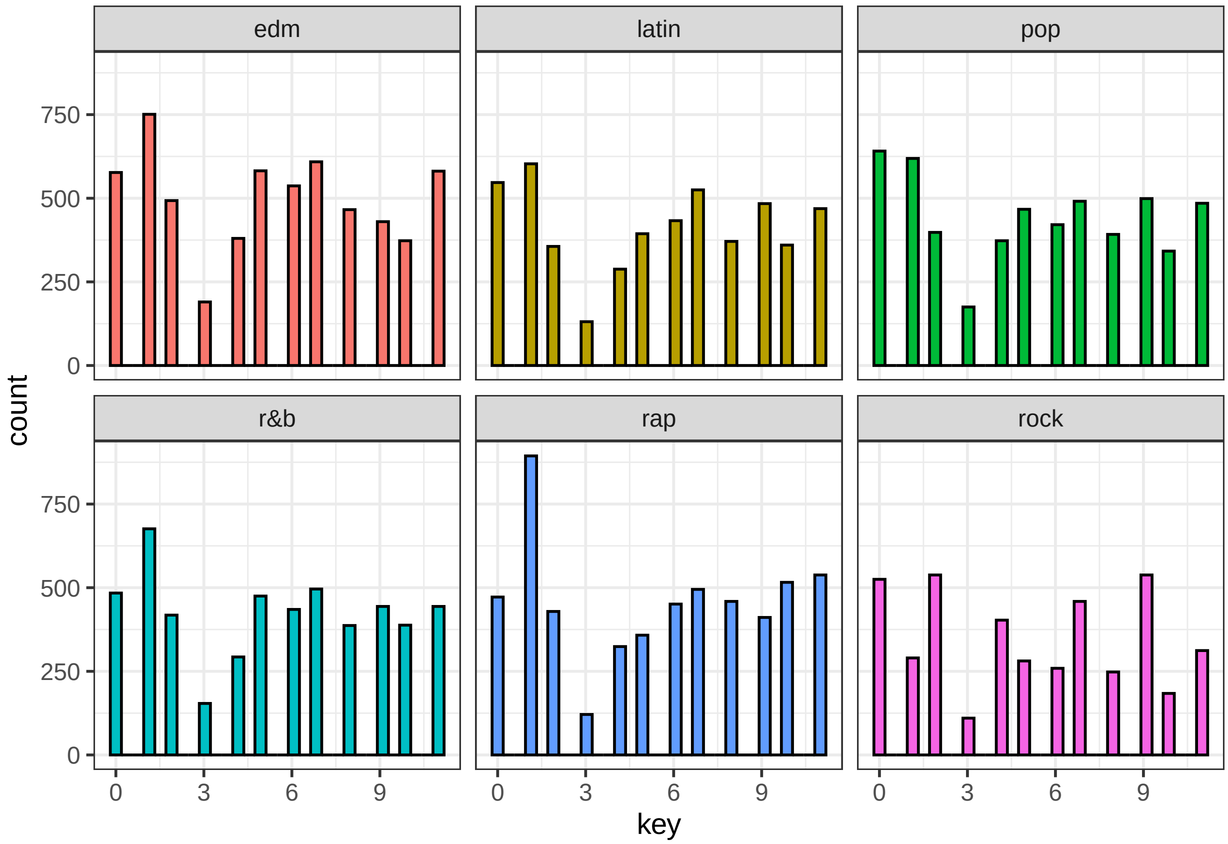


Fig.1: Boxplots of energy against playlist genre

```
# key
spotify_songs %>%
  ggplot(aes(x = key, fill=playlist_genre)) +
  geom_histogram(colour="black", show.legend = FALSE) +
  facet_wrap(.~playlist_genre) +
  theme_bw()
```

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




```r
spotify_songs %>%
  ggplot(aes(playlist_genre, key, fill = playlist_genre)) +
           geom_boxplot() +
  scale_fill_brewer(palette = "Set3") +
  theme_bw() +
  theme(plot.caption = element_text(hjust = 0.5)) +
  labs(caption = "Fig.2: Boxplots of key against playlist genre",
       x = "Playlist Genre",
       y = "Key",
       fill = "Playlist Genre")
```

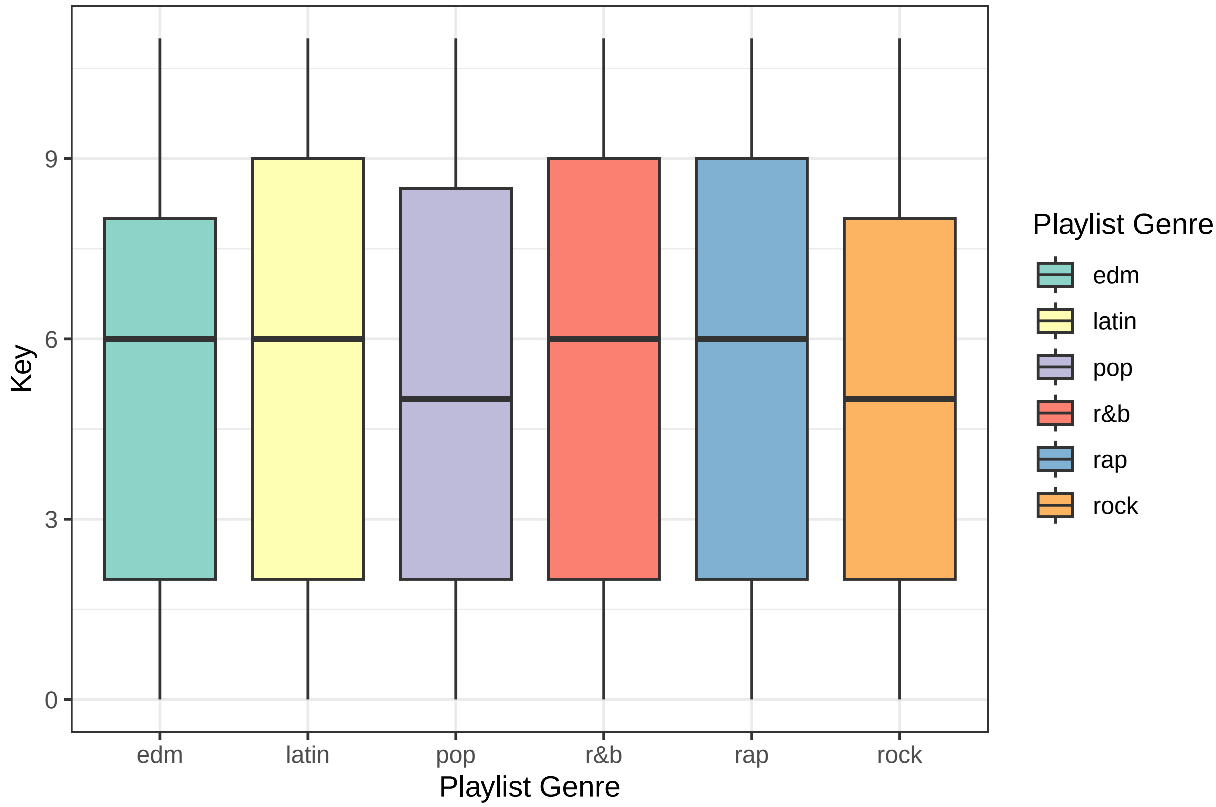
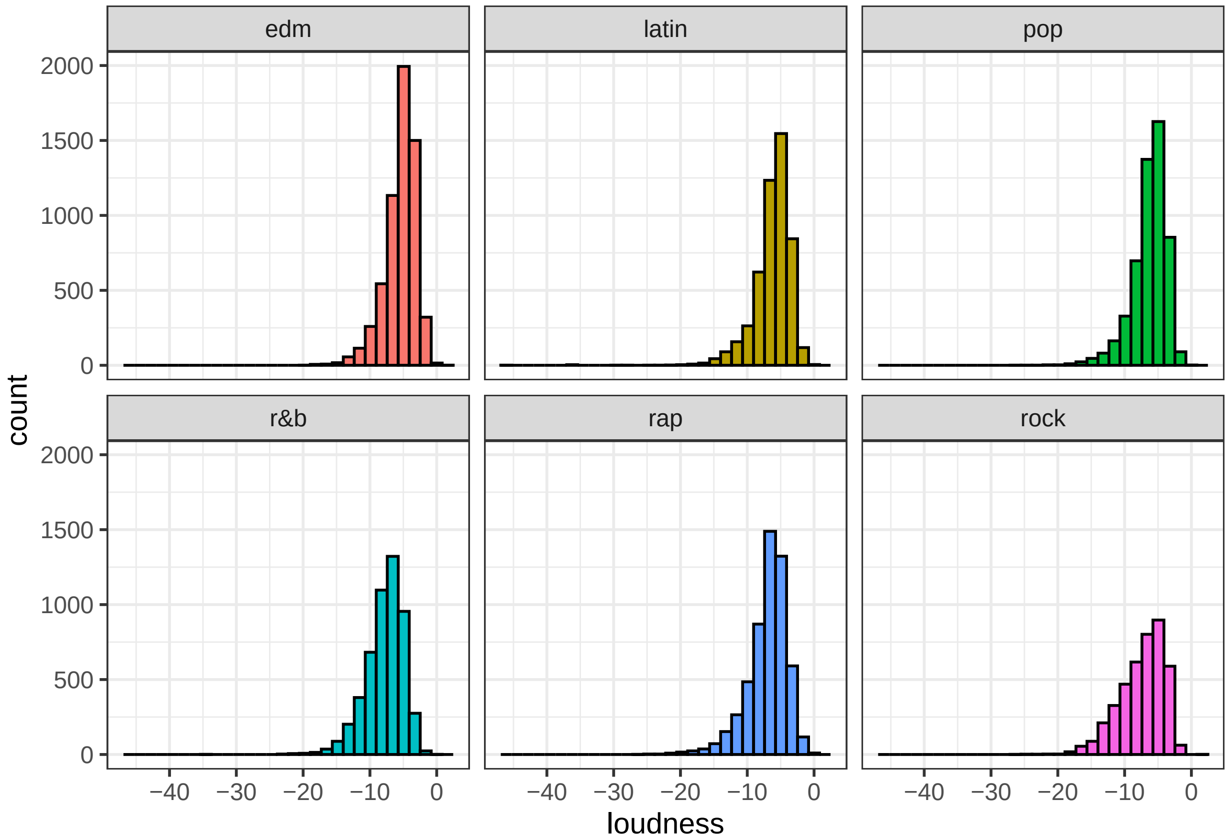


Fig.2: Boxplots of key against playlist genre

```
# loudness
spotify_songs %>%
  ggplot(aes(x = loudness, fill=playlist_genre)) +
  geom_histogram(colour="black", show.legend = FALSE) +
  facet_wrap(.~playlist_genre) +
  theme_bw()
```

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




```r
spotify_songs %>%
  ggplot(aes(playlist_genre, loudness, fill = playlist_genre)) +
         geom_boxplot() +
  scale_fill_brewer(palette = "Set3") +
  theme_bw() +
  theme(plot.caption = element_text(hjust = 0.5)) +
  labs(caption = "Fig.3: Boxplots of loudness against playlist genre",
       x = "Playlist Genre",
       y = "Loudness",
       fill = "Playlist Genre")
```

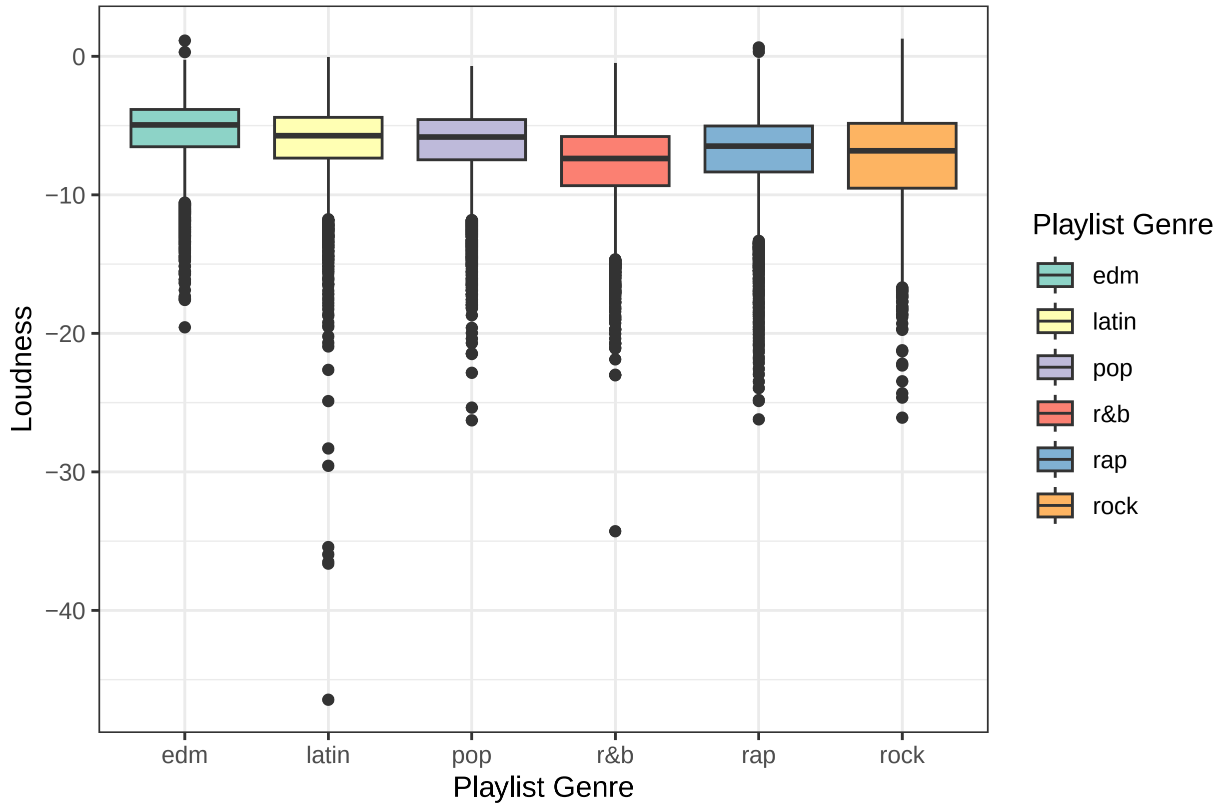
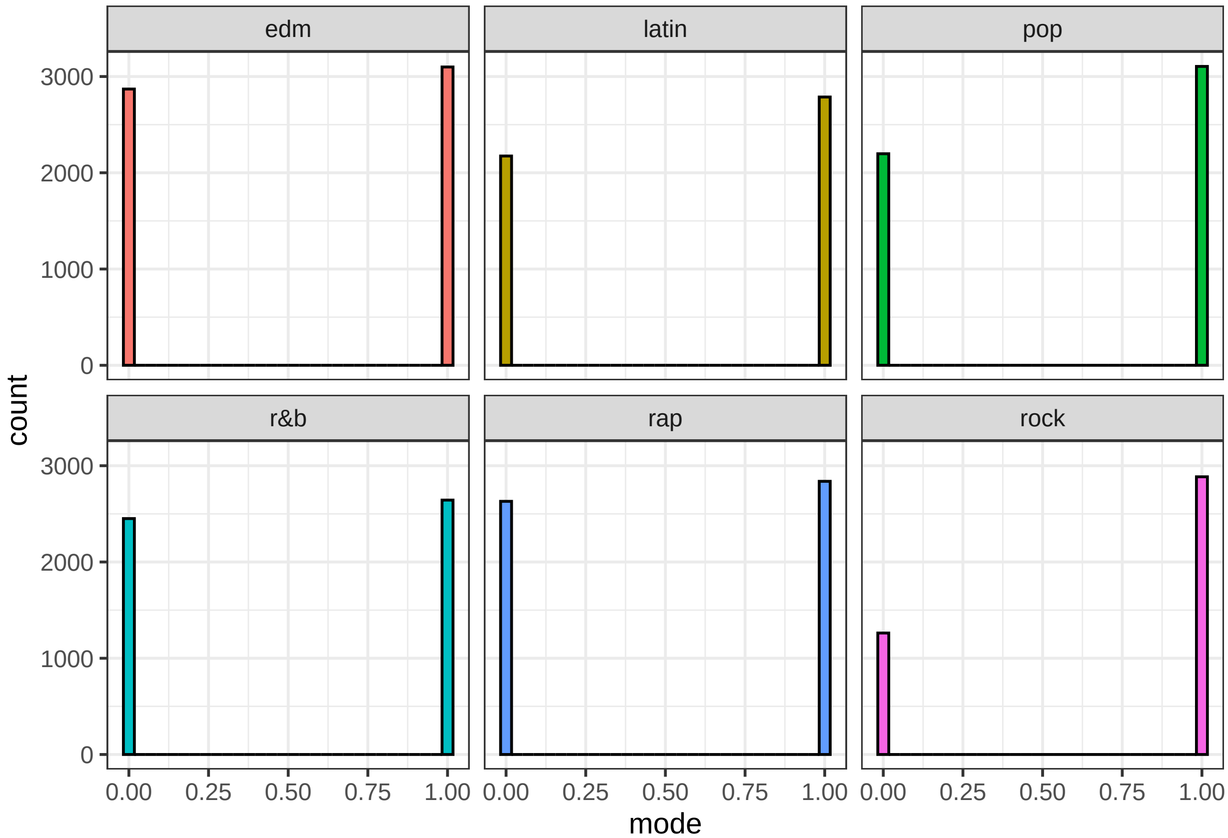


