Optimal song properties for popularity

Group 3

December 4th, 2024

Introduction

Every year, Spotify users receive a summary of their year in music, nicknamed "wrapped" as it comes out before Christmas. As any artist, it can be said that appearing on Spotify users wrapped summaries is a very respectable achievement, and since many users post their wrapped data on social media, it is also good for artist growth. Furthermore, artists partner with Spotify to make videos for their top listeners (people with the most minutes streamed). For artists that did not make users wrapped's this year, they will likely be wondering what it was about their songs, if anything, that prevented them from being relevant.

Goals

The goal of this analysis will be to compare key metrics for Spotify's songs via their API (Application Programming Interface) to see if songs with certain attributes perform better than others. To do this, we will graph various different attributes against each other and measure it relative to popularity to see correlation. If meaningful correlation is found then we will create models for the data so that future artists can optimize their songs.

Libraries

```r
library(tidyverse)
library(tidymodels)
library(devtools)
library(ggrepel)
library(neuralnet)
library(ranger)
library(data.table)
```

```r
folder_path <- "Raw Data Files"
csv_files <- list.files(path = folder_path, pattern = "\\.csv$", full.names = TRUE)
data_list <- lapply(csv_files, fread)

big_dataset <- read_csv("tracks_features.csv")
```

Tidying Data

```r
# Joining the datasets
read_and_convert <- function(file) {
  df <- read_csv(file, col_types = cols(.default = "c"))
  df |> mutate(across(everything(), as.character))
}

all_data <- map(csv_files, read_and_convert)
combined_data <- bind_rows(all_data)

numeric_cols <- c("Duration (ms)", "Popularity", "Danceability", "Energy", "Key", "Loudness",
                  "Mode", "Speechiness", "Acousticness", "Instrumentalness", "Liveness",
                  "Valence", "Tempo", "Time Signature")

combined_data <- combined_data |>
  mutate(across(all_of(numeric_cols), as.numeric),
         `Release Date` = as.POSIXct(`Release Date`, format = "%Y-%m-%d")) |>
  select(Popularity, `Artist Name(s)`, `Track Name`, `Duration (ms)`, `Track ID`)

# Creating the decades column
big_set <- big_dataset |>
  mutate(release_date = as.POSIXct(release_date, format = "%Y-%m-%d")) |>
  mutate(Decade = case_when(
    year >= 2020 ~ "2020+",
    year >= 2010 & year < 2020 ~ "2010 - 2020",
    year >= 2000 & year < 2010 ~ "2000 - 2010",
    year >= 1990 & year < 2000 ~ "1990 - 2000",
    year >= 1980 & year < 1990 ~ "1980 - 1990",
    year >= 1970 & year < 1980 ~ "1970 - 1980",
    year >= 1960 & year < 1970 ~ "1960 - 1970",
    year >= 1950 & year < 1960 ~ "1950 - 1960",
    year < 1950 ~ "1950-",
  ))

# Fixing the duration column to not be in ms but instead in seconds. Also matching the "name" column fo
combined_data <- combined_data |>
  mutate(`Duration (sec)` = `Duration (ms)` / 1000) |>
  rename(
    `artists` = `Artist Name(s)`,
    `id` = `Track ID`,
    `name` = `Track Name`
    )
#combined_data

# Fixing artists names in "big_set" to be more similar to strings instead of lists/arrays
big_set$artists <- gsub("\\[|\\]|'", "", big_set$artists)

# Matching the formatting of the new artists column in combined_data to the one in big_set.
# This line adds a space after every comma...
combined_data$artists <- gsub(",([^ ])", ", \\1", combined_data$artists) #...idek man this is built on

# Adding popularity to the big dataset from our limited dataset
```

```r
complete_set <- big_set |>
  inner_join(
    combined_data,
    by = join_by(id, artists)
  ) |>
  select(-duration_ms) |> # Extra duration column from "big_set"
  # Getting rid of duplicates
  unique() |>
  arrange(artists)
# Modifying the popularity set to be an effective factor variable set (that has ranges)
complete_set <- complete_set |>
  mutate(pop_scale = case_when(
    Popularity <= 10 ~ "0-10",
    Popularity > 10 & Popularity <= 20 ~ "11-20",
    Popularity > 20 & Popularity <= 30 ~ "21-30",
    Popularity > 30 & Popularity <= 40 ~ "31-40",
    Popularity > 40 & Popularity <= 50 ~ "41-50",
    Popularity > 50 & Popularity <= 60 ~ "51-60",
    Popularity > 60 & Popularity <= 70 ~ "61-70",
    Popularity > 70 & Popularity <= 80 ~ "71-80",
    Popularity > 80 & Popularity <= 90 ~ "81-90",
    Popularity > 90 & Popularity <= 100 ~ "91-100"
  ),
    pop_scale = as.factor(pop_scale)
    )
# Creating the artist_count column
complete_set$artist_count <- numeric(nrow(complete_set))
for (i in 1:nrow(complete_set)) {
  count <- 1  # Start with 1 because there's always at least one name
  artist_string <- complete_set$artists[i]

  for (char in strsplit(artist_string, "")[[1]]) {
    if (char == ",") {
      count <- count + 1
    }
  }
  complete_set$artist_count[i] <- count
}

complete_set_split <- initial_validation_split(complete_set, prop = c(0.6, 0.2), strata = Decade)
complete_set_split
```

