Final Report

Jonathan Markey 1805361

Executive Summary

Spotify is the world's largest music streaming provider with over 365 million monthly active users and tens of thousands of songs that cover several genres of music. One of the most important priorities for the company is being able to continually provide the best service and features for its customers, allowing Spotify to retain its customers' loyalty and ultimately continue to remain the best music streaming service available.

To achieve this, the founders of Spotify have requested a predictive model to be created which could predict the genre that a song belongs to based on the song's information. Creating this model would allow Spotify to better recommend/advertise its songs to customers and more effectively update compilation playlists. The founders are also specifically interested in how the release year, 'speechiness', danceability, and tempo of a song could predict the genre of a song. In addition to this, the founders are also interested in an analysis of the popularity of songs between genres, the speechiness of each genre, and how popularity has changed over time.

After completing a full analysis it was found that:

- The most important factors when predicting a song's genre were the year that the song was released, the danceability, the tempo, the energy and the speechiness of a song, in that order.
- The popularity of songs does differ between genres with the pop and latin genres having, on average, the most popular song, and the edm having, on average, the least popular songs.
- There is a speechiness difference between genres with the rap genre having by far the most speechines out of all the genres. The genres with the least level of speechiness are the rock, pop and edm genres.
- The popularity of genres had changed over time. Over the long term, there had been a general downward trend in the popularity of songs. However, from 2010 to 2020 there was a noticeable upward trend in the popularity of nearly all music genres.

The final results showed that the random forest was able to correctly classify 55.53% of its positive predictions as true positives and that it would be able to correctly classify 91.11% of its negative predictions as true negatives. The model was also able to generate an AUC score of 85.02% which indicated that it could differentiate between the classes with 85.02% certainty. The linear discriminant model and k-nearest neighbour models scored slightly lower, with sensitivities and specificities of 46.47% and 89.29% for the LDA and 49.87% and 89.97% for the KNN model. The AUC scores for these models are 80.87% and 81.48% respectively. Analysing the confusion matrix, and the sensitivity and specificity scores for the individual classes showed that there was a lot of variance between the classes. The models were all able to correctly predict the edm, rap and rock classes more often than the latin, pop and r&b classes. This indicated that there could be too little variance in these classes in the original data for the models to differentiate between the classes. Future work to address this could be to subject the data to additional feature engineering as this could allow for more variance to be uncovered which would allow the models to better differentiate the classes from each other. Based on these final results it is recommended that Spotify use the random forest model to predict the genre of a song over the LDA and KNN models.

Methods

The software that was used to undertake this project was RStudio version 4.2.1. Several libraries and packages were used including tidyverse, lubridate, stringr, tidyr, skimr, recipes, inspectdf, dplyr, rsample, themis, ranger, vip, parsnip, dials, discrim, tune, pROC and kknn.

There were several steps involved in this project which included data importing and cleaning, exploratory analysis, split and preprocessing data, model tuning and fitting, and lastly model evaluation.

The first step of the project was to import the data into R and select the variables. The data used in this report was originally derived from the spotifyr package in R and consists of data on 32,833 songs. Included in the data set were 23 variables which cover all the information on each song. Out of these 23 variables, 15 variables were analysed and found to be potentially useful. The track release date information would later be split into the year, month, and day data, bringing the total number of variables to 1 observation variable and 16 predictors. The chosen variables were:

- Playlist_genre - The observation variable. This variable classifies each song into the 6 song genres.
- Energy - A value from 0 to 1 representing the perceptual measure of intensity and activity.
- Key - An integer value describing the overall pitch of the song increasing as pitch increases.
- Loudness - A value ranging from -60 to 0 measuring the loudness of a song. Songs with loudness of 1 are songs with undetected pitch.
- Mode - Binary value describing the scale the melodic content is derived from. A value of 0 is done in minor, a value of 1 is done in major.
- Acousticness - A value from 0 to 1 describing how acoustic the track is. Higher values mean more acoustic.
- Instrumentalness - A value from 0 to 1 describing how instrumental the song is. Higher values imply fewer vocals in the track.
- Liveness - Detects how "live" the music is, i.e. whether it is a studio recording or a live recording.
- Valence - A measure from 0 to 1 describing how positive the song is. A higher value means more positive.
- Duration_ms - The duration of the song in milliseconds.
- Track_popularity - A number from 0 to 100 indicating popularity. The higher the value, the more popular the song.
- Danceability - A value from 0 to 1 describing how danceable a song is. The higher the value, the more danceable the song.
- Speechiness - A value from 0 to 1 describing how 'speechy' the track is. Higher values indicate spoken-word tracks.
- Tempo - The tempo of the song in beats per minute.
- Track_album_release_date - date that the song was released on.

The second step of the project was to skim and initially analyse the data to find any missing or incorrect data. This step involved ensuring variables had been correctly coded as factors if categorical, or numerical if quantitative. Variables such as the song release date were also split into their year, month, and day form in this step. Following this, the data was inspected for any missing or incorrect values. If values were found they were either dropped or transformed to be useable. To ensure that the R scripts ran promptly, the data was also downsampled in this step to 1000 data entries from each genre class, bringing the data down to 6000 total observations. Although the data was analysed for normality and collinearity at this stage, this would not need to be acted on as it would be addressed during the split and preprocessing stages.

The next step was to conduct the exploratory data analysis. In this step the variables were visualised and analysed to determine if there were relationships present in the data and if the data was varied enough to be used in the predictive models. In this section, the additional founders' questions regarding the popularity and speechiness of each genre could be answered.

Following the EDA, the next step was to split and preprocess the data. In this step, the data was split into a training and testing set with a 75% and 25% split respectively. The training data was used to train the

predictive models and the testing data was used to test the predictive powers of the models. The purpose of creating the testing data was to ensure that the models can be run on new data. If the models were tested on the training data that was used to generate the models, there would be a bias towards this data and the models would have a stronger predictive power than it would have on the new data. With the split, the data was preprocessed to normalize the variables, remove any variables with zero variance, and remove any variables that were highly correlated with other variables. The final part of this process was to make a cross-validation data set from the training set. The purpose of the cross-validation set was to test and tune the classification models to produce the best result.

The fifth step in the project was to tune and fit the three classification models. As per the instructions set by the founders, the three predictive approaches that were tested and compared were the random forest, linear discriminant analysis (LDA) and K-nearest neighbours models.

Random forest is a machine-learning technique that can be used for classification or regression. Random forest works by initially constructing a multitude of decision trees that use different values to create an output. Those predictions are then averaged out as a final output. Due to the large number of variables in the data set, random forest classification can also be used to show which variables are most important in classifying the genre of a song (Keboola, 2020). There are three properties of the random forest model that can be tuned, the number of trees in the forest, the number of predictors that are randomly sampled, and the number of data points in a node that are required for the node to be split. As the founders requested that the forest is made of 100 trees, only the number of predictors and data points will be tuned at 5 levels.

Linear discriminant analysis (LDA) is a classification model that is a linear model that can be used to make classifications. LDA uses dimensionality reduction to reduce the number of dimensions in a dataset while retaining as much information as possible. This means that it will reduce the number of variables in a dataset until it finds a linear combination of variables that provide the best separation between classes (Brownlee, 2020). LDA does not require tuning and only requires that the numerical variables are normalised to have a mean of 0 and standard deviation of 1.

K-nearest neighbours is a classification machine learning algorithm that is used to estimate the likelihood that a data point will belong to a specific class of a variable. This algorithm functions under the idea that a data point is more likely to be like those that are close to it compared to those that are far away from it (Javapoint, 2021). Under this assumption, the algorithm is able to classify data by looking at the k data points that are closest to the new value, and then classifying the new value based on the most populated class that these data points belong to. The only property of the K-nearest neighbours model that can be tuned is the number of neighbours that will be compared. The founders requested that the number of neighbours is tuned to a range of 1 to 100 with 20 levels.

Once these models were fitted the last step was to evaluate the models and determine which provided the best results. To determine which model best suited the founders' needs the AUC, the sensitivity and specificity of each model were calculated and compared. The area under the ROC curve (AUC) is a performance metric that is measured as a rate from 0 to 1. The ROC curve is a graph that shows the performance of a classification model at all classification thresholds. It represents the models true positive rate and false positive rate. An AUC of 1 indicates that the predictive model is able to perfectly distinguish between classes and make correct predictions. AUC values that get closer to 0.5 indicate that the predictive model cannot distinguish between classes and is more random. As AUC approaches 0 it's result is similar as to if it approaches 1 but inversed, meaning that it can perfectly misclassify results. Sensitivity and specificity are metrics used to measure the probability of a positive test being a true positive and a negative test being a true negative. A true positive means that if the model predicts a song to be a pop song then it will be a true positive if the actual genre of the song is pop. A true negative means that if the actual song is pop, then so long as the prediction is not pop it is a true negative. As the prediction was done for multiple genres the sensitivity and specificity were calculated for each genre.

Results

Analysing the release data found that there were 1886 missing or incorrect values. As this only constituted approximately 6% of the data and as the data would be down-sampled to 6000 total observations (1000 of each genre) these were dropped from the sample. Analysing the loudness data showed that there were 6 values that could not be measured. These values were replaced with the median value. Once the data was tame and cleaned, the data was down-sampled to 1000 of each genre to reduce the run time of the R scripts.

Exploratory Data Analysis

For the EDA many of the variables were analysed in a similar way. The variables were analysed by initially creating histograms and bar charts faceted by the playlist genre. This allowed for each variable's shape, location, spread, and outliers to be analysed. After this, the summary statistics were printed and the range, interquartile range, mean, and median were analysed for the variable. The summary statistics were for the variable overall and not stratified by genre. Below are the key takeaways from the EDA for each variable. Please refer to Figures 1 - 15 in Appendix A for full plots and summaries of each variable.

Energy (Figure 1)

- Shape: Unimodal and left-skewed for edm, latin, pop and rock classes. Normal for r&b and rap.
- Location: Largest peaks are located in the 0.6 to 0.9 range for most classes. The mean and median are 0.695 and 0.717 respectively.
- Spread: All classes are spread between 0.25 to 1, except edm which is narrower from 0.375 to 1. The range is from 0.000175 to 1. Interquartile range (IQR) is from 0.574 to 0.84.
- Outliers: Present in rock, rap, pop and latin classes at lower values.

Key (Figure 2)

- Shape: The key is categorical and ordinal.
- Location: The peaks of data are quite varied between classes.
- Spread: All classes have a large quantity of data in the 0-2 range and then very little data at 3. After 3 all classes' key varies.
- Outliers: None appear to be present.

Loudness (Figure 3)

- Shape: All appear to be unimodal with a slight left-skew.
- Location: Largest peaks are located in the -7 to -5 range for all classes. The mean and median are located at -6.741 and -6.180 respectively.
- Spread: All classes are spread between -15 to -2. The range is from -35.960 to -0.155. IQR is from -8.191 to -4.662.
- Outliers: Some outliers are present in the rock class at the lower ranges.

Mode (Figure 4)

- Shape: Mode is categorical and binary.
- Location: For all classes, the mode count is larger in the 1 class.
- Spread: There are 2600 values in the 0 class and 3400 in the 1 class.
- Outliers: N/A

Acousticness (Figure 5)

- Shape: All classes are unimodal and heavily right-skewed.
- Location: Peaks on edm and rock classes are much larger than other classes. The mean and median are located at 0.182 and 0.0856 respectively.
- Spread: Spread is very wide for pop, r&b, and rock classes, ranging from 0 to 0.95. Spread for edm, latin and rap classes are narrower from 0 to 0.5 Range is from 0 to 0.992. IQR is from 0 to 0.269.
- Outliers: Outliers appear to be present in all classes for the acousticness at the higher ranges.

Instrumentalness (Figure 6)

- Shape: All classes share the same shape and are unimodal and heavily right-skewed.
- Location: Peaks for all classes are located at the 0 value. The mean and median are located at 0.0831 and 0.0000164 respectively.
- Spread: Spread is very narrow for all classes with little distribution. The range is from 0 to 0.994. IQR is from 0 to 0.00445.
- Outliers: There appear to be outliers in all classes except for latin at the higher ranges.

Liveness (Figure 7)

- Shape: All classes share a similar shape and are unimodal and right-skewed.
- Location: The location of all peaks located at 0.1. The mean and median are located at 0.191 and 0.126 respectively.
- Spread: The majority of data is spread from the 0 to 0.375 range for all classes. The range is from 0 to 0.988. IQR is from 0.0929 to 0.244.
- Outliers: There appear to be outliers in all classes at the higher ranges.

Valence (Figure 8)

- Shape: Shape is varied between classes. Edm is multimodal with a right skew. Latin is multimodal with a left skew. The remaining classes are multimodal with a more even and normal distribution.
- Location: As all classes are multimodal they have multiple peaks spread across their distributions. The r&b and rap share the most similar distribution and have similar peaks. The mean and median are located at 0.507 and 0.508 respectively.
- Spread: Spread is wide for all classes ranging from 0 to 1. The range is from 0 to 0.978. IQR is from 0.327 to 0.687.
- Outliers: There only appears to be one outlier in the latin class at the lower range.

Duration_ms (Figure 9)

- Shape: All classes share a similar shape and are unimodal. Edm, r&b and rock have a slight right skew. Latin, pop and rap have a slightly more normal distribution.
- Location: All classes share similar peaks in the 200000ms to 220000ms range. The mean and median are located at 224325 and 215046 respectively.
- Spread: Range is from 4000ms to 516769ms. IQR is from 187290ms to 251442ms.
- Outliers: there do appear to be some outliers in the edm, pop, r&b and rock classes located at the higher ranges, except for rock, which has some in the lower range.

Track_popularity (Figure 10)

- Shape: All classes are multimodal with a wide distribution and many peaks.
- Location: All classes have a similar peak at 0. The main location of the remaining data for edm is lower than the remaining classes. The mean and median are located at 43.3 and 46.0 respectively.

- Spread: Range is from 0 to 98. IQR is from 26 to 63.
- Outliers: There do not appear to be any obvious outliers in the data.

Danceability (Figure 11)

- Shape: The shape for all classes is unimodal. The edm, pop and rock classes are normally distributed while the latin, r&b and rap classes have a left-skew.
- Location: Mean and median are located at 0.652 and 0.669 respectively.
- Spread: Range is from 0 to 0.979. IQR is from 0.555 to 0.76.
- Outliers: There are some outliers present in all classes except for latin. The outliers are present at the lower stages for all the remaining variables. There are outliers present at the higher range for the rock class.

Speechiness (Figure 12)

- Shape: All classes share a similar shape being unimodal with a high right skew.
- Location: All classes share similar peaks from 0 to 0.1, however, the magnitude of peaks is greater for the rock and pop classes. The mean and median are located at 0.1065 and 0.0635 respectively.
- Spread: The spread is similar for all classes ranging from 0 to 0.3. The rap class is more evenly spread out than the other classes. The range is from 0 to 0.918. IQR is from 0.0407 to 0.130.
- Outliers: There only appear to be outliers present in the rock class at the higher ranges.

Year/Month/Day (Figure 13-15)

- Shape: The shape of the year data is unimodal with a heavy left-skew.
- Location: The location with the highest peak in the year data is located around 2018 for all classes. In the monthly data, the highest peak is in month 1. For the day data, the peak is on day 6.
- Spread: The spread of data is quite varied for the year data. Edm and latin have a narrow spread, pop, r&b and rap have a moderate spread and rock has a wide spread. For the month and day data, the distribution is quite similar between classes with minor differences.
- Outliers: There do appear to be some outliers in the year data for the pop and rock classes.

Key observations:

The loudness, instrumentalness, liveness, mode, month and day variables showed little variance between classes. This was important as the type of predictive models that would be generated were classification models. The little variance between classes for variables could result in false positives or false negatives in the prediction. In the tuning and fitting stages of the project, these variables could be dropped to test if prediction accuracy increases.

Founder Questions:

Does the popularity of songs differ between genres? (Appendix A: Figure 10, 16)

To answer this question the histogram, boxplot and summary statistics of each genre class were analysed. The popularity of songs did appear to differ between genres. The genre classes with the highest mean and median track popularity were latin and pop with 47.74 and 51, and 48.49 and 54 respectively. The genres with the next highest means and medians were the r&b, rap and rock classes with 42.34 and 45, 43.44 and 48. and 42.01 and 45 respectively. The class with the lowest mean and median was edm with 35.76 and 37. The reason for the gap between the mean and median for all classes could be explained by analysing the histogram which showed a large peak at the 0 popularity value for all classes. The range for popularity was quite similar in all genres with a min of 0 for all classes and max in the high 90 ranges for all classes except for the rock class which had a max popularity of 88. Although the IQR location varied between all

the classes the emd, Latin, pop and rap all had a very similar range in the IQR ranging from 31-35 units of popularity. The r&b and rock classes had the widest IQR ranging from 40-41.5 units of popularity. There didn't appear to be any outliers for any of the classes. Based on this analysis it could be concluded that edm was the least popular genre. Latin and pop were on average the most popular genres with r&b, rap and rock close behind. Although r&b and rock were popular genres, they had the widest variance in the ranges of popularity.

Is there a difference in speechiness for each genre? (Appendix A: Figure 12, 17)

To answer this question the histogram, boxplot and summary statistics of each genre class were analysed. The speechiness of songs did appear to differ between genres. The genre class with the highest mean and median by a significant amount was the rap genre with 0.193 and 0.168. The mean and median were very similar for the edm, latin, pop and r&b classes with the mean value ranging from 0.08 to 0.12 and the median ranging from 0.05 to 0.07. Rock had the lowest mean and median of 0.0596 and 0.0422. The range differed significantly between classes with the r&b class having the highest max value of 0.918 and pop having the lowest max of 0.437. The min value was very similar across all values situated in the 0.020 to 0.025 range, except for rock which had a min value of 0. The class with the widest IQR was the rap class with an IQR of 0.0738-0.292 which was 0.218 units. The IQR of the latin and r&b classes were the next largest IQRs ranging from 0.0838 and 0.118 respectively. The IQR then decreased significantly for the edm, pop and rock classes with IQRs ranging 0.0615, 0.0460 and 0.03187 respectively. The larger IQR of the rap class can be seen on the histogram where the values were more spread out and the peak was significantly smaller than in the other classes. There did appear to be many outliers present on the boxplot. Edm, latin, pop, r&b, and rock classes all appeared to have a significant amount of outliers present at the ranges between approximately 0.2-0.5. There are only a few outliers present in the rap class all above the 0.65 level. The r&b's max value of 0.918 did appear to be a significant outlier for the class. Based on this analysis it can be concluded that there was a significant difference in speechiness between genres. Although rap had the highest speechiness on average, followed by r&b and latin, these genres also contained the most variance in speechiness between songs. Edm, pop and rock genres contained the least speechiness and little variance between songs.

How does popularity change over time? (Appendix A: Figures 18-19)

To answer this question two plots were generated. The mean track_popularity had been generated for each year and had been plotted on a graph with the linear model for the mean track_popularity. In Figure 18 this was for all the data and in Figure 19 this had been faceted by genre. Figures 18 and 19 indicated song popularity was decreasing over time for all genres except for the latin genre as its slope appeared to be 0. Although the song's popularity was generally decreasing, the linear model line displayed may not have been as accurate. As can be seen between approximately 2010 and 2020, the popularity of songs did appear to be increasing, especially in the edm, latin, r&b and rap genres. Although the long-term trends indicated that popularity was decreasing, a short-term perspective could show that it was actually increasing.

The linear models for popularity are listed below:

- All genres track popularity = 821.498 - 0.390 x (year)
- Pop track popularity = 384.029 - 0.170 x (year)
- Rock track popularity = 749.896 - 0.376 x (year)
- R&btrack popularity = 638.332 - 0.300 x (year)
- Rap track popularity = 1119.74 - 0.540 x (year)
- Latin track popularity = -69.507 + 0.0554 x (year)
- Edm track popularity = 1424.993 - 0.692 x (year)

As can be seen, the track popularity was decreasing for all the data and for all the genre classes except for latin which was slightly increasing. Based on this analysis it can be concluded that although current long-term linear models were indicating that track popularity was decreasing over time, there was evidence to support that the track popularity for the edm, latin, r&b and rap genres had been increasing over the last 10 years. Further research may be required to confirm this.

Tune and Fit Models

Before the models could be tuned and fitted, the data was split into a 75% training and 25% testing split. Following this, the data was processed to normalize the numeric data, and remove highly correlated variables and variables that had zero variance. In addition to this, a 10-fold cross-validation set was generated from the testing data, stratified by genre.