Fig.3: Boxplots of loudness against playlist genre

```
# mode
spotify_songs %>%
  ggplot(aes(x = mode, fill=playlist_genre)) +
  geom_histogram(colour="black", show.legend = FALSE) +
  facet_wrap(.~playlist_genre) +
  theme_bw()
```

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




```r
spotify_songs %>%
  ggplot(aes(playlist_genre, mode, fill = playlist_genre)) +
           geom_boxplot() +
  scale_fill_brewer(palette = "Set3") +
  theme_bw() +
  theme(plot.caption = element_text(hjust = 0.5)) +
  labs(caption = "Fig.4: Boxplots of mode against playlist genre",
       x = "Playlist Genre",
       y = "Mode",
       fill = "Playlist Genre")
```

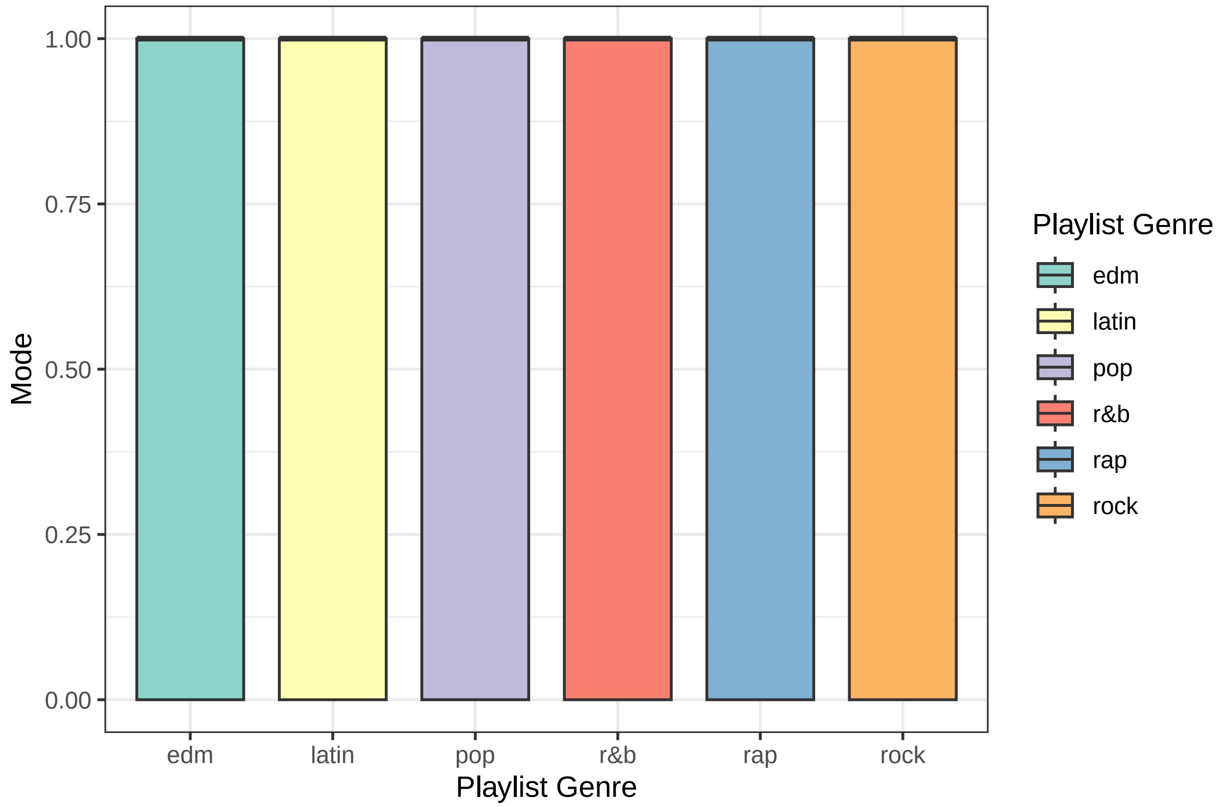
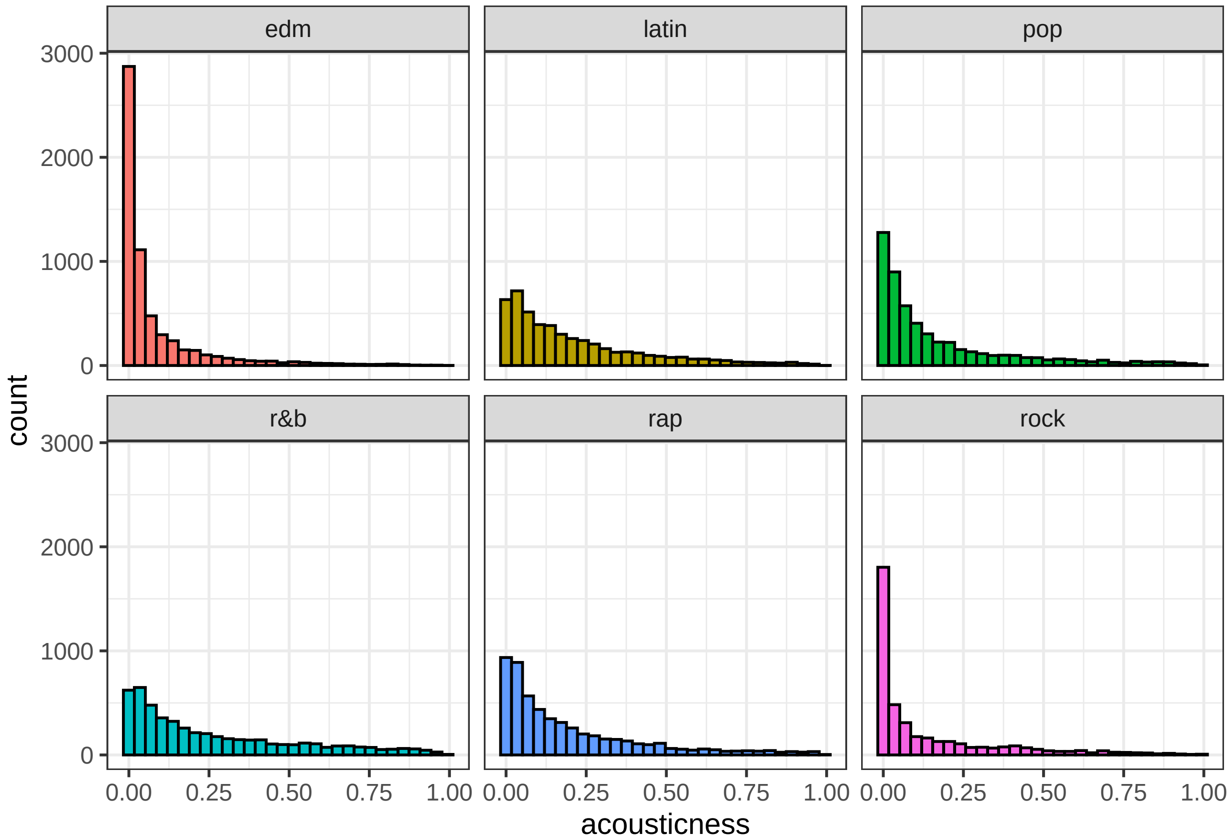


Fig.4: Boxplots of mode against playlist genre

```r
# acousticness
spotify_songs %>%
  ggplot(aes(x = acousticness, fill=playlist_genre)) +
  geom_histogram(colour="black", show.legend = FALSE) +
  facet_wrap(.~playlist_genre) +
  theme_bw()
```

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




```r
spotify_songs %>%
  ggplot(aes(playlist_genre, acousticness, fill = playlist_genre)) +
         geom_boxplot() +
  scale_fill_brewer(palette = "Set3") +
  theme_bw() +
  theme(plot.caption = element_text(hjust = 0.5)) +
  labs(caption = "Fig.5: Boxplots of acousticness against playlist genre",
       x = "Playlist Genre",
       y = "Acousticness",
       fill = "Playlist Genre")
```

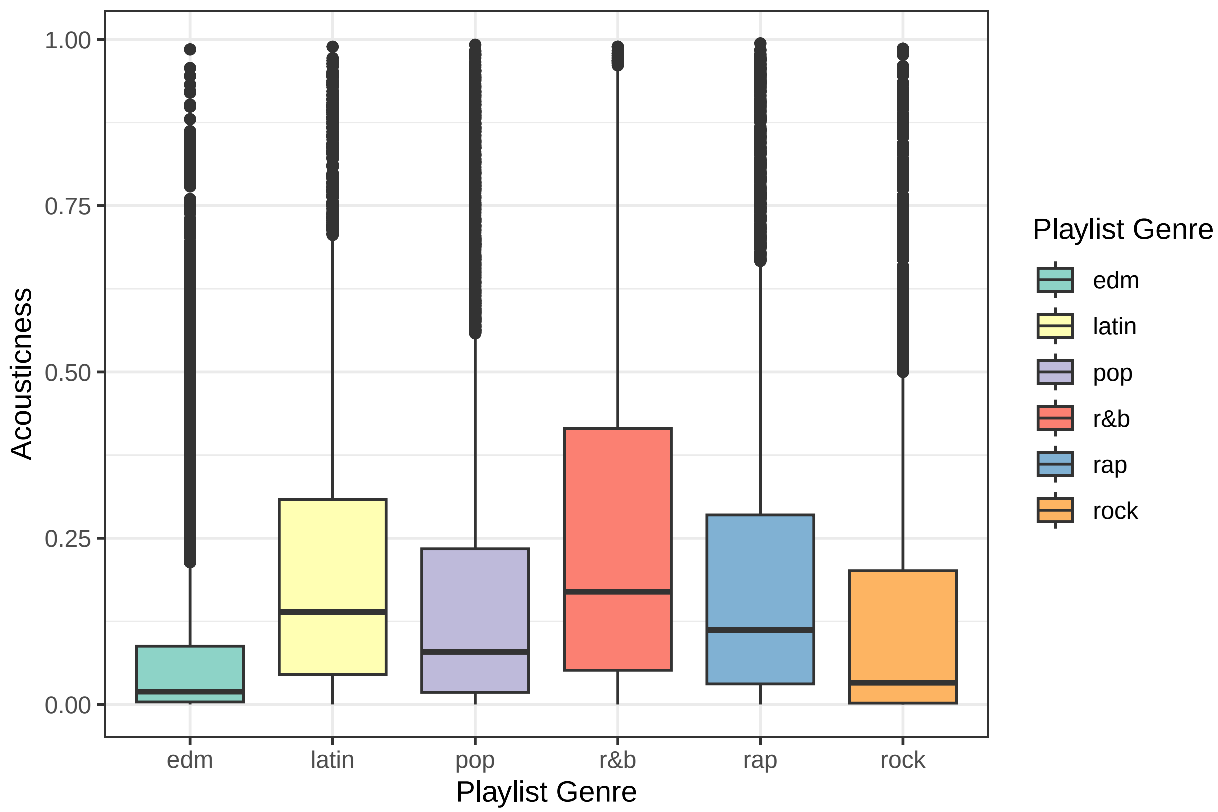
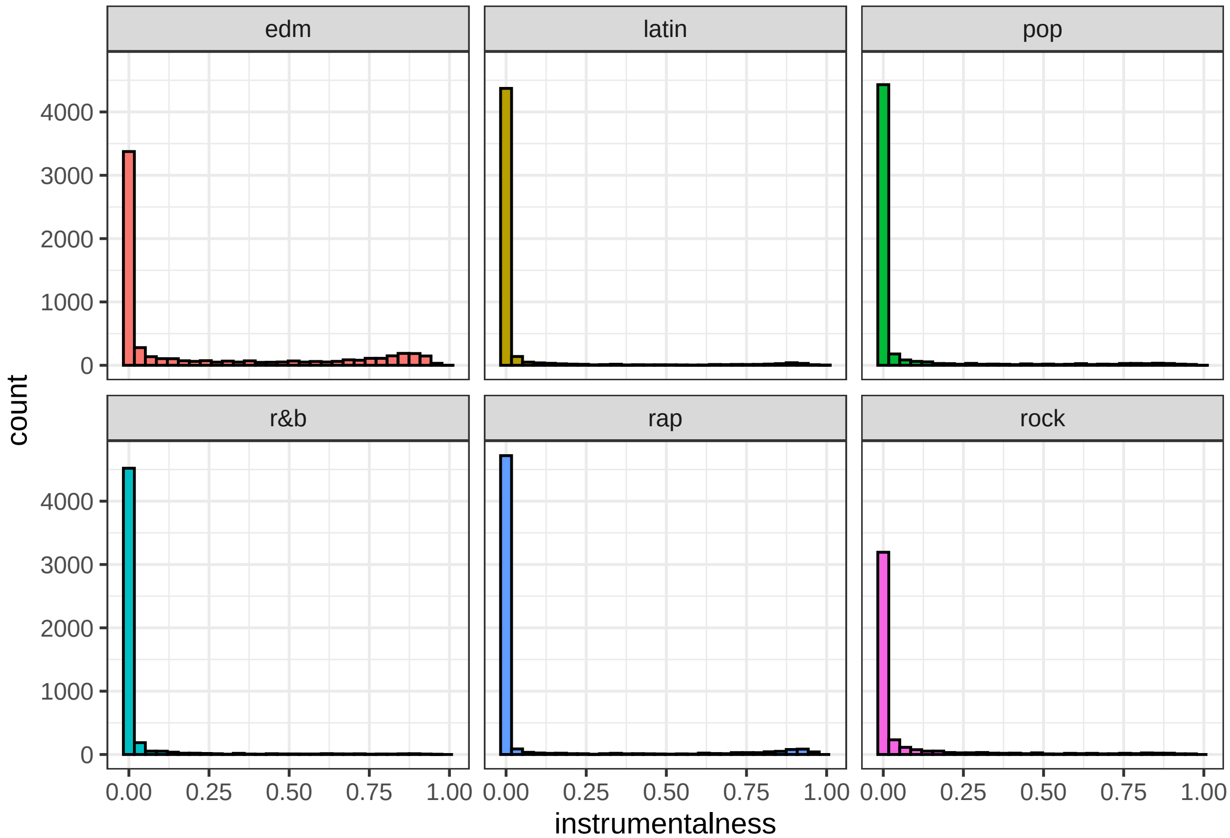