Tidying explanation

We start the data processing flow by reading the CSV files and converting them into a consistent format. The function reads and ensures all columns are then combined into a single combined_data table. In this table, there are numeric columns such as "Duration (ms)" and "Popularity" which are converted from character to numeric. The "Release Date" is also formatted as a date from characters. The columns such as Popularity, Artist Names, and Track Name are kept in the dataset for further analysis because they are important. Next, the big_set dataset gets a Decade column. When categorizing each track it helps to sort by decade ranges such as "2010 - 2020" or "2000 - 2010". This assists in understanding trends across different eras of music. The data was then changed converting the track duration from milliseconds to seconds. As well, columns were renamed to align with the big_set. Artist names were cleaned for consistency. This allowed other datasets to be compatible for merging. The 2 datasets are merged using an inner join based on the ID and Artists columns. After clearing duplicate entries and sorting the data by artist names, the complete_set is created with info about each track.

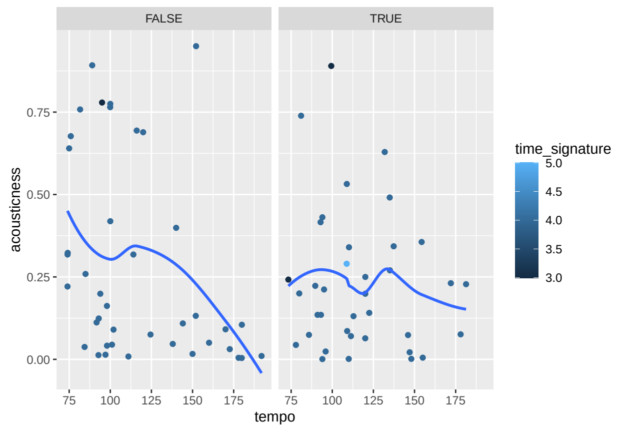
To make the analysis easier to model, we transformed some features into factor variables to categorize data into groups for better comparison. The popularity score is categorized into factor ranges, "0-10" and "11-20" for the decades. This is for better interpretation of popularity distributions. An artist_count column is also added, which counts the number of artists collaborating on each track by identifying the commas in the Artists field. Finally, the cleaned and changed dataset is split into training, validation, and test sets based on the Decade column. This ensures each decade is well represented, creating balance.

Overall, this process creates a structured and well-prepared dataset that is ready for further exploration, such as analyzing trends in music popularity over time or building predictive models based on track attributes.