The first model to be generated was the random forest model. Before the model could be tuned, the feature's importance needed to be determined. As determined in the EDA, some variables contained little variance between classes and because of this they may not have been as important when used to predict a song's genre. Testing the random forest model by removing some of the less important variables could result in a more accurate model. Appendix B Figure 20 shows the importance of features in a random forest model. As can be seen, year, danceability, tempo, energy, and speechiness were the most important features in predicting genre while mode, liveness, month, key, and day were the least. To determine the set of variables that would produce the best results, the least important variables were removed until the estimated AUC failed to increase. The estimated AUC was derived from resamples of fitting the model to the cross-validation set. The estimated AUC result was 0.8450 for the un-tuned random forest with all variables. After testing, removing mode, liveness, month, key and day resulted in a slightly better estimated AUC of 0.8468. After testing, the reduced data set was kept for the LDA and KNN models as these models produced a slightly higher AUC.

Following this, the random forest was generated and its parameters tuned. As the number of trees parameter had already been set by the founders at 100, it did not require tuning. The parameters dictating the number of predictors that are randomly sampled, and the number of data points in a node that are required for the node to be split were tuned. As these parameters had to be tuned to 5 levels there would be a total of 25 different combinations of the levels of the parameters that were tested. To evaluate what combination of parameters would be the best, the estimated AUC was used. After tuning, the random forest model was finalized with the tuned parameters and fitted to the testing data. The model was then used to predict the training data. To measure the performance the categorical metrics and actual AUC were generated from these predictions.

The LDA model did not require tuning and was generated by fitting the model to the training data and used to predict the testing data. The model does not require tuning as it does its dimensionality reduction and setting up dummy variables as part of its processes. The only requirement for the LDA model to run is that the numeric data was normalised with a mean of 0 and standard deviation of 1, which the training data had already been preprocessed for. To measure the performance the categorical metrics and actual AUC were generated from the predictions.

The final KNN model was generated and its parameters were tuned. For the KNN model, the only parameter that needed to be tuned was the number of neighbours that were used to calculate the data points class. The founders had already set the range of neighbours to be from 1 to 100 with 20 levels. This meant that the number of neighbours tested increased by approximately 5 at each level. To evaluate which level would be the best, the estimated AUC was used. This tuning determined the optimal level of neighbours to be 100. After this, the model was then finalized with the best parameter and fitted with the training data. The model was then used to predict the testing data. To measure the performance the categorical metrics and actual AUC were generated from these predictions.

Discussion

The Random forest model produced a sensitivity of 55.53%, a specificity of 91.11%, and an AUC of 85.02%. The LDA model produced a sensitivity of 47.67%, a specificity of 89.53%, and an AUC of 80.97%. The KNN model produced a sensitivity of 49.13%, a specificity of 89.83%, and an AUC of 80.77%. (Appendix B - Table 1)

From these results, it can be seen that the random forest model has the most predictive power when predicting genre out of all the models. The sensitivity of the random forest model indicates that the model predicted

55.53% of the positive predictions were true positives. When analysing the individual sensitivity statistics by class (Appendix B - Table 2), it can be seen the model received moderately high values for the rock, edm and rap classes, with the rock class receiving the highest sensitivity of 78.00%. However, the model also received poor results for the latin, pop and r&b classes, with pop receiving the lowest sensitivity of 34.80%. The specificity is very high indicating the model predicted true negatives 91.11% of the time. Similarly to the specificity, the class with the highest specificity was rock and the class with the lowest specificity was the pop class. The AUC of the model was very high and indicated that the models would be able to correctly classify a prediction as true positive or true negative 85.02% of the time.

The KNN model was the second best performing model with a sensitivity and specificity indicating that the model was able to correctly predict a true positive and true negative 49.87% and 89.97% of the time. When analysing the individual sensitivity and specificity statists by class, it can be seen that the best performing class was the edm class with a sensitivity of 66.00% and a specificity of 93.20%. The worst performing class was the pop class with a sensitivity of 33.60% and specificity of 86.72%. The AUC for the KNN model is high indicating that the model would be able to differentiate between classes and correctly classify a prediction as true positive or true negative 81.48% of the time.

The LDA was the worst performing model with a sensitivity and specificity indicating that the model was able to correctly predict a true positive and true negative of 46.47% and 89.97% of the time. When analysing the individual sensitivity and specificity statistics by class, it can be seen that the best performing class was the rock class with a sensitivity 56.00% and specificity of 91.20%. The worst performing class was the r&b class with a sensitivity of 35.60% and specificity of 87.12%. Although the LDA model had the worst results for sensitivity and specificity the model had the highest consistency across the classes with the range of its results spanning 0.204 compared to the 0.432 of the random forest and 0.3294 of the KNN. The AUC for the LDA model is high indicating that the model would be able to differentiate between classes and correctly classify a prediction as true positive or true negative 80.87% of the time.

Although the results could be seen as low across the models, as there are 6 classes, if the models were truly random in their predictions the sensitivity and specificity would have been much lower. If the models were random the sensitivity and specificity would have been 0.167 and 0.833 as 1/6 of the positive predictions would have been true positives and 5/6 of the negative predictions would have been true negatives. These values served as the benchmark for what the performance of the models should have been measured against. As all the model's results exceeded these benchmarks for all classes, they can be said to have had some level of predictive power. This is supported by the AUC for all three models. Even though the sensitivity was low, the AUC was still quite high for all models. If the models were random then the AUC would have been 0.5. Although the predictions are not as bad as could at first be expected, the level is still lower than what could be considered optimal. The low sensitivity across the three models could indicate that further feature tuning or model tuning could be required to receive a more accurate result. The low sensitivity scores for the latin, pop and r&b classes across all three models indicated that there may not be enough variance in the data for these classes for the models to differentiate them from other classes. This can be supported when looking at the confusion matrix for each model. The confusion matrices indicated that all the models had difficulty differentiating the latin, pop, and r&b classes from each other.

Conclusion

The purpose of this project was to create a predictive model that Spotify could use to predict a song's genre. The data used to create the predictive model originally consisted of 16 predictive variables which consisted of the characteristics of a song. This number was brought down to 11 predictive variables after removing the variables that were lest important in predicting the genre of a song. There were several steps involved in this project which included data importing and cleaning, exploratory data analysis, split and preprocessing data, model tuning and fitting, and lastly model evaluation. During this project, the founders also requested answers to several questions regarding some of the characteristics of the data. The answers to these questions are:

- The most important factors when predicting a song's genre were the year that the song was released,

the danceability, the tempo, the energy and the speechiness of a song, in that order.
- The popularity of songs did differ between genres with the pop and latin genres having, on average, the most popular song, and the edm having, on average, the least popular songs.
- There was a speechiness difference between genres with the rap genre having by far the most speechines out of all the genres. The genres with the least level of speechiness were the rock, pop and edm genres.
- The popularity of genres had changed over time. Over the long term, there had been a general downward trend in the popularity of songs. However, from 2010 to 2020 there was a noticeable upward trend in the popularity of nearly all music genres.

The results of the predictive models indicated that the random forest model had the greatest predictive power. The results showed that the random forest was able to correctly classify 55.53% of its positive predictions as true positives and that it would be able to correctly classify 91.11% of its negative predictions as true negatives. The model was also able to generate an AUC score of 85.02% which indicated that it could differentiate between the classes with 85.02% certainty. The linear discriminant model and k-nearest neighbour models scored slightly lower, with sensitivities and specificities of 46.47% and 89.29% for the LDA and 49.87% and 89.97% for the KNN model. The AUC scores for these models are 80.87% and 81.48% respectively. Analysing the confusion matrix, and the sensitivity and specificity scores for the individual classes showed that there was a lot of variance between the classes. The models were all able to correctly predict the edm, rap and rock classes more often than the latin, pop and r&b classes. This indicated that there could be too little variance in these classes in the original data for the models to differentiate between the classes. Future work to fix this could be to subject the data to additional feature engineering as this could allow for more variance to be uncovered which would allow the models to better differentiate the classes from each other. Based on these final results it is recommended that Spotify use the random forest model to predict the genre of a song over the LDA and KNN models.

Appendix A - Import/Cleaning/EDA

Data Importing

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0      v purrr   0.3.5
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(lubridate, warn.conflicts = FALSE)
```

```
## Loading required package: timechange
```

```
library(stringr)
library(tidyr)

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

```
## 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.
```

```
as_tibble(spotify_songs)
```

```
## # A tibble: 32,833 x 23
##    track_id      track~1 track~2 track~3 track~4 track~5 track~6 playl~7 playl~8
##    <chr>         <chr>   <chr>     <dbl> <chr>   <chr>   <chr>   <chr>   <chr>
##  1 6f807x0ima9a~ I Don'~ Ed She~      66 2oCs0D~ I Don'~ 2019-0~ Pop Re~ 37i9dQ~
##  2 0r7CVbZTWZgb~ Memori~ Maroon~      67 63rPS0~ Memori~ 2019-1~ Pop Re~ 37i9dQ~
##  3 1z1Hg7Vb0AhH~ All th~ Zara L~      70 1HoSmj~ All th~ 2019-0~ Pop Re~ 37i9dQ~
##  4 75FpbthrwQmz~ Call Y~ The Ch~      60 1nqYsO~ Call Y~ 2019-0~ Pop Re~ 37i9dQ~
##  5 1e8PAfcKUYoK~ Someon~ Lewis ~      69 7m7vv9~ Someon~ 2019-0~ Pop Re~ 37i9dQ~
##  6 7fvUMiyapMsR~ Beauti~ Ed She~      67 2yiy9c~ Beauti~ 2019-0~ Pop Re~ 37i9dQ~
##  7 2OAylPUDDfwR~ Never ~ Katy P~      62 7INHYS~ Never ~ 2019-0~ Pop Re~ 37i9dQ~
##  8 6b1RNvAcJjQH~ Post M~ Sam Fe~      69 6703SR~ Post M~ 2019-0~ Pop Re~ 37i9dQ~
##  9 7bF6tCO3gFb8~ Tough ~ Avicii       68 7CvAfG~ Tough ~ 2019-0~ Pop Re~ 37i9dQ~
## 10 1IXGILkPm0tO~ If I C~ Shawn ~      67 4Qxzbf~ If I C~ 2019-0~ Pop Re~ 37i9dQ~
## # ... with 32,823 more rows, 14 more variables: 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>, and abbreviated variable names 1: track_name,
## #   2: track_artist, 3: track_popularity, 4: track_album_id,
## #   5: track_album_name, 6: track_album_release_date, 7: playlist_name, ...
```

```
head(spotify_songs)
```

```
## # A tibble: 6 x 23
##   track_id       track~1 track~2 track~3 track~4 track~5 track~6 playl~7 playl~8
##   <chr>          <chr>   <chr>     <dbl> <chr>   <chr>   <chr>   <chr>   <chr>
## 1 6f807x0ima9a1~ I Don'~ Ed She~      66 2oCs0D~ I Don'~ 2019-0~ Pop Re~ 37i9dQ~
## 2 0r7CVbZTWZgbT~ Memori~ Maroon~      67 63rPS0~ Memori~ 2019-1~ Pop Re~ 37i9dQ~
## 3 1z1Hg7Vb0AhHD~ All th~ Zara L~      70 1HoSmj~ All th~ 2019-0~ Pop Re~ 37i9dQ~
## 4 75FpbthrwQmzH~ Call Y~ The Ch~      60 1nqYsO~ Call Y~ 2019-0~ Pop Re~ 37i9dQ~
## 5 1e8PAfcKUYoKk~ Someon~ Lewis ~      69 7m7vv9~ Someon~ 2019-0~ Pop Re~ 37i9dQ~
## 6 7fvUMiyapMsRR~ Beauti~ Ed She~      67 2yiy9c~ Beauti~ 2019-0~ Pop Re~ 37i9dQ~
## # ... with 14 more variables: 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>, and
## #   abbreviated variable names 1: track_name, 2: track_artist,
## #   3: track_popularity, 4: track_album_id, 5: track_album_name,
## #   6: track_album_release_date, 7: playlist_name, 8: playlist_id
```

```
songs <- spotify_songs[, c('energy','key','loudness','mode','acousticness','instrumentalness',
  'liveness', 'valence','duration_ms','track_popularity', 'track_album_release_date',
  'playlist_genre', 'danceability', 'speechiness', 'tempo')]

head(songs)
```

```
## # A tibble: 6 x 15
##   energy   key loudness  mode acoustic~1 instr~2 liven~3 valence durat~4 track~5
##    <dbl> <dbl>    <dbl> <dbl>      <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1  0.916     6    -2.63     1     0.102  0        0.0653   0.518  194754      66
## 2  0.815    11    -4.97     1     0.0724 4.21e-3  0.357    0.693  162600      67
## 3  0.931     1    -3.43     0     0.0794 2.33e-5  0.11     0.613  176616      70
## 4  0.93      7    -3.78     1     0.0287 9.43e-6  0.204    0.277  169093      60
## 5  0.833     1    -4.67     1     0.0803 0        0.0833   0.725  189052      69
## 6  0.919     8    -5.38     1     0.0799 0        0.143    0.585  163049      67
## # ... with 5 more variables: track_album_release_date <chr>,
## #   playlist_genre <chr>, danceability <dbl>, speechiness <dbl>, tempo <dbl>,
## #   and abbreviated variable names 1: acousticness, 2: instrumentalness,
## #   3: liveness, 4: duration_ms, 5: track_popularity
```

Data Cleaning

```
pacman::p_load(skimr)
skim(songs)
```

Table 1: Data summary

Name	songs
Number of rows	32833
Number of columns	15
Column type frequency:	
character	2
numeric	13
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	4530	0
playlist_genre	0	1	3	5	0	6	0

Variable type: numeric

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	

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
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	
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	
duration_ms	0	1	225799.81	59834.01	4000.00	187819.00	216000.00	253585.00	517810.00	
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	
speechiness	0	1	0.11	0.10	0.00	0.04	0.06	0.13	0.92	
tempo	0	1	120.88	26.90	0.00	99.96	121.98	133.92	239.44	

```
# Getting Date data
pacman::p_load(recipes)
pacman::p_load(inspectdf)
pacman::p_load(dplyr)

songs$date <- as_date(songs$track_album_release_date)
```

```
## Warning: 1886 failed to parse.
```

```
inspect_na(songs)
```

```
## # A tibble: 16 x 3
##    col_name                  cnt  pcnt
##    <chr>                   <int> <dbl>
##  1 date                     1886  5.74
##  2 energy                      0  0
##  3 key                         0  0
##  4 loudness                    0  0
##  5 mode                        0  0
##  6 acousticness                0  0
##  7 instrumentalness            0  0
##  8 liveness                    0  0
##  9 valence                     0  0
## 10 duration_ms                 0  0
## 11 track_popularity            0  0
## 12 track_album_release_date    0  0
## 13 playlist_genre              0  0
## 14 danceability                0  0
## 15 speechiness                 0  0
## 16 tempo                       0  0
```

```
songs <- songs %>% drop_na() #Drop na instead of replacing as it is not a significant amount of data an

songs$year <- year(songs$date)
songs$month <- month(songs$date)
songs$day <- wday(songs$date)
```

```
songs <- dplyr::select(songs, -date, -track_album_release_date)
head(songs)
```

```
## # A tibble: 6 x 17
##   energy   key loudness  mode acoustic~1 instr~2 liven~3 valence durat~4 track~5
##    <dbl> <dbl>    <dbl> <dbl>      <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1  0.916     6    -2.63     1     0.102  0        0.0653   0.518  194754      66
## 2  0.815    11    -4.97     1     0.0724 4.21e-3  0.357    0.693  162600      67
## 3  0.931     1    -3.43     0     0.0794 2.33e-5  0.11     0.613  176616      70
## 4  0.93      7    -3.78     1     0.0287 9.43e-6  0.204    0.277  169093      60
## 5  0.833     1    -4.67     1     0.0803 0        0.0833   0.725  189052      69
## 6  0.919     8    -5.38     1     0.0799 0        0.143    0.585  163049      67
## # ... with 7 more variables: playlist_genre <chr>, danceability <dbl>,
## #   speechiness <dbl>, tempo <dbl>, year <dbl>, month <dbl>, day <dbl>, and
## #   abbreviated variable names 1: acousticness, 2: instrumentalness,
## #   3: liveness, 4: duration_ms, 5: track_popularity
```

```
# Replacing Loudness incorrect values with median
songs %>% count(loudness > 0)
```

```
## # A tibble: 2 x 2
##   `loudness > 0`     n
##   <lgl>          <int>
## 1 FALSE          30941
## 2 TRUE               6
```

```
songs %>% count(loudness == median(songs$loudness))
```

```
## # A tibble: 2 x 2
##   `loudness == median(songs$loudness)`     n
##   <lgl>                                <int>
## 1 FALSE                                30942
## 2 TRUE                                     5
```

```
songs$loudness <- ifelse(songs$loudness > 0, median(songs$loudness), songs$loudness)
songs$loudness %>% median()
```

```
## [1] -6.093
```

```
songs %>% count(loudness == median(songs$loudness))
```

```
## # A tibble: 2 x 2
##   `loudness == median(songs$loudness)`     n
##   <lgl>                                <int>
## 1 FALSE                                30936
## 2 TRUE                                    11
```

```
# Downsampling data to 1000 of each Genre
pacman::p_load(rsample)
pacman::p_load(themis)
set.seed(1805361)

songs %>% count(playlist_genre)
```

```
## # A tibble: 6 x 2
##   playlist_genre     n
##   <chr>          <int>
## 1 edm             5969
## 2 latin           4963
## 3 pop             5303
## 4 r&b             5094
## 5 rap             5471
## 6 rock            4147
```

```
downsample_recipe <- recipe(playlist_genre ~., data = songs) %>%
  themis::step_downsample(playlist_genre, under_ratio = 0.2413)

downsample_prepped <- prep(downsample_recipe)
songs <- juice(downsample_prepped)
summary(songs$playlist_genre)
```

```
##   edm latin   pop   r&b   rap  rock
##  1000  1000  1000  1000  1000  1000
```

```
# Recoding factors - Other factors will be recoded below for graphing reasons
songs$playlist_genre <- as_factor(songs$playlist_genre)
```

Exploratory Data Analysis

```
# Plots
ggplot(data = songs, aes(x= energy, fill = playlist_genre)) +
  geom_histogram() + facet_grid(playlist_genre ~.) +
  labs(caption = "Figure 1 - Energy Histogram") +
  theme(plot.caption = element_text(hjust = 0.5))
```

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

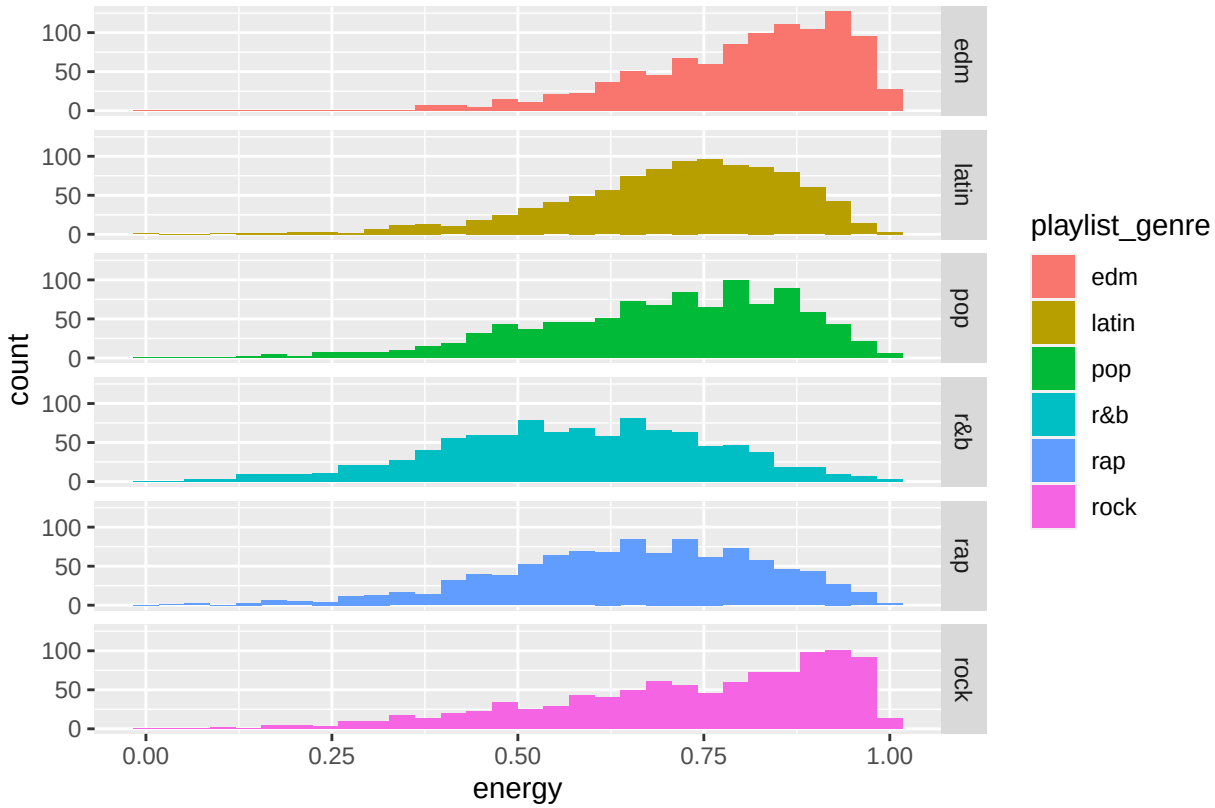

Figure 1 − Energy Histogram

```
summary(songs$energy)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.000175 0.574000 0.717000 0.695048 0.840000 1.000000
```

```
ggplot(data = songs, aes(x = key, fill = playlist_genre)) +
  geom_bar() + facet_grid(playlist_genre ~.) +
  labs(caption = "Figure 2 - Key Bar Chart") +
  theme(plot.caption = element_text(hjust = 0.5))
```