Fig.5: Boxplots of acousticness against playlist genre

```r
# instrumentalness
spotify_songs %>%
  ggplot(aes(x = instrumentalness, fill=playlist_genre)) +
  geom_histogram(colour="black", show.legend = FALSE) +
  facet_wrap(.~playlist_genre) +
  theme_bw()
```

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




```
spotify_songs %>%
  ggplot(aes(playlist_genre, instrumentalness, fill = playlist_genre)) +
         geom_boxplot() +
  scale_fill_brewer(palette = "Set3") +
  theme_bw() +
  theme(plot.caption = element_text(hjust = 0.5)) +
  labs(caption = "Fig.6: Boxplots of instrumentalness against playlist genre",
       x = "Playlist Genre",
       y = "Instrumentalness",
       fill = "Playlist Genre")
```

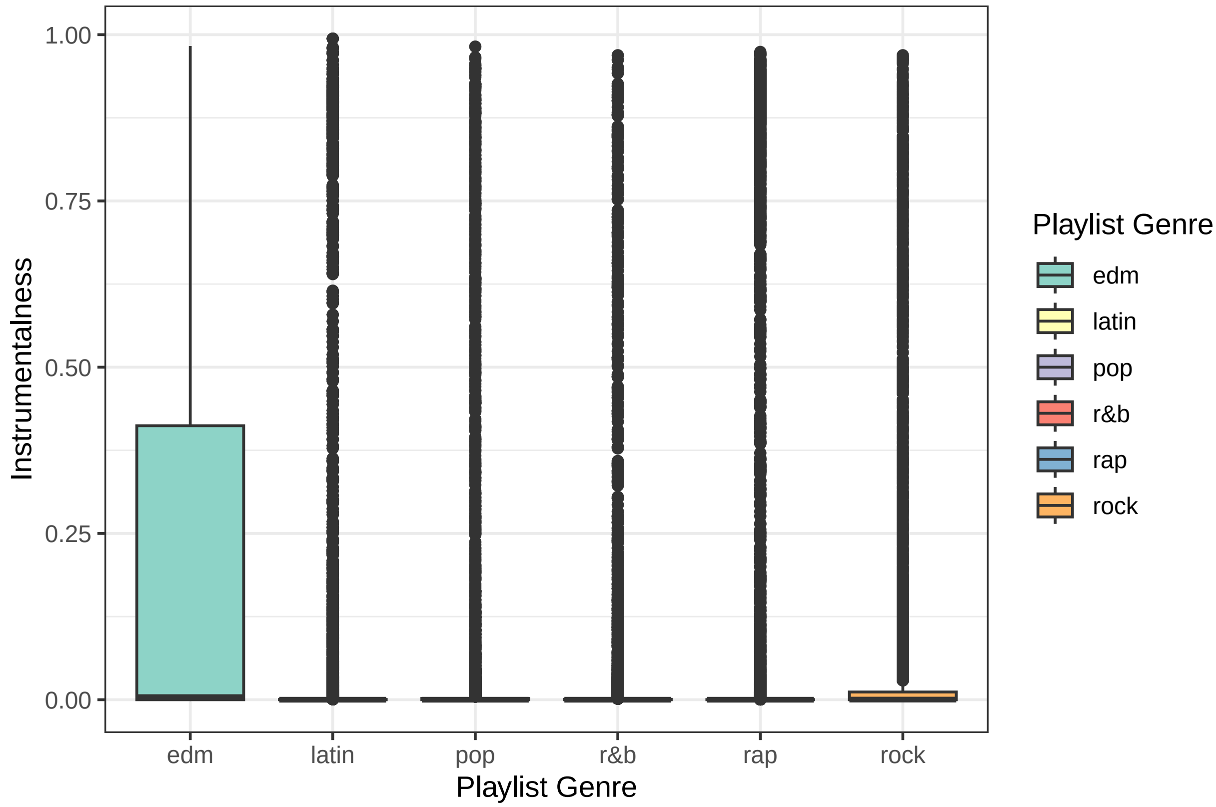
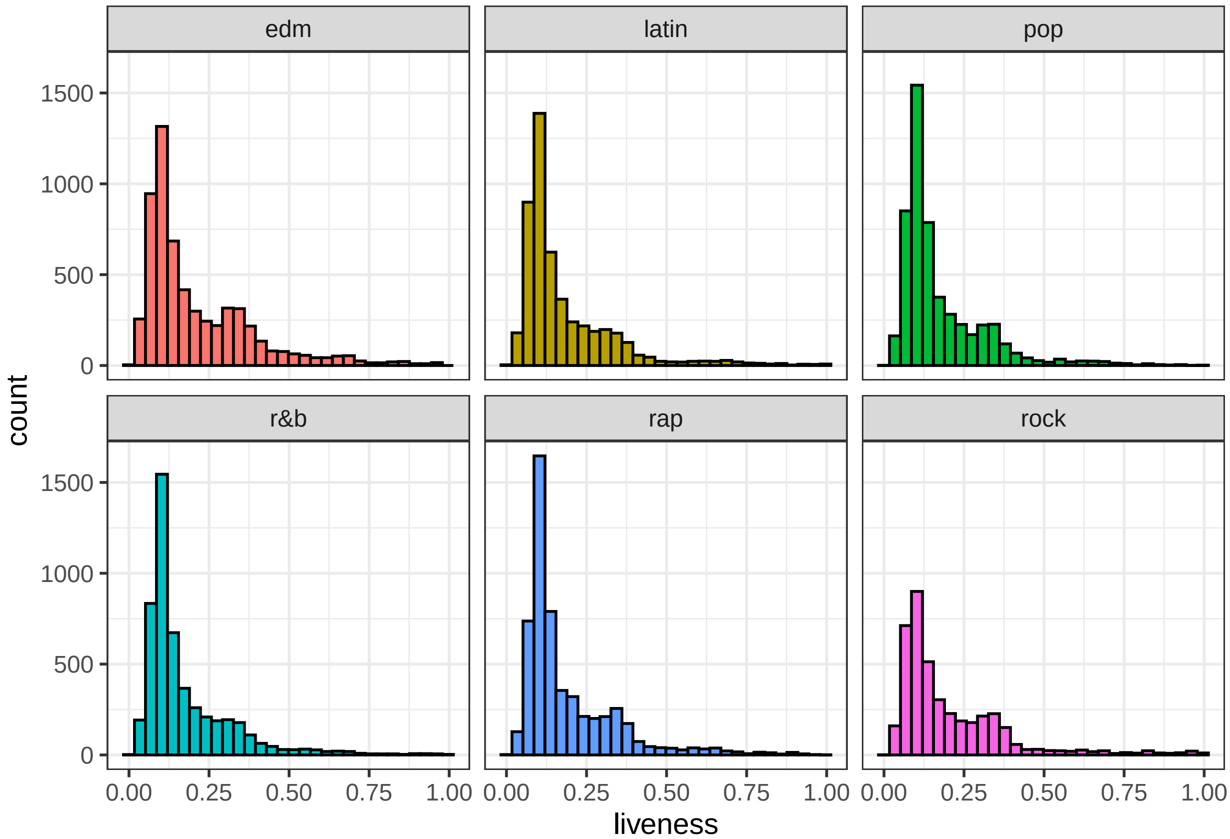


Fig.6: Boxplots of instrumentalness against playlist genre

```
# liveness
spotify_songs %>%
  ggplot(aes(x = liveness, fill=playlist_genre)) +
  geom_histogram(colour="black", show.legend = FALSE) +
  facet_wrap(.~playlist_genre) +
  theme_bw()
```

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




```
spotify_songs %>%
  ggplot(aes(playlist_genre, liveness, fill = playlist_genre)) +
           geom_boxplot() +
  scale_fill_brewer(palette = "Set3") +
  theme_bw() +
  theme(plot.caption = element_text(hjust = 0.5)) +
  labs(caption = "Fig.7: Boxplots of liveness against playlist genre",
       x = "Playlist Genre",
       y = "Liveness",
       fill = "Playlist Genre")
```

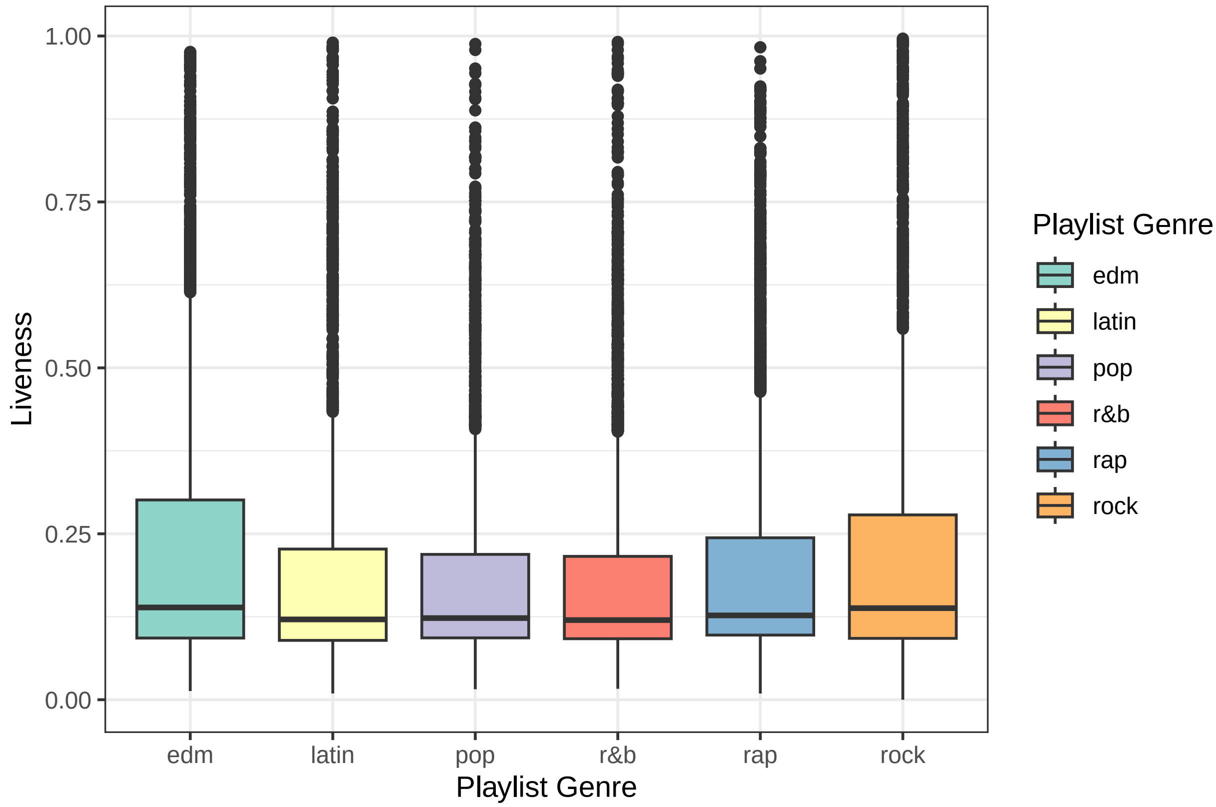
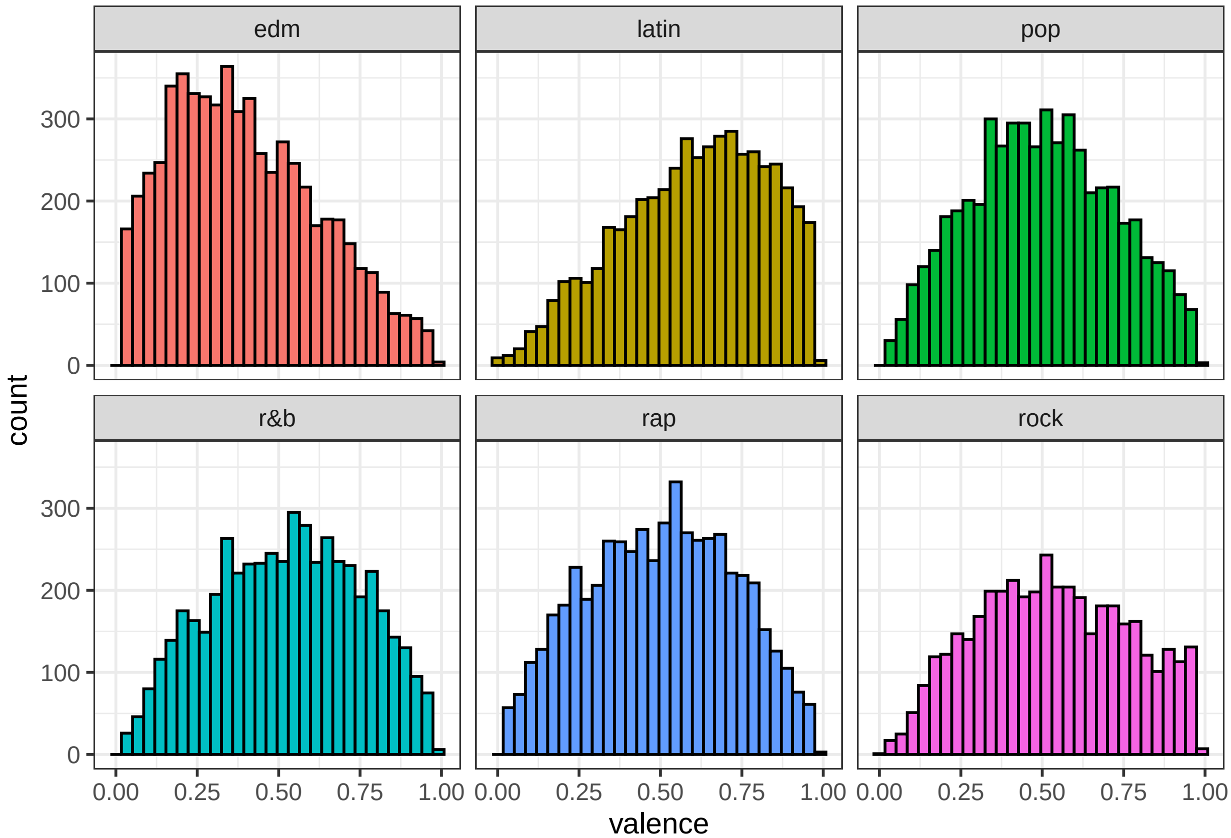