Visualizations

Exploratory Plot 1- Acousticness vs Tempo and Explicity

```
Acous_vs_temp_plot <- training(complete_set_split) |> ggplot(aes(y = acousticness, x = tempo)) +
  geom_point(aes(colour = time_signature)) +
  geom_smooth(se = FALSE) +
  facet_wrap(~explicit)
Acous_vs_temp_plot
```

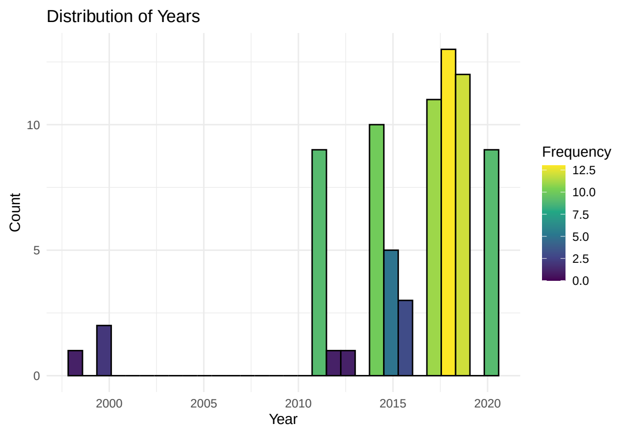


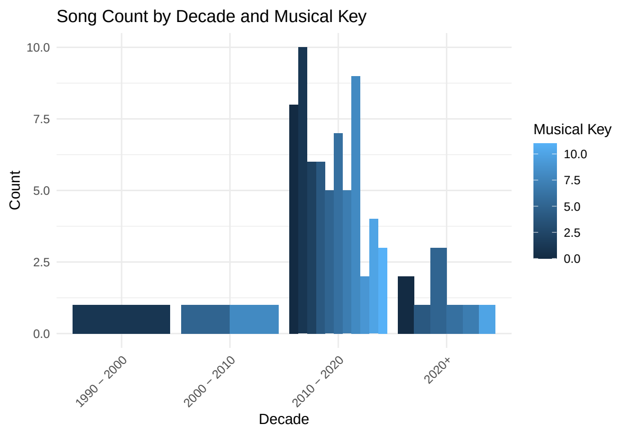
The graph above showcases how different song compositions tend to have a more general pattern, as shown by the graph when the song is not explicit there is an inverse correlation between the acousticness and the tempo of the song. This differs from the graph of the songs which are explicit as the slope is almost flat with there being close to no correlation. With this we can imply the composition of a song based on the factors of tempo and the explicitness. It is important to note that with these conclusions they can be biased based on the small sample size. (see limitations)

Exploratory Plot 2 - Distribution of songs by Year and most common Musical Key by Decade

```
plot_year_histogram <- training(complete_set_split) |>
  ggplot(aes(x = year, fill = after_stat(count))) +
```

```r
  geom_histogram( color = "black") +
  scale_fill_viridis_c() +
  labs(title = "Distribution of Years",
       x = "Year",
       y = "Count",
       fill = "Frequency") +
  theme_minimal()
plot_year_histogram

plot_bar <- training(complete_set_split) |>
  ggplot(aes(x = Decade, group = key, fill = key)) +
  geom_bar(position = "dodge") +
  labs(title = "Song Count by Decade and Musical Key",
       x = "Decade",
       y = "Count",
       fill = "Musical Key") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
plot_bar
```



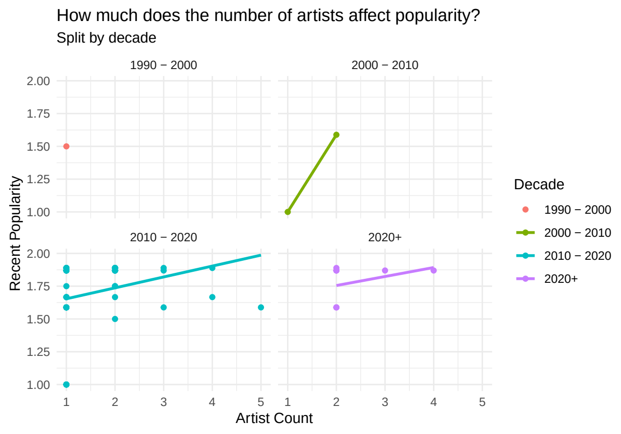


The graph on the left is to show one of the measures we took when cleaning the data. This issue we encountered was the popularity metric that the Spotify api had was less about total popularity and more about relevancy therefore when graphing songs by popularity the data represented was not accurate. Therefore when sourcing the data we only chose songs that were released recently. The second graph showcases how the musical key of songs changes throughout the decades, with this we could suggest the types or taste in music tastes changeover the years. This must be taken with a grain of salt those as the data is such a small sample size. The future of this study could further analyze the impacts of society on music and certain keys potentially being more popular throughout different eras.