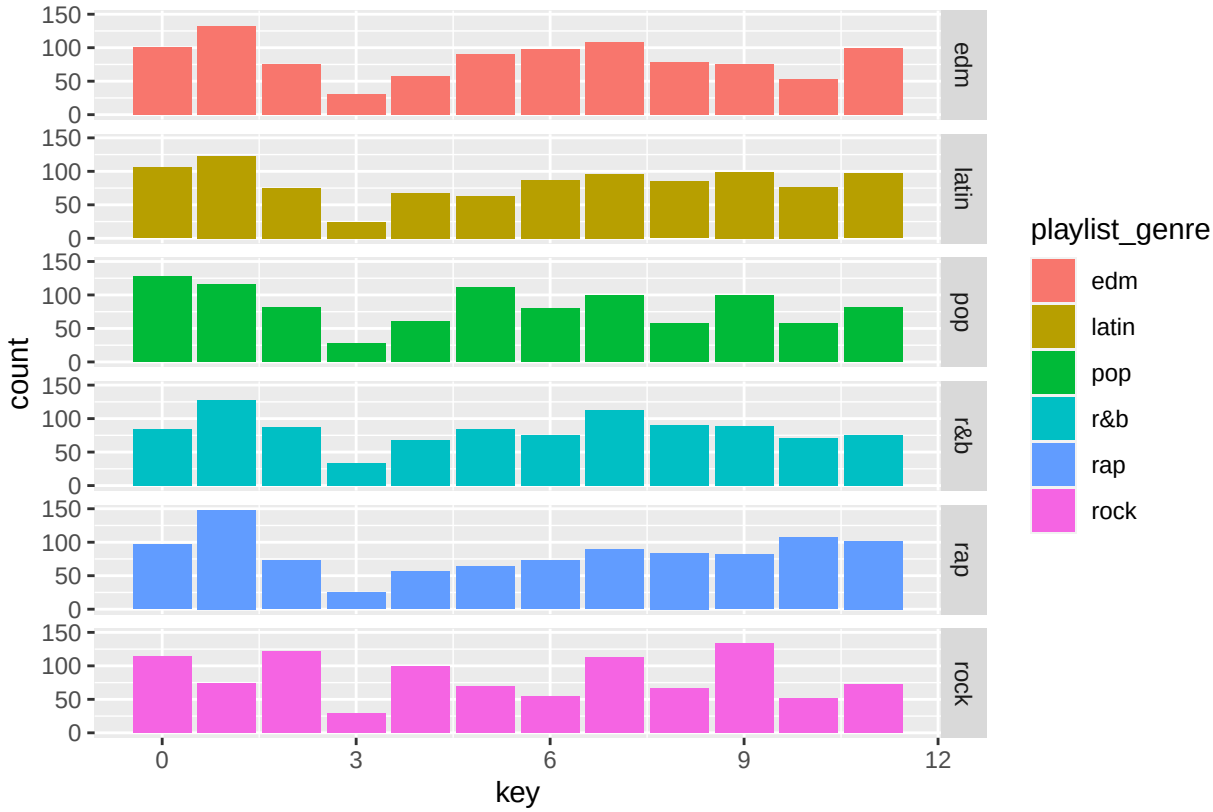

Figure 2 − Key Bar Chart

```
summary(songs$key)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   2.000   6.000   5.389   9.000  11.000
```

```
ggplot(data = songs, aes(x = loudness, fill = playlist_genre)) +
  geom_histogram() + facet_grid(playlist_genre ~.) +
  labs(caption = "Figure 3 - Loudness Histogram") +
  theme(plot.caption = element_text(hjust = 0.5))
```

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

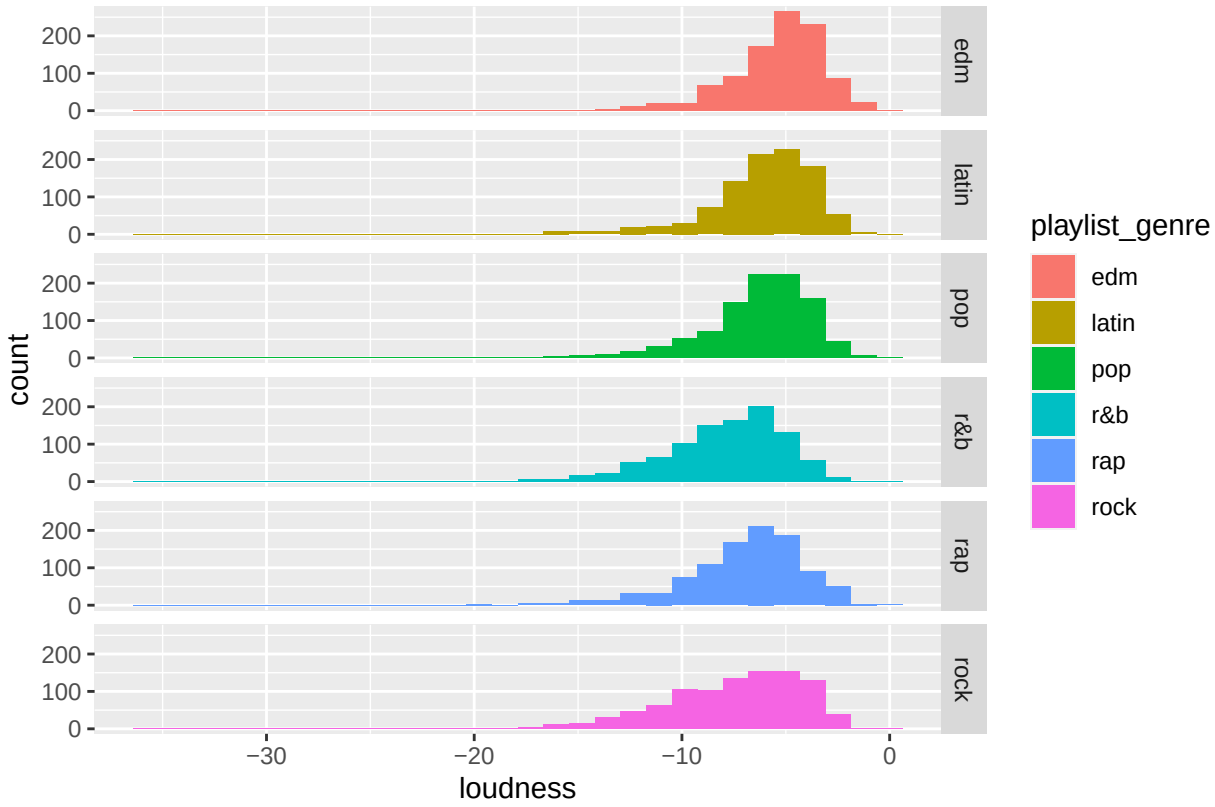

Figure 3 − Loudness Histogram

```
summary(songs$loudness)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -35.960  -8.191  -6.180  -6.741  -4.662  -0.155
```

```
ggplot(data = songs, aes(x = mode, fill = playlist_genre)) +
  geom_bar() + facet_grid(playlist_genre ~.) +
  labs(caption = "Figure 4 - Mode Bar Chart") +
  theme(plot.caption = element_text(hjust = 0.5))
```

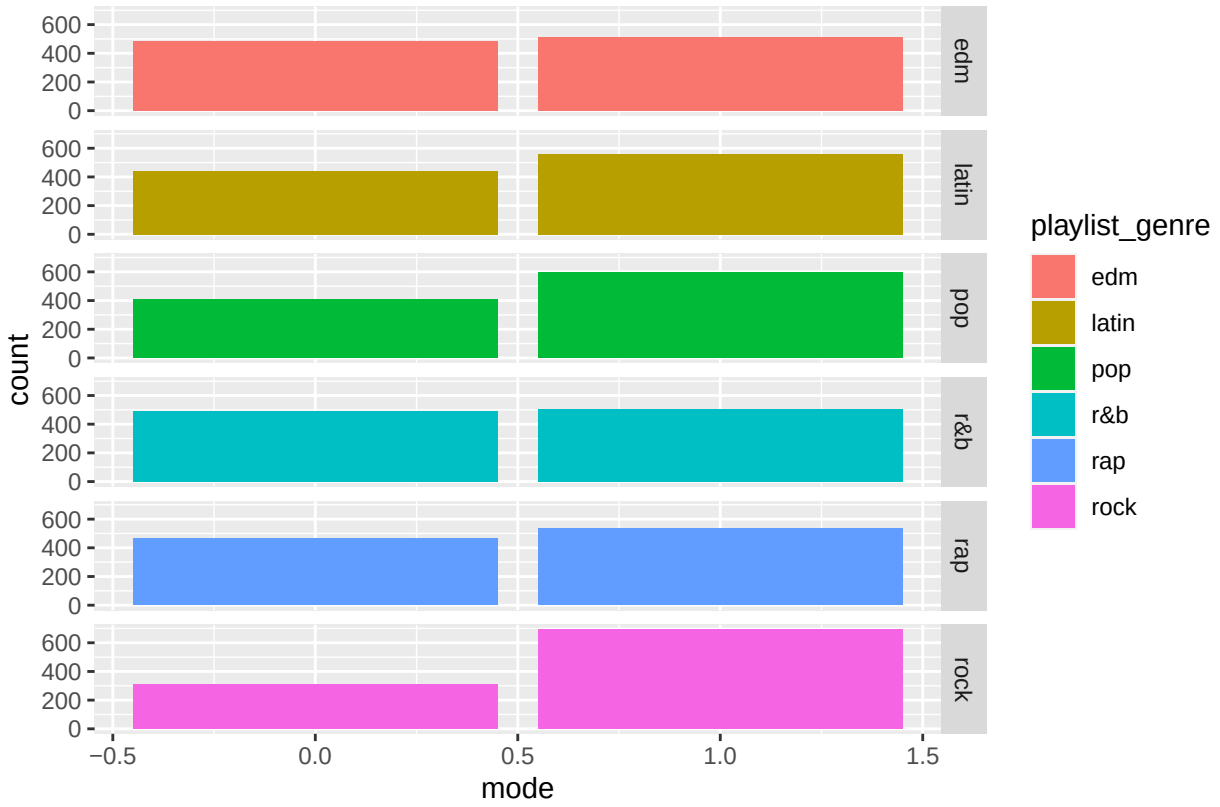

Figure 4 − Mode Bar Chart

```
summary(songs$mode)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  1.0000  0.5667  1.0000  1.0000
```

```
ggplot(data = songs, aes(x = acousticness, fill = playlist_genre)) +
  geom_histogram() + facet_grid(playlist_genre ~.) +
  labs(caption = "Figure 5 - Acousticness Histogram") +
  theme(plot.caption = element_text(hjust = 0.5))
```

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

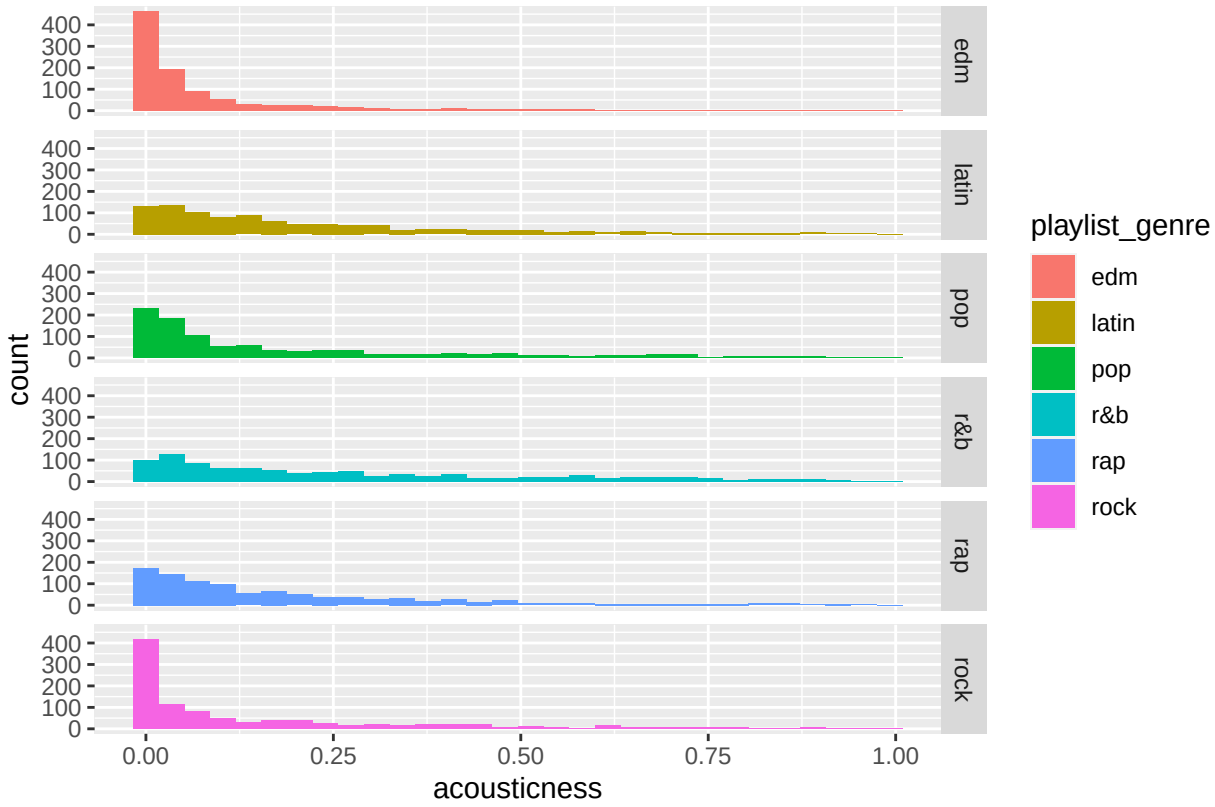

Figure 5 − Acousticness Histogram

```
summary(songs$acousticness)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0168  0.0856  0.1817  0.2690  0.9920
```

```
ggplot(data = songs, aes(x = instrumentalness, fill = playlist_genre)) +
  geom_histogram() + facet_grid(playlist_genre ~.) +
  labs(caption = "Figure 6 - Instrumentalness Histogram") +
  theme(plot.caption = element_text(hjust = 0.5))
```

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

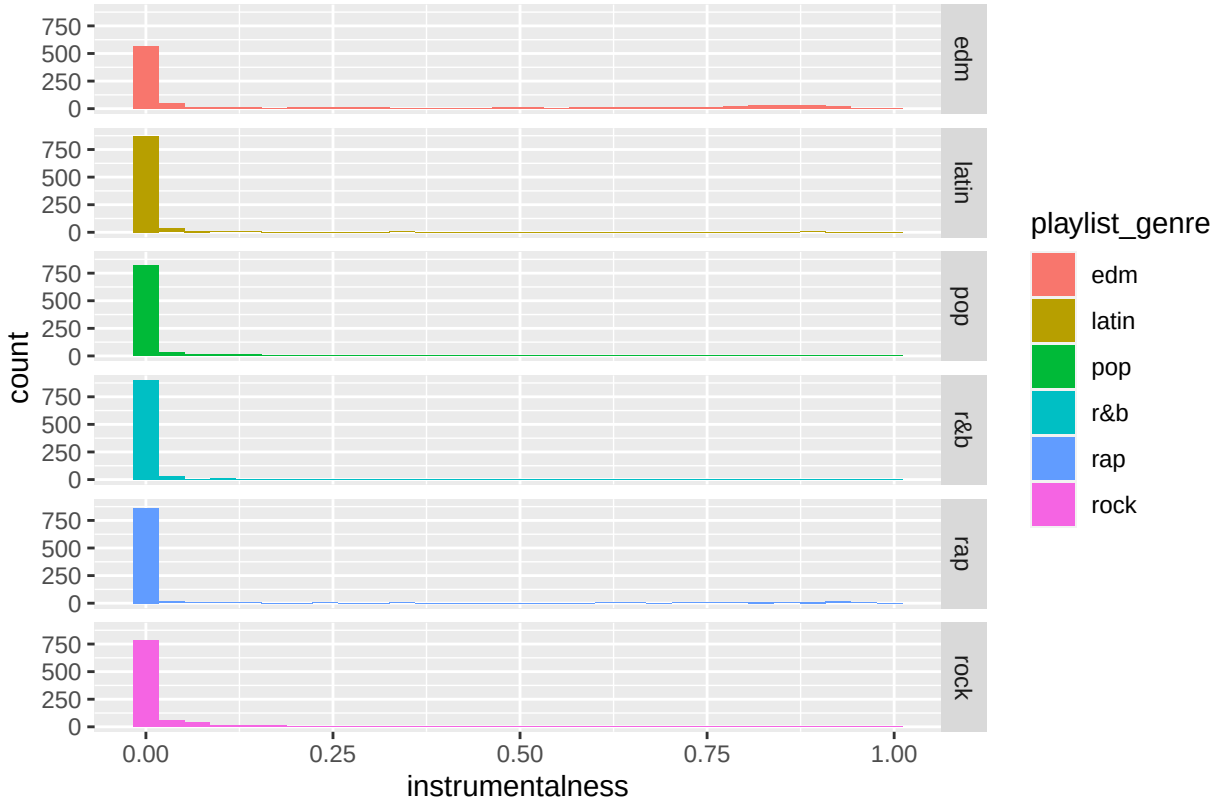

Figure 6 − Instrumentalness Histogram

```
summary(songs$instrumentalness)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.0000000 0.0000000 0.0000164 0.0831157 0.0044500 0.9940000
```

```
ggplot(data = songs, aes(x = liveness, fill = playlist_genre)) +
  geom_histogram() + facet_grid(playlist_genre ~.) +
  labs(caption = "Figure 7 - Liveness Histogram") +
  theme(plot.caption = element_text(hjust = 0.5))
```