Fig.7: Boxplots of liveness against playlist genre

```
# valence
spotify_songs %>%
  ggplot(aes(x = valence, fill=playlist_genre)) +
  geom_histogram(colour="black", show.legend = FALSE) +
  facet_wrap(.~playlist_genre) +
  theme_bw()
```

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




```
spotify_songs %>%
  ggplot(aes(playlist_genre, valence, fill = playlist_genre)) +
          geom_boxplot() +
  scale_fill_brewer(palette = "Set3") +
  theme_bw() +
  theme(plot.caption = element_text(hjust = 0.5)) +
  labs(caption = "Fig.8: Boxplots of valence against playlist genre",
       x = "Playlist Genre",
       y = "Valence",
       fill = "Playlist Genre")
```

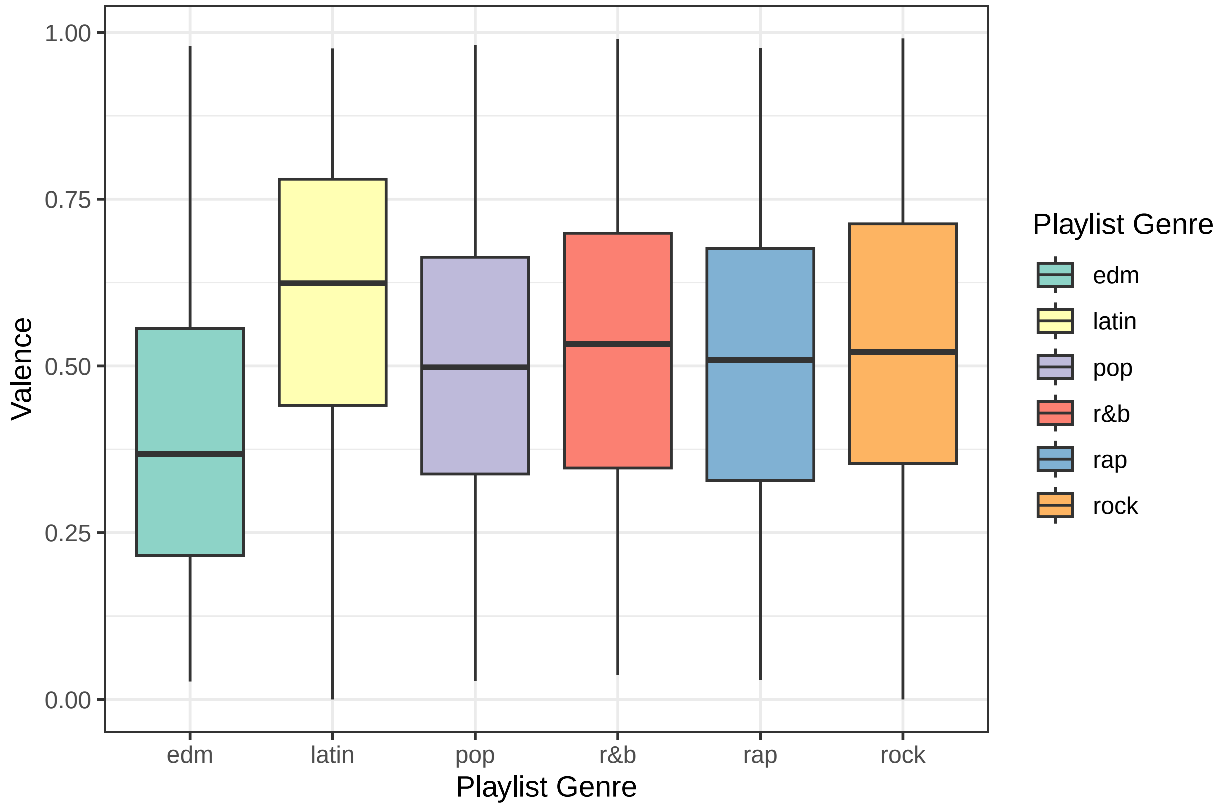
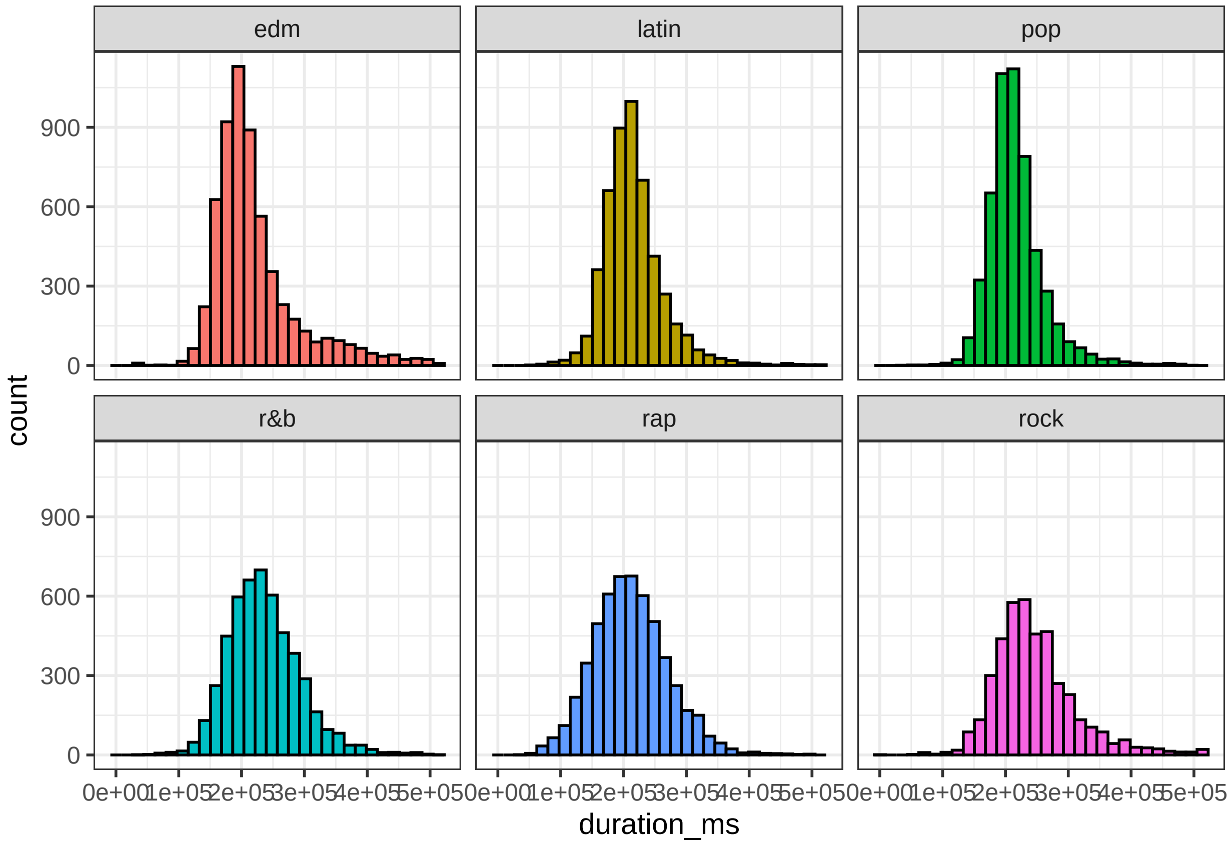


Fig.8: Boxplots of valence against playlist genre

```
# duration_ms
spotify_songs %>%
  ggplot(aes(x = duration_ms, fill=playlist_genre)) +
  geom_histogram(colour="black", show.legend = FALSE) +
  facet_wrap(.~playlist_genre) +
  theme_bw()
```

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




```r
spotify_songs %>%
  ggplot(aes(playlist_genre, duration_ms, fill = playlist_genre)) +
           geom_boxplot() +
  scale_fill_brewer(palette = "Set3") +
  theme_bw() +
  theme(plot.caption = element_text(hjust = 0.5)) +
  labs(caption = "Fig.9: Boxplots of duration (ms) against playlist genre",
       x = "Playlist Genre",
       y = "Duration",
       fill = "Playlist Genre")
```

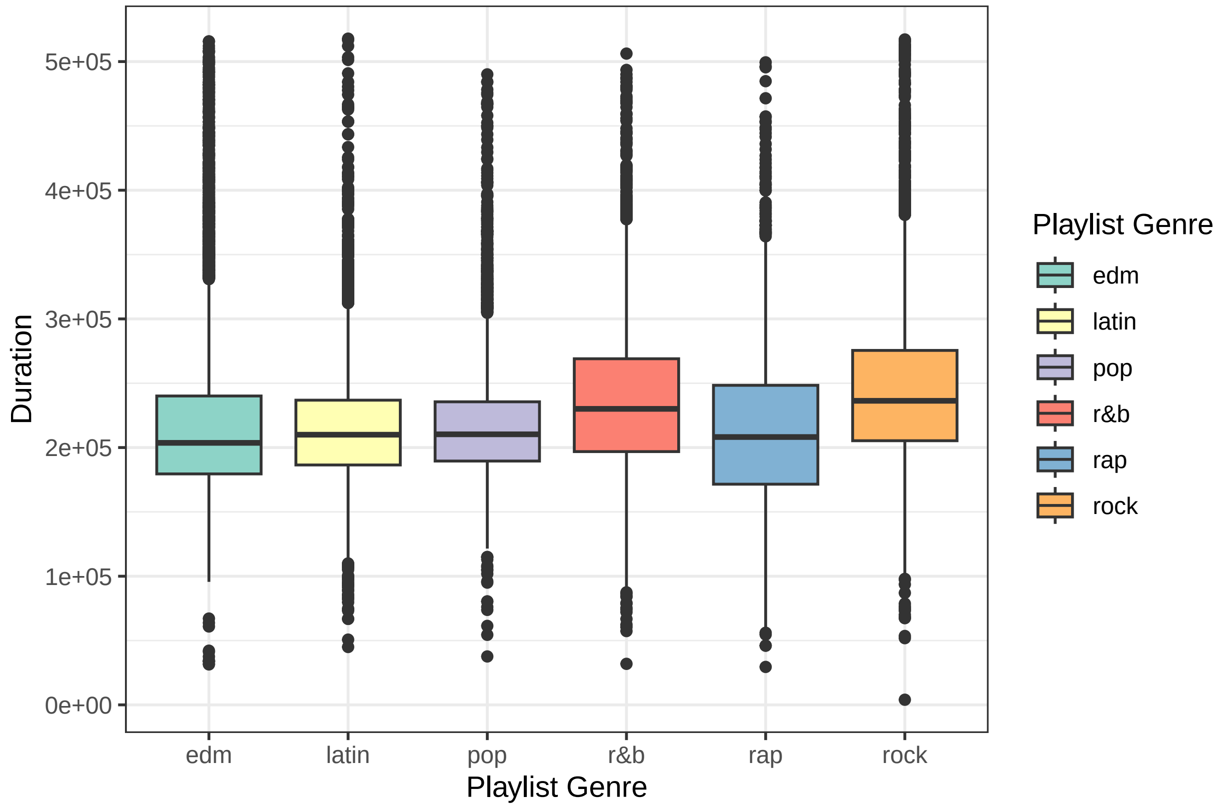


Fig.9: Boxplots of duration (ms) against playlist genre

The year the song was released can help predict a song's genre

```
# year
spotify_songs %>%
  ggplot(aes(x = playlist_genre, y = year, fill = playlist_genre)) +
  geom_boxplot() +
  scale_fill_brewer(palette = "Set3") +
  theme_bw() +
  theme(plot.caption = element_text(hjust = 0.5)) +
  labs(
    caption = "Fig.9: Boxplots of year against playlist genre",
    x = "Playlist Genre",
    y = "Year",
    fill = "Playlist Genre"
  )
```

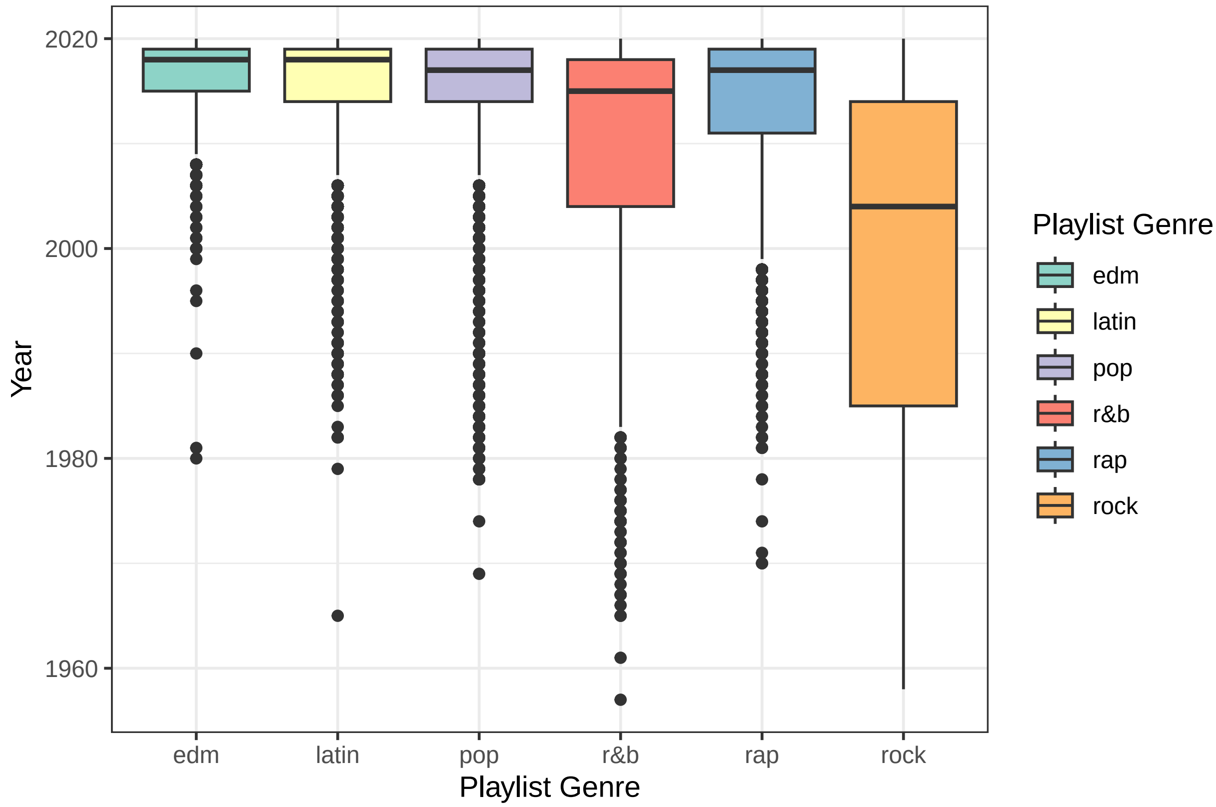
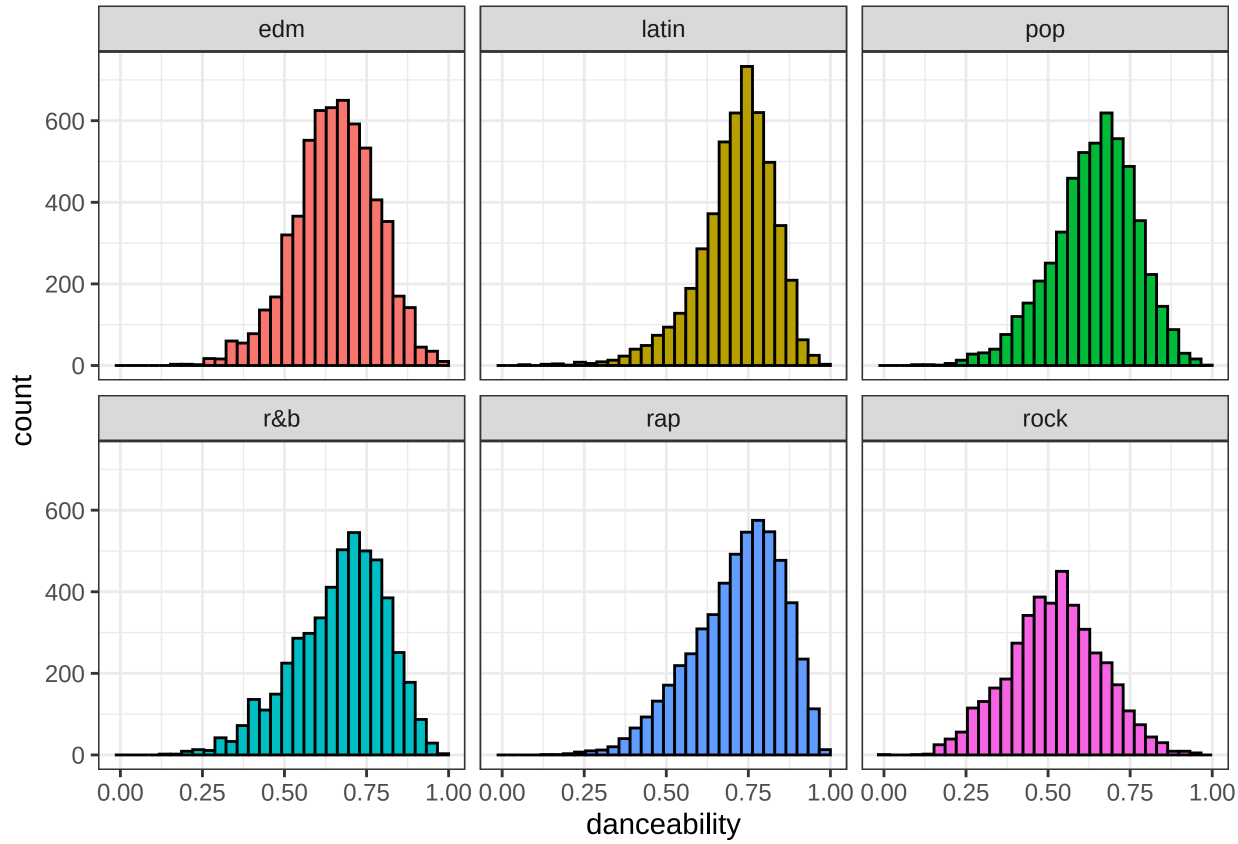