Model Plot 1 - Artist Count vs Popularity Line

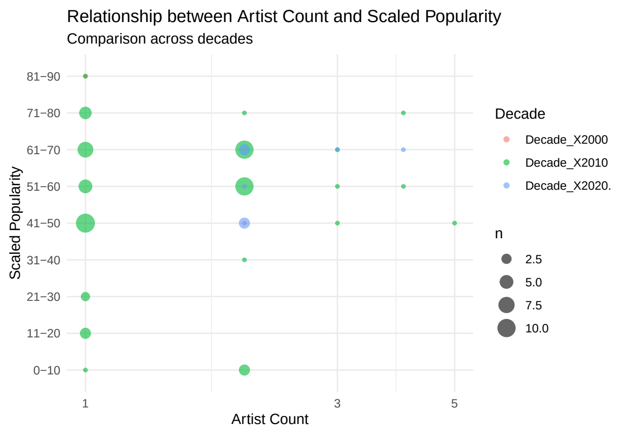
```r
model <- linear_reg() |>
  set_engine("lm")

fit_artist_pop <- model |>
  fit(artist_count ~ pop_scale, data = training(complete_set_split))
```

```r
model_coef <- tidy(fit_artist_pop)
model_aug <- augment(fit_artist_pop, new_data = training(complete_set_split))


recipe_artist_pop <- recipe(pop_scale ~ artist_count + Popularity + Decade, data = training(complete_se
  step_dummy(all_nominal_predictors())

artists_vs_popularity <- model_aug |>
  ggplot(aes(x = artist_count, y = .pred)) +
  geom_point(aes(colour = Decade)) +
  facet_wrap(~Decade) +
  geom_smooth(aes(colour = Decade), method = "lm", se = FALSE) +
  theme_minimal() +
  labs(
    x = "Artist Count",
    y = "Recent Popularity",
    title = "How much does the number of artists affect popularity?",
    subtitle = "Split by decade"
  )

artists_vs_popularity
```



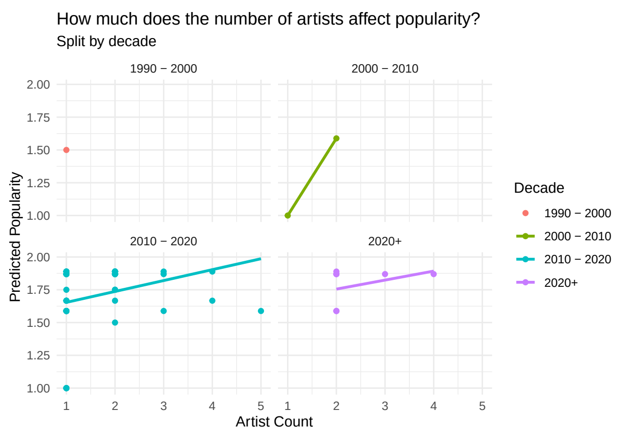
Model Plot 2 - Artist Count vs Popularity Factor

```r
baked_data <- bake(prep(recipe_artist_pop), new_data = training(complete_set_split)) |>
  pivot_longer(
    cols = starts_with("Decade_X"),
    names_to = "Decade",
    values_to = "which_dec",
    values_drop_na = TRUE
  ) |>
  filter(which_dec == 1) |>
  select(-which_dec)

art_test <- ggplot(baked_data, aes(x = artist_count, y = pop_scale, color = Decade)) +
  geom_count(alpha = 0.6) +
  scale_x_log10() +
  theme_minimal() +
  labs(
    title = "Relationship between Artist Count and Scaled Popularity",
    subtitle = "Comparison across decades",
    x = "Artist Count",
    y = "Scaled Popularity",
    color = "Decade"
  )
art_test
```