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

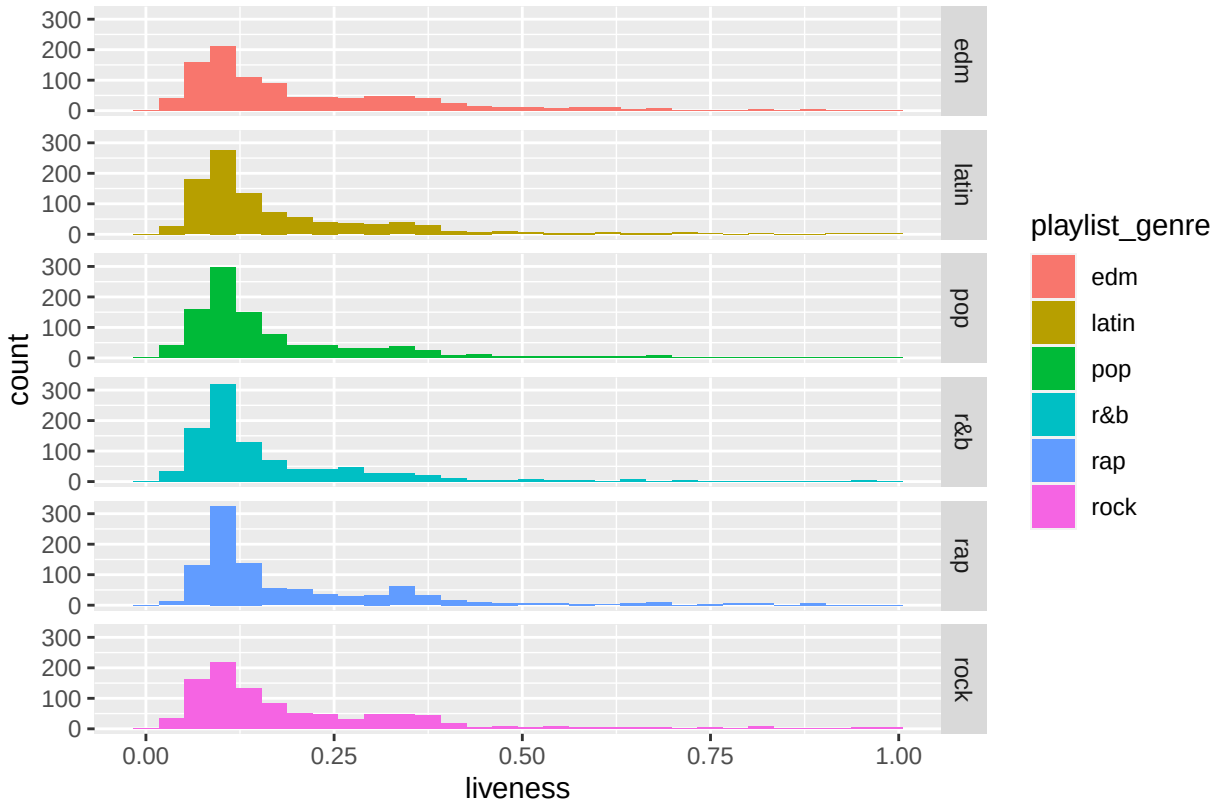

Figure 7 − Liveness Histogram

```
summary(songs$liveness)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0929  0.1260  0.1910  0.2440  0.9880
```

```
ggplot(data = songs, aes(x = valence, fill = playlist_genre)) +
  geom_histogram() + facet_grid(playlist_genre ~.) +
  labs(caption = "Figure 8 - Valence Histogram") +
  theme(plot.caption = element_text(hjust = 0.5))
```

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

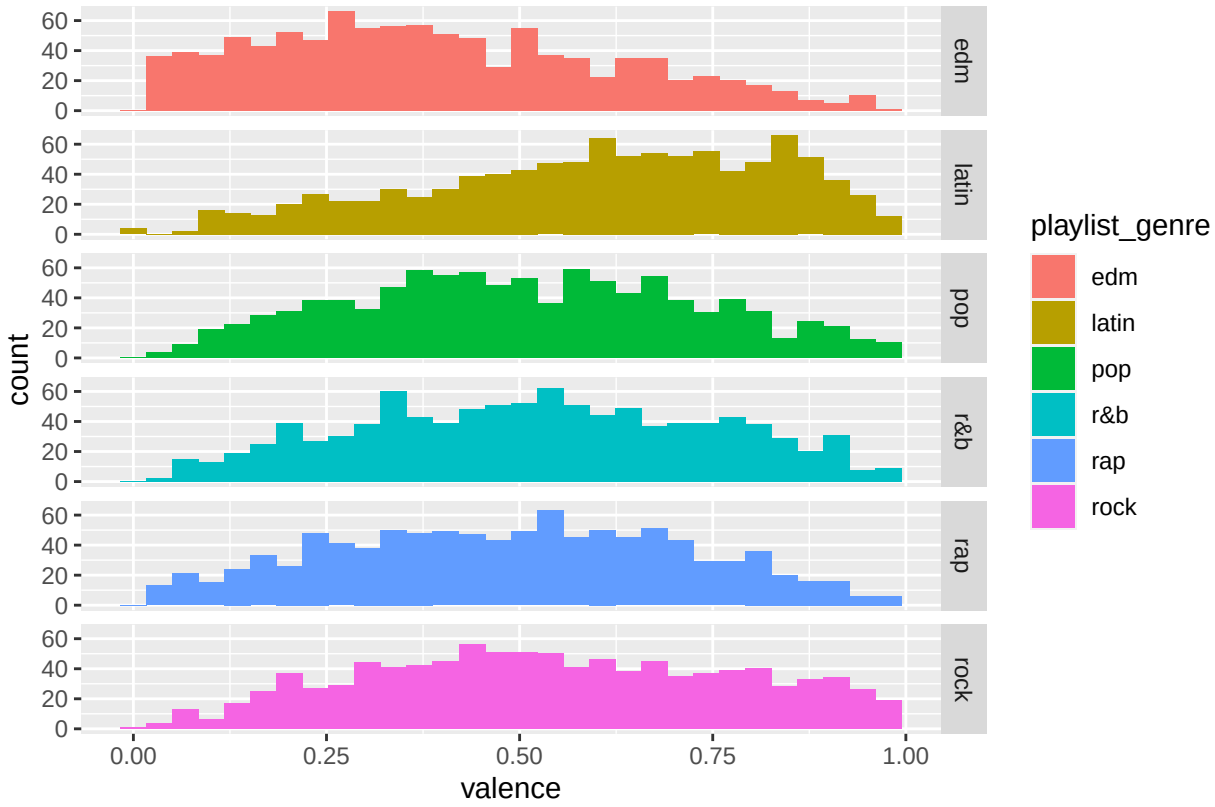

Figure 8 − Valence Histogram

```
summary(songs$valence)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.3270  0.5080  0.5066  0.6870  0.9780
```

```
ggplot(data = songs, aes(x = duration_ms, fill = playlist_genre)) +
  geom_histogram() + facet_grid(playlist_genre ~.) +
  labs(caption = "Figure 9 - Duration Histogram") +
  theme(plot.caption = element_text(hjust = 0.5))
```

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

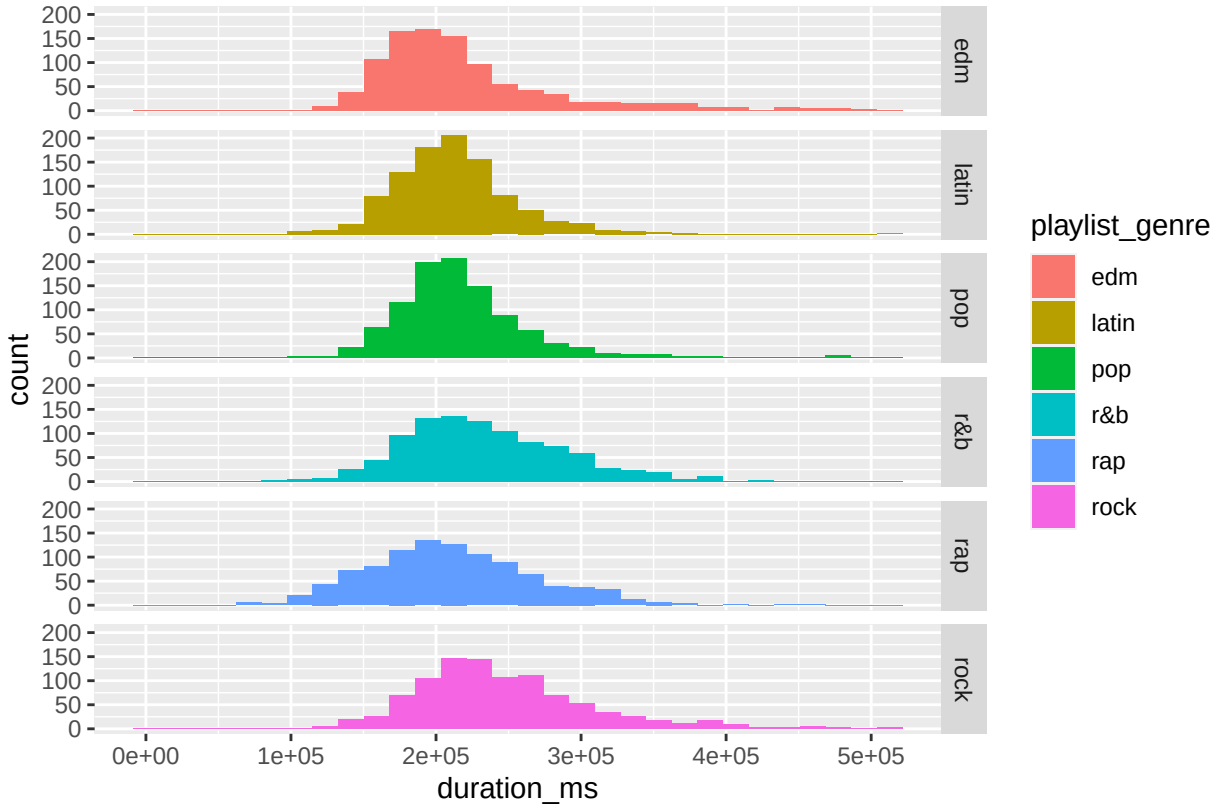

Figure 9 − Duration Histogram

```
summary(songs$duration_ms)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    4000  187290  215046  224325  251442  516760
```

```
ggplot(data = songs, aes(x = track_popularity, fill = playlist_genre)) +
  geom_histogram() + facet_grid(playlist_genre ~.) +
  labs(caption = "Figure 10 - Track Popularity Histogram") +
  theme(plot.caption = element_text(hjust = 0.5))
```

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

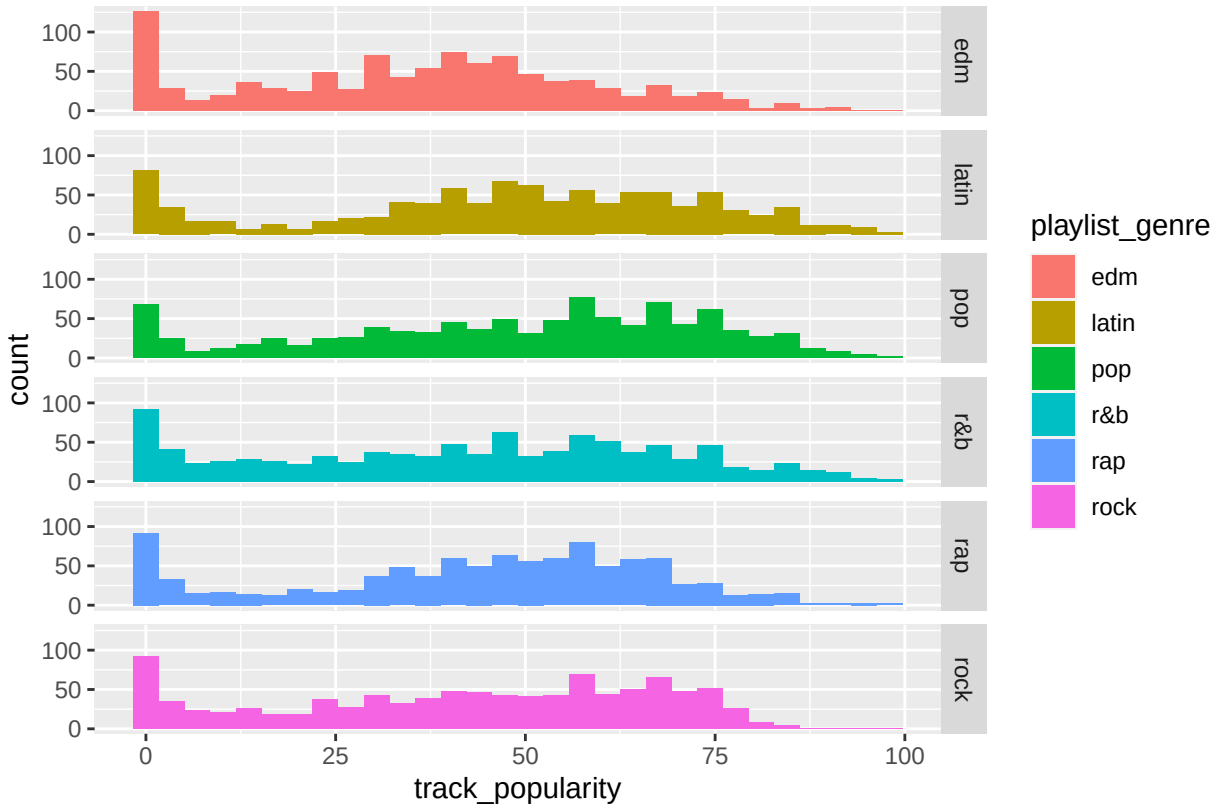

Figure 10 − Track Popularity Histogram

```
summary(songs$track_popularity)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0    26.0    46.0    43.3    63.0    98.0
```

```
ggplot(data = songs, aes(x = danceability, fill = playlist_genre)) +
  geom_histogram() + facet_grid(playlist_genre ~.) +
  labs(caption = "Figure 11 - Danceability Histogram") +
  theme(plot.caption = element_text(hjust = 0.5))
```

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

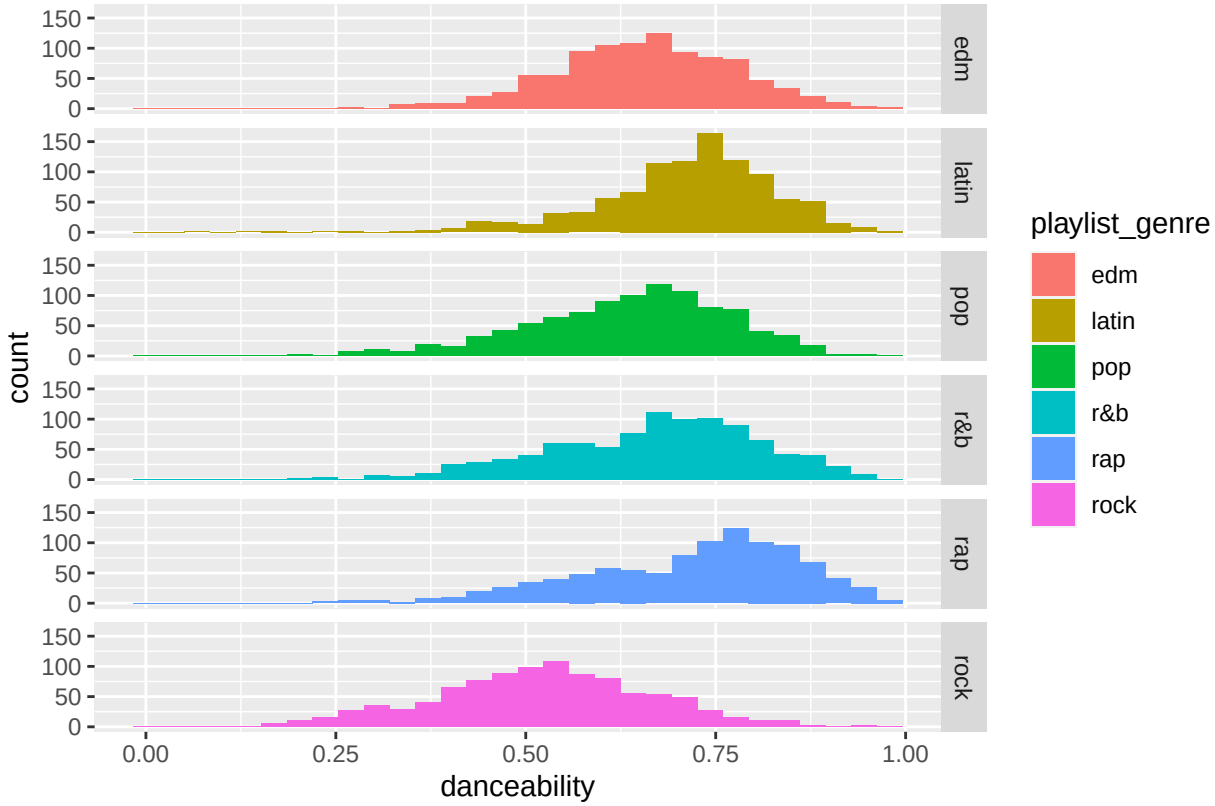

Figure 11 − Danceability Histogram

```
summary(songs$danceability)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.5550  0.6690  0.6516  0.7600  0.9790
```

```
ggplot(data = songs, aes(x = speechiness, fill = playlist_genre)) +
  geom_histogram() + facet_grid(playlist_genre ~.) +
  labs(caption = "Figure 12 - Speechiness Histogram") +
  theme(plot.caption = element_text(hjust = 0.5))
```

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

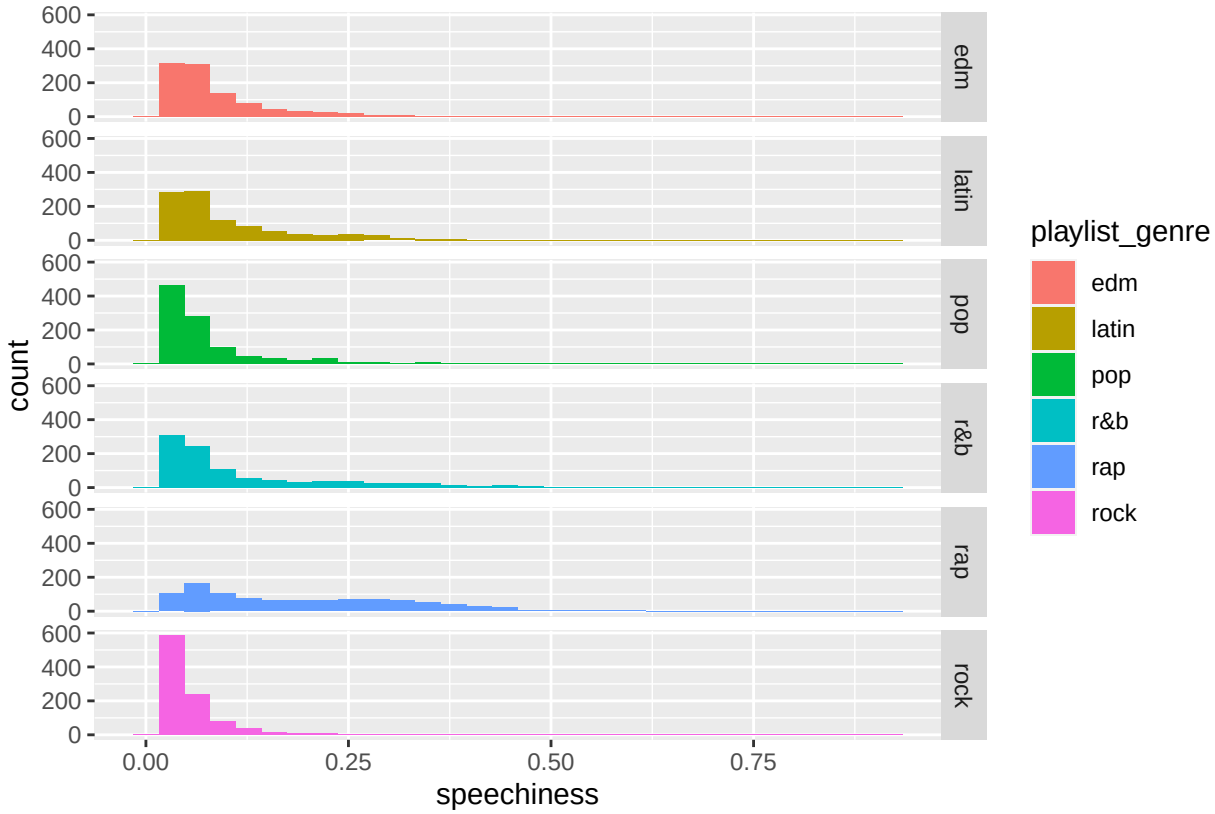

Figure 12 − Speechiness Histogram

```
summary(songs$speechiness)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0407  0.0635  0.1065  0.1300  0.9180
```

```
ggplot(data = songs, aes(x = year, fill = playlist_genre)) +
  geom_bar() + facet_grid(playlist_genre ~.) +
  labs(caption = "Figure 13 - Year Bar Chart") +
  theme(plot.caption = element_text(hjust = 0.5))
```