Fig.9: Boxplots of year against playlist genre

How danceable the song is can predict a song's genre

```
# danceability
spotify_songs %>%
  ggplot(aes(x = danceability, fill=playlist_genre)) +
  geom_histogram(colour="black", show.legend = FALSE) +
  facet_wrap(.~playlist_genre) +
  theme_bw()
```

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




```
spotify_songs %>%
  ggplot(aes(playlist_genre, danceability, fill = playlist_genre)) +
           geom_boxplot() +
  scale_fill_brewer(palette = "Set3") +
  theme_bw() +
  theme(plot.caption = element_text(hjust = 0.5)) +
  labs(caption = "Fig.11: Boxplots of danceability against playlist genre",
       x = "Playlist Genre",
       y = "Danceability",
       fill = "Playlist Genre")
```

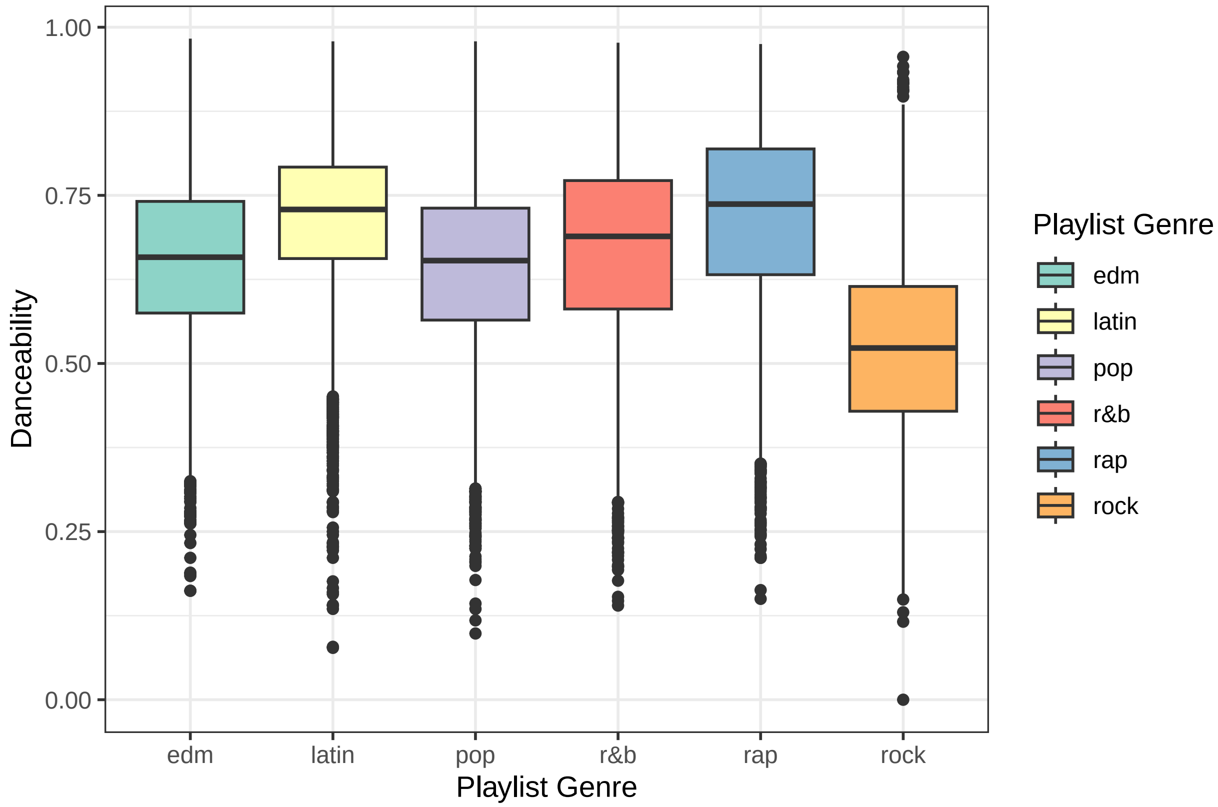
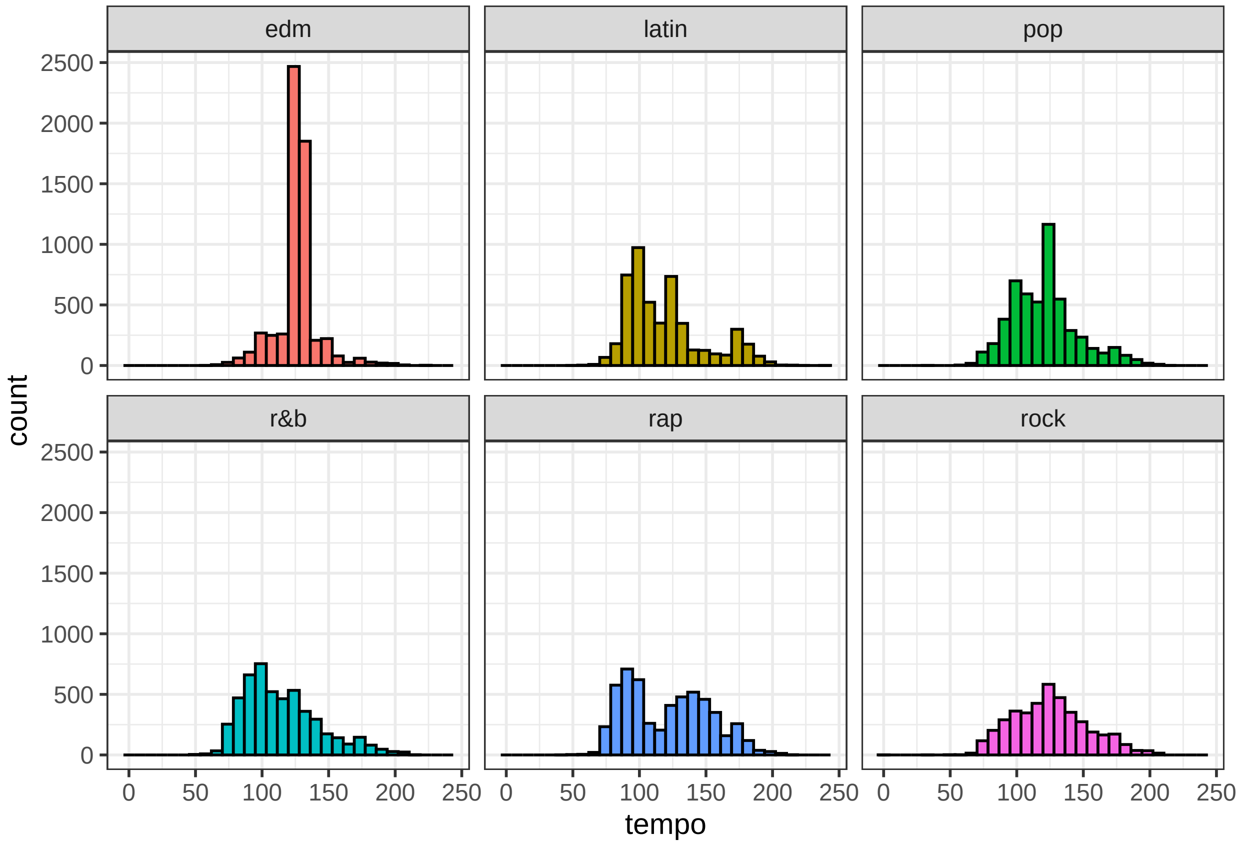


Fig.11: Boxplots of danceability against playlist genre

The tempo of the song can predict a song's genre

```r
# tempo
spotify_songs %>%
  ggplot(aes(x = tempo, fill=playlist_genre)) +
  geom_histogram(colour="black", show.legend = FALSE) +
  facet_wrap(.~playlist_genre) +
  theme_bw()
```

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




```r
spotify_songs %>%
  ggplot(aes(playlist_genre, tempo, fill = playlist_genre)) +
           geom_boxplot() +
  scale_fill_brewer(palette = "Set3") +
  theme_bw() +
  theme(plot.caption = element_text(hjust = 0.5)) +
  labs(caption = "Fig.12: Boxplots of tempo against playlist genre",
       x = "Playlist Genre",
       y = "Tempo",
       fill = "Playlist Genre")
```

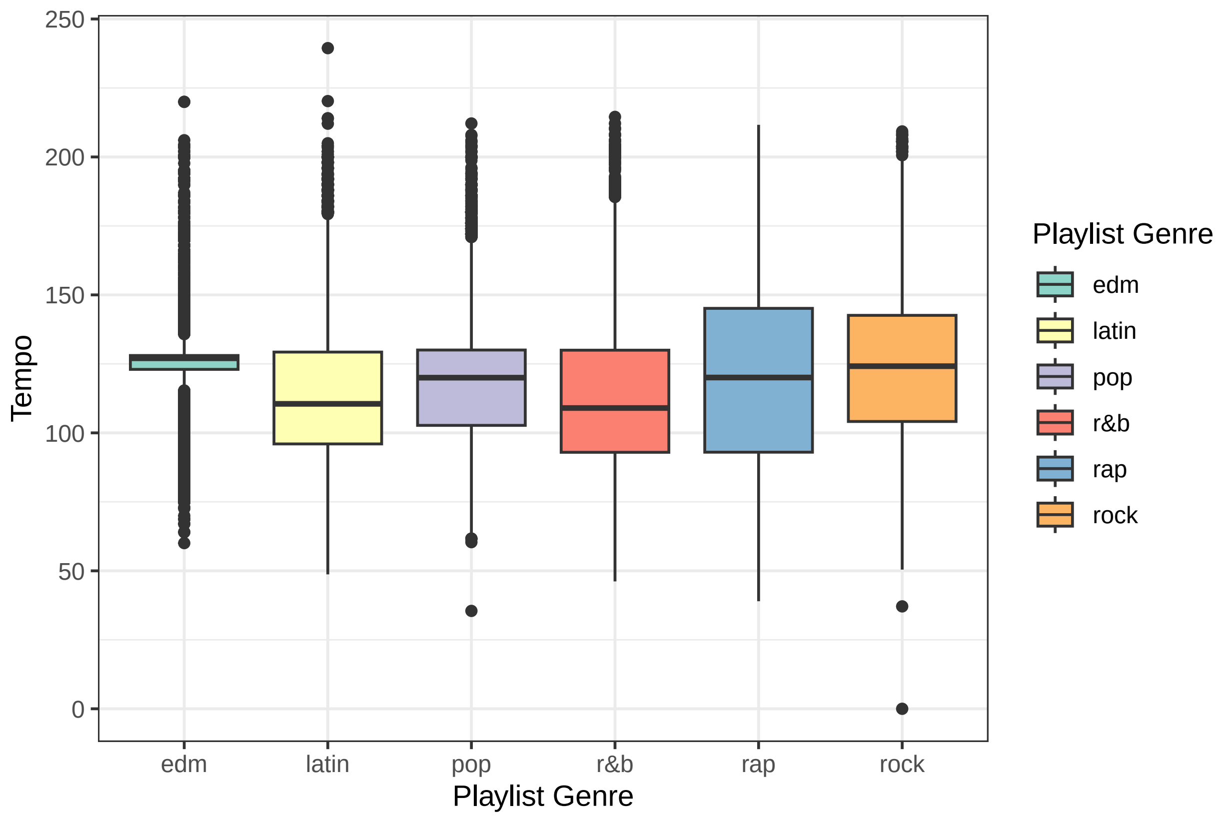
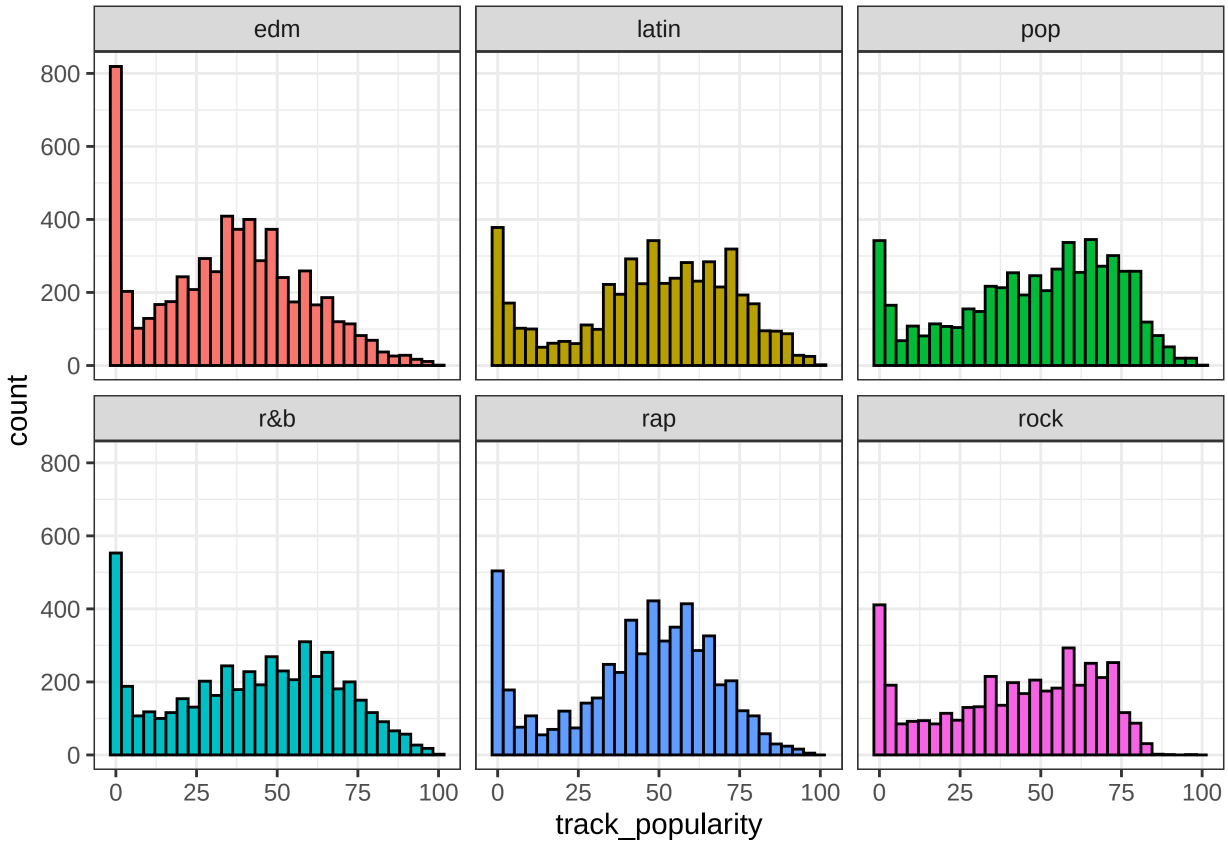