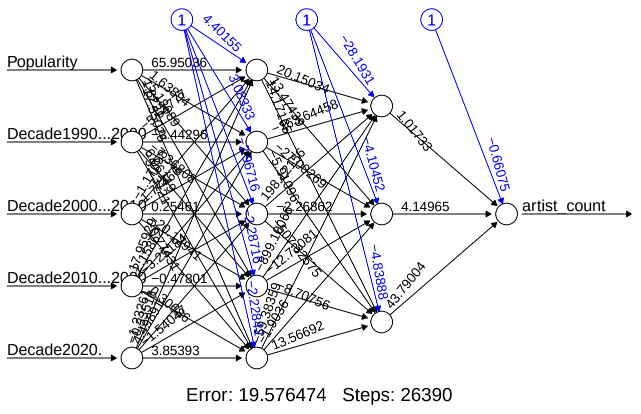
Exploratory Model 1 - Linear Model

```r
set.seed(1234)
valid_set <- vfold_cv(training(complete_set_split), v = 5)

model <- linear_reg() |>
  set_engine("lm")

fit_artist_pop <- model |>
  fit(artist_count ~ pop_scale, data = training(complete_set_split))

model_coef <- tidy(fit_artist_pop)
model_aug <- augment(fit_artist_pop, new_data = training(complete_set_split))

recipe_artist_pop <- recipe(artist_count ~ Popularity + Decade, data = training(complete_set_split)) |>
  step_dummy(all_nominal_predictors())

artists_vs_popularity <- model_aug |>
  ggplot(aes(x = artist_count, y = .pred)) +
  geom_point(aes(colour = Decade)) +
  facet_wrap(~Decade) +
  geom_smooth(aes(colour = Decade), method = "lm", se = FALSE) +
  #geom_smooth(color = "black", method = "loess", se = FALSE) +
  theme_minimal() +
  labs(
    x = "Artist Count",
    y = "Predicted Popularity",
    title = "How much does the number of artists affect popularity?",
    subtitle = "Split by decade"
  )

artists_vs_popularity
```



Exploratory Model 2 - Neural Net/MLP

```
model_data <- model.matrix(~ Popularity + Decade - 1, data = training(complete_set_split))
model_data <- cbind(model_data, artist_count = training(complete_set_split)$artist_count)

scaled_data <- as.data.frame(scale(model_data))
colnames(scaled_data) <- make.names(colnames(scaled_data))

formula <- artist_count ~ Popularity + Decade1990...2000 + Decade2000...2010 + Decade2010...2020 + Deca
mlp_model <- neuralnet(formula,
                       data = scaled_data,
                       hidden = c(5, 3),
                       linear.output = TRUE)

mlp_predictions <- predict(mlp_model, newdata = scaled_data)

plot(mlp_model, rep = "best")
```

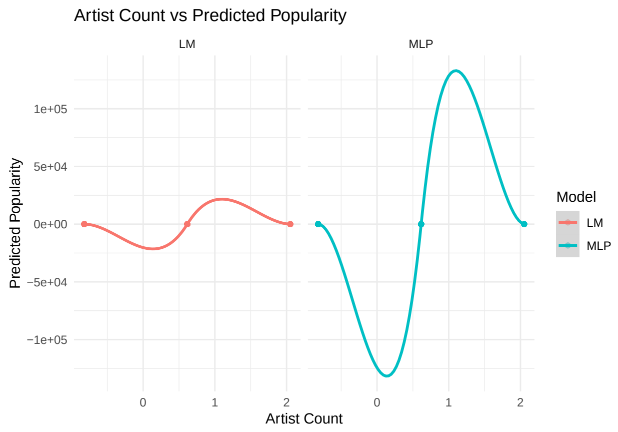


```r
rmse <- sqrt(mean((mlp_predictions - scaled_data$artist_count)^2))
print(paste("RMSE:", rmse))
```

```
## [1] "RMSE: 0.713077729077283"
```