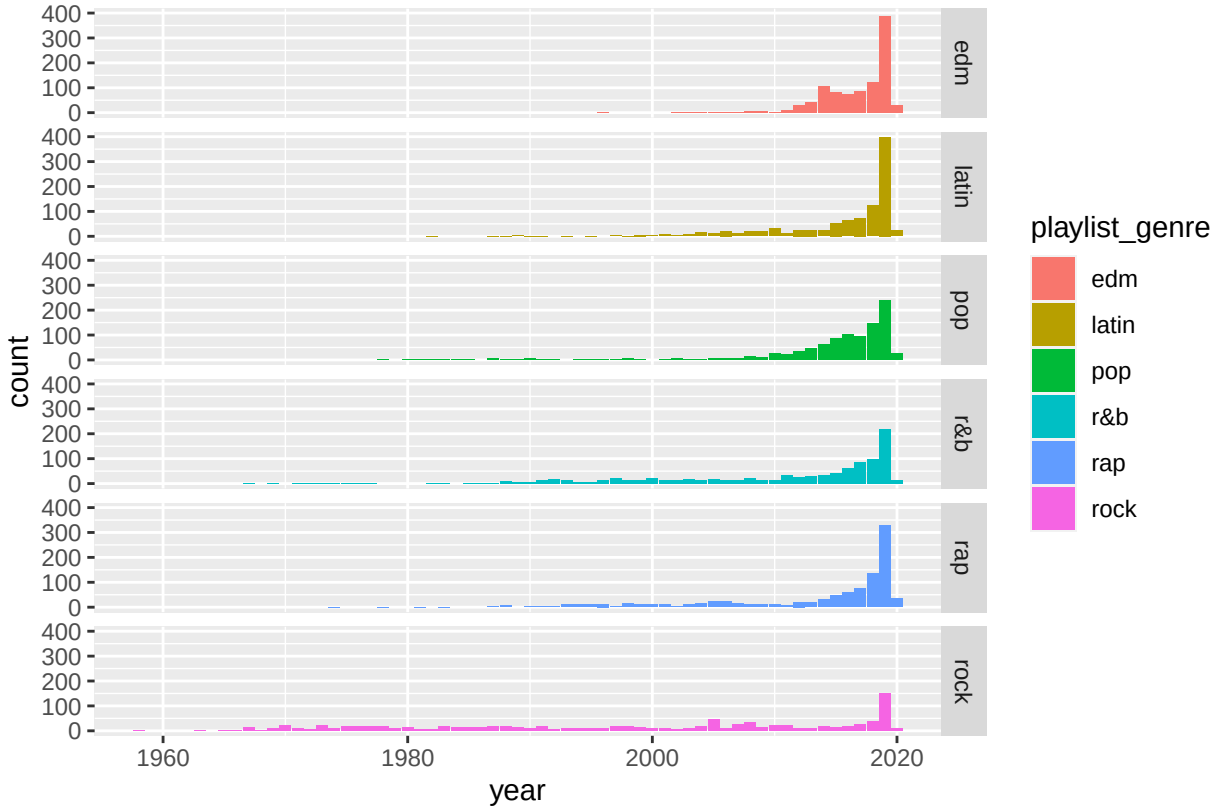

Figure 13 − Year Bar Chart

```
summary(songs$year)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1958    2009    2016    2012    2019    2020
```

```
ggplot(data = songs, aes(x = month, fill = playlist_genre)) +
  geom_bar() + facet_grid(playlist_genre ~.) +
  labs(caption = "Figure 14 - Month Bar Chart") +
  theme(plot.caption = element_text(hjust = 0.5))
```

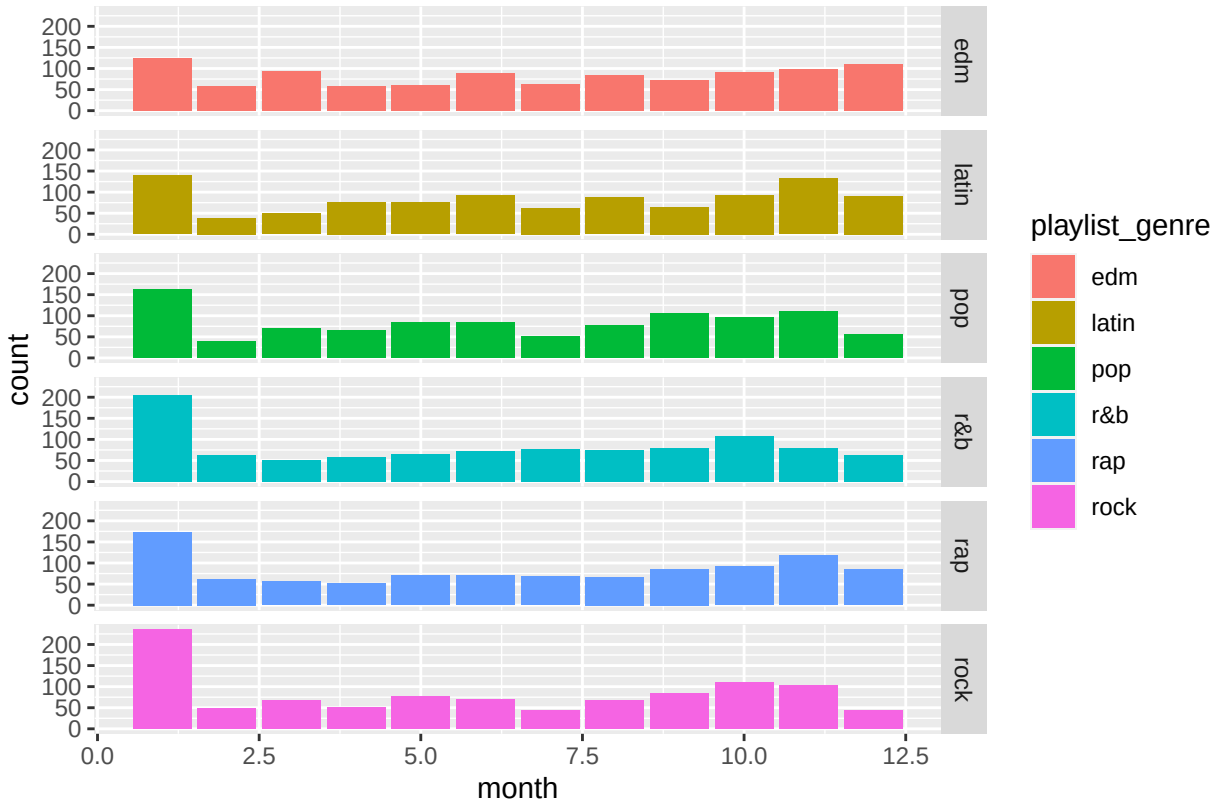

Figure 14 − Month Bar Chart

```
summary(songs$month)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   3.000   6.000   6.375  10.000  12.000
```

```
ggplot(data = songs, aes(x = day, fill = playlist_genre)) +
  geom_bar() + facet_grid(playlist_genre ~.) +
  labs(caption = "Figure 15 - Day Bar Chart") +
  theme(plot.caption = element_text(hjust = 0.5))
```

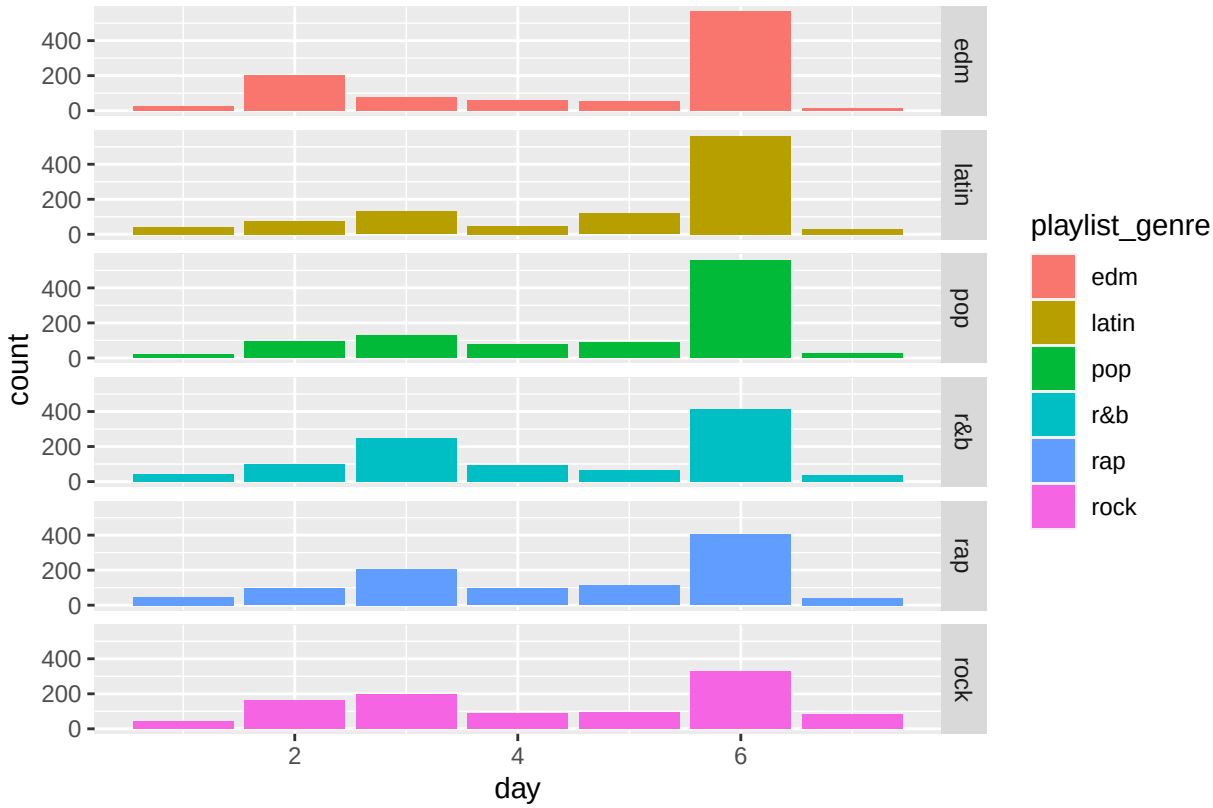

Figure 15 − Day Bar Chart

```
summary(songs$day)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   3.000   6.000   4.633   6.000   7.000
```

Founder Questions

```
# Does the popularity of songs differ between genres
ggplot(data = songs, aes(x = playlist_genre, y = track_popularity, fill = playlist_genre)) +
  geom_boxplot() + labs(caption = "Figure 16 - Track Popularity by Playlist Genre Box-Plot") +
  theme(plot.caption = element_text(hjust = 0.5))
```

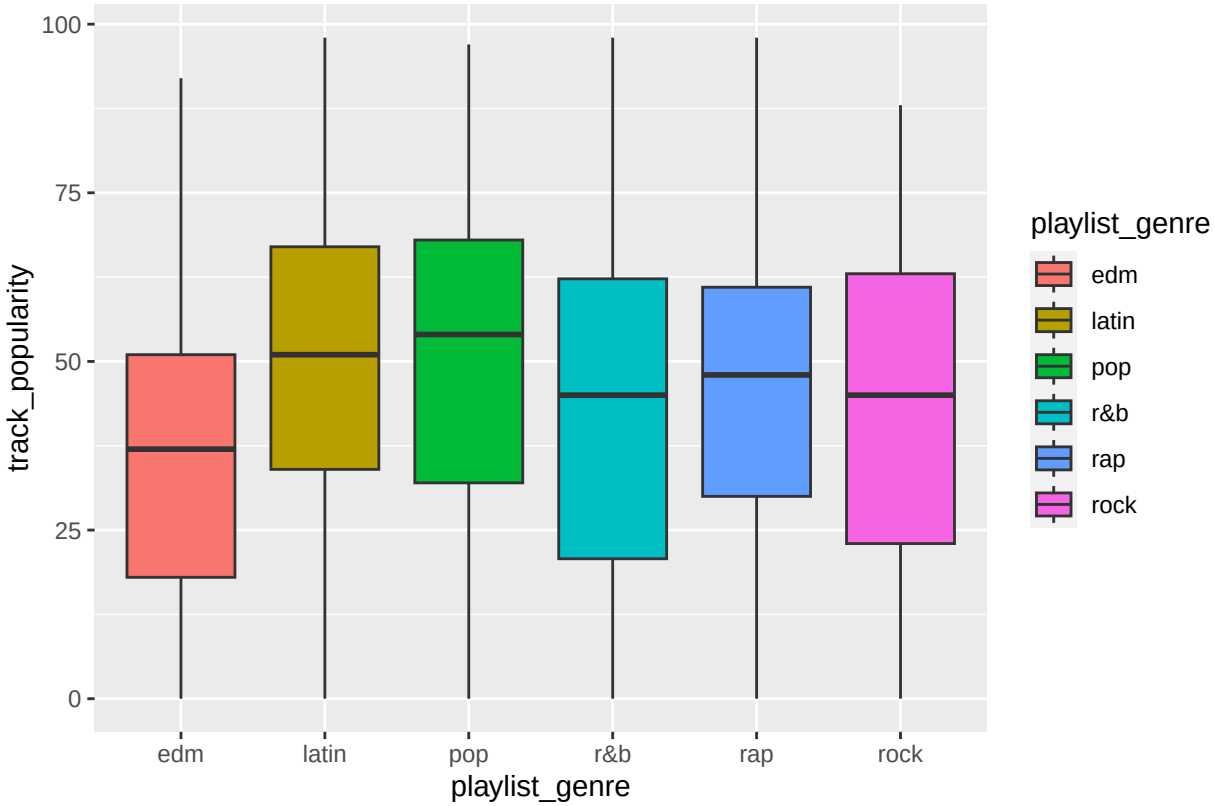

Figure 16 − Track Popularity by Playlist Genre Box−Plot

```
print("POP")
```

```
## [1] "POP"
```

```
songs %>% filter(playlist_genre == "pop") %>%
  dplyr::select(track_popularity) %>% summary()
```

```
##  track_popularity
##  Min.   : 0.00
##  1st Qu.:32.00
##  Median :54.00
##  Mean   :48.49
##  3rd Qu.:68.00
##  Max.   :97.00
```

```
print("RAP")
```

```
## [1] "RAP"
```

```
songs %>% filter(playlist_genre == "rap") %>%
  dplyr::select(track_popularity) %>% summary()
```

```
##  track_popularity
```

```
##  Min.   : 0.00
##  1st Qu.:30.00
##  Median :48.00
##  Mean   :43.44
##  3rd Qu.:61.00
##  Max.   :98.00
```

```
print("ROCK")
```

```
## [1] "ROCK"
```

```
songs %>% filter(playlist_genre == "rock") %>%
  dplyr::select(track_popularity) %>% summary()
```

```
##  track_popularity
##  Min.   : 0.00
##  1st Qu.:23.00
##  Median :45.00
##  Mean   :42.01
##  3rd Qu.:63.00
##  Max.   :88.00
```

```
print("LATIN")
```

```
## [1] "LATIN"
```

```
songs %>% filter(playlist_genre == "latin") %>%
  dplyr::select(track_popularity) %>% summary()
```

```
##  track_popularity
##  Min.   : 0.00
##  1st Qu.:34.00
##  Median :51.00
##  Mean   :47.74
##  3rd Qu.:67.00
##  Max.   :98.00
```

```
print("R&B")
```

```
## [1] "R&B"
```

```
songs %>% filter(playlist_genre == "r&b") %>%
  dplyr::select(track_popularity) %>% summary()
```

```
##  track_popularity
##  Min.   : 0.00
##  1st Qu.:20.75
##  Median :45.00
##  Mean   :42.34
##  3rd Qu.:62.25
##  Max.   :98.00
```

```
print("EDM")
```

```
## [1] "EDM"
```

```
songs %>% filter(playlist_genre == "edm") %>%
  dplyr::select(track_popularity) %>% summary()
```

```
##  track_popularity
##  Min.   : 0.00
##  1st Qu.:18.00
##  Median :37.00
##  Mean   :35.76
##  3rd Qu.:51.00
##  Max.   :92.00
```

```
# Is there a difference in speechiness for each genre
ggplot(data = songs, aes(x = playlist_genre, y = speechiness, fill = playlist_genre)) +
  geom_boxplot() + labs(caption = "Figure 17 - Speechiness by Playlist Genre Boxplot") +
  theme(plot.caption = element_text(hjust = 0.5))
```

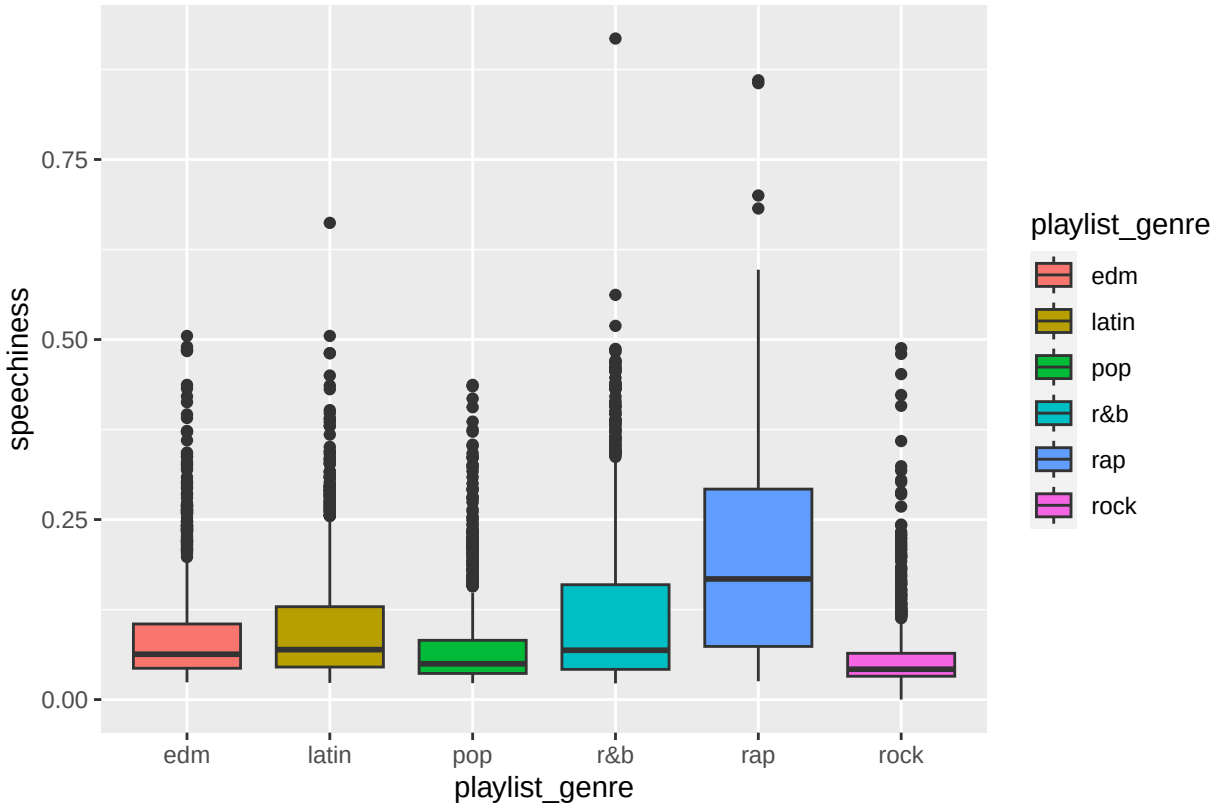

Figure 17 − Speechiness by Playlist Genre Boxplot

```
print("POP")
```

```
## [1] "POP"
```

```
songs %>% filter(playlist_genre == "pop") %>%
  dplyr::select(speechiness) %>% summary()
```

```
##   speechiness
##  Min.   :0.02280
##  1st Qu.:0.03630
##  Median :0.04975
##  Mean   :0.07405
##  3rd Qu.:0.08230
##  Max.   :0.43700
```

```
print("RAP")
```

```
## [1] "RAP"
```

```
songs %>% filter(playlist_genre == "rap") %>%
  dplyr::select(speechiness) %>% summary()
```

```
##   speechiness
##  Min.   :0.02550
##  1st Qu.:0.07383
##  Median :0.16750
##  Mean   :0.19298
##  3rd Qu.:0.29225
##  Max.   :0.86000
```

```
print("ROCK")
```

```
## [1] "ROCK"
```

```
songs %>% filter(playlist_genre == "rock") %>%
  dplyr::select(speechiness) %>% summary()
```

```
##   speechiness
##  Min.   :0.00000
##  1st Qu.:0.03240
##  Median :0.04215
##  Mean   :0.05961
##  3rd Qu.:0.06427
##  Max.   :0.48800
```

```
print("LATIN")
```

```
## [1] "LATIN"
```

```
songs %>% filter(playlist_genre == "latin") %>%
  dplyr::select(speechiness) %>% summary()
```

```
##   speechiness
##  Min.   :0.02320
##  1st Qu.:0.04517
##  Median :0.06935
##  Mean   :0.10260
##  3rd Qu.:0.12900
##  Max.   :0.66200
```

```
print("R&B")
```

```
## [1] "R&B"
```

```
songs %>% filter(playlist_genre == "r&b") %>%
  dplyr::select(speechiness) %>% summary()
```

```
##   speechiness
##  Min.   :0.02240
##  1st Qu.:0.04197
##  Median :0.06850
##  Mean   :0.11941
##  3rd Qu.:0.15950
##  Max.   :0.91800
```

```
print("EDM")
```

```
## [1] "EDM"
```

```
songs %>% filter(playlist_genre == "edm") %>%
  dplyr::select(speechiness) %>% summary()
```

```
##   speechiness
##  Min.   :0.02390
##  1st Qu.:0.04350
##  Median :0.06295
##  Mean   :0.09010
##  3rd Qu.:0.10500
##  Max.   :0.50500
```

```
# How does track popularity change over time
pop_songs <- songs %>% dplyr::select(track_popularity, year) %>%
  group_by(year) %>%
  summarise(track_popularity = mean(track_popularity))

pop_songs %>% ggplot(aes(x = year, y = track_popularity)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  labs(caption = "Figure 18 - Track Popularity by Year Scatter Plot with Linear Model")
```