Fig.12: Boxplots of tempo against playlist genre

The popularity of songs differs between genres

```
# track_popularity
spotify_songs %>%
  ggplot(aes(x = track_popularity, fill=playlist_genre)) +
  geom_histogram(colour="black", show.legend = FALSE) +
  facet_wrap(.~playlist_genre) +
  theme_bw()
```

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




```r
spotify_songs %>%
  ggplot(aes(playlist_genre, track_popularity, fill = playlist_genre)) +
         geom_boxplot() +
  scale_fill_brewer(palette = "Set3") +
  theme_bw() +
  theme(plot.caption = element_text(hjust = 0.5)) +
  labs(caption = "Fig.13: Boxplots of track popularity against playlist genre",
      x = "Playlist Genre",
      y = "Track popularity",
      fill = "Playlist Genre")
```

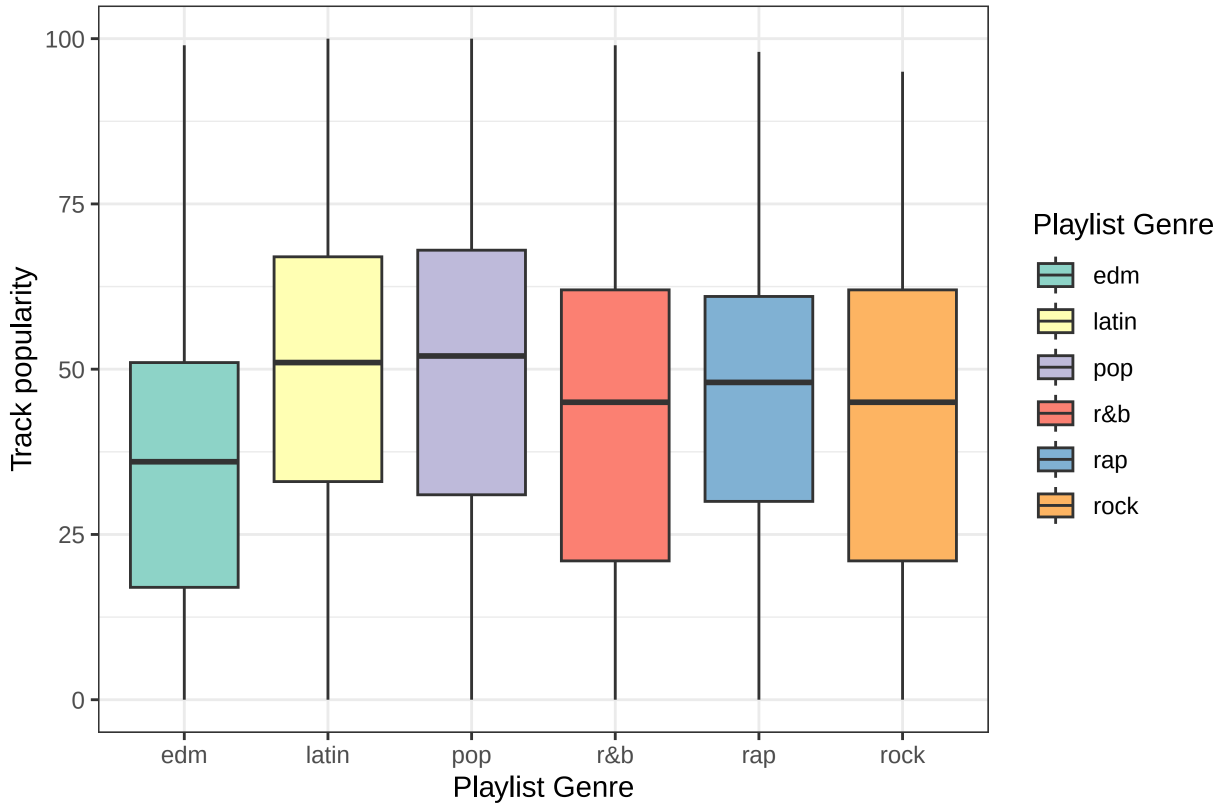


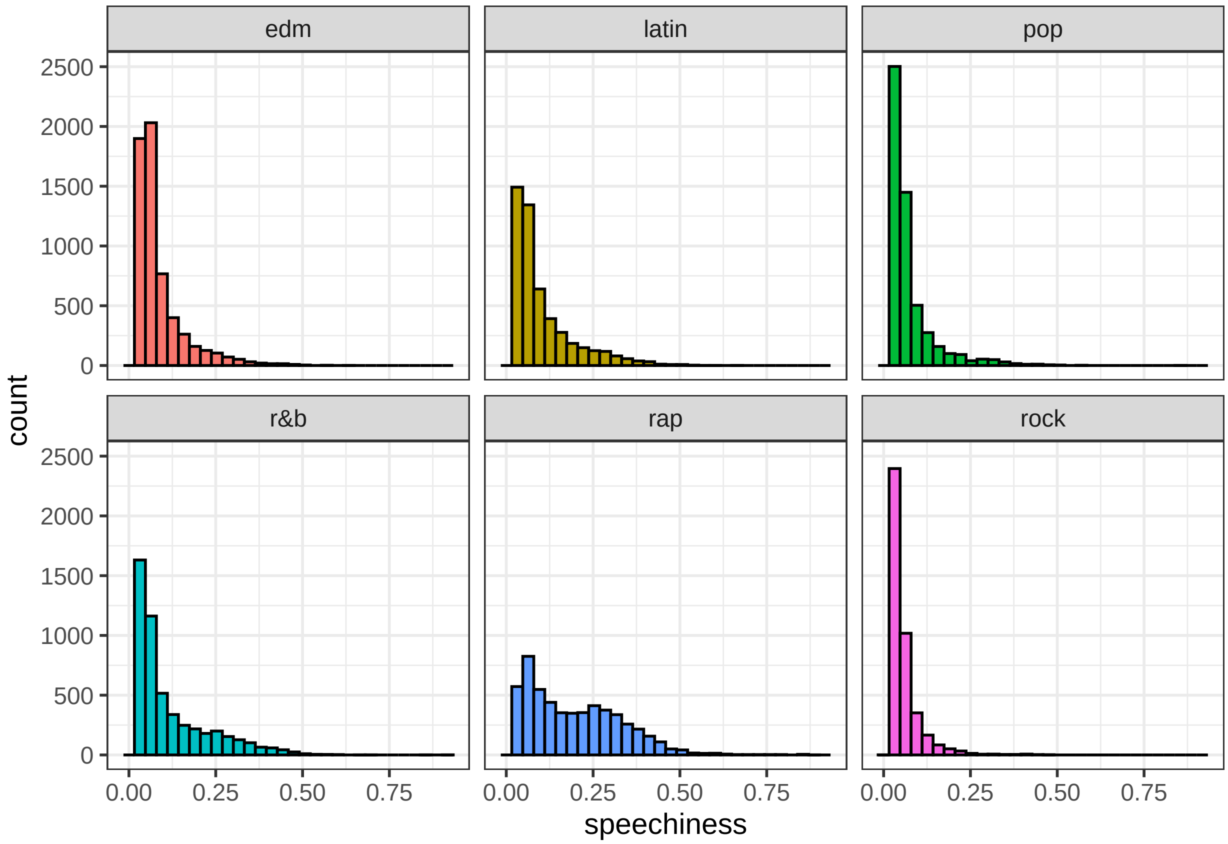
Fig.13: Boxplots of track popularity against playlist genre

How "speechy" the song is can help predict a song's genre.

There is a difference in speechiness for each genre

```r
# speechiness
spotify_songs %>%
  ggplot(aes(x = speechiness, fill=playlist_genre)) +
  geom_histogram(colour="black", show.legend = FALSE) +
  facet_wrap(.~playlist_genre) +
  theme_bw()
```

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




```
spotify_songs %>%
  ggplot(aes(playlist_genre, speechiness, fill = playlist_genre)) +
          geom_boxplot() +
  scale_fill_brewer(palette = "Set3") +
  theme_bw() +
  theme(plot.caption = element_text(hjust = 0.5)) +
  labs(caption = "Fig.14: Boxplots of speechiness against playlist genre",
      x = "Playlist Genre",
      y = "Speechiness",
      fill = "Playlist Genre")
```

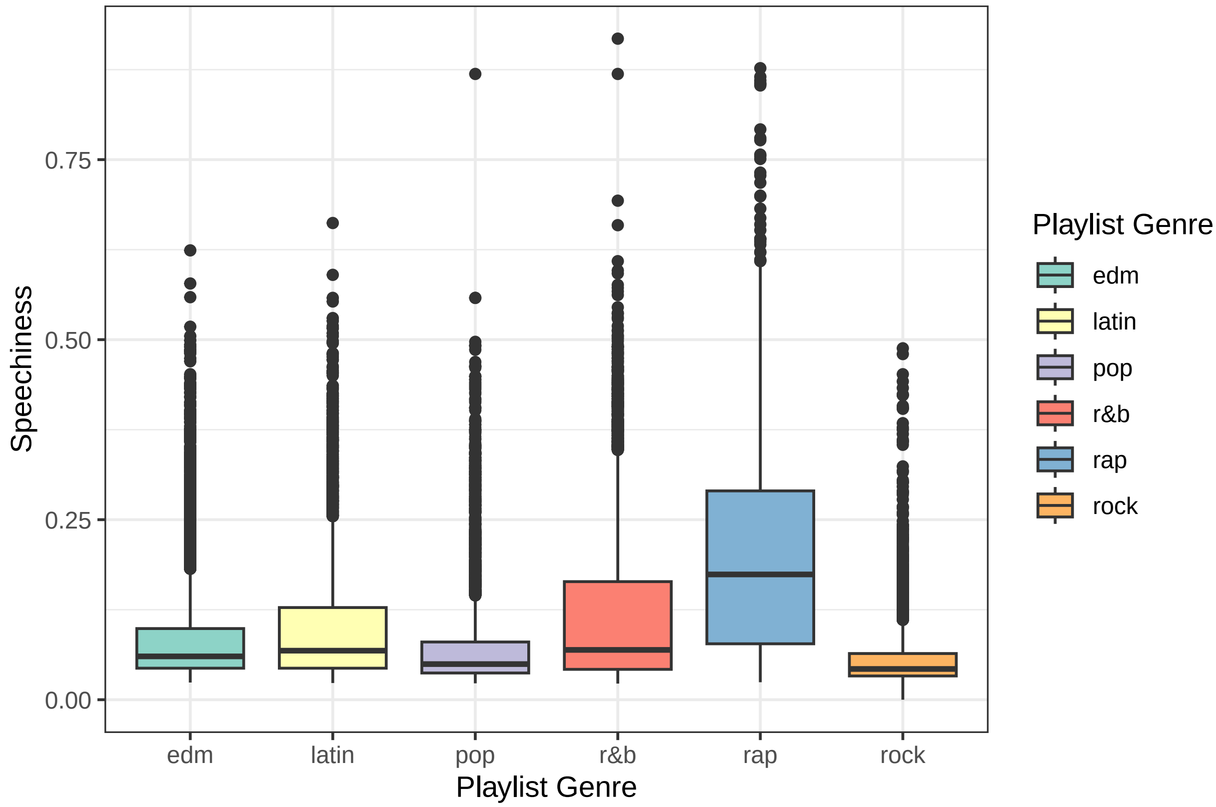


Fig.14: Boxplots of speechiness against playlist genre

Track popularity changes over time

```
spotify_songs %>%
  group_by(year, playlist_genre) %>%
  summarise(mean_popularity = mean(track_popularity)) %>%
  ggplot(aes(x = year, y = mean_popularity, color = playlist_genre)) +
  geom_point() +
  theme_bw() +
  theme(plot.caption = element_text(hjust = 0.5)) +
  labs(caption = "Fig.15: Scatterplot of track popularity over year",
       x = "Playlist Genre",
       y = "Track popularity",
       fill = "Playlist Genre")
```