Final Model Comparison

```r
set.seed(1234)

# Prepare data
model_data <- model.matrix(~ Popularity + Decade - 1, data = training(complete_set_split))
model_data <- cbind(model_data, artist_count = testing(complete_set_split)$artist_count)
scaled_data <- as.data.frame(scale(model_data))
colnames(scaled_data) <- make.names(colnames(scaled_data))

# Define formula based on actual column names
formula <- artist_count ~ Popularity + Decade1990...2000 + Decade2000...2010 + Decade2010...2020 + Deca

# MLP model
mlp_model <- neuralnet(formula,
                       data = scaled_data,
                       hidden = c(5, 3),
                       linear.output = TRUE)
mlp_predictions <- predict(mlp_model, newdata = scaled_data)
mlp_rmse <- sqrt(mean((mlp_predictions - scaled_data$artist_count)^2))

# Define and fit the linear model using the same formula
lm_model <- lm(formula, data = scaled_data)

# Make predictions with the linear model
lm_predictions <- predict(lm_model, newdata = scaled_data)
lm_rmse <- sqrt(mean((lm_predictions - scaled_data$artist_count)^2)) # Note: No need for .pred here

# Compare RMSE
rmse_comparison <- data.frame(
    Model = c("MLP", "LM"),
    RMSE = c(mlp_rmse, lm_rmse)
)

# Visualize predictions
predictions_df <- data.frame(
    Actual = scaled_data$artist_count,
    MLP = as.vector(mlp_predictions),
    LM = lm_predictions
)

predictions_long <- pivot_longer(predictions_df, cols = c(MLP, LM), names_to = "Model", values_to = "Pre

ggplot(predictions_long, aes(x = Actual, y = Predicted, color = Model)) +
    geom_point(alpha = 0.5) +
    geom_smooth() +
    facet_wrap(~ Model) +
    labs(title = "Artist Count vs Predicted Popularity",
         #subtitle = "Lower Variance means more precise",
         x = "Artist Count",
         y = "Predicted Popularity") +
    theme_minimal()
```



Conclusion

In this project, we aimed to predict the popularity of songs using Linear Regression and a Machine Learning model, the Multilayer Perceptron (MLP). One notable approach was creating a custom metric, similar to the one used by Spotify, to better capture the factors influencing track popularity. For the linear model, we found that from 2010–2020, songs with more artists tended to be more popular. However, since 2020, artist count shows little impact, possibly due to overused collaborations or the prominence of specific artists. The Multilayer Perceptron (MLP) model followed the data more closely, showing that tracks with no artists have no popularity, while a few artists boost popularity significantly. Beyond a certain number, popularity plateaus or declines. The project emphasized the value of appropriate features and models. While Linear Regression provided a baseline, the MLP offered greater accuracy, showcasing the potential of advanced modelling for predicting track popularity.

Limitations

Through this project we faced many technical challenges. All of these limitations came from the Spotify API. We used a library (spotipy) to make calls in python for various data. However, the actual data itself proved to a large challenge. The first issue we faced was pulling "dancibility" scores. Spotify actually removed the endpoint, and all other "audio feature" endpoints from their API while we were developing a script to pull the data. This was a huge blow to our progress in the analysis, but luckily we found a workaround: the Spotify IOS api had not yet been updated to reflect the change in functionality. So we manually added songs to a playlist on Spotify and used the IOS API to make relevant calls. This resulted in 2,000 songs and their data being pulled, which is being used in this analysis now. The next issue that we faced was the "popularity" metric. This metric is based on current relevance, which meant older songs would be predestined to do worse. It therefore seems the "total streams" metric should instead be used, except that Spotify does publish that data on the API. This dealt a serious blow to the analysis, and while there are creative workarounds (such as scraping the online web player for each artist) they are both too time consuming and beyond the current scope of our ability for this project. Also, due to our workaround songs were being handpicked. This introduces a degree of bias to our representative sample of Spotify data, and handpicking meant we had a lack of data for modelling which can explain inaccuracies.