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

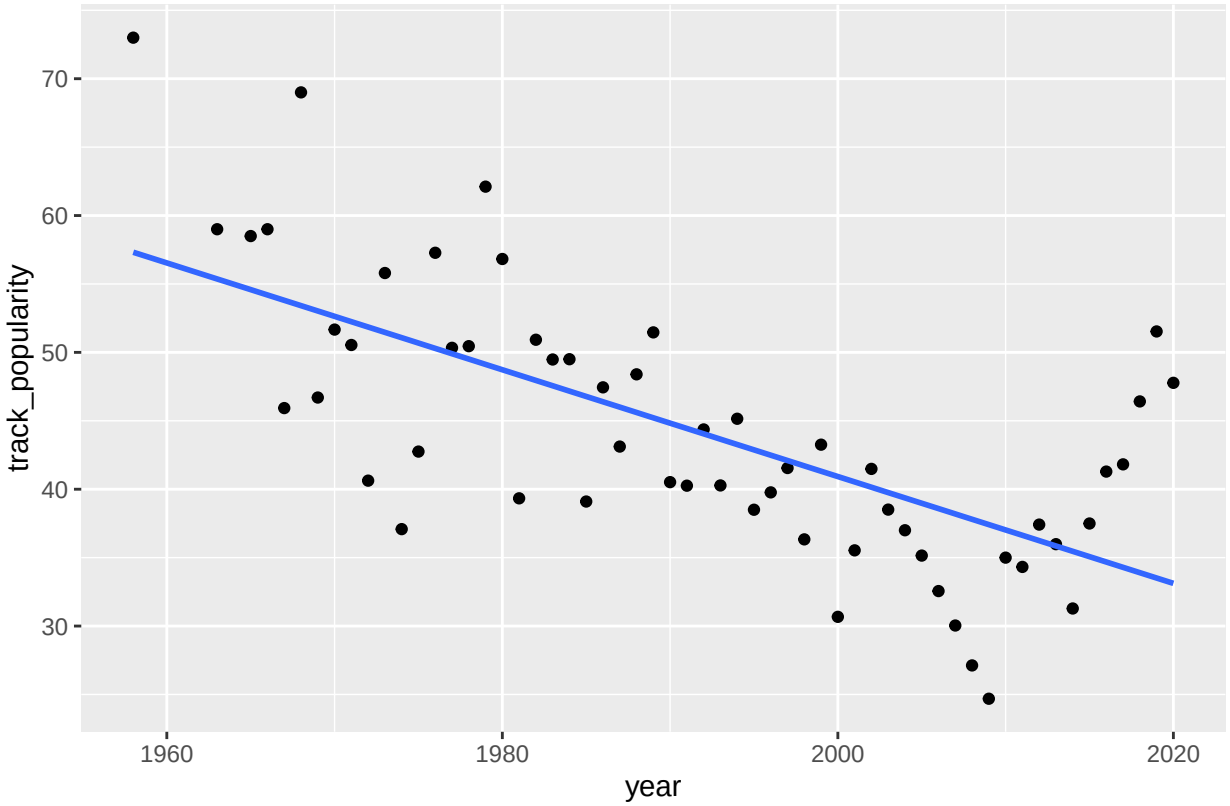

Figure 18 − Track Popularity by Year Scatter Plot with Linear Model

```
summary(lm(data = pop_songs, track_popularity ~ year))
```

```
##
## Call:
## lm(formula = track_popularity ~ year, data = pop_songs)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.9888  -4.3596  -0.2072   3.4694  18.0198
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 821.49760  113.47481   7.239 1.39e-09 ***
## year         -0.39029    0.05698  -6.849 6.13e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.346 on 56 degrees of freedom
## Multiple R-squared:  0.4559, Adjusted R-squared:  0.4461
## F-statistic: 46.92 on 1 and 56 DF,  p-value: 6.134e-09
```

```
pop_facet <- songs %>% dplyr::select(track_popularity, year, playlist_genre) %>%
  group_by(year, playlist_genre) %>%
  summarise(track_popularity = mean(track_popularity))
```

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

```
pop_facet %>%
  ggplot(aes(x = year, y = track_popularity)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(playlist_genre ~., ncol = 2) +
  labs(caption = "Figure 19 - Track Popularity by Year by Class Scatter Plot with Linear Model")
```

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

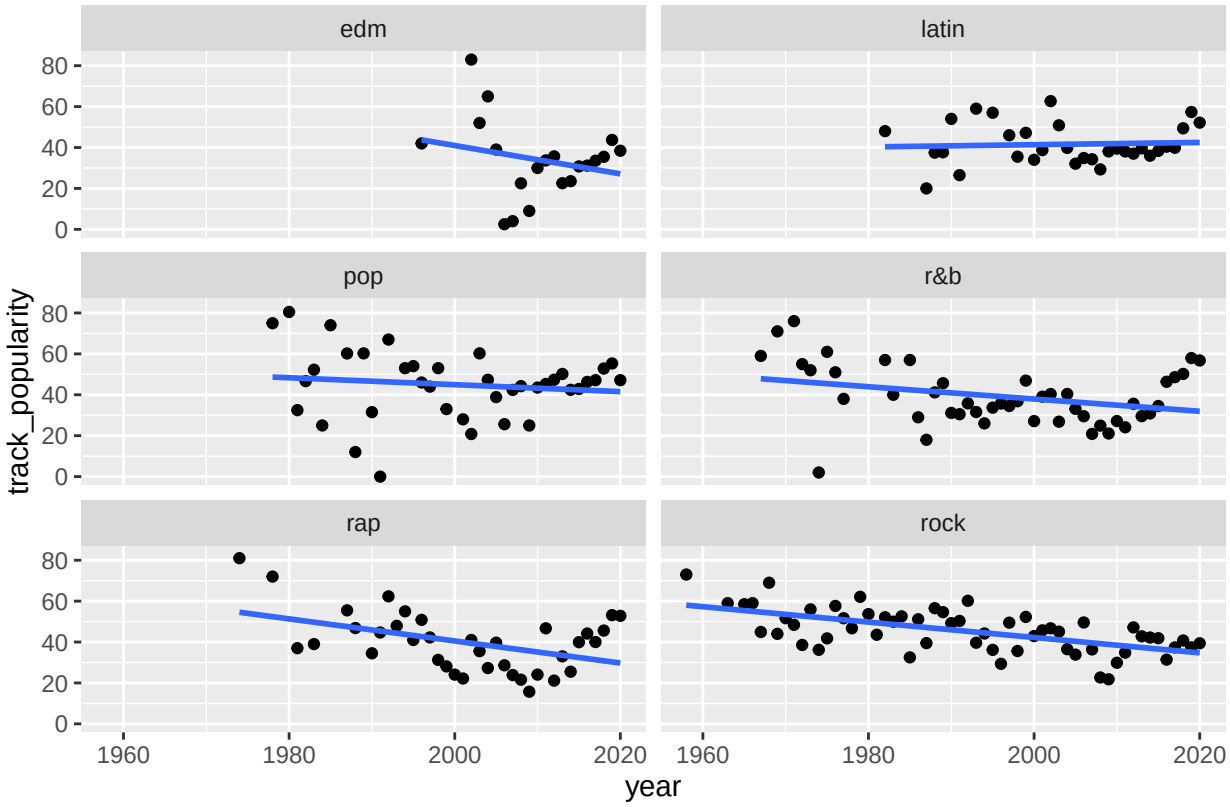

Figure 19 − Track Popularity by Year by Class Scatter Plot with Linear Model

```
pop_facet_pop <- pop_facet %>% filter(playlist_genre == "pop")
summary(lm(data = pop_facet_pop, track_popularity ~ year))
```

```
##
## Call:
## lm(formula = track_popularity ~ year, data = pop_facet_pop)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -46.462  -8.669   2.215   7.970  32.173
##
## Coefficients:
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 384.0287   429.3690   0.894    0.377
## year         -0.1695     0.2147  -0.790    0.435
##
## Residual standard error: 16.6 on 37 degrees of freedom
## Multiple R-squared:  0.01658,    Adjusted R-squared:  -0.01
## F-statistic: 0.6237 on 1 and 37 DF,  p-value: 0.4347
```

```
pop_facet_rock <- pop_facet %>% filter(playlist_genre == "rock")
summary(lm(data = pop_facet_rock, track_popularity ~ year))
```

```
##
## Call:
## lm(formula = track_popularity ~ year, data = pop_facet_rock)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -17.058  -5.656   2.595   5.246  14.964
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 794.89607  128.38986   6.191 7.41e-08 ***
## year         -0.37633    0.06447  -5.837 2.79e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.311 on 56 degrees of freedom
## Multiple R-squared:  0.3783, Adjusted R-squared:  0.3672
## F-statistic: 34.07 on 1 and 56 DF,  p-value: 2.792e-07
```

```
pop_facet_rb <- pop_facet %>% filter(playlist_genre == "r&b")
summary(lm(data = pop_facet_rb, track_popularity ~ year))
```

```
##
## Call:
## lm(formula = track_popularity ~ year, data = pop_facet_rb)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -43.744  -9.134  -2.883   8.670  29.355
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 638.3318   268.2236   2.380   0.0216 *
## year         -0.3002     0.1344  -2.234   0.0305 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.94 on 45 degrees of freedom
## Multiple R-squared:  0.09983,    Adjusted R-squared:  0.07982
## F-statistic:  4.99 on 1 and 45 DF,  p-value: 0.0305
```

```r
pop_facet_rap <- pop_facet %>% filter(playlist_genre == "rap")
summary(lm(data = pop_facet_rap, track_popularity ~ year))
```

```
##
## Call:
## lm(formula = track_popularity ~ year, data = pop_facet_rap)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -19.9499 -11.1085  -0.5187   8.6528  26.4366
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1119.7453   359.2937   3.117  0.00364 **
## year          -0.5396     0.1795  -3.006  0.00488 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.1 on 35 degrees of freedom
## Multiple R-squared:  0.2052, Adjusted R-squared:  0.1824
## F-statistic: 9.034 on 1 and 35 DF,  p-value: 0.004877
```

```r
pop_facet_lat <- pop_facet %>% filter(playlist_genre == "latin")
summary(lm(data = pop_facet_lat, track_popularity ~ year))
```

```
##
## Call:
## lm(formula = track_popularity ~ year, data = pop_facet_lat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -20.658  -5.865  -2.592   7.177  21.177
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -69.50696  340.89752  -0.204    0.840
## year          0.05544    0.17013   0.326    0.747
##
## Residual standard error: 9.999 on 30 degrees of freedom
## Multiple R-squared:  0.003528,   Adjusted R-squared:  -0.02969
## F-statistic: 0.1062 on 1 and 30 DF,  p-value: 0.7468
```

```r
pop_facet_edm <- pop_facet %>% filter(playlist_genre == "edm")
summary(lm(data = pop_facet_edm, track_popularity ~ year))
```

```
##
## Call:
## lm(formula = track_popularity ~ year, data = pop_facet_edm)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

```
## -34.319  -8.162   0.794   8.041  43.413
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1424.9930  1358.5582   1.049    0.308
## year          -0.6920     0.6758  -1.024    0.319
##
## Residual standard error: 18.92 on 18 degrees of freedom
## Multiple R-squared:  0.05504,    Adjusted R-squared:  0.002547
## F-statistic: 1.049 on 1 and 18 DF,  p-value: 0.3194
```

```
# converting variables to factors
songs$playlist_genre <- as_factor(songs$playlist_genre)
songs$mode <- as_factor(songs$mode)
songs$year <- as_factor(songs$year)
songs$month <- as_factor(songs$month)
songs$day <- as_factor(songs$day)
songs$key <- as_factor(songs$key)
```

Appendix B - Split/Preprocessing/Tune and Fit/Results

Split and Preprocessing

```
# Train Test split
set.seed(1805361)
songs_split <- initial_split(songs, strata = playlist_genre)
songs_split
```

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

```
songs_train <- training(songs_split)
songs_test <- testing(songs_split)
head(songs_train, 6)
```

```
## # A tibble: 6 x 17
##   energy key   loudness mode  acoustic~1 instr~2 liven~3 valence durat~4 track~5
##    <dbl> <fct>    <dbl> <fct>      <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1  0.552 0        -5.18 1        0.00121 9.73e-3  0.0929   0.552  291693       0
## 2  0.981 5        -1.49 0        0.0311  1.17e-3  0.312    0.256  183750      16
## 3  0.917 11       -5.07 0        0.00871 2.31e-1  0.681    0.43   355398      39
## 4  0.894 5        -4.76 1        0.0512  3.15e-4  0.234    0.415  204375      62
## 5  0.784 1        -4.35 0        0.00812 5.54e-3  0.116    0.304  183545      62
## 6  0.648 5        -5.96 1        0.00987 2.59e-2  0.254    0.21   215625      24
## # ... with 7 more variables: danceability <dbl>, speechiness <dbl>,
## #   tempo <dbl>, year <fct>, month <fct>, day <fct>, playlist_genre <fct>, and
## #   abbreviated variable names 1: acousticness, 2: instrumentalness,
## #   3: liveness, 4: duration_ms, 5: track_popularity
```

```
head(songs_test, 6)
```

```
## # A tibble: 6 x 17
##   energy key   loudness mode  acoustic~1 instr~2 liven~3 valence durat~4 track~5
##    <dbl> <fct>    <dbl> <fct>      <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1  0.863 5        -4.40 0       0.228     0       0.156   0.672   190400      47
## 2  0.861 9        -5.38 0       0.00646   0.0165  0.0647  0.494   279375      18
## 3  0.822 5        -5.14 1       0.000109  0.851   0.172   0.144   298736       0
## 4  0.574 9        -9.46 0       0.0378    0.916   0.0686  0.0399  208400       0
## 5  0.911 6        -7.10 0       0.0245    0.809   0.0944  0.136   467081      24
## 6  0.89  8        -5.64 1       0.0171    0.252   0.146   0.0897  185916      77
## # ... with 7 more variables: danceability <dbl>, speechiness <dbl>,
## #   tempo <dbl>, year <fct>, month <fct>, day <fct>, playlist_genre <fct>, and
## #   abbreviated variable names 1: acousticness, 2: instrumentalness,
## #   3: liveness, 4: duration_ms, 5: track_popularity
```

```
#Preprocessing
colnames(songs)
```

```
##  [1] "energy"           "key"              "loudness"         "mode"
##  [5] "acousticness"     "instrumentalness" "liveness"         "valence"
##  [9] "duration_ms"      "track_popularity" "danceability"     "speechiness"
## [13] "tempo"            "year"             "month"            "day"
## [17] "playlist_genre"
```

```
songs_recipe <- recipe(playlist_genre ~ ., data = songs_train) %>%
  step_normalize(all_numeric()) %>%
  step_zv(all_predictors()) %>%
  step_corr(all_numeric())

songs_recipe
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor         16
##
## Operations:
##
## Centering and scaling for all_numeric()
## Zero variance filter on all_predictors()
## Correlation filter on all_numeric()
```

```
songs_prepped <- prep(songs_recipe)
songs_juiced <- juice(songs_prepped)
songs_baked <- bake(songs_prepped, songs_test)

head(songs_juiced)
```

```
## # A tibble: 6 x 17
##   energy key   loudness mode  acoustic~1 instr~2 liven~3 valence durat~4 track~5
##    <dbl> <fct>    <dbl> <fct>      <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 -0.774 0        0.520 1         -0.807  -0.329  -0.619   0.204   1.16   -1.73
## 2  1.54  5        1.75  0         -0.672  -0.368   0.760  -1.06   -0.687  -1.09
## 3  1.20  11       0.557 0         -0.773   0.669   3.08   -0.318   2.25   -0.166
## 4  1.07  5        0.662 1         -0.582  -0.372   0.269  -0.383  -0.335   0.758
## 5  0.479 1        0.796 0         -0.775  -0.348  -0.473  -0.858  -0.691   0.758
## 6 -0.256 5        0.259 1         -0.768  -0.256   0.395  -1.26   -0.142  -0.769
## # ... with 7 more variables: danceability <dbl>, speechiness <dbl>,
## #   tempo <dbl>, year <fct>, month <fct>, day <fct>, playlist_genre <fct>, and
## #   abbreviated variable names 1: acousticness, 2: instrumentalness,
## #   3: liveness, 4: duration_ms, 5: track_popularity
```

```
head(songs_baked)
```

```
## # A tibble: 6 x 17
##   energy key   loudness mode  acoustic~1 instr~2 liven~3 valence durat~4 track~5
##    <dbl> <fct>    <dbl> <fct>      <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1  0.906 5        0.781 0          0.213  -0.373  -0.221  0.719   -0.574   0.155
## 2  0.895 9        0.454 0         -0.783  -0.299  -0.796 -0.0442   0.947  -1.01
## 3  0.684 5        0.534 1         -0.812   3.46   -0.121 -1.54     1.28   -1.73
## 4 -0.655 9       -0.911 0         -0.642   3.76   -0.771 -1.99    -0.266  -1.73
## 5  1.16  6       -0.123 0         -0.702   3.28   -0.609 -1.58     4.16   -0.769
## 6  1.05  8        0.364 1         -0.735   0.763  -0.284 -1.78    -0.650   1.36
## # ... with 7 more variables: danceability <dbl>, speechiness <dbl>,
## #   tempo <dbl>, year <fct>, month <fct>, day <fct>, playlist_genre <fct>, and
## #   abbreviated variable names 1: acousticness, 2: instrumentalness,
## #   3: liveness, 4: duration_ms, 5: track_popularity
```

```
summary(songs_juiced)
```

```
##      energy             key          loudness       mode      acousticness
##  Min.   :-3.6920   1      : 530   Min.   :-6.4619   0:1925   Min.   :-0.8120
##  1st Qu.:-0.6608   7      : 483   1st Qu.:-0.4852   1:2575   1st Qu.:-0.7356
##  Median : 0.1170   0      : 468   Median : 0.1881            Median :-0.4269
##  Mean   : 0.0000   9      : 448   Mean   : 0.0000            Mean   : 0.0000
##  3rd Qu.: 0.8029   11     : 391   3rd Qu.: 0.6997            3rd Qu.: 0.3705
##  Max.   : 1.6455   2      : 389   Max.   : 2.1977            Max.   : 3.6481
##                    (Other):1791
##  instrumentalness     liveness          valence           duration_ms
##  Min.   :-0.3731   Min.   :-1.2031   Min.   :-2.161035   Min.   :-3.7597
##  1st Qu.:-0.3731   1st Qu.:-0.6174   1st Qu.:-0.776955   1st Qu.:-0.6280
##  Median :-0.3731   Median :-0.4103   Median :-0.001355   Median :-0.1477
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.000000   Mean   : 0.0000
##  3rd Qu.:-0.3556   3rd Qu.: 0.3448   3rd Qu.: 0.782814   3rd Qu.: 0.4673
##  Max.   : 4.1096   Max.   : 5.0139   Max.   : 2.012631   Max.   : 4.9896
##
##  track_popularity   danceability      speechiness          tempo
##  Min.   :-1.7335   Min.   :-4.4418   Min.   :-1.0739   Min.   :-4.443555
##  1st Qu.:-0.7290   1st Qu.:-0.6641   1st Qu.:-0.6616   1st Qu.:-0.775004
##  Median : 0.1149   Median : 0.1201   Median :-0.4297   Median :-0.003297
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.000000
```

```
##  3rd Qu.: 0.7980   3rd Qu.: 0.7270   3rd Qu.: 0.2428   3rd Qu.: 0.474574
##  Max.   : 2.2044   Max.   : 2.2340   Max.   : 8.2240   Max.   : 4.344071
##
##       year          month      day      playlist_genre
##  2019   :1309   1      : 769   1: 176   edm  :750
##  2018   : 515   11     : 468   2: 566   latin:750
##  2017   : 324   10     : 427   3: 729   pop  :750
##  2016   : 269   9      : 381   4: 348   r&b  :750
##  2015   : 240   8      : 360   5: 393   rap  :750
##  2014   : 211   6      : 357   6:2118   rock :750
##  (Other):1632   (Other):1738   7: 170
```