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

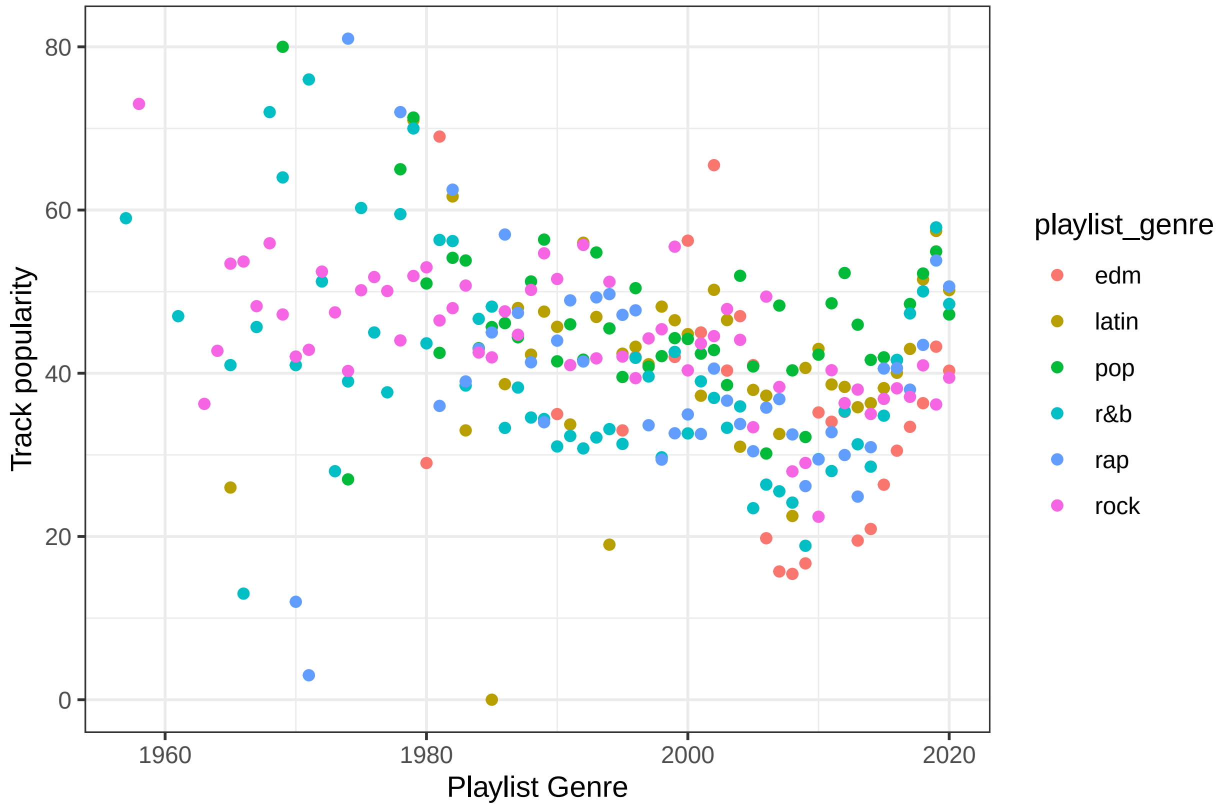


Fig.15: Scatterplot of track popularity over year

Data Sampling, Spliting and Preprocessing

```
# data sampling
set.seed(1879781)
sample_data <- spotify_songs %>%
  group_by(playlist_genre) %>%
  sample_n(size = 1000) %>%
  ungroup()
sample_data
```

```
## # A tibble: 6,000 x 18
##    track_artist  track_popularity playlist_name playlist_genre playlist_subgenre
##    <chr>                    <dbl> <chr>         <chr>          <chr>
##  1 Sam Smith                    7 Pop Hits 200~ edm            pop edm
##  2 Otto Knows                   3 Gym (Melbour~ edm            progressive elec~
##  3 Mahmut Orhan                24 EDM Trap      edm            pop edm
##  4 CLiQ                        34  ELECTRO-HO~ edm            electro house
##  5 W&W                         62 Big Room Bea~ edm            big room
##  6 Ben Yoo Suk                  8 Deep Electro~ edm            progressive elec~
##  7 Shawn Mendes                75 Electro Hous~ edm            electro house
##  8 R3HAB                        1 Fitness Work~ edm            progressive elec~
##  9 Aivarask                    39  BASSBOOSTE~ edm            electro house
## 10 Nath Jennings               17 Bounce United edm            big room
## # i 5,990 more rows
## # i 13 more variables: 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>, year <dbl>
```

```r
count(sample_data, playlist_genre)
```

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

```r
# data spliting
set.seed(1879781)
spotify_split <-
  initial_split(dplyr::select(
    sample_data, -track_artist, -playlist_name, -playlist_subgenre),
    strata = playlist_genre)
spotify_train <- training(spotify_split)
spotify_test <- testing(spotify_split)
spotify_train
```

```
## # A tibble: 4,500 x 15
##    track_popularity playlist_genre danceability energy   key loudness  mode
##               <dbl> <chr>                 <dbl>  <dbl> <dbl>    <dbl> <dbl>
##  1                7 edm                   0.42   0.435     0    -6.44     1
##  2                3 edm                   0.596  0.727     3    -7.07     1
##  3               24 edm                   0.634  0.702     6    -6.32     0
##  4               34 edm                   0.798  0.847    11    -4.61     0
##  5               62 edm                   0.388  0.971     6    -2.27     0
##  6                8 edm                   0.599  0.746    10   -10.7      0
##  7               75 edm                   0.706  0.855    10    -5.38     1
##  8                1 edm                   0.705  0.949    10    -3.58     1
##  9               39 edm                   0.655  0.545     8    -9.80     0
## 10               38 edm                   0.644  0.936     9    -2.92     1
## # i 4,490 more rows
## # i 8 more variables: speechiness <dbl>, acousticness <dbl>,
## #   instrumentalness <dbl>, liveness <dbl>, valence <dbl>, tempo <dbl>,
## #   duration_ms <dbl>, year <dbl>
```

```r
spotify_test
```

```
## # A tibble: 1,500 x 15
##    track_popularity playlist_genre danceability energy   key loudness  mode
##               <dbl> <chr>                 <dbl>  <dbl> <dbl>    <dbl> <dbl>
##  1               17 edm                   0.784  0.648     0    -8.70     0
##  2                0 edm                   0.606  0.925     8    -4.26     0
##  3               63 edm                   0.601  0.726     1    -4.83     1
##  4               36 edm                   0.573  0.92     10    -3.92     0
```

```
##  5              29 edm                     0.707  0.841     6    -6.93     0
##  6              28 edm                     0.469  0.923     2    -1.76     1
##  7              61 edm                     0.478  0.818     4    -5.08     1
##  8               0 edm                     0.615  0.751     6    -5.36     0
##  9              51 edm                     0.633  0.856     1    -4.93     0
## 10              65 edm                     0.719  0.747    11    -6.37     0
## # i 1,490 more rows
## # i 8 more variables: speechiness <dbl>, acousticness <dbl>,
## #   instrumentalness <dbl>, liveness <dbl>, valence <dbl>, tempo <dbl>,
## #   duration_ms <dbl>, year <dbl>
```

```r
# data preprocessing
doParallel::registerDoParallel()
spotify_recipe <- recipe(playlist_genre ~ . , data = spotify_train) %>%
  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: 14

##

## -- Training information

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

##

## -- Operations

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

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

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

```r
spotify_train_preproc <- juice(spotify_recipe)
spotify_test_preproc <- bake(spotify_recipe, new_data = spotify_test)
head(spotify_train_preproc)
```

```
## # A tibble: 6 x 15
##   track_popularity danceability  energy    key loudness   mode speechiness
##              <dbl>        <dbl>   <dbl>  <dbl>    <dbl>  <dbl>       <dbl>
## 1           -1.44        -1.62  -1.43   -1.48    0.0973  0.900      -0.660
## 2           -1.60        -0.401  0.162  -0.653  -0.113   0.900      -0.473
## 3           -0.765       -0.138  0.0253  0.178   0.134  -1.11       -0.697
## 4           -0.366        0.996  0.817   1.56    0.703  -1.11        0.267
## 5            0.753       -1.84   1.49    0.178   1.48   -1.11        0.360
## 6           -1.40        -0.380  0.266   1.29   -1.32   -1.11       -0.651
## # i 8 more variables: acousticness <dbl>, instrumentalness <dbl>,
## #   liveness <dbl>, valence <dbl>, tempo <dbl>, duration_ms <dbl>, year <dbl>,
## #   playlist_genre <fct>
```

Model Specifications

```r
# LDA
lda_spec <- discrim_linear( mode = "classification" ) %>%
  set_engine( "MASS" )
# K-nearest neighbours
knn_spec <- nearest_neighbor(mode = "classification", neighbors = tune()) %>%
  set_engine("kknn")
# random forest
rf_spec <- rand_forest(mode = "classification", mtry = tune(),
                       trees = 100, min_n = tune()) %>%
  set_engine("ranger", importance = "permutation")
```

```r
# create bootstrapped samples
set.seed(1879781)
spotify_boots <- bootstraps(spotify_train_preproc,
                            times = 5, strata = playlist_genre)
spotify_boots
```

```
## # Bootstrap sampling using stratification
## # A tibble: 5 x 2
##   splits              id
##   <list>              <chr>
## 1 <split [4500/1643]> Bootstrap1
## 2 <split [4500/1671]> Bootstrap2
## 3 <split [4500/1663]> Bootstrap3
## 4 <split [4500/1641]> Bootstrap4
## 5 <split [4500/1669]> Bootstrap5
```

Model Tuning

```r
# tune knn
doParallel::registerDoParallel()
k_grid <- grid_regular(neighbors(range = c(1, 100)), levels = 20)
```

```r
knn_tuned <- tune_grid(object = knn_spec,
                               preprocessor = recipe(
                                 playlist_genre ~ .,
                                 data = spotify_train_preproc),
                               resamples = spotify_boots,
                               grid = k_grid)
```

```r
best_knn_acc <- select_best(knn_tuned, "roc_auc")
best_knn_acc
```

```
## # A tibble: 1 x 2
##   neighbors .config
##       <int> <chr>
## 1       100 Preprocessor1_Model20
```

```r
filtered_metrics <- knn_tuned %>% collect_metrics() %>%
  filter(neighbors == 100)
filtered_metrics %>%
  kable(caption = "Metrics for k-NN with 100 neighbors")
```

Table 7: Metrics for k-NN with 100 neighbors

neighbors	.metric	.estimator	mean	n	std_err	.config
100	accuracy	multiclass	0.4827927	5	0.0043619	Preprocessor1_Model20
100	roc_auc	hand_till	0.8011984	5	0.0031053	Preprocessor1_Model20

```r
knn_final <- finalize_model(knn_spec, best_knn_acc)
knn_final
```

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

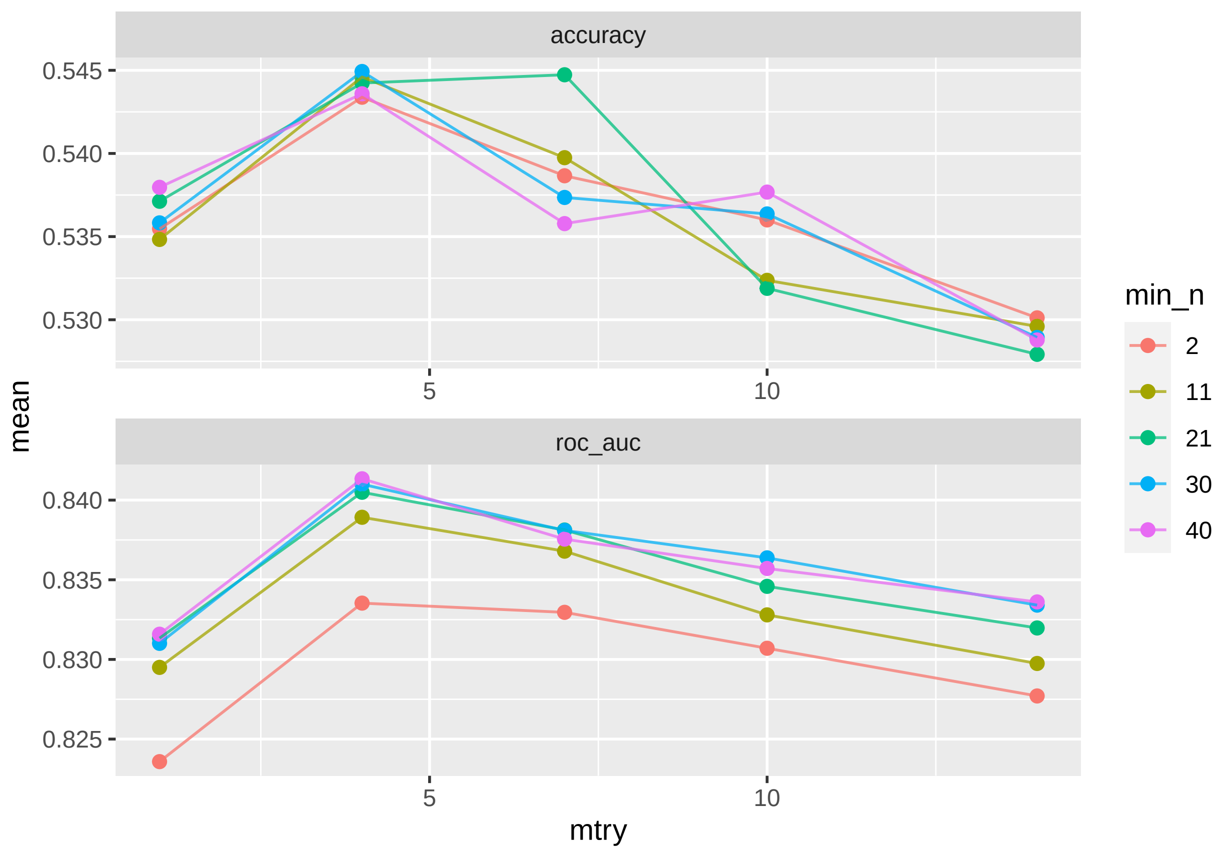
```r
# tune random forest
rand_grid <- grid_regular(finalize(mtry(), spotify_train_preproc %>%
              dplyr::select(-playlist_genre)),
  min_n(),
  levels = 5)
rand_grid
```

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

```
## 4     10     2
## 5     14     2
## 6      1    11
## 7      4    11
## 8      7    11
## 9     10    11
## 10    14    11
## # i 15 more rows
```

```
doParallel::registerDoParallel()
set.seed(1879781)
rf_tuned <- tune_grid( object = rf_spec,
                       preprocessor = recipe(
                         playlist_genre ~ . , data = spotify_train_preproc),
                       resamples = spotify_boots,
                       grid = rand_grid)
```

```
rf_tuned %>%
 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)
```



```
best_rf_acc <- select_best(rf_tuned, "roc_auc")
best_rf_acc
```

```
## # A tibble: 1 x 3
##    mtry min_n .config
##   <int> <int> <chr>
## 1     4    40 Preprocessor1_Model22
```

```
filtered_metrics <- rf_tuned %>% collect_metrics() %>%
  filter(mtry == 4, min_n == 40)
filtered_metrics %>%
  kable(caption = "Metrics for Random Forest with mtry 4 and min_n 40")
```

Table 8: Metrics for Random Forest with mtry 4 and min_n 40

mtry	min_n	.metric	.estimator	mean	n	std_err	.config
4	40	accuracy	multiclass	0.5435901	5	0.0026373	Preprocessor1_Model22
4	40	roc_auc	hand_till	0.8413386	5	0.0015319	Preprocessor1_Model22

```
rf_final <- finalize_model(rf_spec, best_rf_acc)
rf_final
```

```
## Random Forest Model Specification (classification)
##
## Main Arguments:
##   mtry = 4
##   trees = 100
##   min_n = 40
##
## Engine-Specific Arguments:
##   importance = permutation
##
## Computational engine: ranger
```

Model Selection

```
set.seed(1879781)
# create cross-validation folds
spotify_cv <- vfold_cv(spotify_train_preproc, v = 5, strata = playlist_genre)
# LDA
lda_val <- fit_resamples(object = lda_spec,
                        preprocessor = recipe(playlist_genre ~ . ,
                                              data = spotify_train_preproc),
                        resamples = spotify_cv)
lda_val %>% collect_metrics()
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy multiclass 0.478     5 0.00795 Preprocessor1_Model1
## 2 roc_auc  hand_till  0.802     5 0.00288 Preprocessor1_Model1
```

```r
# knn
knn_val <- fit_resamples(object = knn_final,
                        preprocessor = recipe(playlist_genre ~ . ,
                                              data = spotify_train_preproc),
                        resamples = spotify_cv)
knn_val %>% collect_metrics()
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy multiclass 0.496     5 0.0107  Preprocessor1_Model1
## 2 roc_auc  hand_till  0.814     5 0.00286 Preprocessor1_Model1
```

```r
# random forest
rf_val <- fit_resamples(object = rf_final,
                        preprocessor = recipe(playlist_genre ~ . ,
                                              data = spotify_train_preproc),
                        resamples = spotify_cv)
rf_val %>% collect_metrics()
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy multiclass 0.552     5 0.00930 Preprocessor1_Model1
## 2 roc_auc  hand_till  0.847     5 0.00341 Preprocessor1_Model1
```

Model Evaluation

```r
set.seed(1879781)
rf <- rf_final %>%
  fit(playlist_genre ~ . , spotify_train_preproc)
rf %>% vip()
```

```r
rf_predict <- predict(rf, new_data = spotify_test_preproc,
                             type = "class") %>%
  bind_cols(spotify_test_preproc %>% dplyr::select(playlist_genre))
rf_predict
```

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

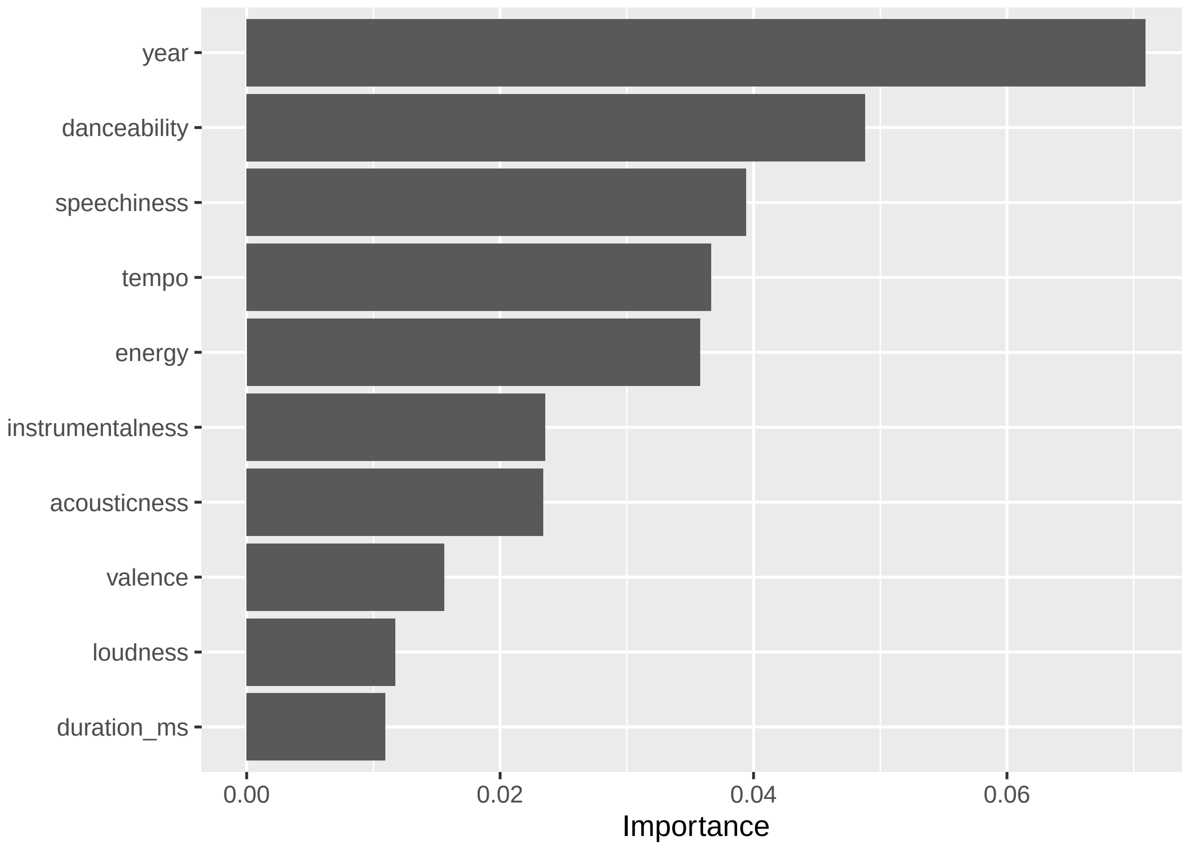


Figure 1: Variable importance plot for a Random Forest

```r
rf_predict %>%
  conf_mat(playlist_genre, .pred_class)
```

```
##           Truth
## Prediction edm latin pop r&b rap rock
##      edm   191    19  30   9  11   10
##      latin   7   118  39  30  26   10
##      pop    26    50  90  25  14   23
##      r&b     9    24  32 116  43   14
##      rap    14    35  24  60 148    2
##      rock    3     4  35  10   8  191
```

```r
sens_spec_rf <- confusionMatrix(table(rf_predict$.pred_class,rf_predict$playlist_genre))
sens_spec_rf <- cbind(rownames(sens_spec_rf$byClass), dplyr::select(as_tibble(sens_spec_rf$byClass), 1,
  rename(genre = `rownames(sens_spec_rf$byClass)`) %>%
  mutate(genre = str_remove(genre, 'Class:')) %>%
  arrange(Sensitivity)
kable(sens_spec_rf, caption = "Sensitivity and specificity for each genre")
```

Table 9: Sensitivity and specificity for each genre

genre	Sensitivity	Specificity
pop	0.360	0.8896
r&b	0.464	0.9024
latin	0.472	0.9104
rap	0.592	0.8920
edm	0.764	0.9368
rock	0.764	0.9520

```r
metrics_table <- rf_predict %>%
  metrics(playlist_genre, .pred_class)
kable(metrics_table, caption = "Metrics for Random Forest Predictions")
```

Table 10: Metrics for Random Forest Predictions

.metric	.estimator	.estimate
accuracy	multiclass	0.5693333
kap	multiclass	0.4832000

```r
rf_predict <- rf_predict %>%
  bind_cols(predict(rf, new_data = spotify_test_preproc,
                               type = "prob"))
rf_predict
```

```
## # A tibble: 1,500 x 8
##    .pred_class playlist_genre .pred_edm .pred_latin .pred_pop `.pred_r&b`
##    <fct>       <fct>              <dbl>       <dbl>     <dbl>       <dbl>
##  1 rap         edm                0.120       0.185    0.0632       0.208
```

```
##   2 edm          edm                  0.478       0.0993     0.227        0.0431
##   3 edm          edm                  0.397       0.0746     0.294        0.0628
##   4 edm          edm                  0.628       0.0404     0.0947       0.0107
##   5 edm          edm                  0.410       0.107      0.0929       0.131
##   6 edm          edm                  0.623       0.0413     0.0969       0.0217
##   7 edm          edm                  0.550       0.0398     0.174        0.0239
##   8 edm          edm                  0.332       0.138      0.282        0.0765
##   9 edm          edm                  0.332       0.163      0.307        0.0419
##  10 edm          edm                  0.489       0.135      0.244        0.0587
## # i 1,490 more rows
## # i 2 more variables: .pred_rap <dbl>, .pred_rock <dbl>
```

```r
roc_data <- rf_predict %>%
  roc_curve(playlist_genre, c(.pred_edm,.pred_latin, .pred_pop, `.pred_r&b`, .pred_rap, .pred_rock))
autoplot(roc_data) +
  labs(
    caption = "Fig.16: ROC Curve for Multiple Genres",
    x = "False Positive Rate",
    y = "True Positive Rate"
  ) +
  theme(
    plot.caption = element_text(hjust = 0.5, margin = margin(t = 10))
  )
```

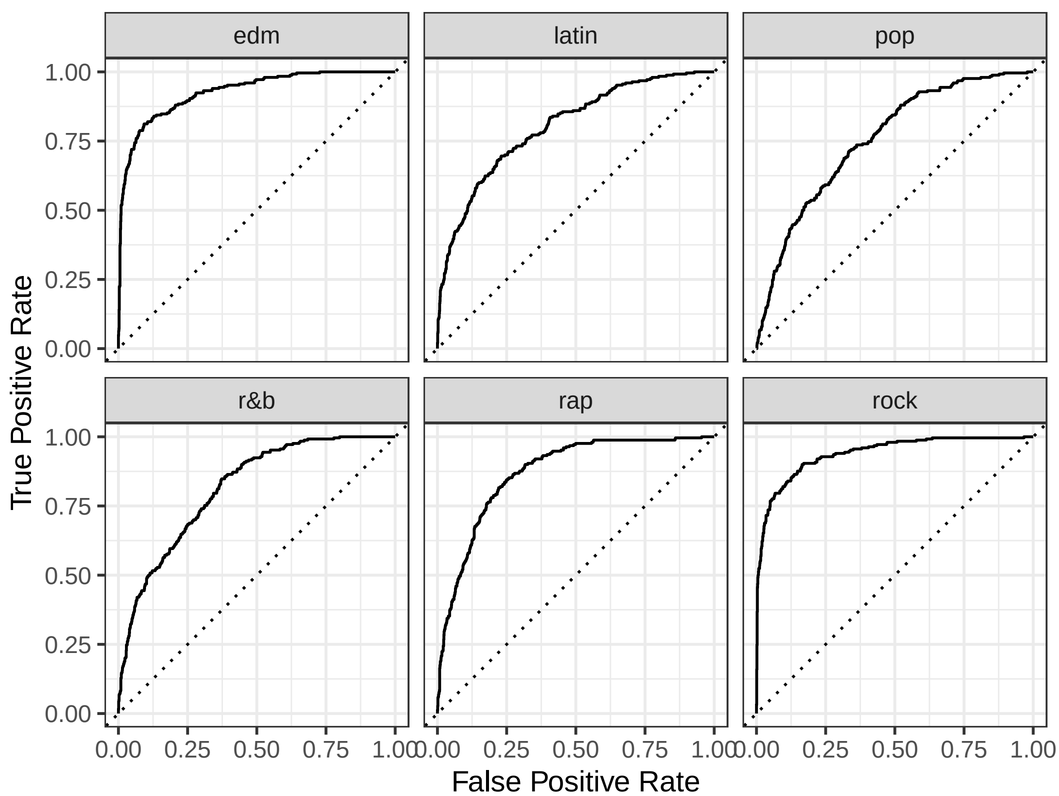


Fig.16: ROC Curve for Multiple Genres

Prediction

```
set.seed(1879781)
spotify_new <- spotify_songs %>%
  group_by(playlist_genre) %>%
  slice_sample(n=100) %>%
  ungroup()
test_preproc_new <- bake(spotify_recipe, new_data = spotify_new)
new_prediction <- predict(rf, new_data = test_preproc_new) %>%
  bind_cols(test_preproc_new %>% dplyr::select(playlist_genre))
new_prediction %>%
  conf_mat(playlist_genre, .pred_class)
```

```
##           Truth
## Prediction edm latin pop r&b rap rock
##      edm    82     5   5   4   7    3
##      latin   3    52  15  11  11    2
##      pop     8    15  45  13   6    6
##      r&b     3    13   8  47  12   10
##      rap     4    13  13  18  62    0
##      rock    0     2  14   7   2   79
```

```
new_sens_spec_rf <- confusionMatrix(table(new_prediction$.pred_class,new_prediction$playlist_genre))
new_sens_spec_rf <- cbind(rownames(new_sens_spec_rf$byClass), dplyr::select(as_tibble(new_sens_spec_rf$b
  rename(genre = `rownames(new_sens_spec_rf$byClass)`) %>%
  mutate(genre = str_remove(genre, 'Class:')) %>%
  arrange(Sensitivity)
kable(new_sens_spec_rf, caption = "Sensitivity and Specificity for New Data")
```

Table 11: Sensitivity and Specificity for New Data

genre	Sensitivity	Specificity
pop	0.45	0.904
r&b	0.47	0.908
latin	0.52	0.916
rap	0.62	0.904
rock	0.79	0.950
edm	0.82	0.952

```
metrics_table <- new_prediction %>%
  metrics(playlist_genre, .pred_class)
kable(metrics_table, caption = "Metrics for New Data")
```

Table 12: Metrics for New Data

.metric	.estimator	.estimate
accuracy	multiclass	0.6116667
kap	multiclass	0.5340000

```r
new_prediction <- new_prediction %>%
  bind_cols(predict(rf, new_data = test_preproc_new,
                                    type = "prob"))
roc_data <- new_prediction %>%
  roc_curve(playlist_genre, c(.pred_edm,.pred_latin, .pred_pop, `.pred_r&b`, .pred_rap, .pred_rock))
autoplot(roc_data) +
  labs(
    caption = "Fig.17: ROC Curve for New Data",
    x = "False Positive Rate",
    y = "True Positive Rate"
  ) +
  theme(
    plot.caption = element_text(hjust = 0.5, margin = margin(t = 10))
  )
```

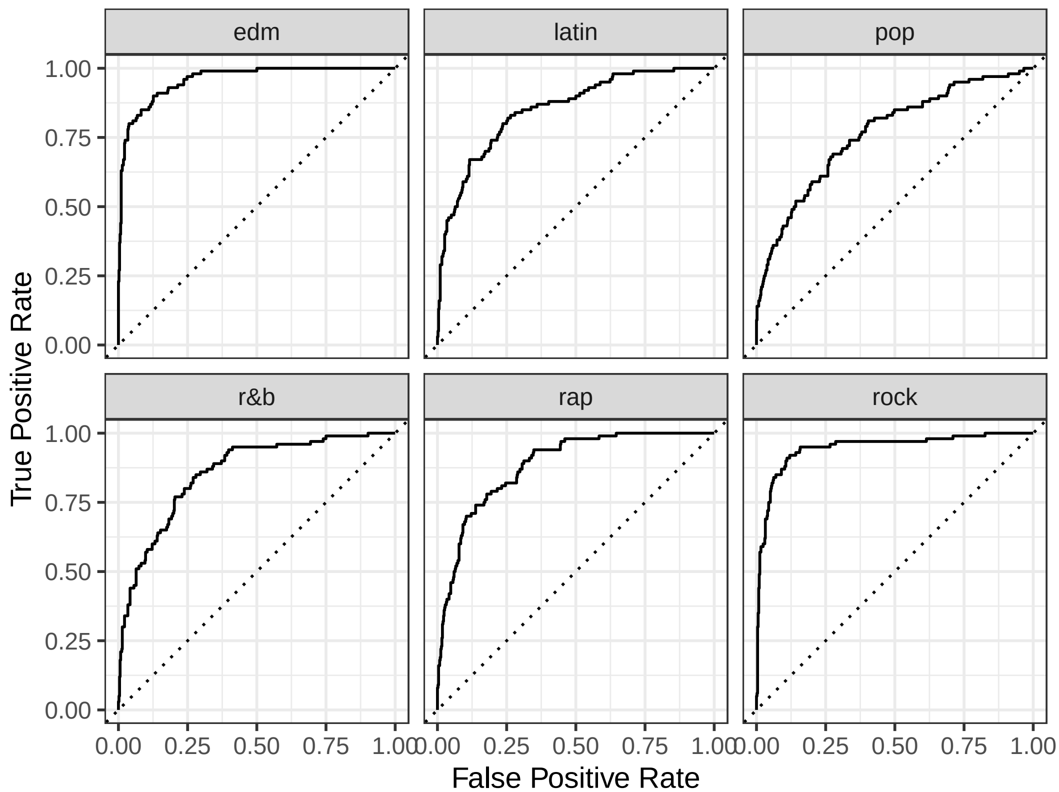


Fig.17: ROC Curve for New Data