```
# 10 fold CV set
pacman::p_load(parsnip)
pacman::p_load(dials)

set.seed(1805361)
songs_cv <- vfold_cv(songs_juiced, v = 10, strata = playlist_genre)
```

Tune and Fit Models

```
# Finding Feature Importance to reduce the number of features
pacman::p_load(ranger)
pacman::p_load(vip)
pacman::p_load(parsnip)
pacman::p_load(tune)

rf_spec <- rand_forest(mode = "classification", trees = 100) %>%
  set_engine("ranger", importance = "permutation")
set.seed(1805361)
songs_rf <- rf_spec %>% fit(playlist_genre ~ ., data = songs_juiced)
songs_rf %>% vip(num_features = 16) + theme_minimal()  +
  labs(caption = "Figure 20 - Variables by Importance")
```

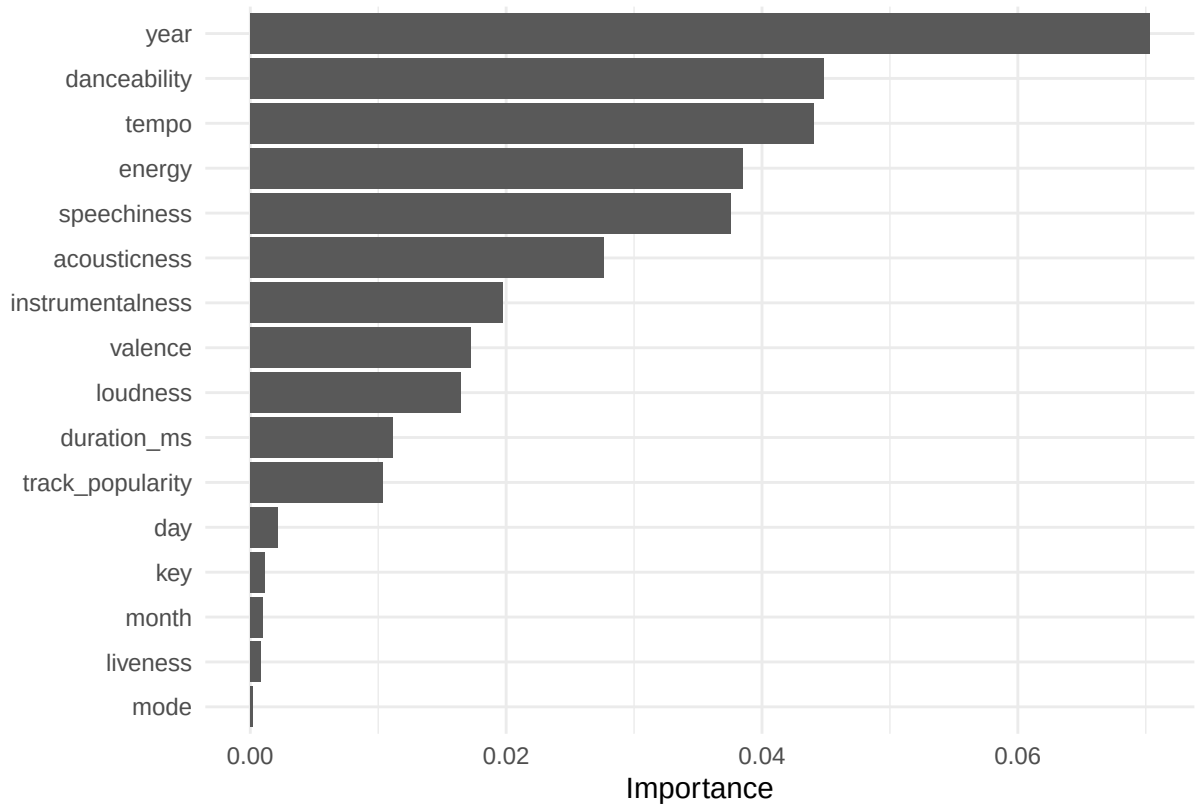

Figure 20 − Variables by Importance

```
#Base
set.seed(1805361)
rf_resamples <- fit_resamples(object = rf_spec,
  preprocessor = recipe(playlist_genre ~., data = songs_juiced),
  resamples = songs_cv)
collect_metrics(rf_resamples)
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy multiclass 0.556    10 0.00642 Preprocessor1_Model1
## 2 roc_auc  hand_till  0.845    10 0.00323 Preprocessor1_Model1
```

```
#1
songs_juiced <- dplyr::select(songs_juiced, -mode)
set.seed(1805361)
rf_resamples <- fit_resamples(object = rf_spec,
  preprocessor = recipe(playlist_genre ~., data = songs_juiced),
  resamples = songs_cv)
collect_metrics(rf_resamples)
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy multiclass 0.553    10 0.00601 Preprocessor1_Model1
```

```
## 2 roc_auc  hand_till  0.846    10 0.00276 Preprocessor1_Model1
```

```
#2
songs_juiced <- dplyr::select(songs_juiced, -liveness)
set.seed(1805361)
rf_resamples <- fit_resamples(object = rf_spec,
  preprocessor = recipe(playlist_genre ~., data = songs_juiced),
  resamples = songs_cv)
collect_metrics(rf_resamples)
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy multiclass 0.557    10 0.00604 Preprocessor1_Model1
## 2 roc_auc  hand_till  0.844    10 0.00323 Preprocessor1_Model1
```

```
#3
songs_juiced <- dplyr::select(songs_juiced, -month)
set.seed(1805361)
rf_resamples <- fit_resamples(object = rf_spec,
  preprocessor = recipe(playlist_genre ~., data = songs_juiced),
  resamples = songs_cv)
collect_metrics(rf_resamples)
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy multiclass 0.555    10 0.00531 Preprocessor1_Model1
## 2 roc_auc  hand_till  0.845    10 0.00299 Preprocessor1_Model1
```

```
#4
songs_juiced <- dplyr::select(songs_juiced, -key)
set.seed(1805361)
rf_resamples <- fit_resamples(object = rf_spec,
  preprocessor = recipe(playlist_genre ~., data = songs_juiced),
  resamples = songs_cv)
collect_metrics(rf_resamples)
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy multiclass 0.555    10 0.00720 Preprocessor1_Model1
## 2 roc_auc  hand_till  0.847    10 0.00344 Preprocessor1_Model1
```

```
#5
songs_juiced <- dplyr::select(songs_juiced, -day)
set.seed(1805361)
rf_resamples <- fit_resamples(object = rf_spec,
  preprocessor = recipe(playlist_genre ~., data = songs_juiced),
  resamples = songs_cv)
collect_metrics(rf_resamples)
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy multiclass 0.555    10 0.00452 Preprocessor1_Model1
## 2 roc_auc  hand_till  0.846    10 0.00254 Preprocessor1_Model1
```

```
songs_baked <- dplyr::select(songs_baked, -mode, -liveness, -month, -key, -day)
```

```
# RF with Tuned features
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
pacman::p_load(yardstick)

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

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

```
rand_spec_grid <- grid_regular(
  finalize(mtry(), songs_juiced %>% dplyr::select(-playlist_genre)),
           min_n(),
           levels = 5)
rand_spec_grid
```

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

```
##  2     3     2
##  3     6     2
##  4     8     2
##  5    11     2
##  6     1    11
##  7     3    11
##  8     6    11
##  9     8    11
## 10    11    11
## # ... with 15 more rows
```

```
set.seed(1805361)

pacman::p_load(tune)
rf_tuned <- tune_grid(object = rf_spec_tune,
                      preprocessor = recipe(playlist_genre ~., data = songs_juiced),
                      resamples = songs_cv,
                      grid = rand_spec_grid)

best_auc <- select_best(rf_tuned, "roc_auc")
final_rf <- finalize_model(rf_spec_tune, best_auc)
songs_rf <- final_rf %>% fit(playlist_genre ~., data = songs_juiced)
songs_rf
```

```
## parsnip model object
##
## Ranger result
##
## Call:
##  ranger::ranger(x = maybe_data_frame(x), y = y, mtry = min_cols(~3L,      x), num.trees = ~100, min.
##
## Type:                             Probability estimation
## Number of trees:                  100
## Sample size:                      4500
## Number of independent variables:  11
## Mtry:                             3
## Target node size:                 21
## Variable importance mode:         permutation
## Splitrule:                        gini
## OOB prediction error (Brier s.):  0.4441888
```

```
set.seed(1805361)
rf_df <- predict(songs_rf, new_data = songs_baked, type = "prob")
set.seed(1805361)
songs_rf_pred <- predict(songs_rf, new_data = songs_baked, type = "class") %>%
  bind_cols(songs_baked %>% dplyr::select(playlist_genre)) %>%
  bind_cols(rf_df)

categorical_metrics <- metric_set(sens, spec)
rf_cat_met <- songs_rf_pred %>%
  categorical_metrics(truth = playlist_genre, estimate = .pred_class)
rf_cat_met_class <- songs_rf_pred %>% group_by(playlist_genre) %>%
  categorical_metrics(truth = playlist_genre, estimate = .pred_class)
```

```
## Warning: While computing multiclass `sens()`, some levels had no true events (i.e. `true_positive + f
## Sensitivity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted events actually occured for each problematic event level
## 'latin': 19
## 'pop': 33
## 'r&b': 10
## 'rap': 16
## 'rock': 4

## Warning: While computing multiclass `sens()`, some levels had no true events (i.e. `true_positive + f
## Sensitivity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted events actually occured for each problematic event level
## 'edm': 20
## 'pop': 37
## 'r&b': 42
## 'rap': 31
## 'rock': 7

## Warning: While computing multiclass `sens()`, some levels had no true events (i.e. `true_positive + f
## Sensitivity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted events actually occured for each problematic event level
## 'edm': 37
## 'latin': 40
## 'r&b': 41
## 'rap': 17
## 'rock': 28

## Warning: While computing multiclass `sens()`, some levels had no true events (i.e. `true_positive + f
## Sensitivity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted events actually occured for each problematic event level
## 'edm': 9
## 'latin': 28
## 'pop': 39
## 'rap': 47
## 'rock': 17

## Warning: While computing multiclass `sens()`, some levels had no true events (i.e. `true_positive + f
## Sensitivity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted events actually occured for each problematic event level
## 'edm': 23
## 'latin': 24
## 'pop': 10
## 'r&b': 24
## 'rock': 9

## Warning: While computing multiclass `sens()`, some levels had no true events (i.e. `true_positive + f
## Sensitivity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted events actually occured for each problematic event level
## 'edm': 8
## 'latin': 6
## 'pop': 23
## 'r&b': 15
## 'rap': 3
```

```
## Warning: While computing multiclass `spec()`, some levels had no true negatives (i.e. `true_negative
## Specificity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted negatives actually occured for each problematic event lev
## 'edm': 82

## Warning: While computing multiclass `spec()`, some levels had no true negatives (i.e. `true_negative
## Specificity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted negatives actually occured for each problematic event lev
## 'latin': 137

## Warning: While computing multiclass `spec()`, some levels had no true negatives (i.e. `true_negative
## Specificity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted negatives actually occured for each problematic event lev
## 'pop': 163

## Warning: While computing multiclass `spec()`, some levels had no true negatives (i.e. `true_negative
## Specificity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted negatives actually occured for each problematic event lev
## 'r&b': 140

## Warning: While computing multiclass `spec()`, some levels had no true negatives (i.e. `true_negative
## Specificity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted negatives actually occured for each problematic event lev
## 'rap': 90

## Warning: While computing multiclass `spec()`, some levels had no true negatives (i.e. `true_negative
## Specificity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted negatives actually occured for each problematic event lev
## 'rock': 55
```

```
rf_cat_met
```

```
## # A tibble: 2 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 sens    macro          0.555
## 2 spec    macro          0.911
```

```
rf_cat_met_class
```

```
## # A tibble: 12 x 4
##    playlist_genre .metric .estimator .estimate
##    <fct>          <chr>   <chr>          <dbl>
##  1 edm            sens    macro          0.672
##  2 latin          sens    macro          0.452
##  3 pop            sens    macro          0.348
##  4 r&b            sens    macro          0.44
##  5 rap            sens    macro          0.64
##  6 rock           sens    macro          0.78
##  7 edm            spec    macro          0.934
##  8 latin          spec    macro          0.890
```

```
##  9 pop            spec    macro          0.870
## 10 r&b            spec    macro          0.888
## 11 rap            spec    macro          0.928
## 12 rock           spec    macro          0.956
```

```
rf_real_values <- matrix(songs_rf_pred$playlist_genre)
rf_preds <- as.matrix(rf_df)
colnames(rf_preds) <- c("edm", "latin", "pop", "r&b", "rap","rock")
rf_auc <- multiclass.roc(rf_real_values, rf_preds)
rf_auc
```

```
##
## Call:
## multiclass.roc.default(response = rf_real_values, predictor = rf_preds)
##
## Data: multivariate predictor rf_preds with 6 levels of rf_real_values: edm, latin, pop, r&b, rap, ro
## Multi-class area under the curve: 0.8502
```

```
# LDA

library(discrim)
```

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

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

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

set.seed(1805361)
songs_lda <- songs_lda %>% fit(playlist_genre ~., data = songs_juiced)
songs_lda
```

```
## parsnip model object
##
## Call:
## lda(playlist_genre ~ ., data = data)
##
## Prior probabilities of groups:
##       edm     latin       pop       r&b       rap      rock
## 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667
##
## Group means:
##            energy   loudness acousticness instrumentalness     valence
## edm    0.59811778  0.4632549 -0.445336669      0.623648185 -0.47630951
## latin  0.07741172  0.1738560  0.144059456     -0.172419503  0.39294387
## pop    0.03114379  0.1473158 -0.009026459     -0.060074650 -0.01392100
## r&b   -0.64097114 -0.3969331  0.442102398     -0.239164102  0.04909536
## rap   -0.25298271 -0.0847866  0.023637775     -0.008412712 -0.10170054
```

```
## rock   0.18728057 -0.3027070 -0.155436501     -0.143577217  0.14989183
##       duration_ms track_popularity danceability speechiness       tempo
## edm   -0.08069681      -0.29177084   0.01220001  -0.1552223  0.17346491
## latin -0.18733921       0.15757858   0.39701170  -0.0496018 -0.09532570
## pop   -0.08457149       0.20810159  -0.11111539  -0.3054376 -0.00764063
## r&b    0.18243892      -0.03181575   0.11857693   0.1485790 -0.26570773
## rap   -0.21436046       0.01484973   0.44697248   0.8348214  0.03466750
## rock   0.38452906      -0.05694331  -0.86364573  -0.4731386  0.16054165
##          year1963    year1965 year1966    year1967    year1968    year1969
## edm   0.000000000 0.000000000    0.000 0.000000000 0.000000000 0.000000000
## latin 0.000000000 0.000000000    0.000 0.000000000 0.000000000 0.000000000
## pop   0.000000000 0.000000000    0.000 0.000000000 0.000000000 0.000000000
## r&b   0.000000000 0.000000000    0.000 0.001333333 0.000000000 0.001333333
## rap   0.000000000 0.000000000    0.000 0.000000000 0.000000000 0.000000000
## rock  0.001333333 0.002666667    0.004 0.013333333 0.001333333 0.008000000
##         year1970    year1971    year1972    year1973    year1974    year1975
## edm   0.00000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## latin 0.00000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## pop   0.00000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## r&b   0.00000000 0.001333333 0.001333333 0.001333333 0.000000000 0.001333333
## rap   0.00000000 0.000000000 0.000000000 0.000000000 0.001333333 0.000000000
## rock  0.01866667 0.012000000 0.004000000 0.022666667 0.008000000 0.022666667
##          year1976    year1977    year1978 year1979    year1980 year1981
## edm   0.000000000 0.000000000 0.000000000    0.000 0.000000000    0.000
## latin 0.000000000 0.000000000 0.000000000    0.000 0.000000000    0.000
## pop   0.000000000 0.000000000 0.001333333    0.000 0.001333333    0.004
## r&b   0.001333333 0.002666667 0.000000000    0.000 0.000000000    0.000
## rap   0.000000000 0.000000000 0.000000000    0.000 0.000000000    0.000
## rock  0.021333333 0.016000000 0.017333333    0.008 0.016000000    0.008
##          year1982   year1983    year1984    year1985    year1986    year1987
## edm   0.000000000 0.00000000 0.000000000 0.000000000 0.000000000 0.000000000
## latin 0.001333333 0.00000000 0.000000000 0.000000000 0.000000000 0.001333333
## pop   0.001333333 0.00400000 0.001333333 0.002666667 0.000000000 0.005333333
## r&b   0.001333333 0.00000000 0.000000000 0.001333333 0.001333333 0.001333333
## rap   0.000000000 0.00000000 0.000000000 0.000000000 0.000000000 0.001333333
## rock  0.010666667 0.01866667 0.016000000 0.017333333 0.013333333 0.016000000
##          year1988    year1989    year1990    year1991    year1992    year1993
## edm   0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## latin 0.001333333 0.004000000 0.002666667 0.002666667 0.000000000 0.001333333
## pop   0.001333333 0.002666667 0.005333333 0.001333333 0.001333333 0.000000000
## r&b   0.013333333 0.005333333 0.006666667 0.017333333 0.020000000 0.012000000
## rap   0.008000000 0.000000000 0.001333333 0.001333333 0.004000000 0.014666667
## rock  0.021333333 0.016000000 0.012000000 0.017333333 0.006666667 0.010666667
##          year1994    year1995    year1996    year1997    year1998    year1999
## edm   0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## latin 0.000000000 0.001333333 0.000000000 0.002666667 0.002666667 0.005333333
## pop   0.001333333 0.001333333 0.005333333 0.001333333 0.005333333 0.004000000
## r&b   0.005333333 0.005333333 0.016000000 0.024000000 0.014666667 0.016000000
## rap   0.009333333 0.012000000 0.010666667 0.004000000 0.013333333 0.013333333
## rock  0.012000000 0.013333333 0.008000000 0.017333333 0.021333333 0.014666667
##          year2000    year2001    year2002   year2003    year2004    year2005
## edm   0.000000000 0.000000000 0.001333333 0.00000000 0.001333333 0.000000000
## latin 0.004000000 0.009333333 0.002666667 0.00800000 0.012000000 0.012000000
## pop   0.000000000 0.005333333 0.006666667 0.00400000 0.004000000 0.009333333
```

```
## r&b   0.018666667 0.012000000 0.013333333 0.02400000 0.013333333 0.021333333
## rap   0.009333333 0.012000000 0.004000000 0.01066667 0.016000000 0.025333333
## rock  0.008000000 0.006666667 0.009333333 0.01466667 0.018666667 0.044000000
##          year2006    year2007    year2008    year2009   year2010    year2011
## edm   0.002666667 0.001333333 0.005333333 0.002666667 0.00400000 0.008000000
## latin 0.024000000 0.010666667 0.022666667 0.016000000 0.03333333 0.009333333
## pop   0.006666667 0.008000000 0.010666667 0.012000000 0.02533333 0.014666667
## r&b   0.013333333 0.017333333 0.020000000 0.014666667 0.01333333 0.038666667
## rap   0.022666667 0.016000000 0.009333333 0.009333333 0.01200000 0.006666667
## rock  0.014666667 0.028000000 0.036000000 0.012000000 0.02266667 0.018666667
##         year2012   year2013   year2014   year2015   year2016   year2017
## edm   0.03066667 0.03866667 0.11466667 0.08000000 0.07066667 0.08266667
## latin 0.02800000 0.02933333 0.02533333 0.04933333 0.06266667 0.06800000
## pop   0.03600000 0.04133333 0.06000000 0.08266667 0.09333333 0.10133333
## r&b   0.02933333 0.02666667 0.03733333 0.04533333 0.05733333 0.08133333
## rap   0.02266667 0.02000000 0.02933333 0.04666667 0.05866667 0.07333333
## rock  0.01333333 0.00800000 0.01466667 0.01600000 0.01600000 0.02533333
##         year2018  year2019    year2020
## edm   0.12533333 0.3973333 0.033333333
## latin 0.12000000 0.4080000 0.018666667
## pop   0.15866667 0.2440000 0.024000000
## r&b   0.10133333 0.2133333 0.014666667
## rap   0.13866667 0.3306667 0.032000000
## rock  0.04266667 0.1520000 0.009333333
##
## Coefficients of linear discriminants:
##                           LD1          LD2         LD3         LD4         LD5
## energy            0.18032696  0.731025436 -0.01521763  0.51420496  0.31497620
## loudness         -0.20550264 -0.006972724 -0.02952177 -0.23455535 -0.02201082
## acousticness     -0.08309701 -0.120975098 -0.29556957  0.04410975  0.40528282
## instrumentalness -0.15888220  0.342154394  0.27249314 -0.16813671  0.07421979
## valence           0.22517787 -0.271528843 -0.51240002  0.30838514  0.03532209
## duration_ms       0.07294695 -0.038809472 -0.09280957 -0.16485197  0.25549443
## track_popularity  0.03239703 -0.107078205 -0.27442342  0.16553574 -0.47510959
## danceability     -0.69108579  0.024689617  0.10625474  0.28674340  0.36549910
## speechiness      -0.37121183 -0.401774324  0.70063196  0.17800407 -0.18292415
## tempo            -0.03860382  0.136390734  0.11483766  0.12377944 -0.12181360
## year1963         -0.44664824  1.409699593  0.19405655  2.45086661 -1.89439340
## year1965         -0.33759723 -1.240110601 -0.30586075 -0.10185664 -0.36459207
## year1966         -1.05505612 -0.792887418 -0.38178393  0.95267118 -0.21398037
## year1967         -1.01878463 -0.637655420 -0.87097690  1.00422678 -0.25814519
## year1968         -1.03657137 -1.174544567 -0.60604817  1.10794028  0.34038399
## year1969         -1.02859946 -1.152045518 -1.26052069  0.18738367  0.32597802
## year1970         -0.53178272 -0.420648587 -0.44427354  0.96285551 -0.47409761
## year1971         -0.90610235 -1.077405734 -1.01319573  0.83280930 -0.35855942
## year1972         -1.58251935 -1.013382824 -0.95574025 -0.20008846 -0.09636650
## year1973         -1.19421208 -0.690893178 -0.81315346  1.23733639 -0.05386645
## year1974         -1.92631153 -1.292231790 -0.59838484  1.55655633 -1.08716835
## year1975         -0.75625526 -0.744219267 -0.72629005  0.76555885 -0.65828712
## year1976         -0.89005205 -0.560620063 -0.89150754  0.84619143 -0.02674125
## year1977         -1.06060334 -1.570803582 -1.07682317  0.10438376 -0.09844969
## year1978         -0.60172616 -0.848478790 -1.26756924  0.87855776 -1.38619823
## year1979         -0.19485764 -0.923863735 -0.78908600  0.78259381 -0.43697488
## year1980         -0.67365505 -0.808795030 -0.91255102  0.80382166 -1.63561064
```

```
## year1981         -1.84367667 -0.821518585 -2.51632007  0.13012951 -4.23728790
## year1982         -1.74008685 -0.742335186 -1.46486120  0.60523474 -0.75434766
## year1983         -1.03497510 -0.999627013 -1.60463968  0.56890595 -2.73730802
## year1984         -1.00344098 -1.092013818 -1.18809501  0.82182148 -0.98392221
## year1985         -1.11752866 -0.768058576 -1.40079098  0.48352776 -2.10881831
## year1986         -1.17491938 -1.409326289 -0.90304179  0.51499803 -0.29016911
## year1987         -2.42153129 -1.189365429 -2.23569576  0.49206420 -2.53389492
## year1988         -2.86802793 -1.847230443 -1.17748740 -0.86631523 -0.69491379
## year1989         -2.54966080 -1.422637109 -2.23696551  0.12273501 -0.13590877
## year1990         -3.04793159 -1.779958237 -2.70365129 -0.87148883 -1.51301452
## year1991         -3.00610477 -2.070043547 -2.14194958 -1.72444925  0.66094953
## year1992         -3.52727928 -2.619304309 -2.22608589 -3.38522628  0.89523197
## year1993         -3.45908279 -1.896289941 -1.45880608 -0.42512998 -1.19671035
## year1994         -3.10589134 -1.653665130 -1.07689907  0.12345197 -1.73866134
## year1995         -2.95378754 -1.643307836 -1.26803039  0.49937276 -2.04049974
## year1996         -3.58104381 -1.930856476 -2.15846246 -1.76148480 -1.85886393
## year1997         -3.24849977 -2.470016709 -2.11834868 -1.81707993  0.82661048
## year1998         -2.86032573 -1.894618167 -1.89819848 -0.44585873 -1.86670979
## year1999         -3.34541650 -1.792739125 -2.06869911 -0.46648837 -0.92689906
## year2000         -3.52657560 -2.396257534 -2.32319846 -1.52471847  0.29750400
## year2001         -4.04073761 -2.156935658 -2.80876992  0.14223220 -1.37262224
## year2002         -3.43775766 -1.930415917 -2.86539558 -1.59606994 -1.19069314
## year2003         -3.19312141 -1.944057165 -2.45236442 -1.21787918 -0.23159697
## year2004         -3.38092392 -1.548564443 -2.59256887  0.57151775 -0.60632917
## year2005         -2.98678154 -1.871609418 -2.10292975  0.30000946 -1.38845048
## year2006         -4.12430733 -1.753409173 -2.77518152  1.34673575 -0.73764321
## year2007         -3.18745815 -1.794606874 -2.39839069  0.15909587 -1.23872054
## year2008         -3.28961459 -1.534284329 -2.98377906  0.37127217 -0.34107804
## year2009         -3.85524867 -1.672698649 -3.66358803  0.27081276 -1.28053501
## year2010         -3.75626821 -1.375463384 -3.70250104  0.79598889 -1.64139596
## year2011         -3.67908268 -1.955750136 -3.23457629 -2.06085251 -0.39725766
## year2012         -4.35346872 -1.042862920 -3.40308686 -0.59115833 -1.44154441
## year2013         -4.50496845 -0.889876027 -3.60965995 -0.56556917 -1.60777675
## year2014         -4.54147438 -0.406178463 -3.15151893 -1.29970644 -1.26182265
## year2015         -4.35822986 -0.712881109 -3.47646174 -0.61674573 -1.84639378
## year2016         -4.33737349 -0.809697795 -3.64079071 -0.38257066 -1.85381498
## year2017         -4.17049219 -0.876556645 -3.41044492 -0.65055970 -1.62026475
## year2018         -4.11046925 -0.891132133 -3.30760569 -0.18915370 -1.79944829
## year2019         -4.12784226 -0.682028408 -3.01340615  0.13458135 -0.33056435
## year2020         -4.19472377 -0.442136411 -2.68907790 -0.28268183 -1.49462656
##
## Proportion of trace:
##    LD1    LD2    LD3    LD4    LD5
## 0.5244 0.2816 0.1210 0.0500 0.0229
```

```
set.seed(1805361)
lda_df <- predict(songs_lda, new_data = songs_baked, type = "prob")
set.seed(1805361)
songs_lda_pred <- predict(songs_lda, new_data = songs_baked, type = "class") %>%
  bind_cols(songs_baked %>% dplyr::select(playlist_genre)) %>%
  bind_cols(lda_df)

lda_cat_met <- songs_lda_pred %>% categorical_metrics(truth = playlist_genre, estimate = .pred_class)
lda_cat_met_class <- songs_lda_pred %>% group_by(playlist_genre) %>%
```

```
  categorical_metrics(truth = playlist_genre, estimate = .pred_class)
```

```
## Warning: While computing multiclass `sens()`, some levels had no true events (i.e. `true_positive + i
## Sensitivity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted events actually occured for each problematic event level
## 'latin': 34
## 'pop': 48
## 'r&b': 14
## 'rap': 20
## 'rock': 2

## Warning: While computing multiclass `sens()`, some levels had no true events (i.e. `true_positive + i
## Sensitivity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted events actually occured for each problematic event level
## 'edm': 17
## 'pop': 60
## 'r&b': 25
## 'rap': 42
## 'rock': 5

## Warning: While computing multiclass `sens()`, some levels had no true events (i.e. `true_positive + i
## Sensitivity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted events actually occured for each problematic event level
## 'edm': 25
## 'latin': 52
## 'r&b': 36
## 'rap': 14
## 'rock': 22

## Warning: While computing multiclass `sens()`, some levels had no true events (i.e. `true_positive + i
## Sensitivity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted events actually occured for each problematic event level
## 'edm': 12
## 'latin': 45
## 'pop': 40
## 'rap': 48
## 'rock': 16

## Warning: While computing multiclass `sens()`, some levels had no true events (i.e. `true_positive + i
## Sensitivity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted events actually occured for each problematic event level
## 'edm': 28
## 'latin': 44
## 'pop': 17
## 'r&b': 20
## 'rock': 7

## Warning: While computing multiclass `sens()`, some levels had no true events (i.e. `true_positive + i
## Sensitivity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted events actually occured for each problematic event level
## 'edm': 32
## 'latin': 17
```

```
## 'pop': 26
## 'r&b': 30
## 'rap': 5

## Warning: While computing multiclass `spec()`, some levels had no true negatives (i.e. `true_negative
## Specificity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted negatives actually occured for each problematic event lev
## 'edm': 118

## Warning: While computing multiclass `spec()`, some levels had no true negatives (i.e. `true_negative
## Specificity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted negatives actually occured for each problematic event lev
## 'latin': 149

## Warning: While computing multiclass `spec()`, some levels had no true negatives (i.e. `true_negative
## Specificity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted negatives actually occured for each problematic event lev
## 'pop': 149

## Warning: While computing multiclass `spec()`, some levels had no true negatives (i.e. `true_negative
## Specificity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted negatives actually occured for each problematic event lev
## 'r&b': 161

## Warning: While computing multiclass `spec()`, some levels had no true negatives (i.e. `true_negative
## Specificity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted negatives actually occured for each problematic event lev
## 'rap': 116

## Warning: While computing multiclass `spec()`, some levels had no true negatives (i.e. `true_negative
## Specificity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted negatives actually occured for each problematic event lev
## 'rock': 110
```

```
lda_cat_met
```

```
## # A tibble: 2 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 sens    macro          0.465
## 2 spec    macro          0.893
```

```
lda_cat_met_class
```

```
## # A tibble: 12 x 4
##    playlist_genre .metric .estimator .estimate
##    <fct>          <chr>   <chr>          <dbl>
##  1 edm            sens    macro          0.528
##  2 latin          sens    macro          0.404
##  3 pop            sens    macro          0.404
##  4 r&b            sens    macro          0.356
```

```
##  5 rap            sens    macro          0.536
##  6 rock           sens    macro          0.56
##  7 edm            spec    macro          0.906
##  8 latin          spec    macro          0.881
##  9 pop            spec    macro          0.881
## 10 r&b            spec    macro          0.871
## 11 rap            spec    macro          0.907
## 12 rock           spec    macro          0.912
```

```
lda_real_values <- matrix(songs_lda_pred$playlist_genre)
lda_preds <- as.matrix(lda_df)
colnames(lda_preds) <- c("edm", "latin", "pop", "r&b", "rap","rock")
lda_auc <- multiclass.roc(lda_real_values, lda_preds)
lda_auc
```

```
##
## Call:
## multiclass.roc.default(response = lda_real_values, predictor = lda_preds)
##
## Data: multivariate predictor lda_preds with 6 levels of lda_real_values: edm, latin, pop, r&b, rap, r
## Multi-class area under the curve: 0.8087
```

```
# KKNN range 1-100, 20 levels
pacman::p_load(tune)
pacman::p_load(kknn)

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

neighbors_grid <- grid_regular(neighbors(range(1, 100)), levels = 20)
neighbors_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
```

```
set.seed(1805361)
knn_tune <- tune_grid(object = songs_nn,
                      preprocessor = recipe(playlist_genre ~., songs_juiced),
                      resamples = songs_cv,
                      grid = neighbors_grid)

best_neighbors <- select_best(knn_tune, "roc_auc")

final_knn <- finalize_model(songs_nn, best_neighbors)
final_knn
```

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

```
songs_knn <- final_knn %>% set_engine("kknn") %>%
  fit(playlist_genre ~., songs_juiced)

set.seed(1805361)
knn_df <- predict(songs_knn, new_data = songs_baked, type = "prob")
set.seed(1805361)
songs_knn_pred <- predict(songs_knn, new_data = songs_baked, type = "class") %>%
  bind_cols(dplyr::select(songs_baked, playlist_genre)) %>%
  bind_cols(knn_df)

knn_cat_met <- songs_knn_pred %>% categorical_metrics(truth = playlist_genre, estimate = .pred_class)
knn_cat_met_class <- songs_knn_pred %>% group_by(playlist_genre) %>%
  categorical_metrics(truth = playlist_genre, estimate = .pred_class)
```

```
## Warning: While computing multiclass `sens()`, some levels had no true events (i.e. `true_positive + i
## Sensitivity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted events actually occured for each problematic event level
## 'latin': 23
## 'pop': 31
## 'r&b': 10
## 'rap': 14
## 'rock': 7
```

```
## Warning: While computing multiclass `sens()`, some levels had no true events (i.e. `true_positive + i
## Sensitivity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted events actually occured for each problematic event level
## 'edm': 25
## 'pop': 48
## 'r&b': 22
## 'rap': 30
## 'rock': 12
```

```
## Warning: While computing multiclass `sens()`, some levels had no true events (i.e. `true_positive + i
## Sensitivity is undefined in this case, and those levels will be removed from the averaged result.
```

```
## Note that the following number of predicted events actually occured for each problematic event level
## 'edm': 45
## 'latin': 45
## 'r&b': 36
## 'rap': 13
## 'rock': 27

## Warning: While computing multiclass `sens()`, some levels had no true events (i.e. `true_positive + f
## Sensitivity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted events actually occured for each problematic event level
## 'edm': 14
## 'latin': 46
## 'pop': 36
## 'rap': 43
## 'rock': 21

## Warning: While computing multiclass `sens()`, some levels had no true events (i.e. `true_positive + f
## Sensitivity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted events actually occured for each problematic event level
## 'edm': 35
## 'latin': 37
## 'pop': 18
## 'r&b': 14
## 'rock': 8

## Warning: While computing multiclass `sens()`, some levels had no true events (i.e. `true_positive + f
## Sensitivity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted events actually occured for each problematic event level
## 'edm': 25
## 'latin': 13
## 'pop': 26
## 'r&b': 21
## 'rap': 7

## Warning: While computing multiclass `spec()`, some levels had no true negatives (i.e. `true_negative
## Specificity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted negatives actually occured for each problematic event lev
## 'edm': 85

## Warning: While computing multiclass `spec()`, some levels had no true negatives (i.e. `true_negative
## Specificity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted negatives actually occured for each problematic event lev
## 'latin': 137

## Warning: While computing multiclass `spec()`, some levels had no true negatives (i.e. `true_negative
## Specificity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted negatives actually occured for each problematic event lev
## 'pop': 166

## Warning: While computing multiclass `spec()`, some levels had no true negatives (i.e. `true_negative
## Specificity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted negatives actually occured for each problematic event lev
## 'r&b': 160
```

```
## Warning: While computing multiclass `spec()`, some levels had no true negatives (i.e. `true_negative
## Specificity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted negatives actually occured for each problematic event lev
## 'rap': 112

## Warning: While computing multiclass `spec()`, some levels had no true negatives (i.e. `true_negative
## Specificity is undefined in this case, and those levels will be removed from the averaged result.
## Note that the following number of predicted negatives actually occured for each problematic event lev
## 'rock': 92
```

```
knn_cat_met
```

```
## # A tibble: 2 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 sens    macro          0.499
## 2 spec    macro          0.900
```

```
knn_cat_met_class
```

```
## # A tibble: 12 x 4
##    playlist_genre .metric .estimator .estimate
##    <fct>          <chr>   <chr>          <dbl>
##  1 edm            sens    macro          0.66
##  2 latin          sens    macro          0.452
##  3 pop            sens    macro          0.336
##  4 r&b            sens    macro          0.36
##  5 rap            sens    macro          0.552
##  6 rock           sens    macro          0.632
##  7 edm            spec    macro          0.932
##  8 latin          spec    macro          0.890
##  9 pop            spec    macro          0.867
## 10 r&b            spec    macro          0.872
## 11 rap            spec    macro          0.910
## 12 rock           spec    macro          0.926
```

```
knn_real_values <- matrix(songs_knn_pred$playlist_genre)
knn_preds <- as.matrix(knn_df)
colnames(knn_preds) <- c("edm", "latin", "pop", "r&b", "rap","rock")
knn_auc <- multiclass.roc(knn_real_values, knn_preds)
knn_auc
```

```
##
## Call:
## multiclass.roc.default(response = knn_real_values, predictor = knn_preds)
##
## Data: multivariate predictor knn_preds with 6 levels of knn_real_values: edm, latin, pop, r&b, rap, r
## Multi-class area under the curve: 0.8148
```

Final Results

```r
# Results
results_table <- rf_cat_met %>% dplyr::select(.metric, .estimate) %>%
  bind_cols(lda_cat_met %>% dplyr::select(.estimate)) %>%
  bind_cols(knn_cat_met %>% dplyr::select(.estimate))
```

```
## New names:
## New names:
## * `.estimate` -> `.estimate...2`
## * `.estimate` -> `.estimate...3`
```

```r
colnames(results_table) <- c("Metric", "RF", "LDA", "KNN")
results_table <- results_table %>% add_row(Metric = "AUC", `RF`= 0.8502,
  LDA = 0.8087, KNN = 0.8148)
results_table <- results_table %>% knitr::kable(digits = 4,
  caption = "Table 1 - Model Sensitivity, Specificity and AUC")
results_table
```

Table 4: Table 1 - Model Sensitivity, Specificity and AUC

Metric	RF	LDA	KNN
sens	0.5553	0.4647	0.4987
spec	0.9111	0.8929	0.8997
AUC	0.8502	0.8087	0.8148

```r
results_table_class <- rf_cat_met_class %>% dplyr::select(playlist_genre, .metric, .estimate) %>%
  bind_cols(lda_cat_met_class %>% dplyr::select(.estimate)) %>%
  bind_cols(knn_cat_met_class %>% dplyr::select(.estimate))
```

```
## New names:
## New names:
## * `.estimate` -> `.estimate...3`
## * `.estimate` -> `.estimate...4`
```

```r
colnames(results_table_class) <- c("Genre", "Metric", "RF", "LDA", "KNN")
results_table_class <- results_table_class %>% knitr::kable(digits = 4, caption = "Table 2 - Sensitivity
results_table_class
```

Table 5: Table 2 - Sensitivity and Specificity by class

Genre	Metric	RF	LDA	KNN
edm	sens	0.6720	0.5280	0.6600
latin	sens	0.4520	0.4040	0.4520
pop	sens	0.3480	0.4040	0.3360
r&b	sens	0.4400	0.3560	0.3600
rap	sens	0.6400	0.5360	0.5520
rock	sens	0.7800	0.5600	0.6320
edm	spec	0.9344	0.9056	0.9320
latin	spec	0.8904	0.8808	0.8904

Genre	Metric	RF	LDA	KNN
pop	spec	0.8696	0.8808	0.8672
r&b	spec	0.8880	0.8712	0.8720
rap	spec	0.9280	0.9072	0.9104
rock	spec	0.9560	0.9120	0.9264

```
print("RF")
```

```
## [1] "RF"
```

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

```
##           Truth
## Prediction edm latin pop r&b rap rock
##      edm   168    20  37   9  23    8
##      latin  19   113  40  28  24    6
##      pop    33    37  87  39  10   23
##      r&b    10    42  41 110  24   15
##      rap    16    31  17  47 160    3
##      rock    4     7  28  17   9  195
```

```
print("LDA")
```

```
## [1] "LDA"
```

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

```
##           Truth
## Prediction edm latin pop r&b rap rock
##      edm   132    17  25  12  28   32
##      latin  34   101  52  45  44   17
##      pop    48    60 101  40  17   26
##      r&b    14    25  36  89  20   30
##      rap    20    42  14  48 134    5
##      rock    2     5  22  16   7  140
```

```
print("KNN")
```

```
## [1] "KNN"
```

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

```
##           Truth
## Prediction edm latin pop r&b rap rock
##      edm   165    25  45  14  35   25
##      latin  23   113  45  46  37   13
##      pop    31    48  84  36  18   26
##      r&b    10    22  36  90  14   21
##      rap    14    30  13  43 138    7
##      rock    7    12  27  21   8  158
```

References

1. The Ultimate Guide to Random Forest Regression. Keboola. Available at: https://www.keboola.com/blog/random-forest-regression (Accessed: December 1, 2022).
2. Brownlee, J. (2020), Linear discriminant analysis for machine learning, MachineLearningMastery.com. Available at: https://machinelearningmastery.com/linear-discriminant-analysis-for-machine-learning/ (Accessed: December 1, 2022).
3. K-Nearest Neighbor(KNN) algorithm for Machine Learning - Javatpoint (no date) www.javatpoint.com. Available at: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning (Accessed: December 1, 2022).